FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

Reenginyeria del dashboard estratègic de l'eina Q-Rapids

Grau en Enginyeria Informàtica

Enginyeria del Software

Autor: **Aleix Balletbó Gregorio**

Directora: **Cristina Gómez Seoane**
Enginyeria de Serveis i Sistemes d'Informació

Codirectora: **Lidia López Cuesta**
Enginyeria de Serveis i Sistemes d'Informació

25 d'octubre de 2019

Resum

Durant el cicle de vida d'un projecte software, es poden produir desviacions que afecten a la mantenibilitat, escalabilitat i robustesa del mateix. L'objectiu d'aquest projecte és analitzar l'estat actual, estudiar les diferents solucions pels problemes identificats, prendre decisions i executar mesures per tal de millorar aspectes d'arquitectura, tecnologies, implementació, proves i metodologia de treball del dashboard estratègic de l'eina Q-Rapids (Quality-Aware Rapid Software Development).

Resumen

Durante el ciclo de vida de un proyecto software, se pueden producir desviaciones que afecten a la mantenibilidad, escalabilidad y robustez del mismo. El objetivo de este proyecto es analizar el estado actual, estudiar las distintas soluciones para los problemas identificados, tomar decisiones y ejecutar medidas con tal de mejorar aspectos de arquitectura, tecnologías, implementación, pruebas i metodología de trabajo del dashboard estratégico de la herramienta Q-Rapids (Quality-Aware Rapid Software Development).

Abstract

During the life cycle of a software project, deviations may occur that affect its maintainability, scalability and robustness. The goal of this project is to analyse the current status, study the different solutions for the identified problems, make decisions and take actions in order to improve aspects of architecture, technologies, implementation, tests and working methodology of the strategic dashboard of the Q-Rapids tool (Quality-Aware Rapid Software Development).

Índex

Figures

Taules

1. Introducció i contextualització

En aquest primer apartat s'introdueixen els aspectes clau que conformen el context del projecte, s'enumeren els actors implicats i es descriu l'estat de l'art.

1.1. Context

1.1.1. Introducció

Aquest és un Treball Final de Grau corresponent a l'especialitat d'Enginyeria del Software del Grau en Enginyeria Informàtica, impartit per la Facultat d'Informàtica de Barcelona (FIB) de la Universitat Politècnica de Catalunya (UPC). Es tracta d'un projecte de modalitat A, és a dir, desenvolupat dins de l'àmbit UPC. En concret, està vinculat al departament d'Enginyeria de Serveis i Sistemes d'Informació (ESSI).

Aquest treball està integrat dins del projecte Q-Rapids (*Quality-Aware Rapid Software Development*) [1], un projecte europeu de recerca que s'emmarca dins del programa *Horizon 2020*, desenvolupat per la Universitat Politècnica de Catalunya, la University of Oulu i el Fraunhofer Institute for Experimental Software Engineering, juntament amb els socis industrials Bittium, Softeam, Itti i Nokia.

El projecte Q-Rapids busca proporcionar una solució al problema d'integrar requisits de qualitat dins d'un procés de desenvolupament de software àgil, amb l'objectiu de millorar la productivitat i reduir el temps de llançament al mercat a la vegada que es garanteixen uns nivells de qualitat apropiats pel producte. Per tal de dur-ho a terme, es desenvolupa una metodologia i una eina de suport al desenvolupament de software centrada en la qualitat i basada en dades, on els requisits de qualitat són obtinguts, refinats i millorats de manera incremental a partir de dades recollides de repositoris de software, eines de gestió, sistemes utilitzats i qualitat del servei. Aquestes dades són posteriorment analitzades i agregades formant indicadors de qualitat, i es presenten a l'usuari final a través d'un panell de control o *dashboard* estratègic altament informatiu en forma d'aplicació web.

L'objectiu d'aquest treball és aplicar un procés de reenginyeria al *dashboard* estratègic de l'eina Q-Rapids, mitjançant el qual es busca incidir en aspectes d'arquitectura, tecnologies, implementació i proves per tal de millorar-ne la mantenibilitat, escalabilitat i robustesa.

1.1.2. Actors implicats

Directora i codirectora del projecte

La directora i la codirectora del projecte són la Dra. Cristina Gómez Seoane i la Dra. Lidia López Cuesta, respectivament. Són les persones encarregades de guiar, assessorar i supervisar a l'autor del projecte durant el transcurs del mateix, aportant la seva experiència i coneixement per tal que aquest es desenvolupi de forma satisfactòria.

Ambdues realitzen tasques d'investigació al grup GESSI, i participen directament en el projecte Q-Rapids. El resultat d'aquest treball final de grau tindrà un efecte directe en aquest projecte de recerca.

Autor del projecte

L'autor d'aquest projecte és l'Aleix Balletbó Gregorio, estudiant del Grau en Enginyeria Informàtica a la FIB, concretament a l'especialitat d'Enginyeria del Software. Serà l'encarregat de dur a terme totes les tasques que conformen aquest projecte, d'entre les quals en destaquen l'anàlisi, implementació i validació, així com tasques transversals de gestió del propi projecte.

Cal destacar que, paral·lelament a la realització d'aquest projecte, l'autor treballa com a desenvolupador dins del projecte Q-Rapids, afegint noves funcionalitats al *dashboard* estratègic i sota la supervisió directa de la codirectora d'aquest projecte i responsable del *dashboard*, la Dra. Lídia López Cuesta. Això implica que les millores derivades del procés de reenginyeria beneficiaran directament a l'autor en les tasques que duu a terme diàriament.

Grup de recerca GESSI

Dins del projecte Q-Rapids, la responsabilitat de desenvolupar el *dashboard* estratègic recau sobre el grup de recerca GESSI, de la UPC. Com és natural, aquest grup duu a terme altres projectes de desenvolupament, i les decisions i consideracions resultants del procés de reenginyeria relatives a aspectes de tecnologies, arquitectura, implementació i proves poden ser aplicades a altres projectes de característiques semblants, ampliant els beneficis.

Socis industrials i futurs usuaris

El projecte Q-Rapids compta amb quatre socis industrials durant la fase de desenvolupament per tal de provar l'eina en un entorn real i avaluar el seu funcionament, així com adaptar-la a les necessitats empresarials. Les empreses participants són Bittium, Softeam, Itti i Nokia. Un cop el projecte hagi finalitzat, altres empreses poden adoptar la metodologia i l'eina Q-Rapids i integrar-les dins dels seus processos de desenvolupament de software. Les millores de qualitat derivades del procés de reenginyeria impactaran directament en aquests usuaris, contribuint a reduir errors un cop el producte ja està desplegat i millorant-ne la robustesa.

Col·laboradors externs

Tot i ser un projecte de recerca desenvolupat per la comunitat acadèmica, l'eina Q-Rapids es troba accessible com a projecte de codi obert. Això vol dir que qualsevol persona amb els coneixements adients pot proposar i implementar millores i noves funcionalitats. El procés de reenginyeria facilitarà aquesta tasca de manteniment als col·laboradors que vulguin contribuir al projecte, ja que es trobaran amb un software on s'han aplicat una sèrie de decisions a nivell tecnològic, arquitectònic, d'implementació i validació que afavoreixen i faciliten la millora i ampliació de les funcionalitats actuals.

1.2. Estat de l'art

1.2.1. Definició, beneficis i necessitats de la reenginyeria del software

La reenginyeria del software és, segons la seva definició formal inicial àmpliament acceptada, l'examinació i alteració d'un sistema software amb l'objectiu de reconstituir-lo en una nova forma, i la subseqüent implementació d'aquesta nova forma [2]. Amb aquest procés es busca que el sistema resultant adquireixi unes propietats determinades absents o poc presents en el sistema inicial, és a dir, introduir un benefici. Entre ells en destaquen la mantenibilitat, la portabilitat, la fiabilitat, la reusabilitat, la qualitat de la documentació, la capacitat de prova i la usabilitat [3, 4]. La necessitat de recórrer a un procés de reenginyeria sovint respon als casos següents [5]:

- El codi no té una estructura clara i lògica, i la documentació és insuficient.
- El suport de determinats sistemes esdevé obsolet degut a canvis en l'organització, en el mercat o en les necessitats empresarials.
- Els desenvolupadors de sistemes heretats no es troben disponibles per verificar i transmetre informació.
- A través d'anys de modificacions, els canvis necessaris en sistemes heretats esdevenen complicats, arriscats i cars.

1.2.2. Model i enfocaments tradicionals

Mentre que l'enginyeria tradicional comença amb unes especificacions i acaba amb un sistema funcional per mitjà de processos de disseny i implementació, un procés de reenginyeria comença amb un sistema existent funcional i en deriva un de nou per mitjà d'anàlisi i transformacions. Per il·lustrar aquest procés, la següent figura (Figura 1) en mostra les diferents etapes, així com els diferents nivells d'abstracció involucrats en els sistemes [3].

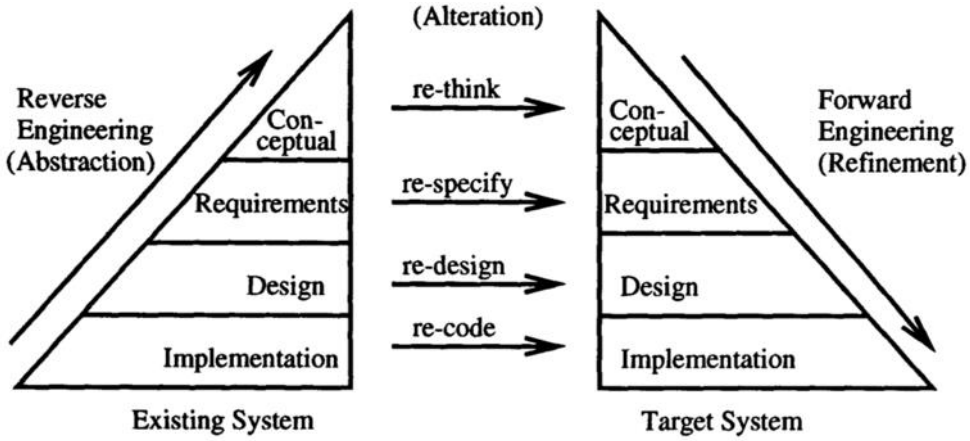

Figura 1. Model general de la reenginyeria del software [3]

Els tres principis clau que s'apliquen en aquest model són l'abstracció, l'alteració i el refinament. L'abstracció correspon a l'increment gradual del nivell d'abstracció de la informació del sistema, des d'un nivell baix a un d'alt. Tal i com descriu la figura, per realitzar aquest increment en el nivell d'abstracció es duu a terme un procés

d'enginyeria inversa. L'alteració correspon a l'execució d'un o més canvis dins d'un mateix nivell d'abstracció. Per acabar, el refinament correspon a l'oposat de l'abstracció. Es tracta d'un decrement gradual en el nivell d'abstracció del sistema, on es transforma la informació present en els nivells d'abstracció del sistema existent per informació més detallada [3].

Per tal de dur a terme un procés de reenginyeria, es contemplen tradicionalment tres enfocaments [5]:

- Enfocament *Big Bang*: es tracta de la substitució completa i d'un sol cop del sistema existent per un de nou. Permet al sistema adaptar-se amb facilitat a nous entorns, però consumeix molt temps i recursos abans de poder apreciar els seus efectes.

- Enfocament incremental: es tracta de realitzar un procés de reenginyeria per parts, substituint els components de l'actual sistema de forma esglaonada. Permet reduir riscos respecte a l'enfocament anterior i avaluar els efectes ràpidament, tot i que la realització completa del procés consumirà més temps.

- Enfocament evolutiu: es tracta d'un enfocament similar a l'incremental, però les parts que es van substituint no són components sinó funcionalitats del sistema. Permet crear nous components altament cohesionats, però canvia en gran mesura l'estructura interna de l'actual sistema.

Aquests enfocaments són molt generals i no s'aborda quin és el procés que cal seguir per realitzar-los. Per tant, es necessita un enfocament més concret, com el que s'exposa en el següent apartat.

1.2.3. La reenginyeria del software centrada en la qualitat

Com s'ha comentat en punts anteriors, un dels motius principals per realitzar un procés de reenginyeria es la qualitat del software. Per dur a terme aquest procés, hi ha autors que proposen fer servir un mètode basat en l'anàlisi de l'arquitectura del software i l'avaluació de la qualitat per dur a terme els canvis que siguin oportuns [6]. Per la seva naturalesa, l'avaluació de la qualitat és un tema complex. Per abordar aquest problema, un dels processos que es proposa fer servir és una avaluació basada en escenaris.

Es tracta de partir de l'arquitectura actual del sistema i identificar escenaris en els quals es veuen involucrats requisits de qualitat, com ara modificacions funcionals, noves dependències externes o canvis en una determinada tecnologia. A partir d'aquests escenaris, cal veure com el sistema actual respon i si afavoreix o perjudica els factors de qualitat que s'estiguin tractant. Si aquests es veuen compromesos, cal proposar una alteració del sistema que resolgui el problema [6].

Aquest serà el procediment que se seguirà per dur a terme aquest projecte de reenginyeria, ja que fins al moment aquesta avaluació dels requisits de qualitat no ha estat present en el desenvolupament, i permet sintetitzar amb claredat quines són les implicacions reals de cada requisit de qualitat en el context del sistema.

2. Abast del projecte

Tot seguit es planteja la formulació del problema i els seus objectius, es defineix l'abast i es descriu la metodologia utilitzada i els elements de rigor.

2.1. Formulació del problema

Durant el transcurs del desenvolupament d'un projecte software, especialment en els casos en els que se segueix una metodologia basada en iteracions curtes, es poden produir desviacions que afecten a la qualitat del producte en qüestió. Aquestes desviacions són el resultat de decisions preses durant el procés de desenvolupament, sovint motivades per factors com un calendari d'entrega ajustat o el desconeixement d'una certa tecnologia o arquitectura. Consisteixen en reduir el temps i esforços necessaris per dur a terme una certa tasca en un moment determinat, però a la vegada contribuint a augmentar la complexitat de futurs canvis, introduint un concepte que rep el nom de deute tècnic [7].

El deute tècnic és una metàfora utilitzada en el món de l'enginyeria del software per tal d'il·lustrar les situacions que, tot i que aparentment poden resultar beneficioses en un moment determinat, provoquen situacions indesitjables en el futur. De la mateixa manera que el deute financer, el deute tècnic comporta uns interessos en forma d'esforç addicional que, en cas de no ser eliminats, es van acumulant fins que la situació es fa insostenible. Per tant, és de vital importància per qualsevol projecte de software mantenir el deute tècnic sota control, i invertir esforços en eliminar-lo de forma adequada.

El *dashboard* estratègic de l'eina Q-Rapids té unes particularitats concretes que fan que sigui especialment susceptible al deute tècnic i a altres factor que en poden alterar la qualitat. Primer de tot, cal recordar que forma part d'un projecte de recerca on l'element principal és la vessant científica, i com és natural és l'àmbit que més esforç requereix. Això ha comportat que una gran part del desenvolupament del *dashboard* hagi recaigut en alumnes del Grau en Enginyeria Informàtica becats pel grup de recerca que, tot i ser supervisats pels investigadors del projecte, disposen d'un cert grau de llibertat a l'hora de prendre determinades decisions en el transcurs de la seva feina. Tot i l'excel·lent formació que els alumnes reben a la FIB, la falta d'experiència en projectes reals de gran envergadura juga en contra dels interessos del projecte.

També cal destacar la metodologia basada en iteracions que s'ha seguit per desenvolupar el *dashboard* estratègic, que té com a objectiu anar construint el producte de forma incremental i adaptant-se als canvis requerits pel propi projecte i els socis industrials. Aquesta voluntat de rapidesa i flexibilitat davant el canvi fa que, si no es té prou cura, s'acabi pensant únicament en el problema actual al que s'enfronta el desenvolupador enlloc de mantenir una visió global.

Aquestes situacions han estat visibles i constatables per l'autor d'aquest treball durant les seves tasques com a desenvolupador del *dashboard* estratègic. La corba d'aprenentatge del projecte és elevada, i això sumat a certes inconsistències a nivell arquitectònic i males pràctiques fan que les modificacions i la incorporació de noves funcionalitats siguin costoses, així com la detecció d'errors. Aquest problema pren

especial rellevància al tractar-se d'un projecte de codi obert, ja que suposa un impediment important pels col·laboradors externs que vulguin implementar noves funcionalitats o millores sense tenir coneixement previ del projecte.

Partint d'aquestes premisses, l'objectiu general d'aquest treball final de grau és l'aplicació d'un procés de reenginyeria sobre el *dashboard* estratègic de l'eina Q-Rapids amb la finalitat de millorar-ne la qualitat i reduir el deute tècnic, incidint en aspectes de mantenibilitat, escalabilitat i robustesa. Per dur-lo a terme, es vol complir amb els següents objectius específics:

- Millorar l'arquitectura actual dels components interns del *dashboard* per tal d'adequar el seu disseny als principis i patrons de l'enginyeria del software.
- Millorar el disseny del codi mitjançant l'aplicació de bones pràctiques que redueixin la seva complexitat i en facilitin la modificació.
- Introduir l'automatització de proves per tal de poder detectar errors durant el procés de desenvolupament i reduir el risc associat a la modificació de codi ja existent.
- Utilitzar eines d'anàlisi i automatització que permetin millorar el procés de desenvolupament de software i contribueixin a evitar una futura davallada en el nivell de qualitat del producte.

2.2. Abast

2.2.1. Acotament del projecte i fases implicades

La primera apreciació tècnica que cal fer respecte al *dashboard* com a aplicació web és que aquesta funciona seguint una arquitectura de tipus client-servidor. Això implica que hi ha una part del sistema, l'encarregada de presentar les dades a l'usuari de forma gràfica i permetre la interacció, que s'executa en els clients (en aquest cas, els navegadors web dels usuaris) utilitzant unes tecnologies determinades, mentre que la lògica i la persistència de dades s'executa en un servidor centralitzat. Partint d'aquest paradigma, s'ha acordat acotar l'abast d'aquest projecte de reenginyeria a la part corresponent al servidor del *dashboard*, degut a les restriccions de temps i a la quantitat d'oportunitats de millora detectades.

Per tal de dur a terme el procés de reenginyeria, aquest es dividirà en les fases següents:

Anàlisi de la situació actual

Durant aquesta fase cal fer un anàlisi exhaustiu de l'estat actual del *dashboard* estratègic. És necessari tractar les tecnologies utilitzades pel seu desenvolupament, així com el disseny arquitectònic utilitzat i l'organització interna dels seus components. També cal descriure l'estat dels processos que acompanyen al desenvolupament, com són les proves i l'entorn d'eines i serveis utilitzats. Per últim, cal tenir present quins són els requisits no funcionals actuals i quin és el seu grau de satisfacció.

Identificació dels problemes, anàlisi d'alternatives i solució

Partint de l'anàlisi de la situació actual, caldrà identificar quins són els problemes que se'n deriven. Per cadascun d'ells, un cop queda justificat que efectivament constitueix un problema per la qualitat del producte, caldrà analitzar quines alternatives es contemplen per tal de millorar la situació actual i, en cas que n'hi hagi més d'una, justificar quina és la que s'acabarà adoptant.

Aplicació de les millores

Un cop s'han delimitat quins canvis i millores es realitzaran, cal procedir a implantar-les al projecte. Per fer-ho, serà necessari realitzar una fase de disseny inicial que agrupi totes les millores referents a arquitectura per tal de tenir una visió global, seguida per un pla d'execució que optimitzi el procés d'implementació. Un cop fet això, serà el torn d'implementar cadascuna de les millores i, per acabar, procedir a validar-ne els resultats.

2.2.2. Possibles obstacles

Durant l'aplicació del procés de reenginyeria, es possible trobar-se amb obstacles i dificultats que compliquin la seva realització. Els elements principals que es contemplen són els següents:

Limitació de temps

Un projecte de reenginyeria pot requerir una quantitat de temps considerable en funció de la quantitat de canvis i millores que calgui introduir. Cal tenir present que aquest projecte es realitza en el marc d'un treball final de grau, i el calendari ha d'anar marcat pel temps disponible fins la data d'entrega. Per tal d'adaptar-se a aquesta restricció, caldrà limitar el nombre d'activitats a realitzar durant el procés de reenginyeria i prioritzar aquelles que tinguin un major impacte en el projecte.

Coordinació entre el procés de reenginyeria i el desenvolupament

Tal i com s'ha mencionat anteriorment, l'autor d'aquest projecte de reenginyeria també treballa com a desenvolupador de l'eina Q-Rapids. Això implica que, per tal d'evitar conflictes entre els canvis realitzats en aquest projecte i les noves funcionalitats que s'afegeixen al *dashboard*, s'haurà de ser conscient de quines repercussions té cada modificació del codi existent i coordinar les dues activitats per minimitzar el risc d'interferències i endarreriments.

Incompatibilitat dels canvis

Un cop s'hagi estudiat l'estat actual del *dashboard* i s'hagin identificat tots els punts de millora, es disposarà d'un llistat amb elements que fan referència a parts molt diverses del sistema. És important agregar tots aquests canvis que es volen introduir dissenyant una solució única que els contempli tots i, en cas que sigui necessari, descartar-ne els que provoquin certes incompatibilitats amb d'altres.

Codi en mal estat

Per tal de poder aplicar els canvis que formaran part del procés de reenginyeria, és necessari primer entendre el comportament del codi que s'està modificant. El mal estat del codi pot influir negativament en aquest aspecte, dificultant la seva comprensió. Per tant, caldrà anar amb cura a l'hora d'estudiar el codi actual i assegurar-se de quin és el funcionament esperat. En cas que sigui necessari, caldrà validar amb la codirectora del projecte aquest anàlisi del comportament per assegurar que la solució que es proposa compleix amb els requisits funcionals presents al projecte.

Introducció d'errors

Degut a la naturalesa d'un projecte de reenginyeria, els canvis que s'introdueixen són transversals i afecten a un gran nombre de funcionalitats del sistema. Un aspecte molt important durant aquest procés és assegurar que no s'introdueixen errors en el codi que afectin a funcionalitats ja finalitzades i que funcionen correctament. Per aquest motiu, caldrà introduir proves automatitzades que s'encarreguin de corroborar el bon funcionament del sistema i contribueixin a minimitzar el risc d'errors.

2.3. Metodologia i rigor

2.3.1. Mètode de treball

Per dur a terme aquest projecte es seguirà una metodologia en cascada o *waterfall* [8], que consisteix en l'execució de forma seqüencial i consecutiva de les fases del projecte, disposades seguint una progressió lògica on el resultat de cada fase és el punt de partida de la següent. Aquesta metodologia busca definir amb el màxim detall cadascuna de les etapes abans de passar a la següent, i no admet solapaments entre elles. Està format per les fases següents: anàlisi de requisits, especificació, disseny, implementació, validació i manteniment (Figura 2).

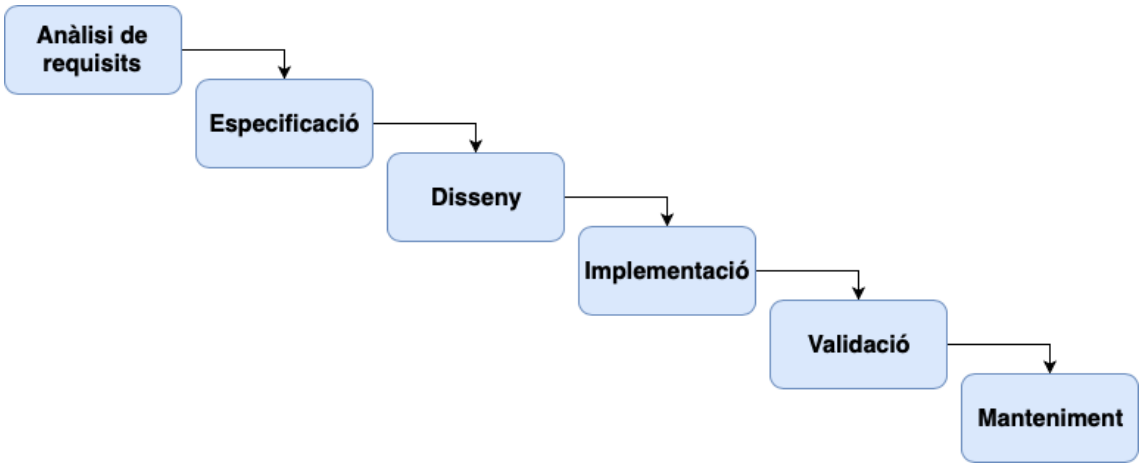

Figura 2. Metodologia en cascada

En particular, per aquest projecte de reenginyeria la fase d'anàlisi de requisits consistirà en determinar exactament quins requisits de qualitat són els que es volen aplicar, i quines millores caldrà introduir per aconseguir-ho. Pel que fa a l'especificació, com que els requisits que es volen aplicar són de tipus no funcional i no es vol modificar cap de les funcionalitats del sistema, no es produiran canvis en aquesta fase. A continuació, la

fase de disseny consistirà en dissenyar una arquitectura que integri aquests nous requisits i proporcioni una solució pels problemes identificats. Durant la fase d'implementació es duran a terme totes les millores identificades i dissenyades, i a la fase de validació es comprovarà que els canvis introduïts satisfan els requeriments de qualitat inicials. La fase de manteniment queda fora de l'abast del projecte.

Aquesta metodologia s'adapta correctament al projecte de reenginyeria que es vol realitzar. Primerament, cal destacar que la metodologia en cascada prové del món de l'enginyeria tradicional on es construeixen productes materials, i on es vol minimitzar la introducció de canvis ja que és un procés costós. Tot i que aquest és un projecte software, es parteix d'uns requisits de qualitat que no canviaran durant el projecte, i les modificacions que es volen introduir són d'una envergadura considerable, cosa que fa que sigui necessari reduir el risc d'haver de fer canvis a meitat del procés degut a un anàlisi i disseny insuficients. A més, aquest projecte compta amb una única persona per desenvolupar-lo, de manera que no és possible realitzar diverses tasques de forma paral·lela, i la distribució lògica de les tasques d'aquesta metodologia fa que se segueixin uns processos de raonament més naturals.

2.3.2. Eines de seguiment

Per tal de realitzar la fase d'implementació del procés de reenginyeria s'utilitzarà la plataforma que actualment es fa servir pel desenvolupament del *dashboard*, GitHub [9]. Aquest servei proporciona repositoris de codi remots del sistema de control de versions Git [10], que permetrà traçar els canvis que s'introdueixin durant el procés. Es treballarà en una branca aïllada per tal de no interferir en el desenvolupament de noves funcionalitats.

Per tal d'obtenir més informació sobre l'estat actual i observar l'impacte dels canvis i millores introduïdes, es farà servir SonarCloud [11] com a eina d'anàlisi estàtic de codi, que s'encarregarà d'analitzar el codi del *dashboard* i proporcionar informació sobre la seva qualitat.

Com a eina de comunicació s'utilitzarà el correu electrònic per mantenir un contacte constant amb la directora i codirectora del projecte.

2.3.3. Mètode de validació

La validació de requisits de qualitat és un procés difícil de dur a terme per part dels arquitectes de software degut a la seva naturalesa, ja que la qualitat és un concepte abstracte i difícil de quantificar [12]. Per tal de validar que el procés de reenginyeria està aplicant canvis que contribueixen a millorar els nivells de qualitat del *dashboard*, es farà servir un procés manual amb el suport d'un d'automatitzat, que és un enfocament molt present a la indústria [13]. Es realitzaran reunions setmanals amb la codirectora del projecte on es discutiran els canvis introduïts amb el suport de les dades proporcionades per l'eina d'anàlisi estàtic de codi, i s'avaluarà el seu impacte.

A més, amb la finalitat de mantenir un control sobre la documentació, s'utilitzarà el correu electrònic tant amb la directora com amb la codirectora per corregir possibles desviacions, sobretot en les fases d'anàlisi i disseny.

3. Planificació temporal

En aquest apartat es descriuen les tasques, temps i recursos necessaris per dur a terme el projecte, així com una valoració d'alternatives i un pla d'acció.

3.1. Descripció de les tasques

En aquest apartat es descriu la planificació general de les tasques que constitueixen aquest treball final de grau. Es comença per la descripció de cadascuna d'elles, seguit pel calendari on es mostren les hores necessàries i l'organització de les tasques i conclou amb els recursos necessaris per dur-les a terme.

3.1.1. Tasques

Organització i planificació del projecte

La primera activitat que es durà a terme consisteix en l'elaboració de documentació que serveix com a punt de partida del projecte i contribueix a la seva bona organització i planificació. Aquestes tasques foment part del mòdul de gestió de projectes (GEP), i en concret es demana el següent:

- Definició de l'abast i contextualització. Consisteix en la definició del context del projecte, l'estat de l'art, la formulació del problema, la definició de l'abast i la metodologia que se seguirà.
- Planificació temporal. Consisteix en la descripció de les tasques a realitzar, el temps i recursos utilitzats, així com la valoració d'alternatives i pla d'acció.
- Gestió econòmica i sostenibilitat. Consisteix en l'elaboració del pressupost del projecte i l'anàlisi de sostenibilitat.

A més, com a darrera activitat cal realitzar una presentació oral on es presenta i defensa la documentació generada.

Anàlisi de la situació actual

Un cop realitzada l'organització i planificació del projecte, cal començar amb el procés de reenginyeria. Durant aquesta primera fase cal fer un anàlisi exhaustiu de l'estat actual del *dashboard* estratègic. És necessari tractar les tecnologies utilitzades pel seu desenvolupament, així com el disseny arquitectònic utilitzat i l'organització interna dels seus components. També cal descriure l'estat dels processos que acompanyen al desenvolupament, com són les proves i l'entorn d'eines i serveis utilitzats.

Identificació dels problemes, anàlisi d'alternatives i proposta de solució

Partint de l'anàlisi de la situació actual, cal identificar quins són els problemes que se'n deriven. Per cadascun d'ells, un cop queda justificat que efectivament constitueix un problema per la qualitat del producte, caldrà analitzar quines alternatives es contemplen per tal de millorar la situació actual i, en cas que n'hi hagi més d'una, justificar quina és la que s'acabarà adoptant.

Refinament dels requisits de qualitat

Amb la informació extreta de la identificació dels problemes i les solucions proposades, és necessari revisar els requisits de qualitat actuals del *dashboard*, especificar-ne millor els existents i afegir-ne de nous, si escau. És necessari definir condicions de satisfacció per tal de poder utilitzar-les a la fase de validació de requisits.

Disseny dels canvis

Un cop es té un coneixement exacte de quins són els canvis i millores que cal aplicar, el següent pas és elaborar un disseny que les agrupi i permeti obtenir una visió global. Aquest disseny consistirà en diagrames pels canvis relacionats amb l'arquitectura, i explicacions detallades per les millores introduïdes en aspectes d'implementació, proves i metodologia.

Pla d'execució dels canvis

Partint del disseny dels canvis i millores, és necessari elaborar un pla d'execució que permeti optimitzar el procés d'implementació i minimitzar el seu risc, a la vegada que es contemplen quines són les modificacions que proporcionaran un major benefici pel projecte.

Implementació dels canvis

Durant aquesta fase cal dur a terme la implementació de cadascun dels canvis proposats, seguint l'ordre establert en el pla d'acció.

Validació dels requisits de qualitat

Un cop s'han implementat els canvis, cal veure quin impacte tenen en els requisits de qualitat i comprovar que contribueixen a millorar la situació inicial per mitjà de la validació de les condicions de satisfacció definides en la fase de refinament dels requisits.

Finalització de la memòria

Tan bon punt s'hagin completat totes les tasques anteriors, cal finalitzar la redacció de la memòria i elaborar els apartats que permetran donar per acabat el projecte. En concret, cal definir els següents punts:

- Desviacions. Descripció dels obstacles, l'estat dels objectius i les desviacions metodològiques, temporals i pressupostàries.
- Accions futures. Descripció de les futures accions que es poden realitzar a nivell de reenginyeria a partir de la finalització d'aquest projecte.
- Conclusions. Valoració personal del projecte.

Preparació de la defensa

Per últim, cal preparar la defensa del projecte davant del tribunal. Es tracta d'elaborar una presentació oral amb diapositives com a material de suport on es descriuen els aspectes més importants del projecte realitzat.

3.1.2. Temps

Per tal de dur a terme les tasques descrites a l'apartat anterior, cal planificar l'esforç que requerirà cadascuna i distribuir-les en el temps. A la taula següent (Taula 1) es poden observar les hores assignades per cada tasca:

Tasca	Hores
Organització i planificació del projecte	70
Anàlisi de la situació actual	80
Identificació dels problemes, anàlisi d'alternatives i proposta de solució	60
Refinament dels requisits de qualitat	20
Disseny dels canvis	40
Pla d'execució dels canvis	10
Implementació dels canvis	150
Validació dels requisits de qualitat	20
Finalització de la memòria	30
Preparació de la defensa	30
Total	**510**

Taula 1. Distribució d'hores per tasca

Amb l'objectiu de distribuir les tasques en el temps que es disposa per realitzar el projecte, es plasmen en un diagrama de Gantt (Figura 3) que permet visualitzar el temps assignat a cadascuna, el temps total del projecte i les dependències existents, explicades a la descripció de les tasques. Degut a que aquestes es realitzaran de forma seqüencial i es disposa d'una única persona encarregada de la seva elaboració, no s'ha considerat necessari elaborar un diagrama de PERT.

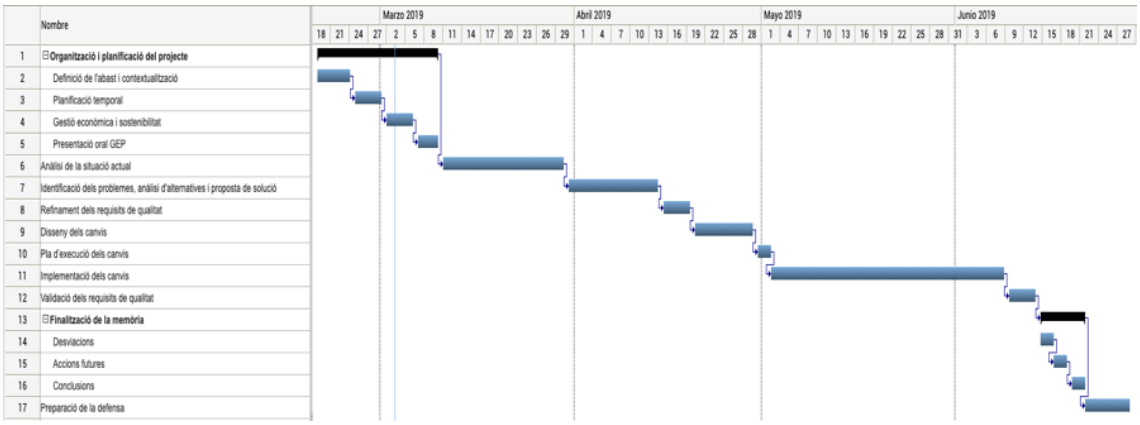

Figura 3. Diagrama de Gantt

3.1.3. Recursos

Tot seguit es descriuen els recursos necessaris, agrupats segons la seva naturalesa.

3.1.3.1. Recursos humans

Aquest projecte comptarà amb una única persona encarregada de la seva elaboració.

3.1.3.2. Recursos materials

Els recursos materials utilitzats aquest projecte són els següents:

- **Ordinador portàtil** per desenvolupar el projecte i provar-ne el funcionament.
- **Pantalla externa** per agilitzar el flux de treball.
- **Teclat i ratolí** per utilitzar l'ordinador portàtil com un ordinador de sobretaula.
- **Taula i cadira d'oficina** on es durà a terme el projecte.
- **Connexió a internet** per buscar informació i treballar en línia.
- **Electricitat** per fer funcionar els aparells electrònics.

3.1.3.3. Recursos software

El desenvolupament d'aquest projecte requereix els següents recursos software:

- **IntelliJ IDEA Ultimate** [14]. Entorn de desenvolupament integrat per dur a terme els canvis i modificacions del procés de reenginyeria.
- **Postman** [15]. Eina per provar l'API (*Application Programming Interface*) del *dashboard*.
- **Apache Tomcat** [16]. Contenidor web per instal·lar i provar el *dashboard*.
- **Git** [10]. Sistema de control de versions pel desenvolupament del projecte.
- **GitHub** [9]. Eina web de gestió de repositoris Git en línia.
- **Travis CI** [17]. Eina web d'integració contínua per automatitzar tasques durant el desenvolupament del projecte.
- **SonarCloud** [11]. Eina web d'anàlisi estàtic de codi per extreure informació sobre la qualitat del *dashboard*.
- **Microsoft Office** [18]. Aplicacions d'ofimàtica per redactar documents i preparar la presentació de la defensa.
- **Gantter** [19]**.** Eina web per gestionar projectes i elaborar diagrames de Gantt.
- **Draw.io** [20]. Eina web per elaborar diagrames.

3.2. Valoració d'alternatives i pla d'acció

Durant el transcurs del projecte, poden aparèixer situacions que alterin el calendari i la planificació, i posin en perill l'acabament del projecte en el temps establert. Per tal d'abordar-les, es proposa el següent.

Es disposa d'aproximadament 18 setmanes des de l'inici del projecte fins a la data d'entrega. El règim inicial de treball serà d'unes 30 hores setmanals, amb jornades de 4 hores diàries els dies laborables i 5 hores diàries els caps de setmana. Amb això s'aconsegueixen 540 hores útils per realitzar el projecte, cosa que encaixa amb la planificació inicial.

Tot i això, és possible que la planificació d'algunes de les tasques no s'ajusti a la realitat. Si l'error ha estat per excés d'hores planificades, senzillament es començarà la següent tasca abans de la data prevista, i permetrà guanyar temps en el futur. En canvi, si l'error ha estat per una manca d'hores, es procedirà a incrementar les hores de treball setmanals per tal de poder completar les tasques problemàtiques dins del calendari establert. Es pot arribar amb facilitat a 40 hores setmanals afegint hores els caps de setmana, i en cas de necessitat ingent també es pot afegir una hora extra els dies laborables, arribant fins a les 45 hores setmanals.

Per la seva naturalesa, la tasca que té un grau més alt de risc és la d'implementació dels canvis. Poden aparèixer imprevistos que compliquin l'aplicació dels canvis i modificacions, i fer que no s'aconsegueixin els objectius inicials. Per aquest motiu, tal i com s'ha exposat a la descripció de les tasques, s'ha afegit una tasca prèvia a la implementació que consisteix en realitzar un pla d'execució dels canvis que es volen introduir. L'objectiu és optimitzar aquest procés i prioritzar les modificacions que tenen un major benefici pel projecte. D'aquesta manera, en cas que la planificació no s'ajusti a la realitat i l'augment d'hores setmanals no sigui suficient, es podran descartar aquells canvis que siguin menors o no tinguin una especial rellevància.

4. Pressupost

A continuació s'identifiquen i estimen els costos del projecte, i es proposen mecanismes de control de gestió.

4.1. Identificació i estimació dels costos

Com a pas previ per tal de poder estimar amb major precisió els costos del projecte, cal determinar el cost per hora de cada rol que participa en les diferents tasques. El salari anual es basa en un estudi dels salaris el sector tecnològic [21], i el salari per hora s'estima partint de la premissa que en un any es treballen unes 225 jornades de 8 hores.

Rol	Salari mínim anual (€)	Salari mínim per hora (€)	Salari màxim anual (€)	Salari màxim per hora (€)	Salari mitjà anual (€)	Salari mitjà per hora (€)
Cap de projecte	40.000	22,22	60.000	33,33	50.000	27,77
Arquitecte	40.000	22,22	60.000	33,33	50.000	27,77
Analista	24.000	13,33	45.000	25	34.500	19,16
Programador	18.000	10	40.000	22,22	29.000	16,11

Taula 2. Salaris per rol

A més del salari per hora que rep el treballador, cal considerar el cost real que aquest suposa per l'empresa. Als valors anteriors cal afegir un percentatge que representa el cost de seguretat social que paga directament l'empresa. Per aquest cas, es determina que aquest cost representa el 35% del salari.

Rol	Salari mitjà per hora (€)	Percentatge seguretat social (%)	Cost per hora (€)
Cap de projecte	27,77	35	37,49
Arquitecte	27,77	35	37,49
Analista	19,16	35	25,87
Programador	16,11	35	21,75

Taula 3. Cost per rol

Un cop determinat el cost per hora d'aquests rols, es detallen tots els conceptes a considerar en l'estimació del pressupost.

4.1.1. Costos directes

A la següent taula s'estimen els costos directes per cada activitat definida al diagrama de Gantt.

Activitat	Hores	Rol	Cost(€)
Organització i planificació del projecte	70	Cap de projecte	2.624,3
Anàlisi de la situació actual	80	Arquitecte	2.999,2
Identificació dels problemes, anàlisi d'alternatives i proposta de solució	60	Arquitecte	2.249,4
Refinament dels requisits de qualitat	20	Analista	516,8
Disseny dels canvis	40	Arquitecte	1.499,6
Pla d'execució dels canvis	10	Cap de projecte	374,9
Implementació dels canvis	150	Programador	3.262,5

Validació dels requisits de qualitat	20	Analista	516,8
Finalització de la memòria	30	Cap de projecte	1.124,7
Preparació de la defensa	30	Cap de projecte	1.124,7
Total			**16.292,90 €**

Taula 4. Costos directes

4.1.2. Costos indirectes

En aquest apartat es calculen els costos indirectes del projecte, desglossats en les amortitzacions del recursos i les despeses generals.

Per les amortitzacions s'ha calculat el cost per hora basant-se en el preu unitari, la vida útil i considerant que un any té 225 jornades de 8 hores.

Per les despeses generals, es considera que els recursos materials que aquest projecte necessita i que consumeixen energia són l'ordinador portàtil i la pantalla externa. Segons les seves respectives especificacions, amb un ús normal l'ordinador portàtil consumeix 20W i la pantalla 16W. Per tant, en una hora generem un consum de 0,036 kWh. Tenint en compte que aquest projecte està estimat en unes 510 hores, el consum total estimat dels recursos materials que es produirà durant el desenvolupament és de 18,36 kWh.

Producte	**Quantitat**	**Preu unitari**	**Vida útil**	**Cost/hora**	**Hores**	**Cost (€)**
	Amortitzacions					
Ordinador portàtil	1 unitat	2.799 €	5 anys	0,311 €/h	510	158,61
Pantalla externa	1 unitat	199 €	5 anys	0,022 €/h	510	11,22
Teclat i ratolí	1 unitat	40 €	2 anys	0,011 €/h	510	5,61
Taula i cadira d'oficina	1 unitat	500 €	6 anys	0,046 €/h	510	23,46
IntelliJ IDEA Ultimate	1 unitat	499 €	1 any	0,277 €/h	510	141,27
Microsoft Office	1 unitat	50 €	1 any	0,028 €/h	510	14,28
	Despeses generals					
Connexió a internet	-	50 €/mes	-	0,208 €/h	510	106,08
Electricitat	18,36 kWh	0,12 €	-	-	-	2,20
Total						**462,73 €**

Taula 5. Costos indirectes

4.1.3. Contingències

Partint dels costos directes i indirectes, es fixa un marge del 10% com a contingència.

Tipus de cost	**Cost total (€)**	**Percentatge (%)**	**Cost (€)**
Costos directes	16.292,9	10	1.629,29
Costos indirectes	462,73	10	46,27
Total			**1.675,56 €**

Taula 6. Contingència

4.1.4. Imprevistos

Els imprevistos que es contemplen són avaries tècniques de l'equip informàtic (ordinador portàtil, pantalla externa, teclat i ratolí) i una previsió insuficient d'hores de

feina. Pel que fa a les avaries, com que tot l'equip informàtic està en garantia no s'imputa cap cost directe. Per la previsió insuficient d'hores, es comptabilitzen les hores extres que s'estima que hauria de fer cada rol. Es parteix de les hores estimades al Gantt i s'aplica un factor de desviació en funció de les tasques que executa cada rol. Per calcular el cost per rol es computa segons la probabilitat de que la desviació es faci efectiva.

Rol	Hores estimades Gantt	Desviació (%)	Hores extres	Cost per hora (€)	Probabilitat (%)	Cost (€)
Cap de projecte	140	20	28	37,49	10	104,97
Arquitecte	180	20	36	37,49	20	269,93
Analista	40	10	4	25,87	5	5,17
Programador	150	25	37,5	21,75	25	203,91
Total						**583,98 €**

Taula 7. Imprevistos

4.1.5. Total

A partir dels costos de les seccions anteriors, el cost total del pressupost per aquest projecte és el següent.

Activitat	Cost (€)
Costos directes	16.292,90
Costos indirectes	462,73
Contingències	1675,56
Imprevistos	583,98
Total	**19.015,17**

Taula 8. Pressupost total

4.2. Control de gestió

Per tal d'avaluar les desviacions que es puguin produir entre el pressupost i els costos reals, cal definir mecanismes de control. Es vol esbrinar on s'ha produït la desviació, quin és el motiu i la seva quantitat.

Es realitzarà un seguiment de les hores dedicades a cada fase per corroborar que s'ajusten a les de la planificació, i els rols que hi intervenen. També caldrà realitzar un control sobre els recursos materials per comprovar si hi ha despeses en recursos que no s'han tingut en compte. Per aquests propòsits s'utilitzaran els següents indicadors:

- Desviament de mà d'obra en preu = (cost estimat – cost real) * hores reals
- Desviament de mà d'obra en consum = (hores estimades – hores reals) * cost estimat
- Desviament d'una tasca en preu = (cost estimat – cost real) * hores reals
- Desviament d'una tasca en consum = (hores estimades – hores reals) * cost estimat
- Desviament de recursos en preu = (cost estimat – cost real) * hores reals

- Desviament total en mà d'obra = cost total estimat en mà d'obra – cost total real en mà d'obra
- Desviament total en tasques = cost total estimat en tasques – cost total real en tasques
- Desviament total en recursos = cost total estimat en recursos – cost total real en recursos

5. Contextualització i descripció del dashboard estratègic

Per tal de proporcionar un context més entenedor al *dashboard* estratègic, es comença per descriure el mètode i l'eina que s'ha desenvolupat en el projecte Q-Rapids. A continuació s'analitza amb detall el *dashboard* tractant aspectes de tecnologies, arquitectura, metodologia de treball i requisits de qualitat.

5.1. El mètode Q-Rapids

Un dels factors clau que actualment determinen l'èxit d'un projecte de desenvolupament de software és la seva qualitat. Per aquest motiu, és important disposar de requisits que s'alineïn amb aquesta idea i que tinguin com a objectiu potenciar aquest aspecte.

Tot i això, les metodologies de desenvolupament àgils actuals basades en cicles curts de desenvolupament ofereixen un suport limitat per assegurar uns certs nivells de qualitat. Tal i com els principis d'aquestes metodologies proposen, els requisits han de ser identificats a mesura que esdevenen necessaris, i definits en el moment adequat per tal de dur a terme el seu desenvolupament. Aquest concepte no és senzill de dur a la pràctica, i generalment els enginyers de requisits han de recórrer a la seva pròpia experiència per tal de realitzar correctament aquestes tasques.

Per tal de mitigar aquesta situació, han aparegut propostes que busquen explotar grans quantitats de dades heterogènies recollides de diverses fonts per tal d'extreure informació rellevant en el camp de l'enginyeria de requisits. L'eina Q-Rapids (Quality-Aware Rapid Software Development) s'emmarca en aquest moviment i té com a objectiu la identificació de candidats a requisits de qualitat i funcionals a partir de les dades obtingudes, tal i com es mostra a la Figura 4. L'impacte d'aquests requisits s'analitza per tal de decidir quina acció cal realitzar en funció dels resultats: adoptar el nou requisit, descartar-lo o posposar la decisió. [22]

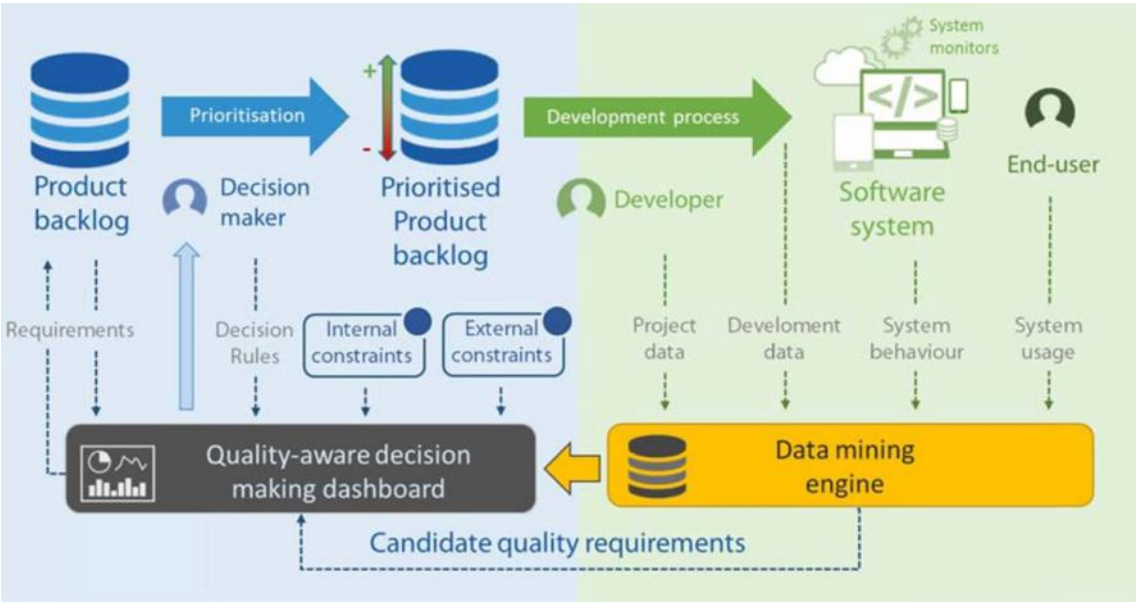

Figura 4. El mètode Q-Rapids [23]

L'eina té com a eix central el concepte de qualitat tant del producte final com del procés de desenvolupament, i busca potenciar-lo a través dels següents objectius [22]:

- Recollir i analitzar dades provinents d'eines de gestió de projectes, repositoris de codi, qualitat del servei i sistemes en ús. L'anàlisi d'aquestes dades permet avaluar de forma sistemàtica i contínua la qualitat del software utilitzant un conjunt d'indicadors relacionats amb la qualitat.

- Proporcionar a les persones encarregades de prendre decisions un *dashboard* que els ajudi a prendre decisions estratègiques en cicles curts, basades en dades i orientades envers a requisits. El *dashboard* agregarà les dades recollides en diversos indicadors estratègics com ara el temps de llançament al mercat, la productivitat de l'equip, la satisfacció dels usuaris i la qualitat general.

- Expandir el procés de desenvolupament àgil considerant la integració de requisits de qualitat i funcionals, de manera que s'afavoreixi la qualitat del software i es contribueixi a incrementar al productivitat del seu cicle de vida.

Per tal de poder definir indicadors estratègics, es defineix un model de qualitat que integra tant característiques abstractes de qualitat com mesures específiques (Figura 5). Permet l'agregació de dades recollides de diverses fonts per tal de poder generar els indicadors estratègics necessaris per avaluar la qualitat i prendre decisions. En concret, les dades sense tractar que generen les fonts s'utilitzen per calcular mètriques, que són posteriorment agregades formant factors de qualitat a nivell de producte i processos. De la mateixa manera, aquest factors són agregats formant indicadors estratègics. [24]

Figura 5. Model de qualitat [23]

Per tal d'il·lustrar el model de qualitat, a la Taula 9 es presenta un exemple dels factors, mètriques i fonts de dades que intervenen en la creació de l'indicador estratègic que mostra les situacions de bloqueig d'un projecte.

Indicador estratègic "Situacions de bloqueig"

Factors	Mètriques	Fonts de dades
Completesa de la definició de les funcionalitats	- Nombre de funcionalitats incompletes al *backlog* - Temps mitjà per completar la definició d'una funcionalitat	Funcionalitats que es troben definides a eines com JIRA, Redmine o Gitlab, amb informació referent al nom, estat, data, tipus, temps estimat, temps real invertit, dependències amb altres tasques i progrés actual, entre d'altres.
Tasques endarrerides	- Nombre de tasques bloquejades - Nombre de tasques de bloqueig - Temps d'espera per finalitzar tasques de bloqueig	
Proves automatitzades fallides	- Nombre de proves fallides - *Coverage* de les proves - Nombre de proves no executades	Proves executades per eines d'integració contínua, com Jenkins, amb informació referent al resultat, data, *coverage*, estat (executat o no) i temps d'execució.
Rendiment de les proves automatitzades	- Temps d'execució de les proves - Predicció del desenvolupament en següents iteracions a partir de l'estat de les proves	

Funcionalitats de baixa qualitat	- Temps per resoldre les violacions de regles de qualitat de les funcionalitats.	Informació extreta d'eines d'anàlisi estàtic de codi com SonarQube, on s'indiquen dependències entre tasques, violacions de regles de qualitat i complexitat del codi.

Taula 9. Factors, mètriques i fonts de dades presents en el càlcul de l'indicador estratègic que mostra les situacions de bloqueig

5.2. L'eina Q-Rapids

Per tal d'integrar el mètode Q-Rapids dins del cicle de vida d'un projecte software, s'està desenvolupat una eina que cobreix els seus dos primers objectius. A nivell funcional, podem dividir l'eina en cinc capes diferenciades [25]:

- **Productors de dades**: eines i sistemes que generen dades sense tractar per tal de ser analitzades dins de l'eina Q-Rapids.
- **Recopiladors de dades**: components encarregats de recollir les dades necessàries i d'actuar d'enllaç entre els productors de dades i l'eina Q-Rapids.
- **Magatzem distribuït de dades**: manteniment i indexació de les dades.
- **Anàlisi de dades, modelització i predicció**: càlcul de factors de qualitat i indicadors, generació de requisits de qualitat, predicció de qualitat i simulació.
- **Dashboard estratègic**: càlcul dels indicadors estratègics, presentació dels resultats i interacció amb l'usuari.

Considerant aquestes cinc capes, l'arquitectura conceptual de l'eina Q-Rapids es pot resumir amb el següent esquema (Figura 6):

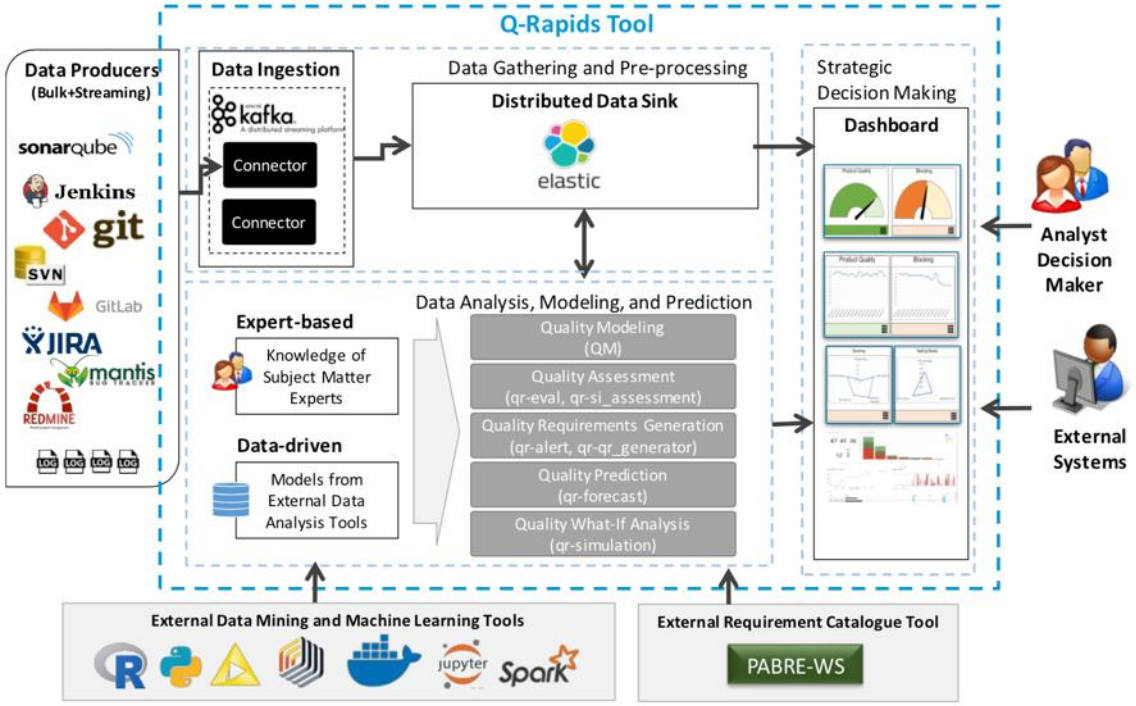

Figura 6. Arquitectura conceptual de l'eina Q-Rapids [26]

A continuació, es descriu amb més detall quin és el paper i les característiques de cadascuna de les capes que conformen l'eina Q-Rapids.

Productors de dades

Com ja s'ha esmentat, aquest és el nom amb el qual es coneixen les eines i sistemes que actuen com a fonts de dades per l'eina Q-Rapids. Es tracta d'un conjunt heterogeni, format per repositoris de codi (SVN, Git, GitLab), eines de gestió d'incidències (Redmine, GitLab, JIRA, Mantis), eines d'anàlisi estàtic de codi (SonarQube) i eines d'integració contínua (Jenkins). [26]

Recopiladors de dades

Aquesta capa consisteix en un conjunt de connectors d'Apache Kafka [27] encarregats d'obtenir la informació necessària dels productors de dades, utilitzant les seves respectives APIs.

Apache Kafka és una plataforma de transmissió de dades distribuïda, que ofereix funcionalitats d'enviament de missatges basades en un esquema de publicació i subscripció així com en un esquema de cua, a més d'oferir avantatges com ara tolerància a fallades, escalabilitat horitzontal i alt rendiment en dades en temps real. S'utilitza primordialment com a passarel·la de dades en temps real entre sistemes o aplicacions i per construir aplicacions que transformen o operen amb dades en temps real.

Els connectors han estat adaptats a les diverses eines i sistemes productors de dades per tal d'integrar-se amb diferents versions de cadascuna i extreure'n les metadades correctes en cada cas. [26]

Magatzem distribuït de dades

El següent pas després de la recopilació de les dades és la seva indexació. Per aquest propòsit, la capa que actua com a magatzem distribuït de dades utilitza ElasticSearch [28] com a eix de les seves funcions, juntament amb Kibana com a interfície gràfica.

ElasticSearch és un motor de cerca orientat a documents (les dades s'emmagatzemen en format JSON) que permet processar consultes i indexar dades. Permet treballar amb dades no estructurades com ara text lliure, a més d'oferir escalabilitat horitzontal. A més, aquesta capa també ofereix visualització de les dades emmagatzemades a través de Kibana, que actua com a interfície d'usuari conjuntament amb ElasticSearch oferint la possibilitat de creació de *dashboards* formats per diferents gràfics i taules, i la construcció i execució de consultes. La creació d'aquests *dashboards* és simple i no requereix programació, i s'ofereixen diverses plantilles adaptables a cada cas d'us.

L'ElasticSearch de l'eina Q-Rapids agrupa les dades en quatre índexs, seguint el model de qualitat [26]:

- Indicadors estratègics
- Factors de qualitat
- Mètriques normalitzades
- Dades sense tractar (obtingudes dels productors de dades)

Anàlisi de dades, modelització i predicció

Un cop les dades ja han estat recollides i es troben a l'ElasticSearch, s'ha de procedir a utilitzar-les per extreure informació rellevant. Per aquest propòsit, es defineix un model de qualitat basat en el coneixement d'experts i que va evolucionant amb l'ajut d'eines externes d'anàlisi i tècniques basades en dades. Aquest model és utilitzat per aconseguir els diferents objectius d'aquesta capa: avaluació, predicció i simulació de la qualitat, i generació de requisits de qualitat. [29]

El model de qualitat és l'element central de la metodologia que proposa Q-Rapids. L'objectiu d'aquest model és capturar les dependències més importants entre aspectes rellevants del desenvolupament i el seu context i la qualitat del software produït. Les dades d'entrada del model són característiques mesurables de diferents artefactes de desenvolupament, producte i processos, i la sortida correspon als valors dels indicadors estratègics. La idea bàsica del mètode Q-Rapids és utilitzar un model de qualitat específic pel seu context, adaptat a característiques particulars en qüestions de desenvolupament de software i l'entorn d'ús.

El primer objectiu que es busca aconseguir és l'avaluació de la qualitat. Es tracta d'estimar el valor actual d'un indicador estratègic a partir de les dades actuals. Per aquest propòsit, els valors actuals de les mètriques extretes de les eines i entorn són utilitzats com a dades d'entrada pel model de qualitat, que posteriorment calcula els valors estimats dels indicadors estratègics. Per aquest objectiu, no cal utilitzar cap model addicional.

El següent objectiu és la predicció de la qualitat, que busca predir el valor futur dels indicadors estratègics. Això s'aconsegueix a partir de prediccions de les mètriques a les quals se les aplica el model de qualitat per derivar-ne els corresponents indicadors estratègics. Per tal de poder dur-ho a terme, cal crear un model addicional de predicció a partir de dades històriques, amb l'ajut d'eines externes de predicció.

L'objectiu de simulació de qualitat té com a finalitat la investigació de l'impacte en els indicadors estratègics que tenen diferents escenaris relatius a aspectes de desenvolupament, producte i processos. Els dos principals beneficis que se n'extreuen són els valors simulats dels indicadors estratègics i l'anàlisi de l'impacte de cadascun dels aspectes sobre els indicadors. Els usuaris poden utilitzar les simulacions dels indicadors estratègics per identificar quins són els valors adequats de les mètriques que permeten obtenir el valor desitjat de l'indicador. A més, també es pot analitzar l'impacte que té cada mètrica en els indicadors estratègics per tal d'identificar aquelles que tenen una major rellevància.

Per acabar, aquesta capa té com a objectiu la generació de requisits de qualitat pel desenvolupament, per tal de mantenir la qualitat del producte i processos de desenvolupament mantenint els valors desitjats dels indicadors estratègics corresponents. Per tal de generar-los, es consideren els elements del model de qualitat de forma independent entre ells, i en deriva els requisits de qualitat adequats en funció d'uns llindars predefinits. Si el valor d'una mètrica, un factor o un indicador estratègic se situa fora dels valors esperats, es genera un requisit de qualitat utilitzant una plantilla amb el contingut adequat.

Dashboard estratègic

L'eina Q-Rapids ofereix una aplicació web que permet la visualització de les dades generades i l'aplicació de les diferents tècniques d'anàlisi, modelització i predicció mencionades anteriorment [25]. Aquestes funcionalitats poden ser accedides per usuaris finals a través d'una interfície d'usuari, o bé per serveis externs a través de serveis REST.

Les funcionalitats principals que ofereix el dashboard són les següents:

- Visualització i exploració de l'avaluació de qualitat actual, així com navegació entre elements.
- Generació de prediccions de qualitat basades en l'avaluació actual i utilitzant diverses tècniques.
- Generació de simulacions de qualitat basades en escenaris proposats per l'usuari final, consistents en la modificació dels valors de mètriques i factors de qualitat per determinar quin impacte tenen en els indicadors estratègics.
- Suggeriment de requisits de qualitat per corregir desviacions en l'avaluació de qualitat, i notificació d'aquestes situacions per mitjà d'alertes.

Per il·lustrar aquestes funcionalitats, es presenten dos exemples. A la Figura 7 es pot veure la pantalla principal del *dashboard*, on es mostra l'avaluació actual dels indicadors estratègics per un projecte concret juntament amb colors que representen les seves categories, així com informació sobre quan es publicarà la pròxima versió del mateix.

Figura 7. Vista dels indicadors estratègics del dashboard

A la Figura 8 es mostra la vista del *dashboard* que, dins de l'apartat de simulació, s'encarrega de simular el resultat dels indicadors estratègics i els factors de qualitat a partir de la introducció d'un nou requisit de qualitat associat a una mètrica, on l'usuari pot ajustar el valor que aquesta ha de tenir per veure els efectes que això provoca.

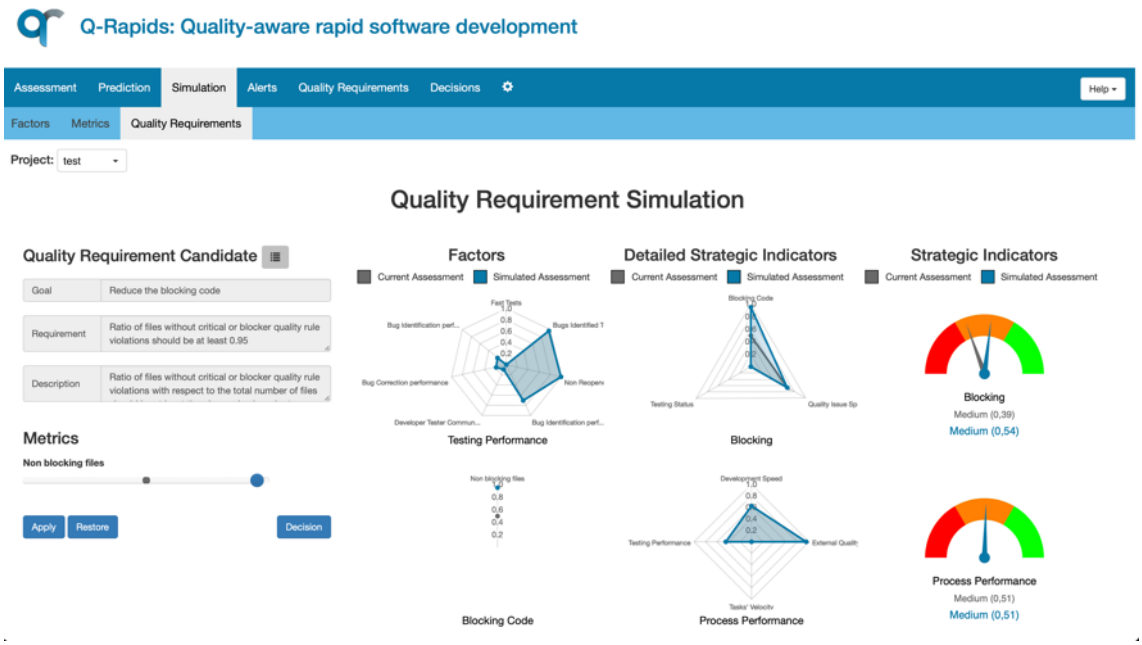

Figura 8. Vista de simulació dels indicadors estratègics i els factors a partir d'un requisit de qualitat

Per tal de mostrar les dades corresponents a l'avaluació de qualitat actual i a la predicció, el dashboard proporciona diverses vistes amb diferents nivells de detall:

- Indicadors estratègics: vista general dels indicadors estratègics, on per cadascun d'ells es mostra el valor de l'avaluació de qualitat.
- Indicadors estratègics detallats: per cada indicador, es mostra informació sobre l'avaluació dels factors de qualitat que intervenen en el seu càlcul.
- Factors de qualitat: per cada factor, es mostra informació sobre l'avaluació de les mètriques que intervenen en el seu càlcul.
- Mètriques: vista general de les mètriques, on per cadascuna d'elles es mostra el valor de l'avaluació de qualitat.

A més, les vistes de l'avaluació de qualitat actual es poden mostrar de forma gràfica (utilitzant gràfiques de diferents tipus per cada nivell de detall) o bé de forma textual (utilitzant taules amb les descripcions i valors de cada element). Paral·lelament, aquestes vistes poden mostrar valors corresponents a l'avaluació actual o bé mostrar un històric de les avaluacions.

De nou, es mostren dos exemples més de les vistes que formen part del *dashboard*. A la Figura 9 es pot observar la vista del *dashboard* que s'encarrega de mostrar l'avaluació actual pels indicadors estratègics detallats, on es mostra l'estat de cadascun dels factors de qualitat que els composen.

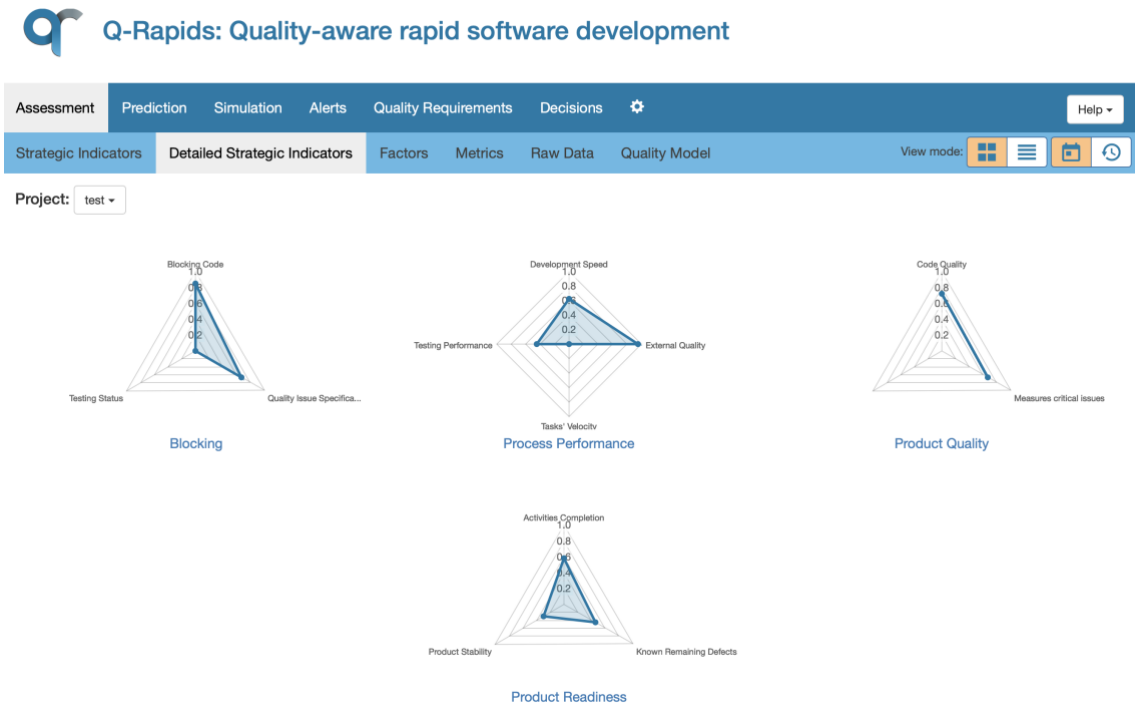

Figura 9. Vista de l'avaluació actual dels indicadors estratègics detallats

A la Figura 10 es mostra de nou la vista dels indicadors estratègics detallats, però aquest cop mostrant l'històric d'avaluacions en forma de gràfiques. Per cada indicador estratègic, es mostren les avaluacions dels seus factors de qualitat, així com els llindars que representen les seves categories.

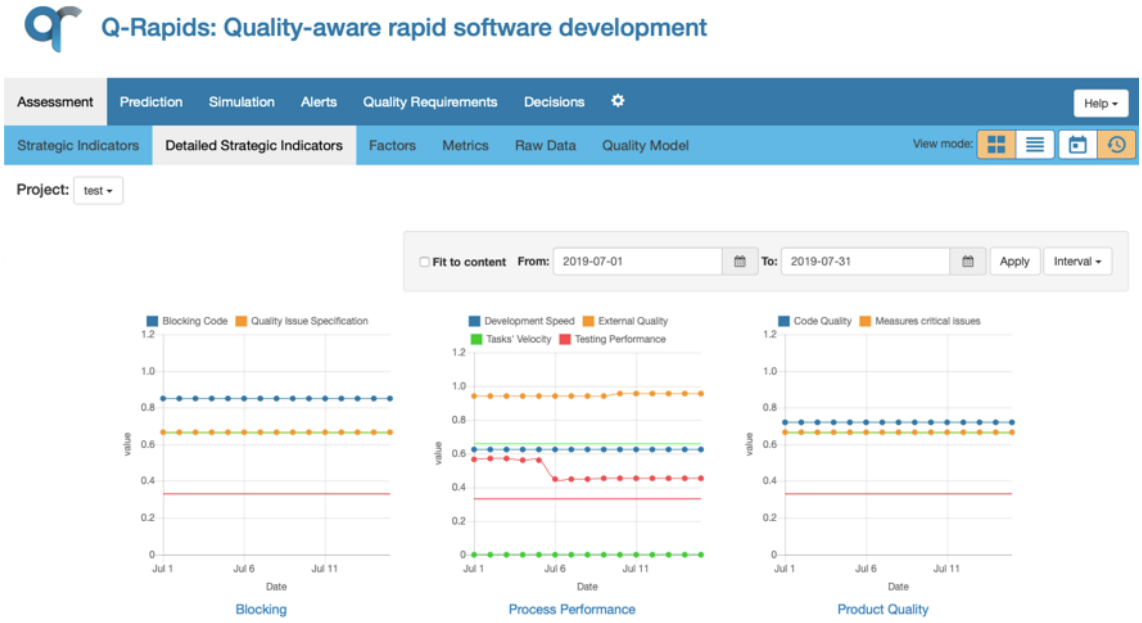

Figura 10. Vista de l'avaluació històrica dels indicadors estratègics detallats

5.3. El dashboard estratègic de Q-Rapids

Un cop vist el mètode i l'eina Q-Rapids d'una forma general, cal focalitzar l'anàlisi en l'element central del procés de reenginyeria, el *dashboard* estratègic. Per fer-ho es detallen les tecnologies utilitzades, l'arquitectura interna del dashboard, la metodologia de treball que se segueix en el seu desenvolupament i els requisits de qualitat presents actualment.

5.3.1. Tecnologies

Spring

La tecnologia principal que utilitza el *dashboard* estratègic de l'eina Q-Rapids és Spring [30]. Es tracta d'un marc de treball o *framework* que proporciona un model de programació i configuració per aplicacions basades en Java. Proporciona suport a nivell d'infraestructura de l'aplicació, centrant-se en aspectes transversals per tal que l'usuari es pugui centrar en la lògica de negoci del sistema.

D'entre les diferents funcionalitats principals que ofereix Spring, com per exemple manipulació i accés a recursos, validació d'objectes, vinculació dinàmica de dades, conversió automàtica de tipus de dades i suport pel model de programació orientat a aspectes, hi ha una que en destaca per sobre de la resta: la injecció de dependències.

Per parlar de la injecció de dependències, primer cal veure el principi d'inversió de control. Es tracta d'un principi de l'enginyeria del software principalment utilitzat en el context de la programació orientada a objectes, per mitjà del qual el control dels objectes o porcions d'un programa és transfereix a un contenidor o un *framework*. A diferència de la programació tradicional en la qual el codi escrit realitza crides a una llibreria externa, el principi d'inversió de control permet a un *framework* prendre el control del flux del programa i realitzar crides al codi de l'usuari [31]. Un dels mecanismes per aplicar el principi d'inversió de control és la injecció de dependències.

La injecció de dependències és un patró que permet invertir el control de l'assignació de les dependències d'un objecte. Enlloc de crear les dependències per si mateix, un objecte depèn d'un altre entitat per tal de proporcionar els objectes que conformen les seves dependències. Spring disposa d'un contenidor que s'encarrega d'instanciar, configurar i relacionar objectes entre si, a més de gestionar el seu cicle de vida. Aquest contenidor és el responsable d'injectar les dependències dels objectes, i ho pot fer per mitjà de tres tècniques diferents: a través dels constructors dels objectes proporcionant la dependència en el moment de la creació, a través de mètodes *setter*[1] del propi objecte proporcionant la dependència un cop aquest ja ha estat creat i a través de reflexió[2] en cas que l'objecte creat no disposi de constructor o mètode *setter* per injectar les dependències.

Cal destacar també que la injecció de dependències és un patró que ajuda a construir codi que segueixi el principi d'inversió de dependències. Aquest principi permet reduir

[1] Mètodes que permeten modificar una propietat d'un objecte.
[2] Habilitat d'un programa de manipular com a dades quelcom que representa l'estat del programa mentre aquest s'executa.

l'acoblament per mitjà de la introducció d'abstraccions entre mòduls d'alt nivell (aquells que contenen lògica complexa) i baix nivell (aquells que proporcionen utilitats menors, les dependències) [32].

A més, Spring també integra altres funcionalitats que afegeixen valor a la plataforma. En el camp de les proves, a més dels avantatges que la injecció de dependències comporta en les proves unitàries, Spring proporciona eines per realitzar proves d'integració de manera senzilla i àgil. Pel que fa a l'accés a dades, Spring integra un servei propi de gestió de transaccions, a més de poder treballar amb diverses eines i tecnologies externes d'accés a dades. Dins del món web, un dels marcs de treball que Spring integra és Spring MVC, que dóna suport a la construcció d'aplicacions web seguint el patró Model-Vista-Controlador.

Spring Boot

La següent tecnologia utilitzada dins del dashboard és Spring Boot [33]. Es tracta d'una eina que permet crear aplicacions i serveis basats en la plataforma Spring d'una forma senzilla i pràctica, reduint en gran mesura la configuració necessària per fer-les funcionar.

Els objectius principals de Spring Boot són els següents:

- Permetre un accés ràpid i senzill al desenvolupament d'aplicacions amb Spring.
- Establir una elecció de dependències bàsiques per tal de simplificar el procés de desenvolupament, però amb facilitat per adaptar-les segons els requisits concrets de cada sistema.
- Proporcionar un ampli ventall de característiques no funcionals adequades per una gran varietat de projectes, com ara servidors incrustats, seguretat, mètriques, verificació de l'estat de l'aplicació i configuració externa.
- Evitar la incorporació de codi generat automàticament, així com la configuració a través d'arxius XML.

Concretament, dins del servidor del dashboard estratègic, Spring Boot s'utilitza principalment per facilitar la configuració. Aquesta eina proporciona diversos conjunts de dependències anomenats *starters*, que s'encarreguen no només d'incloure les dependències adequades sinó també de configurar-les requerint una mínima interacció per part de l'usuari. Alguns dels *starters* més utilitzats i presents dins del *dashboard* són els següents:

- *spring-boot-starter-web*: Permet la creació d'aplicacions web i serveis REST.
- *spring*-boot-starter-security: Permet l'autenticació i autorització d'usuaris dins d'una aplicació.
- *spring*-boot-starter-data-jpa: Permet la utilització de la interfície de persistència de dades de Java amb el sistema de mapeig objecte/relacional Hibernate.
- *spring-boot-starter-test*: Permet la realització de proves unitàries i d'integració.

Hibernate

Per tal de poder dur a terme algunes de les seves funcionalitats, el *dashboard* estratègic necessita guardar algunes de les dades que tracta per la seva posterior utilització. Per poder realitzar aquesta tasca, s'utilitza un sistema de mapeig objecte/relacional (*Object/Relational Mapping* o ORM) anomenat Hibernate [34]. Un sistema ORM permet treballar amb una base de dades relacional a partir d'un model orientat a objectes, automatitzant i solucionant alguns dels problemes que apareixen quan el model relacional i l'orientat a objectes conviuen.

En concret, Hibernate és una implementació de la interfície de persistència de dades de Java (*Java Persistence API* o JPA), de manera que pot ser utilitzat amb facilitat en entorns que accepten aquesta interfície, com és el cas de Spring. Amb Hibernate es poden crear classes persistents seguint les tendències naturals en l'orientació a objectes, com ara herència, polimorfisme, associacions i composicions, així com utilitzar les estructures de dades i col·leccions que defineix Java.

Per tal d'interactuar amb la base de dades, Hibernate proporciona una sèrie de mètodes que permeten realitzar operacions sense necessitat d'haver d'escriure cap mena de codi relacionat amb la base de dades. En cas de que aquests no siguin convenients per l'usuari, se'n poden crear de propis definint directament el comportament desitjat.

PostgreSQL

Per tal de donar suport a la persistència de dades, el *dashboard* utilitza PostgreSQL [35] com a sistema gestor de bases de dades. Es tracta d'un sistema objecte-relacional, que manté l'estructura d'una base de dades relacional però integra els models propis de les bases de dades orientades a objectes, com ara objectes, classes i herència.

PostgreSQL és un sistema àmpliament utilitzat degut a la seva bona arquitectura, seguretat, integritat de les dades, robust conjunt de funcionalitats i extensibilitat. Pot funcionar amb tots els sistemes operatius predominants, té un nivell alt d'adequació al estàndard SQL, compleix les propietats ACID (Atomicitat, Consistència, Isolament i Durabilitat) i permet la incorporació d'extensions per afegir noves funcionalitats.

Dins del *dashboard*, es fa un ús relativament bàsic de la base de dades, de manera que les propietats que s'aprofiten de PostgreSQL són les que fan referència al model relacional, sense entrar en l'orientació a objectes. Tal i com s'ha comentat a l'apartat anterior, es fa servir Hibernate per interactuar amb PostgreSQL.

Gradle

Amb l'objectiu de gestionar les dependències i la compilació del dashboard, s'utilitza una eina d'automatització de la construcció del codi anomenada Gradle [36]. Aquesta eina utilitza un llenguatge de domini específic (*Domain-Specific Language* o DSL) clar i entenedor que permet definir la configuració d'un projecte amb facilitat. Està dissenyat per funcionar amb un gran nombre de llenguatges de programació, no només amb Java, i disposa d'un sistema de gestió de dependències sòlid.

L'arxiu on es defineix la configuració d'un projecte s'anomena *build.gradle*, i dins d'aquest la unitat bàsica de treball són les tasques o *tasks*. Cada una d'aquestes tasques és un bloc de codi que executa una o diverses operacions bàsiques. Tot projecte disposa d'una seria de tasques per defecte, però podem modificar l'arxiu *build.gradle* per afegir-n'hi de noves. Les tasques prenen forma de graf dirigit acíclic (*Directed Acyclic Graph* o DAG) per determinar el seu ordre d'execució.

Dins del *dashboard*, Gradle permet definir totes les dependències necessàries pel projecte, així com aspectes relatius a la configuració general del projecte i a tasques específiques, com ara la creació d'un arxiu en format *war* per ser desplegat a un servidor.

5.3.2. Arquitectura

A nivell arquitectònic, el *dashboard* estratègic està compost per diversos components funcionals [25]. D'entre ells, el principal és el *qr-dashboard*, que integra l'aplicació web i proporciona la interfície pels usuaris i pels sistemes externs. Els altres components s'encarreguen de proporcionar diversos serveis que consumeix el *qr-dashboard*, necessaris pel seu funcionament. A la següent figura (Figura 11) es pot veure quins són aquests components i la relació que existeix entre ells:

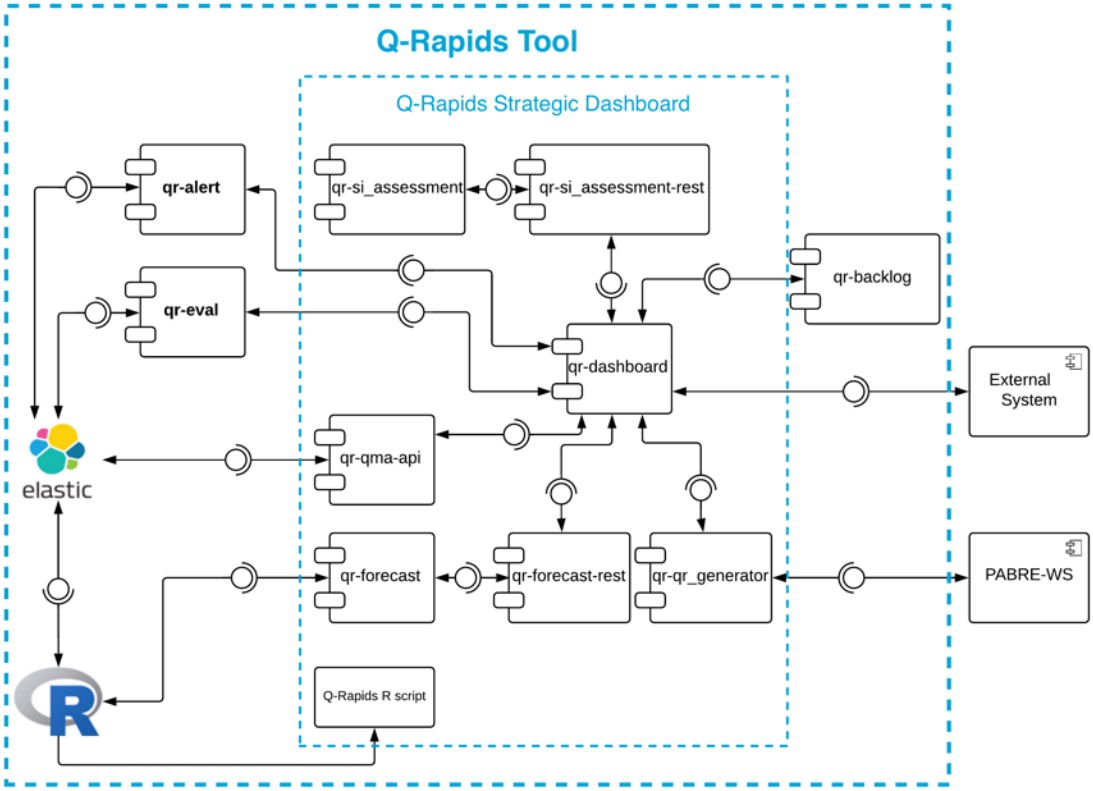

Figura 11. Arquitectura del dashboard estratègic [25]

A continuació es descriuen breument els components perifèrics del *dashboard* estratègic [25]:

- ***qr-qma-api***: El component *qr-qma-api* és una llibreria que s'encarrega de fer transparent l'accés al magatzem distribuït de dades. Aquest component llegeix les avaluacions dels indicadors estratègics, els factors de qualitat i les mètriques

del ElasticSearch i les transmet en un format específic. Aquesta llibreria està integrada al *qr-dashboard.*

- ***qr*-si_assessment** i ***qr-si_assessment-rest***: El component *qr-si_assessment* és una llibreria encarregada de calcular les avaluacions dels indicadors estratègics quan aquests estan definits mitjançant una xarxa Bayesiana. El component *qr-si_assessment-rest* és un servei web que integra la llibreria *qr-si_assessment* i proporciona accés a les seves funcionalitats per mitjà de serveis REST.

- ***qr-forecast*** i ***qr-forecast-rest***: El component *qr-forecast* és una llibreria que s'encarrega de computar prediccions per les avaluacions de factors de qualitat i mètriques. Extreu les dades actuals de l'ElasticSearch, i disposa de diverses tècniques de predicció. El component *qr-forecast-rest* és un servei web que integra la llibreria *qr-forecast* i proporciona accés a les seves funcionalitats per mitjà de serveis REST.

- ***qr-alert***: El component *qr-alert* s'encarrega de monitoritzar i crear alertes de qualitat en base a llindars predefinits sobre els elements del model de qualitat. Quan se sobrepassa un d'aquests llindars, s'envia al *qr-dashboard* la informació pertinent.

- ***qr-qr_generator*** i ***PABRE-WS***: El component *qr-qr_generator* és una llibreria que s'encarrega de suggerir requisits de qualitat en base a la informació que proporciona el component *qr-alert*. Utilitza l'eina externa PABRE-WS per identificar els potencials requisits de qualitat que poden contribuir a millorar els problemes de qualitat.

- ***qr-eval***: El component *qr-eval* realitza les avaluacions del model de qualitat a partir de les dades recollides de les eines i sistemes utilitzats per les empreses. Els càlculs de les avaluacions de mètriques i factors de qualitat es realitzen directament en el component *qr-eval*, però les avaluacions dels indicadors estratègics es realitzen en el component *qr-dashboard*, mitjançant un servei que activa el propi *qr-eval*.

- ***qr-backlog***: El component *qr-backlog* és un servei web accessible a través d'un servei REST que s'encarrega de transformar els requisits de qualitat generats pel *dashboard* estratègic en ítems d'una determinada plataforma externa de gestió de projectes.

Un cop analitzats els components perifèrics, el següent pas és analitzar amb detall el component principal del *dashboard* estratègic, el *qr-dashboard*.

Tal i com s'ha esmentat anteriorment, el *qr-dashboard* és una aplicació web que s'encarrega d'integrar totes les funcionalitats que el *dashboard* estratègic proporciona, i permet accedir a elles per mitjà d'una interfície gràfica pels usuaris, i serveis REST pels sistemes externs que vulguin aprofitar les seves funcionalitats.

Per parlar de l'arquitectura del *qr-dashboard*, primer cal destacar que aquest component està dissenyar seguint una arquitectura en tres capes. La primera d'aquestes és la capa de presentació, encarregada de mostrar les dades d'una forma gràfica i

permetre la interacció dels usuaris. A continuació es troba la capa de domini, que integra la lògica de l'aplicació i permet executar les seves funcionalitats. Per últim, la capa de dades s'encarrega de gestionar la persistència i proporcionar accés a les dades necessàries pel funcionament del sistema. A la següent figura (Figura 12) es mostra un diagrama corresponent a les capes esmentades, així com als diferents paquets que les conformen i les relacions entre ells:

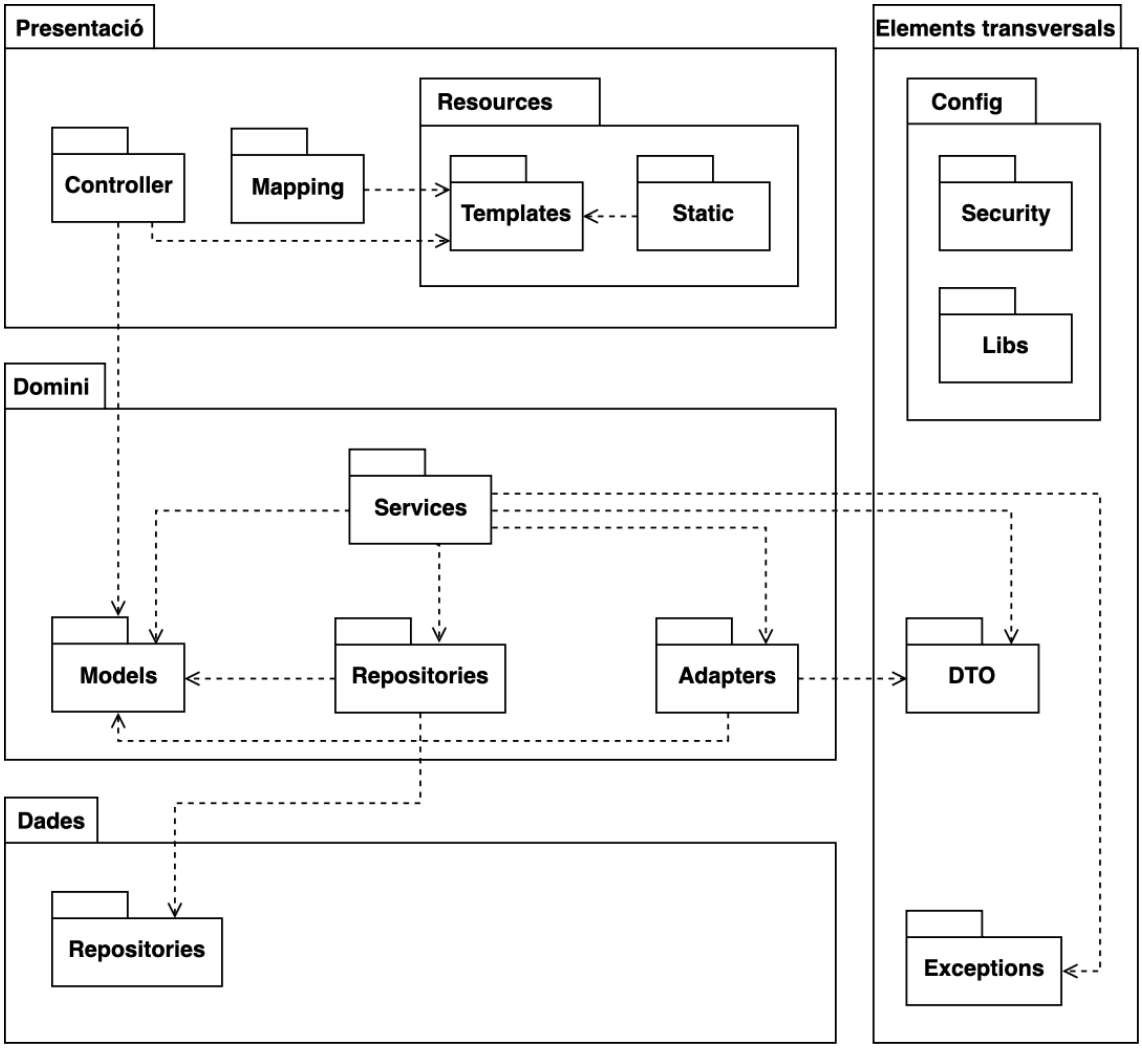

Figura 12. Arquitectura del qr-dashboard

Capa de presentació

La següent apreciació que cal fer és que, com a aplicació web, el *qr-dashboard* segueix el paradigma client-servidor. Això implica que hi ha una part del sistema que s'executa en els clients, és a dir, en els navegadors web, i que s'encarrega de mostrar els resultats i permetre la interacció amb l'usuari. Per altra banda, el sistema també consta d'una part que s'executa en el servidor, i que conté la lògica de l'aplicació i la persistència de les dades. Tal i com es menciona a l'apartat 2.2 d'aquest document, el procés de reenginyeria que es vol aplicar s'acotarà a la part corresponent al servidor del *dashboard*. Tot i això, la frontera entre el client i el servidor queda lleugerament diluïda degut a que el *qr-dashboard* treballa seguint dos enfocaments diferents a l'hora de presentar les dades: renderització en el servidor i renderització en el client.

Amb la renderització en el servidor, quan el navegador web demana per una ruta concreta de l'aplicació web, el servidor respon amb una pàgina completament formada que ja inclou totes les dades necessàries per la seva correcta visualització. El servidor ha executat la lògica i ha accedit a les dades necessàries, i ha construït la pàgina abans d'enviar-la cap al navegador web (Figura 13). Seguint aquest enfocament, la capa de presentació es troba al servidor, i tota la funcionalitat inherent a aquesta capa s'executa allà. El paper del client queda rellevat a ser un intèrpret de les dades enviades pel servidor, mostrant els elements gràfics que formen la vista i executant petites funcionalitats com ara la comprovació del text introduït per l'usuari.

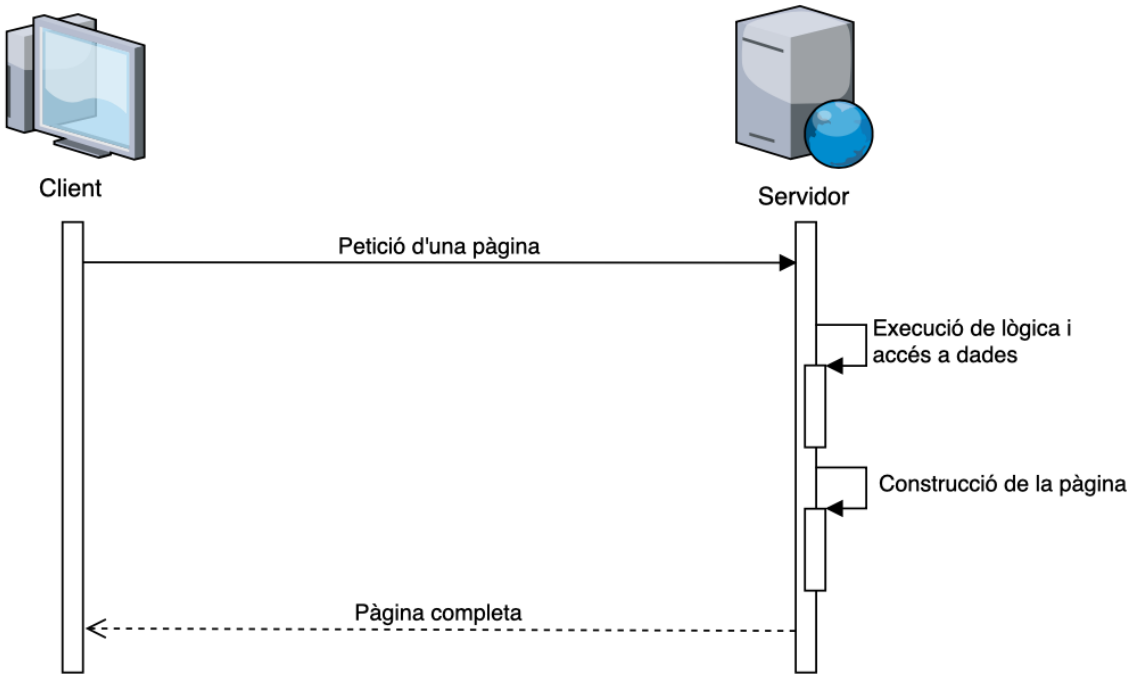

Figura 13. Renderització en el servidor

En canvi, amb la renderització en el client succeeix una cosa diferent. Quan el navegador web demana per una ruta, el servidor respon amb una pàgina que conté únicament un esquelet molt bàsic de la mateixa, juntament amb codi Javascript. Aquest codi s'executarà al navegador web i serà l'encarregat de construir la pàgina, creant les seves estructures internes de manera dinàmica i carregant les dades necessàries en base a les interaccions que l'usuari realitzi. Cal parar atenció a aquest darrer pas, ja que per poder carregar les dades el navegador web haurà de realitzar crides addicionals a través d'internet al servidor demanant l'enviament dels recursos necessaris (Figura 14). Amb aquest enfocament, la capa de presentació es troba localitzada als clients, i el seu paper consisteix no només en interpretar i mostrar de forma gràfica el contingut sinó també en construir els elements i obtenir les dades necessàries.

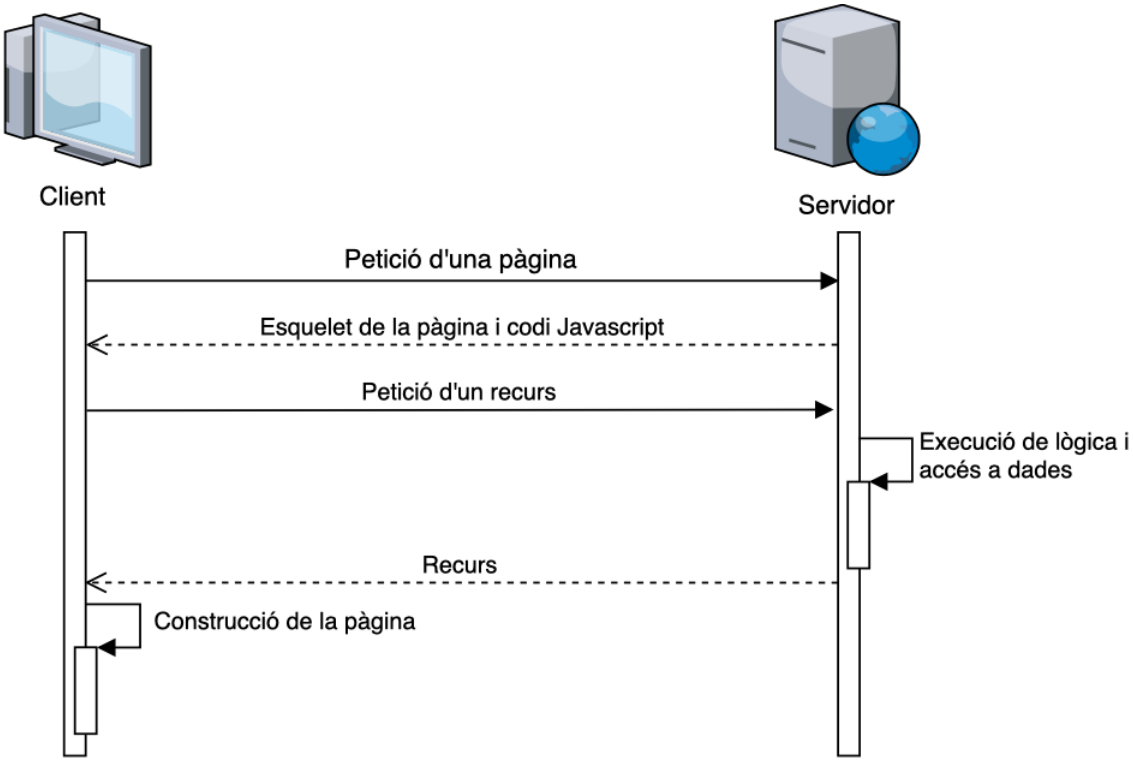

Figura 14. Renderització en el client

El *qr-dashboard* combina aquests dos enfocaments en la seva arquitectura interna, a la part que correspon a la capa de presentació. Tot i que el que predomina àmpliament és la renderització en el client, les funcionalitats relacionades amb l'autenticació i el control d'usuaris treballen realitzant la renderització en el servidor. Per aquest propòsit, es fa servir el patró Model-Vista-Controlador per tal de construir les vistes, aprofitant les característiques que proporciona Spring MVC per treballar amb aquest patró (Figura 15). Quan es produeix una petició, el controlador situat dins del paquet *Controller* executa la lògica necessària i accedeix a les dades que representen el model. A continuació, el controlador carrega la vista requerida situada dins del paquet *Templates*, que consisteix a un embolcall buit amb codi que representa la seva aparença i funcionament, però sense dades. El darrer pas consisteix en omplir la vista, vinculant les dades obtingudes anteriorment amb el seu embolcall, i enviar la vista completament formada cap al client.

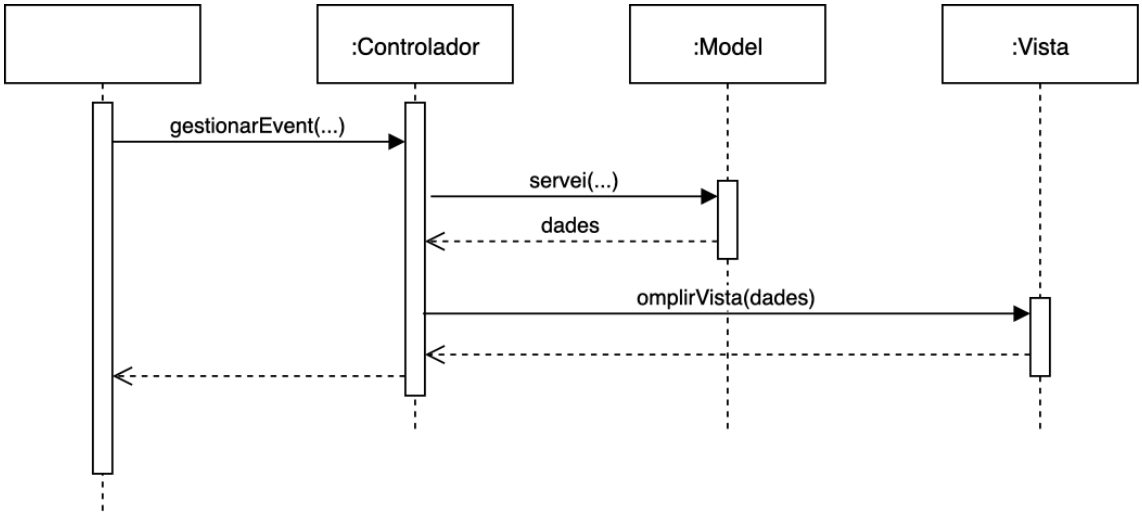

Figura 15. Diagrama de seqüència del patró Model-Vista-Controlador

Per la resta de funcionalitats que treballen amb renderització en el client, la capa de presentació consisteix en vistes molt simples que seran creades i omplertes amb detall per mitjà d'arxius Javascript que contenen tota la funcionalitat necessària i que seran executats pels navegadors web. Quan es produeix una petició, els controladors que es troben dins del paquet *Mapping* només s'encarreguen d'enviar la vista amb els arxius incrustats, sense executar cap mena de lògica. Tal i com s'ha comentat anteriorment, aquesta part del *qr-dashboard* queda fora de l'abast del procés de reenginyeria.

Capa de domini

Un cop vista la capa de presentació, la següent capa que cal analitzar és la de domini. Aquesta és la capa que conté la lògica i els elements funcionals de l'aplicació, que aporten les funcionalitats pròpies del dashboard. Aquesta capa consta de diversos paquets d'elements, agrupats segons la responsabilitat que recau en cadascun d'ells.

El primer paquet d'elements s'anomena *Models*. Es tracta de les classes d'objectes que representen el domini de l'aplicació, és a dir, les entitats amb les que treballa el sistema. Es tracta de classes sense funcionalitat pròpia, únicament defineixen les propietats dels objectes en qüestió i en permeten la seva modificació i consulta. Una àmplia majoria dels objectes definits per les classes d'aquest paquet hauran de tenir persistència. Per aquest propòsit s'utilitzen les funcionalitats que proporciona Hibernate per tal de definir les taules que formaran la base de dades, per mitjà d'anotacions a les classes que permeten definir el nom, les columnes, els identificadors i les claus foranes de les taules necessàries. D'aquesta manera, tan bon punt l'aplicació s'engegui i es connecti a una base de dades buida, les taules es crearan amb l'esquema definit.

El següent paquet present a la capa de domini s'anomena *Repositories* i conté les interfícies dels repositoris, que són les entitats encarregades de gestionar l'accés a la base de dades. Per tal d'accedir a aquesta informació, es defineixen interfícies per a cadascuna de les entitats del model que permeten persistència de dades (Figura 16). En concret, cada entitat compta amb dues interfícies creades manualment, que segueixen la nomenclatura *Repository* i *CustomRepository*, juntament amb el nom de l'entitat. La primera d'elles estén tres interfícies, dues d'elles proporcionades automàticament per la *Java Persistence API*, anomenades *JpaRepository* i *PagingAndSortingRepository*, que proporcionen operacions bàsiques de persistència i de paginació i ordenació respectivament, i no requereixen realitzar cap implementació per tal de beneficiar-se de les seves funcionalitats. A més, aquesta primera interfície també estén la *CustomRepository*, que consisteix en la definició manual de signatures de mètodes addicionals per tal d'accedir a les dades.

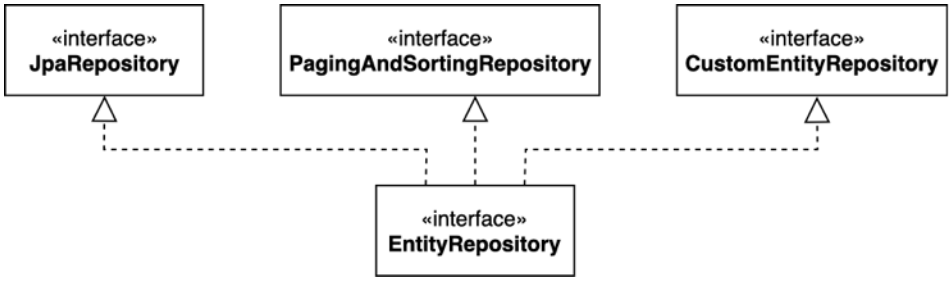

Figura 16. Diagrama d'interfícies dels repositoris

El tercer paquet que constitueix la capa de domini s'anomena *Adapters*. Consisteixen en classes que segueixen el patró adaptador (Figura 17), i actuen com a intermediaris dels components externs al *qr-dashboard* proporcionant una manera d'accedir a les seves funcionalitat des de diferents parts de l'aplicació sense necessitat d'acoblar-se amb aquests components directament. En concret, aquest paquet compta amb adaptadors per la llibreria *qr-qma-api* i pels serveis web *qr-forecast-rest* i *qr-si_assessment-rest*. Com que aquests dos darrers són serveis web, els adaptadors s'encarreguen de realitzar la corresponent crida a través de la xarxa i recollir-ne els resultats.

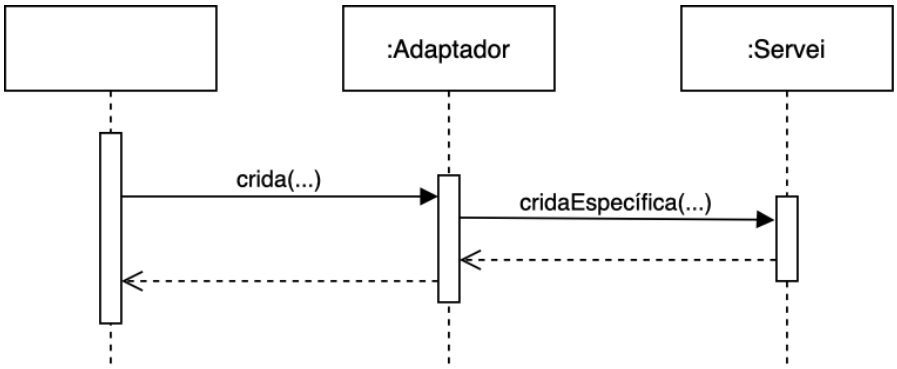

Figura 17. Diagrama de seqüència del patró adaptador

Per últim, la capa de domini compta amb el paquet anomenat *Services*. Es tracta de controladors que s'encarreguen d'associar URLs en format REST amb mètodes que executen una determinada funcionalitat. Aquests controladors fan servir els repositoris per accedir a la base de dades i els adaptadors per accedir a les funcionalitats externes, i cada mètode inclou la lògica necessària per dur a terme la funcionalitat de la que és responsable. Cadascun dels serveis agrupa mètodes segons les entitats amb les que tracta, tot i que també existeix un servei addicional que conté diversos mètodes que no han pogut ser encaixats en un altre servei més adient.

Capa de dades

Seguint l'arquitectura en tres capes, la darrera que cal analitzar és la capa de dades. Dins del *qr-dashboard*, aquesta capa inclou el paquet *Repositories* que conté les implementacions d'algunes de les interfícies definides a la capa de domini corresponents als repositoris (Figura 18). Concretament, les interfícies que s'implementen són les *CustomRepository*, que defineixen mètodes per accedir a les dades sense utilitzar les funcionalitats que proporciona Hibernate a través de les interfícies de la Java Persistence API. A diferència dels anteriors repositoris, s'utilitza Hibernate d'una forma més manual, on cada mètode utilitza un objecte anomenat *EntityManager* que permet crear instàncies d'objectes de tipus *Query*, que representen sentències en llenguatge SQL que un cop executades permeten interactuar amb la base de dades realitzant modificacions o consultes, i rebre'n el resultat.

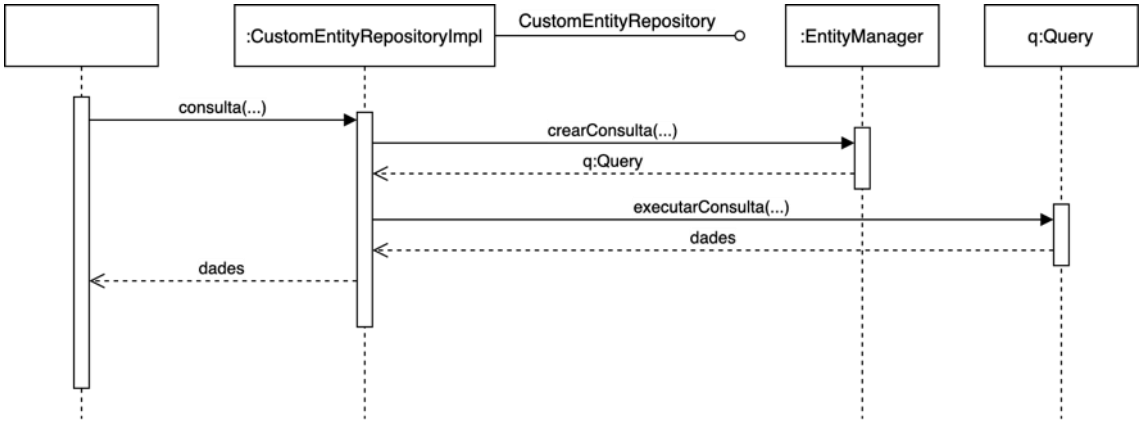

Figura 18. Diagrama de seqüència d'una operació d'un repositori implementat manualment

Elements transversals

A més de les capes analitzades anteriorment, el *qr-dashboard* també inclou paquets que s'encarreguen d'encapsular elements transversals que no estan associats a cap capa concreta.

El primer d'ells és el paquet *Config*. Conté classes que s'encarreguen de configurar elements com ara la connexió a la base de dades i a algunes de les dependències externes, l'autenticació i autorització d'usuaris i elements bàsics de l'aplicació per tal que Spring pugui realitzar totes les seves funcionalitats correctament.

El següent paquet d'elements transversals s'anomena DTO. Aquest és l'acrònim de *Data Transfer Object*, i tal com el seu nom indica són objectes que s'encarreguen de transportar dades. En concret, dins del *qr-dashboard* aquests DTOs es fan servir com a entitats de retorn en algunes de les funcions corresponents als repositoris de tipus *CustomRepository*, tot i que el seu ús principal és com a entitats de retorn dels mètodes definits als serveis. D'aquesta manera es controla quines dades s'envien quan es rep una determinada petició REST a través dels controladors, i Spring s'encarregarà de transformar el DTO de retorn en un JSON respectant l'estructura i les propietats corresponents.

Per últim, es troba el paquet *Exceptions*. Aquest paquet està destinat a agrupar les excepcions que l'aplicació pot generar, segons el tipus d'error que es produeixi. Tot i això, actualment només conté una única excepció definida i utilitzada en poques ocasions. Per norma general, el *qr-dashboard* treballa amb les excepcions bàsiques de Java.

5.3.3. Metodologia

Un cop analitzades les tecnologies i l'arquitectura del *dashboard*, cal veure quina és la metodologia que s'utilitza en el seu desenvolupament. L'eina principal utilitzada pels desenvolupadors del *dashboard* és GitHub, un servei web que permet l'allotjament en línia de repositoris del sistema de control de versions Git, a més d'oferir diferents funcionalitats de gestió. Cada component funcional descrit a l'arquitectura general del *dashboard* consta d'un repositori propi.

Per tal d'afegir noves funcionalitats al *qr-dashboard*, la metodologia de treball és la següent: amb l'objectiu d'organitzar la feina i afegir valor al producte d'una forma àgil, es plantegen fites on caldrà publicar una nova versió del *dashboard*. Al començament d'una fita es decideix quina feina caldrà realitzar a partir de la informació present en forma d'*issues* al repositori de GitHub, que representen funcionalitats noves a implementar, millores de funcionalitats ja existents i correcció d'errors. Dins de cada fita, es realitza un reunió setmanal on es comenta quin ha estat el progrés durant la setmana anterior, i en funció d'això es concreta quina feina caldrà realitzar per la setmana següent amb l'objectiu de prioritzar i aconseguir finalitzar el màxim nombre tasques possible.

Per gestionar el repositori *qr-dashboard*, se segueix una adaptació lleugerament simplificada de la metodologia anomenada *GitFlow* (Figura 19), que organitza el contingut del repositori en branques. La branca principal s'anomena *master*, i és on es troben les funcionalitats ja acabades que proporcionen valor al producte. Per tal d'anar afegint funcionalitats noves i correccions dins d'una fita determinada s'utilitza la branca *develop*, que conté elements que encara no formen part de la versió actual del producte.

Per tal de dur a terme la feina que requereix cadascuna de les funcionalitats noves, es crea una branca nova a partir de *develop* per cadascuna de les tasques que es vulgui realitzar, seguint la terminologia *feature/* seguit pel nom de la tasca en qüestió. Un cop aquestes branques contenen els canvis necessaris per considerar la tasca com a acabada, cal integrar-la amb la branca *develop*.

Per tal de realitzar una correcció d'errors se segueix un enfocament similar, tot i que la branca d'origen serà *master* o *develop* en funció d'on es trobi l'error, i la terminologia utilitzada serà *fix/* seguit pel nom de la tasca. Un cop l'error s'hagi corregit caldrà integrar aquesta nova branca amb *develop* en cas que l'error estigui localitzat en aquesta branca, o amb *master* i *develop* en cas que s'hagi localitzat a la branca *master*.

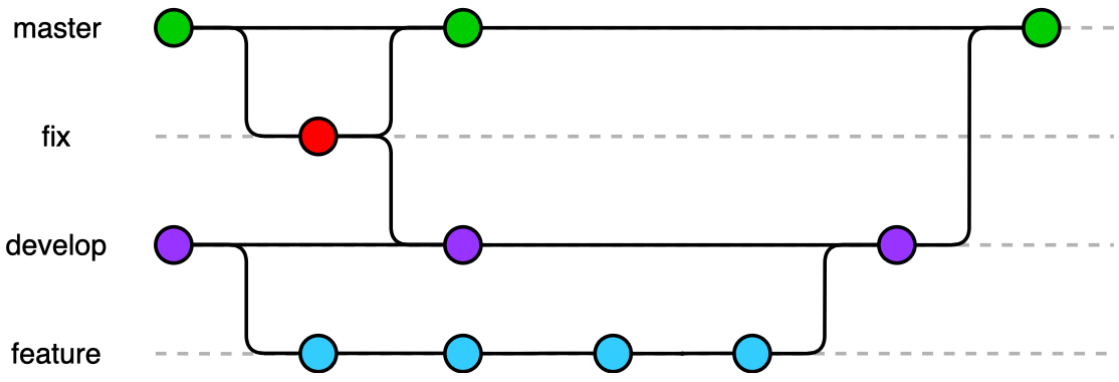

Figura 19. Gitflow

Cal destacar també que per tal de considerar una funcionalitat com a acabada no es requereix la realització de cap prova automatitzada. El *qr-dashboard* no compta amb cap prova definida en el seu codi.

5.3.4. Requisits de qualitat

Per tal de guiar el disseny i el desenvolupament del *dashboard* estratègic, existeixen requisits de qualitat que, a diferència dels requisits funcionals que indiquen què ha de

fer el sistema, descriuen com ha de ser. Els requisits de qualitat presents al sistema són els següents [23]:

- Usabilitat. El dashboard ha de poder ser utilitzat i configurat amb facilitat, ha de respectar aspectes d'accessibilitat, ha de permetre una navegació fluida i clara i ha de descriure amb claredat els elements que es visualitzen.

- Compatibilitat. Ha de ser possible integrar el *dashboard* amb eines externes i proporcionar interoperabilitat amb sistemes existents.

- Confiabilitat. El *dashboard* ha d'informar sobre la completesa de les dades i la qualitat de les avaluacions, a més de mostrar informació sobre els errors que es puguin produir.

- Seguretat. El *dashboard* ha de permetre un accés autenticat d'usuaris, ha d'encriptar les dades confidencials i ha d'incorporar una comunicació segura entre els components.

- Mantenibilitat. El codi que conforma els components interns del dashboard ha de mantenir un nivell elevat de reusabilitat i un disseny que faciliti el seu manteniment per part de l'equip de desenvolupament.

- Portabilitat. Els components que formen el *dashboard* han de poder ser configurats amb facilitat en el moment de la posada en marxa del sistema i durant el seu ús i manteniment.

- Documentació: El *dashboard* ha de comptar amb una guia d'usuari descrivint com utilitzar el sistema, un vídeo tutorial descrivint i mostrant les funcionalitats principals i documentació tècnica amb la informació necessària per integrar el *dashboard* amb altres productes.

6. Anàlisi dels problemes

A continuació es descriu l'anàlisi dels problemes detectats al *dashboard*, la solució dels quals conformarà el nucli del procés de reenginyeria. Per tal de realitzar aquest anàlisi, es descriuen escenaris que deriven de la situació actual del *dashboard* i que representen canvis o modificacions apreciats des de diferents punts de vista. A partir de cada escenari s'argumenta quins problemes se'n deriven, es plantegen les alternatives possibles per tal de resoldre'ls (en cas que se'n contemplin diverses) i s'acaba per exposar la solució final que es pretén adoptar. Els problemes detectats s'enumeren en la següent taula (Taula 10):

Problemes identificats
Falta de proves automatitzades
Documentació manual dels serveis REST
Disseny inconsistent dels serveis REST
Múltiples responsabilitats dels controladors REST
Estructura complexa i implementació manual dels repositoris
Deficiències en l'estat general del codi
Manca d'integració en el procés de desenvolupament

Taula 10. Problemes identificats

Per cadascun dels problemes considerats, es comença per fer una síntesi de tot el contingut tractat en l'anàlisi del problema per tal d'establir de forma concisa quin és l'objectiu que es pretén aconseguir. A continuació, es realitza una anàlisi detallat on es descriu el problema amb més profunditat, s'analitzen les característiques de els diferents alternatives considerades i s'exposa el raonament seguit per tal d'arribar a la solució proposada.

6.1. Falta de proves automatitzades

6.1.1. Síntesi

Escenari
En un moment determinat, es decideix modificar el sistema afegint una nova funcionalitat o modificant-ne una de ja existent. Durant aquest procés, el desenvolupador ha d'assegurar que l'increment que afegeix al producte funciona de la manera esperada i, a la vegada, que cap de les funcionalitats desenvolupades anteriorment pateix cap alteració no intencionada.
Problema
La falta de proves automatitzades provoca un augment de l'esforç necessari per tal de descobrir els errors introduïts en la modificació del sistema, ja que cal provar manualment i de forma reiterada tant la funcionalitat nova o modificada com la resta del sistema per tal de detectar fallades en el funcionament.

Alternatives de solució	**Opcions escollides**
Proves unitàries	✓
Proves d'integració	✓
Proves de sistema	✗

Solució
La solució proposada passa per la integració de proves automatitzades dins del sistema, corresponents a la categoria de proves unitàries i d'integració. Les proves unitàries es realitzaran únicament en aquelles funcions i components que continguin lògica crítica pel funcionament de l'eina, i les proves d'integració comprovaran el funcionament dels controladors dels serveis REST i els repositoris que treballen amb la base de dades.

6.1.2. Descripció del problema

Prenent com a punt de partida la situació actual del *dashboard*, es pot apreciar com el requisit de mantenibilitat es veu afectat en aquest escenari. L'esforç dedicat al desenvolupament ha estat centrat en afegir noves funcionalitats i, un cop acabades, a realitzar proves manuals a través de la interfície gràfica del *dashboard* per comprovar que el seu funcionament és correcte. Per tant, a mesura que el sistema va creixent, l'esforç que el desenvolupador ha de dedicar a la comprovació d'errors també augmenta.

Amb aquest enfocament es perd l'oportunitat d'automatitzar el procés de prova i reduir el temps necessari per executar-les de forma repetitiva. També dificulta la descoberta d'errors, ja que les accions que es poden realitzar a través de la interfície gràfica sovint no contemplen tots els casos possibles que la funcionalitat introduïda ha de gestionar (per exemple, fallades en la xarxa o errors en sistemes externs). A més, per tal de detectar errors col·laterals en altres parts del sistema cal provar manualment tot el *dashboard*, cosa que augmenta dràsticament el temps necessari i la dificultat per dur-les a terme.

6.1.3. Anàlisi de les alternatives

Un dels conceptes clau que formen part del procés de desenvolupament de software àgil són les proves automatitzades. Es tracta de la utilització de software addicional que s'encarrega d'executar porcions del sistema que s'està desenvolupant i de comprovar que el seu comportament és l'esperat. Amb això s'aconsegueix facilitar la descoberta d'errors introduïts durant el desenvolupament, tant en les noves funcionalitats que s'estan afegint al sistema com en les funcionalitats ja existents.

Per tal d'introduir proves automatitzades al *dashboard* estratègic, cal considerar diferents estratègies i enfocaments aplicables per part dels desenvolupadors del sistema, així com les tecnologies necessàries per dur-los a terme.

Proves unitàries

El primer tipus de proves automatitzades que es poden introduir al *dashboard* són les proves unitàries. La paraula "unitàries" fa referència al seu objectiu, que és comprovar el funcionament de les unitats més petites que admeten proves dins d'un sistema. En un entorn orientat a objectes, aquestes unitats acostumen a ser els mètodes d'un objecte.

Cadascuna de les proves unitàries d'un sistema ha de ser independent de les altres, i ha d'assegurar que únicament està comprovant el funcionament de la unitat que s'encarrega de provar. Per tal d'aconseguir-ho, cal substituir els elements amb els quals interacciona la unitat que s'està provant per altres que es comportin d'una manera controlada per part del desenvolupador, de manera que, en cas que es produeixi un error en la prova unitària, es garanteixi que el problema es troba localitzat a la unitat que s'està provant i no als elements amb els quals aquesta interactua. El terme genèric per aquest tipus d'objectes que substitueixen a altres en un entorn de proves és *double*.

Per tal de dur a terme proves unitàries, l'eina més utilitzada en l'entorn Java és JUnit [37]. Es tracta d'un *framework* que permet l'execució controlada de classes per poder avaluar si cada un dels mètodes que conté es comporta de la manera esperada. Es pot integrar fàcilment amb Spring gràcies a un dels paquets de dependències o *starters* que proporciona Spring Boot (*spring-boot-starter-test*), i la major part dels entorns de desenvolupament permeten l'execució de proves escrites amb JUnit d'una manera simple i visual.

Per tal d'aconseguir aïllar correctament les unitats que s'estan provant en cada cas i substituir les seves dependències per altres objectes, existeix una eina àmpliament utilitzada anomenada Mockito [38]. D'entre els diferents tipus de *doubles* que existeixen, Mockito se centra en els *mocks*. Un *mock* és una implementació trivial d'una interfície o una classe en la qual el programador defineix quina serà el resultat corresponent a la crida de certs mètodes. El comportament dels *mocks* es configura en temps d'execució pels mateixos tests, i registren quines interaccions han tingut amb la resta del sistema per tal de ser validades posteriorment. Mockito es troba integrat en el mateix *starter* de Spring Boot que JUnit, i està dissenyat per ser utilitzat juntament amb aquest *framework* de proves de manera simple i senzilla.

Proves d'integració

Si pugem un nivell respecte les proves unitàries, trobem les proves d'integració. Es tracta de processos que comproven que diversos elements individuals funcionen de manera correcta quan interactuen uns amb els altres.

Per tal de realitzar proves d'integració, cal destacar que la granularitat d'aquestes proves pot variar en funció de la quantitat i el tipus d'elements que formin part del conjunt que s'estigui provant. En cas de treballar amb un nombre reduït d'elements que formen part d'un mateix mòdul funcional, l'enfocament és similar al les proves unitàries i es poden utilitzar les mateixes eines, tot i que el nivell al que es realitza l'aïllament amb altres elements variarà en funció de l'abast de les proves d'integració. En canvi, si el que s'està provant és la integració entre diferents mòduls (per exemple, amb serveis externs) o una certa funcionalitat que requereix infraestructura addicional com ara una base de dades o un servidor, és necessari realitzar aquestes proves en un entorn real o bé disposar d'eines que proporcionin substituts per aquesta infraestructura i permetin controlar-la per tal de realitzar les proves adients.

De la mateixa manera que per les proves unitàries, l'*starter* que proporciona les seves dependències també ofereix eines per realitzar proves d'integració. En concret, per tal de realitzar proves als controladors encarregats d'associar certes URIs amb

funcionalitats de l'aplicació, aquest paquet proporciona una eina encarregada d'executar un servidor web i permetre realitzar crides a les URIs que es volen provar, personalitzar els paràmetres i el contingut de cada crida i recollir-ne el resultat, sense necessitat d'haver de posar en funcionament l'aplicació sencera. A més, per tal d'analitzar si el resultat ha estat l'adequat, una altra de les eines proporcionades permet realitzar comprovacions directament sobre el JSON retornat per un servei REST. Per últim, en cas de realitzar proves que requereixin persistència de dades, també es disposa d'una eina encarregada de posar en funcionament una base de dades a memòria que substitueix la base de dades real, i sobre la qual es podrà controlar el contingut abans i després de les proves.

Proves de sistema

El següent tipus de proves que es poden introduir són les proves de sistema. Es tracta d'una ampliació de les proves d'integració on el conjunt d'elements que es prova esdevé molt més transversal, i en general integra elements que van des de la interfície gràfica fins a la base de dades, executats en un entorn similar al de producció. L'objectiu d'aquestes proves és assegurar que el sistema compleix amb els requisits especificats, tant funcionals com no funcionals.

Spring Boot no proporciona cap eina específica per realitzar proves de sistema en cap dels seus *starters*, però existeixen diverses opcions externes que es poden utilitzar. En el cas de les proves de sistema orientades envers als requisits funcionals, dins de l'entorn web una eina popular és Selenium [39]. Es tracta d'un entorn de proves que permet realitzar accions de manera automàtica sobre la interfície gràfica a través d'un navegador web de la mateixa manera que ho faria un usuari, i comprovar que el contingut de la pàgina generada un cop completades totes les accions és correcte.

Pel que fa a les proves de requisits no funcionals, el procediment per realitzar-les varia en funció de quin sigui el requisit que se sotmet a prova. Si bé hi ha requisits on les eines d'automatització són poc efectives o inexistents, com és el cas del requisit d'usabilitat on en la majoria dels casos es realitzen proves amb usuaris reals i s'extreuen conclusions de forma manual, hi ha altres requisits les proves dels quals sí poden ser automatitzades amb èxit. Per exemple, en el cas del requisit de rendiment trobem l'eina Apache JMeter [40], que permet comprovar de manera automàtica la capacitat de resposta del nostre sistema davant d'una forta càrrega de treball.

6.1.4. Justificació de la solució

Un cop vistos els tres possibles nivells de proves automatitzades que es poden introduir al *dashboard*, cal prendre una decisió sobre quins són els més adients i eficaços per resoldre el problema plantejat.

Per començar, es descarta realitzar proves de sistema. Aquesta decisió bé donada per la falta de la infraestructura necessària que caldria per tal d'executar aquest tipus de proves correctament, i per la quantitat de temps necessari per dur-les a terme. A més, els requisits no funcionals que formen part del *dashboard* no faciliten la seva prova de forma automàtica, i en alguns casos el factor que busquen introduir dins el sistema cal que sigui validat pels usuaris finals. En conclusió, donades les característiques del

projecte, les proves de sistema esdevenen massa costoses d'aplicar i el benefici que se'n pot extreure no justifica el seu cost. Les proves manuals se seguiran realitzant per tal de comprovar que les funcionalitats desenvolupades funcionin correctament.

Per altra banda, les proves unitàries i d'integració sí que es consideren adients pel *dashboard*. La seva mida més reduïda i la facilitat per executar-les fa que es puguin integrar de manera senzilla amb els processos de desenvolupament actuals. Tot i això, cal considerar de quina manera i amb quins components cal realitzar aquests tipus de proves.

Per una banda, per introduir proves unitàries cal mesurar bé on és realment necessari aplicar-les. Actualment, és comú trobar-se amb projectes que fan gala del seu alt nombre de proves en relació a les línies de codi que aquestes cobreixen (concepte anomenat *coverage*), i això es relaciona directament amb una alta qualitat del sistema. Si bé és cert que un *coverage* elevat és una mesura generalment positiva, no s'ha de prendre com quelcom inequívocament desitjable en un sistema.

En particular, les proves unitàries serveixen per provar la lògica dels components d'una aplicació, per tal d'assegurar que els algorismes plasmats en aquesta lògica es comporten de la manera desitjada. També cal considerar que les proves unitàries, com la resta del codi, requereixen d'un manteniment a mesura que el sistema evoluciona. Per tant, si els components pels quals s'introdueixen proves unitàries manquen de lògica i tenen un comportament trivial, s'estarà introduint codi dins del sistema que, a part de ser poc útil per trobar errors, afegirà un cost de manteniment que no serà compensat per la utilitat de les proves.

Un cop considerat això, cal ser conscient del paper del dashboard dins de l'eina Q-Rapids. Tal i com s'ha vist en apartats anteriors, l'objectiu principal d'aquest component és presentar les dades a l'usuari i permetre realitzar les accions que desitgi. Per la gran majoria de les seves funcionalitats, el *dashboard* actua com a pont entre components que executen càlculs o emmagatzemen dades i l'usuari final. En aquests casos, on la lògica acostuma a ser trivial, les proves unitàries no són necessàries. Tot i això, hi ha funcionalitats del dashboard que sí contenen algorismes suficientment complexos i rellevants pel sistema com per disposar de proves unitàries. És el cas del càlcul dels indicadors estratègics, la responsabilitat del qual recau directament al *dashboard*. En aquest cas sí que s'inclouran proves unitàries per determinar que el funcionament és correcte i prevenir futurs errors.

Per altra banda, per les proves d'integració cal seguir un enfocament similar. Cal trobar els punts on s'integrin components amb rellevància pel sistema i on els errors es puguin donar amb més facilitat. Per fer-ho, caldrà actuar en dues zones diferenciades del *dashboard*.

En primer lloc, els controladors dels serveis REST són components encarregats no només d'executar una funcionalitat del domini i obtenir el resultat, sinó que depenen completament de les funcionalitats de servidor per obtenir una determinada ruta, els seus paràmetres i el cos que envia el client i retornar un determinat resultat amb el codi HTTP corresponent. A més, també hi juga un paper el format de la resposta, el JSON que s'envia ha de tenir l'estructura adequada. Amb tot això, es poden concebre proves

d'integració entre les funcionalitats de servidor, els controladors REST i el mecanisme per transformar objectes Java en objectes JSON. Degut a la importància que les crides REST tenen en el sistema, aquestes proves esdevenen rellevants per assegurar el seu bon funcionament.

En segon lloc, per tal de provar els components interns que interactuen amb la base de dades també caldrà realitzar proves d'integració, substituint la base de dades real per una altra gràcies a les eines que proporciona Spring per tal de poder comprovar que les funcions que realitzen els repositoris són correctes.

6.2. Documentació manual dels serveis REST

6.2.1. Síntesi

Escenari	
Es produeix un canvi en els serveis REST que el *dashboard* proporciona, ja sigui afegint un nou servei o modificant la URL, els paràmetres necessaris o el format de la resposta d'un servei existent. El desenvolupador ha d'actualitzar la documentació per tal de plasmar aquest canvi i transmetre correctament la nova informació.	
Problema	
La documentació dels serveis REST es troba en un document de text extern al sistema, l'actualització del qual es fa de forma manual i amb una freqüència no continuada que no segueix el ritme dels canvis que experimenta el sistema. Això augmenta la dificultat de mantenir aquesta documentació i produeix situacions on el seu contingut pot no representar la realitat del sistema, afectant a la seva qualitat.	
Alternatives de solució	**Opcions escollides**
SpringFox	✗
Spring REST Docs	✓
Solució	
Per donar solució a aquest problema, es proposa fer servir Spring REST Docs per tal de donar suport al procés de documentació dels serveis REST. Aquesta eina aprofita les proves d'integració dels serveis REST per tal de generar documentació de manera automàtica a partir de les crides, paràmetres i respostes dels serveis que es produeixen durant les proves. Amb aquest enfocament s'aconsegueix que el contingut de la documentació sempre concordi amb la realitat del sistema, ja que en cas que no sigui així les proves d'integració fallaran i la documentació no es generarà.	

6.2.2. Descripció del problema

Tal i com es pot observar, el requisits de qualitat involucrats en aquest escenari són la documentació i la mantenibilitat. Si bé no es fa referència al manteniment del codi del sistema, el fet de mantenir la documentació també té una rellevància clau en el procés de desenvolupament de software. Quan un sistema disposa de documentació, s'adquireix un contracte entre l'usuari extern i els desenvolupadors mitjançant el qual la informació que hi figura ha d'esdevenir un reflex fidel del sistema que documenta. Un usuari que vulgui utilitzar l'aplicació com un sistema extern utilitzarà aquesta documentació i no posarà en dubte cap dels elements que hi apareixen, de manera que,

en cas de no estar actualitzada o no representar exactament la realitat del sistema, farà que l'usuari no el pugui utilitzar correctament i dificultarà la localització dels errors que s'estan produint.

En l'estat actual del *dashboard*, la documentació dels serveis REST es duu a terme en els documents d'ús intern que es realitzen en determinades fites marcades pel projecte de recerca, on s'exposa l'estat actual de l'eina Q-Rapids. Dins del document que recull tot allò relatiu a les funcionalitats del *dashboard* estratègic, es detallen alguns dels serveis REST principals que ofereix. En concret, per cada servei es defineix el seu títol, la URL necessària per fer la crida, el mètode HTTP, els paràmetres i el cos de la crida en cas que siguin necessaris i els codis HTTP que retorna cada servei, acompanyats per la seva resposta en format JSON. Tots aquests atributs es defineixen de forma manual, i en cas que el sistema canviï també cal actualitzar manualment els camps necessaris. També cal destacar que el document en el qual està inclosa la documentació s'actualitza amb una freqüència relativament llarga de temps, de manera que els canvis que pateix el sistema no hi queden reflectits fins passats diversos mesos.

6.2.3. Anàlisi de les alternatives

Per tal de millorar la situació actual i que l'escenari que es planteja no suposi un problema per la qualitat de la documentació generada ni per la mantenibilitat del *dashboard*, cal utilitzar alguna eina que doni suport al procés de documentació de serveis REST i que faci més senzilla la seva actualització. Les opcions que es consideren inicialment són les següents:

Springfox

Springfox [41] és una llibreria que s'encarrega d'automatitzar el procés de generació de documentació d'APIs JSON de sistemes basats en Spring, obtenint com a resultat una especificació que és accessible i comprensible tant per usuaris com per màquines. Aquesta llibreria funciona examinant l'aplicació en temps d'execució per tal d'inferir la semàntica de la API a partir de la configuració de Spring, l'estructura de les classes i diverses anotacions de compilació de Java.

Es tracta d'un component que no forma part del Spring Framework, sinó que està desenvolupat i mantingut per desenvolupadors externs. Per tal de documentar les APIs, Springfox suporta els formats de documentació i especificació més utilitzats enfocats a APIs JSON, com són Swagger, RAML i JSON:API. Un cop generada la documentació, es podrà visualitzar en forma de pàgina web mitjançant les eines que proporciona el format escollit. A més, aquests formats permeten fer servir aquesta pàgina de documentació per realitzar proves reals amb la API, en cas que disposem del servei desplegat en un servidor.

Per tal d'utilitzar Springfox, cal afegir anotacions en diferents nivells del nostre sistema, que contindran la informació necessària per construir la documentació: les classes que fan de controladors dels serveis REST s'anoten amb una descripció del servei que implementen. A continuació, cadascun dels mètodes pertanyents a aquests controladors contenen anotacions indicant la informació necessària per cada servei (descripció, paràmetres, codis HTTP d'estat i missatges de les diferents respostes). Dins

de la signatura de cada mètode, cal anotar els paràmetres de la crida amb una descripció i un valor per defecte. Les descripcions dels objectes fets servir per recollir el cos de les crides HTTP i utilitzats també per construir els objectes JSON de retorn caldrà fer-les directament a les classes que defineixen aquests objectes, on cada propietat contindrà una anotació amb la seva descripció. Cal destacar també que, com que es tracta d'una llibreria externa, caldrà introduir classes especials que s'encarregaran de configurar Springfox i els seus components perifèrics, que variarà en funció del format de la documentació escollit.

Spring REST Docs

Spring REST Docs [42] és una eina que proporciona suport al procés de documentació de serveis REST per mitjà de la combinació de documentació escrita a mà i fragments generats automàticament produïts per les proves d'integració dels propis serveis. Es tracta d'un enfocament on es trasllada part de la responsabilitat del disseny i maquetació de la documentació als desenvolupadors, però el fet de basar el seu contingut en el resultat de les proves d'integració fa que aquest esdevingui molt més fiable.

Per tal d'utilitzar Spring REST Docs cal partir d'un requisit important, ja que per tal de documentar un servei REST cal que aquest consti de proves d'integració realitzades amb les eines que proporciona Spring per aquesta tasca. El codi necessari per la documentació estarà exclusivament en les classes destinades a realitzar aquestes proves, de maner que no cal modificar ni introduir anotacions en el codi que constitueix la part funcional del sistema. A més, com que es tracta d'una eina que forma part del Spring framework, la seva integració és senzilla i no requereix de classes addicionals.

Dins de cada prova d'integració, cal afegir una tasca que s'encarrega de generar els fragments que posteriorment formaran part de la documentació. En concret, es tracta d'examinar la petició feta en la prova i la resposta rebuda, indicant manualment quina és la URL, els paràmetres i el cos de la petició i el contingut del JSON de la resposta, acompanyats per una descripció de la funció de cadascun. Un cop fet això, quan s'executin les proves d'integració Spring REST Docs generarà diversos fragments amb diferent contingut (la petició i la resposta HTTP completa, el cos de la petició i de la resposta i els paràmetres amb la seva descripció de la petició i de la resposta) per cadascuna i, en cas que la petició o la resposta executada en la prova no coincideixi amb allò especificat, aquesta fallarà i indicarà que hi ha un error. D'aquesta manera el contingut de la documentació mai pot quedar obsolet, ja que per generar nova documentació cal executar les proves i que aquestes finalitzin correctament.

Per últim, per tal de seleccionar els fragments que cal mostrar en cada cas i fer una maquetació de la documentació, cal crear manualment un document addicional que farà la funció de plantilla. Aquest document té una estructura lliure, però generalment s'indicaran per una banda aspectes transversals com el títol, una descripció general i la data i versió de la documentació, i per altra banda per cada operació s'acostuma a indicar el seu nom, una descripció i quins són els fragments generats que cal incloure en cada cas. El format d'aquest document i dels fragments que genera Spring REST Docs és AsciiDoc, un format de text molt configurable que permet ser convertit a un gran nombre de formats addicionals, entre ells HTML. Un cop la documentació hagi estat

generada, l'usuari podrà veure en forma de pàgina web tota la informació dels serveis documentats, amb la particularitat que aquesta web no permet realitzar proves reals amb el servei en cas que aquest estigui desplegat.

6.2.4. Justificació de la solució

Tal i com s'ha exposat, les dues eines que es consideren per tal de donar suport al procés de documentació dels serveis REST del *dashboard* tenen el mateix propòsit general, però segueixen enfocaments molt diferents. L'eina que s'utilitzarà per documentar els serveis REST del *dashboard* és Spring REST Docs, i la seva elecció es veu motivada per diversos factors que fan referència tant a les característiques de la pròpia eina com del *dashboard*.

Per començar, cal esmentar que per prendre aquesta decisió s'ha considerat el problema tractat en l'apartat anterior relatiu a la introducció de proves automatitzades, i on s'ha considerat convenient afegir proves d'integració pels serveis REST del dashboard. Aquest és precisament el requisit necessari per fer servir Spring REST Docs, i per tant es poden aprofitar aquestes proves per produir la documentació. A més, tot i que amb Springfox la informació que es farà servir per generar la documentació es troba localitzada juntament amb el codi i això facilita el seu manteniment, el desenvolupador no disposa de cap mètode per assegurar que el contingut sigui l'adequat, i això implica que cal llegir i comprovar manualment la veracitat de la informació. En canvi, amb Spring REST Docs aquesta comprovació es farà de manera automàtica, i seran les proves d'integració les que informaran al desenvolupador sobre quan cal actualitzar el contingut de la documentació.

En segon lloc, el fet que Spring REST Docs estigui integrada dins del Spring *framework* fa que el procés de configuració sigui molt més senzill que amb Springfox, i que no siguin necessàries classes ni codi addicional per tal de posar en funcionament l'eina. Per últim, també cal considerar que el *dashboard* estratègic és un sistema on cada organització que l'utilitzi haurà de desplegar la seva pròpia instància, cosa que fa que els formats de documentació on es permeten realitzar proves des de la seva pàgina web esdevinguin irrellevants degut a que no es disposa de cap instància general amb la qual vincular la documentació generada.

6.3. Disseny inconsistent dels serveis REST

6.3.1. Síntesi

Escenaris
Es vol introduir una nova funcionalitat al dashboard que quedarà exposada a través d'un servei REST. Per fer-ho, el desenvolupador haurà de triar una forma d'accedir-hi que segueixi els principis dels serveis REST i que a més s'adeqüi a l'estil de la resta de serveis de l'aplicació, per tal de mantenir la seva homogeneïtat.
Per tal d'integrar i aprofitar les funcionalitats del *dashboard* estratègic en un sistema extern, el desenvolupador d'aquest sistema farà servir els serveis REST que el dashboard proporciona. Per poder aconseguir una integració satisfactòria i robusta,

caldrà que aquest pugui identificar els diferents tipus de resposta a les seves peticions per mitjà dels codis HTTP d'estat per poder actuar en conseqüència.
Problema
El disseny actual dels serveis REST exposa els recursos d'una manera no consistent i amb mètodes HTTP que no corresponen a les accions que es realitzen. Això provoca una dificultat afegida a l'hora de mantenir aquests serveis, ja que es va en contra de la intuïció dels desenvolupadors. Per altra banda, la falta de detall en les respostes d'error dels serveis REST dificulta la compatibilitat del *dashboard* amb sistemes externs, ja que els codis HTTP s'utilitzen sense cap significat intencionat ni s'acompanyen de missatges informatius.

Principis i bones pràctiques	**Opcions escollides**
Patró *Resource API*	✓
URIs basades en recursos	✓
Mètodes HTTP per transmetre intencionalitat	✓
Codis HTTP informatius acompanyats de missatge	✓

Solució
La solució proposada passa per aplicar tots els principis i bones pràctiques pel disseny de serveis REST esmentats. Caldrà identificar i anomenar correctament els recursos i definir els mètodes necessaris per les operacions dels serveis, i assignar codis HTTP a les diferents respostes així com establir missatges adequats pels casos d'error.

6.3.2. Descripció del problema

En el cas del primer escenari, el requisit de qualitat que queda afectat és la mantenibilitat. Si s'analitza l'estat actual dels serveis del dashboard es pot observar com aquests, tot i estar definits com a serveis REST, no segueixen en tots els casos els principis que caracteritzen aquests tipus de serveis. Els objectes que s'exposen a través de recursos no ho fan de manera consistent, i sovint s'utilitzen noms d'accions o mètodes enlloc dels noms dels recursos. A més, els mètodes HTTP no s'utilitzen correctament, ja que sovint no es respecta el seu propòsit i no es fan servir tots quan cal. Amb aquesta situació, quan un desenvolupador ha d'afegir un nou servei es troba amb la disjuntiva de si ha d'utilitzar els principis REST de la manera correcta o si ha de seguir l'estil que marquen els altres serveis, però ambdues opcions no són compatibles.

Per altra banda, el segon escenari fa referència al requisit de qualitat de compatibilitat. Si bé és cert que les funcionalitats són accessibles des d'un altre sistema existent, en l'estat actual les respostes que proporciona el *dashboard* a la majoria de les crides no contenen els detalls suficients, especialment en els casos d'error. No és possible diferenciar entre errors produïts per les peticions del clients o per una fallada interna del servidor, i no es proporcionen missatges que indiquin detalls sobre quina és la causa del problema. Tot això contribueix a disminuir la qualitat de la integració amb sistemes existents, dificultant l'execució de totes les funcionalitats.

6.3.3. Principis i bones pràctiques

Es conclou doncs que cal modificar el disseny dels serveis REST per tal d'adequar la forma d'accedir-hi a les bones pràctiques i principis propis d'aquests serveis, i que cal

modificar les respostes dels mateixos per tal que utilitzin correctament els codis HTTP d'estat i siguin el màxim d'informatius possible.

Quan es parla de REST (REpresentational State Transfer), es fa referència a una arquitectura de tipus client-servidor sense estat, en la qual els serveis web s'exposen com a recursos que poden ser identificats per les seves respectives URIs. En aquest apartat, s'exposaran quins són els principis i bones pràctiques que cal seguir en el disseny de serveis REST.

Per tal de dissenyar un servei web que segueixi l'arquitectura REST, el primer pas és crear una API que permeti accedir a aquest servei. El patró que s'utilitza per aquest tipus de serveis és l'anomenat *Resource* API. Els serveis que segueixen aquest patró utilitzen la URI i el mètode HTTP de la petició que realitza el client juntament amb el tipus de dades (*Media Type*) per tal de determinar quin és el propòsit del client.

El patró *Resource* API proporciona accés a recursos a través de les seves representacions, que són accessibles mitjançant els mètodes HTTP bàsics (GET, POST, PUT, DELETE). Un recurs es defineix com quelcom suficientment important per ser referenciat com una entitat per si mateix, i pot ser un arxiu de text, un arxiu multimèdia, una fila específica d'una base de dades, una col·lecció de dades, un procés de negoci, etc. Les representacions d'aquests recursos capturen el seu estat actual, i permeten als clients manipular-lo mitjançant diferents formats (XHTML, JSON, XML, etc.).

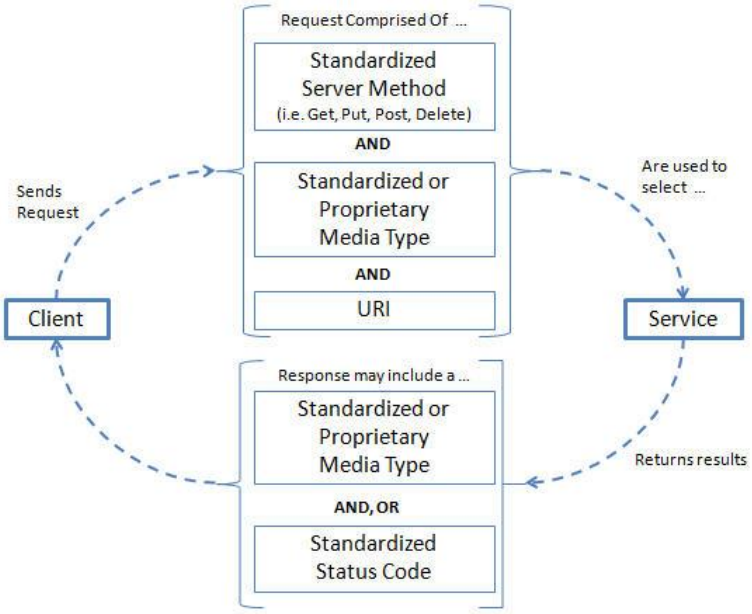

Figura 20. Esquema del patró Resource API [43]

Per tal d'identificar els recursos en un servei REST, usualment es treballa amb dos nivells d'abstracció:

- **Col·lecció**: recurs que representa una classe o conjunt d'elements.
- **Entrada**: recurs que representa una instància o individu dins d'una determinada classe o conjunt.

Per tal d'anomenar les URIs que defineixen els recursos del servei, es defineixen dues URIs base que permeten fer referència a una col·lecció i a una entrada per separat. Per

la col·lecció es fa servir el nom del conjunt d'elements en plural, i per l'entrada es parteix de la col·lecció a la que pertany i s'afegeix l'identificador de la instància.

- Col·lecció: /**stocks**
- Entrada: /stocks/**IBM**

Per tal d'indicar atributs especials, s'afegeixen paràmetres de consulta addicionals a les URIs que permeten filtrar segons les característiques i propietats dels recursos als que es fa referència, o bé introduir certes restriccions en la resposta que es vol obtenir i que permeten introduir conceptes com ara la paginació, on els elements d'un conjunt es retornen en porcions enlloc de tots de cop.

- /dogs**?color=red&state=running&location=park**
- /stocks**?limit=25&offset=50**

Pel que fa als casos on cal permetre l'accés a un servei que no representa un recurs sinó una determinada funcionalitat, la URI base ha d'estar formada per verbs i utilitzar els paràmetres de consulta per afegir informació addicional.

- **/convert?from=EUR&to=CNY&amount=100**

Juntament amb les URIs, tal i com s'exposa en el diagrama anterior, el que defineix quina és la funcionalitat que el client vol executar és el mètode HTTP utilitzat en la petició. Cada mètode HTTP bàsic té un significat en funció de sobre quin dels dos nivells d'abstracció s'apliqui, i no totes les combinacions estan permeses. En la següent taula es descriu quin és aquest significat i quins casos no estan admesos.

Recurs	**GET**	**PUT**	**POST**	**DELETE**
Col·lecció	Obtenir una llista de totes les entrades que pertanyen a una col·lecció.		Crear una nova entrada a la col·lecció. L'identificador es genera automàticament i es retorna l'entrada sencera com a resposta de l'operació.	
Entrada	Obtenir la representació d'una determinada entrada que forma part de la col·lecció.	Actualitzar una entrada que forma part de la col·lecció o, si no existeix, crear-la.		Esborrar una entrada que forma part de la col·lecció.

Taula 11. Mètodes HTTP

El mètode GET es considera un mètode segur, i això vol dir que el seu objectiu és únicament obtenir informació i la seva utilització no suposa un canvi en l'estat del servidor, cosa que implica que no ha de tenir cap mena d'efecte col·lateral. Els mètodes PUT i DELETE es defineixen com a idempotents, en el sentit de que diverses peticions idèntiques han de tenir el mateix efecte que una única petició. El mètode GET, a part de ser segur, també ha de ser idempotent, ja que HTTP és un protocol sense estat. Per

últim, el mètode POST no ha de ser necessàriament idempotent, i per tant l'enviament de múltiples peticions idèntiques amb aquest mètode tindrà un efecte més ampli en l'estat del servidor.

Un cop vistes les bones pràctiques i principis REST pel que fa a les peticions, cal veure quines són les pautes que es marquen pel que fa a les respostes que retornen els serveis. Aquestes respostes consten d'un codi HTTP que indica quin és l'estat de la resposta que el servidor retorna a partir d'una petició del client i, opcionalment, capçaleres HTTP que poden indicar, entre altres coses, el tipus de contingut que la petició retorna, i la representació del recurs involucrat en la petició. És important utilitzar correctament els codis HTTP d'estat, ja que a partir d'ells es pot obtenir molta informació de què està passant al servidor i el client pot actuar en conseqüència. En cas d'error, és convenient enviar no només el codi HTTP corresponent sinó també un missatge que descrigui quin ha estat el motiu de l'error. Els codis HTTP d'estat que defineix l'estàndard HTTP/1.1 (RFC 7231) s'enumeren a continuació, juntament amb la descripció de les 5 categories en les quals es classifiquen:

- **Informació** (1xx): La petició ha estat rebuda, i es continua el seu processament.
- **Èxit** (2xx): La petició ha estat rebuda, entesa i acceptada correctament.
- **Redirecció** (3xx): Es requereixen accions addicionals per part del client per tal de completar la petició.
- **Error del client** (4xx): La petició conté una sintaxi errònia o no pot ser satisfeta.
- **Error del servidor** (5xx): El servidor ha fallat en completar una petició aparentment vàlida.

Informació	Èxit	Redirecció	Error del client	Error del servidor
100 Continue	**200** OK	**300** Multiple Choices	**400** Bad Request	**500** Internal Server Error
101 Switching protocols	**201** Created	**301** Moved Permanently	**401** Unauthorized	**501** Not Implemented
	202 Accepted	**302** Found	**402** Payment Required	**502** Bad Gateway
	203 Non-Authoritative Information	**303** See Other	**403** Forbidden	**503** Service Unavailable
	204 No Content	**304** Not Modified	**404** Not Found	**504** Gateway Timeout
	205 Reset Content	**305** Use Proxy	**405** Method Not Allowed	**505** HTTP Version Not Supported
	206 Partial Content	**307** Temporary Redirect	**406** Not Acceptable	
			407 Proxy Authentication Required	
			408 Request Timeout	
			409 Conflict	
			410 Gone	
			411 Length Required	

			412 Precondition Failed	
			413 Payload Too Large	
			414 URI Too Long	
			415 Usupported Media Type	
			416 Range Not Satisfiable	
			417 Expectation Failed	
			426 Upgrade Required	

Taula 12. Codis d'estat HTTP

6.3.4. Justificació de la solució

Un cop vistes totes les bones pràctiques i els principis necessaris, cal veure com s'aplicaran al *dashboard* estratègic. Per aquest problema, a diferència dels anteriors, no es compta amb diferents tècniques ni alternatives ja que els conceptes exposats tenen una gran acceptació i són àmpliament utilitzats actualment. Així doncs, la solució que es proposa és fer servir tots i cadascun d'aquests conceptes per redissenyar els serveis REST del *dashboard*.

Per dur-ho a terme, caldrà partir dels serveis disponibles actualment per tal de saber quines funcionalitats exposen. A continuació, caldrà identificar i anomenar correctament i de forma homogènia els recursos que conformen el nucli dels serveis REST, tant a nivell de col·leccions com d'entrades, juntament amb els paràmetres necessaris per cadascun d'ells. En funció de les operacions necessàries sobre cada recurs, caldrà escollir el mètode adequat per aquest propòsit a partir de la informació present a la Taula 11. Per últim, caldrà veure quins són els diferents tipus de resposta que cada petició pot retornar, i assignar un dels codis d'estat HTTP presents a la Taula 12 en funció de a quina categoria pertanyi la resposta i quina és la informació que vol expressar.

6.4. Múltiples responsabilitats dels controladors REST

6.4.1. Síntesi

Escenari
Es vol modificar un dels aspectes del nucli funcional del dashboard, canviant el comportament de la lògica del sistema de manera que les accions que involucren entitats del domini quedin alterades. Aquesta lògica pot formar part de diverses funcionalitats, i cal mantenir el mateix comportament per cadascuna d'elles.
Problema
En l'estat actual del *dashboard*, la lògica del sistema es troba dins dels serveis. Aquestes classes tenen múltiples responsabilitats, començant per la d'exercir com a controladors dels serveis REST, obtenir les dades de les fonts corresponents i executar la lògica del domini. Això provoca que la reusabilitat del sistema es vegi afectada, donant lloc a repeticions del codi que són complicades de mantenir.

Solució
Es crearan controladors del domini que s'encarregaran d'encapsular la lògica del sistema, treballant amb les entitats del domini i obtenint les dades necessàries mitjançant els repositoris i adaptadors. Els controladors dels serveis REST es traslladaran juntament amb els DTOs des de la capa de domini on es troben ara fins la capa de presentació, ja que la seva responsabilitat ha de correspondre únicament a fer d'interfície dels serveis REST delegant la funcionalitat cap a la capa de domini.

6.4.2. Descripció del problema

Aquest escenari està relacionat amb el requisit de mantenibilitat. Partint de l'estat actual, es pot observar com els components encarregat d'implementar la lògica del sistema són els serveis. Aquestes classes no només actuen com a controladors dels serveis REST, sinó que també s'encarreguen d'obtenir les dades de la font corresponent i realitzen els càlculs i modificacions de les dades necessàries per tal de dur a terme cada funcionalitat del domini correctament. Per obtenir i modificar dades, els serveis fan servir directament els repositoris per realitzar operacions a la base de dades i els adaptadors per comunicar-se amb els serveis externs.

Un cop vista aquesta situació, és senzill adonar-se que els serveis trenquen amb el *Single Responsibility Principle* (SRP), o principi de responsabilitat única. Aquest és un principi del disseny orientat a objectes que indica que un mòdul, classe o funció ha de ser responsable d'una única part de la funcionalitat que proporciona el sistema, i que aquesta responsabilitat ha d'estar completament encapsulada per la classe. Una manera alternativa d'entendre aquest principi és afirmant que una classe ha de tenir una única raó per ser modificada.

A més, el fet de que la lògica del domini estigui dins de les funcions que implementen els serveis fa que això afecti a la reusabilitat del sistema. En els casos on hi ha operacions del domini que formen part de diverses funcionalitats, es repeteix el mateix codi en diverses funcions donant lloc a una potencial font de problemes i a un manteniment més complex.

6.4.3. Justificació de la solució

Per tal de donar solució a aquest problema, cal realitzar una millor distribució de les responsabilitats que inclouen les classes del *dashboard* que corresponen als serveis. Per una banda, els serveis s'encarreguen d'actuar com a controladors dels serveis REST, establint els recursos disponibles juntament amb les seves URIs i els mètodes HTTP, i proporcionant un accés directe a través de codi Java a les peticions dels clients i permetent la generació de respostes, amb la representació dels recursos i els codis d'estat HTTP. Per altra banda, trobem les funcionalitats del domini, que treballen amb els models i les diferents fonts de dades per tal de realitzar operacions de consulta o modificació.

Seguint aquest raonament, el primer que cal fer és separar aquestes dues responsabilitats. Per tal d'encapsular la lògica del sistema, es crearan classes que faran de controladors de la capa de domini, i que contindran mètodes encarregats d'executar tot allò que representa el nucli funcional del *dashboard*. Treballaran amb les entitats del

domini i obtindran i modificaran les dades necessàries mitjançant els repositoris i els adaptadors, de forma que aquests controladors seran els únics components acoblats amb les fonts de dades. Cada controladors s'encarregarà de les operacions sobre una entitat o entitats molt relacionades entre si, de manera que cadascun d'ells farà referència a un aspecte funcional concret. Com és natural, aquests controladors formaran part de la capa de domini.

Per altra banda, un cop la lògica dels serveis s'hagi extret cap als controladors del domini, aquests serveis passaran a ocupar-se únicament de la gestió dels serveis REST, delegant tota la funcionalitat principal cap als controladors del domini. Un cop arribats a aquesta situació, es pot observar com la localització d'aquests serveis ja no hauria de ser la capa de domini. En aquest punt, la seva funció principal és permetre l'accés a una determinada funcionalitat, iniciar l'execució de la funcionalitat del domini corresponent i retornar un resultat concret. Si extrapolem aquest escenari a un altre entorn on en comptes de serveis REST estiguéssim treballant amb vistes d'una interfície gràfica, podem veure com aquesta situació encaixa amb el patró MVC. La diferència radica en que ara les vistes no estaran destinades a ser interpretades de forma gràfica, sinó què seran representacions en format JSON dels recursos. Els serveis que actuen com a controladors REST accediran a les funcionalitats proporcionades pels controladors del domini, i ompliran les "vistes" per mitjà de la creació dels objectes DTO per tal que aquests siguin convertits a format JSON automàticament per Spring, tal i com es feia fins ara. D'aquesta manera podem justificar que tant els serveis com els DTOs han de traslladar-se de la capa de domini fins la capa de presentació.

6.5. Estructura complexa i implementació manual dels repositoris

6.5.1. Síntesi

Escenari
Dins d'una funcionalitat determinada ja present en el sistema, es vol modificar la forma en la que s'accedeix a la base de dades alternat la consulta o modificació que es vol executar. Per fer-ho, cal modificar les interfícies i les implementacions dels repositoris de les entitats del domini involucrades.
Problema
Actualment, l'estructura dels repositoris requereix de diverses interfícies en funció de si els mètodes són implementats automàticament per Spring o bé la implementació la realitza l'usuari, cosa que fa que sigui necessària una classe d'implementació. Tots aquests elements conviuen dins del paquet de repositoris, i no se segueix cap criteri clar per assignar operacions a una interfície o a una altra. Aquesta situació afecta a la mantenibilitat i provoca confusió a l'hora d'afegir o modificar operacions als repositoris.

Alternatives de solució	**Opcions escollides**
Repository	✗
CrudRepository	✓
PagingAndSortingRepository	✗
JpaRepository	✓

Solució
A partir de l'anàlisi de la jerarquia de les interfícies que proporciona Spring, es proposa reduir a dues les interfícies que s'estendran per tal de crear els repositoris. Per una banda, en els casos on les operacions siguin consultes, creacions o modificacions simples d'una entitat es farà servir CrudRepository per tal que Spring implementi els mètodes a partir de la signatura. Per altra banda, en els casos on calgui fer consultes complexes on intervinguin diverses entitats i sigui necessari aplicar certa lògica caldrà fer servir JpaRepository per tal de poder anotar les signatures amb la operació escrita amb el llenguatge de consulta de JPA o bé en SQL. Amb tot això s'elimina la necessitat de disposar d'una classe d'implementació pels mètodes dels repositoris, i tot el treball el realitza Spring.

6.5.2. Descripció del problema

Tal i com s'ha explicat a l'apartat Arquitectura dins de les seccions que parlen sobre la capa de domini i de dades, el dashboard fa servir repositoris per realitzar operacions a la base de dades. Aquests repositoris estan formats, en primer lloc, per diverses interfícies que defineixen els mètodes que s'utilitzen per accedir i modificar el contingut de la base de dades (Figura 16). D'entre aquestes, dues interfícies de les que s'utilitzen (*JpaRepository* i *PagingAndSortingRepository*) són proporcionades per Spring, mentre que les dues restants (*EntityRepository* i *CustomEntityRepository*) són creades pel desenvolupador.

La interfície *EntityRepository* estén les dues interfícies proporcionades per Spring, i defineix signatures de mètodes amb operacions bàsiques de consulta i modificació. Gràcies al fet d'estendre les dues interfícies mencionades anteriorment, aquesta interfície no necessita una classe que la implementi, ja que Spring crearà automàticament implementacions basades en la nomenclatura dels mètodes definits. Per altra banda, la interfície *EntityRepository* també estén la *CustomEntityRepository*, que conté altres operacions que no són implementades automàticament per Spring i que requereixen d'una classe que les implementi, treballant amb altres objectes per tal d'accedir a la base de dades (Figura 18).

El problema apareix en aquesta distinció que es fa entre les interfícies EntityRepository i CustomEntityRepository, ja que no es fa servir un criteri clar per dividir les operacions entre aquestes dues interfícies. Si bé és cert que hi ha situacions on els mecanismes que proporciona Spring no són suficients per generar automàticament el codi que accedeix a la base de dades, en el *dashboard* no hi ha cap operació que tingui una complexitat tan gran com per requerir-ho. Davant d'aquesta situació, un desenvolupador que vulgui afegir o modificar operacions dels repositoris es troba amb operacions molt similars que es duen a terme de maneres diferents en funció de les entitats amb les que es treballa, sense cap criteri aparent. Això dificulta la mantenibilitat i fa que l'esforç per realitzar una modificació dels repositoris augmenti.

6.5.3. Anàlisi de les alternatives

Per tal de millorar la situació actual, cal veure quins són els mecanismes que proporciona Spring per construir repositoris i veure quin s'adapta millor a les característiques del dashboard, alhora que contribueix a simplificar l'estructura actual.

El primer que cal veure són les interfícies que Spring posa a la disposició dels desenvolupadors per definir les operacions que formaran els repositoris. Les més importants es mostren a la figura següent, que plasma la jerarquia existent entre elles:

Figura 21. Interfícies dels repositoris

Com es pot apreciar, es tracta d'interfícies genèriques que requereixen dos tipus de dades: el primer és el tipus de l'entitat del domini que el repositori gestiona, i el segon és el tipus de l'identificador d'aquesta entitat. Les responsabilitats de cada interfície són les següents:

- ***Repository***: Es tracta de la interfície bàsica per definir repositoris, i el seu propòsit principal és mantenir informació sobre els tipus de dades a més de permetre a Spring descobrir interfícies que estenen a aquesta durant l'escaneig del *classpath* per tal que aquest pugui crear les implementacions adients. No defineix cap signatura d'operació.

- ***CrudRepository***: Interfície que defineix operacions CRUD genèriques (creació, lectura, modificació i esborrat) sobre una determinada entitat del domini.

- ***PagingAndSortingRepository***: Interfície que defineix operacions que permeten obtenir entitats del domini d'una forma controlada, mitjançant paginació (obtenció del resultats dividits en blocs o pàgines) o seguint un determinat ordre.

- ***JpaRepository***: Interfície que defineix operacions específiques de la *Java Persistence API*, com ara la eliminació dels canvis pendents (*flush*) a la base de dades i l'eliminació de contingut en lot (*batch*).

La part més interessant d'aquestes interfícies no són només els mètodes que aquestes defineixen, sinó la possibilitat de definir-ne de nous en funció de les necessitats dels sistema. Quan es crea una nova interfície que estén a una de les esmentades, Spring permet definir signatures de nous mètodes seguint una nomenclatura específica per tal de realitzar un conjunt molt divers d'operacions, que s'adapten a l'entitat del domini que s'estigui gestionant. El nom d'aquests mètodes conté el tipus d'operació que es vol realitzar i el nom dels atributs de l'entitat (o d'objectes que l'entitat té com a atribut) rellevants per la mateixa, que es passaran com a paràmetres del mètode en qüestió. Amb aquesta informació, Spring és capaç de crear una implementació de manera automàtica que realitzi les operacions definides per la interfície sense necessitat d'afegir més codi addicional. En cas que la signatura del mètode no estigui ben definida i la operació no es pugui realitzar, Spring ens advertirà al respecte en temps de compilació.

Tot i que aquest mecanisme és molt potent, a vegades no és suficient per realitzar totes les operacions necessàries pel sistema. Per aquests casos es disposa de la anotació *@Query* que, situada sobre la signatura d'un mètode definit en una interfície que estén a *JpaRepository*, permet definir manualment la operació que es realitzarà a la base de dades. Aquesta operació pot estar escrita en el llenguatge de consulta de JPA o bé en SQL natiu, tot i que en cas d'escollir la segona opció es perd la capacitat d'independència

de la plataforma de base de dades que aporta el llenguatge de JPA. Aquestes operacions també admeten paràmetres i, de la mateixa manera que els mètodes anteriors, en cas de que la operació no es pugui dur a terme correctament Spring mostrarà un error informant del problema.

6.5.4. Justificació de la solució

Per tal de simplificar els repositoris del dashboard, el primer que cal fer és triar correctament la interfície que aquests implementen. Si s'analitza la jerarquia descrita a la Figura 21, es pot veure com la interfície *JpaRepository* estén a la *PagingAndSortingRepository* cosa que implica que, quan s'estén a *JpaRepository* per crear una nova interfície, aquesta nova també inclourà els mètodes que defineix *PagingAndSortingRepository*. En l'estat actual, les interfícies *EntityRepository* estenen explícitament aquestes dues interfícies introduint una redundància en el codi, i per tant cal que únicament facin extensió de *JpaRepository*.

Per altra banda, cal valorar si *JpaRepository* és la interfície adequada en tots els casos. Tal i com s'ha vist a l'apartat anterior, aquesta interfície permet fer servir mètodes propis de la JPA, a més de permetre anotar mètodes amb @Query per definir operacions complexes. En els casos on les operacions que hagi de proporcionar un *EntityRepository* tractin únicament amb la entitat que aquest gestiona, es pot estendre la interfície *CRUDRepository* per obtenir els mètodes bàsics i complementar-la amb la definició manual de mètodes que seran implementats automàticament per Spring, si és necessari. En els casos on aquestes operacions bàsiques no siguin suficients, caldrà estendre JpaRepository amb la finalitat de fer servir l'anotació @Query amb la operació escrita de forma manual allà on sigui necessari. Amb aquest enfocament s'elimina la necessitat d'implementar de forma manual les operacions, cosa que fa que sigui del tot innecessari disposar d'interfícies *CustomEntityRepository* i les seves respectives implementacions *CustomEntityRepositoryImpl*.

6.6. Deficiències en l'estat general del codi

6.6.1. Síntesi

Escenari	
Es vol modificar el comportament de la funcionalitat del *dashboard* que s'encarrega de realitzar el càlcul de les avaluacions dels indicadors estratègics. Per fer-ho, cal examinar més de 250 línies de codi que contenen, entre altres coses, variables amb noms críptics, funcions amb un nombre molt elevat d'arguments, comentaris no actualitzats i un estil de programació irregular.	
Problema	
Aquelles seccions de codi on la lògica és complexa sovint presenten *code smells* i un disseny del codi millorable, que combinat amb alguns problemes de seguretat i *bugs* dificulten la comprensió del codi per part dels desenvolupadors i afecten al manteniment d'aquestes funcionalitats.	
Alternatives de solució	**Opcions escollides**
Codebeat	❌

Codacy	✗
SonarCloud	✓
Solució	
Per mitigar aquesta situació s'utilitzarà l'eina d'anàlisi estàtic de codi SonarCloud per tal d'analitzar quins són els problemes actuals que el sistema presenta i per mantenir un control sobre els problemes solucionats. Es començarà per resoldre *bugs* i vulnerabilitats del codi, i el darrer pas serà rebaixar en la major quantitat possible els *code smells* detectats per l'eina.	

6.6.2. Descripció del problema

Tal i com es pot apreciar, en aquest escenari torna a entrar en joc el requisit de mantenibilitat. Tot i que en general el *dashboard* no té funcionalitats massa complexes, sí que existeixen certes parts del sistema on el codi necessari per dur-les a terme és extens, i qualsevol modificació que es vulgui realitzar ha d'anar precedida per la comprensió per part del desenvolupador del codi d'aquesta funcionalitat. Aquesta tasca es veu entorpida pels anomenats *code smells*, que és el nom amb el que es coneixen a les males pràctiques que fan que el codi sigui confús i difícil de mantenir, i que sovint són el símptoma d'un error de disseny.

A part dels *code smells*, hi ha altres elements que es posen de manifest en funcionalitats complexes com la descrita l'escenari anterior, com poden ser els errors de seguretat. El fet d'utilitzar Spring com a *framework* per l'aplicació web fa que s'hagin de seguir una sèrie de regles tant per la configuració com pel contingut del sistema per tal d'evitar problemes relacionats amb la seguretat de l'aplicació. Aquestes regles no sempre se segueixen en l'estat actual del *dashboard*, ja que existeix un gran desconeixement de quines són i no és trivial adonar-se que representen un problema pel sistema.

Per últim, els fragments complexes de codi també són més propensos a contenir erros de funcionament o *bugs*. Aquests errors poden passar desapercebuts pel desenvolupador quan hi ha una gran quantitat de codi per gestionar, i si es troben en parts de la funcionalitat que no s'executen amb freqüència (com per exemple, la gestió d'errors) dificultarà molt la seva detecció en temps d'execució.

6.6.3. Anàlisi de les alternatives

Per tal de donar solució als problemes exposats a l'apartat anterior, un enfoc manual és massa costós de dur a terme degut a la gran quantitat de codi que s'ha de revisar i a la variabilitat dels resultats en funció del grau de coneixement i habilitats del desenvolupador. Afortunadament, existeixen eines que poden automatitzar aquest procés mitjançant una tècnica coneguda com a anàlisi estàtic de codi, que consisteix en la inspecció del codi d'un sistema sense executar per tal de descobrir errors i problemes en la qualitat del mateix mitjançant un conjunt extens de normes.

Avui en dia existeixen una gran quantitat d'eines d'anàlisi estàtic de codi que ofereixen funcionalitats molt potents per tal de millorar l'estat general del codi d'una aplicació. Una primera distinció que es pot fer entre aquestes eines és en funció de si requereixen la instal·lació i manteniment d'una instància pròpia en un servidor o si estan allotjades al núvol i l'eina s'ofereix com un servei. Degut a la falta d'infraestructura pròpia i als

inconvenients que suposa mantenir una eina i un servidor propis, s'opta per contemplar únicament aquelles eines allotjades al núvol. Dins d'aquesta categoria, algunes de les més utilitzades són les següents:

Codebeat

Codebeat [44] monitoritza i analitza codi en 9 llenguatges de programació diferents, i permet enviar informació a les eines de gestió de repositoris git i serveis de comunicació més coneguts per tal de poder veure els resultats de l'anàlisi sense necessitat de sortir de l'eina en qüestió.

Per tal d'analitzar l'estat actual del codi, Codebeat fa servir els seus propis algorismes enlloc d'utilitzar-ne alguns més reconeguts com fan altres eines. Alguns dels factors que aquests algorismes inspeccionen són els següents:

- Nombre d'assignacions, delegacions de control i condicionals d'una secció del codi.
- Profunditat d'aniuament dels blocs de codi.
- Complexitat ciclomàtica (nombre de camins d'execució independents en una secció de codi).
- Nombre d'arguments que una funció accepta.
- Duplicació de blocs de codi.

A més, Codebeat permet mostrar el percentatge de *coverage* de proves que té el sistema mitjançant la importació d'informes elaborats per eines externes. Tot i això, aquesta funcionalitat només està disponible pels llenguatges Ruby, JavaScript, TypeScript i Golang.

Codacy

Codacy [45] permet analitzar 15 llenguatges de forma nativa, a més d'incorporar-ne d'altres mitjançant extensions realitzades per la comunitat d'usuaris. També s'integra amb les eines de gestió de repositoris git i serveis de gestió d'equips i comunicació més utilitzats amb l'objectiu de proporcionar informació sobre l'anàlisi.

Les mesures de qualitat estan agrupades en 8 categories diferents:

- Complexitat: mètodes i classes amb una alta complexitat.
- Compatibilitat: problemes de visualització en diferents versions de navegadors.
- Codi propens a errors: codi que pot amagar *bugs* i paraules reservades dels llenguatges que han de ser utilitzades amb cautela.
- Seguretat: problemes de seguretat.
- Estil de codi: problemes de format i sintaxi.
- Documentació: mètodes i classes que no tenen la documentació adequada.
- Rendiment: codi que pot ser ineficient.
- Codi no utilitzat: variables i mètodes no utilitzats i codi que mai pot ser executat.

Dins d'aquestes categories es troben els problemes i errors que Codacy ha trobat al codi per mitjà de la utilització de diversos motors d'anàlisi externs en funció del llenguatge

de programació que s'analitza. Això implica que el rendiment aconseguit per cada projecte és variable, i dependrà molt de la potència del motor que s'utilitza pel seu llenguatge.

Aquesta eina permet definir objectius pels projectes, tant per fitxer com per categoria, i recomana accions a seguir i problemes a resoldre per tal d'assolir els objectius fixats. A més, gràcies a la potent integració amb els gestors de repositoris git Codacy permet definir llindars de qualitat per les *Pull Request*, de manera que l'increment que es vol introduir pot ser descartat o aprovat en funció de la qualitat del nou codi.

Pel que fa al *coverage* de les proves, Codacy mostra els resultats dels informes que elaboren una gran varietat d'eines externes que analitzen les proves dels següents llenguatges de programació: Java, JavaScript, TypeScript, CoffeeScript, PHP, Python Scala i Ruby.

SonarCloud

SonarCloud [46] és la versió al núvol de la popular eina d'anàlisi estàtic de codi anomenada SonarQube [47]. Permet analitzar 23 llenguatges de programació i s'integra amb facilitat amb les principals eines de gestió de repositoris git, tot i que no permet la integració amb eines de gestió de projectes i comunicació de forma nativa.

Aquesta eina utilitza motors d'anàlisi propis anomenats *sonars*, que consten d'un conjunt de regles dividides en tres categories: *bugs*, *code smells* i vulnerabilitats. A la vegada, les regles informen sobre l'impacte de cada problema indicant el seu grau de severitat. Aquests *sonars* es beneficien de l'experiència adquirida durant el llarg període de temps que SonarQube porta en funcionament i incorporen conjunts molt grans i robustos de regles per la major part dels llenguatges que suporta. Per exemple, en el cas del *sonar* de Java, consta de més de 530 regles que fan referència tant al llenguatge en si com a *frameworks* populars com ara Struts, Spring i Hibernate.

Un altre dels aspectes a destacar dels sonars és la incorporació de regles que deriven de estàndards reconeguts de seguretat, com ara el CWE (Common Weakness Enumeration), SANS Top 25, OWASP (Open Web Application Security Project), MISRA (Motor Industry Software Reliability Association) i SEI CERT Coding Standards. D'aquesta manera s'aconsegueix una inspecció de seguretat profunda i fiable, que cobreix els principals punts d'interès segons els estàndards mencionats.

La informació extreta dels anàlisis s'exposa dividida en les següents categories:

- Confiabilitat: codi on el comportament pot ser diferent de l'esperat.
- Seguretat: codi vulnerable davant un possible atac.
- Mantenibilitat: codi on una modificació és més costosa del que hauria de ser.
- Cobertura: percentatge de codi executat per les proves automatitzades.
- Duplicacions: fragments de codi repetits.
- Complexitat: grau de complexió del flux de control d'un fragment de codi. Inclou tant la complexitat ciclomàtica com la cognitiva.

Per tal de configurar l'anàlisi, SonarCloud utilitza perfils de qualitat on es poden definir quines són les regles concretes que s'utilitzaran en cas que els *sonars* no s'adaptin a les

necessitats del projecte. També permet definir els llindars de qualitat per tal d'adaptar el resultat de l'anàlisi als requisits del sistema.

De la mateixa manera que en les eines anteriors, el *coverage* de les proves automatitzades s'obté per mitjà de la integració d'informes elaborats per eines externes compatibles amb els següents llenguatges de programació: C, C++, C#, Objective-C, Flex, Go, Java, JavaScript, Kotlin, PHP, Python, Ruby, Scala, Swift i VisualBasic.

6.6.4. Justificació de la solució

Un cop vistes les eines més utilitzades per realitzar anàlisi estàtic de codi, cal prendre una decisió sobre quina opció és la més adequada pel *dashboard*. Per començar, es descarta utilitzar Codebeat degut a que és la opció que menys prestacions ofereix, amb un conjunt de regles molt limitat i sense possibilitat d'integrar informes sobre el *coverage* de les proves automatitzades en Java.

D'entre les dues opcions restants, l'eina escollida ha estat SonarCloud. En general, tant aquesta eina com Codacy són dues alternatives robustes i potents que proporcionen resultats molt complets. Tot i això, hi ha certs aspectes que fan decantar la balança cap a SonarCloud. En primer lloc, el fet de derivar de SonarQube confereix a SonarCloud una avantatja estratègica important, ja que s'aprofita el coneixement adquirit durant anys per desenvolupar i millorar els *sonars*. D'entre les eines analitzades, SonarCloud és la que té un conjunt de regles més extens i, a més, compta amb l'afegit que conté regles dedicades a Spring, cosa que encaixa perfectament amb el *dashboard*. Un altre punt a favor de SonarCloud són els estàndards de seguretat, ja que proporcionen informació contrastada i verídica sobre quins problemes reals de seguretat representen una amenaça pel sistema i com solucionar-los.

Si bé és cert que Codacy disposa d'una major integració amb eines externes, aquest fet esdevé irrellevant si es té en compte que l'única eina que cal integrar GitHub. Tant Codacy com SonarCloud permeten comunicar-se i interactuar amb aquesta eina, tot i que pel que fa a les *Pull Request* Codacy proporciona un major control al incorporar la possibilitat de definir llindars d'acceptació per les mateixes.

Un cop decidit que l'eina d'anàlisi estàtic de codi serà SonarCloud, cal veure quins passos cal seguir en el moment d'integrar-la amb el *dashboard*. Tal i com s'ha comentat, aquest tipus d'eines estan dissenyades per analitzar el codi de manera incremental a mesura que es va introduint, per tal de solucionar quantitats petites de problemes en un període curt de temps a partir del moment en el que es detecten. En el cas del *dashboard*, quan es realitzi el primer anàlisi amb SonarCloud es detectarà una quantitat molt gran de problemes i errors. Per tal de donar-hi solució, es prioritzaran els problemes en funció de la categoria a la qual pertanyen i la seva severitat. En primer lloc, caldrà modificar el codi on es detectin *bugs* i vulnerabilitats per tal de resoldre les situacions que tenen un efecte directe en el funcionament del *dashboard*. Un cop fet això, el següent pas és resoldre els *code smells* per tal de millorar la mantenibilitat del codi. Com que aquesta categoria és la que més regles incorpora SonarCloud, també serà la més nombrosa en quant a errors detectats. Com que l'abast d'aquest projecte és limitat, es començarà per resoldre aquells que tinguin una severitat més alta i se seguirà fins que el temps dedicat a aquesta tasca ho permeti.

6.7. Manca d'integració en el procés de desenvolupament

Per aquest últim problema, el plantejament inicial varia una mica respecte als anteriors. Així com en els punts tractats fins ara s'han analitzats escenaris que deriven directament de la situació actual del *dashboard*, en aquest es vol analitzar quin és l'efecte que provoca l'aplicació de les solucions proposades en punts anteriors davant d'un determinat escenari. En concret, es parteix de la premissa que s'han introduït al *dashboard* les solucions proposades als apartats Falta de proves automatitzades i Deficiències en l'estat general del codi.

6.7.1. Síntesi

Escenari	
Un desenvolupador ha creat una nova funcionalitat pel *dashboard* en una branca separada del repositori, i vol integrar-la amb el sistema. La resta de membres de l'equip de desenvolupament del *dashboard* ha de poder comprovar que el codi d'aquesta nova funcionalitat compleix amb els paràmetres de qualitat esperats per tal d'evitar introduir errors i deute tècnic al sistema.	
Problema	
Partint de la integració de proves automatitzades i de l'eina SonarCloud tractades en apartats anteriors, es pot observar com el resultat d'aplicar aquests processos no s'integra de forma efectiva dins del procés de desenvolupament, donant lloc a una necessitat de realitzar accions addicionals per tal de rebre informació al respecte per part de la resta de l'equip de desenvolupament.	
Alternatives de solució	**Opcions escollides**
Codeship	✗
Travis CI	✓
Circle CI	✗
Solució	
Per tal de millorar el procés de desenvolupament, es proposa fer servir l'eina d'integració contínua Travis CI. Aquesta eina permetrà executar les proves automatitzades i iniciar un anàlisi a SonarCloud cada cop que es realitzi un *commit* al repositori de GitHub, mostrant els resultats d'ambdues accions directament dins de les *Pull Requests* creades a partir de les branques que contenen noves funcionalitats. D'aquesta manera s'aconsegueix tancar el cicle i mantenir a l'equip de desenvolupament informat en tot moment, així com aprofitar els beneficis de la integració contínua per detectar errors amb més rapidesa i facilitat.	

6.7.2. Descripció del problema

El requisits que es veu afectat per aquest escenari és la mantenibilitat. Partint de la integració de l'eina SonarCloud per tal d'analitzar la qualitat del codi, es pot observar que seguint la solució plantejada els resultats d'aquest anàlisi queden aïllats dins de l'eina, i per esbrinar quina és l'avaluació de la qualitat cal obrir manualment la pàgina de SonarCloud. Si aquest procés no es realitza amb regularitat es poden passar per alt situacions susceptibles de millora que es poden acumular i esdevenir un problema greu pel dashboard. A més, aquesta informació no queda disponible de forma senzilla pels

desenvolupadors, ja que aquests poden no estar al corrent de que aquesta eina està sent utilitzada en el procés de desenvolupament.

Per altra banda, per assegurar que no s'introdueix cap error que afecti a aquesta funcionalitat o a la resta del *dashboard* cal executar les proves automatitzades. Quan un desenvolupador fa una proposta de canvis, es confia en que aquest ha afegit les proves necessàries i les ha executat juntament amb les ja existents per tal de corroborar que el sistema funciona tal i com ho ha de fer. Aquest procés no queda plasmat enlloc, i no es pot tenir plena confiança en que s'hagi dut a terme abans d'integrar el nou codi amb la resta de l'aplicació. En cas que no hagi sigut així i el nou codi introdueixi algun tipus d'error que faci fallar les proves, el procés necessari per descobrir l'error, notificar al desenvolupador implicat i esperar una correcció es pot allargar més de l'esperat.

6.7.3. Anàlisi de les alternatives

Tal i com es pot constatar, el problema al que es vol donar solució consisteix en optimitzar el procés de desenvolupament per tal d'aprofitar al màxim les proves automatitzades i l'eina d'anàlisi estàtic de codi introduïdes en apartats anteriors. Per tal de dur-ho a terme, s'aprofitaran algunes de les característiques que la plataforma GitHub posa a la disposició dels desenvolupadors. En concret, l'enfocament proposat té com a eix central les anomenades *Pull Requests*.

Una Pull Request és una petició d'integració d'una sèrie de canvis al repositori principal de l'aplicació. Aquests canvis provenen d'una branca del mateix repositori creada específicament per la funcionalitat que es vol introduir, i generalment tenen com a destí una de les branques estables del repositori (*develop* o *master*). L'objectiu és comunicar a altres desenvolupadors quins són els canvis que es volen introduir, de manera que es disposi d'un lloc on es puguin veure, revisar i discutir possibles modificacions. En cas de que una *Pull Request* sigui acceptada, els *commits* que la conformen passaran a formar part de la branca de destí.

Com a característica addicional, GitHub permet la integració d'eines externes dins de les *Pull Requests* per tal d'afegir noves funcionalitats que aporten més informació sobre el codi que es vol introduir. Com ja s'ha comentat, una de les eines que es vol integrar dins de les *Pull Requests* és SonarCloud, i aquesta disposa de compatibilitat de forma nativa amb elles. Es pot configurar l'eina per tal que mostri informació sobre l'estat de l'avaluació de qualitat del codi, i informar en cas que es detectin problemes per tal que puguin ser corregits abans d'integrar el nou codi amb la branca corresponent del repositori.

L'altre aspecte que queda per integrar dins de les *Pull Requests* és la execució de les proves automatitzades. Per tal de poder realitzar aquesta acció cal recórrer a eines addicionals que s'encarreguin de recopilar el nou codi que es vol integrar, executar aquestes proves i informar sobre l'estat d'execució de les mateixes. Per aconseguir aquest objectiu cal recórrer a les eines d'integració contínua.

La integració contínua és una tècnica de desenvolupament que consisteix en l'actualització freqüent del codi d'un repositori centralitzat per tal de realitzar integracions automàtiques d'un projecte amb l'objectiu de detectar possibles errors

amb més diligència i facilitat. S'entén per integració el procés de compilació i execució de proves de tot un projecte. Per aquest propòsit s'utilitzen eines externes que permeten controlar aquestes execucions i realitzar diverses tasques a partir del codi que es troba als repositoris.

Actualment existeixen una gran varietat d'eines d'integració contínua disponibles, i de la mateixa manera que per les eines d'anàlisi estàtic de codi en trobem algunes que requereixen la instal·lació i manteniment d'una instància pròpia en un servidor i d'altres que estan allotjades al núvol. Seguint el mateix raonament que en el punt anterior, només es contemplen les eines que es proporcionen com a servei des del núvol. Aquestes són algunes de les eines més utilitzades:

Codeship

Codeship [48] ofereix un conjunt de comandes bàsiques per una gran quantitat de tecnologies, i a més permet definir-ne d'altres manualment en cas que les tasques a executar no estiguin disponibles. També proporciona accés a diverses bases de dades, per les quals ofereix una configuració automàtica que facilita la seva posada en funcionament.

La configuració de Codeship es realitza exclusivament des de la interfície d'usuari del seu portal web, cosa que dificulta la reutilització d'aquesta configuració per altres projectes. Aquest portal també mostra informació sobre les tasques executades, mostrant per cada *commit* quin és l'estat d'execució de la seva respectiva *build*[3] amb informació detallada per cadascuna de les tasques i els *logs* generats. En la seva versió gratuïta, Codeship ofereix l'execució de 100 *builds* mensuals.

Travis CI

Travis CI [49] és un dels serveis d'integració contínua que porta més anys al mercat, i és àmpliament utilitzat. Un dels aspectes pel qual destaca és per la seva dinàmica comunitat i per la seva bona documentació. És compatible amb un ampli conjunt de tecnologies, bases de dades i serveis externs, la configuració dels quals es realitza a través d'un arxiu anomenat *.travis.yml*. A més, permet distribuir les tasques executades de forma paral·lela, creant diverses *builds* a partir d'un sol *commit* amb l'objectiu d'accelerar la seva execució.

La pàgina principal del seu portal web mostra, per cada projecte, quina és la darrera *build* executada i el seu estat. Dins d'un projecte es mostra l'històric de *builds* executades, amb informació sobre els logs de les tasques, quin *commit* ha provocat la *build* i un enllaç cap al repositori centralitzat per veure els canvis que introdueix el *commit* en qüestió. Per últim, una de les grans avantatges de Travis CI es que ofereix *builds* il·limitades per projectes de codi obert de forma gratuïta.

[3] Execució automàtica de les tasques escollides per l'usuari per part del servei d'integració contínua. En general, aquestes tasques consisteixen en la compilació i l'execució de les proves automatitzades, entre d'altres.

Circle CI

Circle CI [50] és una eina d'integració contínua que es caracteritza principalment per la forma en la que ofereix els seus serveis. La unitat bàsica dins d'aquesta eina són els contenidors, que representen una única instància d'una màquina virtual on es poden executar diverses tasques. Circle CI ofereix diversos plans de subscripció que permeten obtenir diversos contenidors per tal de maximitzar el paral·lelisme de les *builds* que es realitzen, tot i que la versió gratuïta només n'ofereix un.

De la mateixa manera que les eines anteriors, Circle CI permet utilitzar un conjunt molt ampli de llenguatges de programació, de bases de dades i de serveis externs. Per tal de descriure la configuració del projecte, s'utilitza un arxiu anomenat circle.yml que conté informació sobre les tecnologies i serveis, així com del contingut de les tasques que cal executar.

El portal web de Circle CI mostra l'estat de les darreres *builds* executades per cada projecte, ordenades cronològicament. Dins d'una *build*, es mostra la informació relativa a cada tasca acompanyada dels *logs* de la mateixa.

6.7.4. Justificació de la solució

Un cop vistes les alternatives, cal considerar quina és l'eina d'integració contínua més adequada pel *dashboard*. Si bé és cert que no hi ha diferències massa significatives pel que fa les funcionalitats principals de les eines tractades (totes permeten executar les proves i integrar els resultats dins de les *Pull Requests*), hi ha una que destaca per sobre de les altres: Travis CI. Els motius principals són la bona documentació i comunitat que recolza l'eina, i el fet que proporcioni *builds* il·limitades per projectes de codi obert, com és el cas del *dashboard*.

Considerant llavors que l'eina escollida és Travis CI, cal reconsiderar quin serà el procés que se seguirà quan es vulgui afegir nou codi al sistema per mitjà d'una *Pull Request* a través de GitHub. Recapitulant sobre el que s'ha comentat en aquest apartat, el que es vol és integrar per una banda SonarCloud per realitzar anàlisi estàtic del codi i per altra executar les proves automatitzades dins de Travis CI. Tal com s'ha explicat anteriorment, SonarCloud ja disposa d'un mecanisme natiu per integrar-se amb GitHub i executar anàlisis cada cop que es vol afegir nou codi al repositori. Tot i això, si se segueix aquest enfocament s'impedeix utilitzar una de les funcionalitats que proporciona SonarCloud: l'informe sobre el *coverage* de les proves automatitzades.

De la mateixa manera que totes les eines d'anàlisi estàtic de codi, SonarCloud no té capacitat per si mateixa d'esbrinar quin és el percentatge de *coverage* de les proves ni quines línies de codi són executades per aquestes. Per aquest propòsit cal proporcionar a SonarCloud un arxiu amb un informe extret en el moment de l'execució de les proves automatitzades, que posteriorment l'eina interpretarà i mostrarà de forma gràfica. Gràcies al fet d'incorporar Travis CI dins del procés de desenvolupament, aquesta eina es pot encarregar d'obtenir aquest informe i invocar manualment a SonarCloud per realitzar l'anàlisi proporcionant les dades necessàries, sense necessitat de recórrer a la integració nativa. Dins de les *Pull Request* el resultat de l'anàlisi seguirà mostrant-se de la mateixa forma, fent aquest procés transparent pels desenvolupadors.

Seguint aquest enfocament, els passos que se seguiran dins de Travis CI són els següents. Primerament, quan s'activa una nova *build* la primera tasca que s'executarà són les proves automatitzades juntament amb la inspecció del codi per determinar-ne el *coverage*. Aquestes dues tasques s'executen simultàniament, ja que la inspecció es fa de manera dinàmica en el moment de l'execució de les proves. A continuació caldrà executar una tasca per obtenir l'informe de *coverage* en un format adequat per SonarCloud (en aquest cas, en format XML) i guardar-lo a la ubicació indicada dins de la configuració de l'eina. Un cop fet això, Travis CI invocarà l'anàlisi de SonarCloud per tal que aquesta eina pugui analitzar el codi amb totes les dades necessàries.

Dins de les *Pull Requests*, els desenvolupadors podran veure quin és l'estat de l'execució de Travis CI i de l'anàlisi de SonarCloud, per separat. GitHub permet configurar les Pull Request de manera que només puguin ser acceptades si es donen una sèrie de condicions prèvies. En aquest cas, es configuraran per tal que només es puguin acceptar si la *build* de Travis CI s'executa correctament (és a dir, que les proves no fallin i que es pugui invocar a l'anàlisi de SonarCloud). Els errors que detecti SonarCloud en el seu anàlisi no es consideraran un factor bloquejant per la *Pull Request*, ja que es poden donar situacions on els errors que detecti siguin negligibles o no adequats pels interessos del *dashboard*. En aquests casos, mitjançant el portal web de SonarCloud es podran desactivar les regles que es considerin oportunes, sempre seguint les decisions consensuades per l'equip de desenvolupament.

Per últim, cal saber com i quan s'activarà una nova *build* i s'executaran totes aquestes tasques. Quan es realitza un *commit* en una branca del repositori principal, Travis CI s'activa i inicia una *build* nova amb el contingut d'aquest nou *commit*. En canvi, quan es crea una *Pull Request* a partir d'una de les branques del repositori, Travis CI executa la nova *build* sobre el resultat d'integrar tots els canvis presents a la branca d'origen amb la branca de destí. Això és possible gràcies a que GitHub crea un *commit* especial de *merge* en el moment de crear una nova *Pull Request* i cada cop que aquesta s'actualitza amb nou contingut, que entre altres coses permet a GitHub determinar si la *Pull Request* es pot integrar amb la branca de destí sense problemes. Aquest *commit* no apareix a l'històric, però gràcies a ell es pot executar una *build* que permet esbrinar quin és el resultat de la integració abans de fer-la efectiva.

7. Requisits

En aquest apartat s'exposen quins són els requisits que cal tenir presents per tal d'aplicar el procés de reenginyeria. Tal i com ja s'ha explicat anteriorment, el propòsit d'aquest procés se centra en millorar la qualitat del *dashboard* estratègic a la vegada que es preserva el seu funcionament actual, cosa que fa que els requisits que s'han de tenir presents siguin requisits de qualitat.

Dins d'aquesta categoria, a partir de l'anàlisi dels problemes es pot observar com els tres requisits de qualitat que queden afectats en l'estat actual són la mantenibilitat, la documentació i la compatibilitat. En la taula següent (Taula 13) es pot veure quina és la relació entre els problemes identificats en l'apartat anterior i els requisis de qualitat esmentats:

Problemes identificats	**Mantenibilitat**	**Documentació**	**Compatibilitat**
Falta de proves automatitzades	✓		
Documentació manual dels serveis REST	✓	✓	
Disseny inconsistent dels serveis REST	✓		✓
Múltiples responsabilitats dels controladors REST	✓		
Estructura complexa i implementació manual dels repositoris	✓		
Deficiències en l'estat general del codi	✓		
Manca d'integració en el procés de desenvolupament	✓		

Taula 13. Relació entre els problemes identificats i els requisits de qualitat

Aquests requisits ja es contemplen a la documentació actual del dashboard, però el seu reduït nivell de detall fa que siguin insuficients per tal de conservar el seu propòsit. A continuació es detallen cadascun d'ells, i s'afegeixen criteris de satisfacció per tal de poder comprovar que els requisits s'hagin complert un cop aplicats els canvis que conformen el procés de reenginyeria.

Requisit
Mantenibilitat
Descripció
El codi que conforma els components interns del dashboard ha de mantenir un nivell elevat de reusabilitat i un disseny que faciliti el seu manteniment per part de l'equip de desenvolupament.
Justificació
La mantenibilitat té una importància clau dins del *dashboard* estratègic, ja que al tractar-se d'un projecte de recerca cal poder adaptar amb rapidesa i facilitat el sistema als nous requisits que puguin sorgir durant la investigació. A més, el fet de que el *dashboard* sigui accessible com a projecte de codi obert fa que la mantenibilitat del codi sigui indispensable per tal de poder encoratjar a la comunitat a afegir i modificar funcionalitats.
Criteris de satisfacció

Per tal d'assegurar la mantenibilitat del *dashboard*, cal reorganitzar i reescriure diverses parts del sistema per tal d'adequar-les a les bones pràctiques i principis del disseny de software i seguir patrons de disseny quan sigui convenient. Els principis i patrons propis del disseny de software han estat àmpliament validats per nombrosos estudis, i la seva utilització garanteix un benefici de cara a la mantenibilitat del sistema.
En finalitzar el procés de reenginyeria, els problemes detectats per l'eina SonarCloud relacionats amb la mantenibilitat s'hauran de veure reduïts fins arribar a les següents fites: eliminar completament els *bugs*, les vulnerabilitats, i els *code smells* classificats com a severitat bloquejant i crítica. A partir d'aquí, a mesura que el sistema creixi i incorpori noves funcionalitats, els problemes detectats per aquesta eina no hauran de superar el 20% respecte al nombre de línies de codi del sistema.
Per últim, el sistema haurà de comptar amb proves unitàries allà on la lògica sigui complexa i crítica pel funcionament del sistema i amb proves d'integració pels serveis REST i els repositoris.

Taula 14. Requisit de mantenibilitat

Requisit
Documentació
Descripció
El *dashboard* ha de comptar amb una guia d'usuari descrivint com utilitzar el sistema, un vídeo tutorial descrivint i mostrant les funcionalitats principals i documentació tècnica amb la informació necessària per integrar el *dashboard* amb altres productes.
Justificació
Per una banda, cal disposar de documentació orientada als usuaris per tal de poder transmetre tot el potencial de l'eina i permetre que la coneguin a fons i obtinguin el major benefici possible. Per altra banda, la documentació tècnica és fonamental per tal de poder dur a terme integracions del dashboard amb altres aplicacions i serveis, de manera que la informació necessària estigui disponible en tot moment i faciliti aquest procés.
Criteris de satisfacció
El *dashboard* estratègic ha de comptar amb documentació tècnica accessible en format web que ha de descriure els diferents serveis REST, mostrant informació sobre les URIs, mètodes HTTP, paràmetres de consulta, format de les respostes i codis HTTP. La documentació orientada als usuaris ja es troba present en l'estat actual, i per tant no s'inclou dins dels criteris de satisfacció corresponents a aquest procés de reenginyeria.

Taula 15. Requisit de documentació

Requisit
Compatibilitat
Descripció
Ha de ser possible integrar el *dashboard* amb eines externes i proporcionar interoperabilitat amb sistemes existents.

Justificació
El *dashboard* estratègic és una eina que s'integra i forma part del procés de desenvolupament de software àgil, i com a tal conviu amb eines i serveis diversos dins d'aquest cicle. És important que es pugui integrar directament amb aquestes eines i serveis per tal que pugui dur a terme les seves funcionalitats amb una major flexibilitat i adaptant-se a les noves formes d'ús que convinguin als usuaris.
Criteris de satisfacció
Per satisfer aquest requisit no és suficient amb que el dashboard sigui compatible amb sistemes externs, sinó que la qualitat d'aquesta integració s'ha de maximitzar. Per tant, els serveis REST del dashboard s'hauran d'adequar a les bones pràctiques de disseny utilitzant el patró *Resource API*, utilitzant els mètodes HTTP per transmetre intencionalitat, URIs basades en recursos i missatges d'error clars i entenedors.

Taula 16. Requisit de compatibilitat

8. Disseny

En aquest apartat es plasma el disseny de les millores a realitzar durant el procés de reenginyeria. Degut a la seva naturalesa i al tipus de feina que impliquen, no totes les millores exposades anteriorment són compatibles amb la realització d'un disseny. Les millores que consten en aquest apartat són aquelles que representen el tronc estructural de l'aplicació i que requereixen alterar un nombre considerable de components per tal de dur-les a terme. Per aquesta raó, les millores que es tracten en aquest apartat són el disseny dels serveis REST, la separació dels controladors REST i el domini i la simplificació dels repositoris.

8.1. Disseny dels serveis REST

Per tal de mostrar quins són els canvis en el disseny dels serveis REST, es descriuen diversos atributs de les funcionalitats que proporcionen aquests serveis i que, en l'estat inicial, contenen elements que no s'ajusten a tot allò descrit a l'apartat 6.3.3. Principis i bones pràctiques. A la següent taula (Taula 17) es poden observar quines són les característiques d'una de les funcionalitats que cal modificar, indicant tant l'estat inicial del qual es parteix com el resultant del nou disseny. La resta de taules amb els canvis dissenyats per les altres funcionalitats es poden trobar al capítol 16. Annex.

Crear indicador estratègic		
	Inicial	**Nou**
Mètode	POST	POST
URI	/api/newStrategicIndicator	**/api/strategicIndicators**
Paràmetres	-	**-**
Cos	name description network quality_factors	name description network quality_factors
Respostes	Correcte	
	202 Accepted	**201 Created**
	Error en guardar indicador estratègic	
	405 Method Not Allowed	**500 Internal Server Error**

Taula 17. Crear indicador estratègic

La funcionalitat que es tracta a la taula anterior permet crear un nou indicador estratègic. Tal i com es pot veure, el mètode HTTP de la petició es manté sense canvis, ja que el mètode adequat per crear un nou recurs és el POST. En canvi, la URI sí que ha de ser modificada, ja que en l'estat inicial la nomenclatura utilitzada fa referència a l'acció que es realitza sobre el recurs, enlloc del recurs únicament. Cal recordar que l'acció es defineix per mitjà de la combinació del mètode HTTP i la URI, i per tant cal canviar-la pel nom del recurs tal i com es pot veure a la taula.

Els paràmetres i el cos de la petició es mantenen sense canvis, tot i que les respostes sí que cal modificar-les. El codi HTTP per transmetre que la creació d'un nou recurs s'ha executat correctament és el 201, i per indicar que hi ha hagut un problema intern en l'execució de la funcionalitat cal utilitzar el codi 500, acompanyat d'un missatge adient.

8.2. Separació dels controladors REST i el domini

Tal i com s'ha exposat en apartats anteriors, els controladors REST del dashboard tenen un fort component de lògica dins del seu comportament actual, cosa que fa que el seu nombre de responsabilitats així com el seu acoblament augmentin i que siguin complicats de mantenir. Per veure-ho, es mostra com a exemple el diagrama de seqüència de la funció que s'encarrega d'afegir un nou requisit de qualitat associat a una alerta a partir d'una crida REST (Figura 22).

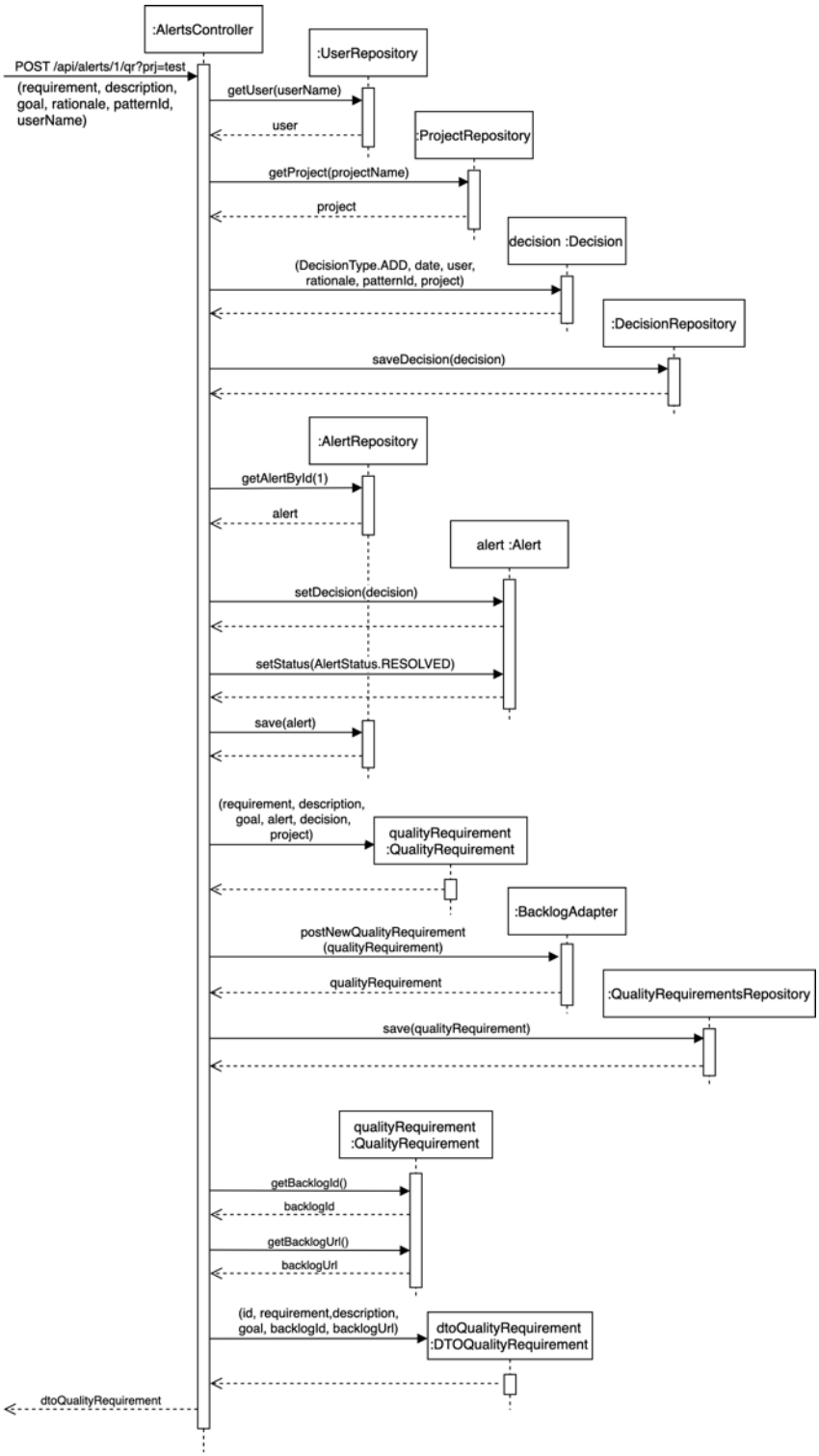

Figura 22. Nou requisit de qualitat - Disseny actual

Com es pot observar, el controlador *AlertController* és l'encarregat de recollir els paràmetres i el cos de la crida REST, interactuar amb diversos repositoris i entitats del domini per tal de dur a terme la funcionalitat i posteriorment construir i retornar un nou objecte DTO amb el resultat.

Per millorar aquesta situació, cal separar i moure la lògica cap a un controlador addicional que s'encarregui únicament d'executar funcionalitats del domini. D'aquesta manera s'aconsegueix reduir les funcions del controlador REST, de manera que ara passarà a encarregar-se de recollir els paràmetres i el cos de les crides externes, invocar la operació corresponent del controlador de domini i construir i retornar un DTO a partir del resultat obtingut. Amb aquest enfocament, el controlador REST passa a formar part de la capa de presentació i s'aconsegueix seguir el patró MVC, on la vista és el DTO que cal construir i enviar. Per tal de preservar la separació entre capes, és important que els DTOs siguin objectes que es creïn a partir de tipus primitius de dades i que no interactuïn ni tinguin cap coneixement dels objectes del domini del sistema. Aquest nou disseny queda plasmat en el següent diagrama de seqüència (Figura 23), on s'apliquen les mesures esmentades fins ara.

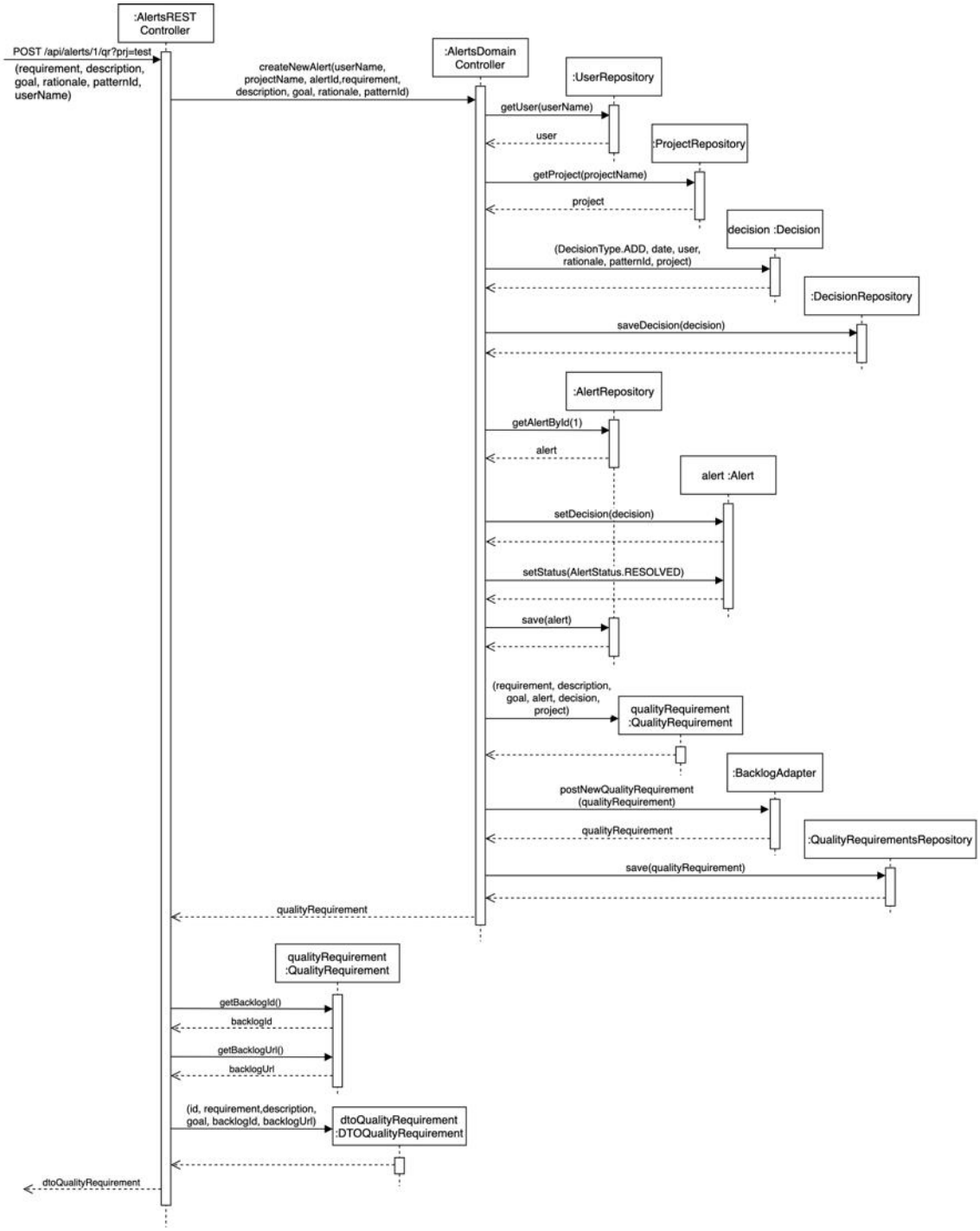

Figura 23. Nou requisit de qualitat - Disseny millorat

El disseny mostrat en aquest apartat fa referència a una funcionalitat concreta dins del *dashboard*, mitjançant el qual s'aconsegueix la separació entre els controladors REST i la lògica del domini per mitjà de la incorporació de controladors del domini. Degut a l'estructura de l'aplicació, aquest mateix disseny es pot aplicar a tots els controladors REST actuals per tal d'aconseguir el mateix objectiu, obviant els detalls propis de cada funcionalitat. Per tant, no es mostren de forma explícita diagrames de seqüència de cadascuna de les funcionalitats modificades, tot i que aquest mateix disseny és el que s'aplicarà a la fase d'implementació per tots els controladors REST, creant els corresponents controladors de domini i traslladant la lògica cap a ells.

La llista de controladors de domini que emergeixen a partir d'aquest canvi es pot veure a la Taula 18. Aquest llista s'obté a partir de l'anàlisi de les funcionalitats del *dashboard* que actualment ofereixen els controladors REST, agrupant-les en funció del model de domini amb els quals treballen per tal de definir quins d'aquests models han de tenir controladors que s'encarreguin d'executar funcionalitats sobre ells. Cadascun d'ells comptarà amb els mètodes necessaris per dur a terme les funcionalitats sobre les entitats del model associades a cada controlador.

Controladors de domini
AlertsController
DecisionsController
FeedbackController
MetricsController
ProductsController
ProjectsController
QRPatternsController
QualityFactorsController
QualityRequirementsController
StrategicIndicatorsController
UsersController

Taula 18. Llistat dels controladors de domini del dashboard

8.3. Simplificació dels repositoris

Els darrers elements pels quals es mostra el disseny de la solució proposada són els repositoris. En l'estat actual, tal i com s'ha mencionat anteriorment, s'utilitzen diverses interfícies sense un criteri clar per tal de definir-los, a més d'implementar manualment algunes de les operacions quan no és necessari fer-ho. En la figura següent (Figura 24) es pot veure quin és el disseny actual, utilitzant el repositori encarregat de gestionar les alertes com a exemple:

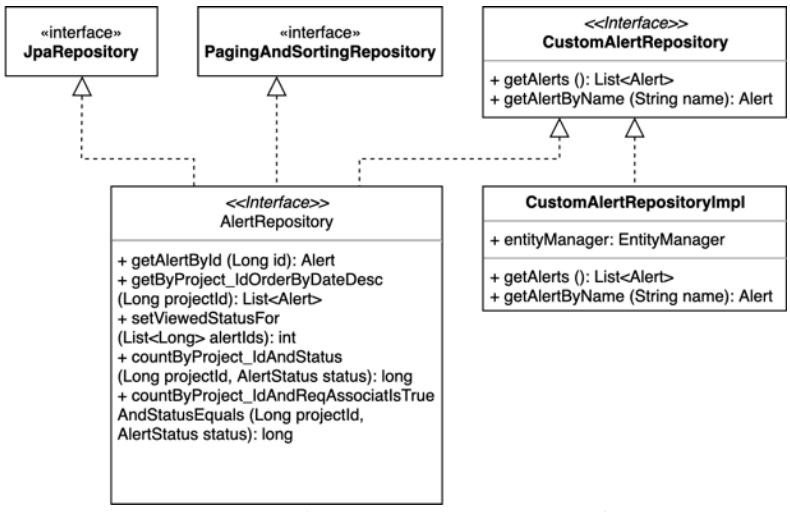

Figura 24. AlertsRepository - Disseny actual

En aquest cas concret, no s'aprofiten les característiques que introdueix la interfície *PagingAndSortingRepository*, i els mètodes que es defineixen a la interfície *CustomAlertRepository* i que s'implementen manualment a *CustomAlertRepositoryImpl* són innecessaris, ja que poden ser implementats automàticament a partir de la

signatura de la mateixa manera que es fa amb tots els mètodes definits a *AlertRepository*.

Seguint aquest raonament, el disseny final (Figura 25) del repositori prescindirà de *CustomAlertRepository* i de *CustomAlertRepositoryImpl*, així com de l'extensió que es fa de *PagingAndSortingRepository*. Els mètodes presents a CustomAlertRepository es definiran ara a *AlertRepository* seguint les mateixes signatures, que ja proporcionen la informació necessària per tal que el mètodes s'implementin automàticament. Pel que fa a la interfície base de la qual cal estendre, en aquest cas cal utilitzar *JpaRepository* degut a que un dels mètodes presents (*setViewedStatusFor (List<Long> alertIds): int*) necessita que la operació sigui escrita en llenguatge SQL per mitjà d'una anotació. En altres casos on això no sigui necessari, es pot utilitzar la interfície *CrudRepository* com a interfície base.

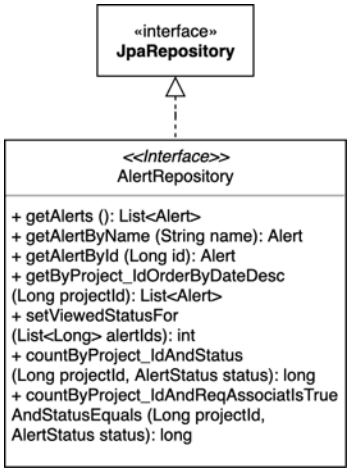

Figura 25. AlertRepository - Disseny millorat

De nou, en aquest apartat es fa referència al disseny d'un element concret del *dashboard*, en aquest cas el repositori d'alertes. De la mateixa manera que a l'apartat anterior, si s'abstrauen els detalls propis d'aquest cas particular, el disseny aplicat pot ser utilitzat en tots els altres repositoris del sistema. D'aquesta manera no és necessari crear diagrames per cadascun d'ells, i els mateixos principis aplicats en aquest disseny poden ser aplicats a la resta de repositoris per tal d'obtenir un sistema on la capa de dades amb les implementacions dels repositoris desaparegui, utilitzant les interfícies base JpaRepository o CrudRepository en funció de les necessitats de cada repositori.

A la Taula 19 es poden veure tots els repositoris que formen part del dashboard, i quina és la interfície base que han d'estendre per tal de poder implementar automàticament el comportament desitjat per cadascun dels mètodes que contenen, sense necessitat de disposar d'una implementació manual.

Repositori	**Interfície base**
AlertRepository	JpaRepository
UserRepository	CrudRepository
DecisionRepository	CrudRepository
FeedbackRepository	CrudRepository
FeedbackValueRepository	CrudRepository
MetricCategoryRepository	CrudRepository

ProductRepository	CrudRepository
ProjectRepository	CrudRepository
QFCategoryRepository	CrudRepository
QRRepository	JpaRepository
QuestionRepository	CrudRepository
RouteRepository	CrudRepository
SICategoryRepository	CrudRepository
StrategicIndicatorRepository	CrudRepository
UserGroupRepository	JpaRepository

Taula 19. Repositoris del sistema amb les seves interfícies base

8.4. Arquitectura resultant del nou disseny del dashboard

Un cop vistos tots els canvis dissenyats que afecten a l'arquitectura, és moment de veure quines implicacions tenen el disseny complet del sistema. A la Figura 26 es mostra novament el diagrama inicial de l'arquitectura del dashboard, i a la Figura 27 es pot apreciar el diagrama resultant de l'arquitectura un cop aplicats els canvis i millores del procés de reenginyeria.

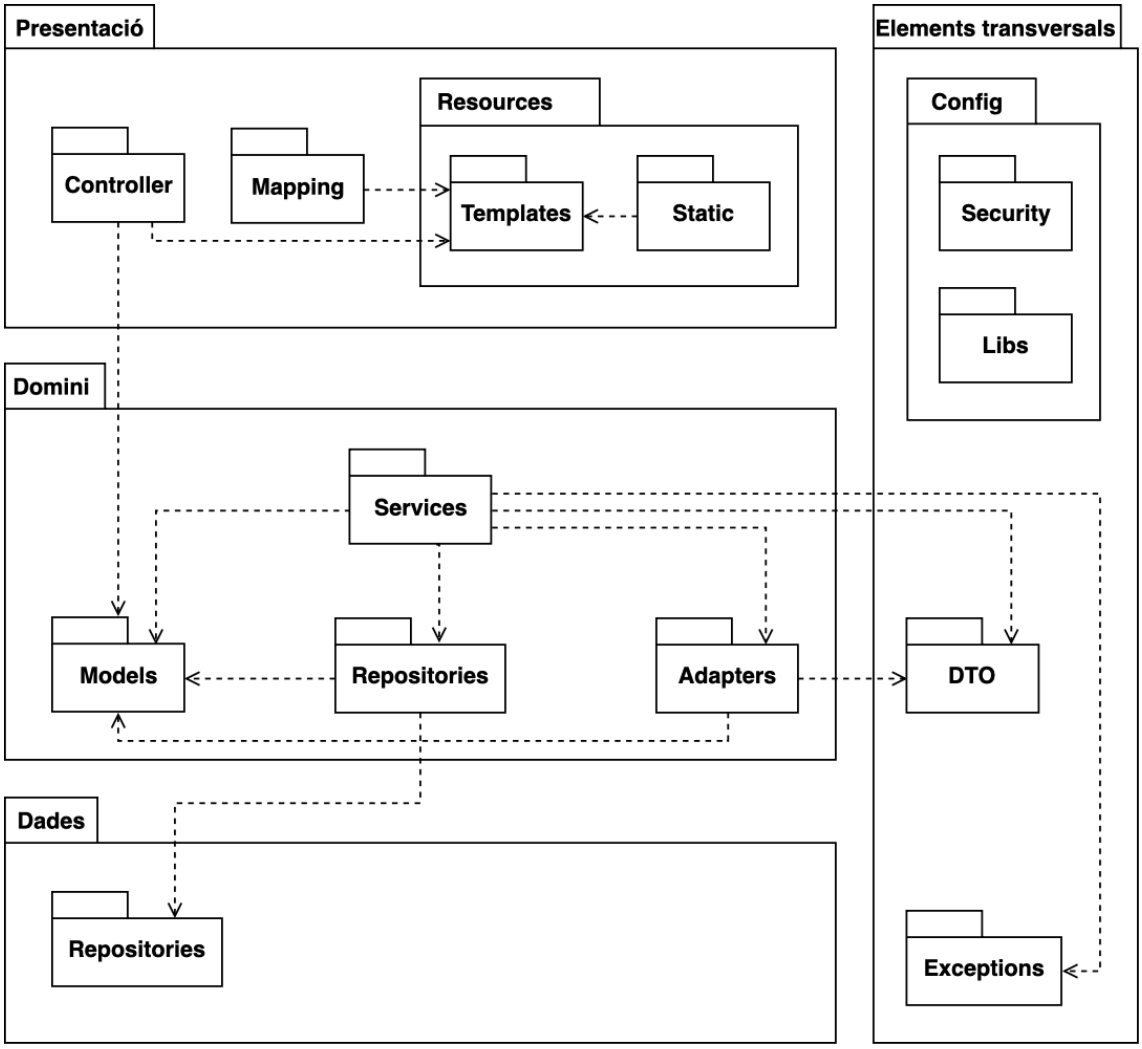

Figura 26. Diagrama inicial de l'arquitectura del dashboard

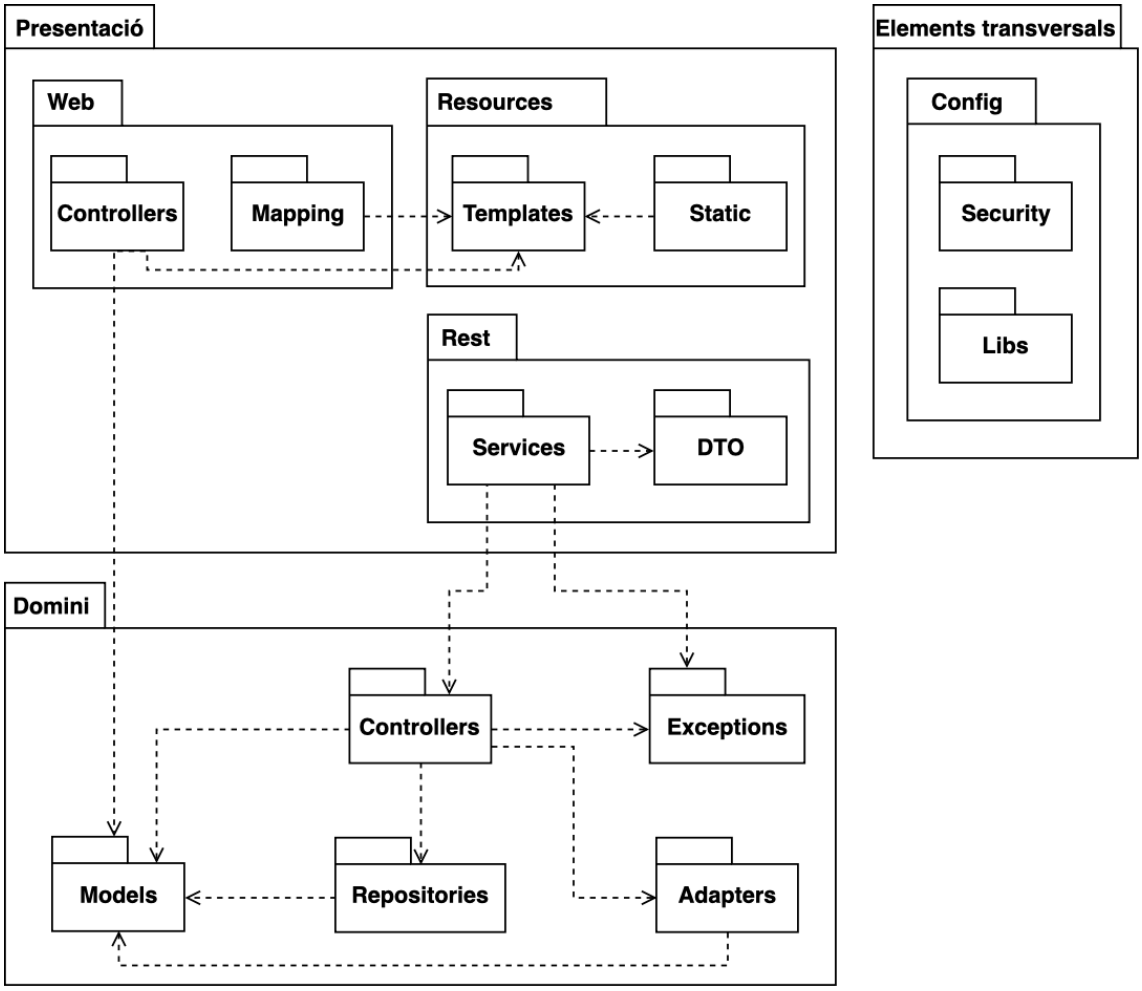

Figura 27. Arquitectura resultant del nou disseny del dashboard

Per començar, el canvi més evident respecte a l'arquitectura inicial té a veure amb la capa de dades. Tal i com s'ha comentat a l'apartat 6.5. Estructura complexa i implementació manual dels repositoris on es descriuen les possibilitats que ofereixen les interfícies que proporciona Spring, la responsabilitat de la implementació dels repositoris per accedir a la base de dades passa a ser del propi *framework*, i per tant no constitueixen elements propis del sistema. Per aquest motiu no es representa aquesta capa al diagrama, i tot i que la implementació dels repositoris existeix, ja que no són elements que s'hagin de mantenir manualment.

El segon canvi que es pot apreciar és la separació entre els serveis REST i la lògica del domini. Tant els serveis com els DTOs passen a formar part de la capa de presentació, sota el paraigua del paquet anomenat Rest. L'acoblament dels serveis amb la resta de d'elements de la capa de domini s'ha reduït gràcies a l'aparició dels controladors de domini, que són els elements amb els que els serveis interaccionen per tal de dur a terme les funcionalitats pròpies del sistema. Aquests controladors només tracten amb elements del domini, i comuniquen els resultats cap als serveis REST.

Per últim, les excepcions també es mouen des de la zona d'elements transversals fins a la capa de domini, reduint els components marcats com a elements transversals.

9. Implementació

En aquest capítol, es tractaran els aspectes tècnics de la implementació dels diferents canvis i millores dutes a terme durant el procés de reenginyeria, a més de determinar quin és el millor ordre per realitzar-les elaborant un pla d'execució.

9.1. Pla d'execució

Un cop vistes i analitzades totes les millores que cal aplicar al dashboard, abans de passar a implementar-les és important establir un pla d'execució que marqui l'ordre que cal seguir. Amb això s'aconsegueix maximitzar el benefici del procés de reenginyeria per mitjà d'una planificació que prioritzi aquells elements que tenen un major impacte en el sistema, i que a la vegada permeti minimitzar el risc i agilitzar la introducció de les millores.

El raonament seguit per planificar el procés és el següent: en primer lloc, es posaran en funcionament i es configuraran les aplicacions externes que permetran optimitzar el procés de desenvolupament, cosa que permetrà aprofitar els seus beneficis durant el procés de reenginyeria. A més, les funcionalitats que es desenvolupin de forma paral·lela a partir de la introducció d'aquest canvi també podran beneficiar-se d'aquestes eines.

A continuació, es passarà a realitzar aquelles tasques que requereixen modificar el codi del sistema. Tot i això, abans d'establir el seu ordre, cal parlar de les proves automatitzades. Si bé és cert que la falta de proves automatitzades es considera com un problema en si mateix, durant la fase d'implementació no es destinarà una tasca exclusivament a aquest propòsit. Per contra, l'enfocament que se seguirà consisteix en integrar aquestes proves dins de les altres tasques, de manera que siguin un prerequisit necessari per tal de dur-les a terme. D'aquesta manera s'aprofita el fet que les àrees del sistema on es vol introduir proves automatitzades són les mateixes que cal modificar per altres propòsits, i es defineix un cicle de treball que comença amb la realització de les proves, la implementació de la pròpia millora i la posterior actualització de les proves automatitzades per reflectir aquests canvis.

Per dur a terme els canvis i millores, es partirà d'aquells elements que pertanyin a un nivell d'abstracció més alt, començant pel disseny dels serveis REST i la seva respectiva documentació, seguint amb la simplificació dels repositoris i acabant amb la separació dels controladors REST i el domini. L'últim canvi a aplicar dins del procés de reenginyeria serà la millora de l'estat general del codi, corresponent a aquells elements amb un nivell d'abstracció més baix. A la Taula 20 es pot veure el resum del pla d'execució, amb les tasques ordenades tal i com s'ha descrit en aquest apartat.

Ordre dels canvis i millores
Optimització del procés de desenvolupament
Millora en el disseny i documentació dels serveis REST
Simplificació dels repositoris
Separació dels controladors REST i el domini
Millora de l'estat general del codi

Taula 20. Ordre dels canvis i millores

9.2. Descripció de les tasques d'implementació

A continuació es descriu amb detall les accions que s'han dut a terme per implementar cadascuna de les fases anteriors, i s'aporten exemples que les il·lustren.

9.2.1. Optimització del procés de desenvolupament

El primer pas que s'ha realitzat en aquest procés de reenginyeria és la optimització del procés de desenvolupament per mitjà de la integració de les eines Travis CI i SonarCloud dins del repositori de GitHub on es troba el codi del *dashboard*. Aquesta integració resulta molt simple gràcies al fet de no haver d'instal·lar ni mantenir cap instància d'aquestes eines, ja que es s'allotgen al núvol i s'ofereixen com a serveis.

Per tal de dur a terme aquesta integració i tenir les dues eines en funcionament, el primer pas ha consistit en accedir a Travis CI i SonarCloud amb l'usuari de GitHub i seleccionar el repositori del *dashboard* per tal de crear un projecte dins de les dues eines. Un cop fet això, ha calgut afegir dos fitxers anomenats *.travis.yml* i *sonar-project.properties* dins del repositori de codi que serveixen com a arxius de configuració per Travis CI i SonarCloud, respectivament.

L'arxiu *.travis.yml* es mostra a la Figura 28, i conté informació sobre el llenguatge en el que està desenvolupat el projecte, els complements que són necessaris per tal d'integrar altres eines (en aquest cas SonarCloud) i un conjunt de comandes que s'executaran cada cop que es faci una *build* per tal de dur a terme les tasques necessàries. En aquest cas, el que es vol fer és executar les proves automatitzades, generar un informe sobre el *coverage* d'aquestes i invocar a l'eina que s'encarrega de dur a terme l'anàlisi de SonarCloud.

```
language: java

addons:
  sonarcloud:
    organization: q-rapids

script:
  - ./gradlew test
  - ./gradlew jacocoTestReport
  - sonar-scanner
```

Figura 28. Arxiu .travis.yml

Per altra banda, l'arxiu *sonar-project.properties* mostrat a la Figura 29 conté elements que descriuen el projecte, com ara el nom i la versió, les rutes en les que es troben els diferents elements que cal analitzar i l'informe de *coverage*, entre d'altres.

```
sonar.projectKey=q-rapids_qrapids-dashboard

sonar.projectName=qrapids-dashboard
sonar.projectVersion=0.3.6

sonar.sources=src/main/java
sonar.tests=src/test/java
sonar.language=java
sonar.java.binaries=build/classes

sonar.sourceEncoding=UTF-8

sonar.coverage.jacoco.xmlReportPaths=build/reports/jacoco/test/jacocoTestReport.xml
```

Figura 29. Arxiu sonar-project.properties

9.2.2. Millora en el disseny i documentació dels serveis REST

Un cop millorat el procés de desenvolupament, la primera etapa que ha requerit modificar el codi del dashboard ha estat la millora en el disseny i la documentació dels serveis REST. La *Pull Request* amb el *commits* realitzats per dur a terme aquestes tasques es pot consultar en aquest enllaç: https://github.com/q-rapids/qrapids-dashboard/pull/77.

Dins d'aquesta etapa, s'ha començat per introduir les proves automatitzades d'integració encarregades de comprovar el correcte funcionament dels serveis REST en l'estat inicial. Aquestes proves s'encarreguen de realitzar peticions HTTP als serveis REST i comprovar la correctesa de la resposta que aquests retornen, tant del cos de la resposta com del codi HTTP. L'estructura de cadascuna d'aquestes proves, així com de totes les altres proves automatitzades realitzades en aquest procés de reenginyeria, segueix el patró de les 4 fases, explicades a continuació:

- Configuració (*setup*): inicialització de tot allò necessari per poder comprovar el comportament del sistema sota prova, tant aspectes de configuració de l'entorn com creació dels objectes amb els quals el sistema sota prova interactua.
- Exercici (*exercise*): execució del sistema sota prova.
- Verificació (*verify*): realització de les comprovacions necessàries per determinar si s'ha obtingut el resultat esperat.
- Desmuntatge (*teardown*): restabliment del sistema al seu estat inicial per mitjà de desfer tot allò realitzat a l'etapa de configuració.

En la fase de configuració, s'inicialitza la infraestructura necessària per dur a terme les peticions HTTP als serveis REST, i per cadascun dels casos de prova s'inicialitzen els objectes necessaris i es defineix el comportament de les dependències del sistema mitjançant la creació de *mocks* amb l'eina Mockito. En la fase d'exercici, es duu a terme la petició HTTP, incloent el mètode, la URI i els paràmetres necessaris. Durant la fase de validació es comproven les dades retornades pel servei REST, així com les diferents interaccions que els *mocks* han tingut amb el sistema. Per últim, a la fase de desmuntatge s'eliminen tots els objectes i la configuració creada i es torna a l'estat inicial per tal de poder començar de nou amb una altra prova.

A la Figura 30 es pot veure el codi d'una de les proves automatitzades d'integració per una de les funcionalitats dels serveis REST, en concret, la que s'encarrega d'afegir un nou requisit de qualitat associat a una alerta. Les fases de configuració, exercici i verificació es troben dins d'aquesta prova i es poden veure comentades al codi, tot i que la fase de desmuntatge es realitza automàticament per Spring i no es mostra en aquest fragment.

```
@Test
public void newQRFromAlert() throws Exception {
    // Setup
    Project project = domainObjectsBuilder.buildProject();
    when(projectsDomainController.findProjectByExternalId(project.getExternalId()))
           .thenReturn(project);

    Alert alert = domainObjectsBuilder.buildAlert(project);
    when(alertsDomainController.getAlertById(alert.getId())).thenReturn(alert);

    Decision decision = domainObjectsBuilder.buildDecision(project, DecisionType.ADD);
    QualityRequirement qualityRequirement= domainObjectsBuilder.buildQualityRequirement(
           alert,
           decision,
           project);
    int patternId = 100;
    when(qualityRequirementDomainController.addQualityRequirementForAlert(
           qualityRequirement.getRequirement(),
            qualityRequirement.getDescription(),
            qualityRequirement.getGoal(),
            decision.getRationale(),
            patternId, alert,
            null,
            project)).thenReturn(qualityRequirement);

    // Exercise
    RequestBuilder requestBuilder = RestDocumentationRequestBuilders
            .post("/api/alerts/{id}/qr", alert.getId())
            .param("prj", project.getExternalId())
            .param("rationale", decision.getRationale())
            .param("patternId", String.valueOf(patternId))
            .param("requirement", qualityRequirement.getRequirement())
            .param("description", qualityRequirement.getDescription())
            .param("goal", qualityRequirement.getGoal());

    this.mockMvc.perform(requestBuilder)
            // Verify
            .andExpect(status().isCreated())
            .andExpect(jsonPath("$.id", is(qualityRequirement.getId().intValue())))
            .andExpect(jsonPath("$.date", is(decision.getDate().getTime())))
            .andExpect(jsonPath("$.requirement",
                   is(qualityRequirement.getRequirement())))
            .andExpect(jsonPath("$.description",
                   is(qualityRequirement.getDescription())))
            .andExpect(jsonPath("$.goal", is(qualityRequirement.getGoal())))
            .andExpect(jsonPath("$.backlogId", is(qualityRequirement.getBacklogId())))
            .andExpect(jsonPath("$.backlogUrl", is(qualityRequirement.getBacklogUrl())))
            .andExpect(jsonPath("$.backlogProjectId", is(nullValue())))
            .andExpect(jsonPath("$.alert", is(nullValue())));

    verify(projectsDomainController, times(1)).
            findProjectByExternalId(project.getExternalId());
    verifyNoMoreInteractions(projectsDomainController);

    verify(alertsDomainController, times(1)).getAlertById(alert.getId());
    verifyNoMoreInteractions(alertsDomainController);

    verify(qualityRequirementDomainController, times(1)).addQualityRequirementForAlert(
            qualityRequirement.getRequirement(),
            qualityRequirement.getDescription(),
            qualityRequirement.getGoal(),
            decision.getRationale(),
            patternId, alert,
            null,
            project);
    verifyNoMoreInteractions(qualityRequirementDomainController);
}
```

Figura 30. Prova automatitzada d'integració d'una funcionalitat dels serveis REST

Un cop el sistema ja compta amb proves d'integració per l'estat inicial dels serveis REST, s'ha procedit a modificar els recursos dels serveis REST adaptant-los a les bones pràctiques i principis de disseny explicats a l'apartat 6.3.3. Principis i bones pràctiques.

Aquestes modificacions han consistit, per cada recurs, en l'actualització del mètode HTTP, la URI i el codi de resposta HTTP per defecte definits en anotacions de Spring situades sobre el mètode que actua com a controlador. A més, també s'han introduït codis de resposta i missatges pels casos d'error, capturant les excepcions que es poden produir dins del controlador i retornant un codi i missatge adequat per cada cas. Tots aquests passos s'han dut a terme seguint allò dissenyat a l'apartat 8.1. Disseny dels serveis REST. Per cada recurs modificat, i abans de passar al següent, s'ha procedit a actualitzar les proves d'integració corresponents per tal que reflectissin els canvis realitzats i es pogués comprovar que el funcionament seguia sent el correcte.

A la Figura 31 es poden observar les anotacions que defineixen el mètode POST, la URI del recurs amb l'identificador parametritzat i el codi HTTP de resposta per defecte. També es poden observar les diferents excepcions que es capturen, i les respostes que es retornen en cada cas, constituïdes per un codi HTTP i un missatge que es troben situats en un arxiu de constants per tal de poder-los reutilitzar. El cos del mètode no es mostra, ja que no és rellevant per aquesta explicació.

```
@PostMapping("/api/alerts/{id}/qr")
@ResponseStatus(HttpStatus.CREATED)
public DTOQualityRequirement newQRFromAlert(@PathVariable String id,
                                    @RequestParam(value = "prj") String prj,
                                    HttpServletRequest request,
                                    Authentication authentication) {
    try {

        ...

    } catch (HttpClientErrorException e) {
        logger.error(e.getMessage(), e);
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
            Messages.BACKLOG_ERROR);
    } catch (AlertNotFoundException e) {
        logger.error(e.getMessage(), e);
        throw new ResponseStatusException(HttpStatus.NOT_FOUND,
            Messages.ALERT_NOT_FOUND);
    } catch (ProjectNotFoundException e) {
        logger.error(e.getMessage(), e);
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
            Messages.PROJECT_NOT_FOUND);
    }catch (Exception e) {
        logger.error(e.getMessage(), e);
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
            Messages.INTERNAL_SERVER_ERROR);
    }
}
```

Figura 31. Controlador REST amb el nou disseny implementat

El darrer pas realitzat per cadascuna de les modificacions dels recursos és la documentació. Per dur-la a terme, s'ha utilitzat Spring REST Docs dins de les proves d'integració, per tal de plasmar l'estat real del sistema i generar la documentació de forma automàtica a partir dels resultats de les proves. Per tal de documentar un recurs, el primer que cal fer és afegir codi dins de les proves d'integració encarregat de capturar les diferents parts de la resposta (mètode HTTP, URI, paràmetres de petició, cos de la resposta, ...), i afegir descripcions als elements que en requereixin, com són els paràmetres de petició i els diferents elements de l'entitat retornada en el cos de la resposta.

A la Figura 32 es pot veure de nou un fragment del codi de la prova automatitzada mostrada a la Figura 30, on ara s'afegeix el codi encarregat de documentar el recurs just

després de les verificacions dels resultats, que ara no es mostren. En concret, aquest fragment captura la petició i la resposta, i documenta el paràmetre de la URI, els paràmetres de consulta i tots els camps de la resposta.

```
this.mockMvc.perform(requestBuilder)
        // Verify

        ...

        .andDo(document("alerts/add-qr-from-alert",
              // Capture and format request
               preprocessRequest(prettyPrint()),
              // Capture and format response
               preprocessResponse(prettyPrint()),
              // Document path parameter
               pathParameters(
                       parameterWithName("id")
                               .description("Alert identifier")
               ),
              // Document request parameters
               requestParameters(
                       parameterWithName("prj")
                               .description("Project external identifier"),
                       parameterWithName("rationale")
                               .description("User rationale of the decision"),
                       parameterWithName("patternId")
                               .description("Identifier of the added quality
                                       requirement pattern"),
                       parameterWithName("requirement")
                               .description("Text of the added quality requirement"),
                       parameterWithName("description")
                               .description("Description of the added quality
                                       requirement"),
                       parameterWithName("goal")
                               .description("Goal of the added quality requirement")
               ),
              // Document response fields
               responseFields(
                       fieldWithPath("id")
                               .description("Identifier of the added quality
                                       requirement"),
                       fieldWithPath("date")
                               .description("Quality requirement creation date"),
                       fieldWithPath("requirement")
                               .description("Text of the added quality requirement"),
                       fieldWithPath("description")
                               .description("Description of the added quality
                                       requirement"),
                       fieldWithPath("goal")
                               .description("Goal of the added quality requirement"),
                       fieldWithPath("backlogId")
                               .description("Quality requirement identifier inside the
                                       backlog"),
                       fieldWithPath("backlogUrl")
                               .description("Link to the backlog issue containing the
                                       quality requirement"),
                       fieldWithPath("backlogProjectId")
                               .description("Backlog identifier of the project
                                       containing the quality requirement"),
                       fieldWithPath("alert")
                               .description("Alert object which caused the quality
                                       requirement addition")
               )
        ));
```

Figura 32. Codi per documentar un recurs d'un servei REST

A continuació, cal ajuntar aquests fragments capturats dins de les proves d'integració per tal de donar el format i el contingut desitjat a la documentació. Això s'aconsegueix mitjançant la creació d'un fitxer en format AsciiDoc anomenat *index.adoc*, que conté un llistat de tots els fragments que es volen mostrar agrupats per funcionalitat. Un cop es

disposa de nous fragments, cal modificar aquest arxiu incorporant una nova secció representat una nova funcionalitat i afegir el contingut que es vulgui mostrar, tal i com es mostra a l'exemple de la Figura 33 on s'inclouen els fragments generats prèviament dins de la secció *snippets*. Per últim, per tal de transformar aquest arxiu en un document HTML que es pugui visualitzar a través d'un navegador web, cal executar la comanda *gradle asciidoctor*.

```
=== Add quality requirement for alert

operation::alerts/add-qr-from-alert[snippets='path-parameters,request-parameters,curl-
request,response-fields,http-response']
```

Figura 33. Codi per introduir un nou recurs a la documentació

9.2.3. Simplificació dels repositoris

Per tal de dur a terme la implementació de la simplificació dels repositoris, s'han executat una sèrie de passos de manera incremental, simplificant un repositori cada cop fins a acabar amb la millora de tots els presents al sistema. La *Pull Request* amb el *commits* realitzats per dur a terme aquestes tasques es pot consultar en aquest enllaç: https://github.com/q-rapids/qrapids-dashboard/pull/86.

En primer lloc, abans de procedir a modificar codi dels repositoris, cal realitzar proves d'integració de l'estat actual per tal de prevenir errors en les modificacions que es realitzin. Aquestes proves d'integració consisteixen en executar les funcionalitats del repositori contra una base de dades en memòria (en concret, una instància de la base de dades anomenada H2 [51]) creada específicament per aquest entorn de proves, i que simula la base de dades real amb la que treballa el sistema. L'estructura de les proves comença amb la fase de configuració, on es defineixen i es guarden els objectes del domini dins de la base de dades per tal de disposar d'informació amb la que fer proves.

A continuació, a la fase d'exercici, s'executa la funcionalitat del repositori i es recullen els resultats, si n'hi ha. Durant la fase de validació es comproven que els resultats són els esperats, o bé que l'estat de la base de dades és el correcte si el que s'ha fet és una inserció o modificació, per exemple. Per últim, a la fase de desmuntatge es buida la base de dades per tal que es pugui començar amb una altra prova, acció que es realitza de manera automàtica per Spring. Totes aquestes fases es poden apreciar a l'exemple que es mostra a la Figura 34.

```
@Test
public void findAlertById() {
    // Setup
    Alert alert = new Alert();
    entityManager.persistAndFlush(alert);

    // Exercise
    Optional<Alert> alertFound = alertRepository.findById(alert.getId());

    // Verify
    assertTrue(alertFound.isPresent());
    assertEquals(alert.getId(), alertFound.get().getId());
}
```

Figura 34. Prova automatitzada d'integració per una funcionalitat d'un repositori

Quan el repositori que es vol simplificar ja consta de proves automatitzades, és moment de passar a modificar el seu codi. Per aquest propòsit, tal i com s'ha definit a l'apartat de disseny corresponent a aquesta millora, s'eliminen tant la interfície com la

implementació destinades a elaborar manualment la interacció amb la base de dades, i els seus mètodes passen a formar part de la interfície única que defineix el repositori. Aquests mètodes s'escriuen seguint les directrius de Hibernate per tal que la implementació es pugui realitzar de forma automàtica a partir de la signatura, o bé a partir de les anotacions en cas que sigui necessari.

A més, s'eliminen les interfícies innecessàries que s'estenen en el repositori, establint una única interfície base de la qual cal estendre, tal i com s'ha definit a l'apartat de disseny. Per últim, es modifiquen les proves automatitzades per plasmar els canvis realitzats i per comprovar que les modificacions no han alterat el funcionament original del repositori.

A la Figura 35 es poden observar les diferents signatures dels mètodes que composen el repositori *AlertRepository*. La única interfície que s'estén és *JpaRepository*, tal i com s'ha dissenyat, i la interfície i la implementació innecessàries han desaparegut. A més, les implementacions es realitzen automàticament a partir de les signatures, excepte en el cas del mètode *setViewedStatusFor*, que requereix d'anotacions per descriure la modificació que es vol realitzar.

```
public interface AlertRepository extends JpaRepository<Alert, Long> {

    List<Alert> findAlertByName(String name);

    List<Alert> findByProject_IdOrderByDateDesc(Long projectId);

    @Transactional
    @Modifying(clearAutomatically = true)
    @Query("update Alert a set a.status = 1 where a.status = 0 and a.id in ?1")
    int setViewedStatusFor(List<Long> alertIds);

    long countByProject_IdAndStatus(Long projectId, AlertStatus status);

    long countByProject_IdAndReqAssociatIsTrueAndStatusEquals(Long projectId,
                                                              AlertStatus status);
}
```

Figura 35. Signatures dels mètodes que pertanyen al repositori d'alertes

9.2.4. Separació dels controladors REST i el domini

En aquesta fase de la implementació, l'objectiu és moure la lògica del sistema des dels controladors REST cap a uns controladors de domini, per tal de poder separar responsabilitats i moure els controladors REST a la capa de presentació. Aquest canvi s'ha implementat de manera incremental, modificant d'una en una totes les funcionalitats que ofereix el *dashboard*. La *Pull Request* amb el *commits* realitzats per dur a terme aquestes tasques es pot consultar en aquest enllaç: https://github.com/q-rapids/qrapids-dashboard/pull/87.

A diferència d'en modificacions anteriors, aquest cop les proves automatitzades ja estaven presents al sistema, ja que les proves d'integració que comproven el funcionament dels controladors REST també executen la lògica que ara es vol separar. Per tant, donada una funcionalitat de la qual se'n vol separar el domini del controlador REST, el primer pas consisteix en crear un controlador per la entitat del domini amb la qual es treballa (en cas que no s'hagi creat anteriorment) i definir un mètode que dugui a terme les accions necessàries sobre els objectes de domini. A la Figura 36 es pot veure el codi d'un dels mètodes del controlador d'alertes, que en retorna una a partir del seu

identificador o bé indica que no existeix per mitjà d'una excepció. Un cop fet això, cal modificar el controlador REST per tal que aquest cridi al controlador de domini corresponent, i en reculli els resultats.

```
public Alert getAlertById(long alertId) throws AlertNotFoundException {
    Optional<Alert> alertOptional = alertRepository.findById(alertId);
    if (alertOptional.isPresent()) {
        return alertOptional.get();
    } else {
        throw new AlertNotFoundException();
    }
}
```

Figura 36. Codi d'un mètode del controlador d'alertes

Per últim, cal modificar les proves automatitzades del controlador REST i crear-ne de noves pel controlador de domini. Això requereix moure el codi que s'encarregava de comprovar el funcionament de la lògica cap a les noves proves automatitzades. Ara, les proves d'integració del controlador REST no executen cap funcionalitat de la lògica del sistema, sinó que interactuen amb controladors de domini en forma de *mocks* on el seu comportament es troba predeterminat. Per altra banda, les proves del controlador de domini, que corresponen a la categoria de proves unitàries i que segueixen de nou l'estructura de les 4 fases, contenen la configuració dels objectes necessaris i el comportament dels *mocks* (en cas que en calguin), la execució de la funcionalitat del propi controlador i la verificació dels resultats, acabant amb el restabliment del sistema a l'estat inicial en cas que sigui necessari. A la Figura 37 es pot observar el codi que pertany a una d'aquestes proves.

```
@Test
public void getAlertById() throws AlertNotFoundException {
    // Setup
    Project project = domainObjectsBuilder.buildProject();
    Alert alert = domainObjectsBuilder.buildAlert(project);
    when(alertRepository.findById(alert.getId())).thenReturn(Optional.of(alert));

    // Exercise
    Alert alertFound = alertsController.getAlertById(alert.getId());

    // Verify
    assertEquals(alert, alertFound);
}
```

Figura 37. Prova automatitzada unitària d'un mètode del controlador d'alertes

9.2.5. Millora de l'estat general del codi

Per últim, la darrera fase de la implementació ha consistit en la millora de l'estat general del codi. Per aquest propòsit, s'ha partit dels resultats de l'anàlisi de SonarCloud i s'ha focalitzat l'atenció sobre aquells problemes de severitat més alta, tal i com s'estableix al requisit de qualitat que contempla aquest aspecte. En concret, els problemes resolts han estat els *bugs* i les vulnerabilitats, així com els *code smells* etiquetats amb severitat bloquejant i crítica. La *Pull Request* amb el *commits* realitzats per dur a terme aquestes tasques es pot consultar en aquest enllaç: https://github.com/q-rapids/qrapids-dashboard/pull/91.

L'abast de cadascuna d'aquestes modificacions és d'una dimensió petita en general, però en canvi el nombre de problemes detectats és suficientment gran com per constituir una tasca costosa. Per tal de resoldre'ls, s'han dut a terme les recomanacions que indica SonarCloud per cada problema, i que condueixen el sistema cap a un estat

de qualitat més elevat. A la Figura 38 es pot veure la descripció d'un *code smell* proporcionada per SonarCloud, així com un exemple d'un cas erroni i la seva respectiva solució. També cal destacar que hi ha un nombre elevat de problemes, generalment de severitat menor, que han estat resolts de manera indirecta al dur a terme les altres millores del procés de reenginyeria, i que han contribuït a millorar l'estat general del codi sense necessitat de destinar temps exclusivament a aquesta tasca.

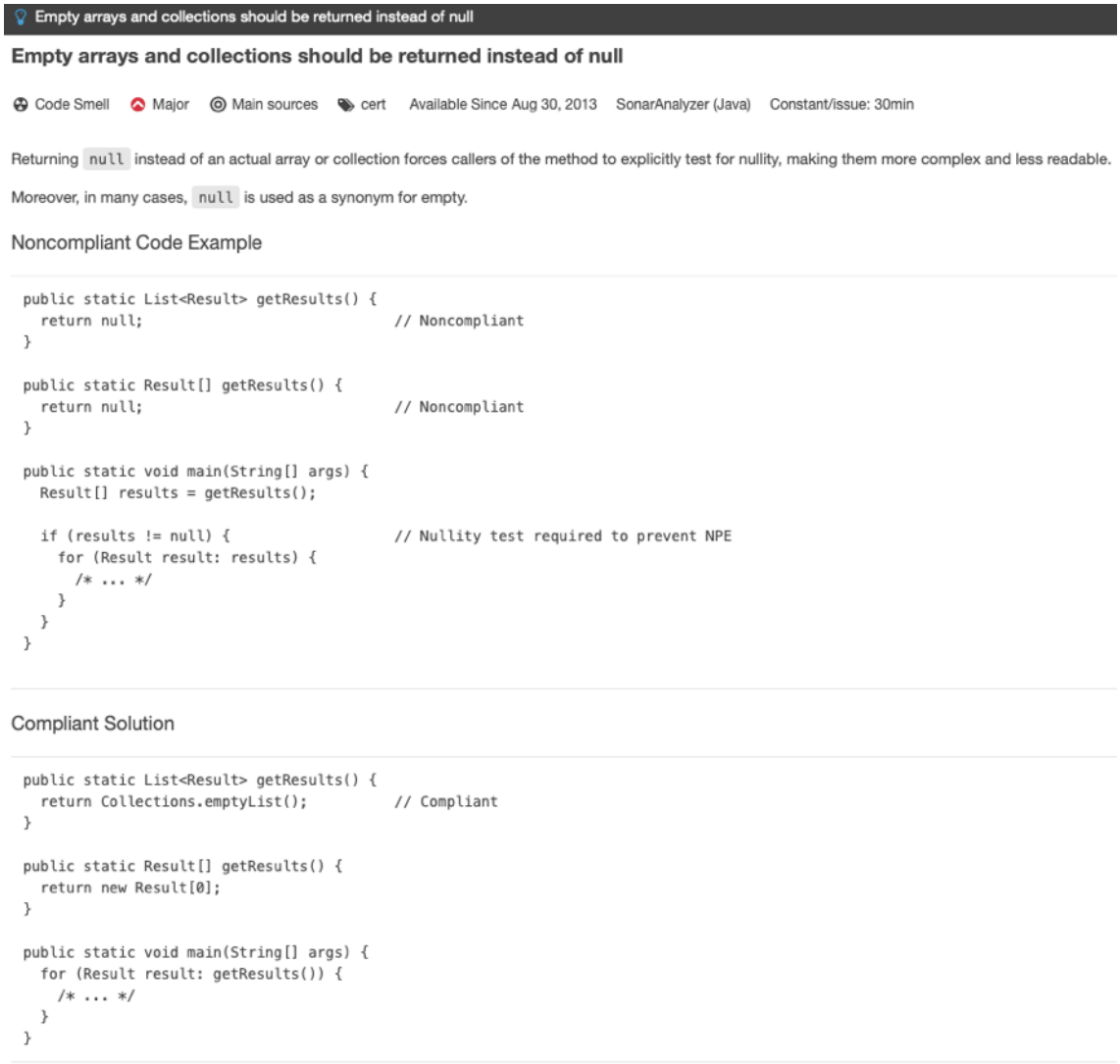

Figura 38. Recomanacions de SonarCloud per resoldre un code smell

10. Validació

En aquest apartat s'exposa el resultat del procés d'implementació, explicant l'abast dels diferents canvis i millores i validant que els requisits de qualitat identificats es satisfacin correctament.

10.1. Canvis i millores implementades

Un cop acabat tot el procés d'implementació dels canvis i millores, cal veure quin és l'estat en el qual es deixa el *dashboard* i si s'han complert o no els objectius marcats a la fase d'anàlisi de requisits. Per començar, cal veure quina feina s'ha pogut dur a terme amb èxit dins de cada fase de l'etapa d'implementació.

10.1.1. Optimització del procés de desenvolupament

S'ha implantat amb èxit l'eina Travis CI dins del procés de desenvolupament, configurada per compilar el projecte i executar les proves automatitzades per cada nou *commit* realitzat al repositori, tant en branques principals com a les *Pull Requests*. A més, s'ha integrat correctament amb SonarCloud per tal d'executar anàlisis i enviar la informació sobre la cobertura de les proves executades. La metodologia de desenvolupament basada en *Pull Requests* s'ha utilitzat tant en el desenvolupament del propi *dashboard* com en les següents millores implementades, mostrant correctament informació sobre les eines integrades amb el projecte.

A la Figura 39 es pot observar la informació que SonarCloud i Travis CI afegeixen a una Pull Request de GitHub, en forma de *checks*. El primer *check* indica que l'anàlisi de SonarCloud s'ha executat correctament i que no hi ha hagut una davallada important en els nivells de qualitat. El segon *check* correspon a l'estat de la *build* de Travis CI resultant d'integrar els canvis introduïts en la nova branca amb la branca destí de la Pull Request, i el tercer *check* mostra l'estat de la *build* corresponent a l'últim *commit* de la branca.

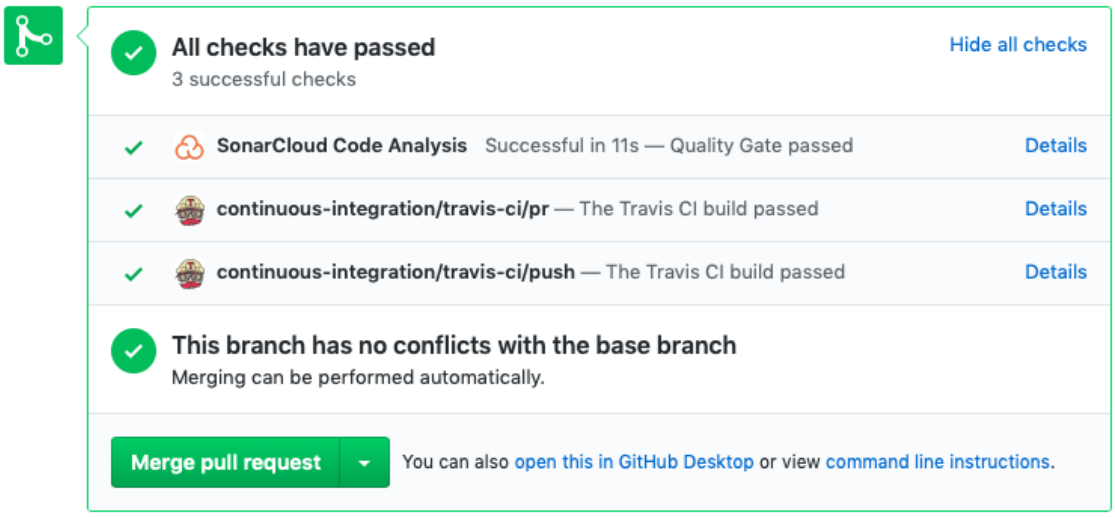

Figura 39. Informació de SonarCloud i Travis CI dins d'una Pull Request de Github

10.1.2. Introducció de proves automatitzades

S'han introduït proves d'integració en els controladors dels serveis REST i en els repositoris que treballen amb la base de dades, i proves unitàries en la major part de la lògica complexa del sistema. En el moment d'acabar la fase d'implementació, es compta amb un total de 249 proves automatitzades. El resultat i el temps d'execució de totes les proves es poden veure a la Figura 40.

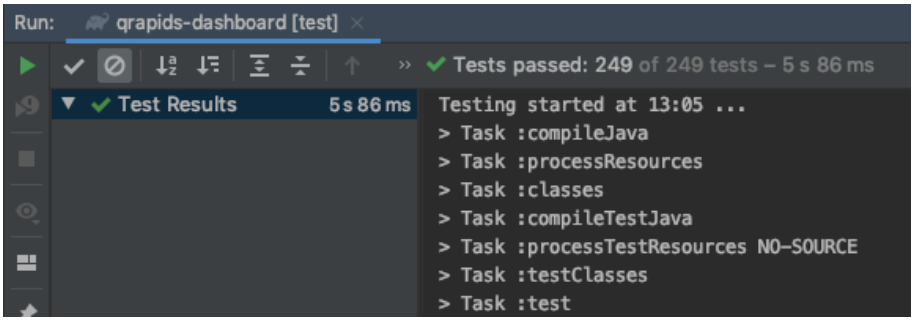

Figura 40. Temps d'execució i nombre de proves automatitzades

10.1.3. Millora en el disseny dels serveis REST

S'han aplicat tots i cadascun dels principis de disseny mencionats a l'anàlisi del problema identificat, utilitzant el patró *Resource API*, modificant les URIs per basar-les en els recursos amb els quals tracta el sistema, utilitzant els mètodes HTTP correctes en cada cas per tal de transmetre intencionalitat i afegint codis HTTP i missatges informatius en els casos d'error.

10.1.4. Documentació dels serveis REST

Aprofitant les proves d'integració creades pels serveis REST, s'ha generat la documentació de tots els serveis utilitzant Spring REST Docs per tal d'aconseguir una documentació que es genera automàticament a partir de les proves automatitzades. Aquesta documentació es desplega automàticament fent servir el servei GitHub Pages, actualitzant el seu contingut cada cop que s'actualitza la branca master. Es pot consultar en el següent enllaç: https://q-rapids.github.io/qrapids-dashboard/ . A la Figura 41 es pot observar quin és l'aspecte d'aquesta pàgina web, on es troba disponible tota la documentació generada per cadascuna de les funcionalitats dels serveis REST, navegable a partir del menú situat a la part esquerra de la pantalla.

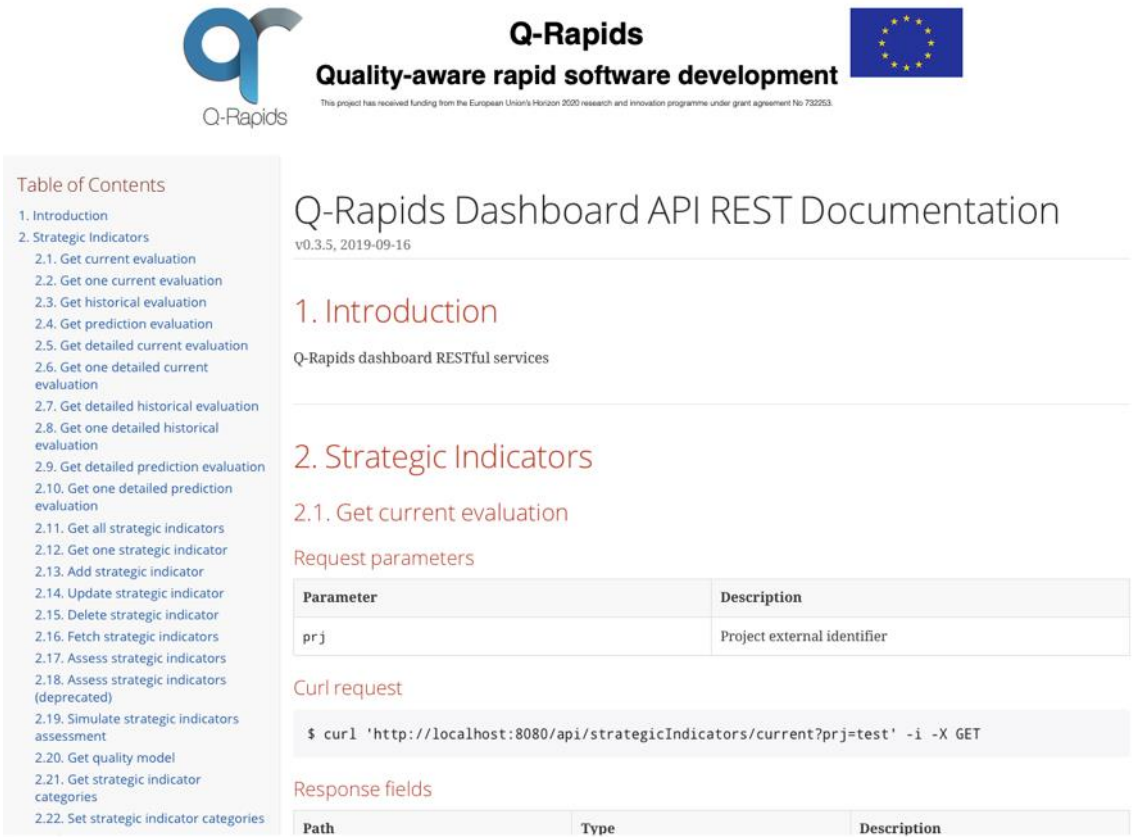

Figura 41. Documentació dels serveis REST

10.1.5. Separació dels controladors REST i el domini

S'han creat controladors del domini que encapsulen totes les funcionalitats que ofereix el dashboard, reduint l'acoblament dels controladors REST amb adaptadors i repositoris. Aquests s'acoblen únicament amb els controladors de domini per delegar tota funcionalitat relacionada amb el domini del *dashboard*. Els controladors REST i els DTOs s'han traslladat a la capa de presentació, ja que ara la seva responsabilitat està molt més delimitada i clara.

Aquestes modificacions es poden apreciar a la Figura 42, on es pot veure com els controladors dels serveis REST i els DTOs passen a formar part de la capa de presentació, i apareix un nou paquet anomenat *controllers* dins de la capa de domini destinat als controladors d'aquesta capa. Aquestes modificacions en l'estructura del *dashboard* coincideixen amb allò establert a l'apartat de Disseny, i plasmat en l'esquema de l'arquitectura resultant del procés de reenginyeria (Figura 27).

Figura 42. Estructura de paquets del dashboard

10.1.6. Simplificació dels repositoris

S'han modificat les interfícies que fan servir tots els repositoris per tal d'utilitzar únicament CrudRepository o JpaRepository en funció de les seves necessitats. També s'han eliminat totes les implementacions dels repositoris, i s'han substituït per signatures de mètodes que són implementats automàticament per Spring. Aquesta darrera modificació es pot apreciar a la Figura 42, on la capa de dades s'ha eliminat de l'estructura de paquets del *dashboard*, deixant lloc únicament al paquet de repositoris constituït per les interfícies dins de la capa de domini.

10.1.7. Millora de l'estat general del codi

S'ha millorat de forma substancial l'estat general del codi, eliminant tots el *bugs* i vulnerabilitats i reduint significativament el nombre de *code smells*. A la Figura 43 es pot veure quin era l'estat de l'anàlisi de SonarCloud abans de començar el procés de millora de qualitat del dashboard, i a la Figura 44 es mostra l'estat final un cop implementades totes les millores.

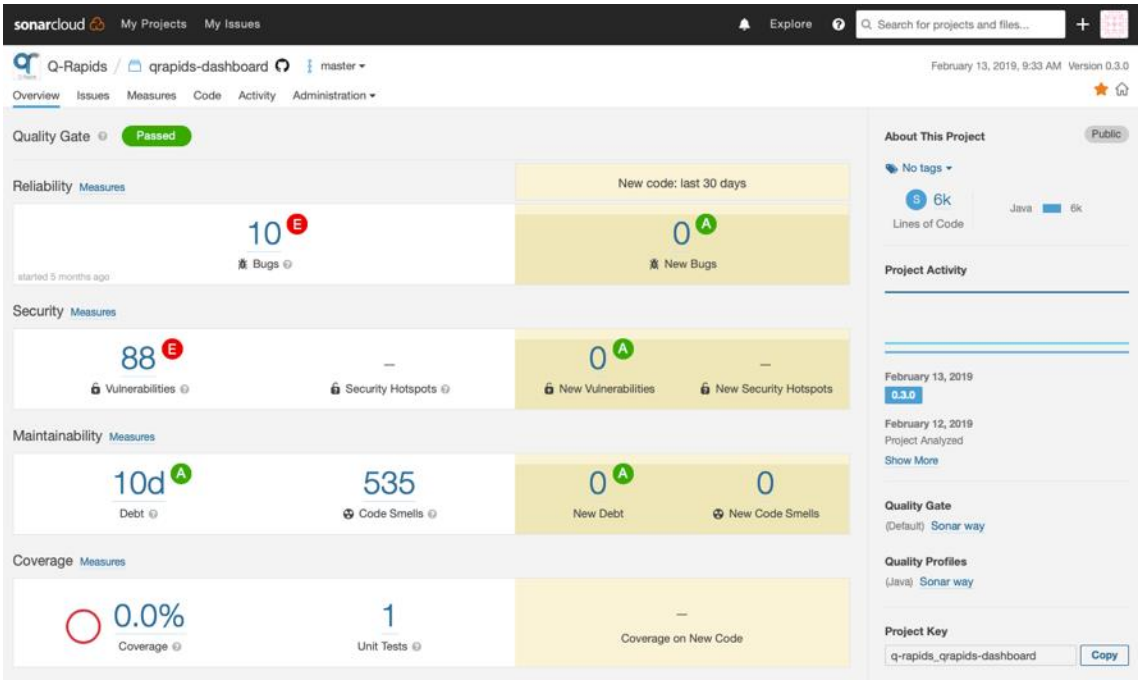

Figura 43. Anàlisi inicial de qualitat a SonarCloud

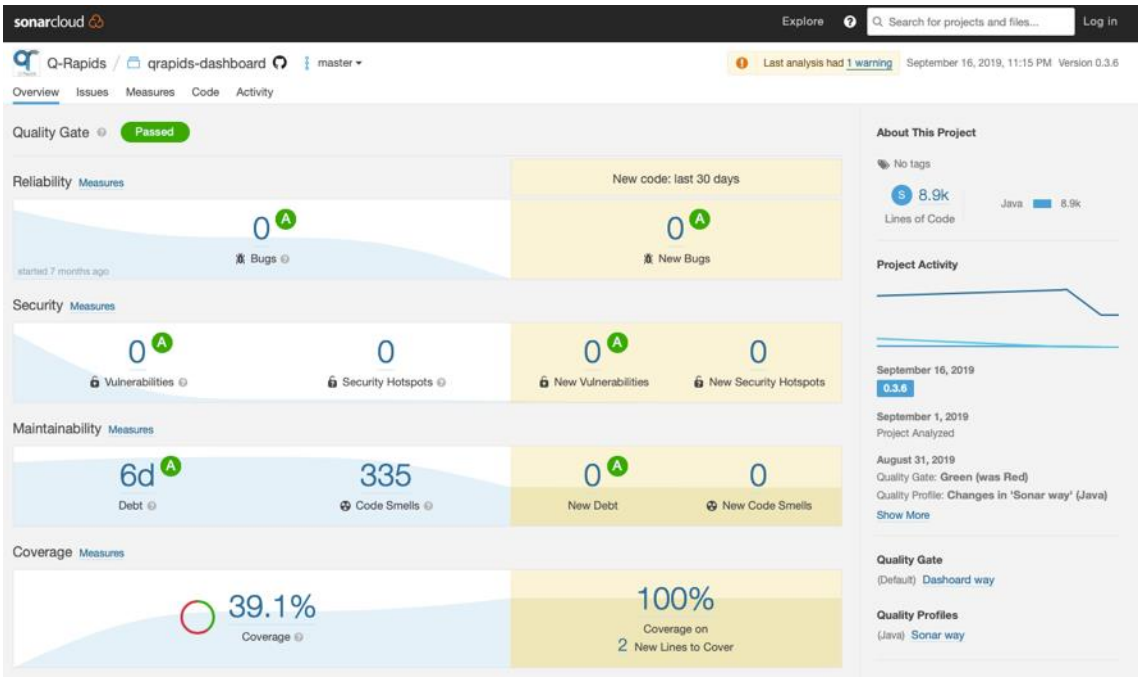

Figura 44. Anàlisi final de qualitat a SonarCloud

Si es posa el focus sobre els *code smells*, es pot observar a la Figura 45 i la Figura 46 quina ha estat la seva evolució, eliminant completament aquells classificats com a problema bloquejant i crític, i reduint significativament el nombre de problemes majors i menors.

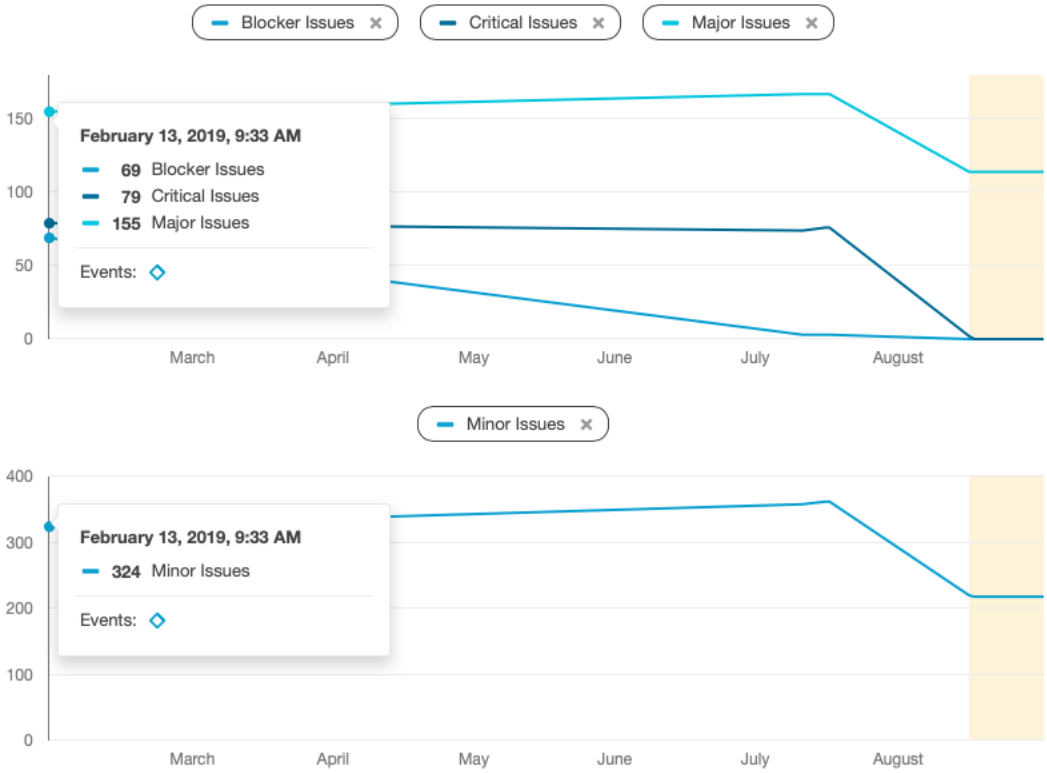

Figura 45. Detall dels code smells inicials

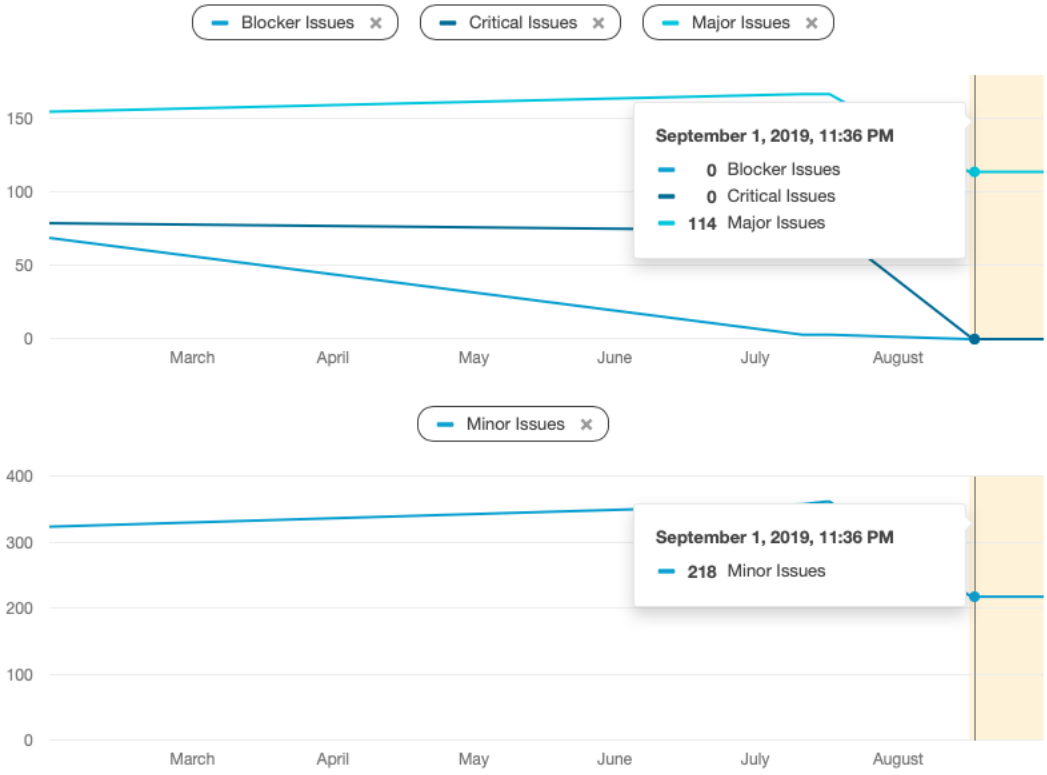

Figura 46. Detall dels code smells finals

10.2. Satisfacció dels requisits de qualitat

Un cop vista la feina feta en les diferents fases de millora del *dashboard*, cal comprovar que els requisits de qualitat definits durant la fase d'anàlisi del projecte es compleixin de forma satisfactòria. Per fer-ho, cal revisar els criteris de satisfacció que es van proposar i avaluar el seu grau de compliment.

10.2.1. Mantenibilitat

El primer requisit de qualitat que es vol aplicar al *dashboard* és la mantenibilitat. Aquests són els criteris de satisfacció que es proposen:

- Per tal d'assegurar la mantenibilitat del *dashboard*, cal reorganitzar i reescriure diverses parts del sistema per tal d'adequar-les a les bones pràctiques i principis del disseny de software i seguir patrons de disseny quan sigui convenient. Els principis i patrons propis del disseny de software han estat àmpliament validats per nombrosos estudis, i la seva utilització garanteix un benefici de cara a la mantenibilitat del sistema.
- En finalitzar el procés de reenginyeria, els problemes detectats per l'eina SonarCloud relacionats amb la mantenibilitat s'hauran de veure reduïts fins arribar a les següents fites: eliminar completament els *bugs*, les vulnerabilitats, i els *code smells* classificats com a severitat bloquejant i crítica. A partir d'aquí, a mesura que el sistema creixi i incorpori noves funcionalitats, els problemes detectats per aquesta eina no hauran de superar el 20% respecte al nombre de línies de codi del sistema.
- Per últim, el sistema haurà de comptar amb proves unitàries allà on la lògica sigui complexa i crítica pel funcionament del sistema i amb proves d'integració pels serveis REST i els repositoris.

Tal i com es pot veure, per tal de satisfer aquest requisit s'han de complir diverses condicions. La primera d'elles és l'aplicació de bones pràctiques, principis i patrons en el disseny del *dashboard*, introduïts al sistema gràcies a les accions realitzades en diversos dels canvis i millores. Per començar, s'han redissenyat per complet els serveis REST per tal d'adaptar-los als principis i patrons d'aquest tipus de sistema, gràcies als quals el propòsit de cada funcionalitat pot ser determinat únicament a partir de la URI, el mètode HTTP i els diferents codis de resposta.

A més, la separació dels controladors REST i la lògica del *dashboard* millora l'arquitectura general del sistema, reduint l'acoblament dels controladors REST i limitant les seves responsabilitats. Els controladors REST deleguen tota funcionalitat cap als controladors del domini, de manera que els primers es poden traslladar a la capa de presentació del sistema. Per últim, la simplificació dels repositoris fa que s'aprofitin al màxim les interfícies que Spring proporciona per aquest tipus de components, eliminant les implementacions innecessàries que poden esdevenir fonts d'errors.

En segon lloc, el criteri de satisfacció fa referència a la millora de la qualitat de l'estat general del codi analitzat per SonarCloud. Tal i com es pot veure a la Figura 44, els *bugs* i vulnerabilitats s'han reduït a 0, i el nombre de *code smells* s'ha reduït en 200. Veient amb més detall els code smells, a la Figura 46 es pot veure com actualment s'han

eliminat tots aquells classificats com a bloquejant i crític, a més de reduir significativament els problemes majors i menors. Pel que fa a la segona part d'aquest criteri d'acceptació, no es pot avaluar en l'estat actual ja que fa referència a una norma que caldrà complir durant el procés de desenvolupament de noves funcionalitats.

Per últim, pel que fa a les proves automatitzades, s'han realitzar correctament proves unitàries en la major part del domini del sistema, així com proves d'integració a tots els serveis REST i tots els repositoris.

Així doncs, com que els tres criteris de satisfacció es compleixen en la seva totalitat, es considera que el requisit de qualitat ha estat aplicat correctament.

10.2.2. Documentació

El següent requisit que es contempla és el de documentació, que consta del següent criteri de satisfacció:

- El *dashboard* estratègic ha de comptar amb documentació tècnica accessible en format web que ha de descriure els diferents serveis REST, mostrant informació sobre les URIs, mètodes HTTP, paràmetres de consulta, format de les respostes i codis HTTP.

Tal i com s'ha mostrat a l'apartat anterior, la documentació dels serveis REST s'ha realitzat correctament tal i com estava previst, per tots els serveis REST que ofereix el *dashboard*. Per cada una de les funcionalitats, la documentació mostra la URI, el mètode HTTP, els paràmetres de consulta i de resposta amb la seva respectiva descripció, el format complet de la resposta satisfactòria i les diferents respostes pels casos d'error, acompanyades del mètode HTTP corresponent.

Així doncs, la documentació compta amb tots els elements necessaris requerits pel criteri de satisfacció, i es troba accessible en format web gràcies a GitHub Pages. A més, gràcies a l'enfocament seguit de documentació basada en proves, s'assegura un manteniment i actualització òptim de la documentació de cara a futures funcionalitats del *dashboard*. Amb tot això, es conclou que el criteri de satisfacció es compleix en la seva totalitat i el requisit de qualitat s'ha aplicat de forma correcta.

10.2.3. Compatibilitat

Per últim, el requisit de qualitat que es tracta és la compatibilitat. El criteri de satisfacció associat és el següent:

- Per satisfer aquest requisit no és suficient amb que el dashboard sigui compatible amb sistemes externs, sinó que la qualitat d'aquesta integració s'ha de maximitzar. Per tant, els serveis REST del dashboard s'hauran d'adequar a les bones pràctiques de disseny utilitzant el patró *Resource API*, utilitzant els mètodes HTTP per transmetre intencionalitat, URIs basades en recursos i missatges d'error clars i entenedors.

Les condicions que s'expressen al criteri de satisfacció consisteixen en integrar una sèrie de bones pràctiques en el disseny dels serveis REST, per tal de maximitzar i facilitar la

compatibilitat amb serveis externs. Tots aquests principis s'apliquen a la fase de disseny d'aquest projecte, on per cada funcionalitat dels serveis REST es mostra quins canvis s'han de produir per tal de tenir una qualitat òptima, seguint les directrius descrites a la fase d'anàlisi dels problemes identificats. Durant la fase d'implementació s'han dut a terme tots i cadascun dels canvis dissenyats, i per tant podem concloure que es compleix el criteri de satisfacció i el requisit de qualitat ha estat aplicat correctament.

11. Estat final i desviacions

En aquest apartat es fa una síntesi de l'estat final del projecte, revisant els compromisos adquirits en les primeres fases del projecte i analitzant les desviacions que s'han produït.

11.1. Resolució d'objectius i abast

Un cop acabada la part tècnica del projecte, cal veure quin és el grau de compliment dels objectius que es van plantejar en un inici. Per tal de recordar quins van ser els objectius establerts, es llisten de nou a continuació:

- Millorar l'arquitectura actual dels components interns del *dashboard* per tal d'adequar el seu disseny als principis i patrons de l'enginyeria del software.
- Millorar el disseny del codi mitjançant l'aplicació de bones pràctiques que redueixin la seva complexitat i en facilitin la modificació.
- Introduir l'automatització de proves per tal de poder detectar errors durant el procés de desenvolupament i reduir el risc associat a la modificació de codi ja existent.
- Utilitzar eines d'anàlisi i automatització que permetin millorar el procés de desenvolupament de software i contribueixin a evitar una futura davallada en el nivell de qualitat del producte.

Per començar, el primer objectiu que es va plantejar és la millora de l'arquitectura actual. Per tal de dur-lo a terme, s'han realitzar diverses accions durant el transcurs d'aquest projecte que contribueixen a assolir aquest objectiu. Per començar, a la fase d'anàlisi de la situació actual es fa una descripció detallada de l'arquitectura inicial del dashboard, per tal de poder determinar quins són els punts febles del sistema. A continuació, a la fase de detecció de problemes, s'analitzen les implicacions negatives de la situació inicial i es proposen alternatives per tal de millorar-la.

En concret, per la millora de l'arquitectura, es detecten i s'analitzen tres problemes amb les seves respectives solucions: la separació dels controladors REST i la lògica del domini, la millora del disseny dels serveis REST i la simplificació dels repositoris. Aquestes millores es dissenyen amb detall a l'apartat de disseny, fins a obtenir una idea precisa de quins hauran de ser els canvis concrets que caldrà aplicar per tal de millorar la situació inicial. Durant la fase d'implementació es duen a terme de manera satisfactòria aquestes modificacions, i tal i com s'ha exposat a l'apartat de Validació contribueixen a dur a terme amb èxit els requisits de qualitat involucrats amb aquests canvis, la mantenibilitat i la compatibilitat.

En segon lloc, es proposa l'objectiu d'aconseguir una millora en el disseny del codi. A diferència del primer objectiu, l'anàlisi de la situació actual i el disseny no contribueixen directament a millorar aquesta situació. Tot i això, l'enfocament per assolir aquest objectiu passa per la utilització de SonarCloud per tal de detectar problemes en la qualitat del codi i poder millorar el seu disseny. Durant la fase d'implementació s'ha dedicat temps exclusivament a resoldre problemes d'aquest tipus, i això queda plasmat a la Validació, on es veu quina ha estat la davallada en els problemes identificats i quin

és el resultat final de l'anàlisi de qualitat del codi, complint amb les condicions marcades pel criteri de satisfacció del requisit de qualitat de mantenibilitat.

El següent objectiu és la introducció de proves automatitzades dins del *dashboard*. Durant la fase de detecció de problemes es considera la falta de proves automatitzades com un risc potencial pel *dashboard* i un problema per la seva mantenibilitat, i es consideren diferents alternatives considerant els diversos tipus de proves automatitzades que es poden implementar i els components que més poden aprofitar els seus beneficis. A la fase d'implementació es duen a terme aquestes proves en els components adequats, complint amb allò requerit per tal d'assolir un dels criteris d'acceptació del requisit de mantenibilitat.

Per últim, es proposa l'objectiu d'utilització d'eines d'anàlisi i automatització dins del procés de desenvolupament del *dashboard*. Un cop vist quin problema representa la seva absència a l'apartat d'anàlisi de problemes, es consideren diferents alternatives tant per eines d'anàlisi de qualitat de codi com per serveis d'integració contínua, fins a acabar seleccionant SonarCloud i Travis CI respectivament. Gràcies a que les opcions escollides són compatibles entre elles i amb GitHub, a la fase d'implementació es configura tota la infraestructura necessària per tal de poder automatitzar la compilació, l'execució de les proves automatitzades i l'anàlisi de la qualitat de manera que aquest procés sigui transparent al desenvolupador i la informació que aquestes eines proporcionen millorin el procés de desenvolupament.

Amb tot això, es pot veure com els objectius establerts inicialment s'assoleixen de forma satisfactòria, gràcies a la feina feta durant els diferents apartats d'aquest projecte.

Pel que fa a l'abast del projecte, durant la fase inicial es va establir que l'àmbit d'actuació d'aquest projecte de reenginyeria seria el servidor del *dashboard*, tal i com s'ha dut a terme. A més, a l'abast també es defineixen tres fases que caldrà realitzar durant aquest procés: l'anàlisi de la situació actual, la identificació dels problemes, anàlisi d'alternatives i proposta de solució i l'aplicació de les millores. Tal i com s'ha pogut veure en aquesta memòria, totes aquestes fases es duen a terme de manera satisfactòria tal i com s'havia previst inicialment.

11.2. Obstacles

Tot i la correcta resolució dels objectius establerts a l'inici del projecte, durant el transcurs del mateix han aparegut obstacles que han comportat un element afegit de dificultat al procés de reenginyeria.

D'entre els obstacles considerats inicialment a l'apartat 2.2.2. Possibles obstacles, el primer d'ells que l'autor d'aquest projecte ha hagut d'afrontar és la coordinació entre el procés de reenginyeria i el desenvolupament. Degut a que els canvis en el procés de reenginyeria són d'un abast gran, han aparegut incompatibilitats entre les modificacions que es volien realitzar i les noves funcionalitats que calia introduir al *dashboard*. Això ha comportat endarreriments en la realització d'algunes d'aquestes modificacions, i s'ha hagut d'estudiar molt bé quan era el millor moment per realitzar-les.

Un altre dels obstacles considerats inicialment i que s'han manifestat durant aquest projecte ha estat el codi en mal estat. La naturalesa d'un projecte de reenginyeria fa que

gran part de l'esforç es dediqui a llegir i entendre el codi existent, i la falta de documentació i el mal estat del codi han dificultat aquest procés. Això ha comportat que la inversió en hores dedicades a comprendre el codi existent hagi estat alta, repercutint també en la planificació temporal.

Per últim, l'obstacle principal d'aquest projecte ha estat la limitació de temps. Les situacions descrites fins ara i la dificultat no prevista d'alguns apartats, com ara l'anàlisi dels problemes, han comportat un augment significatiu de les hores previstes per realitzar aquest projecte. Degut a això es va haver de posposar la data d'entrega del projecte per tal de poder realitzar un procés de reenginyeria que realment tingués un impacte significatiu en el *dashboard*, en comptes de limitar les tasques a realitzar i reduir l'abast.

Afortunadament, els obstacles que han aparegut en aquest projecte són de caràcter temporal, i el fet de poder disposar d'una segona data d'entrega ha fet que es poguessin invertir les hores necessàries per dur a terme totes les tasques. No ha aparegut cap obstacle que hagi impedit la realització de les tasques planificades en un inici, i la dificultat tècnica no ha estat un impediment important.

11.3. Desviacions metodològiques

Durant la fase inicial del projecte, es va proposar fer servir una metodologia en cascada o *waterfall*, que consisteix en l'execució de forma seqüencial i consecutiva de les fases del projecte, disposades seguint una progressió lògica on el resultat de cada fase és el punt de partida de la següent. Aquesta metodologia busca definir amb el màxim detall cadascuna de les etapes abans de passar a la següent, i no admet solapaments entre elles. Està formada per les fases següents: anàlisi de requisits, especificació, disseny, implementació, validació i manteniment.

En particular, per aquest projecte de reenginyeria la fase d'anàlisi de requisits consisteix en determinar exactament quins requisits de qualitat són els que es volen aplicar, i quines millores caldrà introduir per aconseguir-ho. Pel que fa a l'especificació, com que els requisits que es volen aplicar són de tipus no funcional i no es vol modificar cap de les funcionalitats del sistema, no es produeixen canvis en aquesta fase. A continuació, la fase de disseny consisteix en dissenyar una arquitectura que integri aquests nous requisits i proporcioni una solució pels problemes identificats. Durant la fase d'implementació es duen a terme totes les millores identificades i dissenyades, i a la fase de validació es comprovarà que els canvis introduïts satisfan els requeriments de qualitat inicials. La fase de manteniment queda fora de l'abast del projecte.

Les fases i el contingut d'aquesta metodologia s'ha mantingut sense canvis des de la fase inicial del projecte, i ha demostrat ser d'utilitat per dur a terme amb èxit les diverses tasques necessàries dins del procés de reenginyeria. Contràriament a les metodologies àgils, el que es busca amb una metodologia en cascada és reduir al màxim la necessitat d'introduir canvis un cop el projecte ha començat. El fet de treballar amb requisits de qualitat i no pas amb requisits funcionals fa que els canvis que es volen introduir siguin d'un abast i envergadura gran, cosa que fa que eventuals canvis i modificacions durant l'execució del projecte siguin arriscats i costosos per la integritat del mateix.

Tal i com s'ha pogut comprovar, la feina feta durant les fases d'anàlisi de requisits i disseny ha proporcionat una base sòlida per dur a terme el projecte, i ha permès tenir una visió clara i detallada tant de l'estat actual del dashboard com dels canvis i millores que cal implementar, de manera que la fase d'implementació s'ha pogut realitzar sense imprevistos. Per tant, es pot concloure que la metodologia utilitzada ha estat efectiva i eficaç pel projecte, i no ha calgut fer cap modificació en els mètodes inicials.

Pel que fa a les eines utilitzades, GitHub ha estat eficaç per dur a terme la implementació dels canvis i millores en un entorn aïllat sense interferir en el normal desenvolupament de les funcionalitats del *dashboard*. A més, els cicles de *feedback* per correu electrònic i presencials amb la directora i codirectora del projecte han estat positius i han servit per prendre millors decisions i encaminar el projecte cap a la seva correcta execució.

11.4. Desviacions temporals

Partint de la planificació inicial, les tasques identificades i el seu contingut s'han mantingut sense alteracions. El propòsit de cadascuna d'elles s'ha complert tal i com estava previst, i ha servit per afegir valor al projecte i per avançar en el procés de reenginyeria. Tot i això, el temps assignat a cadascuna d'elles sí que ha patit variacions significatives. La planificació inicial es queda curta en quant a hores assignades a cada tasca, i això ha fet que s'hagi ampliat el termini de realització de les mateixes. Les hores planificades i invertides en les diverses tasques es poden veure a la Taula 21.

	Tasca	Hores estimades	Hores reals
1	**Organització i planificació del projecte**	70	72
2	**Anàlisi de la situació actual**	80	75
3	**Identificació dels problemes, anàlisi d'alternatives i proposta de solució**	60	149
4	**Refinament dels requisits de qualitat**	20	14
5	**Disseny dels canvis**	40	40
6	**Pla d'execució dels canvis**	10	3
7	**Implementació dels canvis**	150	232
8	**Validació dels requisits de qualitat**	20	17
9	**Finalització de la memòria**	30	33
10	**Preparació de la defensa**	30	30
	Total	**510**	**665**

Taula 21. Distribució d'hores per tasca

Tal i com es pot observar, hi ha dues tasques que han presentat desviacions significatives respecte a les hores planificades: la identificació dels problemes, anàlisi d'alternatives i proposta de solució i la implementació dels canvis. D'aquestes dues, la primera ha estat la que més s'ha desviat respecte a les hores planificades, en un factor de 2,5. La raó d'aquesta desviació ha estat que, per cada problema identificat, cal aportar diverses solucions i alternatives per tal de poder justificar amb coherència una solució final. Per tal de comparar aquestes alternatives i escollir correctament, cal familiaritzar-se amb les eines i tecnologies que es presenten i argumentar amb coneixement de causa quina és la millor opció. Tot això comporta una quantitat de temps considerable tenint en

compte el nombre de problemes que es volen solucionar, i el temps dedicat a cercar i descriure alternatives raonables va ser molt superior a allò esperat en un principi.

A més, la implementació dels canvis també ha sofert una desviació considerable, tot i que en menor mesura. Aquest és un apartat delicat per la seva naturalesa, ja que està subjecte a moltes incerteses i situacions imprevistes que poden fer més difícil la implementació de certes modificacions. Tot i això, s'han pogut implementar amb èxit tots els canvis i millores identificats en tasques anteriors.

Arrel de la primera desviació, que va tenir lloc en una fase relativament inicial del projecte, va ser necessari replantejar els terminis per tal de poder acabar el treball de manera satisfactòria. Com que ja s'havia produït una desviació important, i per tal de poder administrar possibles desviacions futures, es va optar per posposar la data d'entrega del projecte fins al torn d'octubre. Una altra opció possible que en un inici es va considerar era reduir l'abast del projecte i resoldre un nombre inferior de problemes. Tot i això, amb aquesta opció l'entitat del procés de reenginyeria es veia massa afectada, i les millores en els nivells de qualitat del projecte no haguessin estat tant palpables. A més, la disponibilitat de temps per part de l'autor d'aquest projecte i la voluntat de seguir contribuint en un projecte en el qual havia estat treballant durant un any com a desenvolupador van fer que la opció escollida fos allargar el termini d'entrega i dur a terme tot allò que estava previst des d'un inici.

11.4.1. Diagrama de Gantt final

Amb aquests canvis, el diagrama de Gantt inicial es veu afectat. La versió actualitzada amb les desviacions és la següent (Figura 47 i Figura 48):

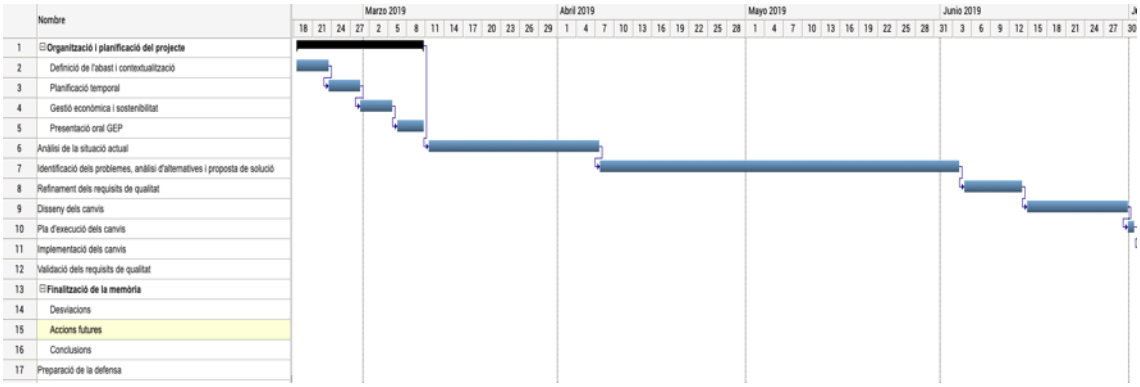

Figura 47. Diagrama de Gantt actualitzat (Primera part)

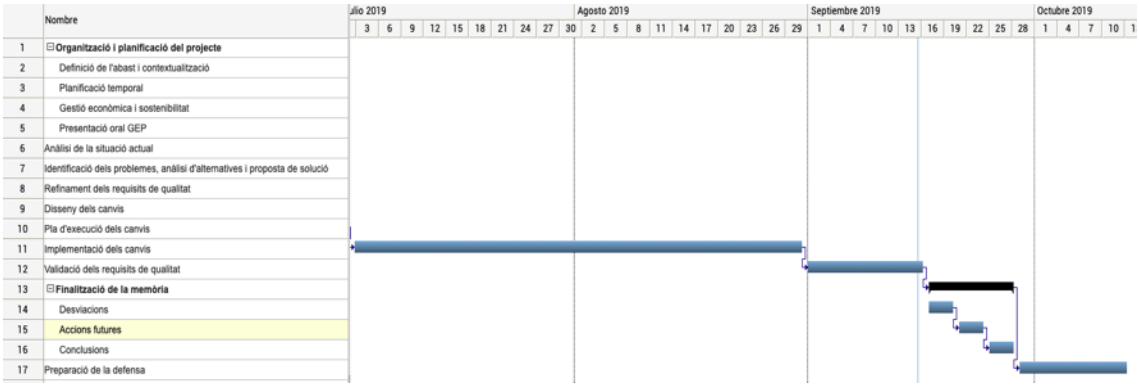

Figura 48. Diagrama de Gantt actualitzat (segona part)

11.5. Desviacions pressupostàries

Les desviacions mencionades anteriorment han tingut un cert efecte sobre el projecte, especialment en la part econòmica. Tal i com s'ha mencionat abans, les desviacions són de caràcter temporal, i gràcies a la possibilitat d'aplaçar la data d'entrega s'han pogut mantenir tots els objectius inicials del projecte utilitzant els recursos humans, materials i de software que es tenien previstos. Per contra, l'aspecte econòmic s'ha vist afectat significativament, ja que les hores dedicades han augmentat respecte a les previstes inicialment.

Partint del pressupost inicial, cal calcular de nou les hores per cada rol, i obtenir el costos directes i indirectes. Així doncs, el cost final amb les hores invertides queda plasmat en les següents taules. La Taula 22 i la Taula 23 mostren els salaris i costos per rol, que es mantenen sense canvis. A la Taula 24 es calculen els costos directes, actualitzant les hores reals de les tasques ja finalitzades. La Taula 25 conté els costos indirectes amb les hores reals totals invertides en el projecte. Per últim, la Taula 26 mostra el cost total del projecte. En el càlcul d'aquest cost s'han eliminat les contingències i imprevistos, ja que aquests corresponen al càlcul del pressupost inicial.

Rol	Salari mínim anual (€)	Salari mínim per hora (€)	Salari màxim anual (€)	Salari màxim per hora (€)	Salari mitjà anual (€)	Salari mitjà per hora (€)
Cap de projecte	40.000	22,22	60.000	33,33	50.000	27,77
Arquitecte	40.000	22,22	60.000	33,33	50.000	27,77
Analista	24.000	13,33	45.000	25	34.500	19,16
Programador	18.000	10	40.000	22,22	29.000	16,11

Taula 22. Salaris per rol

Rol	Salari mitjà per hora (€)	Percentatge seguretat social (%)	Cost per hora (€)
Cap de projecte	27,77	35	37,49
Arquitecte	27,77	35	37,49
Analista	19,16	35	25,87
Programador	16,11	35	21,75

Taula 23. Cost per rol

Activitat	Hores	Rol	Cost(€)
Organització i planificació del projecte	72	Cap de projecte	2.699,28
Anàlisi de la situació actual	75	Arquitecte	2.811,75
Identificació dels problemes, anàlisi d'alternatives i proposta de solució	149	Arquitecte	5.586,01
Refinament dels requisits de qualitat	14	Analista	362,18
Disseny dels canvis	40	Arquitecte	1.499,60
Pla d'execució dels canvis	3	Cap de projecte	112,47
Implementació dels canvis	232	Programador	5.046

Validació dels requisits de qualitat	17	Analista	439,79
Finalització de la memòria	33	Cap de projecte	1.237,17
Preparació de la defensa	30	Cap de projecte	1.124,7
Total			**20.918,95 €**

Taula 24. Costos directes

Producte	Quantitat	Preu unitari	Vida útil	Cost/hora	Hores	Cost (€)
	Amortitzacions					
Ordinador portàtil	1 unitat	2.799 €	5 anys	0,311 €/h	665	206,82
Pantalla externa	1 unitat	199 €	5 anys	0,022 €/h	665	14,63
Teclat i ratolí	1 unitat	40 €	2 anys	0,011 €/h	665	7,32
Taula i cadira d'oficina	1 unitat	500 €	6 anys	0,046 €/h	665	30,59
IntelliJ IDEA Ultimate	1 unitat	499 €	1 any	0,277 €/h	665	184,21
Microsoft Office	1 unitat	50 €	1 any	0,028 €/h	665	18,62
	Despeses generals					
Connexió a internet	-	50 €/mes	-	0,208 €/h	665	138,32
Electricitat	23,94 kWh[4]	0,12 €	-	-	-	2,87
Total						**603,38 €**

Taula 25. Costos indirectes

Activitat	Cost (€)
Costos directes	20.918,95
Costos indirectes	603,38
Total	**21.522,33 €**

Taula 26. Cost total

Un cop calculat el cost total final, es pot apreciar com aquest ha estat major de l'esperat. Inicialment, el pressupost contemplava una despesa de 19.015,17 €, i tenint en compte les desviacions patides durant el projecte el cost total puja fins als **21.522,33 €**.

Per tal de detallar les desviacions que s'han produït, cal calcular els indicadors de desviació definits a l'apartat 4.2. Control de gestió. En primer lloc, a la Taula 27 es poden veure les desviacions de mà d'obra. Per una banda es calculen les desviacions en preu, i per altra banda les desviacions en consum, així com les desviacions totals. Tal i com es pot apreciar, les desviacions que s'han produït corresponen a la categoria de desviacions de consum.

[4] El consum que s'ha produït durant el projecte es calcula a partir de dels 0,036 kWh que es consumeixen en un hora per part de l'ordinador portàtil i la pantalla externa, multiplicat per les 665 hores finals de dedicació.

Rol	Cost/hora estimat (€)	Cost/hora real (€)	Hores estimades	Hores reals	Desviació en preu (€)	Desviació en consum (€)
Cap de projecte	37,49	37,49	140	138	0	74,98
Arquitecte	37,49	37,49	180	264	0	-3.149,16
Analista	25,87	25,87	40	31	0	232,83
Programador	21,75	21,75	150	232	0	-1.783,5
Total					**0**	**-4.624,85**

Taula 27. Desviacions de mà d'obra

A la Taula 28 es calculen les desviacions de les tasques que han conformat aquest projecte. De la mateixa manera que a la taula anterior, es calculen les desviacions en preu, en consum i les desviacions totals. De nou, les desviacions que s'han produït són desviacions de consum.

Activitat	Cost/hora estimat (€)	Cost/hora real (€)	Hores estimades	Hores reals	Desviació en preu(€)	Desviació en consum (€)
Organització i planificació del projecte	37,49	37,49	70	72	0	-74,98
Anàlisi de la situació actual	37,49	37,49	80	75	0	187,45
Identificació dels problemes, anàlisi d'alternatives i proposta de solució	37,49	37,49	60	149	0	-3.336,61
Refinament dels requisits de qualitat	25,87	25,87	20	14	0	155,22
Disseny dels canvis	37,49	37,49	40	40	0	0
Pla d'execució dels canvis	37,49	37,49	10	3	0	262,43
Implementació dels canvis	21,75	21,75	150	232	0	-1.783,5
Validació dels requisits de qualitat	25,87	25,87	20	17	0	77,61
Finalització de la memòria	37,49	37,49	30	33	0	-112,47
Preparació de la defensa	37,49	37,49	30	30	0	0
Total					**0**	**-4.624,85**

Taula 28. Desviacions de tasques

Per últim, a la Taula 29 es mostren les desviacions en el recursos utilitzats en aquest projecte. De la mateixa manera que per les altres desviacions, es mostra la desviació en preu, en consum i les desviacions totals. Com a cas particular, les desviacions en electricitat no s'han pogut calcular a partir del cost per hora, de manera que es calculen en funció del cost estimat menys el cost real, ja que presenta una desviació degut a la diferència en les hores consumides i no en el preu. De nou, les desviacions en consum són les úniques que es produeixen en els costos dels recursos.

Producte	Cost/hora estimat (€)	Cost/hora real (€)	Hores estimades	Hores reals	Desviació en preu (€)	Desviació en consum (€)
	Amortitzacions					
Ordinador portàtil	0,311 €/h	0,311 €/h	510	665	0	-48,21
Pantalla externa	0,022 €/h	0,022 €/h	510	665	0	-3,41
Teclat i ratolí	0,011 €/h	0,011 €/h	510	665	0	-1,71
Taula i cadira d'oficina	0,046 €/h	0,046 €/h	510	665	0	-7,13
IntelliJ IDEA Ultimate	0,277 €/h	0,277 €/h	510	665	0	-42,94
Microsoft Office	0,028 €/h	0,028 €/h	510	665	0	-4,34
	Despeses generals					
Connexió a internet	0,208 €/h	0,208 €/h	510	665	0	-32,24
Electricitat	-	-	510	665	0	-0,67
Total						**-140,65**

Taula 29. Desviacions de recursos

12. Sostenibilitat i compromís social

En aquest apartat es tracta la sostenibilitat del projecte des de les seves tres dimensions: l'econòmica, l'ambiental i la social.

12.1. Dimensió ambiental

Tal i com s'especifica en el càlcul dels costos indirectes detallats en l'informe del cost final, el consum dels aparells elèctrics per aquest projecte és de 23,94 kWh. A aquest valor cal sumar el consum dels recursos humans, que per aquest projecte és d'una persona que ha treballat 665 hores. En una rutina habitual, una persona consumeix per hora 0,1 kWh, consumint en total 66,5 kWh durant la realització del projecte. En total, el valor final del consum d'aquest Treball Final de Grau és de 90,44 kWh.

Tot i aquest impacte generat, els recursos utilitzats per aquest projecte no són exclusius del mateix, sinó que formen part dels recursos que actualment ja disposa l'autor. Per tant, es redueix l'impacte ambiental de la fabricació dels recursos ja que no cal adquirir-ne de nous per realitzar el projecte. Aquestes han tingut, tenen i continuaran tenint diferents usos més enllà d'aquest treball final de grau. A més, els recursos utilitzats durant aquest projecte han estat els mínims necessaris per tal de dur-lo a terme, aprofitant correctament tot allò del que l'autor ha disposat. En cas de fer-lo de nou, els recursos utilitzats es mantindrien sense canvis, tot i que les hores emprades en la seva realització disminuirien degut als coneixements adquirits.

Pel que fa a la vida útil d'aquest projecte, els recursos que s'usaran coincideixen amb aquells emprats pels desenvolupadors del *dashboard* estratègic, i en general l'impacte ambiental d'aquests coincideix amb el consum produït durant aquest projecte, adaptat en funció de les hores que es dediquin al desenvolupament en un futur. Tot i això, aquest projecte de reenginyeria no suposa l'ús de més recursos en el futur, sinó tot el contrari, ja que l'objectiu és facilitar la feina als futurs desenvolupadors per tal que el consum d'hores es redueixi, i per tant també ho faci l'impacte ambiental. Globalment, la petjada ecològica es veurà millorada gràcies a la feina feta durant aquest procés de reenginyeria i a la reducció de l'ús dels recursos que implica.

Per acabar, cal mencionar que un dels riscos que podria fer augmentar la petjada ecològica del projecte és una eventual modificació o desmantellament d'alguns dels canvis introduïts en aquest projecte per part d'un futur desenvolupador, degut a que allò proposat en aquest procés de reenginyeria ja no s'ajustés a les necessitats del sistema. Això comportaria dedicar més hores a aquesta tasca, reduint el benefici ambiental del qual es disposa en un inici.

12.2. Dimensió econòmica

La quantificació del cost econòmic dels recursos s'ha dut a terme a l'apartat 11.5.Desviacions pressupostàries, mostrant quin és el cost atribuïble tant a les activitats derivades directament de la realització d'aquest projecte com als elements indirectes que intervenen i fan possible la realització de les mateixes. Tal i com es comenta en aquest apartat, els recursos humans i materials s'han mantingut sense canvis respecte a allò planificat originàriament, tot i que el temps dedicat ha estat més elevat.

El fet de tractar-se d'un projecte realitzat dins de la universitat i que forma part d'un projecte d'investigació fa que els costos es limitin al mínim necessari, sense buscar el benefici econòmic. S'han mantingut uns costos raonables per un projecte d'aquestes característiques, utilitzant els mínims recursos materials i humans. A més, el temps dedicat a cada tasca és proporcional a la seva importància, i els rols que hi participen són els necessaris en cada cas.

Un cop acabat el projecte, s'ha pogut comprovar que el cost ha estat més elevat respecte al plasmat en el pressupost. La raó que ha causat aquesta desviació és la planificació temporal, que no s'ha ajustat a la realitat del projecte. Aquestes desviacions s'exposen a l'apartat 11.4. Desviacions temporals, on s'argumenta que la falta d'experiència en projectes d'aquest estil i la dificultat tècnica d'alguns apartats han estat els motius principals de la desviació temporal. Per tal de reduir aquesta desviació en projectes futurs, caldrà dur a terme una planificació temporal més conservadora, afegint més hores a aquelles tasques amb un alt grau d'incertesa.

Pel que fa a la vida útil, la naturalesa d'aquest projecte de reenginyeria fa que no suposi un cost afegit pel sistema, sinó un estalvi. Durant tot el que resta de desenvolupament, les tasques realitzades en aquest projecte contribuiran a facilitar la feina dels desenvolupadors i a abaratir els costos econòmics, ja que el temps dedicat a afegir noves funcionalitats i detectar i solucionar problemes es veurà reduït.

Per acabar, és necessari mencionar que, com que es aquest és un treball realitzat dins d'un projecte d'investigació d'una universitat pública, l'aspecte econòmic sempre està subjecte als recursos dels que disposi la universitat, i un risc que cal tenir en compte és la impossibilitat de fer front a la despesa econòmica que comporta aquest projecte degut a retallades en el pressupost.

12.3. Dimensió social

A nivell professional, aquest projecte ha contribuït a millorar i aprofundir el coneixement sobre diverses àrees de l'enginyeria del software gràcies als aspectes transversals que s'han pogut abordar, sempre centrats al voltant del concepte de qualitat. La diversitat d'eines i tecnologies utilitzades juntament amb la possibilitat d'aplicar principis i tècniques de disseny i elements de gestió de projectes fan que aquest hagi estat un projecte molt complet des del punt de vista tècnic, i molt enriquidor a nivell professional.

A més, a nivell personal aquest projecte també ha estat molt beneficiós ja que, tot i el gran esforç que ha comportat, ha permès tancar un cicle que es va començar fa més d'un any amb la participació de l'autor com a desenvolupador del *dashboard*, i que conclou amb l'aplicació de canvis i millores que permeten deixar el projecte en un estat de qualitat superior respecte a l'estat inicial amb el qual es va començar.

Durant la vida útil d'aquest projecte, els principals beneficiaris seran els futurs desenvolupadors, que podran disposar d'una infraestructura i un estat intern del sistema amb uns nivells superiors de qualitat, que permetrà dur a terme la seva feina amb unes millors condicions i destinar els esforços a afegir nou valor al sistema. Els efectes d'aquest projecte no perjudiquen a cap col·lectiu, ja que els caps de projecte i

fins i tot els usuaris finals podran apreciar efectes positius gràcies a la realització d'aquest projecte.

Cal destacar també que aquest projecte dona resposta a tots i cadascun dels objectius plantejats inicialment tal i com s'exposa a l'apartat 11.1. Resolució d'objectius i abast, i que la situació problemàtica que s'identifica inicialment es veu millorada de forma substancial. Tot i això, com que els conceptes que tenen relació amb la qualitat són difícils de solucionar del tot, encara es disposa de marge de millora que caldrà dur a terme en un futur, tal i com s'exposa en el capítol següent.

Per últim, a nivell de riscos socials no es contempla cap situació que pogués convertir aquest projecte en perjudicial per algun segment de població, ja que l'abast de persones afectades per ell és reduït. A més, tampoc es genera als usuaris cap tipus de dependència, ja que el fet de ser una millora sobre un projecte ja existent fa que, un cop duta a terme, el projecte continuï el seu curs i evolucioni en funció de les seves necessitats.

12.4. Conclusions sobre la sostenibilitat

Per acabar, es presenten quines són les conclusions que es poden extreure sobre la sostenibilitat d'aquest projecte. Tal i com s'ha comentat en els apartats anteriors, l'impacte que el procés de reenginyeria té sobre les tres dimensions de la sostenibilitat és positiu, ja que la seva naturalesa implica la resolució de diversos elements problemàtics que formen part de l'estat inicial del *dashboard* estratègic, i que un cop resolts permeten obtenir uns beneficis que es veuen reflectits tant en l'aspecte ambiental, econòmic i social.

La millora de la qualitat del dashboard permetrà implementar noves funcionalitats en un menor temps i amb menys esforç, cosa que permet reduir la petjada ecològica produïda durant la construcció del sistema, reduir els costos econòmics derivats del seu desenvolupament i influir positivament sobre el col·lectiu social que s'encarrega d'aquestes tasques. A més, per cada dimensió s'ha analitzat els possibles riscos que es poden presentar, tot i que el nombre és molt reduït i les accions que es poden realitzar per reduir-los queden fora de l'abast de l'autor del projecte.

Per tant, es conclou que aquest és un projecte sostenible, on es presenta una millora real en totes les dimensions de la sostenibilitat.

13. Accions futures

En aquest punt del projecte, el procés de reenginyeria es dona per acabat de manera satisfactòria. S'han complert els objectius marcats a l'inici del projecte, duent a terme totes les tasques planificades i implementant tots els canvis i millores per tal de solucionar els problemes analitzats, millorant així els nivells de qualitat del *dashboard*. Tot i això, cal tenir en compte que aquest és un projecte amb una abast limitat degut a les restriccions d'un Treball Final de Grau, i que hi ha diversos elements del sistema sobre els quals es pot aprofundir més o bé no s'han considerat dins del procés de reenginyeria, i que constitueixen una base sobre la que seguir iterant en el futur per tal de millorar encara més els nivells de qualitat del *dashboard*.

Per començar, cal parlar dels problemes detectats per SonarCloud. Tot i que s'han eliminat els problemes de qualitat més greus i el nombre total de problemes ha disminuït de forma significativa, en un futur caldrà seguir treballant per tal d'eliminar la resta de problemes identificats per l'eina. El fet d'haver introduït SonarCloud en un punt avançat del desenvolupament del *dashboard* ha fet que el nombre de problemes detectats hagi estat alt, però partint de la feina feta durant aquest procés de reenginyeria, es podran introduir futures millores fetes a partir de la informació facilitada per aquesta eina per tal d'aconseguir uns nivells de qualitat de codi superiors.

Un altre punt que ha quedat fora de l'abast d'aquest projecte és la millora de les entitats de domini del *dashboard*. Degut a la metodologia àgil seguida durant el desenvolupament de funcionalitats del dashboard, hi ha certs elements del domini que han evolucionat condicionats per versions anteriors, sense alterar massa el funcionalment inicial o bé mantenint elements que jo no són necessaris. Això es deu, en gran part, a l'estret lligam entre el domini i la base de dades degut a la utilització de Hibernate com a ORM. Durant el desenvolupament de noves funcionalitats, s'ha intentat reduir l'impacte que els canvis tenen en la base de dades (principalment, modificacions de tipus de dades, canvis en la clau primària, etc.) per tal que el desplegament fos el més senzill possible per part dels socis tecnològics del projecte.

Un cop el projecte de recerca es doni per finalitzat, una possible millora que es pot implementar és una avaluació detallada i un redisseny del domini i de la base de dades per tal de representar d'una forma més concisa i correcte l'actual estat del *dashboard* i les seves funcionalitats. A més, els DTOs que es fan servir com a vista dins dels serveis REST també es veuran afectats, reduint la seva complexitat i modificant els camps que siguin necessaris.

Per últim, cal parlar de la capa de presentació del *dashboard*. Tal i com s'ha comentat en apartats anteriors, l'abast d'aquest projecte es limita a la part de servidor del sistema, sense analitzar ni alterar cap element relatiu a la part de client. Durant aquest procés de reenginyeria, s'ha modificat la capa de presentació per tal d'incloure-hi els controladors i les vistes dels serveis REST, ja que degut a les seves responsabilitats aquests elements han de formar part d'aquesta capa. Tot i això, no s'ha fet referència a la part del client del sistema, és a dir, a les vistes i components que permeten interacció amb l'usuari i mostren els resultats. Aquesta és una part del dashboard que s'executa als navegadors web, i que degut a les seves particularitats tecnològiques i complexitat mereix un tractament a part.

Actualment, la part de client del *dashboard* està formada per vistes HTML i arxius de codi Javascript. Aquests arxius s'encarreguen de realitzar peticions al servidor del *dashboard*, recollir i processar els resultats i omplir les vistes per tal de mostrar-los. Tot i això, aquest codi no segueix cap estructura ni arquitectura definides, i els arxius contenen a la vegada tot tipus d'instruccions encarregades de les responsabilitats abans esmentades, esdevenint en arxius llargs i amb molta repetició de codi entre ells.

Per tal de resoldre aquesta situació, es proposa com a acció de futur refer la part de client del dashboard, utilitzant alguna de les nombroses eines o framework existents com ara React, Angular o Vue per tal de dotar a aquesta part del sistema d'una estructura sòlida i una arquitectura que permeti fer canvis amb facilitat, a part d'aprofitar els beneficis funcionals que aquest tipus d'eines aporten per tal de potenciar la part visual del *dashboard*.

14. Conclusions

Per acabar, es presenten les conclusions del Treball Final de Grau exposant les conclusions del propi projecte, la valoració personal de l'autor i l'anàlisi de les competències tècniques associades al desenvolupament del mateix.

14.1. Conclusions del projecte

Un cop finalitzat el procés de reenginyeria, cal veure quines són les conclusions que es poden extreure de tot el treball realitzat. En primer lloc, cal destacar que el motiu de la realització d'aquest projecte respon a una necessitat real. La metodologia de desenvolupament utilitzada durant la construcció del *dashboard* estratègic i les característiques del propi projecte Q-Rapids han fet que, durant tot el període de temps en el que s'ha estat desenvolupant, no s'hagi pogut destinar temps ni esforços per aplicar un procés de millora de la qualitat del sistema. Les conseqüències d'aquesta situació es feien constatables durant les darreres fases del desenvolupament, on els nivells de qualitat sovint penalitzaven les noves funcionalitats que calia afegir.

Dins de la feina realitzada en aquest procés de reenginyeria, gran part de l'esforç ha estat destinat a realitzar un anàlisi de la situació actual i dels diversos problemes que afectaven a la seva qualitat. Si bé és cert que no s'han pogut analitzar tots els problemes que pot presentar el *dashboard* degut a que l'abast limitat d'aquest projecte al tractar-se d'un Treball Final de Grau, els problemes que es tracten en aquest projecte són de naturalesa diversa i es busca poder influir de manera transversal en tot el sistema. A més, dins de cada problema es contemplen diverses alternatives per tal de solucionar la situació inicial, de manera que la proposta de solució que es fa per cadascun d'ells s'adapti realment a les característiques del projecte.

Un altre aspecte a destacar és que aquest anàlisi detallat ha estat clau per tal de poder dur a terme amb èxit la fase d'implementació de les millores. Gràcies a la comprensió del problema a resoldre i de les eines que calia utilitzar, s'ha pogut dissenyar i implementar tot allò necessari per tal de modificar correctament el *dashboard*, amb el suport d'un pla d'execució que ha permès treure el màxim profit dels canvis i minimitzar errors durant tot el procés. Els efectes de tots aquests canvis i millores han estat validats seguint els criteris definits als requisits de qualitat establerts inicialment, assegurant que la qualitat del *dashboard* ha millorat gràcies a les solucions implementades.

En conclusió, tot i les desviacions temporals que ha patit aquest projecte i que han allargat el seu període d'entrega, la feina realitzada durant totes les etapes del procés de reenginyeria ha servit per millorar la situació inicial i incrementar els nivells de qualitat del projecte, de manera que els seus beneficis es podran apreciar en futures fases de desenvolupament del *dashboard*. De la mateixa manera, aquest projecte també situa els fonaments per tal que les futures millores que encara són necessàries realitzar es puguin dur a terme amb més facilitat i seguretat. Amb tot això, es considera que el procés de reenginyeria del *dashboard* estratègic de l'eina Q-Rapids ha estat satisfactori i exitós.

14.2. Valoració personal

Aquest ha estat un projecte molt gratificant i enriquidor per mi. Per començar, el fet de poder desenvolupar aquest treball sobre el *dashboard* estratègic de l'eina Q-Rapids ha fet que hagi pogut aprofitar tot allò après durant els mesos que he estat treballant com a desenvolupador dins del projecte. La necessitat de dur a terme aquest procés de reenginyeria va ser una voluntat personal, i el fet de poder veure i entendre la necessitat dels canvis i els beneficis que aquests aportaven al desenvolupament ha estat una motivació molt gran, ja que he pogut donar solució als entrebancs i dificultats que he pogut experimentar en primera persona durant el desenvolupament de les funcionalitats del sistema.

Un altre aspecte que he trobat molt interessant és la oportunitat de treballar sobre un sistema ja començat, amb moltes funcionalitats existents. Tot i que és cert que la corba d'aprenentatge ha estat molt suau degut a la meva feina com a desenvolupador, sí que he hagut d'analitzar i modificar codi fet per altres desenvolupadors, i examinar parts del sistema on encara no havia tingut la oportunitat d'aprofundir. El fet d'haver de millorar un sistema ja existent s'allunya bastant de tot allò que es treballa a les assignatures del Grau, on la manera de treballar acostuma a ser desenvolupant des de zero projectes i aplicacions. Considero que el fet de poder identificar problemes de qualitat en aplicacions existents i poder millorar-los és una habilitat molt important per un enginyer de software, ja que en la vida laboral les oportunitats de començar de zero un projecte solen ser escasses, i en general es treballa sobre sistemes que porten un cert temps en desenvolupament.

També cal destacar la varietat dels canvis introduïts durant el procés de reenginyeria, que m'ha permès modificar aspectes que van des d'elements d'un nivell d'abstracció molt baix, com poden ser certes línies de codi, fins a elements del nivell més alt d'abstracció, modificant l'estructura de paquets i l'arquitectura general del sistema, passant per proves automatitzades de diferents tipus. També s'han introduït conceptes de gestió de projectes i eines externes per donar suport al procés de desenvolupament. Tot això ha comportat que els coneixements que he adquirit i les eines que he descobert hagin estat molt transversals, i hagi pogut influir en gairebé tots els aspectes del desenvolupament del *dashboard*. Crec que aquesta varietat i transversalitat ha estat molt positiva per mi, ja que la visió que ara tinc sobre el propi sistema i sobre certes pràctiques, principis i patrons aplicables a qualsevol projecte de software és molt més àmplia i rica, i considero que han fet de mi un millor enginyer de software.

14.3. Anàlisi de les competències tècniques

En aquest apartat cal analitzar si el grau de compliment de les competències tècniques que formen part del desenvolupament del projecte es correspon amb allò triat a la fase inicial. Per tal de fer-ho, es menciona cadascuna d'aquestes competències i s'argumenta com s'ha aplicat durant el transcurs del projecte.

CES1.1: Desenvolupar, mantenir i avaluar sistemes i serveis software complexos i/o crítics. [Bastant]

L'objecte del procés de reenginyeria ha estat el *dashboard* estratègic de l'eina Q-Rapids, que constitueix un element complex i d'elevada importància dins de la pròpia eina. La seva complexitat radica, per una banda, en l'estructura dels components que formen part l'eina Q-Rapids i que el *dashboard* s'encarrega d'integrar, i per altra banda en la pròpia complexitat dels components interns del *dashboard*, tant per l'elevat nombre de mòduls que el composen com pels problemes de qualitat inicials. Per tal de dur a terme les millores en la qualitat, s'ha hagut d'avaluar amb detall l'estat actual del sistema i desenvolupar els canvis necessaris per resoldre els problemes identificats.

CES1.3: Identificar, avaluar i gestionar els riscos potencials associats a la construcció de software que es poguessin presentar. [Una mica]

Durant l'anàlisi de l'estat actual i la identificació dels problemes presents al dashboard, una de les tasques realitzades ha estat comprendre quines eren les implicacions de cadascuna de les alternatives considerades per tal de resoldre els diferents problemes. D'entre les diferents alternatives, s'ha escollit aquella que representava un major benefici i un menor risc pel desenvolupament normal del dashboard, així com pel propi procés de reenginyeria. A més, existeix un apartat dedicat exclusivament a ordenar les tasques que cal realitzar dins del projecte per tal de reduir al màxim els possibles efectes que les desviacions en la implementació de les millores poguessin tenir en el projecte.

CES1.4: Desenvolupar, mantenir i avaluar serveis i aplicacions distribuïdes amb suport de xarxa. [En profunditat]

Tal i com s'ha comentat, el dashboard estratègic és una aplicació web que segueix el paradigma client servidor. Aquest procés de reenginyeria ha girat al voltant de la part de servidor de l'aplicació, analitzant amb detall l'estat actual, proposant millores pels problemes de qualitat identificats i desenvolupant les solucions necessàries. Tot i que hi ha millores implementades que responen a principis i bones pràctiques presents en qualsevol tipus de software, la condició d'aplicació distribuïda s'ha tingut en compte a l'hora de dissenyar els canvis necessaris per tal de millorar-ne la qualitat.

En concret, una de les millores realitzades ha estat centrada en el disseny dels serveis REST, que constitueixen la interfície entre el client i el servidor en una aplicació distribuïda amb suport de xarxa, de manera que ha calgut avaluar el disseny actual i identificar els canvis necessaris per tal de dissenyar i implementar una interfície adequada als criteris de disseny d'aquest tipus de serveis. A més, també s'ha alterat l'arquitectura interna del servidor del dashboard pensant en l'estructura més adequada per una aplicació distribuïda, separant la lògica del domini dels controladors REST de manera que aquests últims puguin passar a formar part de la capa de presentació del servidor.

CES1.7: Controlar la qualitat i dissenyar proves en la producció de software. [En profunditat]

L'objectiu d'aquest Treball Final de Grau ha estat elevar els nivells de qualitat del *dashboard* estratègic de l'eina Q-Rapids. Per aquest motiu, ha calgut analitzar amb detall la qualitat de l'estat inicial del sistema, descrivint la seva estructura i característiques per tal de poder identificar problemes en la qualitat. S'han avaluat diferents alternatives

pels problemes identificats, i s'han dissenyat i implementat solucions que han permès millorar la qualitat inicial del *dashboard*.

Tot i que totes les solucions proposades i implementades han contribuït a la millora de la qualitat inicial, cal destacar-ne algunes que, més enllà de les implicacions descrites en aquest projecte, tindran un efecte controlador en els nivells de qualitat durant la resta del desenvolupament del *dashboard*. Per una banda, les proves automatitzades introduïdes en forma de proves unitàries i d'integració permetran detectar amb més facilitat futurs errors introduïts en funcionalitats ja existents, facilitant la seva correcció i prevenint problemes greus.

Per altra banda, la integració d'eines com SonarCloud i Travis CI dins del procés de desenvolupament fa que es puguin controlar i monitoritzar els nivells de qualitat del producte a mesura que es van afegint noves funcionalitats, gràcies a l'automatització dels anàlisis de qualitat executats per SonarCloud i l'execució de les proves automatitzades per part de Travis CI.

15. Bibliografía

[1] «Q-Rapids,» [En línea]. Available: www.q-rapids.eu.

[2] E. J. Chikofsky y J. H. Cross, «Reverse Engineering and Design Recovery: A Taxonomy,» *IEEE Software,* 1990.

[3] E. Byrne, «A conceptual foundation for software re-engineering,» 2003.

[4] A. S. Abbas, W. Jeberson y V. V. Klinsega, «Journal of Engineering and Technology The Need of Re-engineering in Software Engineering,» *International Journal of Engineering and Technology,* vol. 2, nº 2, 2012.

[5] M. Majthoub, M. H. Qutqui y Y. Odeh, «Software Re-engineering: An Overview,» de *2018 8th International Conference on Computer Science and Information Technology, CSIT 2018*, 2018.

[6] P. Bengtsson y J. Bosch, «Scenario-based software architecture reengineering,» 2002.

[7] M. Fowler, «Technical Debt,» [En línea]. Available: martinfowler.com/bliki/TechnicalDebt.html.

[8] Wikipedia, «Waterfall model,» [En línea]. Available: en.wikipedia.org/wiki/Waterfall_model.

[9] GitHub Inc, «GitHub,» [En línea]. Available: https://github.com.

[10] Software Freedom Conservancy, «Git,» [En línea]. Available: https://git-scm.com.

[11] SonarSource SA, «SonarCloud,» [En línea]. Available: https://sonarcloud.io/about.

[12] D. Ameller, C. Ayala, J. Cabot y X. Franch, «How do software architects consider non-functional requirements: An exploratory study,» de *2012 20th IEEE International Requirements Engineering Conference, RE 2012 - Proceedings*, 2012.

[13] A. Caracciolo, M. F. Lungu y O. Nierstrasz, «How do software architects specify and validate quality requirements?,» de *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014.

[14] JetBrains, «IntelliJ IDEA,» [En línea]. Available: https://www.jetbrains.com/idea/.

[15] Postman Inc, «Postman,» [En línea]. Available: https://www.getpostman.com.

[16] The Apache Software Foundation, «Apache Tomcat,» [En línea]. Available: https://tomcat.apache.org.

[17] Travis CI GMBH, «Travis CI,» [En línea]. Available: https://travis-ci.org.

[18] Microsoft, «Office,» [En línea]. Available: https://products.office.com/es-es.

[19] Smartapp.com Inc, «Gantter,» [En línea]. Available: https://www.gantter.com.

[20] JGraph Ltd, «Draw.io,» [En línea]. Available: https://www.draw.io.

[21] PageGroup, «Tendencias del mercado laboral,» 2018. [En línea]. Available: https://www.pagepersonnel.es/sites/pagepersonnel.es/files/PG_ER_IT_2018.pdf.

[22] X. Franch, C. Ayala, L. López, S. Martínez-Fernández, P. Rodríguez, C. Gómez, A. Jedlitschka, M. Oivo, J. Partanen, T. Räty y V. Rytivaara, «Data-driven Requirements Engineering in Agile Projects: The Q-Rapids Approach,» *Just-In-Time Requirements Egineering Conference,* 2017.

[23] Q-Rapids consortium, «D3.1 Dashboard Specification».

[24] L. López, S. Martínez-Fernández, C. Gómez, M. Choras, R. Kozik, L. Guzmán, A. M. Vollmer, X. Franch y A. Jedlitschka, «Q-Rapids Tool Prototype: Supporting Decision-Makers in Managing Quality in Rapid Software Development».

[25] Q-Rapids consortium, «D3.3 Dashboard consolidated prototype».

[26] Q-Rapids consortium, «D4.3 Architectural Blueprint».

[27] Apache Software Foundation, «Apache Kafka,» [En línea]. Available: https://kafka.apache.org.

[28] Elasticsearch B.V., «Elasticsearch,» [En línea]. Available: https://www.elastic.co/es/products/elasticsearch.

[29] Q-Rapids Consortium, «D1.3 Data gathering and analysis prototype».

[30] Pivotal Software, Inc, «Spring Framework,» [En línea]. Available: https://spring.io/projects/spring-framework.

[31] M. Fowler, «Inversion Of Control,» [En línea]. Available: https://martinfowler.com/bliki/InversionOfControl.html.

[32] R. C. Martin, Agile Software Development, Principles, Patterns and Practices, Prentice Hall, 2002.

[33] Pivotal Software, Inc, «Spring Boot,» [En línea]. Available: https://spring.io/projects/spring-boot#overview.

[34] Redhat, «Hibernate ORM,» [En línea]. Available: http://hibernate.org/orm/.

[35] The PostgreSQL Global Development Group, «PostgreSQL,» [En línea]. Available: https://www.postgresql.org/about/.

[36] Gradle Inc, «Gradle Build Tool,» [En línea]. Available: http://gradle.org/features/.

[37] The JUnit Team, «JUnit,» [En línea]. Available: https://junit.org.

[38] Mockito, «Mockito,» [En línea]. Available: https://site.mockito.org.

[39] Selenium, «Selenium,» [En línea]. Available: https://www.seleniumhq.org.

[40] Apache Software Foundation, «Apache JMeter,» [En línea]. Available: https://jmeter.apache.org.

[41] SpringFox, «SpringFox,» [En línea]. Available: https://springfox.github.io/springfox/.

[42] Pivotal Software, Inc., «Spring REST Docs,» [En línea]. Available: https://spring.io/projects/spring-restdocs.

[43] R. Daigneau, Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Series, Addison-Wesley, 2011.

[44] Codequest, «Codebeat,» [En línea]. Available: https://codebeat.co.

[45] Codacy, «Codacy,» [En línea]. Available: https://www.codacy.com.

[46] SonarSource SA, «SonarCloud,» [En línea]. Available: https://sonarcloud.io/about.

[47] SonarSource SA, «SonarQube,» [En línea]. Available: https://www.sonarqube.org.

[48] CloudBees, «CodeShip,» [En línea]. Available: https://codeship.com.

[49] Travis CI, GMBH, «Travis CI,» [En línea]. Available: https://travis-ci.org.

[50] Circle Internet Services, Inc., «Circle CI,» [En línea]. Available: https://circleci.com.

[51] T. Mueller, «H2 Database Engine,» [En línea]. Available: https://www.h2database.com.

16. Annex

En aquest annex es mostra el disseny de totes les funcionalitats del servei REST que requereixen algun tipus de canvi, recollits en les següents taules (Taula 30 fins a Taula 61). Els atributs descrits són el mètode HTTP, URI, paràmetres de consulta, cos de la petició i codis de resposta, i es descriu tant l'estat inicial com el nou disseny resultant, destacant en negreta aquells atributs que canvien.

Alertes

Obtenir nombre d'alertes noves		
	Inicial	**Nou**
Mètode	GET	GET
URI	**/api/alerts/new**	**/api/alerts/countNew**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Falta paràmetre	
	400 Bad Request	400 Bad Request
	Projecte no existeix	
	500 Internal Server Error	**400 Bad Request**

Taula 30. Obtenir nombre d'alertes

Obtenir patrons de requisits de qualitat per una alerta		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/alerts/{id}/qrPatterns	/api/alerts/{id}/qrPatterns
Paràmetres	-	-
Respostes	Correcte	
	200 OK	200 OK
	Alerta no existeix	
	500 Internal Server Error	**404 Not Found**
	Error de connexió amb PABRE	
	500 Internal Server Error	500 Internal Server Error

Taula 31. Obtenir patrons de requisits de qualitat per una alerta

Obtenir decisions preses per una alerta		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/alerts/{id}/decisions	/api/alerts/{id}/decisions
Paràmetres	-	-
Respostes	Correcte	
	200 OK	200 OK
	Alerta no existeix	
	500 Internal Server Error	**404 Not Found**
	Error de connexió amb PABRE	

	500 Internal Server Error	500 Internal Server Error

Taula 32. Obtenir decisions preses per una alerta

Ignorar el requisit de qualitat associat a una alerta		
	Inicial	**Nou**
Mètode	POST	POST
URI	/api/alerts/{id}/ignore	**/api/alerts/{id}/qr/ignore**
Paràmetres	prj	**-**
Cos	rationale patternId	rationale patternId
Respostes	Correcte	
	200 OK	200 OK
	Alerta no existeix	
	500 Internal Server Error	**404 Not Found**

Taula 33. Ignorar el requisit de qualitat associat a una alerta

Afegir un requisit de qualitat associat a una alerta		
	Inicial	**Nou**
Mètode	POST	POST
URI	/api/alerts/{id}/qr	/api/alerts/{id}/qr
Paràmetres	prj	**-**
Cos	rationale patternId requirement description goal	rationale patternId requirement description goal
Respostes	Correcte	
	200 OK	**201 Created**
	Alerta no existeix	
	500 Internal Server Error	**404 Not Found**
	Error de connexió amb backlog	
	500 Internal Server Error	500 Internal Server Error
	Alerta ja té un requisit de qualitat associat	
	405 Method Not Allowed	**409 Conflict**

Taula 34. Afegir un requisit de qualitat associat a una alerta

Afegir una nova alerta		
	Inicial	**Nou**
Mètode	POST	POST
URI	/api/notifyAlert	**/api/alerts**
Paràmetres	-	-
Cos	element: - Id - Name - Type - Value - Threshold - Category	element: - Id - Name - Type - Value - Threshold - Category

	- Project_id	- Project_id
Respostes	Correcte	
	200 OK	**201 Created**
	Error de connexió amb PABRE	
	500 Internal Server Error	500 Internal Server Error

Taula 35. Afegir una nova alerta

Categories

Crear categories pels indicadors estratègics i els factors		
	Inicial	**Nou**
Mètode	GET	**POST**
URI	/api/newCategories	**/api/categories**
Paràmetres	SICat QFCat	**-**
Cos	-	**SICat** **QFCat**
Respostes	Correcte	
	202 Accepted	**201 Created**
	Error en guardar les categories	
	405 Method Not Allowed	**500 Internal Server Error**

Taula 36. Crear categories pels indicadors estratègics i els factors

Esborrar categories d'indicadors estratègics i factors		
	Inicial	**Nou**
Mètode	GET	**DELETE**
URI	/api/deleteCategories	**/api/categories**
Paràmetres	-	-
Respostes	Correcte	
	200 OK	200 OK
	Error en esborrar les categories	
	400 Bad Request	**500 Internal Server Error**

Taula 37. Esborrar categories d'indicadors estratègics i factors

Indicadors estratègics

Crear indicador estratègic		
	Inicial	**Nou**
Mètode	POST	POST
URI	/api/newStrategicIndicator	**/api/strategicIndicators**
Paràmetres	-	**-**
Cos	name description network quality_factors	name description network quality_factors
Respostes	Correcte	
	202 Accepted	**201 Created**

	Error en guardar indicador estratègic	
	405 Method Not Allowed	**500 Internal Server Error**

Taula 38. Crear indicador estratègic

Obtenir un indicador estratègic		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/EditStrategicIndicator/{id}	**/api/strategicIndicators/{id}**
Paràmetres	-	**-**
Respostes	Correcte	
	202 Accepted	**200 OK**
	Error en obtenir indicador estratègic	
	400 Bad Request	**500 Internal Server Error**

Taula 39. Obtenir un indicador estratègic

Modificar un indicador estratègic		
	Inicial	**Nou**
Mètode	POST	**PUT**
URI	/api/EditStrategicIndicator/{id}	**/api/strategicIndicators/{id}**
Paràmetres	-	**-**
Cos	name description network quality_factors	name description network quality_factors
Respostes	Correcte	
	202 Accepted	**200 OK**
	Discordança en els factors	
	400 Bad Request	400 Bad Request
	Violació de restricció d'integritat	
	409 Conflict	409 Conflict
	Error genèric	
	405 Method Not Allowed	**500 Internal Server Error**

Taula 40. Modificar un indicador estratègic

Importar indicadors estratègics des de ElasticSearch		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/fetchSIs	**/api/strategicIndicators/fetch**
Paràmetres	-	**-**
Respostes	Correcte	
	202 Accepted	**200 OK**
	Error en la importació dels indicadors estratègics	
	400 Bad Request	**500 Internal Server Error**

Taula 41. Importar indicadors estratègics des de ElasticSearch

Avaluar indicadors estratègics		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/assessStrategicIndicators	**/api/strategicIndicators/assess**
Paràmetres	prj from train	prj from train
Respostes	Correcte	
	202 Accepted	202 Accepted
	Error en el càlcul dels indicadors estratègics	
	400 Bad Request	**500 Internal Server Error**

Taula 42. Avaluar indicadors estratègics

Simular avaluació dels indicadors estratègics		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/Simulate	**/api/strategicIndicators/simulate**
Paràmetres	prj factors	prj factors
Respostes	Correcte	
	202 Accepted	**200 OK**
	Error en el càlcul de la simulació	
	400 Bad Request	**500 Internal Server Error**

Taula 43. Simular avaluació dels indicadors estratègics

Obtenir avaluació actual detallada dels indicadors estratègics		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/detailedStrategicIndicators/ CurrentEvaluation	**/api/strategicIndicators/ qualityFactors/current**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació actual	
	400 Bad Request	**500 Internal Server Error**

Taula 44. Obtenir avaluació actual detallada dels indicadors estratègics

Obtenir avaluació actual detallada d'un indicador estratègic		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/detailedStrategicIndicators/ currentEvaluation/{id}	**/api/strategicIndicators/{id}/ qualityFactors/current**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació actual	

	400 Bad Request	**500 Internal Server Error**

Taula 45. Obtenir avaluació actual detallada d'un indicador estratègic

Obtenir avaluació històrica detallada dels indicadors estratègics		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/detailedStrategicIndicators/ historicalData/	**/api/strategicIndicators/ qualityFactors/historical**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació històrica	
	400 Bad Request	**500 Internal Server Error**

Taula 46. Obtenir avaluació històrica detallada dels indicadors estratègics

Obtenir avaluació històrica detallada d'un indicador estratègic		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/detailedStrategicIndicators/ historicalData/{id}	**/api/strategicIndicators/{id}/ qualityFactors/historical**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació històrica	
	400 Bad Request	**500 Internal Server Error**

Taula 47. Obtenir avaluació històrica detallada d'un indicador estratègic

Obtenir predicció de l'avaluació detallada dels indicadors estratègics		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/detailedStrategicIndicators/ predictionData/	**/api/strategicIndicators/quality Factors/prediction**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en la predicció de l'avaluació	
	400 Bad Request	**500 Internal Server Error**

Taula 48. Obtenir predicció de l'avaluació detallada dels indicadors estratègics

Obtenir predicció de l'avaluació detallada d'un indicador estratègic		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/detailedStrategicIndicators/ predictionData/{id}	**/api/strategicIndicators/{id}/ qualityFactors/prediction**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK

	Error en la predicció de l'avaluació	
	400 Bad Request	**500 Internal Server Error**

Taula 49. Obtenir predicció de l'avaluació detallada d'un indicador estratègic

Obtenir avaluació detallada dels factors que composen un indicador estratègic		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/qualityFactors/ currentEvaluation/{id}	**/api/strategicIndicators/{id}/ qualityFactors/metrics/current**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació històrica	
	400 Bad Request	**500 Internal Server Error**

Taula 50. Obtenir avaluació detallada dels factors que composen un indicador estratègic

Obtenir avaluació històrica detallada dels factors que composen un indicador estratègic		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/qualityFactors/ historicalData/{id}	**/api/strategicIndicators/{id}/ qualityFactors/metrics/historical**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació històrica	
	400 Bad Request	**500 Internal Server Error**

Taula 51. Obtenir avaluació històrica detallada dels factors que composen un indicador estratègic

Obtenir predicció detallada de l'avaluació dels factors que composen un indicador estratègic		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/qualityFactors/ predictionData/{id}	**/api/strategicIndicators/{id}/ qualityFactors/metrics prediction**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en la predicció de l'avaluació	
	400 Bad Request	**500 Internal Server Error**

Taula 52. Obtenir predicció detallada de l'avaluació dels factors que composen un indicador estratègic

Factors de qualitat

Obtenir tots els factors de qualitat		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/qualityFactors/getAll	**/api/qualityFactors**

Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció dels factors de qualitat	
	400 Bad Request	**500 Internal Server Error**

Taula 53. Obtenir tots els factors de qualitat

Obtenir avaluació de les mètriques que composen un factor		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/metrics/currentEvaluation /{id}	**/api/qualityFactors/{id}/ metrics/current**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació	
	400 Bad Request	**500 Internal Server Error**

Taula 54. Obtenir avaluació de les mètriques que composen un factor

Obtenir avaluació històrica de les mètriques que composen un factor		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/metrics/historicalData/{id}	**/api/qualityFactors/{id}/ metrics/historical**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació històrica	
	400 Bad Request	**500 Internal Server Error**

Taula 55. Obtenir avaluació històrica de les mètriques que composen un factor

Obtenir predicció de l'avaluació de les mètriques que composen un factor		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/metrics/predictionData/{id}	**/api/qualityFactors/{id}/ metrics/prediction**
Paràmetres	prj	prj
Respostes	Correcte	
	200 OK	200 OK
	Error en la predicció de l'avaluació	
	400 Bad Request	**500 Internal Server Error**

Taula 56. Obtenir predicció de l'avaluació de les mètriques que composen un factor

Productes

Crear nou producte		
	Inicial	**Nou**
Mètode	POST	POST
URI	/api/newProduct	**/api/products**
Paràmetres	-	-
Cos	name description logo projects	name description logo projects
Respostes	Correcte	
	202 Accepted	**201 Created**
	Error en les noves dades del producte	
	409 Conflict	409 Conflict
	Error genèric	
	405 Method Not Allowed	**500 Internal Server Error**

Taula 57. Crear nou producte

Modificar producte		
	Inicial	**Nou**
Mètode	POST	PUT
URI	/api/updateProduct	**/api/products/{id}**
Paràmetres	-	-
Cos	id name description logo projects	**name** **description** **logo** **projects**
Respostes	Correcte	
	202 Accepted	**200 OK**
	Error en les noves dades del producte	
	409 Conflict	409 Conflict
	Error genèric	
	405 Method Not Allowed	**500 Internal Server Error**

Taula 58. Modificar producte

Esborrar producte		
	Inicial	**Nou**
Mètode	POST	DELETE
URI	/api/deleteProduct	**/api/products/{id}**
Paràmetres	id	**-**
Respostes	Correcte	
	202 Accepted	**200 OK**
	Error genèric	
	405 Method Not Allowed	**500 Internal Server Error**

Taula 59. Esborrar producte

Obtenir avaluació d'un producte		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/products/currentEvaluation/ {id}	**/api/products/{id}/current**
Paràmetres	-	**-**
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació	
	200 OK	**500 Internal Server Error**

Taula 60. Obtenir avaluació d'un producte

Obtenir avaluació detallada d'un producte		
	Inicial	**Nou**
Mètode	GET	GET
URI	/api/products/detailedCurrent Evaluation/{id}	**/api/products/{id}/projects/ current**
Paràmetres	-	**-**
Respostes	Correcte	
	200 OK	200 OK
	Error en l'obtenció de l'avaluació	
	200 OK	**500 Internal Server Error**

Taula 61. Obtenir avaluació detallada d'un producte