UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAT D'INFORMÀTICA DE BARCELONA

GRAU EN ENGINYERIA INFORMÀTICA (GEI)
SOFTWARE ENGINEERING

Definition and execution of security protocols through conceptual modeling languages

Memory

Author: Gerard Pradas San José

Supervisor: Ernest Teniente Lopez

Cosupervisor: Xavier Oriol Hilari

16th January 2018

Abstract

Nowadays, the definition and execution of security protocols is done direct by means of the implementation through imperative languages.

Indeed, the protocol may suffer from conceptual errors that are difficult to detect by looking at the code, as well as the typical bugs that may appear during its development. On the other hand, today there are conceptual modeling languages that allow to define, in a clear and unambiguous way, processes, information systems and software components.

In this thesis, we want to use these conceptual modeling languages to define security protocols. In this way, it's pretended that the definition of a security protocol consists only on the unambiguous definition of the protocols own concepts.

Taking advantage of the non-ambiguity of these descriptions, it is also pretended to build an environment in which these protocols can be executed in an automated way.

Resum

Actualment, la definició i execució de protocols de seguretat es realitza directament mitjançant la implementació a través de llenguatges imperatius.

De fet, el protocol pot patir errors conceptuals que són difícils de detectar observant el codi, així com el típic bugs que pot aparèixer durant la seva programació. D'altra banda, avui hi ha llenguatges de modalització conceptuals que permeten definir, de manera clara i sense ambigüitats, processos, sistemes d'informació i components software.

En aquesta tesi, volem utilitzar aquests llenguatges de modalització conceptual per definir protocols de seguretat. D'aquesta manera, es pretén que la definició d'un protocol de seguretat consisteix únicament en la definició inambigua dels conceptes propis del protocol.

Aprofitant la no ambigüitat d'aquestes descripcions, també es pretén construir un entorn en què aquests protocols es puguin executar de forma automàtica.

Resumen

Hoy en dia, la definición y ejecución de los protocolos de seguridad se realiza de forma directa mediante la implementación a través de lenguajes imperativos.

De hecho, el protocolo puede sufrir errores conceptuales que son difíciles de detectar al mirar el código, así como el típico bugs que puede aparecer durante su desarrollo. Por otro lado, hoy en día existen lenguajes de modelización conceptual que permiten definir, de manera clara e inequívoca, procesos, sistemas de información y componentes de software.

En esta tesis, queremos utilizar estos lenguajes de modelización conceptual para definir protocolos de seguridad. De esta forma, se pretende que la definición de un protocolo de seguridad consista solo en la definición inequívoca de los propios conceptos de los protocolos.

Aprovechando la no ambigüedad de estas descripciones, también se pretende construir un entorno en el que estos protocolos se puedan ejecutar de forma automática.

Contents

List of Tables

List of Figures

Glossary

agile Agile software development describes a set of values and principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams.. 7–9, 38

API An application programming interface (API) is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components.. 8, 11, 12, 36, 39

Authorization Code Grant The Authorization Code Grant is a flow where the browser receives an Authorization Code from OAuth server and sends this to the web app. The web app will then interact with OAuth server and exchange the Authorization Code for an access_token, and optionally an id_token and a refresh_token. The web app can now use this access_token to call the API on behalf of the user.. viii, 9, 11, 17, 18, 26, 27, 38, 48

back-end The backend developer is one that is on the server side, that is, this person is responsible for languages such as PHP, Python, .Net, Java, etc., is one that is responsible for interacting with databases, verify session management of users, mount the page on a server, and from this "serve" all the views that the FrontEnd creates, that is, one as a backend is responsible more than anything for the manipulation of data.. 35

bug Is a problem causing a program to crash or produce invalid output. i–iii, 1, 3

classic Classic methodology is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall) through several phases, typically: requirements analysis resulting in a software requirements specification, software design, implementation, testing, integration, if there are multiple subsystems, deployment and maintenance. . 7

Client Credentials Grant In Client Credentials Grant a non interactive client, can directly ask OAuth server for an access_token, by using its client credentials (client id and client secret) to authenticate. In this case the token represents the non interactive client itself, instead of an end user.. vii, viii, 9, 10, 15, 19, 20, 23, 38, 48

framework A framework establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community.. 8, 32, 35, 36

front-end The frontend are all those technologies that run on the client's side, that is, all those technologies that run on the side of the web browser, generalizing more than anything in three languages, Html, CSS and JavaScript.. 35

Gradle Gradle is an open-source build automation system that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven for declaring the project configuration.. 35

Implicit Grant The Implicit Grant is similar to the Authorization Code Grant, but the main difference is that the client app receives an access_token directly, without the need for an authorization_code. This happens because the client app, which is typically a JavaScript app running within a browser, is less trusted than a web app running on the server, hence cannot be trusted with the client_secret. Also, in the Implicit Grant, no refresh tokens for are returned, for the same reason.. 48

JSON Is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value). It is a very common data format used for asynchronous browser–server communication, including as a replacement for XML in some AJAX-style systems.. 35

library A software library is a suite of data and programming code that is used to develop software programs and applications. It is designed to assist both the programmer and the programming language compiler in building and executing software.. 2, 3, 6, 9, 12–18, 31, 32, 34–36, 39, 47, 48

OAuth 2.0 OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 supersedes the work done on the original OAuth protocol created in 2006. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.. 1–3, 5, 6, 8–13, 15–20, 27, 36, 38, 39, 47, 48

Resource Owner Grant The Resource Owner Grant can be used directly as an authorization grant to obtain an access token, and optionally a refresh token. This grant should only be used when there is a high degree of trust between the user and the client and when other authorization flows are not available.. vii, viii, 9, 10, 16, 17, 22, 23, 48

scrum Scrum is a framework for managing work with an emphasis on software development.. 8

Twitter Bootstrap Bootstrap is a free and open-source front-end web framework for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many web frameworks, it concerns itself with front-end development only.. 35

volere Is considered to be the most usable and accessible template. It is a distillation of hundreds of requirements documents, and serves as a guide to writing your specifications.. 12

Acronyms

AWS Amazon Web Services. 36

BPMN Business Process Model and Notation. 4, 6, 19, 23, 27, 34, 35, 38, 39, 49

GUI Graphical User Interface. 8

IDE Integrated Development Environment. 7, 8

LOPD Ley Organica de Protección de Datos. 46

MPI Information Modelling and Processing. 32

OCL Object Constraint Language. 4, 6, 19, 34, 35, 39

SQL Structured Query Language. 32

UML Unified Modeling Language. 4, 6, 19

Chapter 1

Introduction

Nowadays, the definition and execution of security protocols is done direct by means of the implementation of the protocol through imperative languages. This praxis tends to be slow, and susceptible to errors.

Indeed, the protocol may suffer from conceptual errors that are difficult to detect by looking at the code, as well as the typical bugs that may appear during its development. On the other hand, today there are conceptual modeling languages that allow to define, in a clear and unambiguous way, processes, information systems and software components.

The models of these languages allow engineers to abstain from the implementation details and concentrate on the fundamental concepts of the process / system to develop.

The security protocols that we are going to deal with, are the authorization OAuth 2.0 protocols. OAuth 2.0 protocols are the most used way to protect APIs and are used by companies around the world such as Google, Microsoft, Facebook ...

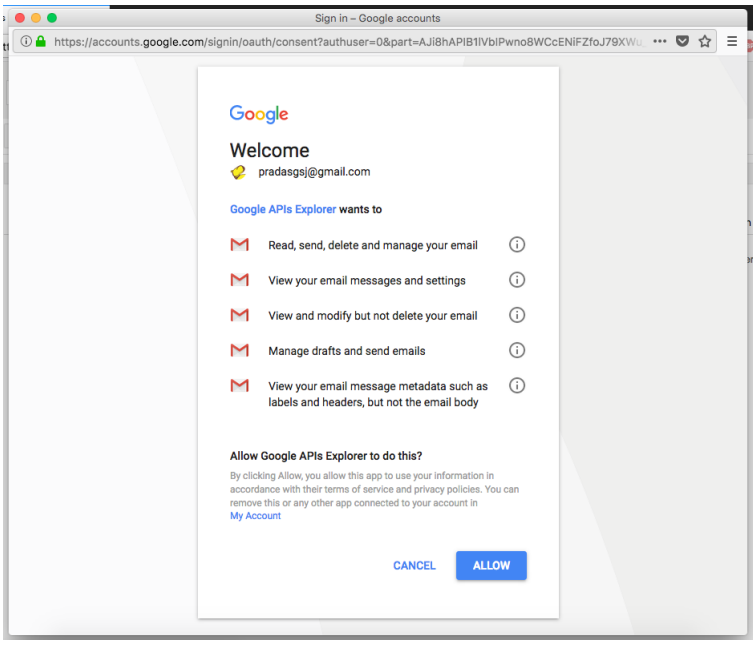

Figure 1.1: Google APIs explorer OAuth 2.0

In this thesis, we want to use these conceptual modeling languages to define security protocols. In this way, it's pretended that the definition of a security protocol consists only on the unambiguous definition of the protocols own concepts.

Our solution is to create a library such that, given an application that wants to consume data from a protected resource in an OAuth 2.0 server, our library will receive as an input, the description of the protocol to use to access that resource, and the library will be capable of executing the protocol and return the resource data to the main application.

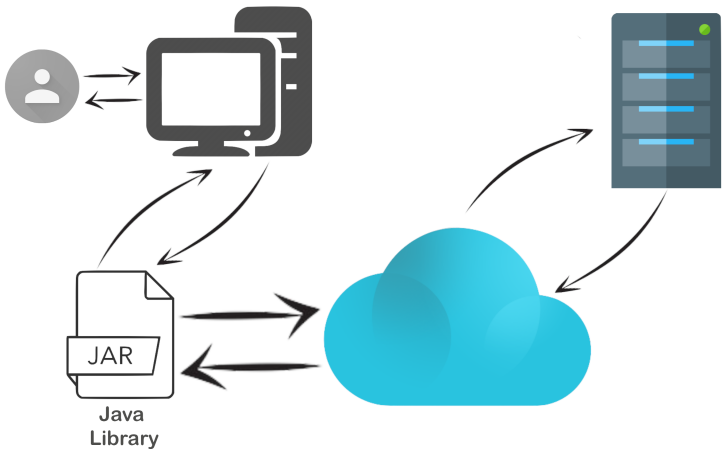

Figure 1.2: Components scheme

In this way, if the server decides to change the protocol, the only thing we must do is to change the input of the library.

1.1 Context

This bachelor thesis is a modalty "A" project from the specialization of Software Engineering done at the Faculty of Informatics of Barcelona. This project is supervised by Ernest Teniente and co-supervised by Xavier Oriol, which arises from the need to apply in an automated way, several security protocols.

1.2 Formulation of the problem

The main problem that has decided to treat in this project is when a programmer wants to use an authorization protocol, such as OAuth 2.0, it falls to the programmer that is who implements this protocol, and this practice tends to have errors. That is why we want to create a library, made from the conceptual modeling of these protocols, which means that they are free of bugs, and can be executed automatically.

1.2.1 Objectives

The main objective is to design and develop a library that allows to automate the whole process when a programmer wants to implement the OAuth 2.0 protocol in one of its programs. In this way, the programmer will only have to import that library, configure a few basic parameters, and the library will take care of everything. From this main objective, we can extract several more specific objectives:

- Improve the time a developer uses to authorize their applications using an OAuth 2.0 protocol.

- Make the execution of the protocol based on its conceptual model, not its imperative implementation by code.

1.3 Scope

The scope of this thesis, focuses on the whole development process of a library to automate several OAuth 2.0 protocols for authorization, and depends on the time it takes to achieve this point, we proceed to add more OAuth 2.0 protocols.

The whole process entails, first to perform the conceptual modeling of the security protocol, which will be performed using BPMN and UML. For the definition of each BPMN task it will be done using the OCL.

Once the protocol is modeled correctly, we will proceed to the translation of the BPMN flows and OCL contracts into executable code. In this step we will research if we can use developed software to save time, but the assumption is not assured.

The next step, we will join the two parts in a library, so that later, a programmer can use it, just configuring a few basic parameters.

Finally, we will develop a simple web application, so we can verify that the work done works correctly.

1.3.1 Possible obstacles

The obstacles that are most at risk when carrying out this project would be the following:

Many technological incompatibilities

In the development of this project, many different tools are integrated, and one possible drawback is that they are not compatible. And this obstacle may appear very early or at the end of the project, which will probably force us to modify part of the project that was previously validated.

Poor information on the web

This project has the risk of being less documented on the internet as it deals with more abstract topics such as conceptual modeling and very specific topics.

Limited time

The final thesis has a limited time, and this time can play against if the planning does not follow its course correctly. For this reason, if we do not do a good planning that contemplates some deviations, it can be a risk for the project

1.4 Stakeholders

The stakeholders are those people or entities interested and involved in the development of the project. In this project we can distinguish two types of stakeholders, the direct ones, which

are the project team and the software developers, and the indirect ones, which are basically the internet users.

1.4.1 Project Team

Here we refer to the team in charge of the project. From the system design to the validation of their requirements.

1.4.2 Software Developers

These are the the people the thesis is headed for. They will use the result of this thesis to simplify the part of developing the implementation of a OAuth 2.0 protocol.

1.4.3 Internet Users

The internet users are one of our stakeholders because their credentials will be safer in those sites that take profit of this project

Chapter 2

Contextualization

2.1 State-of-the-art

Developers who are currently developing software using the OAuth 2.0 authentication method need to implement these protocols by hand, and each time they interact with an OAuth 2.0 for a specific service, it has its own configuration. Currently there are OAuth 2.0 libraries for several programming languages, but these libraries only facilitate the use of the protocol, and we intend to be able to automate the entire process.

As we have said, a security protocol ends up being a process (activities + data + modification of data per task).

2.1.1 Linking Data and BPMN Processes to Achieve Executable Models

Linking Data and BPMN Processes to Achieve Executable Models[1] is a report in which is described a formal way to describe a process (link data and processes conceptually, based on adopting UML class diagrams to represent data, and BPMN to represent the process and OCL to represent the task definitions) and execute that process.

This report is the base of this thesis and we will take advantage from a library that this report uses, that works for executing described contracts in OCL. So our project will focus on these aspects:

- Execution of operations in the order indicated in the BPMN.
- Invocation of remote services.
- Calls by value between tasks.

2.2 Methodology and rigor

2.2.1 Work methodology

The working methodology of this project follows some aspects related to the agile methodology, but by the nature of the project, it follows mainly a classic methodology.

More concretely, the agile aspects that this project follows[2] are:

- **Sprint planning**, which means that we meet every 2 weeks, to discuss, revise and validate the work of the last 2 weeks, and to plan the work for the next 2 weeks.
- **Sprint review**, is a meeting that is done at the end of every sprint, to validate the work done.
- **Iterative and incremental development**,that means that in every iteration (or sprint), the result is an ended valuable piece of the project, produced by the result of past iterations.

2.2.2 Monitoring tools

For monitoring the project evolution we mainly use 3 artifacts. The fisrts two are the meetings in the "sprint planning" (that we take every 2 weeks), and the email.

These two are used for the communication of the team, and have served mainly for the monitarization of the design and specification.

To monitor the evolution of the implementation a version control has been used, more concretely Git and GitHub, which is a Web-based Git version control repository hosting service.

2.2.3 Developing tools

During the software development different tools are used, each one for a specific functionality. The tools that have been used in this project are the following:

GenMyModel

GenMyModel is a modeling platform in the cloud for software architects, developers and business process analysts. Is the one of the few online modeling platform that allows to create BPMN 2.0 diagrams.

IntelliJ IDEA

IntelliJ IDEA is a Java Integrated Development Environment (IDE) for developing computer software.

IntelliJ IDEA is one of the most complete IDE and it is developed by JetBrains, and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition.

Sequel Pro

Sequel Pro is a fast, easy-to-use Mac database management application for working with MySQL databases. Sequel Pro gives you direct access to your MySQL Databases on local and remote servers.

PhpStorm

PhpStorm is a commercial, cross-platform IDE for PHP built on JetBrains' IntelliJ IDEA platform.

PhpStorm provides an editor for PHP, HTML and JavaScript with on-the-fly code analysis, error prevention and automated refactorings for PHP and JavaScript code.

Postman

Postman is a powerful GUI platform to make development faster and easier, from building requests through testing, documentation and sharing.

2.2.4 Validation methodology

The validation of this project, as we are using some artifacts from the agile framework Scrum, will be done during the sprint review of each iteration. Additionally, to validate the implementation, an OAuth 2.0 server will be required to verify its works correctly.

Also, during the sprint review, the tasks that have not been validated, would remain pending for the next iteration.

Chapter 3

Requirement analysis

This section presents the analysis of requirements to clearly and precisely define the functionalities and restrictions that must be taken into account when starting to develop the library.

As in our case the project is a proof of concept that an stakeholder has already defined, all functionalities are defined from the beginning, unlike what could happen in agile methodologies.

3.1 Functional requirements

The functional requirements define the functionalities or services that the software must have. The functionalities that will be implemented in our library will be that it works with several OAuth 2.0 protocols, more specifically the Client Credentials Grant, Resource Owner Grant and Authorization Code Grant protocols, through their conceptual definitions.

These protocols work in the following way:

Client Credentials Grant

In Client Credentials Grant a non interactive client, can directly ask OAuth server for an access_token, by using its client credentials (client id and client secret) to authenticate. In this case the token represents the non interactive client itself, instead of an end user.

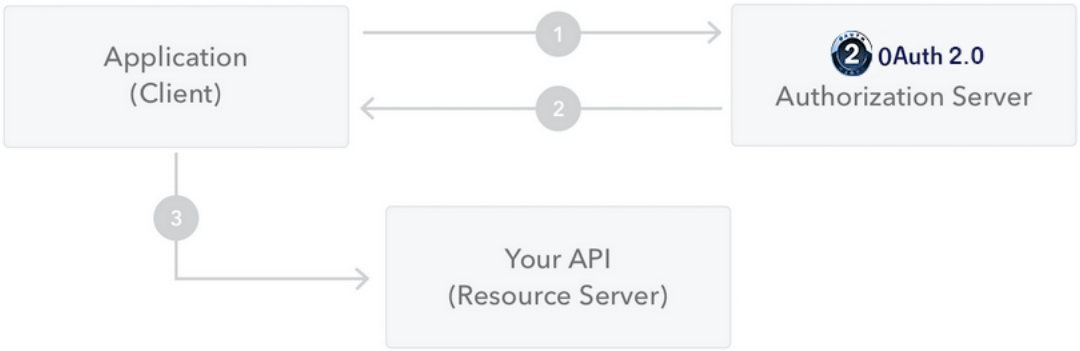

Figure 3.1: Client Credentials Grant schema

1. The application authenticates with OAuth 2.0 server using its Client Id and Client Secret.
2. OAuth 2.0 server validates this information and returns an access_token.
3. The application can use the access_token to call the on behalf of itself.

Resource Owner Grant

The Resource Owner Grant can be used directly as an authorization grant to obtain an access token, and optionally a refresh token. This grant should only be used when there is a high degree of trust between the user and the client and when other authorization flows are not available.

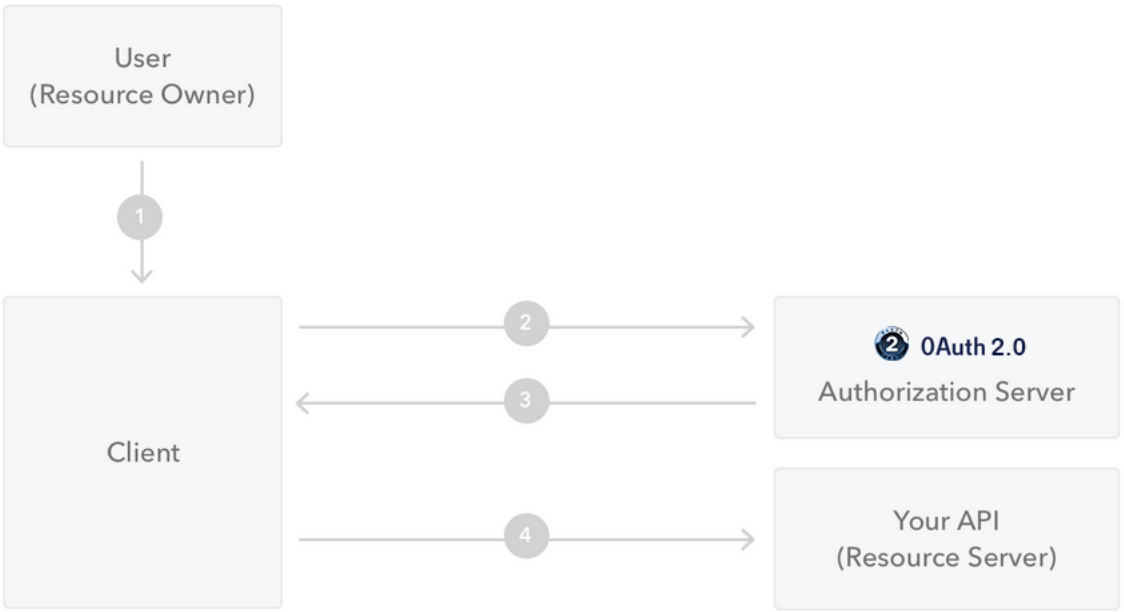

Figure 3.2: Resource Owner Grant schema

1. The end user enters the credentials into the client application.
2. The client forwards the credentials to the OAuth 2.0 server.
3. OAuth 2.0 server validates the information and returns an access_token, and optionally a refresh_token.
4. The client can use the access_token to call the on behalf of the end user.

Authorization Code Grant

The Authorization Code Grant is a flow where the browser receives an Authorization Code from OAuth server and sends this to the web app. The web app will then interact with OAuth server and exchange the Authorization Code for an access_token, and optionally an id_token and a refresh_token. The web app can now use this access_token to call the API on behalf of the user.

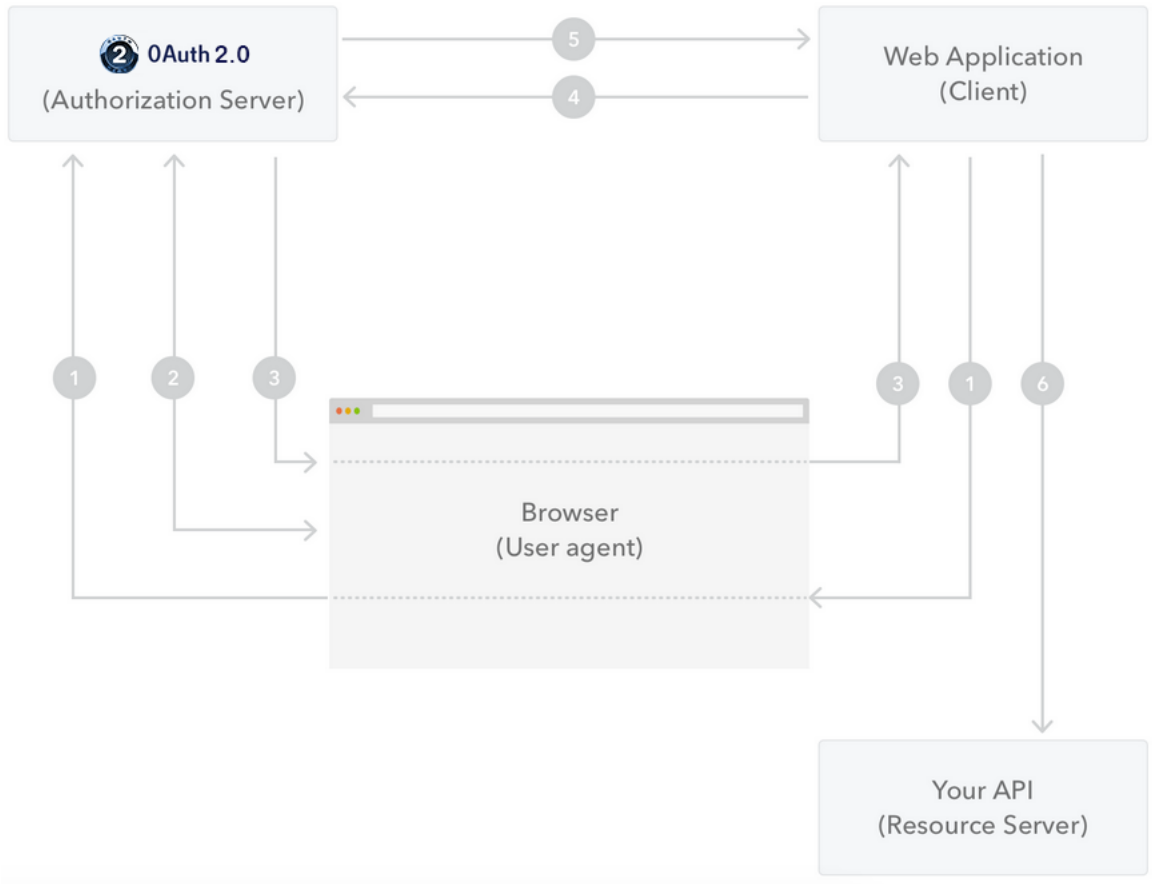

Figure 3.3: Authorization Code Grant schema

1. The web app initiates the flow and redirects the browser to OAuth 2.0 server, so the user can authenticate.

2. OAuth 2.0 server authenticates the user (via the browser). The first time the user goes through this flow a consent page will be shown where the permissions are listed that will be given to the Client (for example: post messages, list contacts, and so forth).

3. OAuth 2.0 server redirects the user to the web app (specifically to the redirect_uri, as specified in the /authorize request) with an Authorization Code in the querystring (code).

4. The web app sends the Authorization Code to Auth0 and asks to exchange it with an access_token (and optionally an id_token and a refresh_token). When making this request, the web app authenticates with OAuth 2.0 server, using the Client Id and Client Secret.

5. OAuth 2.0 server authenticates the web app, validates the Authorization Code and responds back with the token.

6. The web app can use the access_token to call the on behalf of the user.

3.2 Non-functional requirements

The non-functional requirements represent general and crosscutting characteristics of the application to be implemented, and are defined in the Volere[3] template.

Usability and Humanity Requirements: Ease of Use Requirements

- **Description:** this describes your client's aspirations for how easy it is for the intended users of the product to operate it. The product's usability is derived from the abilities of the expected users of the product and the complexity of its functionality.
- **Justification of the requirement:** the library has to be very simple to use, so that only configuring the minimum possible parameters, can be as automated the maximum as possible.
- **Acceptance criteria:** the number of instructions that the library user has to do to use the library will be counted, and that count can't be bigger than 5.

Maintainability and Support Requirements: Maintenance Requirements

- **Description:** a quantification of the time necessary to make specified changes to the product.
- **Justification of the requirement:** the library has to be very flexible, since it is a proof of concept and it is an unfinished product, so it has to be easy to modify the work done in the future.

- **Acceptance criteria:** certain development patterns have been followed to focus on the maintenance of the library.

Performance Requirements: Scalability or Extensibility Requirements

- **Description:** this specifies the expected increases in size that the product must be able to handle. As a business grows (or is expected to grow), our software products must increase their capacities to cope with the new volumes.
- **Justification of the requirement:** the library has to be scalable, since it is a proof of concept and is an unfinished product, this will not include all the protocols defined by OAuth 2.0, but the library must be ready so they can be added later.
- **Acceptance criteria:** certain development patterns have been followed to focus on the scalability of the library.

Security Requirements: Privacy Requirements

- **Description:** specification of what the product has to do to ensure the privacy of individuals about whom it stores information. The product must also ensure that all laws related to privacy of an individual's data are observed.
- **Justification of the requirement:** the library can not have security issues that allow third parties to obtain the tokens that are saved, without the authorization of the library user.
- **Acceptance criteria:** the code has been encapsulated, so that the library user can only execute the few methods we allow.

Chapter 4

Software requirements specification

This chapter presents the complete description of the implemented library behavior. The functionalities of the library and the conceptual model of the protocols to be implemented are specified.

4.1 Functionalities

In this section we will define the functionalities that this project requires in more detail.

4.1.1 Client Credentials Grant functionalities

Title	Consume a web resource using the Client Credentials Grant authorization protocol of OAuth 2.0.
Main stakeholder	Library user.
Precondition	Library user has valid credentials of the server OAuth 2.0 to which he will ask authorization.
Trigger	The library user wants to consume an web resource under an OAuth 2.0 server.
Main stage of success	1. Library user creates a Client Credentials Grant instance and populates the necessary parameters. 2. The library is in charge of obtaining and storing the access_token, and making the request to the resource, to return the info to the library user.
Extension of the the use case	-
Alternative courses of the use case	2.a Access_token is available. 2.a.1 The library is responsible for searching the saved access_token, and,making the request to the resource, to return the info to the library user.

Table 4.1: Client Credentials Grant use case.

4.1.2 Resource Owner Grant functionalities

Title	Consume a web resource using the Resource Owner Grant authorization protocol of OAuth 2.0.
Main stakeholder	Library user.
Precondition	Library user has valid credentials of the server OAuth 2.0 to which he will ask authorization.
Trigger	The library user wants to consume an web resource under an OAuth 2.0 server.
Main stage of success	1. Library user creates a Resource Owner Grant instance and populates the necessary parameters. 2. Library user asks the lib if he has a valid token. 3. The library responds that it has no valid token. 4. Library user configures the username and password in the library. 5. The library is in charge of obtaining and storing the access_token and the refresh_token, and making the request to the resource, to return the info to the library user.
Extension of the the use case	-
Alternative courses of the use case	4.a The user credentials are incorrect. 4.a.1 The library user writes the username or password incorrectly. 4.a.2 Library returns an exception warning of the error. 5.a A valid token is available. 5.a.1 The library is responsible for searching the saved access_token, and making the request to the resource, to return the info to the library user. 5.b An expired access_token is available. 5.b.1 The library is responsible for searching the saved refresh_token, obtains and stores a new access_token, and makes the resource request, to return the info to the library user.

Table 4.2: Resource Owner Grant use case.

4.1.3 Authorization Code Grant functionalities

Title	Consume a web resource using the Authorization Code Grant authorization protocol of OAuth 2.0.
Main stakeholder	Library user.
Precondition	Library user has valid credentials of the server OAuth 2.0 to which he will ask authorization.
Trigger	The library user wants to consume an web resource under an OAuth 2.0 server.
Main stage of success	1. Library user creates a Authorization Code Grant instance and populates the necessary parameters. 2. Library user asks the lib if he has a valid token. 3. The library responds that it has no valid token. 4. Library user configures the username and password, as well as the ids of the browser elements, in the library. 5. The library is in charge of obtaining and storing the access_token and the refresh_token, and making the request to the resource, to return the info to the library user.
Extension of the the use case	-
Alternative courses of the use case	4.a The user credentials or the ids of the html elements of the server OAuth 2.0 are incorrect. 4.a.1 The library user writes the username, password or the id of the html elements of the server OAuth 2.0 incorrectly. 4.a.2 Library returns an exception warning of the error. 5.a A valid token is available. 5.a.1 The library is responsible for searching the saved access_token, and making the request to the resource, to return the info to the library user. 5.b An expired access_token is available. 5.b.1 The library is responsible for searching the saved refresh_token, obtains and stores a new access_token, and makes the resource request, to return the info to the library user.

Table 4.3: Resource Owner Grant use case.

4.1.4 Alter the definition of an OAuth 2.0 protocol.

Title	Consume a web resource using the Authorization Code Grant authorization protocol of OAuth 2.0.
Main stakeholder	Developer.
Precondition	The developer has a set of correct definition files.
Trigger	The developer wants to update a definition of an OAuth 2.0 protocol.
Main stage of success	1. The developer replaces the definition files of the protocol in a repository. 2. The library user download the new protocol definitions and replace them with the current ones. 3. The library already detects the new definitions and starts to work with them.
Extension of the the use case	-
Alternative courses of the use case	-

Table 4.4: Alter definition use case.

4.2 Conceptual models

In this section we will show the conceptual model of the OAuth 2.0 protocols that are implemented in this project, more specifically for each one they will show:

- **Data conceptual schema (in UML):** these are the objects that the client needs to store in order to make the protocol work. To simplify the automation, we will save the information using the minimal data structures.
- **process flow (in BPMN):** is the flow of tasks that follows the protocol according to its definition. We have chosen to represent the process flow with the Business Process Model and Notation (BPMN) annotation since it is a language that is understood for both technical and non-technical people.
- **Description of each task from the process flow (in OCL):** are the definitions of each task. The OCL language has been chosen because it is a specification formal language.

4.2.1 Client Credentials Grant conceptual models

Data schema

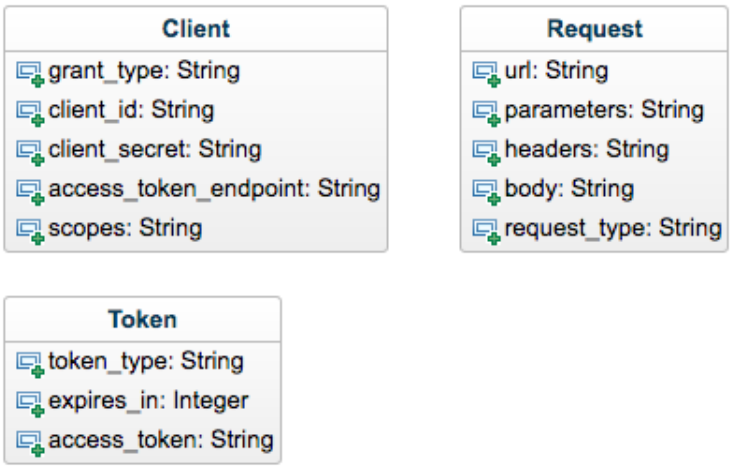

Figure 4.1: Client Credentials Grant data schema

For the conceptual data schema of Client Credentials Grant, we only need to save the basic information related to client, request, and token.

Process flow

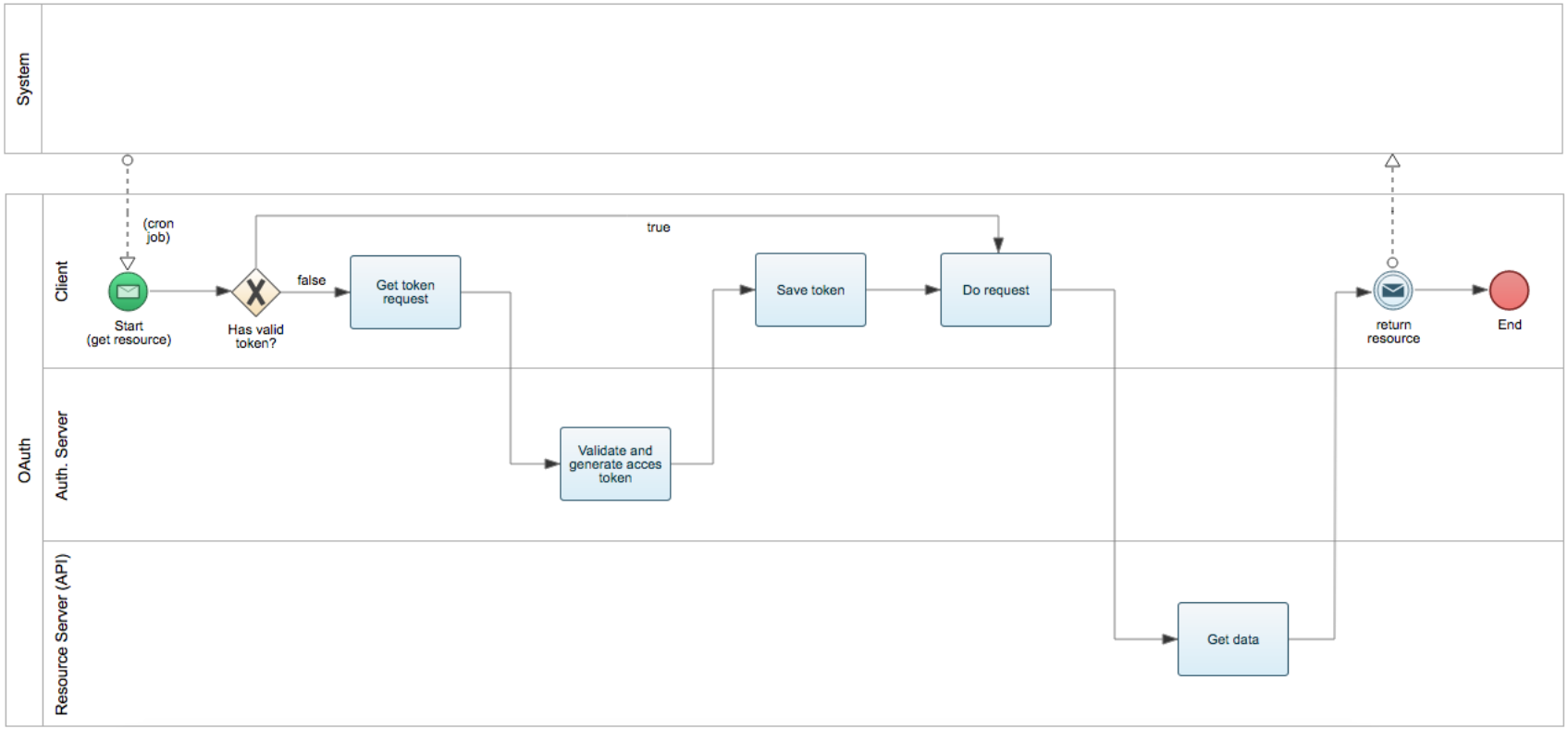

Figure 4.2: Client Credentials Grant process flow

As we see in the diagram, when the protocol receives a request to a resource, it search if it has a valid token, if so, it sends the request to the resource server, if not, it asks for a token to the server OAuth 2.0 with its credentials.

Task definition

```
context:    System::start(url:String, parameters:String,
                headers:String, body:String, type:String)
pre:
post:       Request.allInstances()->exist(
                r | r.oclIsNew() and r.url = url and
                r.parmeters = parameters and
                r.headers = headers and r.body = body and
                r.request_type = type
            )
---
context:    System::hasValidToken():Boolean
pre:
body:       result = Token.allInstances()
                ->exists(t | t.acces_token -> notEmpty())
---
context:    System::getTokenRequest() : TupleType(grant:String,
                client_id:String, client_secret:String,
                token_endpoint:String, scopes:String)
pre:
body:       let cli:Client = Client.allInstances()->select
                (c | c.client_id -> NotEmpty())
            in result =
                Tuple{grant = cli.grant,
                    client_id = cli.client_id,
                    client_secret = cli.client_secret,
                    token_endpoint = cli.token_endpoint,
                    scopes = cli.scopes}
---
context:    System::saveToken(accessToken:String,
                tokenType:String, expiresIn:String)
pre:
post:       Token.allInstances()->exists(
                t | t.oclIsNew() and
                t.acces_token = accessToken and
                t.token_type = tokenType and
                t.expires_in = expiresIn
            )
```

```
---
context:    System::doResourcetRequest() : TupleType(
                access_token:String, url:String,
                parameters:String, headers:String,
                body:String, request_type:String)
pre:
body:       let req:Request = Request.allInstances()->select
                (r | r.url -> NotEmpty())
            let tok:Token = Token.allInstances()->select
                (t | t.acces_token -> NotEmpty())
            Request.allInstances()-> reject
                (r | r.url -> NotEmpty())
            in result =
                Tuple{access_token = tok.access_token,
                    url = req.url,
                    parameters = req.parameters,
                    headers = req.headers,
                    body =  req.body,
                    request_type = req.request_type}
```

4.2.2 Resource Owner Grant conceptual models

Data schema

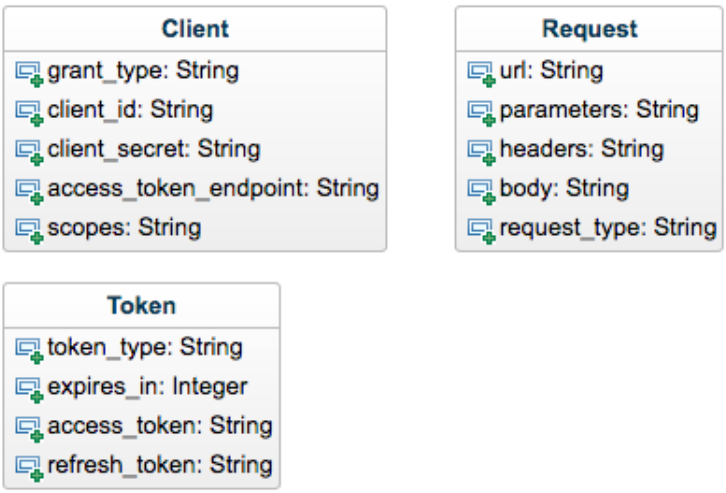

Figure 4.3: Resource Owner Grant data schema

For the conceptual data schema of Resource Owner Grant, in addition to all the information of the previous protocol, we will need to save the refresh token as an additional field in the token.

Process flow

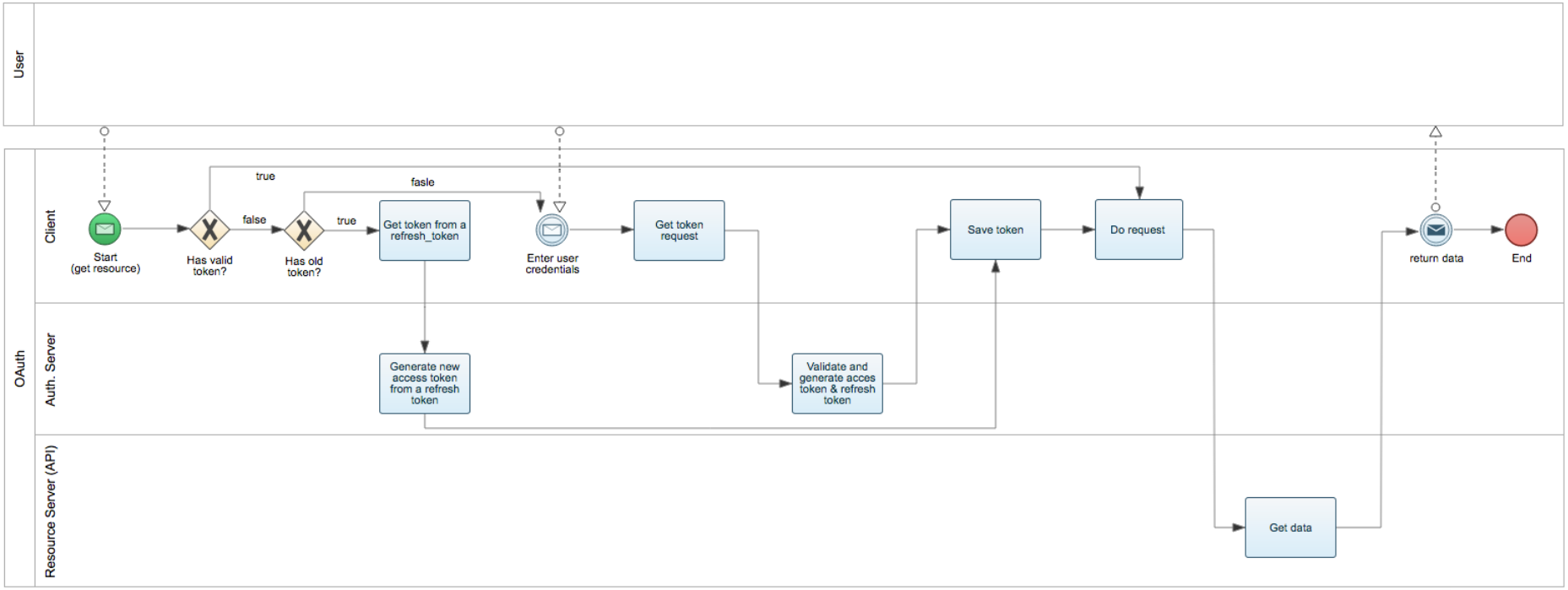

Figure 4.4: Resource Owner Grant process flow

The BPMN of Resource Owner Grant, is similar to the Client Credentials Grant process flow, with the difference that here we also need a username and password to obtain the token, and in case the token is no longer valid, we can request another token without the need to enter the user again thanks to the refresh token.

Task definition

```
context:    System::start(url:String, parameters:String,
                headers:String, body:String, type:String)
pre:
post:       Request.allInstances()->exist(
                r | r.oclIsNew() and
                r.url = url and
                r.parmeters = parameters and
                r.headers = headers and r.body = body and
                r.request_type = type
            )
---
context:    System::hasValidToken():Boolean
pre:
body:       result = Token.allInstances()
                ->exists(t | t.acces_token -> notEmpty())
---
context:    System::getOldToken() : TupleType(
                refresh_token:String, token_endpoint:String)
pre:
body:       let cli:Client = Client.allInstances()->select
                (c | c.client_id -> NotEmpty())
            let tok:Token = Token.allInstances()->select
                (t | t.acces_token -> NotEmpty())

            in result =
                Tuple{refresh_token = tok.refresh_token,
                    token_enpoint = cli.token_endpoint
                }
---
context:    System::getTokenRequest() : TupleType(grant:String,
                client_id:String, client_secret:String,
                token_endpoint:String, scopes:String)
pre:
body:       let cli:Client = Client.allInstances()->select
                (c | c.client_id -> NotEmpty())
            in result =
                Tuple{grant = cli.grant,
```

```
                client_id = cli.client_id,
                client_secret = cli.client_secret,
                token_endpoint = cli.token_endpoint,
                scopes = cli.scopes
            }
---
context:    System::saveToken(accessToken:String,
                tokenType:String, expiresIn:String)
pre:
post:       Token.allInstances()->exists(
                t | t.oclIsNew() and
                t.acces_token = accessToken and
                t.token_type = tokenType and
                t.expires_in = expiresIn
            )
---
context:    System::doResourcetRequest() : TupleType(
                access_token:String,
                url:String,
                parameters:String,
                headers:String,
                body:String,
                request_type:String)
pre:
body:       let req:Request = Request.allInstances()->select
                (r | r.url -> NotEmpty())
            let tok:Token = Token.allInstances()->select
                (t | t.acces_token -> NotEmpty())
            Request.allInstances()-> reject
                (r | r.url -> NotEmpty())

            in result =
                Tuple{access_token = tok.access_token,
                    url = req.url,
                    parameters = req.parameters,
                    headers = req.headers,
                    body =  req.body,
                    request_type = req.request_type
                }
```

4.2.3 Authorization Code Grant conceptual models

Data schema

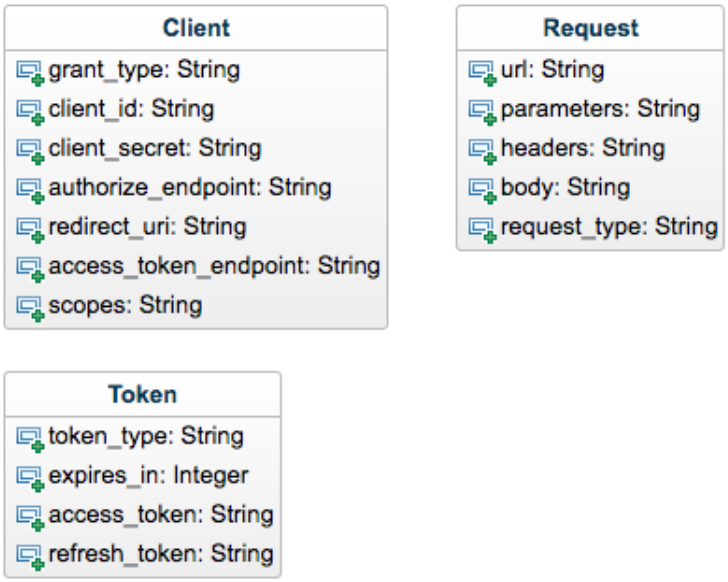

Figure 4.5: Authorization Code Grant data schema

For the conceptual schema of acg data, in addition to all the information of the previous protocol, we will need to save the authentication endpoint and the redirection uri.

Process flow

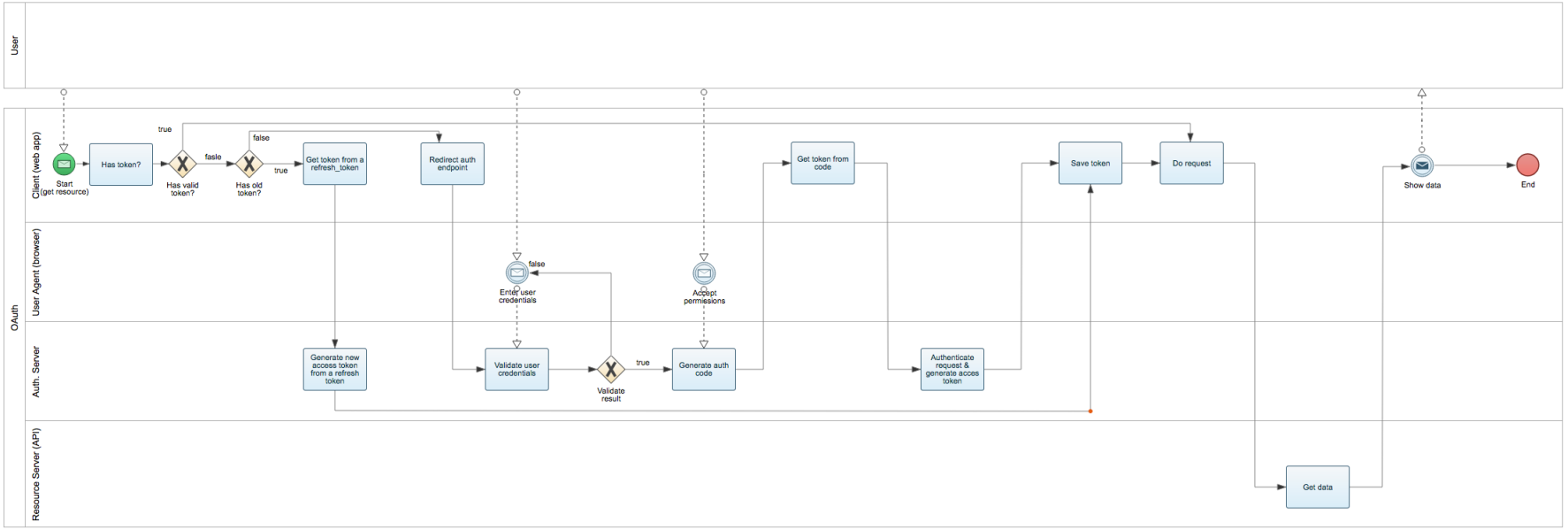

Figure 4.6: Authorization Code Grant process flow

The BPMN of glsacg is different from the others, since it needs to be used in a web application, because a web browser is involved. In this case we also need to login with a username and password, but this time we will not receive the token, but we will receive a code that we can change in the OAuth 2.0 server for the token. In this protocol we also have the possibility to update the access_token with the refresh_token.

Task definition

```
context:    System::start(url:String, parameters:String,
                headers:String, body:String, type:String)
pre:
post:       Request.allInstances()->exist(
                r | r.oclIsNew() and
                r.url = url and
                r.parmeters = parameters and
                r.headers = headers and r.body = body and
                r.request_type = type
            )
---
context:    System::hasValidToken():Boolean
pre:
body:       result = Token.allInstances()
                ->exists(t | t.acces_token -> notEmpty())
---
context:    System::getOldToken() : TupleType(
                refresh_token:String, token_endpoint:String)
pre:
body:       let cli:Client = Client.allInstances()->select
                (c | c.client_id -> NotEmpty())
            let tok:Token = Token.allInstances()->select
                (t | t.acces_token -> NotEmpty())

            in result =
                Tuple{refresh_token = tok.refresh_token,
                    token_enpoint = cli.token_endpoint
                }
---
context:    System::getAuthEndpoint() : TupleType(
                auth_endpoint:String, client_id:String,
                client_secret:String, scopes:String,
                redirect_uri:String)
pre:
body:       let cli:Client = Client.allInstances()->select
                (c | c.client_id -> NotEmpty())
            in result =
```

```
                Tuple{auth_endpoint = cli.authorize_endpoint,
                    client_id = cli.client_id,
                    client_secret = cli.client_secret,
                    scopes = cli.scopes,
                    redirect_uri = cli.redirect_uri
                }
---
context:    System::exchangeAuthCode() : TupleType(
                grant:String, client_id:String,
                client_secret:String, token_endpoint:String,
                scopes:String, code:String)
pre:
body:       let cli:Client = Client.allInstances()->select
                (c | c.client_id -> NotEmpty())
            in result =
                Tuple{grant = cli.grant,
                    client_id = cli.client_id,
                    client_secret = cli.client_secret,
                    token_endpoint = cli.token_endpoint,
                    scopes = cli.scopes
                }
---
context:    System::saveToken(accessToken:String,
                tokenType:String, expiresIn:String)
pre:
post:       Token.allInstances()->exists(
                t | t.oclIsNew() and
                t.acces_token = accessToken and
                t.token_type = tokenType and
                t.expires_in = expiresIn
            )
---
context:    System::doResourcetRequest() : TupleType(
                access_token:String,
                url:String,
                parameters:String,
                headers:String,
                body:String,
                request_type:String)
```

```
pre:
body:       let req:Request = Request.allInstances()->select
                (r | r.url -> NotEmpty())
            let tok:Token = Token.allInstances()->select
                (t | t.acces_token -> NotEmpty())
            Request.allInstances()-> reject
                (r | r.url -> NotEmpty())

            in result =
                Tuple{access_token = tok.access_token,
                    url = req.url,
                    parameters = req.parameters,
                    headers = req.headers,
                    body =  req.body,
                    request_type = req.request_type
                }
```

Chapter 5

Design

In this chapter we describe the design of the system and the technologies that have been used.

5.1 Software architecture

In this section the design and the system architecture of the library system is explained.

5.1.1 Database schema

The schema of the database is extracted from the previous conceptual models. So in our case we will use several databases, more specifically one for each protocol implemented.

5.1.2 Dessing patterns

When it comes to making the library we have used mainly 2 patterns, to improve the maintainability and scalability of the code, these patterns are the strategy pattern and the template template.

The **strategy pattern** is a behavioral software design pattern that enables selecting an algorithm at runtime. The strategy pattern:

- Defines a family of algorithms
- Encapsulates each algorithm
- Makes the algorithms interchangeable within that family

The **template method pattern** is a behavioral design pattern that defines the program skeleton of an algorithm in an operation, deferring some steps to subclasses. It lets one redefine certain steps of an algorithm without changing the algorithm's structure.

5.2 Technology

This section specifies the tools used for the development of the library.

When choosing a programming language, a preliminary study of the currently used technologies must be done, why are used and if they suit to our project.

In our case it was decided to develop the library in **Java**, because we used a library of the Information Modelling and Processing (MPI) research group written in Java, and it was the simplest thing to do.

The database management system to save the data managed by the library is **MySQL** and for the web application, to test the library, the **Spring Boot** framework is chosen since for simple projects it is very easy to use and comes with an embed Tomcat server.

5.2.1 Java

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture.

5.2.2 MySQL

MySQL is a relational database manager system based on Structured Query Language (SQL).

MySQL runs on virtually all platforms, including Linux, UNIX, and Windows. Although it can be used in a wide range of applications, MySQL is most often associated with web-based applications and online publishing.

5.2.3 Spring Boot

Spring is a very popular Java-based framework for building web and enterprise applications. Unlike many other frameworks, which focus on only one area, Spring framework provides a wide verity of features addressing the modern business needs via its portfolio projects.

Spring Boot aims to make it easy to create Spring-powered, production-grade applications and services with minimum fuss. It takes an opinionated view of the Spring platform so that new

and existing users can quickly get to the bits they need. You can use it to create stand-alone Java applications that can be started using ‘java -jar’ or more traditional WAR deployments.

Chapter 6

Development

In this chapter we explain how the process has been and the development of the project.

6.1 Library

The implementation of the library has focused on the execution of the process flow in BPMN, since the translation and execution of the OCL definitions have been done by the co-director of the project, Xavier Oriol.

Because in a typical case, a BPMN diagram is not meant to be some executable tasks attached to processes, we researched several libraries for the automation of the BPMN diagram, such as:

- **jBPT** (https://github.com/jbpt/codebase)
- **jBPM** (http://jbpm.jboss.org/)
- **Camuda** (https://camunda.com/products/bpmn-engine/)
- **Activiti** (https://www.activiti.org/)

And tried to apply the one that had a behavior that more resembled what we wanted to represent with our BPMN, that was jBPT, but given that the main use of BPMN is different from the one we are giving, none of the libraries ended up working at all and we ended up implementing our own representation of the BPMN process flow.

Our representation of the execution flow is based on three basic artifacts and their subclasses. These artifacts are:

- **Node:** it is the basic unit of the process flow and represents an executable task.

- **ProcessModel:** it is a set of nodes, with their navigabilities and basic functions.
- **Grant:** it is the definition of a protocol in BPMN. It contains an instance of a ProcessModel, and all the instances of the nodes that are necessary.

As we have used the strategy pattern together with the template pattern. These basic artifacts are interfaces, which are implemented by abstract classes, which at the same time, are implemented by their specific classes.

We have also created a factory class, which creates instances of the protocol to be used, to abstract the final user from the implementation of the library.

Another critical point of the library are pool switches that happen in the BPMN, since in our library they are translated as requests to a server, and we can not make these requests from OCL. Therefore we have to make a technological leap, and implement templates for each type of request that is made to the server.

6.2 Web Application

The back-end of the web application consists of only 3 routes, in which a single controller and 3 different views have been used. The biggest issue here was learning to use Gradle, which is a dependency manager for java.

For the front-end of the web, we used the Twitter Bootstrap framework, and to show the result of the request, as the server returns a string in JSON format, we used the vkBeautify library that improves the visualization of a string in JSON format.

Chapter 7

Validation

This chapter focuses on the validations that are done in the project.

7.1 Sprint Review Meeting

In the meetings at the end of each iteration, the work done is validated. For the specification of the OAuth 2.0 protocols there is a security expert present in the meeting.

7.2 OAuth 2.0 Server

To prove that the library works correctly with a real server, an EC2 test server of AWS has been created.

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

All the software necessary to create a web server with its database is installed and configured in our EC2 instance. More specifically, an Apache server with MySQL.

This server runs a web application to make the OAuth 2.0 server. This web application is in PHP using the Laravel[4] framework, since this framework has native libraries that implement all OAuth 2.0 protocols, preparing the server is relatively simple. It is also implemented a basic API to consume data.

Chapter 8

Management

This chapter shows the documentation related to the management of the project: general planning, temporary planning, economic management and a small analysis of sustainability and the laws.

8.1 Project planning

The length of the project is four months. The project began on September 12, 2017 and is expected to be delivered in mid-January 2018. In this project we want to accomplish a proof of concept, therefore, the planning will include the basic tasks for the realization of the project, but in case we don't face any issue and progress well, complementary tasks will be done for the completition of the proof of concept.

8.1.1 Resources

For the realization of this project, we will need 3 types of resources: personal resources, material resources and technological resources.

Personal resources

For the development of this project, we have had a team of four people:

Director and co-director of the project: two people in charge of both directing and structuring the project, and validating the work done.

Security expert: a person to consult doubts in the security protocols and in charge to validate the steps that involve security protocols.

Developer: a person in charge of the realization of the project.

Material resources

Computer: to develop the project, and the memory of the final thesis.

Work space: to be able to work on the project.

Paper, pencil and whiteboard: to make notes, make quick conceptual models or to clarify project doubts in meetings.

Technological resources

IntelliJ: the chossen IDE for the developing.

ShareLaTEX: web tool for doing documents on LaTEX.

MyGenmodel: web tool for doing digital conceptual modeling.

Email: the tool we use to comumnicate between the team.

8.2 Time management

8.2.1 Project iterations

As the meetings with the director are regular every 2 weeks, we can group the functionalities in iterations of two weeks, which is known in the agile methodologies, as sprints.

Iteration 1: Understanding and initial research

The first thing to do, will be to understand exactly what the project consists of, and to do research on the subject, existing bibliography or technologies involved.

Iteration 2: Conceptual modeling of Authorization Code Grant

During this iteration, the conceptual design of the authorization process will be done, by means of the Authorization Code Grant protocol of OAuth 2.0.

In this iteration is also included the learning of Business Process Model and Notation (BPMN), which is the type of diagram with which the design of the protocol will be made.

Iteration 3: Definitions of BPMN process in OCL

In this iteration, we will design another authorization process, this time between two computers, using the protocol Client Credentials Grant of OAuth 2.0.

This protocol is simpler than the one of the previous iteration, together with the already acquired knowledge, it will be faster to be done.

Also in this iteration, we will perform the definitions of the processes for the two diagrams, in Object Constraint Language (OCL) language.The learning curve of OCL will be smaller, since it has been given in a subject, and the definitions are not complex.

Iteration 4: Translation of OCL definitions to executable code

In iteration 4 the translations of the definitions of OCL will be done by logical derivation rules. This functionality is already developed from a previous work, but the work has to be understood (which is a complicated topic), and integrated into the project.

Iteration 5: Automation of the BPMN flow

The translation of the definitions to code will be completed in this iteration, also in iteration 5 will be developed the automation of the previous definitions, following the Business Process Model and Notation (BPMN) diagram. To perform the automation, a research to find the best solution have to be done, since in the market there are several libraries that seem that can work as we want, but the integration with the definitions, will be a difficult task.

Iteration 6: Automation of the BPMN flow

In the iteration 6, we will finish the automation of the BPMN diagram flow. As is considered one of the difficult tasks to deal with, we consider to finish it in two iterations.

Iteration 7: Creation of a simple WebApp

In iteration 6 will be develop a basic web application, where it will be integrated with the automation and the definitions already made, and where we will be able to prove that the whole authentication process really works and consumes data of an API.

Iteration 8: Project completion

The iteration 8 will be the last one of the project, and will be to finish the pending task of the project, as well as to refine other parts that can be improved.

Depending on the time that is dedicated to the finalization of the project, in this iteration, we will make the designs and definitions of the other protocols of OAuth 2.0. Optionally, we will also create a basic server to consume our API.

GEP

This block is performed during the first 4 iterations, and consists of the necessary documentation for the subject of GEP.

Memory and exposure

This block will be made in parallel with the project, starting from iteration 5, and is composed of the realization of all the memory, the presentation and final defense of the project.

8.2.2 Scheduling

Task	Dates	Estimated time
Iteration 1	12/09/2017 - 25/09/2017	25
Iteration 2	26/09/2017 - 09/10/2017	30
Iteration 3	10/10/2017 - 23/10/2017	30
Iteration 4	24/10/2017 - 06/11/2017	40
Iteration 5	07/11/2017 - 20/11/2017	40
Iteration 6	21/11/2017 - 04/12/2017	40
Iteration 7	05/12/2017 - 18/12/2017	35
Iteration 8	19/12/2017 - 02/01/2018	30
GEP	12/09/2017 - 30/10/2017	80
Memory and exposure	06/11/2017 - 12/01/2018	95
Total	12/09/2017 - 12/01/2018	445

Table 8.1: Hour estimation

Gantt chart

This is the resulting gantt chart of this project.

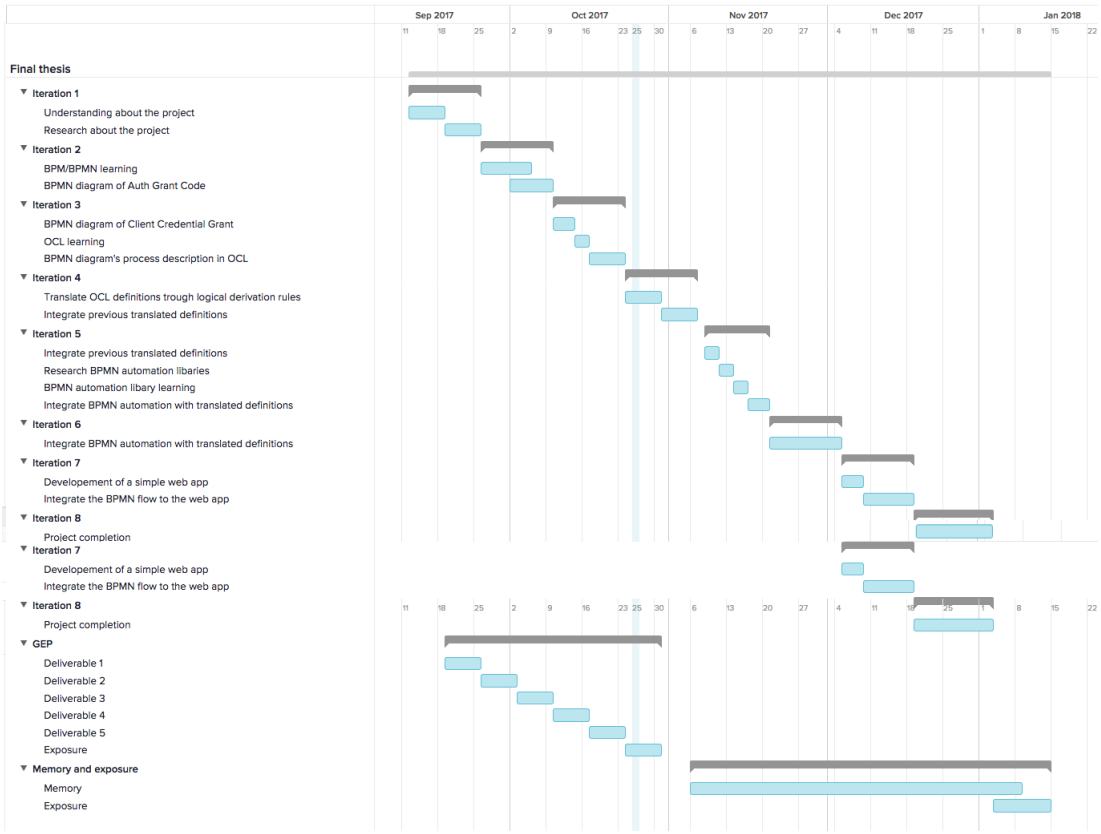

Figure 8.1: Gantt diagram

(an interactive and better quality chart can be found in `goo.gl/694wJn`)

8.3 Budget management

Once the project is planned, it is necessary to do an economic study to determine whether the project is viable or not, according to the budget and the costs of the necessary resources.

8.3.1 Identification of costs

Identification and estimation of project costs, is the previous step in the making of a budget. Therefore, first of all we have to make an estimate of wages. The wages has been obtained directly from the UPC website.[5]

Direct costs for the activity

Direct costs for the activity are disaggregated between the hourly estimation of every member, their job and their salary. Keep in mind that the direct costs will be low since most of the hours

Role	Salary	Hourly price	Members
University professor	2.563,97€	16€	2
Associate teacher (6 hours dedication)	1.354,34€	11,28€	1
Student	-	-	1

Table 8.2: Human resource's cost

are from the student, and these hours are at zero cost.

Activity	Estimated hours	Role	Members	Cost (€)
Iteration 1	1,5	University professor	1	24
	1,5	Associate teacher	1	16,92
	22	Student	1	0
	25			**40,92**
Iteration 2	1,5	University professor	1	24
	1,5	Associate teacher	1	16,92
	27	Student	1	0
	30			**40,92**
Iteration 3	2	University professor	2	64
	2	Associate teacher	1	22,56
	24	Student	1	0
	30			**86,56**
Iteration 4	2	University professor	2	64
	2	Associate teacher	1	22,56
	34	Student	1	0
	40			**86,56**
Iteration 5	2	University professor	2	64
	2	Associate teacher	1	22,56
	34	Student	1	0
	40			**86,56**
Iteration 6	2	University professor	2	64
	2	Associate teacher	1	22,56
	34	Student	1	0
	40			**86,56**
Iteration 7	1	University professor	2	36
	1	Associate teacher	1	11,28
	32	Student	1	0
	35			**47,28**
Iteration 8	1	University professor	1	16
	1	Associate teacher	1	11,28
	28	Student	1	0
	30			**27,28**
GEP	80	Student	1	0
Memory and exposure	95	Student	1	0
TOTAL	**445**			**502,64**

Table 8.3: Direct costs

Indirect costs

Apart from the direct costs that come from labor, the indirect costs caused by the project must be taken into account. These indirect costs include everything necessary for the project to be carried out.

- **Work space.** In order to carry out the project we do not need any special work space since we can work in it from any place. For this reason we have made the assumption that 50% of the work will be done at home and the other 50% of the work will be done in the library. So, in this section, only a percentage of the home expenses will be taken into account. This percentage will be the 5% of the total price , which cost 1200 €.
- **Transport.** In this section, we include the cost of transportation from home to university. As the trips are short, most times they will be on foot, but for reasons of comfort or weather conditions, it is expected that of the total of the monthly trips, 25% will be on public transport. For this reason, the cost of public transport will be taken into account.
- **Internet connection.** An internet connection is necessary because some tools need it and also to be able to search on the web. In this section only the cost of the internet at home will be taken into account. At home we are 4 people and the bandwidth used for the project will be 75% of a person.
- **Hardware amortization.** The hardware to develop the project will be a personal computer. This computer costs 1,800 euros, and its useful life is 10 years. A 70% use is expected during the duration of the project.
- **Software.**As all the software used is free or we can obtain a free license (respecting the use contract), no cost will be taken into account in the software section.

Resource	Duration	Price	Percentage	Cost
Work space	4 months	1.200€ / month	5%	240€
Transport	4 months	9,95€ (T-10 1 zone)	50%	19,9€
Internet connection	4 months	40€ / month	18,75%	30€
Hardware amortization	4 months	1.800€ / 10 years	70%	42€
Software	4 months	0€	-	0€
TOTAL				**331,90€**

Table 8.4: Indirect costs

Contingency costs

As in any project, a contingency cost has to be estimated to correct possible errors in the estimates of the previous costs. The contingency cost is generally 15% of the direct and indirect costs.

Concept	Total cost	Percentage	Contingency cost
Direct cost	502,64€	15%	75,40€
Indirect cost	331,90€	15%	49,79€
TOTAL			**125,19€**

Table 8.5: Contingency costs

Risk costs

We can face several types of economic incidents, for this reason, we have to include a section of incidence costs, to protect us in case any incident happens. These incidents are:

- **The need for more meetings.** It may happen that more meetings are needed with the experts, from those initially planned and this fact would increase the estimated budget.If we calculate the hour price of the 3 experts, a total of 43.28 euros comes out. Due to the nature of the project, there is a probability of 20% on average of this happening in each sprint.

- **Computer failure.** In case the computer is damaged, it would have to be replaced, because of the age it has, the repair would be expensive and would not be worth it. If this happened, the new computer would not be a high-end one like the current one, so the cost would be lower. The estimated probability of computer failure during the development of the project will be 2.5%.

Incident	Total cost	Probability	Incidence cost
More meetings	346,24€	20%	69,25€
Computer failure	700€	2,5%	17,50€
TOTAL			**86,75€**

Table 8.6: Risk costs

Final budget

Finally, all the costs calculated previously for the realization of the budget are added.

Concept	Cost
Direct costs	502,64€
Indirect costs	331,90€
Contingency costs	125,19€
Risk costs	86,75€
TOTAL	**1.046,48€**

Table 8.7: Final budget

8.3.2 Control management

To control expenses, a constant inspection will be made to detect possible deviations as soon as possible in order to act quickly. This inspection consists of recalculating the direct and indirect costs in each iteration, and in case the spending trend is greater than that estimated in several iterations, it will be assessed whether these increases are covered with the contingency costs. In case the expenses are greater than the contingency cost, actions will be taken.

8.4 Sustainability

This section shows the study of sustainability related to the project. The following table presents the results of the study in the form of a matrix, proposed by the GEP course guide.

	PPP	Useful life	Risks
Environmental	9	20	-1
Economic	8	20	-3
Social	7	17	0
Sustainability Rank	24	57	-4
	77		

Table 8.8: Sustainability matrix

8.4.1 Economic

An evaluation of the costs of the project has been done, both material and human resources, taking into account the cost of repairs and / or substitutions of the material and of temporary deviations in any of the deliveries within the unforeseen item.

It is necessary to emphasize that in a project, what increases the price is the direct cost of human resources, and in this project the majority of hours are student, for this reason, the economic viability is assured.

For these reasons, economic viability is assessed with an 8.

8.4.2 Social

The goal of the project is to automate a recurring task that developers have to deal with often. With this automation, we will improve the life of the developers when using our library, due to not having to continually develop the authentication mechanisms.

In addition, as our library does not contain bugs, it is also a headache less for the developer, since he is assured that this library will not produce errors later.

For these reasons, social viability is assessed with an 7.

8.4.3 Environmental

During the development of the project, only personal computers have been used, therefore, the environmental impact will only be the electricity. The vast majority of the diagrams have also been made in digital format, so that no excess paper has been spent.

For these reasons, environmental viability is assessed with an 8.

8.5 Laws and regulations

On this section the laws and regulations that affect this project are ex- plained, but this project is not affected by any law, since it does not store any kind of sensitive data.

Even we store the tokens, those are not affected by the Ley Organica de Protección de Datos (LOPD), so there is no special regulation applied to this project.

Chapter 9

Conclusions

In the last chapter we will be evaluate the work done and the future work and the personal assessment of the project.

9.1 Contributions

This project has ended as a proof of concept. This prototype will not be a final version, because it lacks of protocols to implement and has not been thoroughly tested, but it can be said that thanks to this project it has been possible to validate the possibility of automating protocols, based on its specification.

Therefore, it could be said that its main contribution is being able to execute a protocol (in our case OAuth 2.0), based on the definitions of specification, which means that if the protocol suffers from modifications, with replacing the old definitions with the new ones the protocol will be updated, without touching any code.

9.2 Objective achievement

Once the project has finished, we must check if the objectives that were marked at the beginning have been archived.

The main objective of designing and developing a library to automate the execution of the OAuth 2.0 protocols has not been fully achieved.

The objectives derived from the main objective were:

- Improve the time a developer uses to authorize their applications using an OAuth 2.0 protocol.

- Make the execution of the protocol based on its conceptual model, not its imperative implementation by code.

In principle with this library those objectives are achieved, but before we said that the main objective has not been fully achieved.

The specification and implementation of the Client Credentials Grant and Resource Owner Grant protocols has been achieved, but the implementation of the Authorization Code Grant protocol has not been achieved due to time constraints. However, the protocol has been specified and there is an approximate idea of how to make it work.

9.3 Personal assessment

Personally, this project has been satisfactory, since it has been a project that I started from 0, and I personally prefer that instead of inherit code, since it takes time to understand the work that is already done.

In addition, the subject of the project has been somewhat different from the typical information system, which is what I was looking for, and I have had the opportunity to be able to see the work that is done at the university.

It has been a great experience, because apart from being the first project of this kind, I have learned many technical skills that are not directly related to the specialty I've done, but I have found them very useful for any computer engineer.

Apart from the experience and being able to put into practice many things that I have learned in these 4.5 years of degre, I believe that I have grown as a person acquiring non-technical knowledge such as learning to defend the work done, always also from the self-criticism.

9.4 Future work

The first thing of future work of the library would be to finish the implementation that could not be achieved. Apart from finishing the project's objectives, there are also several objectives that were not defined at the beginning, but that are important for the future. These objectives are:

- Add the Implicit Grant protocol, since it is the last protocol that defines OAuth 2.0.
- Do deep testing on the library.

- Include a battery of unit tests.
- Include a parser for the definition of the protocol itself (not the BPMN tasks), since now it is done in a programmatic way.

9.5 Special thanks

Finally, I would like to thank all the people who have been involved in this project, either directly or indirectly.

To Ernest Teniente and Xavier Oriol for trusting me and giving me this opportunity to turn this project into my bachelor final thesis. Also to Juan Hernandez who has been the security expert and has been present at several of the meetings.

To all the people that I met from these bachelor degree years, of which some have already finished, others end these days, others still have some subjects left and others have abandoned. In particular I want to thank the people who have been closer in the different stages of the bachelor degree, and these are: Petru, Gerard, Juan, Xavier, Alex and Albert. And to Miguel Angel, with whom 10 years ago, we started in the world of computer science.

Finally to several coworkers, of the more than 2 years that I have been in the inLab as a scholar, for everything what they have taught me. Especially to Carla, Manel, Ricard, Adria and Josep.

Thanks to each of the people who have supported me.

Bibliography

[1] Montserrat Estañol Ernest Teniente Giuseppe De Giacomo, Xavier Oriol. Linking data and bpmn processes to achieve executable models. *CAiSE 2017: Advanced Information Systems Engineering*, pages 612–628, 2017.

[2] Jordi Pradel. Gestió de projectes de software: gestió àgil de projectes. Technical report, Universitat Politecnica Catalunya, 2015.

[3] Volere template. `http://www.volere.co.uk/template.htm`.

[4] Laravel framework. `https://laravel.com/`.

[5] Upc remunerable tables. `https://www.upc.edu/transparencia/ca/informacio-de-personal/`.

Appendix A

Web Application Screenshots

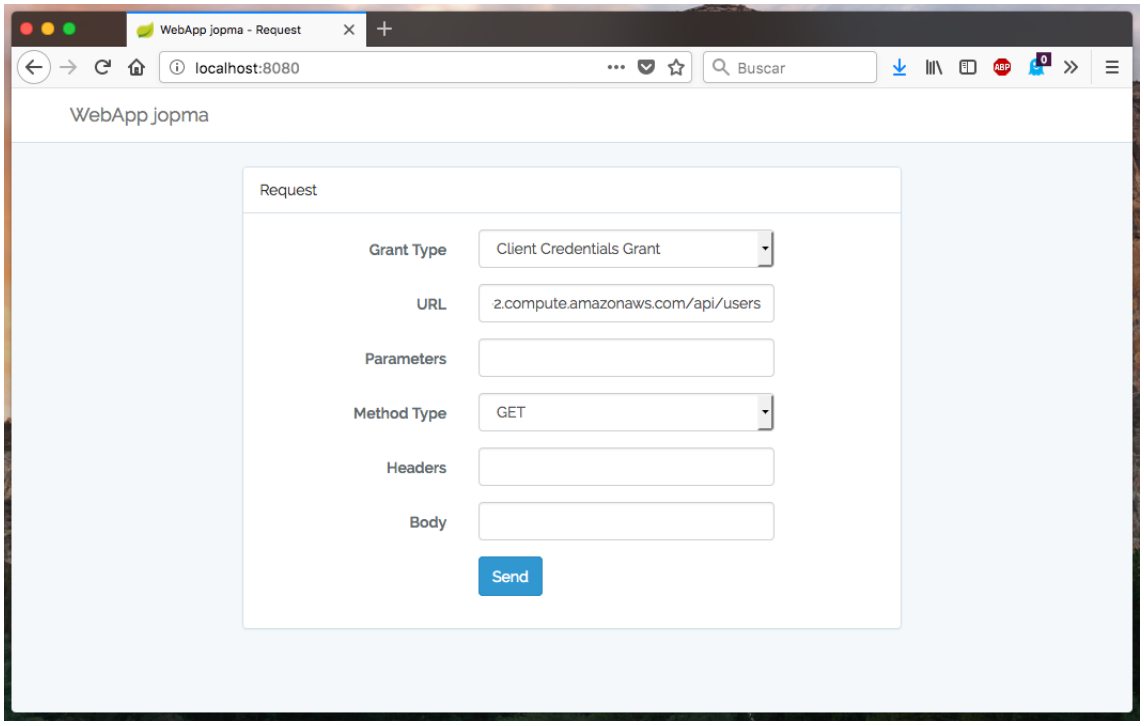

Figure A.1: Web Application request resource form

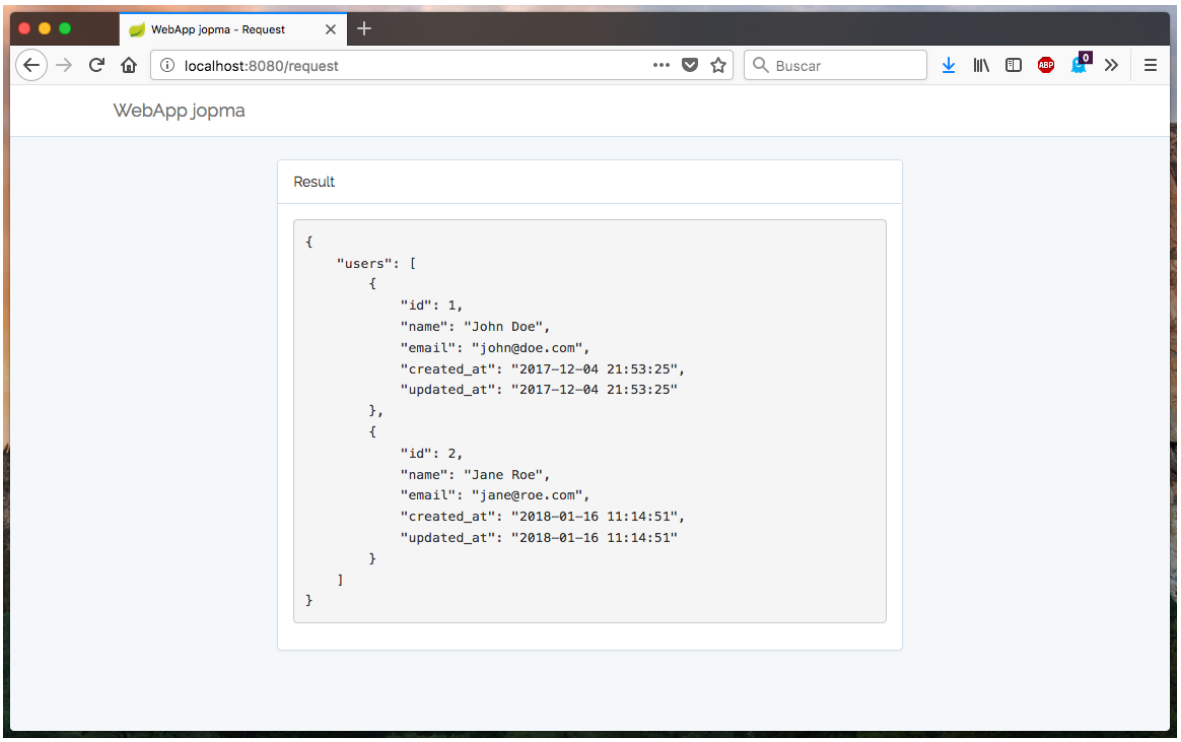

Figure A.2: Web Application request response

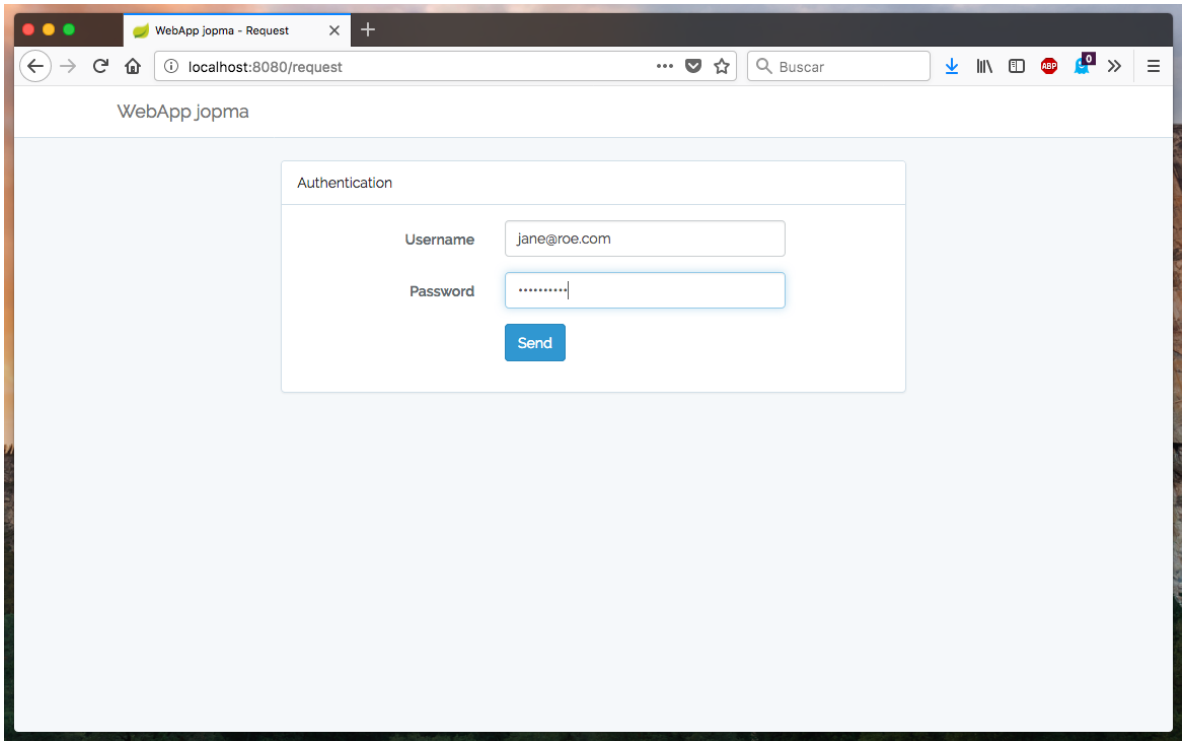

Figure A.3: Web Application authentication on an authenticated protocol

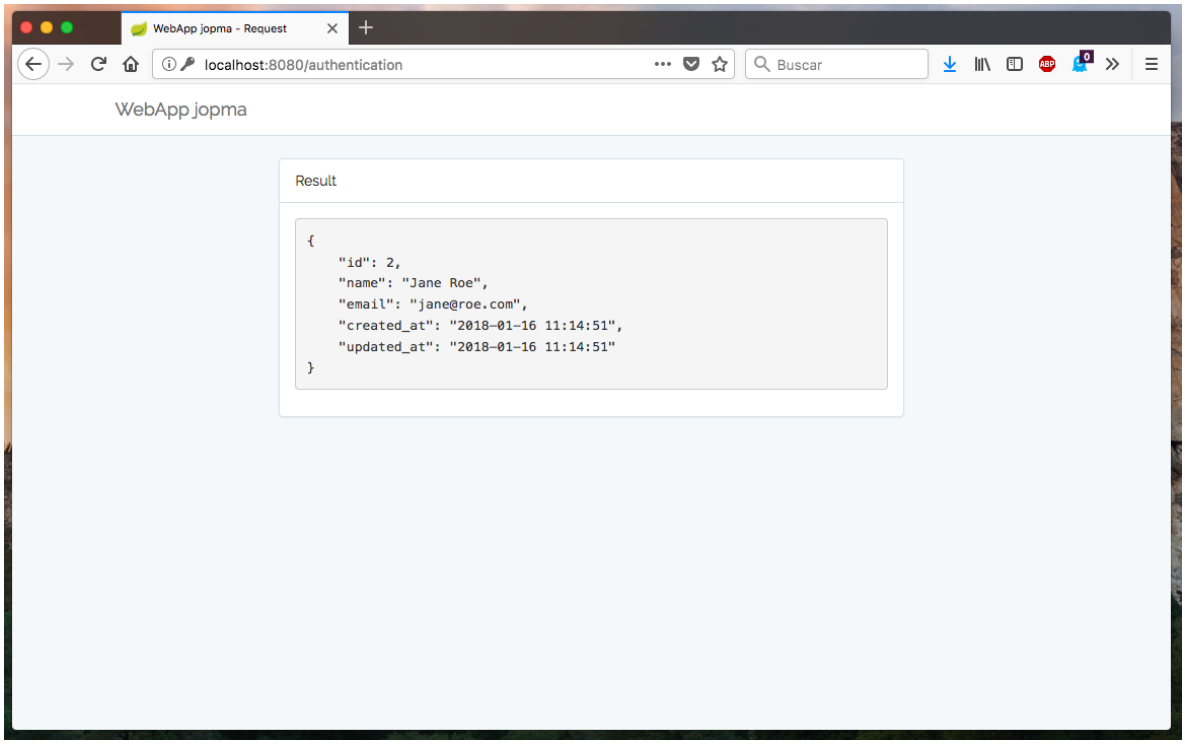

Figure A.4: Web Application request response on an authenticated protocol