



Escola Tècnica Superior d'Enginyeria  
de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# PROJECTE FINAL DE CARRERA

UNA API PER A LA PLATAFORMA DE  
CROWDFUNDING GOTEU.

AN API FOR THE GOTEU CROWDFUNDING  
PLATFORM

*Estudis: Enginyeria de Telecomunicació*

*Autor: Ivan Vergés Pascual*

*Director/a: Juan Jose Costa*

*Any: 2016*



# Índex general

Índex general.....	1
Índex de figures.....	3
Col·laboracions.....	4
Agraïments.....	5
Resum del Projecte.....	6
Resumen del Proyecto.....	7
Abstract.....	8
1. Introducció.....	9
1.1 Context del projecte.....	9
1.2 Objectius.....	11
1.3 Estructura de la memòria.....	11
2. Antecedents. El crowdfunding.....	13
2.1 El <i>crowdfunding</i> , orígens i actualitat.....	13
2.2 La plataforma Goteo. Particularitats.....	14
3. Disseny de la API.....	17
3.1 Introducció.....	17
3.2 Especificació.....	18
3.3 El model de recursos.....	20
3.3.1 Especificació de recursos.....	20
4. Implementació de la API.....	25
4.1 Models operacionals i protocols.....	25
4.1.1 Arquitectura del model: REST.....	25
4.1.2 Model operacional.....	28
4.1.3 Model d'autenticació.....	32
Altres mecanismes de control. Limit de peticions.....	38
4.1.4 Model de dades.....	39
4.2 Arquitectura del codi font.....	41
4.2.1 Components de programari.....	41
4.2.2 Entorn de desenvolupament.....	42
Comandes d'inicialització:.....	43
4.2.3 Control de qualitat.....	44
4.3 Estructura del programari.....	45

4.3.1 Estructura d'arxius:.....	46
Arbre d'arxius:.....	46
Primer nivell d'arxius.....	47
Segon nivell d'arxius.....	48
4.3.2 Mòdul <i>goteoapi</i> .....	49
4.3.3 Extensibilitat.....	51
Submòdul <i>goteoapi_reports</i> .....	52
Submòdul <i>goteoapi_digests</i> .....	52
4.4 Casos d'interès.....	54
4.4.1 <i>Framework</i> de treball: Flask.....	54
4.3.2 Estratègies de <i>cache</i> .....	57
4.4.3 Filtrat per coordenades geogràfiques.....	59
5. Documentació de la API.....	63
5.1 Swagger - Open Api Initiative.....	63
5.2 Estructura bàsica.....	64
5.2.1 Atributs principals.....	66
5.3 Aplicació.....	66
5.3.1 Integració en el codi.....	66
5.3.2 Integració en els tests.....	68
5.3.3 Auto generació de la documentació.....	68
6. Desplegament.....	71
7. Exemples i proves de funcionament.....	73
7.1 Goteo Stats.....	73
7.2 Mapes dinàmics.....	74
7.3 Demo ECF Labs.....	75
7.4 Matchfunding visualizations.....	76
7.5 Telegram @GoteoBot.....	77
Conclusions.....	79
Apèndix.....	81
Referències.....	83

# Índex de figures

Figura 1: API.....	6
Figura 2: www.goteo.org.....	9
Figura 3: Flux monetari del crowdfunding.....	13
Figura 4: Flux de multiplicació d'aportacions gràcies al capital regadiu.....	15
Figura 5: Campanyes de Capital Regadiu actives a Goteo durant els anys 2011 i 2016.....	15
Figura 6: API com a capa intermèdia en l'accés a dades.....	17
Figura 7: HTTP en la pila del model OSI.....	28
Figura 8: Primera línia de text en el protocol HTTP.....	29
Figura 9: Petició i resposta en HTTP.....	29
Figura 10: Autenticació a nivell d'aplicació.....	33
Figura 11: Autenticació a nivell d'usuari.....	33
Figura 12: Flux i capçaleres intercanviades entre client i servidor en un procés d'autenticació Basic.....	34
Figura 13: Flux de peticions en un procés d'autenticació OAuth 2.0 implícit.....	37
Figura 14: Procés de cache a l'API usant Redis.....	58
Figura 15: Projectió de l'àrea d'un cercle en la superfície d'una esfera.....	59
Figura 16: Swagger-UI, aplicació de test d'APIs tipus REST.....	69
Figura 17: Aspecte de la documentació de l'API un cop formatada.....	70
Figura 18: Esquema de la configuració en producció de l'API.....	71
Figura 19: Aplicació Goteo Stats.....	73
Figura 20: Exemple de geolocalització usant Google Maps.....	74
Figura 21: Demostració de com programar l'accés a l'API usant el framework AngularJS.....	75
Figura 22: Aplicació animada de les aportacions Capital Regadiu.....	76
Figura 23: Bot experimental per el programa de missatgeria instantània Telegram.....	77

# Col·laboracions



Fundación Goteo  
<http://fundacion.goteo.org>

# Agraïments

En primer lloc, vull agrair a la meua parella, Tracy Sirés per donar-me l'impuls necessari per completar aquesta memòria.

També a la Fundació Goteo per deixar-me presentar part de la feina realitzada durant els anys 2015 i 2016 com a base per aquest projecte.

A Pablo Castellano per la seva ajuda inicial en la programació en llenguatge Python.

A tot el personal de la UPC que m'ha animat i encoratjat a acabar el projecte per tal de tenir finalitzats els estudis. En especial a la Margarita Cabrera Bean (antiga cap d'estudis de la UPC), Josep Peguerols (Sotsdirector ETSETB de relacions amb l'empresa), Conchita Buesa (Secretaria Acadèmica ETSETB).

I, finalment, un agraïment especial al meu director de projecte, Juan José Costa per la seva bona disponibilitat i humor que han fet molt més senzilla la tasca de completar aquesta memòria.

# Resum del Projecte

Goteo és una plataforma web (<http://www.goteo.org>) impulsada per la fundació Goteo[55] (anteriorment *Fundación Fuentes Abiertas*) de finançament col·lectiu amb la missió d'impulsar projectes amb retorns oberts a la societat. Entren dins la categoria d'aquests retorns projectes de codi lliure, documentació oberta, projectes de caràcter social, etc.

Goteo és una plataforma de codi lliure que porta ja funcionant 5 anys i ara s'enfronta a nous reptes de futur. Un d'ells té a veure amb l'estratègia futura de la plataforma per adaptar-se als nous formats d'accés a Internet així com de la intercomunicació entre altres plataformes o serveis.

L'accés actual a la plataforma és la web principal, una interfície dissenyada només per a navegadors web i en un context determinat (per exemple l'adaptació a la visualització de dispositius mòbils és bastant limitada encara).

El problema principal a resoldre és doncs, dotar de major independència als diferents components de la plataforma: per una banda la lògica interna de procés (inclou models d'accés a la base de dades, seguretat, implementació de pagaments, etc.) i de l'altra la implementació concreta a un format d'accés per part de l'usuari final (en el cas actual la web). La separació completa de les dues parts de programari permetrà desvincular els diferents components de programari i simplificar-ne els respectius desenvolupaments.

L'objectiu del projecte serà doncs la implementació d'aquesta capa d'abstracció normalitzada d'accés a la plataforma, anomenada API (*Application Programming Interface*) que farà d'intermediari entre les aplicacions i la implementació dels serveis de la plataforma (*Fig. 1*).

També la documentació pública del funcionament d'aquesta API formen part d'aquest projecte, ja que es vol que qualsevol pugui generar les seves pròpies aplicacions seguint un manual clar de funcionament.

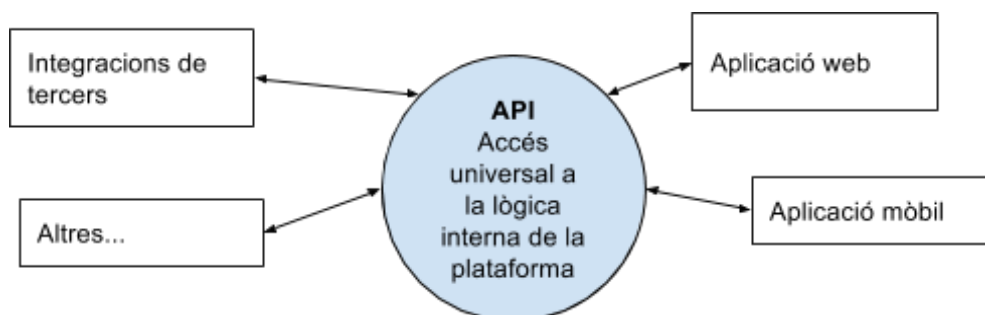


Figura 1: API

# Resumen del Proyecto

Goteo es una plataforma web (<http://www.goteo.org>) impulsada por la fundación Goteo[55] (anteriormente Fundación Fuentes Abiertas) de financiación colectiva con la misión de impulsar proyectos con retornos abiertos a la sociedad. Entran dentro de la categoría de estos retornos proyectos de código libre, documentación abierta, proyectos de carácter social, etc.

Goteo es una plataforma de código libre que lleva funcionando ya 5 años y que ahora se enfrenta a nuevos retos de futuro. Uno de ellos tiene que ver con la estrategia futura de la plataforma para adaptarse a los nuevos formatos de acceso a Internet así como la intercomunicación entre otras plataformas o servicios.

El acceso actual a la plataforma es la web principal, una interfaz diseñada sólo para navegadores web y en un contexto determinado (por ejemplo la adaptación de la visualización a dispositivos móviles es bastante limitada todavía).

El problema principal a resolver es pues, dotar de mayor independencia a los diferentes componentes de la plataforma: por un lado la lógica interna de proceso (incluye modelos de acceso a la base de datos, seguridad, implementación de pagos, etc.) y del otro la implementación concreta a un formato de acceso por parte del usuario final (en el caso actual la web). La separación completa de las dos partes de software permitirá desvincular los diferentes componentes de software además de simplificar los respectivos desarrollos.

El objetivo del proyecto será pues la implementación de esta capa de abstracción normalizada de acceso a la plataforma, denominada API (*Application Programming Interface*) que hará de intermediaria entre las aplicaciones y la implementación de los servicios de la plataforma (*Fig. 1*).

También la documentación pública del funcionamiento de esta API forman parte de este proyecto, puesto que se quiere que cualquiera pueda generar sus propias aplicaciones siguiendo un manual claro de funcionamiento.

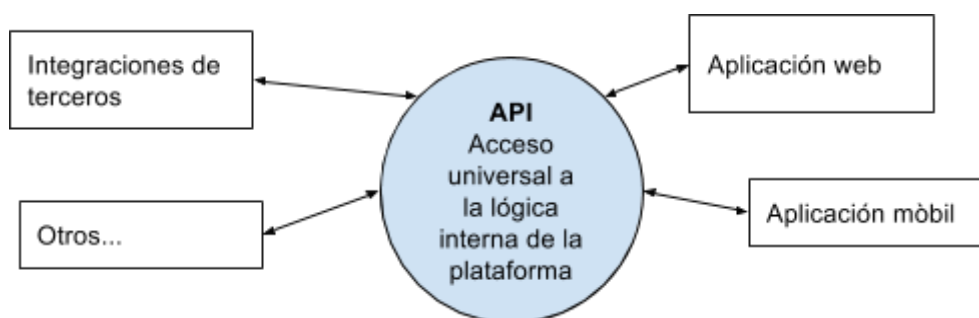


Figura 1: API

# Abstract

Goteo is a web platform (<http://www.goteo.org>) created by the Goteo Foundation[55] (previously Fundación Fuentes Abiertas) with the mission to “crowd-fund” (or collective financing) projects with the addition of collective returns to the society. By “collective returns” we include free software projects, open documentation, social projects, etc.

Goteo itself is a platform available as free software, it has been on-line for 5 years and now has to face new challenges for the future. One of them is the strategy of the platform for adapting the new formats of Internet access, as well as the intercommunication between other platforms or services.

The current access to the platform is the main website, an interface designed only for web browsers in some contexts (for example, the acces to the platform by using mobile devices is still quite limited in terms of user experience).

Therefore, the main problem to resolve will be to achieve a higher independence of the different platform components: on the one hand, the internal business logic (including database access model, security, payments implementations, etc.), and on the other, the specifics of the implementation of one access format for the final user (in our case, the Web). The complete separation of the two parts of software will allow to decouple the different components of software and significantly simplify their respective developments.

The aim of the project will be the implementation of this layer of standardized abstraction for accessing to the platform, we will call that an API (Application Programming Interface) and it will act as an intermediary agent between applications and the platform services (*Fig. 1*).

Additionally, we will include the public documentation of this API as part of this project, since we want anyone to be able to create his own applications following a clear operation’s manual.

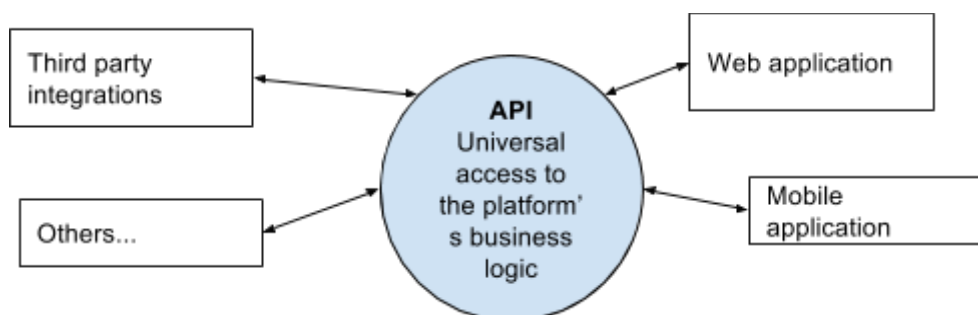


Figura 1: API

# 1. Introducció

Com a introducció, es farà una mirada a les tecnologies que condicionen el projecte, ja que aquest s'ha hagut d'adaptar a un sistema ja existent. Per altra banda, també s'establiran una llista d'objectius a complir durant el desenvolupament de l'aplicació proposada i, a més, un resum de com està organitzada aquesta memòria.

## 1.1 Context del projecte

La plataforma actual de Goteo està implementada amb tecnologies molt comunes actualment en el món del desenvolupament web. Es tracta d'una web a la que es pot accedir a través de qualsevol navegador en l'adreça <http://www.goteo.org/>[1].

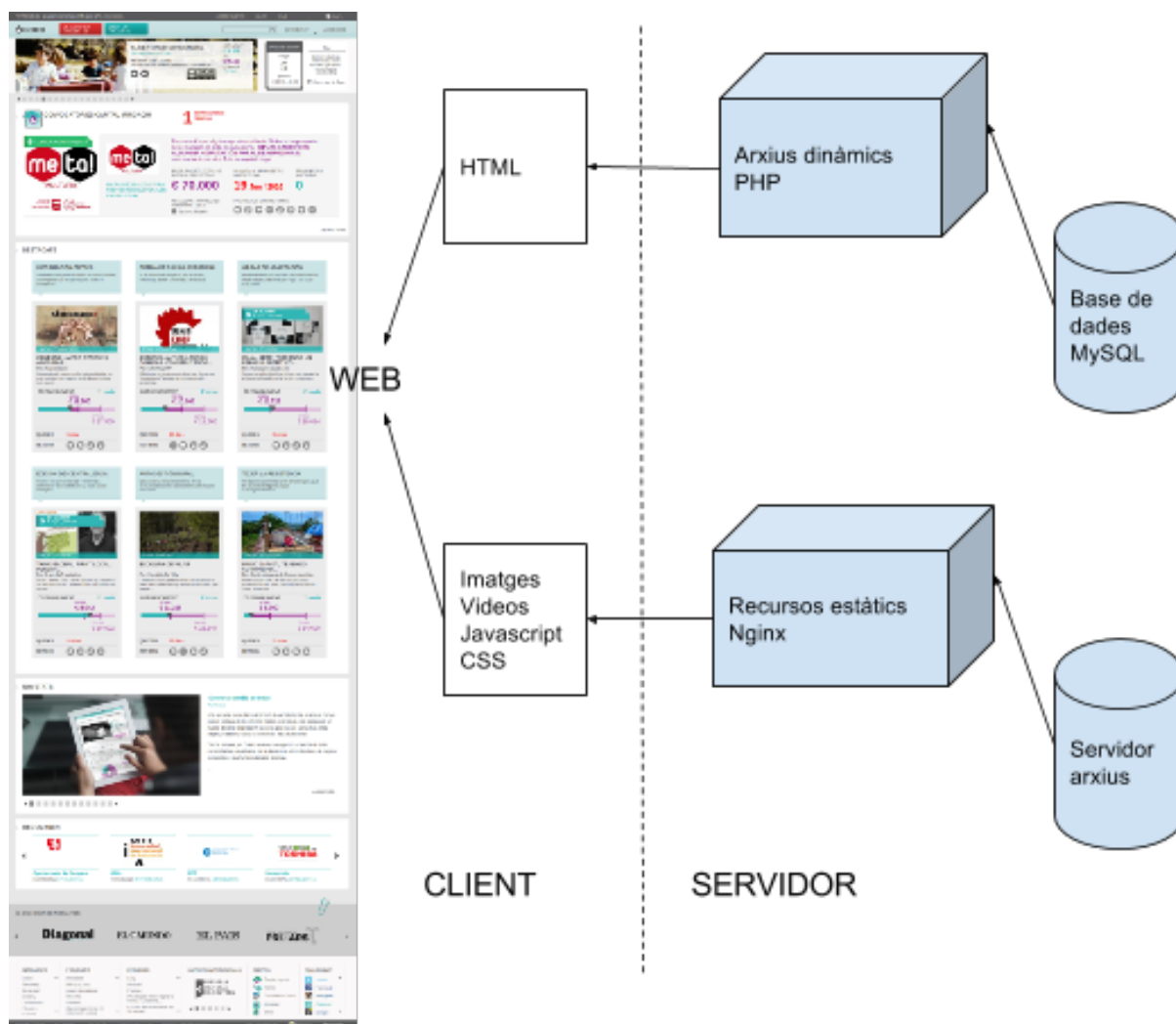


Figura 2: [www.goteo.org](http://www.goteo.org)

La web serveix diversos tipus de recursos (*Fig. 2*), ja siguin estàtics (com imatges o fulles d'estil) o dinàmics (com podrien ser les pròpies pàgines formatades amb contingut de la base de dades).

Tecnologies emprades a la web de Goteo:

- Llenguatges HTML[2], CSS[3] i Javascript per al navegador que utilitza l'usuari final i ha de mostrar el resultat final en forma de web.
- Llenguatge de preprocessat PHP[4] que genera les pàgines HTML finals necessàries al servidor.
- Llenguatge SQL[5] per al servidor de base de dades (MySQL[6]), accessible només per part del motor d'interpretació del llenguatge PHP.

Aquest model de funcionament té el problema de l'alt grau d'acoblament entre els diferents components de la plataforma. Per exemple qualsevol modificació d'una part del funcionament de la web final pot requerir modificacions internes tant a la base de dades com a la capa de preprocessat.

Això té el desavantatge inicial de què qualsevol canvi pot generar errors imprevistos en altres parts que depenen de la mateixa lògica interna, a banda de fer necessari que tots els programadors que hi intervenen tinguin coneixements de totes les parts de la plataforma així com un alt grau de coordinació entre ells.

Un altre desavantatge és que qualsevol desenvolupament independent de la web actual (per exemple, desenvolupar una aplicació mòbil) requereix iniciar un projecte des de zero amb accés directe als recursos bàsics (la base de dades i d'altres recursos estàtics). En moltes situacions això suposaria un problema important de seguretat, seria molt difícil garantir un accés segur i controlat a aplicacions de tercers amb accés directe a la base de dades per exemple.

Finalment, un problema a abordar és el rendiment de l'aplicació. Amb accessos directes als recursos primitius (base de dades, arxius) en els seus servidors d'origen és fàcil sobrecarregar la capacitat computacional d'aquestes màquines.

Una capa d'abstracció intermèdia (API) podria fer-se càrrec de processos de control i limitació d'accés als recursos i inclús proveir d'una capa *cache* (memòria cau) que eviti sobrecàrregues en els servidors per a peticions complexes.

## 1.2 Objectius

L'objectiu del projecte és la creació del programari necessari per a la implementació de l'API d'accés als recursos interns. Alhora, és igualment important la creació de la documentació pública necessària perquè qualsevol consumidor la pugui utilitzar. També és la intenció d'aquest projecte de construir i/o recopilar algunes aplicacions independents que facin ús de l'API i que serveixin com a punt de partida per al seu ús i exemple.

Finalment, es vol que la creació d'aquesta API es mantingui en el temps i sigui, ella mateixa, accessible i modificable per qualsevol que vulgui. Així, es publicarà tot el codi amb una llicència de codi lliure i es donarà a qualsevol la possibilitat de contribuir en millores al seu codi.

## 1.3 Estructura de la memòria

La memòria farà un repàs a tots els processos que han sigut necessaris per a la construcció de l'API. Des de situar el context i motius inicials per desenvolupar-la per part de la *Fundación Goteo*, establir la llista de requeriments i les dades a compartir fins a entrar en detall de com s'ha programat l'aplicació i amb quines tecnologies.

Tenint en compte que l'aplicació és una eina real, pensada per a un ús públic, i que ja es troba en funcionament, també es descriurà com s'ha fet la posada en producció, com s'ha decidit crear-ne la documentació d'ús i, finalment, alguns exemples de com es pot fer servir l'eina generada.

Aquesta memòria es divideix en set capítols (més les conclusions) amb els següents punts:

1. Introducció, on es justifica la necessitat i motius que han portat a desenvolupar l'aplicació Goteo API a part de detallar com està organitzada la memòria.
2. Antecedents. Capítol que ha de servir per donar una idea de què és una plataforma de crowdfunding i que com funciona la de Goteo en particular, que és i quines són les seves diferències en enfront d'altres plataformes similars.
3. Disseny de l'API. En aquest apartat es veurà quin paper juga l'API en l'intercanvi de dades entre els consumidors i les bases de dades originals. També es donarà el llistat de requeriments que ha de resoldre.
4. Implementació de l'API. Detall de l'arquitectura de la programació, capítol més centrat en la tecnologia, s'explicaran els protocols que intervenen en el model de comunicació, s'explicarà com s'ha programat l'aplicació i s'entrarà en detall amb alguns aspectes concrets.

5. Documentació de l'API. Aquí s'explica el treball de documentació pública de com s'utilitza l'API pensat per a programadors externs que en vulguin fer ús. Com està dissenyada i com es genera.
6. Desplegament. L'API és una aplicació real i la seva posada en marxa en un entorn de producció implica certes diferències amb el seu desenvolupament que s'expliquen aquí.
7. Implementació d'alguns exemples de funcionament i/o llibreries d'ús per a alguns llenguatges de programació. Capítol que permet demostrar el funcionament de l'API mitjançant casos d'ús existents.
8. Conclusions.

El codi de l'aplicació, per la seva llargada, no estarà inclòs en aquest document, en el seu lloc, acompanyaran a la memòria dos documents adjunts:

El primer (*codi\_api.zip*) amb el codi de l'aplicació per la seva revisió (el codi també està disponible on-line però és possible que ja sigui una versió més avançada en el moment de la revisió de la memòria).

El segon document (*documentacio.zip*) contindrà la documentació generada segons s'explica en el capítol 5.

## 2. Antecedents. El crowdfunding

Com a base per entendre quin valor aporten les plataformes de micromecenatge o crowdfunding[7], es farà un resum de què signifiquen i com funciona l'intercanvi econòmic que s'hi realitza. Es farà èmfasi en la particularitat del cas de la *Fundación Goteo*[8], que pretén aportar alguns elements diferenciadors (com ara retorns col·lectius, que no solen ser en forma de béns físics).

### 2.1 El *crowdfunding*, orígens i actualitat

S'entén com a *crowdfunding* (o micromecenatge) un tipus de cooperació col·lectiva destinada a tirar endavant projectes de promotors, normalment en forma d'aportacions econòmiques de reduïda quantitat a canvi de la promesa que, un cop realitzat el projecte, s'obtindrà per part del promotor algun tipus de retorn (normalment un bé físic o un servei) a canvi de la seva aportació.

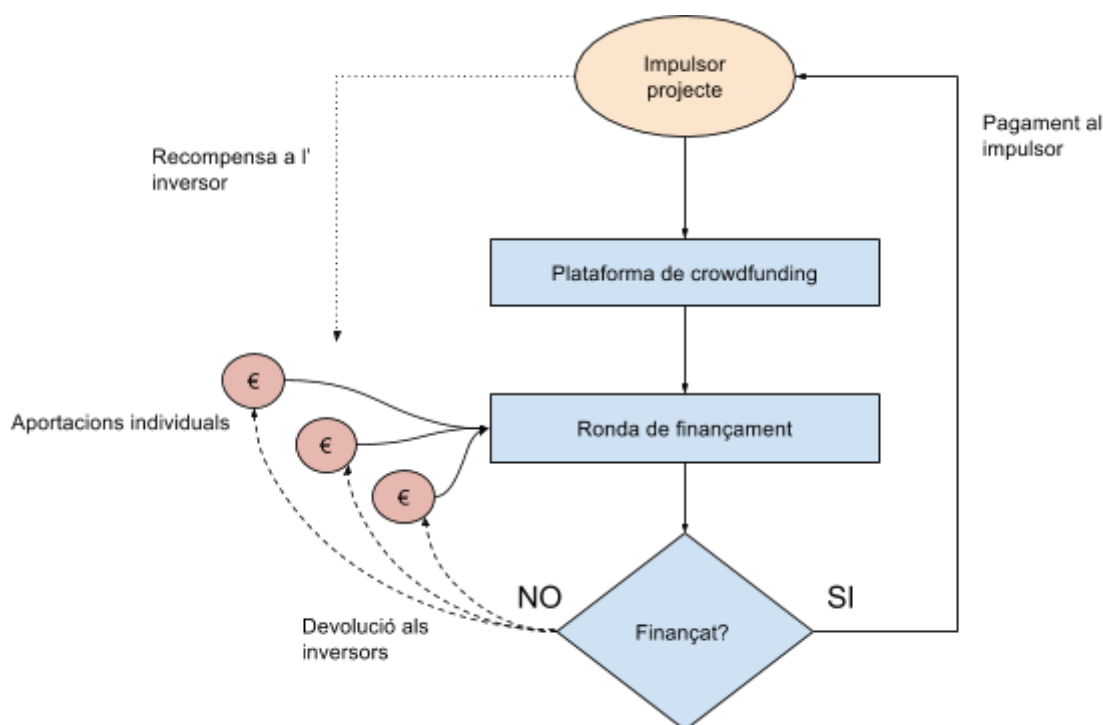


Figura 3: Flux monetari del crowdfunding

Si bé hi ha hagut exemples de crowdfunding anteriors al naixement d'Internet, el gran èxit d'aquesta pràctica ha vingut lligada al fenomen de l'expansió de l'accés a Internet i les xarxes socials. Això ha permès l'aparició de multitud de plataformes que faciliten a l'impulsor la difusió del seu projecte a més de la recaptació de fons durant un temps determinat per part dels petits inversors. Un cop acabada la fase de recaptació de diners, la plataforma sol quedar-se amb una petita comissió i pagar la resta a l'impulsor del projecte perquè el porti a terme. En molts casos, si al cap d'un temps no s'arriba a la quantitat mínima necessària (*Fig. 3*), el projecte es descarta i els diners es tornen íntegrament als inversors.

## 2.2 La plataforma Goteo. Particularitats

Goteo és una plataforma de *crowdfunding* que neix l'any 2011 impulsada per la *Fundación Goteo*, una entitat jurídica pròpia sense ànim de lucre que es regeix per la Llei Espanyola 50/2002, de 26 de desembre[9].

Com les altres plataformes de *crowdfunding*, disposa d'una web on presentar els projectes i realitzar les aportacions individuals. Els impulsors dels projectes també poden definir recompenses que es donaran a l'usuari inversor segons el valor monetari aportat.

### Retorn col·lectiu

La majoria de projectes impulsats per Goteo solen tenir un component addicional conegut com a "retorn col·lectiu", que pot ser de caràcter social, ecològic, etc. Un tipus de recompensa pública que ha de beneficiar a un col·lectiu de persones més gran que només el que hi ha contribuït a la campanya de finançament.

Per altra banda, en ser una fundació sense ànim de lucre, les aportacions que fan els usuaris finals es consideren donacions i tenen beneficis fiscals.

Finalment la fundació té un alt compromís amb la transparència i la cultura copyleft, per això totes les dades són d'accés públic, el codi font de l'aplicació està disponible amb una llicència oberta.

### Capital regadiu

També és una plataforma pensada per a l'impuls i la coparticipació d'institucions i/o particulars que vulguin fer inversions a projectes lligades a les aportacions individuals. Així, aquestes institucions poden tenir una personalitat pròpia dins la plataforma, destacar-ne projectes i fer aportacions econòmiques directes lligades a les aportacions individuals (per exemple, quan un usuari aporta a un projecte, la institució en fa un aportament addicional del mateix import, *veure Fig. 4*).

Aquest tipus de funcionament es coneix com a Capital Regadiu[10] i ja ha permès a diverses campanyes multiplicar els diners recaptats (*Fig. 5*).

Aquests últims aspectes donen una dimensió addicional a la plataforma i serà necessari reflectir-ho en el disseny de l'API.

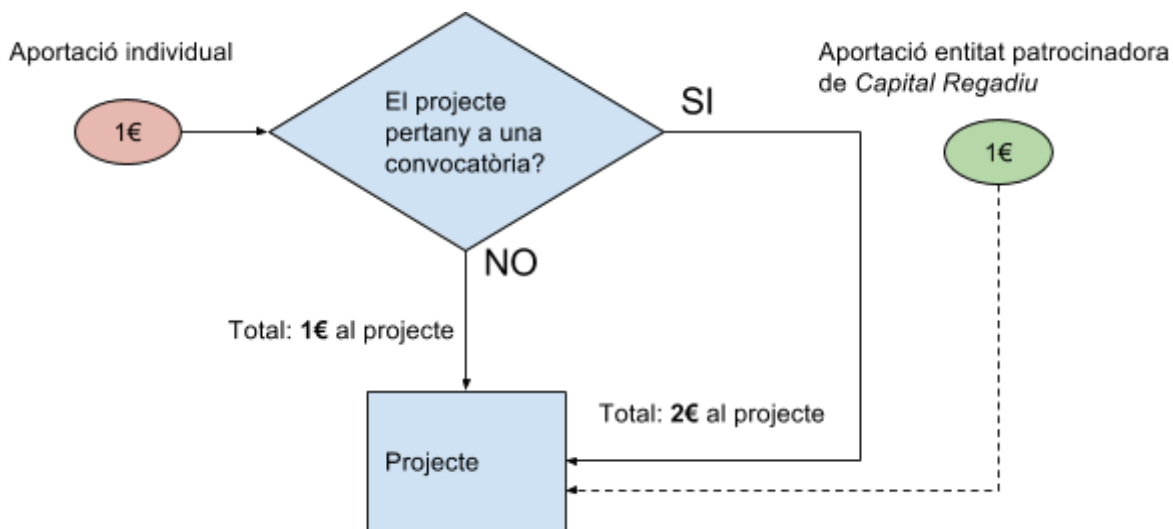


Figura 4: Flux de multiplicació d'aportacions gràcies al capital regadiu



Figura 5: Campanyes de Capital Regadiu actives a Goteo durant els anys 2011 i 2016



## 3. Disseny de la API

En aquest capítol es detallaran els aspectes de disseny de l'API i en quina part del procés entre un client i les dades finals se situa aquesta aplicació. S'especificaran les característiques que haurà de complir l'API, el tipus de protocol en què es comunicarà i com aquest es farà servir per donar servei al client. També quins conceptes de seguretat i autenticació s'aplicaran.

### 3.1 Introducció

Les APIs són sempre aplicacions intermèdies (*Fig. 6*) entre una aplicació ja existent que proporciona certes funcionalitats i dades a un client que les consumeix.

En certa manera permeten replicar el funcionament (o certes parts) d'una aplicació ja existent en clients remots o distribuïts utilitzant les mateixes dades que aquesta. Es tracta doncs d'un model d'intercanvi de dades entre un servidor (o servidors) i clients a través de xarxes. En el cas que ens ocupa les xarxes són aquelles que utilitzen el protocol HTTP[11].

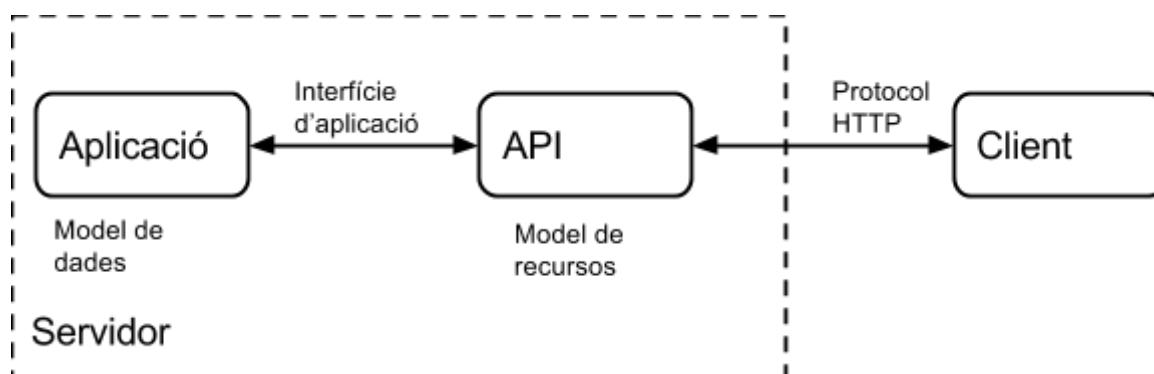


Figura 6: API com a capa intermèdia en l'accés a dades

L'aplicació, origen i destí de les peticions, implementa un model de dades propi per decisió de disseny. En el cas d'aplicacions web sol ser un conjunt de taules relacionades entre ells per diferents camps i utilitzant alguna tecnologia específica (MySQL per exemple).

L'API, per tant, haurà d'utilitzar un tipus d'interfície definida per l'aplicació o les tecnologies que utilitza per a comunicar-s'hi. Alhora, proporcionarà un model definit de recursos que hauran de ser utilitzats pel client per interactuar amb l'aplicació.

Finalment, el client, com a consumidor, haurà d'implementar el model de recursos definit per l'API per tal d'obtenir les dades necessàries que li permetran proporcionar les funcionalitats requerides.

## 3.2 Especificació

L'API és una interfície de dades, basada en el protocol HTTP, caldrà definir doncs quines són les dades que es volen proporcionar i amb quines limitacions. Aquesta especificació ve donada principalment per les necessitats de l'organització Goteo, l'objectiu final de la qual és proporcionar un accés obert a totes les dades de la plataforma web.

Inicialment l'API ha de proporcionar recursos que proporcionin accés a dades de només lectura (no hi ha necessitat inicial d'introduir noves dades usant l'API). No obstant això, la programació haurà de preveure la possibilitat futura d'ampliar-se a un sistema amb accés complet (de lectura i escriptura).

Un requisit imprescindible és el protocol HTTP i, per tant, cadascun dels recursos haurà d'estar disponible un URL[12] únic sobre la que s'executaran les operacions de lectura (o escriptura en el futur).

Com a exemple d'URL que serveixi les dades dels usuaris del sistema pot ser:

```
http://api.goteo.org/v1/users/?from_date=2014-01-01
```

L'URL es divideix en tres parts diferenciades: protocol (en l'exemple: `http://`), host o servidor (`api.goteo.org`), un prefix arbitrari opcional (`/v1`), el recurs d'accés a les dades (`/users/`) i algun paràmetre opcional per filtrar les dades (`?from_date=2014-01-01`).

Tant el protocol com el host i el prefix poden ser configurables en la posada en producció de l'API. El conjunt de recursos romandrà inalterable un cop definit.

Els grups de dades als quals s'ha de proporcionar accés de lectura són:

- Usuaris de la plataforma.
- Dades de projectes, incloent-hi dades de geolocalització
- Convocatòries de "capital regadiu", també amb dades de geolocalització
- Llicències i categories utilitzades pels projectes en la plataforma.
- Les aportacions monetàries, on interessa oferir dades de geolocalització però, per qüestions de privacitat, sense oferir una relació directe amb els usuaris (és a dir, no indicar en cadascuna de les aportacions quin usuari l'ha fet).
- Addicionalment, també altres conjunts de dades estadístiques en mòduls opcionals (que no formen part estrictament d'aquesta memòria però que sí que interessin a la fundació).

D'altra banda, també serà un requisit implementar un sistema de limitació i control d'accés als recursos, establir un format estructurat de presentació de dades i proporcionar un mecanisme per filtrar les dades que estiguin disponibles en forma de taula o llistat (els filtres poden variar depenen del tipus de dades a consultar):

1. Caldrà **limitar el nombre de peticions** que podran fer els clients a l'API en un període de temps determinat (tots dos paràmetres hauran de ser configurables).

Tant el nombre de peticions disponibles com el període de temps que resta haurà de ser consultable pel client que haurà d'utilitzar els mecanismes de què disposa el protocol HTTP per fer-ho.

2. La seguretat també es basarà en els mecanismes estàndards de l'HTTP. Inicialment es pot implementar el mecanisme més senzill d'**autenticació**, basat en usuari i contrasenya, però ha de ser possible configurar els diferents recursos amb diferents nivells de permisos o protocols d'autenticació en el futur si fos necessari (el que requereix separar clarament la lògica que implementa el model de seguretat de la que implementa l'obtenció de dades).
3. Hi ha diverses possibilitats per implementar com s'hauran de proporcionar per part del client els paràmetres de **filtratge**. Totes contemplades dins el protocol HTTP. En un principi només caldrà que l'API llegeixi paràmetres que li arribin com a part de l'URL corresponent a l'anomenat "query string", pensat precisament per enviar variables.

No obstant això, en versions futures, per a l'API de lectura i escriptura, si haurà de ser possible enviar variables en formats més complexos per tal d'actualitzar dades.

4. El **format de dades** de sortida no està especificat en la definició del protocol HTTP. Els requisits que haurà de complir el format de dades són:
  - Estructurat, que sigui de fàcil processat per programari
  - Idealment, que també sigui de fàcil comprensió per a humans
  - Que suporti una àmplia varietat de dades.
  - Alguns formats populars de dades que compleixen aquests requisits són XML[13], YAML[14] o JSON[15]. El format més pràctic per a aplicacions basades en la web és el JSON i, per tant, serà el preferit per defecte (sense que vulgui dir que no es puguin implementar altres formats de sortida en el futur).

## 3.3 El model de recursos

Anomenarem model de recursos al conjunt de regles i especificacions que estableixen com es connectaran els clients que faran ús de l'aplicació a través de l'API.

En la implementació de l'API, la funcionalitat de l'aplicació ja ve definida donat que ja existeix, així com les tecnologies emprades a més baix nivell (com la tecnologia de la base de dades).

Aquest model de recursos haurà de proporcionar un entorn en el qual els clients podran obtenir informació i manipular-la en la manera que creguin convenient. El model de recursos els proporcionarà com i amb quins límits podran obtenir-la. L'objectiu és que el model de recursos pugui evolucionar en el temps mantenint la compatibilitat amb versions anteriors.

Ens proposem aconseguir:

- Alta disponibilitat (escalabilitat)
- Compatibilitat amb el màxim de tecnologies i aplicacions existents actuals
- Facilitat d'ús: Usable per humans i per màquines
- Aplicació d'estàndards actuals
- Basat en el protocol HTTP (Internet)
- Documentació estructurada en anglès (idioma *de facto*)

### 3.3.1 Especificació de recursos

Els recursos a implementar es definiran per tal de satisfer les necessitats inicials de la *Fundación Goteo*. Les necessitats inicials només comprenen la presentació de dades en mode de lectura (sense accés d'escriptura). No obstant això, l'especificació de l'API permetrà l'ampliació gradual a una API d'escriptura (així com l'afegit de recursos addicionals) en el moment en què sigui necessari.

Els recursos inicials a implementar són:

Recurs	Tipus	Descripció
Usuaris <code>/users/</code>	<i>Col·lecció</i>	Llistat d'usuaris del sistema
Usuari <code>/users/ID</code>	<i>Objecte</i>	Accés a la informació d'un usuari en concret
Categories <code>/categories/</code>	<i>Col·lecció</i>	Llistat de les categories relacionades amb els projectes i usuaris
Llicències <code>/licenses/</code>	<i>Col·lecció</i>	Tipus de llicències pels retorns col·lectius
Projectes <code>/projects/</code>	<i>Col·lecció</i>	Llistat de projectes publicats
Projecte <code>/projects/ID</code>	<i>Objecte</i>	Accés a la informació completa d'un projecte en concret
Donant d'un projecte <code>/projects/ID/donors/</code>	<i>Col·lecció</i>	Llistat d'usuaris que fan aportacions econòmiques a un projecte
Aportacions monetàries <code>/invests/</code>	<i>Col·lecció</i>	Llistat de donacions als projectes
Aportació monetària <code>/invests/ID</code>	<i>Objecte</i>	Accés a la informació concreta d'una aportació monetària
Convocatòries <code>/calls/</code>	<i>Col·lecció</i>	Llistat de convocatòries de "capital regadiu" actives
Convocatòria <code>/calls/ID</code>	<i>Objecte</i>	Accés a la informació concreta d'una convocatòria
Projectes d'una convocatòria <code>/calls/ID/projects/</code>	<i>Col·lecció</i>	Llistat de projectes que han estat escollits per una convocatòria

S'han dividit els tipus de recursos en dos tipus: objectes i col·leccions.

Els objectes representen informació d'algun element de la plataforma existent independentment del moment temporal o d'altres opcions de filtratge. Per exemple, un usuari o un projecte de la plataforma.

Per contra, les col·leccions són llistats d'objectes que es generen dinàmicament en funció dels paràmetres d'entrada (o filtratge) i es mostren en blocs amb un límit màxim d'objectes per bloc. Cada bloc es mostra en una petició a l'API diferent. A part del llistat, col·lecció haurà de donar informació del nombre total d'elements que hi ha i el bloc actual que s'està mostrant. La majoria d'objectes tindran una col·lecció associada: llistats d'usuaris, llistats de projectes, etc.

A més, les col·leccions es podran filtrar en funció de diversos paràmetres, cada col·lecció pot tenir alguns paràmetres particulars (depenent del tipus d'element a filtrar) però, en general, els filtres necessaris són:

Paràmetre	Tipus	Descripció
Pàgina <code>page</code>	<i>Nombre enter</i>	S'utilitza en el cas que el nombre d'elements disponibles superi el límit d'elements per bloc per definir el bloc a retornar.
Límit de pàgines <code>limit</code>	<i>Nombre enter</i>	Permet definir el nombre d'elements per bloc (sempre menor que un màxim intern preestablert). En augmentar el límit es disminuiran el nombre de pàgines i viceversa.
Idioma <code>lang</code>	<i>Cadena caràcters (múltiple)</i>	Permet retornar la informació en l'idioma especificat. És un paràmetre múltiple, es pot especificar diverses vegades per establir una prioritat en l'ordre d'idiomes a substituir en cas que l'idioma desitjat no estigui disponible per a un objecte en concret.
Projecte <code>project</code>	<i>Cadena caràcters (múltiple)</i>	Si aquest paràmetre està present, es limitarà la cerca al projecte (o projectes) especificats. També és un paràmetre múltiple, es podrà especificar més d'un projecte.
Categoria <code>category</code>	<i>Nombre enter (múltiple)</i>	Permetrà restringir la cerca a les categories indicades (valor múltiple). El nombre enter correspondrà a l'identificador únic que té cada categoria (disponible al recurs de categories).
Canal <code>node</code>	<i>Cadena caràcters (múltiple)</i>	Alguns dels elements de la plataforma estan agrupats en canals o nodes. Hi ha per exemple el node de Barcelona, el d'Euskadi i Andalusia. Aquest paràmetre permetrà filtrar segons aquesta agrupació.
Data inicial <code>from_date</code>	<i>Data</i>	Si es volen restringir els resultats a una determinada franja temporal, aquest paràmetre n'especificarà el moment d'inici.
Data final <code>to_date</code>	<i>Data</i>	Serveix per especificar el límit superior de la franja temporal.
Localització geogràfica <code>location</code>	<i>Llista de nombres decimals i enters</i>	Amb aquest paràmetre es filtraran per àrea geogràfica els resultats. Vindrà definit per un punt geogràfic (en coordenades de latitud i longitud) i un enter addicional que indicarà el radi en kilòmetres a la rodona en la qual es vol restringir.

Adicionalment, també es volen implementar recursos amb informació estadística de les dades de la plataforma (com, per exemple, les quantitats de diners recaptats, estadístiques d'èxit dels projectes, etc.).

Tot i que tècnicament aquests recursos són tots objectes (doncs no representen llistats) sí que tenen la particularitat de poder ser filtrats pels mateixos paràmetres que les col·leccions per tal de generar diferents representacions estadístiques. Podríem així obtenir estadístiques d'ús segons una localització geogràfica, una franja temporal, categories implicades, etc.

Tota aquesta informació anirà definida en els recursos:

Recurs	Tipus	Descripció
Comunitat <code>/reports/community/</code>	<i>Objecte</i>	Informe amb dades acumulatives i estadístiques sobre l'ús de la plataforma en relació als usuaris que la formen.
Aportacions monetàries <code>/reports/money/</code>	<i>Objecte</i>	Informe sobre les quantitats i mitjanes de diners donats pels usuaris als projectes.
Projectes <code>/reports/projects/</code>	<i>Objecte</i>	Informe dels projectes de la plataforma, quantitat total, dels més exitosos, dels fallits, categories més emprades, etc.
Recompenses <code>/reports/rewards/</code>	<i>Objecte</i>	Informe amb els tipus de recompenses emprades pels projectes i les llicències utilitzades per cadascun d'ells.
Sumari resum <code>/reports/summary/</code>	<i>Objecte</i>	Informe a tall de resum que aglutina alguns dels valors més destacats dels recursos anteriors.
Agrupacions de recursos <code>/digests/</code>	<i>Col·lecció</i>	Aquest és un recurs especial que permetrà agrupar qualsevol dels recursos anteriors en intervals predefinitos de temps. Els intervals podran ser divisions anuals de tota la història de la plataforma o els 12 mesos d'un any concret. La col·lecció resultant estarà formada pels mateixos objectes que s'obtidrien en successives peticions filtrades per franges de data equivalent.



## 4. Implementació de la API

En aquest capítol es detallaran tots els aspectes tècnics implicats en el desenvolupament de l'aplicació API. Inicialment es farà una descripció extensiva de les tecnologies i protocols que intervindran en totes les fases de la comunicació client-servidor.

Per altra banda, també es descriurà com configurar l'entorn, les eines necessàries i els requeriments de software per tal de poder fer un ús efectiu de l'aplicació. En la part d'arquitectura del codi font es farà una introducció a com s'han programat i organitzat els components de software que la componen.

Finalment, alguns dels casos concrets de la programació que, per la seva especificitat, s'ampliaran i es detallaran de manera més específica.

El codi complet estarà disponible en un annex a aquesta memòria.

### 4.1 Models operacionals i protocols

Com a model operacional entenem els processos que intervenen en la comunicació entre el client i el servidor. Es descriuran amb detall com s'han de comunicar les aplicacions amb la API i com aquesta les respondrà i en quins formats.

El model proposat correspon a l'anomenat REST, que agrupa tot el conjunt de protocols i formats de comunicació que utilitzarà la API, tan en l'entrada com la sortida de dades. El model REST forma part integral de l'aplicació i tota la programació anirà encaminada a implementar-lo.

#### 4.1.1 Arquitectura del model: REST

REST[16], és una de les formes més esteses actualment per organitzar i estructurar l'accés a dades en xarxes distribuïdes. El concepte va aparèixer de la mà de Roy Fielding l'any 2000 com una alternativa més simple als estils imperants a l'època. Actualment s'ha convertit en la manera preferida per la majoria de serveis web a Internet per tal de donar accés als seus continguts en aplicacions de tercers.

REST (*Representational State Transfer*) no és un estàndard en si mateix, per contra, es tracta d'una col·lecció de 6 principis fonamentals de disseny que si se serveixen d'estàndards. L'aplicació més o menys estricta d'aquests principis queda en mans del desenvolupador. L'aplicació completa dels principis se sol conèixer amb el nom de RESTful.

El concepte central REST gira entorn del punt d'accés URI[17] (*Uniform Resource Identifier*), una forma d'identificar recursos de forma única emprada massivament a la WWW[18].

### **Principis REST:**

- **Client-server:** Client i servidor estan físicament i conceptualment separats. L'únic que els enllaça és el canal de comunicació (HTTP). Ni el client té perquè conèixer com manipula les dades el servidor, d'on provenen o quina tecnologia emprà per generar-les. Igualment el servidor no té cap coneixement de com presenta la informació el client.
- **Stateless:** El servidor no ha de mantenir cap "context d'estat" respecte a una petició d'un client. Cada petició ha de portar influïda en si mateix tota la informació necessària per obtenir les dades demanades (Per exemple, l'autenticació). Això allibera al servidor de la responsabilitat de mantenir una sessió coherent i mantinguda en el temps amb el client, cosa que, per exemple, en facilita l'escalabilitat.
- **Cache:** Per tal de millorar l'eficiència de la comunicació, el sistema ha d'implementar els mecanismes estàndards de la WWW per tal d'assegurar que el client pugui optimitzar les peticions mitjançant l'ús de contingut prèviament emmagatzemat en *cache*.
- **Uniform interface:** L'èmfasi en una interfície uniforme d'accés és el punt més important de l'arquitectura REST. Defineix la interfície entre el client i el servidor seguint els principis:
  - Recursos identificables de manera unívoca mitjançant URIs: els recursos en si mateix estan conceptualment separats de la representació que es retorna al client. Aquesta representació serà algun format estandarditzat de dades (per exemple JSON, XML, etc.).
  - Manipulació de recursos a través de la representació: hi ha prou informació retornada al client per permetre a aquest modificar-la o esborrar-la del servidor (sempre que tingui permís).
  - Missatges auto descriptius: cada missatge ha d'incloure tota la informació necessària perquè el client la pugui processar sense recursos addicionals (per exemple, si s'envia una imatge, cal enviar-ne també el tipus MIME per saber de quin tipus es tracta)
  - *HATEOAS*[19] (*Hypermedia as the Engine of Application State*): El client podrà obtenir de la resposta a una petició, informació addicional (coneguda com a *hypermedia*) que el permetrà executar transicions o accions addicionals (per exemple, en requerir el nom d'un usuari se li pot retornar un URL on resideix el perfil de l'usuari o altres meta-dades).
- **Layered system:** El client sempre es connecta al servidor sense tenir coneixement de si és un servidor final o simplement un servidor intermediari que replica contingut. Això permet dissenyar un sistema "per capes" on aplicar principis d'escalabilitat, seguretat o *cache* on poden actuar múltiples servidors de manera transparent.

- **Code-on-demand:** Aquest és l'únic component de l'arquitectura REST que és opcional. Significa que el servidor pot (temporalment) estendre les funcionalitats del client mitjançant la transferència de lògica executable. Un exemple seria l'enviament de fragments de codi JavaScript[20] a un navegador web.

### **Estàndards utilitzats en l'arquitectura REST:**

- **HTTP (*Hypertext Transfer Protocol*):** Com a canal de comunicació, proveeix un protocol de transferència d'informació al nivell d'aplicació. És l'estàndard usat en la WWW.
- **URL (*Uniform Resource Locator*):** Es tracta d'una seqüència de caràcters que identifica de manera unívoca cadascun dels recursos presents a internet. L'estructura bàsica que segueix és:  
esquema://maquina:port/arxiu
- **Presentació de recursos:** XML/HTML/JSON/GIF/JPEG/... En el nostre cas farem servir exclusivament el format JSON per la presentació de recursos. Altres arxius multimèdia addicionals podran ser entregats en forma d'enllaços a llocs externs.
- **Tipus MIME:** text/xml, text/html, text/json... Text/JSON en el nostre cas.

## 4.1.2 Model operacional

El model operacional fa referència al protocol de comunicació entre el servidor i el client. Aquest protocol és l'HTTP o *Hypertext Transfer Protocol* definit per l'estandard RFC 2616[11] publicat a la web de la *Internet Engineering Task Force*.

HTTP és un protocol al nivell d'aplicació, setè nivell de la pila del model OSI[21] (Fig. 7), i se serveix dels protocols TCP/IP[22] del nivell de transport i xarxa. En si mateix és un protocol sense estat, no desa cap informació sobre connexions anteriors.

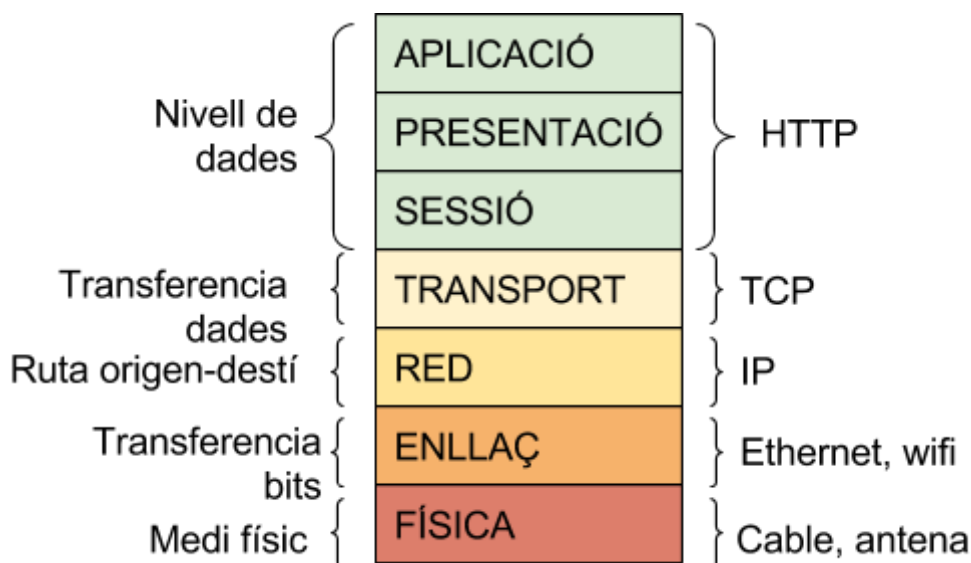


Figura 7: HTTP en la pila del model OSI

Les capes de presentació i sessió del model OSI no es fan servir en el model de comunicació d'Internet via HTTP.

El protocol està orientat a transaccions client - servidor i sempre segueix l'esquema petició - resposta.

Per tal de seguir els principis *RESTful*, és important conèixer algunes de les particularitats del protocol HTTP que seran part fonamental del funcionament de l'API.

### Transacció HTTP

Una transacció HTTP està formada sempre per un encapçalament i, si és necessari, una línia en blanc i dades addicionals.

En el cas del client la primera línia de l'encapçalament indica l'acció a realitzar sobre un recurs concret. Aquesta acció es defineix amb una sola paraula en majúscules i se sol conèixer com a "verb". A continuació segueix el recurs requerit i coincideix amb la part del URL que ve després del nom de servidor.



Per a la construcció de l'API el més important és: per una banda, conèixer els verbs que es poden definir en una transacció HTTP i el seu significat; per l'altra, els codis d'estat retornats pel servidor.

Cal definir quins verbs (o mètodes) s'implementaran i el tipus de resposta retornada per a cada un d'ells. La definició de cadascuna d'aquestes accions haurà d'estar inclosa en la documentació de l'API per facilitar-ne l'ús per part de tercers. La paraula que defineix cada mètode (o verb) està definit en cada versió del protocol. En la versió HTTP/1.0, per exemple, només els verbs GET, POST i HEAD estaven definits, la versió HTTP/1.1 va incrementar en 5 el nombre de verbs que es podien utilitzar en afegir les paraules OPTIONS, PUT, DELETE, TRACE i CONNECT.

En una primera versió de l'API, com que només és de lectura, aquests seran els mètodes implementats:

### Mètodes HTTP

GET	Serà el mètode més utilitzat en l'API. Les peticions amb aquest mètode només poden esperar obtenir una representació concreta de les dades referents a cert recurs.
OPTIONS	Aquest mètode serveix per informar el client de quins mètodes estan disponibles al servidor. En el nostre cas GET, OPTIONS i HEAD.
HEAD	Aquest mètode funciona exactament igual al mètode GET però amb la diferència que no retorna les dades, només les capçaleres. Això permet al client preveure el tipus i quantitat d'informació que haurà de gestionar en una consulta GET real.

Qualsevol petició feta amb altres verbs retornarà un codi d'error estàndard de "Mètode no existent". Altres verbs (com POST o PUT) es definiran en el futur per tal d'ampliar les capacitats de l'API.

### Capçaleres

A partir de la segona línia que indica el mètode i, abans de la línia en blanc, poden diverses línies anomenades capçaleres (o *headers*). Aquestes solen indicar metadades útils, sigui de les dades que seguiran o bé de la mateixa transacció. Per exemple, com que HTTP és un protocol sense estat, les implementacions de seguretat i autenticació es faran a partir de capçaleres.

Algunes de les capçaleres més utilitzades són:

- **Date:** Dia i hora de la transacció. En format especificat al *RFC 7231*[23].
- **Content-Type:** Indica el tipus de dades que es retornaran en format MIME. Per exemple un arxiu HTML tornaria un tipus "text/html". Una imatge JPEG un tipus "image/jpeg".

- **Content-Length:** Indica la llargada en bytes de les dades retornades.
- **Accept:** El tipus de dades que seran acceptables com a resposta. Aquesta capçalera només la sol enviar el client.
- **Authorization:** Es fa servir per indicar el tipus d'autenticació i les credencials necessàries.
- **User-Agent:** El nom i versió del software utilitzat en la petició o resposta.
- **Location:** L'adreça d'un recurs. Es fa servir en redireccions per part del servidor.

Exemple de petició amb capçaleres:

```
OPTIONS /v1 HTTP/1.1
User-Agent: curl/7.35.0
Host: api.goteo.org
Accept: */*
```

Exemple de resposta a l'anterior petició amb capçaleres:

```
HTTP/1.1 200 OK
Date: Wed, 30 Sep 2015 11:56:02 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Allow: HEAD, OPTIONS, GET
```

Com es veu, la resposta retornada per al mètode OPTIONS inclou en la capçalera "Allow:" tots els mètodes vàlids que es poden utilitzar al servidor. La resposta no inclou dades perquè el mètode no ho requereix.

### **Codis d'estat retornats pel servidor**

Així com una petició sempre ha d'incloure el mètode que s'emprarà, la resposta sempre inclourà un codi d'estat. El codi d'estat ve en la primera línia després de la versió del protocol i es compon d'un número de tres xifres seguit d'una descripció curta.

El tipus de numeració indicarà el tipus de resposta, hi ha 5 grups:

- **1xx, Informativa:** Petició rebuda, en procés.
- **2xx, Exitosa:** La petició s'ha rebut, entès i processat correctament.
- **3xx, Redirecció:** La petició requereix més accions per part del client per tal de completar la resposta.
- **4xx, Error en el client:** La petició és incorrecta o no pot ser completada per algun altre motiu (no hi ha permisos, no existeix el recurs, etc).

- **5xx, Error en el servidor:** La petició sembla correcta per part del client però ha fallat el servidor en intentar respondre.

L'API, doncs, farà servir aquests codis d'estat per transferir informació del resultat consulta.

Alguns codis d'estat comuns son:

- HTTP/1.1 200 OK: Resposta exitosa, i si el mètode ho requereix, les dades necessàries seguiran a continuació.
- HTTP/1.1 307 Temporary Redirect: El recurs demanat pel client es troba en un altre lloc de forma temporal. El lloc del nou recurs anirà especificat per una capçalera tipus "*Location:*". Properes consultes hauran de tornar a provar aquesta mateixa localització doncs la redirecció és temporal.
- HTTP/1.1 301 Moved Permanently: Idèntic a l'anterior amb la diferència que la redirecció és permanent. El client no cal que torni a prova mai més aquesta adreça doncs no es farà servir més.
- HTTP/1.1 400 Bad Request: Una petició que està mal formada retornarà aquest codi d'estat.
- HTTP/1.1 401 Unauthorized: En el cas que el servidor implementi algun tipus d'autenticació, si aquesta no és vàlida, es retornarà aquest codi d'error.
- HTTP/1.1 403 Forbidden: Aquesta resposta es torna quan el client no té permís per accedir al recurs sol·licitat encara que estigui autenticat correctament.
- HTTP/1.1 404 Not found: Es retorna quan un recurs no existeix al servidor.
- HTTP/1.1 405 Method not allowed: Es retorna quan es fa servir un mètode que no està implementat al servidor.
- HTTP/1.1 429 Too Many Requests: Aquest és un codi addicional, definit l'estàndard *RFC 6585*[24]. Serveix per indicar que el client ha fet massa peticions en un determinat espai de temps.

### 4.1.3 Model d'autenticació

A l'hora de dissenyar l'API, proveir d'un sistema d'autenticació per part del client és desitjable en molts casos. Per una banda es pot mantenir una lògica de permisos que restringeixen l'accés a certes parts depenent de les credencials del client. Un altre motiu pot ser mantenir un control o establir certs límits a qui accedeix als recursos.

Tot i que HTTP és un protocol sense estat (i per tant no conserva informació sobre el client i els seus permisos), proveeix de diferents esquemes d'autenticació bastats en

informació present en les capçaleres. L'API podrà implementar un o més d'aquests esquemes.

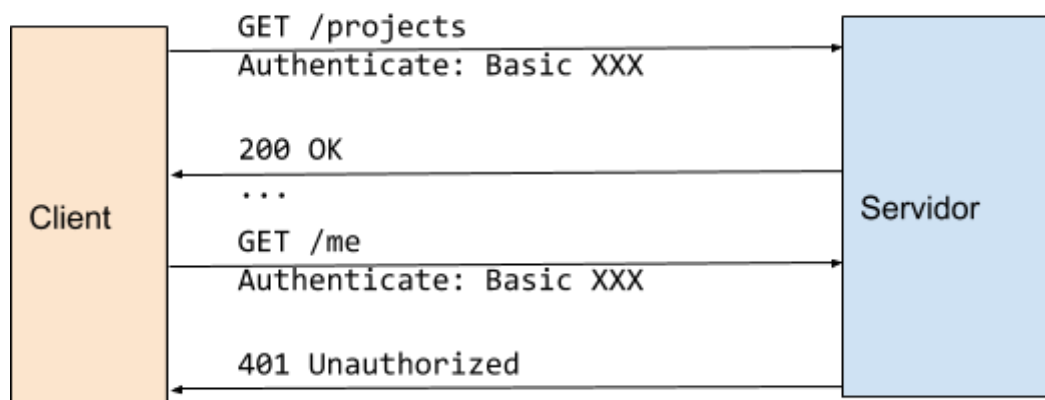
### **Mecanismes d'autenticació i àmbits d'aplicació**

Cal distingir entre dues potencials maneres d'accedir a l'API.

D'una banda, aquella que permet a clients accedir a recursos generals, informació que no necessita el consentiment d'un usuari. És una autenticació al nivell d'aplicació.

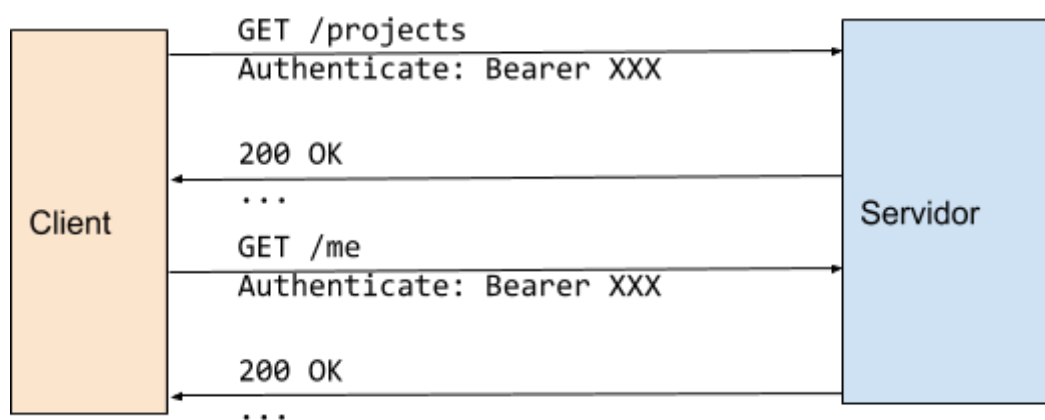
De l'altra, tenim aquella informació que si necessita el consentiment de l'usuari (per exemple per realitzar una acció en nom d'aquest). Aquest últim mètode també dóna accés als recursos del primer.

1. **Autenticació al nivell d'aplicació:** no es poden realitzar accions en nom d'un usuari de la plataforma (*Fig. 10*).



*Figura 10: Autenticació a nivell d'aplicació*

2. **Autenticació al nivell d'usuari:** es poden realitzar accions en nom de l'usuari final (*Fig. 11*).



*Figura 11: Autenticació a nivell d'usuari*

Depenent doncs, del tipus de recurs al que es vol accedir el client haurà d'utilitzar un tipus d'autenticació o un altre. Cadascun d'aquests tipus seguirà el seu propi mètode.

### Tipus d'autenticació "Basic"

Aquest mètode, el més senzill, anomenat *Basic* està definit a l'estàndard *RFC 2617*[25]. Es basa en l'enviament de dues capçaleres específiques (*Fig. 12*), una per la banda del servidor i l'altre pel client verificant que, tant el servidor com el client, coneixen una clau secreta (normalment usuari i contrasenya).

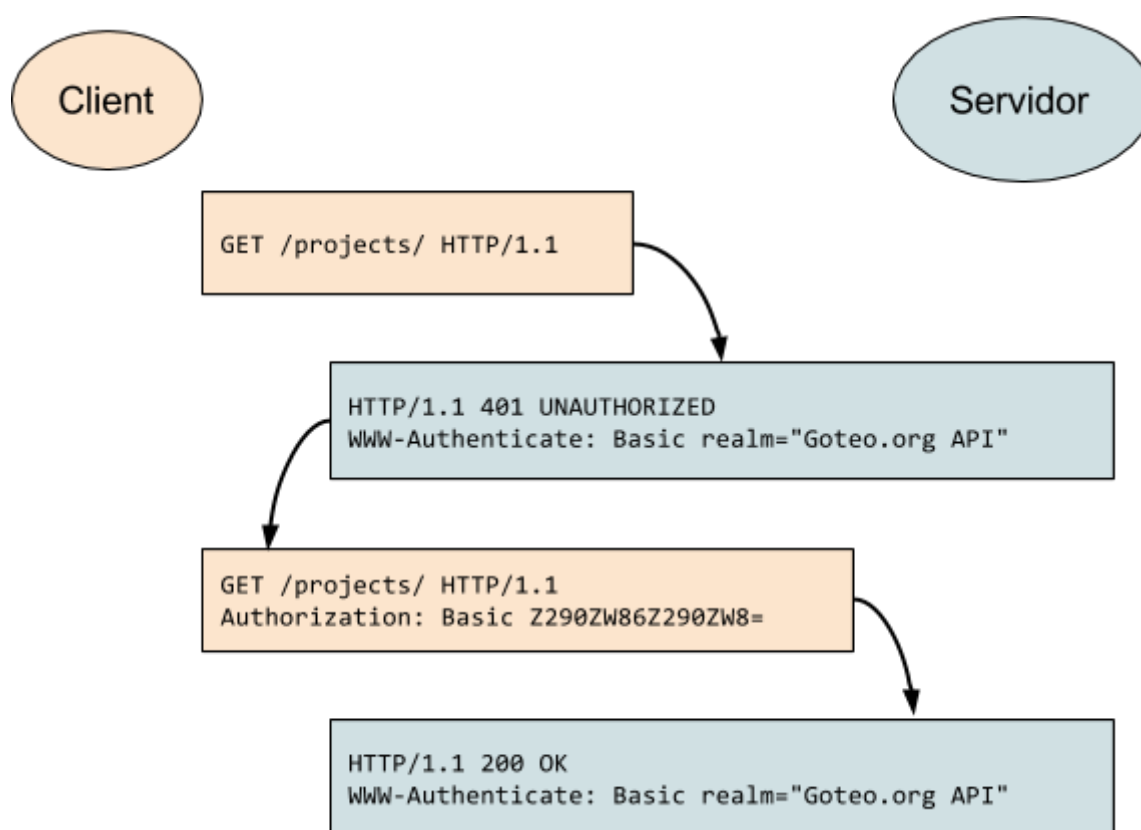


Figura 12: Flux i capçaleres intercanviades entre client i servidor en un procés d'autenticació Basic

El procés que segueix un sistema d'autenticació HTTP és:

1. El client pot provar d'accedir al recurs sol·licitat mitjançant una petició estàndard i sense enviar cap capçalera específica.
2. El servidor respon amb un codi d'estat 401, no autoritzat. També inclou una capçalera específica de tipus "WWW-authenticate:". Aquesta capçalera indicarà al client quin tipus d'autenticació s'espera que proveeixi per tal d'accedir al recurs sol·licitat. En el nostre cas tipus Basic.

3. El client pot aleshores tornar a intentar la connexió enviant una capçalera pròpia, de tipus "Authorization:" amb el contingut de l'autenticació.
4. El servidor respon el codi d'estat 200 i el contingut requerit. També podria respondre amb un codi 403 si el recurs sol·licitat no està disponible per a l'usuari en qüestió tot i estar correctament autenticat.

### Capçalera WWW-authenticate

Aquesta capçalera és proporcionada pel servidor quan es sol·licita un recurs pel qual es necessita autenticació. Indicarà el tipus d'autenticació i d'altres paràmetres que dependran del tipus d'aquesta. En el cas de *Bàsic*, respondrà un missatge arbitrari que pot ser mostrat al client com a informació de l'autenticació:

```
WWW-Authenticate: Basic realm="Goteo.org API"
```

### Capçalera Authorization

Per la seva banda, el client ha de fer la petició incloent-hi la capçalera *Authorization*, que, a l'igual que el cas anterior, contindrà el mètode d'autenticació (Bàsic) i una cadena de caràcters que contindrà un nom d'usuari i contrasenya que el servidor hauria de considerar com a vàlids. En aquest cas el servidor retornaria el codi d'estat 200, o 40x si fos incorrecte.

La cadena de caràcters està formada concatenant el nom d'usuari i la contrasenya i separant-los amb el símbol de dos punts:

```
usuari:contrasenya
```

El resultat es codifica aleshores mitjançant l'algoritme base64 (definit al document *RFC 2045*):

```
Authorization: Basic Z290ZW86Z290ZW8=
```

### Àmbits d'actuació

Aquest mètode d'autenticació s'implementarà en els recursos accessibles al nivell d'aplicació i, addicionalment, en el recurs de *login* que permet obtenir una autenticació del nivell d'usuari final.

### Tipus d'autenticació "OAuth 2.0"

Hi ha situacions en què és desitjable permetre a clients de tercers actuar en nom d'un usuari de la plataforma sense que conegui les credencials d'accés (usuari i contrasenya). A més, és possible també restringir a quines zones de l'aplicació es dona accés al client, conegut com a *scope* (àmbit).

Hi ha diversos protocols pensats per proveir d'aquesta funcionalitat en arquitectures tipus REST, OpenID, OAuth 1.0 i OAuth 2.0 són els més populars. En el nostre cas, en serà suficient amb una variació del protocol OAuth 2.0 coneguda com a *Implicit OAuth Authentication* o autenticació implícita. La documentació completa d'aquest protocol està definit a l'estàndard RFC 6749[26] (capítol 4.2 per l'autenticació implícita).

### Autenticació implícita

L'autenticació implícita és una simplificació del protocol OAuth 2.0 (en el qual intervé una petició addicional al servidor d'autenticació). El protocol complet se sol utilitzar en aplicacions finals, en el nostre cas ens interessa per afegir una capa de seguretat addicional a l'accés a l'API (doncs d'aquesta manera no han de viatjar les credencials de l'usuari a cada petició a un recurs).

El protocol simplificat (o implícit) es basa a fer dues peticions per tal d'aconseguir accés als recursos (*Fig. 13*). La primera petició es fa al servidor d'autenticació (no cal que sigui el mateix que el servidor de recursos) identificat amb una URI pròpia. Aquesta petició utilitza les mateixes capçaleres que per una autenticació tipus Basic. És a dir, s'envia usuari i contrasenya amb l'objectiu d'obtenir un *token* (un identificador temporal) que es farà servir d'ara endavant per autenticar les peticions a recursos.

Cada subseqüent petició a recursos haurà de fer ús d'aquest *token* proporcionat pel servidor d'autenticació. Aquests *tokens* tenen l'avantatge que, en cas de ser compromesos, poden ser anul·lats per forçar al client a demanar-ne un de nou. Inclús és possible fer que tinguin una validesa temporal després del qual el *token* queda invalidat automàticament.

Com en el protocol *Basic*, s'utilitza la mateixa capçalera *Authorization* per fer les peticions als recursos i al servidor d'autenticació. La diferència, però, és que es fa servir la paraula *Bearer* per indicar al servidor de recursos que l'autenticació que es farà servir està basa en un *token* i no en usuari i contrasenya.

Capçalera enviada al servidor d'autenticació:

```
Authorization: Basic Z290ZW86Z290ZW8=
```

Capçalera enviada al servidor de recursos:

```
Authorization: Bearer eyJleHAiOjE0NjM5MTQxO...
```

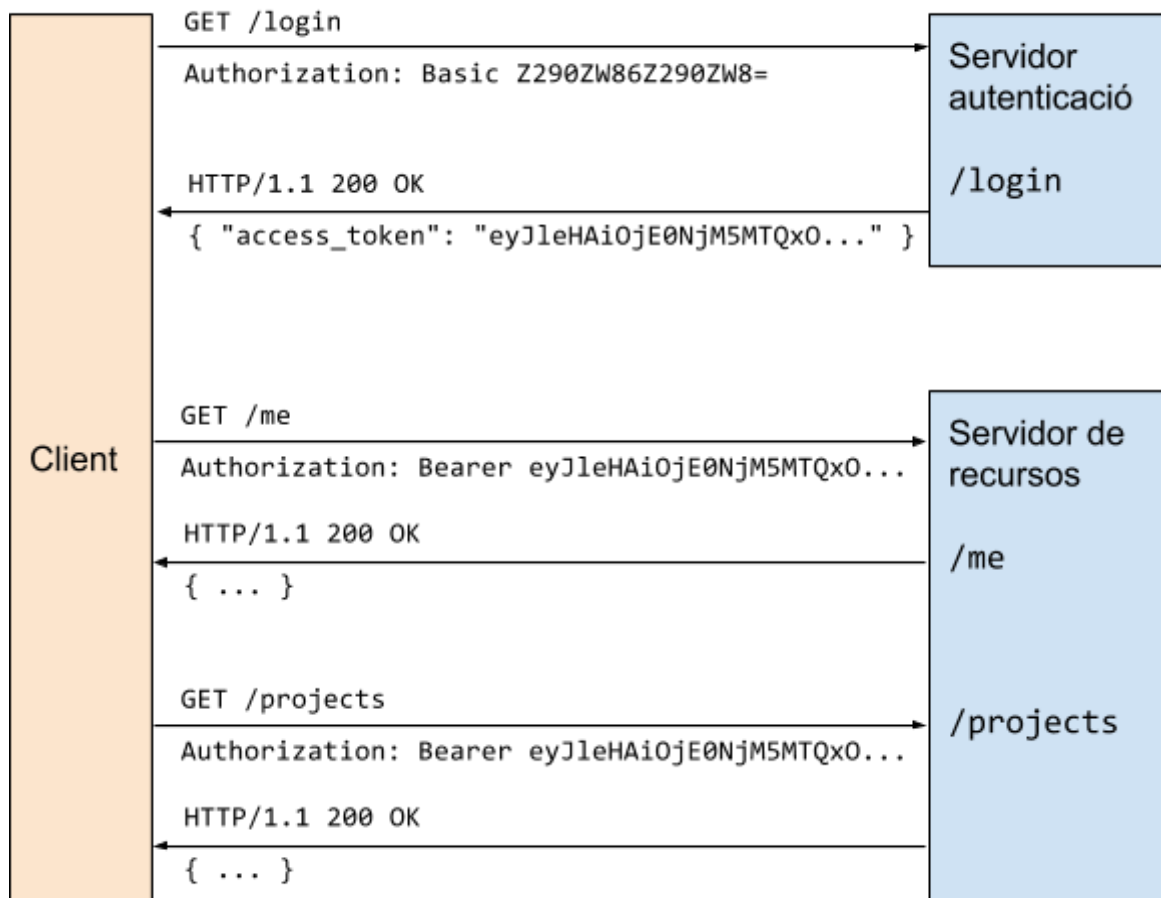


Figura 13: Flux de peticions en un procés d'autenticació OAuth 2.0 implícit

### Àmbits d'actuació

Aquest mètode d'autenticació s'implementarà en els recursos accessibles al nivell d'usuari final. Es a dir, els mateixos que els del nivell d'aplicació més tots aquells en els quals el client ha d'actuar en nom de l'usuari (per exemple realitzar un pagament).

## Altres mecanismes de control. Limit de peticions

Un altre dels mecanismes de control que convé implementar, és l'establiment de límits en l'accés a recursos. Aquest és un aspecte de l'API bàsic per tal d'evitar abusos d'ús per part del client o bé com a resposta en el model de permisos (alguns usuaris poden tenir permís per fer més peticions que d'altres).

El que es pretén doncs, és limitar el nombre de peticions per a un client determinat en un cert espai de temps. Per exemple, 300 peticions en un període de 15 minuts. Un cop exhaurit el nombre de peticions el sistema haurà de respondre amb un codi de resposta "429 Too Many Requests".

El sistema a seguir per especificar el límit de peticions per unitat de temps i el nombre actual de peticions que el client ja ha fet servir no està definit com un estàndard al protocol HTTP. Tot i així, l'especificació de capçaleres addicionals és lliure, per tant se'n poden incloure de pròpies que permetin al client valorar en quina situació es troba abans d'abusar del sistema.

Les capçaleres pròpies que es faran servir són:

1. X-RateLimit-Remaining: XXX  
Nombre de peticions que encara té disponibles el client en la finestra de temps donada.
2. X-RateLimit-Limit: XXX  
Màxim nombre de peticions que es poden fer en una finestra de temps.
3. X-RateLimit-Reset: XXXXXXXXXXXX  
Data en què es reiniciarà el comptador de peticions (o que començarà una nova finestra de temps). Un cop s'ha esgotat el nombre de peticions disponibles en una finestra de temps, no es permetrà a l'usuari fer noves peticions fins que s'arribi en aquest moment.  
El format serà en temps Unix (nombre de segons que han transcorregut des de l'1 de gener de 1970).

Exemple de capçaleres rebudes en resposta a una petició del client de tipus GET o HEAD:

```
HTTP/1.1 200 OK
Date: Wed, 30 Sep 2015 15:51:04 GMT
Content-Type: application/json
Content-Length: 6756
X-RateLimit-Remaining: 298
X-RateLimit-Limit: 300
X-RateLimit-Reset: 1443628800
```

Aquesta resposta indicaria que el client ha fet 2 peticions de 300 disponibles en la finestra de temps actual i que encara pot fer 298 peticions més. El moment en què el comptador es posarà a zero serà en el temps Unix de 1443628800 (30 de setembre 2015).

Les capçaleres retornades per en el cas de sobrepassar els límits, seran amb el codi http “429 Too many requests”.

### 4.1.4 Model de dades

L'objectiu de l'API és proporcionar certes dades en resposta a les peticions fetes per part d'un client. El format en què es presenten hauria de ser qualsevol que s'ajusti a l'arquitectura REST, *hypermedia* (dades i metadades).

Per conveniència i facilitat d'ús haurien de complir una sèrie de requisits:

1. Ser un format de fàcil lectura i escriptura per part d'humans.
2. Ser un format de fàcil processat i generació per part de màquines (software).
3. Que proporcioni els suficients mecanismes per a oferir gran varietat i tipus de dades de forma estructurada (*hypermedia*[27]).
4. El més lleuger possible. Que tingui la menor redundància possible en la seva sintaxi.

Es busca doncs un format codificat en text pla i que proporcioni dos tipus d'estructures diferenciades. Per una banda col·leccions de parells clau/valor encapsulades com a objectes o diccionaris. Per l'altre, llistes ordenades de valors (arrays).

Un format que compleix aquestes condicions és el JSON (JavaScript Object Notation) definit per l'organització ECMA[28] (European Computer Manufacturers Association) en l'estàndard ECMA-404[15].

És un format en text pla, amb un ampli suport en multitud de llenguatges de programació, concís, senzill de llegir i interpretar i que integra els dos tipus d'estructures de dades necessàries, objectes i arrays.

#### **Estructures de dades en JSON**

- **Objecte:** Un objecte consisteix en una llista no ordenada de parells clau/valor. Un objecte comença per un símbol { (clau esquerra) i acaba per un altre símbol } (clau dreta). Cada clau és seguida del símbol de : (dos punts). Per definir una propietat addicional s'interposa el símbol , (coma). Espais o salts de línia es poden posar on es vulguin per tal de facilitar-ne la lectura. Les claus han de ser estructures de tipus *string* (cadena de caràcters). Els valors al seu torn poden ser d'altres objectes o tipus de dades finals (com cadenes de caràcters, *arrays*, o valors escalars).

Exemple:

```
{
  "propietat-1" : "valor 1",
  "propietat-2" : "valor 2",
```

```
    "propietat-3" : 12.34
  }
```

- **Llista** o *array*: Els llistats són col·leccions ordenades de valors. Els índexs dels llistats estan implícits i per tant només cal separar els valors per el símbol , (coma). Els *arrays* hauran d'estar limitats per claudàtors d'obertura i tancament: [ ]

Exemple:

```
[ "valor 1", "valor 2", "valor 3" ]
```

- **Valor**: Un valor pot ser qualsevol tipus de les estructures vàlides de JSON (objectes, *arrays*, *strings* o números) a més dels valors *null*, *true* i *false*. Això permet crear estructures jerarquitzades.

Exemple:

```
{
  "propietat-1" : null,
  "propietat-2" : true,
  "propietat-3" : [ 1, 2, 3]
  "propietat-4" : {
    "sub-propietat-1" : false,
    "sub-propietat-2" : [ "str 1", "str 2" ]
  }
}
```

- **Cadena** o *string*: Una cadena és un seguit de caràcters Unicode encapsulats mitjançant el símbol " (cometes dobles). De manera similar al que seria una cadena en els llenguatges de programació C o Java, inclosos els caràcters d'escapament.

Exemple:

```
"cadena amb \"cometes escapades\" i salt de línia\n"
```

- **Número**: Números enters o decimals. Segueix el mateix concepte de número que els llenguatges C o Java amb l'excepció que els formats Octals i Hexadecimals no estan suportats.

Exemple:

```
12, 12.23, -2, -12.234, 0.123
```

## 4.2 Arquitectura del codi font

El codi del projecte s'implementarà en el llenguatge de programació Python. Les raons d'aquesta tria rau en:

- És un llenguatge ben establert popular i ben suportat per una comunitat activa de programadors.
- Disposa d'un gran repositori de mòduls ja programats i de fàcil accés amb llicències obertes amb funcionalitats que es poden reaprofitar de manera senzilla.
- És un llenguatge interpretat (no requereix compilació), de sintaxi senzilla, permet un ràpid prototipat i està àmpliament distribuït en multitud de sistemes operatius.
- És conegut pels programadors de l'organització Goteo que hauran de mantenir i estendre el codi font en el futur.

### 4.2.1 Components de programari

Per tal de facilitar el manteniment i dedicar els màxims recursos a programar les funcionalitats específiques de l'API, es procurarà reutilitzar el màxim de components de codi obert disponibles en algun dels repositoris de codi Python.

Els components necessaris per generar l'entorn de desenvolupament són:

- Intèrpret de llenguatge **Python[29]**, com a mínim en la versió 3.4. En el desenvolupament d'aquest programari es farà en base a la implementació estàndard, **CPython[29]** (versió 3.4).
- Base de dades **MySQL[6]** (versió 5.6), component que ve predeterminat per la natura de la implementació original de la plataforma Goteo.
- Base de dades volàtils (no persistents) **Redis[30]** (versió 3.2). S'utilitzarà per emmagatzemar dades temporals (com els límits d'ús o dades de *cache*)
- L'eina **Virtualenv[31]** (versió 15), necessària per crear entorns de treball virtuals aïllats. Permet que totes les dependències de programari s'instal·lin de manera independent de les que hi pugui haver per defecte en el sistema de desenvolupament. Això permetrà que l'entorn de desenvolupament sigui fàcilment reproducible.
- L'eina d'administració de paquets **Pip[32]** (*Pip Installs Packages*, versió 8.1). Escrita també en Python, en servirà per descarregar i actualitzar la resta de components que es faran servir del repositori de programari **PyPI[33]** (*Python*

*Package Index*). Un dels principals repositoris de distribució de mòduls reutilitzables de Python.

D'altra banda, els components (lliberies Python) més importants que es faran servir per a automatitzar i simplificar les tasques comunes a qualsevol API RESTful són:

- El *microframework* o entorn de treball **Flask**[34] (versió 0.10.1). Un conjunt de lliberies que faciliten la creació d'aplicacions web genèriques amb el mínim de dependències externes. Està basat al seu torn en la llibreria **Werkzeug**[35] (versió 0.11.9), un entorn de desenvolupament Python per a HTTP i Web.
- El *toolkit* **SQLAlchemy**[36] (versió 1.0.13), un conjunt d'eines per traslladar el llenguatge SQL propi de les bases de dades a l'entorn de programació Python.
- L'extensió **Flask-RESTful**[37] (versió 0.3.5), una abstracció de les convencions REST aplicades al microframework **Flask** que facilitaràn l'organització i programació del codi.
- L'entorn de test **Nose**[38] (versió 1.3.7). Una utilitat que facilitarà la creació de testos unitaris i la mesura de la qualitat del codi.

La llista completa de components es pot trobar al fitxer `requeriments.txt` en el codi font del programari.

## 4.2.2 Entorn de desenvolupament

Com a exemple d'entorn de desenvolupament s'utilitzarà una màquina amb el sistema operatiu **Ubuntu 14.04**[39]. Serveix tant una màquina virtual com una física.

Són necessaris diversos paquets de programari addicionals per tal de complir amb els mínims requeriments. Per conveniència, utilitzarem l'aplicació **Vagrant**[40] (versió 1.8.4) juntament amb **Virtualbox**[41] (versió 5.0), una combinació d'eines que ens permetran generar i configurar automàticament màquines virtuals. Ambdós programes són de codi obert.

La instal·lació d'aquests dos programes queda fora de l'abast d'aquest projecte, ja que es pot fer en qualsevol sistema operatiu de manera molt simple seguint les instruccions disponibles en els seus llocs de descàrrega.

La configuració i inicialització de l'aplicació **Vagrant** es troba en el fitxer `Vagrantfile`, inclosa en el codi, cosa que fa fàcilment reproduïble per qualsevol que vulgui instal·lar tot l'entorn virtual de desenvolupament amb una sola comanda de terminal.

## Comandes d'inicialització:

Caldrà executar aquestes comandes en un directori on prèviament s'hi hagi copiat tot el codi font de l'aplicació.

1. Inicialització de l'entorn de treball (el primer cop que s'executi aquesta comanda baixarà tots els arxius necessaris i pot tardar un temps considerable):

```
$ vagrant up
```

2. Entrada (*login*) a la màquina virtual:

```
$ vagrant ssh
```

3. Finalment ja només queda iniciar l'aplicació de l'API en mode desenvolupament, això crearà un servidor web amb l'aplicació a la qual es podrà accedir des de qualsevol client adequat:

```
(goteoapi) vagrant@vagrant-ubuntu-trusty-64:~/goteo-api$ ./run
```

L'aplicació quedarà executant-se esperant peticions, la sortida del programa serà una cosa similar a:

```
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger pin code: 321-622-619
```

En aquests moments ja es pot provar l'aplicació, només cal apuntar a l'URL <http://localhost:5000/> amb qualsevol client web (per exemple un navegador web o la comanda `curl`).

4. Per interrompre el programa es pot prémer la combinació de tecles `Ctrl-C` i, per sortir (*logout*) de la màquina virtual només caldrà executar la comanda:

```
$ exit
```

5. Per destruir completament la màquina virtual:

```
$ vagrant destroy
```

Alternativament, si ja es disposa d'un sistema amb l'entorn **Ubuntu 14.04** o no es vol fer servir Vagrant, es poden executar algunes de les comandes incloses en l'arxiu `vagrant_provision.sh` o instal·lar els paquets de manera manual, tot depenent dels paquets ja existents en sistema. La instal·lació en altres sistemes operatius també és possible i s'hauran de seguir procediments similars per a cadascun d'ells.

Aquestes altres possibilitats d'instal·lació estan detallades en l'arxiu `INSTALL.md` que es distribueix amb el codi.

### 4.2.3 Control de qualitat

És convenient la simulació d'estats ficticis en què es pot trobar l'aplicació per tal de provar-ne totes les seves funcionalitats. Automatitzar totes aquestes situacions de prova permetrà establir un protocol de test del codi abans de posar-lo en producció.

La idea és generar un conjunt de dades a la base de dades que simulin casos reals en els quals es pot trobar l'aplicació en funcionament. Aleshores, s'escriurà codi de test addicional que simularà peticions sobre aquestes dades. D'aquesta manera, es poden detectar errors en el codi abans que es publiquin a un servidor de producció. Com més situacions de casos reals es puguin simular, més ben cobert estarà el codi i més segurs podrem estar del fet que la nostra aplicació no fallarà. Naturalment, aquest no és un procés infal·libre però és molt millor que només provar casos aïllats manualment.

A més, un avantatge considerable és que permet desenvolupar futures característiques en l'aplicació estan raonablement segurs que es mantindrà la compatibilitat amb l'especificació anterior (doncs cada vegada s'hauran de complir els tests programats).

Un dels paràmetres per mesurar la quantitat de situacions reals analitzades ens la dóna el grau de cobertura del codi generat pels tests escrits. És a dir, quan s'executa un test, es mesura quines línies de codi s'executen perquè aquest es dugui a terme satisfactòriament. Un grau de cobertura del 100% indicaria que els nostres tests han executat totes les línies de codi que hem escrit en el nostre programa. De totes maneres sempre hi ha situacions on la nostra aplicació encara podria tenir fallades tot i passar els tests correctament. Serà responsabilitat futura dels mantenidors del programari anar afegint més tests per tal de tenir més situacions cobertes.

Naturalment, el llenguatge de programació (o les eines associades) ha de permetre aquesta mesura. En el nostre cas, **Python** disposa de diverses eines sobre les quals escriure tests i mesurar la qualitat del codi. Farem ús de la llibreria **nose** per executar els tests que s'escriuran i mesurar el grau de cobertura del codi que ens indicarà el nivell de qualitat.

Establirem un objectiu d'un mínim del 80% de cobertura de codi per a la nostra API.

Convenientment, agruparem tots els tests en un *script* que ens permetrà executar i mesura l'índex de cobertura amb una sola comanda:

```
(goteoapi) vagrant@vagrant-ubuntu-trusty-64:~/goteo-api$ ./run-tests
```

Aquesta comanda és només un *script* que configura la llibreria **nose**, encarregada d'executar tots els tests. Per mesurar el grau de cobertura només cal indicar-ho amb una comanda addicional:

```
...:~/goteo-api$ ./run-tests --with-coverage
```

El resultat d'aquesta comanda hauria de ser alguna cosa similar a:

```
.....
Name                               Stmts  Miss  Cover
-----
goteoapi.py                         45     15    67%
goteoapi/auth/controllers.py        3      0   100%
goteoapi/auth/decorators.py         95     13    86%
goteoapi/auth/resources.py          15      0   100%
goteoapi/base_resources.py          133     17    87%
goteoapi/cacher.py                  118     25    79%
goteoapi/calls.py                   0      0   100%
goteoapi/calls/controllers.py        8      0   100%
goteoapi/calls/models.py            187     19    90%
goteoapi/calls/resources.py         109      4    96%
...
goteoapi/projects.py                0      0   100%
goteoapi/projects/controllers.py      8      0   100%
goteoapi/projects/models.py          382     81    79%
goteoapi/projects/resources.py       170     30    82%
goteoapi/ratelimit.py                43      4    91%
goteoapi/users.py                   0      0   100%
goteoapi/users/controllers.py         8      0   100%
goteoapi/users/models.py             229     59    74%
goteoapi/users/resources.py           73     15    79%
-----
TOTAL                               2999   559    81%
-----
Ran 63 tests in 7.749s

OK
```

En aquest exemple veiem un grau de cobertura del 81% que satisfaria les nostres pretensions.

## 4.3 Estructura del programari

El programari està desenvolupat seguint una estructura modular. Per una banda, s'estructurarà el codi al voltant d'un nucli que proporcioni una funcionalitat mínima i un conjunt de mètodes reutilitzables. Per l'altra, es programaran diferents paquets o mòduls que afegiran funcionalitat al codi base. Alhora, això proporcionarà una base per tal que terceres parts puguin estendre l'API de manera independent.

Els mòduls addicionals s'incorporaran al codi mitjançant una variable en l'arxiu de configuració del sistema. En aquest arxiu de configuració també s'hi especificaran les claus d'accés a la base de dades i d'altres valors personalitzables d'interès. Es farà servir la capacitat de Python d'importar llibreries en temps d'execució per executar el codi dels mòduls addicionals quan calgui.

El conjunt de mètodes reutilitzables programats en el nucli del sistema seran accessibles per la resta de mòduls per tal de facilitar tasques comunes. Dins dels

mètodes comuns trobarem funcions de control d'accés (que implementen els protocols d'autenticació i limitació de peticions), mètodes per facilitar el *cache* d'informació que té un cost computacional alt, validació del format de variables d'entrada, classes base que poden heretar cada mòdul final, etc.

### 4.3.1 Estructura d'arxius:

En Python, els paquets o mòduls es defineixen: o bé, mitjançant el nom de l'arxiu que conté el codi, o a través d'un subdirectori que contingui l'arxiu d'inicialització `__init__.py`. Al seu torn, successius submòduls poden ser definits a dins de cada subdirectori.

Per facilitar-ne la llegibilitat, el test i la capacitat de personalització establirem tants mòduls i submòduls com ens sigui necessari per a minimitzar el nombre de tasques que ha de fer cada mòdul.

#### Arbre d'arxius:

```
├── config.py
├── config.py.example
├── config_test.py.dist
├── config_vagrant.py.dist
├── console.py
├── deployer
├── goteoapi/
│   ├── auth/
│   │   ├── ...
│   ├── base_resources.py
│   ├── cacher.py
│   ├── calls/
│   │   ├── ...
│   ├── categories/
│   │   ├── ...
│   ├── config.py
│   ├── controllers.py
│   ├── helpers.py
│   ├── __init__.py
│   ├── invests/
│   │   ├── ...
│   ├── licenses/
│   │   ├── ...
│   ├── location/
│   │   ├── ...
│   ├── models/
│   │   ├── ...
│   ├── projects/
│   │   ├── ...
│   ├── ratelimit.py
│   ├── tests/
│   │   ├── ...
│   ├── users/
│   │   ├── ...
├── goteoapi_cli/
│   ├── ...
├── goteoapi_digests/
│   ├── ...
├── goteoapi_reports/
```

```

├── ...
├── INSTALL.md
├── LICENSE
├── README.md
├── requirements.txt
├── run.py
├── Vagrantfile
├── vagrant_provision.sh

```

## Primer nivell d'arxius

Al primer nivell de l'arbre d'arxius trobem els arxius de configuració i les utilitats que ens permeten executar una instància del codi, executar tests o d'altres utilitats bàsiques.

Arxiu	Descripció
<code>config.py</code>	Arxiu de configuració per defecte, és l'únic arxiu que no es distribueix amb el codi. Cada instal·lació haurà de generar aquest arxiu per funcionar.
<code>config.py.example</code>	Exemple amb les variables de configuració comentades per distribuir amb el codi.
<code>config_test.py.dist</code>	Arxiu de configuració que s'usa en els tests per defecte (es possible definir un arxiu de test personal)
<code>config_vagrant.py.dist</code>	Arxiu de configuració preparat per funcionar amb la base de dades de desenvolupament de la màquina virtual Vagrant.
<code>Vagrantfile</code>	Arxiu de configuració de la màquina virtual Vagrant
<code>vagrant_provision.sh</code>	Arxiu d'aprovisionament de la màquina virtual Vagrant. S'encarrega de configurar-hi tots els serveis necessaris i d'instal·lar-hi totes les dependències automàticament.
<code>console.py</code>	Arxiu base per fer funcionar les utilitats de la línia de comandes (paquet <i>goteoapi_cli</i> ).
<code>run.py</code>	Arxiu base per executar l'aplicació API. S'encarrega d'importar tots els mòduls i iniciar un servidor web en el port 5000 en cas necessari (una instal·lació de producció no necessita d'aquest servidor propi).
<code>INSTALL.md</code>	Arxiu de documentació (en anglès) sobre totes

	les possibilitats de com instal·lar el sistema.
<code>README.md</code>	Arxiu de documentació (en anglès) amb el resum del propòsit de l'aplicació, descripció d'ús i comandes bàsiques.
<code>LICENSE</code>	La llicència amb la que es distribueix el codi (Llicència Pública General Affero de GNU).
<code>deployer</code>	<i>Script</i> programat en BASH que permet l'automatització en la instal·lació de dependències.
<code>requirements.txt</code>	Llistat de dependències: llibreries Python requerides pel sistema (arxiu usat per l' <i>script deployer</i> per instal·lar-les)
<code>goteoapi/</code>	Mòdul amb el nucli de l'aplicació. Conté els submòduls que implementen els recursos principals a part de l'estructura primària.
<code>goteoapi_cli/</code>	Mòdul amb comandes per executar en mode CLI (via terminal). Eines addicionals que faciliten el manteniment o la gestió d'alguns aspectes de l'aplicació (com la gestió del <i>cache</i> ).
<code>goteoapi_reports/</code>	Mòdul extra que implementa els recursos d'informació estadística.
<code>goteoapi_digests/</code>	Mòdul extra capaç d'agrupar d'altres recursos per franges de dates i presentar els resultats en una sola petició.

## Segon nivell d'arxius

Al segon nivell de l'arbre d'arxius hi trobem els mòduls principals que component l'API, el nucli (paquet `goteoapi`) i els mòduls extra (`goteoapi_reports`, `goteoapi_digests`, `goteoapi_cli`).

Cadascun d'aquests mòduls contindrà un arxiu d'inicialització i diferents submòduls en funció de les seves necessitats:

### 4.3.2 Mòdul *goteoapi*

El mòdul *goteoapi* és la base de l'aplicació. Aquí s'implementen totes les rutines principals que poden ser reutilitzades per altres mòduls. Per exemple el sistema d'autenticació, el de *cache* o els models d'accés a la base de dades.

Arxiu	Descripció
<code>__init__.py</code>	En Python, un mòdul definit com a directori sempre ha d'incloure aquest arxiu, encara que estigui buit. En aquest cas s'inicialitzen aquí l'accés a la base de dades, el subsistema de documentació i el <i>cache</i> de l'aplicació.
<code>config.py</code>	Arxiu de les variables de configuració per defecte. En cas que l'arxiu de configuració no defineixi alguna variable s'agafarà el valor aquí present.
<code>controllers.py</code>	Porta d'entrada al mòdul principal, el paquet <i>goteoapi</i> . L'execució de l'arxiu <code>run.py</code> crida aquest arxiu juntament amb els paquets addicionals configurats en <code>config.py</code> .  Aquí es defineixen les rutes generals del sistema ( <code>/</code> ), la gestió d'errors, l'afegit de capçaleres HTTP comunes i es fa la crida a la resta de submòduls que gestionen les altres rutes del paquet ( <code>/projects/</code> , <code>/login/</code> , <code>/users/</code> , etc.).
<code>base_resources.py</code>	Aquí s'agrupen classes i funcions utilitzades per qualsevol submòdul que vulgui implementar els filtres comuns i el mateix tipus de resposta JSON.  Les classes finals poden heretar d'algunes definides aquí per tal de proveir-se de mètodes de comprovació de les variables d'entrada sense haver de reescriure codi innecessàriament.
<code>cache.py</code>	Funcions i decoradors (a l'estil Python: funcions que permeten canviar el comportament d'altres funcions sense mantenir-ne el codi) que implementen el sistema de <i>cache</i> de l'API. Aquest sistema el pot fer servir qualsevol submòdul per desar informació calculada per una funció que tingui un cost computacional alt en una base de dades Redis i retornar-la en successives peticions automàticament. La validesa de l'estona de <i>cache</i> és configurable.
<code>ratelimit.py</code>	De manera similar al sistema de <i>cache</i> , en aquest arxiu es proporcionen funcions i decoradors que els submòduls poden fer servir per limitar el nombre de peticions a cada recurs.

	També es fa servir la base de dades Redis per desar les peticions per usuari remot.
<code>helpers.py</code>	Distintes classes i funcions auxiliars del sistema. El càlcul d'un percentatge, càlcul de dates, obtenció d'URLs de la web de Goteo, etc.
<code>categories/</code>	Submòdul per la gestió del recurs <code>/categories/</code> i derivats. Es crida a aquest submòdul (i de la resta del paquet <code>goteoapi</code> ) des de l'arxiu <code>controllers.py</code> . Retorna les categories de la web de Goteo on s'hi organitzen usuaris i projectes.
<code>auth/</code>	Submòdul d'autenticació per la gestió del recurs <code>/login/</code> . S'encarrega de comprovar les credencials dels usuaris, generar els <i>tokens</i> d'autenticació i validar-los. Proporciona el decorador <code>requires_auth</code> que es fa servir per qualsevol altre submòdul per definir quin tipus d'autenticació requereix el recurs.
<code>calls/</code>	Submòdul per la gestió del recurs <code>/calls/</code> i derivats. En aquest recurs s'obtenen llistats i detalls de les convocatòries capital regadiu així com dels projectes implicats.
<code>invests/</code>	Submòdul per la gestió del recurs <code>/invests/</code> i derivats. Proporciona tota la informació referent a les aportacions monetàries dels usuaris als projectes. Per qüestions de privacitat, no dona la relació directe amb l'usuari que ha fet la donació.
<code>licenses/</code>	Submòdul per la gestió del recurs <code>/licenses/</code> i derivats. Les llicències (i les metadades associades) usades pels retorns col·lectius dels projectes es llisten aquí.
<code>projects/</code>	Submòdul per la gestió del recurs <code>/projects/</code> i derivats. El llistat de projectes disponibles a Goteo i el detall de cadascun d'ells es troben sota aquest recurs.
<code>users/</code>	Submòdul per la gestió del recurs <code>/users/</code> i derivats. Llistat i detalls dels usuaris de Goteo.
<code>location/</code>	Dins aquest submòdul s'agrupen les classes que implementen les particularitats del sistema de geolocalització dels ítems de Goteo.  Són elements geolocalitzables els projectes, les aportacions monetàries i les convocatòries capital regadiu. Cada subsistema que implementi geolocalització fa servir aquests mètodes comuns per, per exemple, fer cerques en una àrea al voltant d'un punt geogràfic (latitud

	i longitud).
<code>models/</code>	En aquest submòdul hi trobarem la resta de models per accedir a la base de dades i que no es troben en d'altres submòduls per no disposar (encara) d'un recurs d'accés propi.
<code>tests/</code>	Col·lecció de scripts que són cridats quan es volen executar els tests del sistema. Estan escrites aquí simulacions de peticions a l'API reals de manera que es provin tots els recursos implementats.  L'execució d'aquests tests en cada modificació del codi ens permetrà estar (raonablement) segurs que no s'introduiran falles (bugs) en situacions ja provades.

### 4.3.3 Extensibilitat

L'estructura del programari de l'API és extensible. Això és així per permetre un alt grau de personalització en diferents instal·lacions sense necessitat de tocar el codi principal. Per una banda qualsevol correcció o millora que pugui fer una tercera part interessada en el codi podrà ser incorporada en la següent versió del software per al benefici de tots. Per l'altra, aquesta tercera part podrà generar un mòdul propi amb els canvis que consideri oportuns sense haver de tocar el nucli.

Com a part del programari, s'han programat tres mòduls addicionals que proporcionen funcionalitats extres (el mòdul d'estadístiques *goteoapi\_reports*, el de agrupacions de dades *goteoapi\_digest* i les utilitats de la línia de comandes *goteoapi\_cli*). Aquests mòduls s'inclouen juntament amb el mòdul principal (*goteoapi*) però cal activar-los explícitament a l'arxiu de configuració.

Per altra banda, el fet de generar alguns mòduls ja distribuïts amb el codi serveix d'exemple o punt de partida per a qualsevol que en vulgui incorporar de nous.

El procediment per generar un nou mòdul és simple, l'script principal `run.py` s'encarrega d'importar tots els arxius inclosos en la variable de configuració `MODULES`.

`run.py` (detall de codi):

```
# import sub-modules controllers
# Minimal endpoints:
__import__('goteoapi.controllers')

# Additional modules from config
if hasattr(config, 'MODULES'):
    for i in config.MODULES:
        __import__(i)
```

`config.py` (detall de codi):

```
MODULES = {  
    # reports endpoints  
    'goteoapi_reports.controllers',  
    # digests endpoints  
    'goteoapi_digests.controllers'  
}
```

Els exemples anteriors mostren com es carreguen els mòduls addicionals, per una banda a l'arxiu `config.py` es genera un objecte de tipus diccionari amb els arxius a processar (per exemple `goteoapi_reports.controllers`). La sentència interna de Python, `__import__()`, s'encarregarà d'afegir i processar el codi present en l'arxiu equivalent del mateix nom (`goteoapi_reports.controllers` carregarà l'arxiu `goteoapi_reports/controllers.py`).

El submòdul podrà mantenir una organització interna pròpia que serà responsabilitat del seu dissenyador. Amb aquest senzill procediment es poden afegir tants submòduls com calgui.

### Submòdul *goteoapi\_reports*

Aquest submòdul està dissenyat per la *Fundación Goteo* i està pensat per obtenir dades estadístiques de diversos paràmetres de l'activitat de la web. Valors com el nombre de projectes publicats, percentatges d'èxit en el finançament de projectes, total de diners obtinguts, retornats, etc. La llista completa es troba documentada en la documentació oficial de l'API (<https://developers.goteo.org/doc/reports/> també disponible com annex).

El mòdul proporciona quatre recursos propis en els URLs `/reports/projects/`, `/reports/community/`, `/reports/money/`, `/reports/rewards/` i `/reports/summary/`. Totes les estadístiques es poden generar d'acord amb diferents paràmetres de cerca (per exemple una franja temporal, un projecte en concret, una localització geogràfica, etc.).

### Submòdul *goteoapi\_digests*

En alguns casos, pot ser convenient agrupar les dades d'altres resultats per evitar haver de fer multitud de consultes a l'API amb diferents paràmetres de cerca. Aquest submòdul proporciona aquesta funcionalitat i retorna en un format de llista (*array*) la mateixa informació d'altres recursos per franges de dates predefinides.

La informació retornada pot en grups de mesos o anys segons s'especifiqui en la petició. Es pot obtenir la informació dividida en 12 períodes corresponent a cada mes d'un any concret. O bé dividida en tants períodes com anys de vida té la plataforma Goteo, en aquest cas es presentaran les dades de cada any en global per separat.

Tots els recursos proporcionats per aquesta extensió s'organitzen amb el prefix `/digests/`. La documentació completa també està disponible en la pàgina oficial de l'API (<https://developers.goteo.org/doc/digests/>, i en l'annex inclòs amb aquesta memòria).

L'aplicació en línia <https://stats.goteo.org> fa un ús intensiu d'aquestes dades juntament amb el submòdul `goteoapi_reports`.

## Submòdul `goteoapi_cli`

L'últim submòdul inclòs amb el codi de l'API és el que proporciona accés a utilitats de la línia d'ordres. Són utilitats de gestió interna i no proporcionen cap recurs addicional a l'API.

L'accés a les utilitats de la línia d'ordres es fa mitjançant un terminal amb la comanda `./console`. Una crida sense paràmetres proporciona un llistat de les comandes disponibles:

```
$ ./console
usage: console.py [-?] {renewcache,clearcache,shell,crontab,runserver} ...

positional arguments:
  {renewcache,clearcache,shell,crontab,runserver}
    renewcache      Renew cached keys by executing the functions with
                    stored params
    clearcache      Clears all cache keys
    shell           Runs a Python shell inside Flask application context.
    crontab         Installs/Removes a crontab if is installed
    runserver       Runs the Flask development server i.e. app.run()

optional arguments:
  -?, --help       show this help message and exit
```

Les comandes realment programades (les altres les proporciona el *framework* de treball Flask per defecte) en el submòdul `goteoapi_cli` són:

Comanda	Descripció
<code>clearcache</code>	Si es proporciona aquest argument, es realitzarà un buidatge complet del <i>cache</i> intern de l'aplicació. És útil en situacions en què la <i>cache</i> està activada i es vol forçar l'obtenció de contingut nou de l'API.
<code>renewcache</code>	Aquesta ordre està pensada per a recórrer totes les claus de <i>cache</i> emmagatzemades i tornar a fer la mateixa petició a l'API de manera interna per tal de renovar-ne les dades. D'aquesta manera, operacions que poden trigar un temps substancial a completar-se es fan internament, cosa que farà que les peticions externes s'obtinguin de resultats en <i>cache</i> i, per tant, se serveixin molt més ràpid.
<code>crontab</code>	L'ordre anterior, necessita executar-se periòdicament per tal de renovar les claus de <i>cache</i> que estiguin a punt

d'expirar. Aquesta comanda permet configurar automàticament el dimoni CRON dels sistemes tipus UNIX per tal d'automatitzar-ho.
---

## 4.4 Casos d'interès

En aquest capítol es volen mostrar algunes particularitats de la programació de l'API que poden ser interessants, ja sigui pel tipus d'algorisme implementat, comprendre millor com està organitzat el codi en conjunt o entrar en detall en com s'han solucionat alguns dels problemes que l'aplicació pràctica de l'API ha posat de manifest.

Es farà una introducció a l'entorn de treball (*framework Flask*), que és la base sobre la qual se sustenta tota la programació que segueix. Això permetrà comprendre millor les regles i convencions que se segueixen en el codi dels diferents mòduls.

D'altra banda, també es farà una mirada a un parell de situacions particulars, en concret el sistema de *cache*, pensat per optimitzar l'accés a l'API, especialment en casos de peticions elevades o concurrents.

Finalment, donat que l'organització Goteo dona una gran importància a la possibilitat de desenvolupar aplicacions que facin ús de dades geogràfiques en els recursos més importants oferts (com són per exemple els projectes, les aportacions econòmiques o les campanyes de capital regadiu), s'entrarà una mica en detall de com s'ofereixen aquestes dades i com es calculen (Filtrat per coordenades geogràfiques).

### 4.4.1 Framework de treball: Flask

Flask és un entorn de desenvolupament web per a Python enfocat a la facilitat d'ús i, sobretot, al rendiment i extensibilitat. Normalment anomenat "micro-framework" per la seva filosofia de no incloure més que el mínim de requeriments per a la construcció d'una aplicació web. Proporciona un conjunt de mètodes amb els quals començar ràpidament un projecte que compleixi la filosofia RESTful.

Per altra banda, no fa assumpcions sobre la resta de components a usar (com per exemple l'accés a bases de dades o la generació de vistes HTML). Gràcies a la simplicitat inherent a les representacions de dades de l'API (doncs només es presenta informació amb tipus de dades JSON), aquest entorn sembla l'ideal per la construcció de l'API.

La mínima aplicació escrita en Flask: Es crea un servidor web amb el recurs principal (o ruta) `/` que retorna el text "*Hello, World!*" en visitar-ho amb un navegador web.

Exemple, `hello.py`:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello, World!\n"

if __name__ == "__main__":
    app.run()
```

Descripció del codi:

1. La primera línia importa la classe Flask.
2. La segona inicia una instància d'aquesta classe. El paràmetre rebut correspondrà al nom del mòdul programat. És pràctica comú utilitzar la propietat interna `__name__` que corresponent conté el nom de l'aplicació segons el mètode en què s'ha iniciat (això és útil perquè el mateix codi sigui vàlid en diverses maneres d'iniciar l'aplicació).
3. Després es defineix una funció qualsevol (en el nostre cas `hello_world()`) embolcallada amb un decorador (`@app.route('/')`). Aquest decorador indica a Flask quin URL ha d'executar el codi de la funció. Aquesta funció retorna el contingut desitjat.
4. Finalment, per facilitar el desenvolupament, les dues últimes línies indiquen que si el mètode d'execució és `__main__` (corresponent a una execució per línia d'ordres), s'iniciï un servidor web inter en el port 5000 per proves. Altres mètodes d'inicialització posaran un valor diferent de la variable `__name__` i aquest servidor no s'iniciarà.

Per iniciar el servidor, es pot fer via terminal:

```
$ python hello.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

En un altre terminal podem provar l'execució del codi amb la comanda `curl`:

```
$ curl http://127.0.0.1:5000/
Hello, World!
$
```

Adicionalment, caldrà afegir algunes extensions que ens proporcionaran accés a la base de dades, facilitaran una mica més la construcció d'un servei RESTful, generaran documentació, etc.

Una d'aquestes extensions és *Flask-RESTful*, llibreria que permet organitzar els recursos en classes separades alhora que permet formatar les capçaleres i dades automàticament.

Exemple d'aplicació *Flask-RESTful*:

```
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, '/')
```

En aquests cas, en comptes d'utilitzar “decoradors”, simplement assignem les classes que gestionaran els nostres recursos a l'URL assignat. La llibreria també s'encarregarà de transformar un objecte tipus diccionari de Python en un format de dades JSON.

Per exemple, en el cas anterior el retorn complet a la petició HTTP (capçaleres incloses) seria:

```
$ curl -D- http://127.0.0.1:5000/
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 25
Server: Werkzeug/0.11.9 Python/3.5.1
Date: Sun, 19 Jun 2016 15:37:20 GMT

{
  "hello": "world"
}
```

Una altra extensió àmpliament utilitzada és *SQLAlchemy*, una llibreria que ens proporcionarà les eines necessàries per generar sentències SQL d'accès a la base de dades MySQL sense necessitat de conèixer aquest llenguatge en profunditat. *SQLAlchemy* transforma les taules SQL en classes Python, permetent una millor integració amb la resta de components.

En general les classes corresponents a les taules MySQL que formen l'estructura de Goteo estan definides als arxius “models.py” dels diferents submòduls de l'API (per exemple, els models de projectes els trobarem dins de `goteoapi/projects/models.py`).

### 4.3.2 Estratègies de *cache*

Hi ha situacions on la lògica de processament de certes dades és complexa i requereix un temps considerable de càlcul. Especialment complexes són les operacions que requereixen el filtratge i encreuament de moltes taules diferents de la base de dades, com es dona, per exemple, en el càlcul de dades estadístiques o en el filtrat per coordenades geogràfiques.

Amb la quantitat de dades actual de la base de dades de Goteo i, utilitzant una màquina amb capacitat de computació limitada (com podria ser un ordinador personal), els temps d'espera per a algunes dades pot superar els 60 segons.

Naturalment, aquests temps d'espera poden ser crítics perquè els clients que facin servir l'API siguin realment útils, especialment si van destinats al consum de persones.

Un altre aspecte a considerar és el fet que si es reben moltes peticions concurrents al mateix recurs, això pot col·lapsar ràpidament la màquina de servei, cosa que pot resultar molt ineficient i car (doncs només es podrien superar aquestes situacions mitjançant l'escalat del maquinari).

Per aquestes raons, s'ha dissenyat un sistema general de memòria cau o *cache* de dades (*Fig. 14*) que permet, primer, desar les dades calculades d'una petició complexa en una base de dades alternativa i, després, recuperar aquestes dades en subseqüents peticions.

Com a base de dades alternatives s'usa Redis, per la seva simplicitat ràpida i escalabilitat (to i que es possible usar d'altres solucions). L'extensió *Flask-Cache* ens facilitarà aquesta tasca ja que proveeix de mètodes i "decoradors" pensats per desar i recuperar les dades d'una funció qualsevol.

Amb la idea de millorar el sistema simple de *cache* de *Flask-Cache*[42], crearem un nou decorador per fer servir en la nostra API. Aquest mètode ens permetrà controlar el temps de *cache* de manera arbitrària (temps fins que qualsevol petició al recurs tornarà a executar el càlcul de la base de dades).

Una altra millora que s'introduirà, serà la capacitat de poder fer el pas invers al procés de *cache*, és a dir, a partir del resultat desat, poder obtenir la funció i els paràmetres originals que l'han originat. D'aquesta manera serà possible renovar periòdicament tots els resultats en *cache* abans que expirin. Aquesta estratègia permetrà executar un procés paral·lel a l'API que s'encarregui de fer les peticions costoses a la base de dades (i alliberant els clients web de la tasca, ja que sempre rebran el contingut desat).

Totes aquestes tasques estan implementades en l'arxiu `goteoapi/cacher.py`. Els mètodes que vulguin fer ús del *cache* només cal que facin servir el decorador `@cacher`, això crearà una clau única per la funció i els paràmetres i, en el cas que no existeixi un resultat per a aquesta clau, executarà la funció i desat el resultat a la

base de dades Redis. En el cas que existeixi un resultat previ, simplement retornarà el resultat en *cache*.

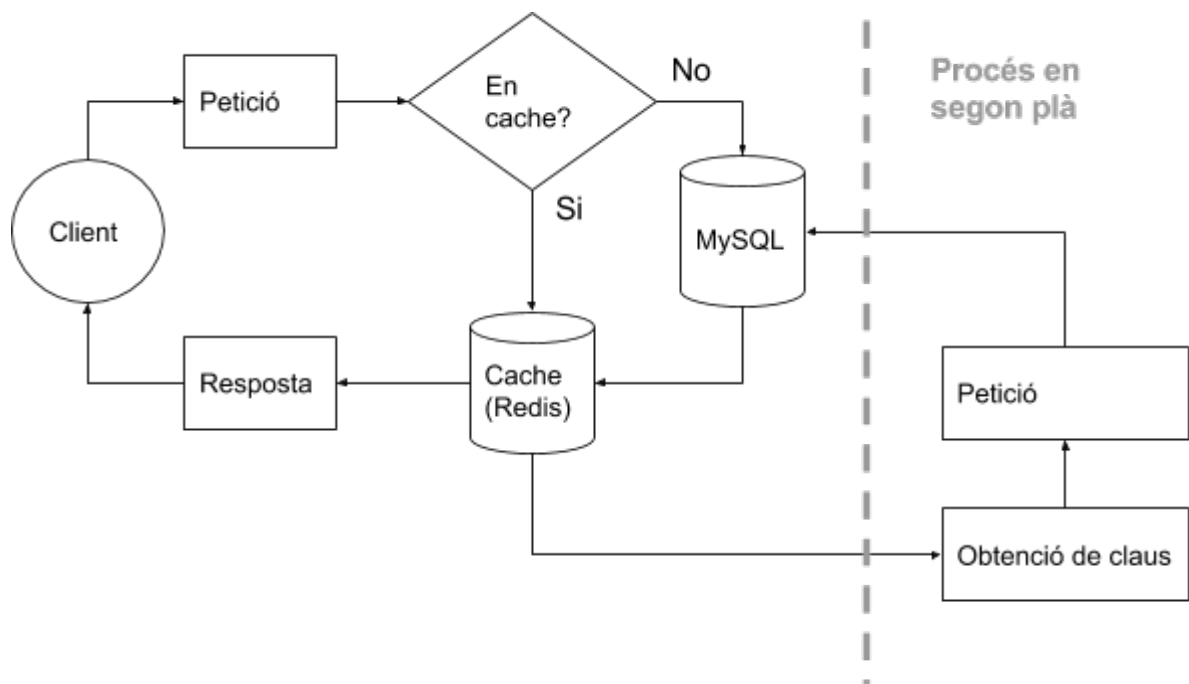


Figura 14: Procés de cache a l'API usant Redis

Exemple de codi usant el decorador `@cacher` usat en una funció:

```

from goteoapi.cacher import cacher

@cacher
def get_simple(num=0):
    return num
  
```

Com a procés en segon pla, farem ús d'una comanda en la línia d'ordres, proporcionada pel mòdul `goteoapi_cli`. Aquesta comanda simplement recorrerà totes les claus una per una i anirà executant cadascuna de les funcions associades. El resultat serà desat a la base de dades Redis.

Aquest procés es pot executar amb la comanda `renewcache` del paquet `goteoapi_cli`:

```

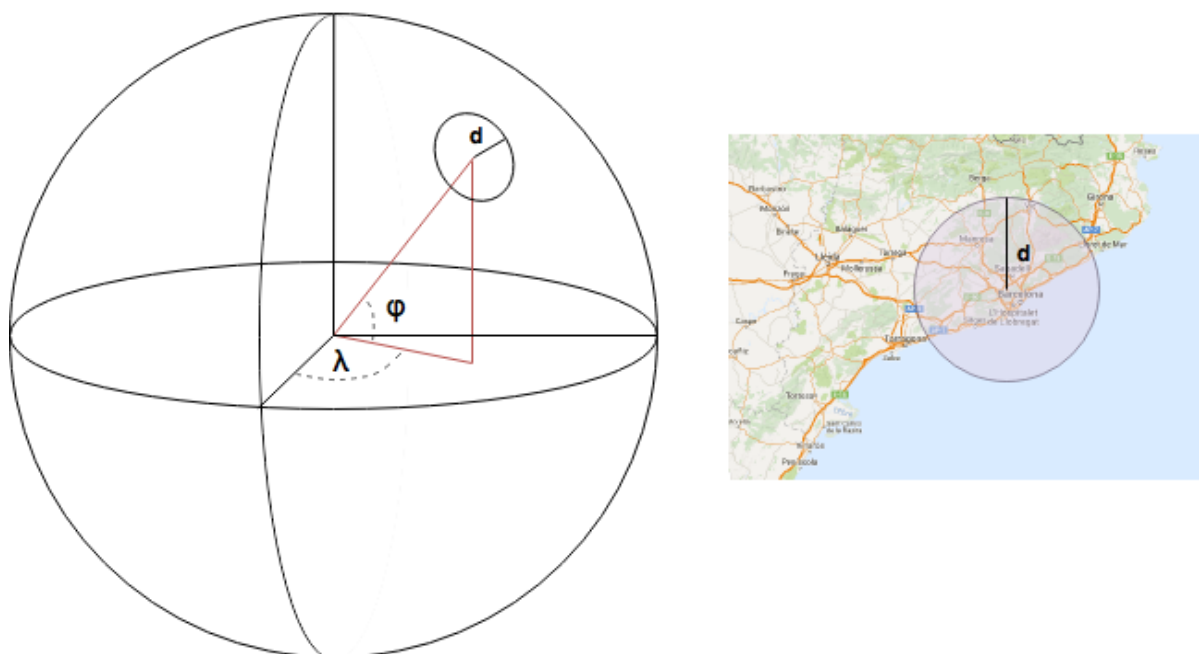
$ ./console renewcache -e
...
2016-06-19 19:40:21,999 INFO sqlalchemy.engine.base.Engine SELECT
count(distinct(message.project)) AS count_1
FROM message
WHERE message.user = %s
2016-06-19 19:40:21,999 INFO sqlalchemy.engine.base.Engine ('owner-project-
passing',)
Run with option --force (-f) in order to force the renew of still active keys
2016-06-19 19:40:22,016 INFO sqlalchemy.engine.base.Engine ROLLBACK
$
  
```

Aquesta comanda es pot executar de manera automàtica periòdicament en el servei *Cron* o *Systemd* de les màquines Unix/Linux.

### 4.4.3 Filtrat per coordenades geogràfiques

Algunes de les dades que es poden obtenir contenen informació geogràfica associada (en forma de latitud i longitud). Els projectes, les convocatòries i les aportacions monetàries, per exemple, tenen taules associades amb aquestes dades. Per requeriments, serà necessari poder filtrar aquestes taules segons la seva proximitat a un punt geogràfic concret.

Ens interessa poder trobar totes les entrades d'una taula amb dades de latitud i longitud dins d'un radi  $d$  (en kilòmetres, *Fig. 15*) a partir d'un punt de referència. Això ens permetria, per exemple, trobar tots els projectes que s'han finançat a 70 km a la rodona de Barcelona.



*Figura 15: Projecció de l'àrea d'un cercle en la superfície d'una esfera*

Com a solució, es podria implementar la fórmula de Haversine[43], que permet calcular la distància més curta entre dos punts en la superfície d'una esfera a partir de les seves coordenades geogràfiques (latitud i longitud).

La fórmula de Haversine és prou complexa perquè suposi un problema de rendiment en haver-la d'aplicar en una cerca per tots els resultats d'una base de dades.

Una altra possibilitat és aplicar la llei esfèrica dels cosinus[44] que és més simple però amb la possibilitat de generar més errors per arrodoniment. No obstant la donarem per vàlida doncs no cal una gran precisió.

Llei esfèrica dels cosinus:

$$d = \text{acos}(\sin \varphi_1 \cdot \sin \varphi_2 + \cos \varphi_1 \cdot \cos \varphi_2 \cdot \cos \Delta\lambda) \cdot R$$

d: Distància entre els dos punts (en kilòmetres)  
 $\varphi_1$ : Latitud del punt d'inici  
 $\varphi_2$ : Latitud del punt de destí  
 $\Delta\lambda$ : Diferència entre la longitud de destí i d'inici  
R: Radi de la terra (6.371km)

La implementació en llenguatge SQL seria:

```
Select Id, Lat, Lng,
      (acos(sin( $\varphi$ )*sin(radians(Lat)) +
          cos( $\varphi$ )*cos(radians(Lat))*cos(radians(Lng)- $\lambda$ )) * R) As D
From taula
Where acos(sin( $\varphi$ )*sin(radians(Lat)) +
        cos( $\varphi$ )*cos(radians(Lat))*cos(radians(Lng)- $\lambda$ )) * R < d
```

On els símbols:  $d$ ,  $\varphi$ ,  $\lambda$  i  $R$  seran valors que es passaran en el moment de fer la crida.  $Id$ ,  $Lat$  i  $Lng$  son les columnes de la taula MySQL on es desen l'identificador de la taula i els valors de latitud i longitud.

D'altra banda, per augmentar l'eficiència de la cerca, es farà un filtre previ, un "primer tall" amb un càlcul previ de les latituds i longituds màximes i mínimes que representen el cercle. Així, no s'haurà de processar tota la base de dades per trobar només la petita part que correspon al cercle de filtratge. L'eficiència també dependrà de la grandària del cercle, com és gran menys eficient. En el nostre cas, però, tindrem un radi màxim de cerca de 500 Km.

Les latituds màximes i mínimes del primer tall es poden obtenir sumant i restant respectivament de la latitud central l'angle que formen  $d$  (la distància màxima) i  $R$  (el radi de la terra). Com que la diferència entre  $d$  i  $R$  és molt gran, ens servirà:

$$\varphi_{max,min} = \varphi \pm \text{asin}\left(\frac{d}{R}\right) \approx \varphi \pm \frac{d}{R}$$

Calcular les longituds màximes i mínimes es fa de forma similar (en aquest cas es té en compte que l'angle de longitud disminueix en incrementar la latitud):

$$\lambda_{max,min} = \lambda \pm \frac{\text{asin}\left(\frac{d}{R}\right)}{\cos \varphi}$$

La implementació en llenguatge SQL incloent aquest primer tall seria:

```
Select Id, Lat, Lng,  
        acos(sin( $\varphi$ )*sin(radians(Lat)) +  
cos( $\varphi$ )*cos(radians(Lat))*cos(radians(Lng)- $\lambda$ )) * R As D  
From taula  
Where Lat Between  $\varphi_{\min}$  And  $\varphi_{\max}$   
        And Lng Between  $\lambda_{\min}$  And  $\lambda_{\max}$   
        And acos(sin( $\varphi$ )*sin(radians(Lat)) + cos( $\varphi$ )*cos(radians(Lat))*cos(radians(Lng)-  
 $\lambda$ )) * R < d  
Order By D
```

On  $\varphi_{\min}$ ,  $\varphi_{\max}$ ,  $\lambda_{\min}$ ,  $\lambda_{\max}$  sera valors que es passaran prèviament calculats en el moment de fer la crida SQL.

La implementació final d'aquesta algoritme es troba a l'arxiu `goteoapi/location/models.py`. En comptes de tractar directament amb el llenguatge SQL, es fan servir les eines de la llibreria *SQLAlchemy*.



## 5. Documentació de la API

Si el propòsit d'una API és facilitar la creació d'eines externes, serà imprescindible la creació d'una documentació adequada que serveixi de guia de referència per a desenvolupadors.

La documentació que ens proposem realitzar haurà de complir els següents requisits:

- Ser el més universal possible, usará l'idioma de facto d'Internet: l'anglès.
- Mantenir una estructura lògica que es pugui processar per programari extern, separant el disseny del contingut.
- El **contingut haurà d'estar inclòs en el mateix codi de l'API** (però no el disseny).
- El format de la documentació integrada haurà de ser permetre la possibilitat de utilitzar eines de generació automàtica de documentació en diferents formats (pensats per la consulta per part de qui en vulgui fer ús).

Amb aquests requisits es facilita la realització d'una documentació estable i actualitzada en el temps a mesura que l'API avança en el seu desenvolupament. Mitjançant la separació entre contingut i disseny, també serà molt més portable a diferents formats de presentació. En el nostre cas, també programarem un format de sortida inicial en HTML simple, a punt per ser consultat<sup>1</sup> en línia o fora de línia.

En aquest capítol es detallaran el format de documentació triada, com s'hi accedirà i com s'integra en el codi de l'aplicació. Com que la intenció és utilitzar formats estàndards i acceptats àmpliament per la indústria actual, l'opció triada ha estat l'especificació Swagger, un format de documentació en format JSON, automatitzada i que compleix amb els requisits imposats. Es donarà una visió general de com és aquesta especificació i quina estructura segueix i també quins components de software s'han utilitzat per integrar-ho en el codi.

### 5.1 Swagger - Open Api Initiative

Existeix un projecte, l'*Open Api Initiative*[45], esponsoritzada per la *Linux Foundation*[46] juntament amb una bona representació d'empreses que promou precisament la creació d'un estàndard de documentació per a APIs de tipus *RESTful*.

L'*Open Api Initiative* ha adoptat i promou l'anomenat *OpenAPI Specification* (antigament *Swagger Specification*). Una especificació que defineix un conjunt de regles vàlides tant per maquinari com humans amb l'objectiu de descobrir i entendre

<sup>1</sup> Disponible a l'apèndix de documentació (arxiu ZIP)

les capacitats d'un servei sense necessitat d'accedir al codi font, llegir documentació a part o estudiar el flux de dades.

Alhora, existeixen multitud d'eines que permeten processar la documentació generada i d'aquesta manera obtenir-ne distintes representacions. En el nostre cas generarem una representació en HTML com a referència pràctica per a programadors. També aprofitarem software ja preparat per llegir una especificació Swagger capaç de generar to un entorn de test on provar les diferents funcionalitats de l'API de manera molt senzilla.

## 5.2 Estructura bàsica

A grans trets, l'especificació *Swagger* (versió 2.0) es compon bàsicament d'un o més arxius en format JSON on es defineixen diversos blocs corresponents a diferents aspectes dels recursos REST. El fet que l'arxiu de documentació sigui en format JSON no implica que l'entrada/sortida de l'API també ho hagi de ser.

Els diversos blocs s'anomenen objectes, el principal dels quals és l'objecte arrel "*Swagger*" dins del qual es declaren la resta. Els atributs més importants d'aquest objecte són les definicions (*definitions*) i les rutes (*paths*), a part d'altres més descriptius (informació general, versió del protocol) o opcionals (per exemple, el tipus de seguretat implementada).

Cadascun d'aquests atributs pot contenir objectes o referències a altres objectes al seu torn fins a arribar als tipus primitius (enters, cadenes de caràcters, etc.) d'entrada o sortida.

Exemple d'especificació *Swagger*:

```
{
  "definitions": {
    "api_auth_token_get_Token": {
      "properties": {
        "access_token": {
          "description": "Acces token that can be used to authenticate ... ",
          "type": "string"
        },
        "expires_in": {
          "description": "Number of seconds wich the access_token ... ",
          "type": "string"
        },
        "token_type": {
          "description": "Currently, only `bearer`",
          "type": "string"
        }
      }
    }
  },
  "info": {
    "description": "Goteo.org API",
```

```

    "termsOfService": "Terms of service",
    "title": "Goteo.org API",
    "version": "1.1"
  },
  "paths": {
    "/calls/": {
      "get": {
        "description": "This resource returns call information",
        "parameters": [
          {
            "collectionFormat": "multi",
            "description": "Filter by individual node(s). Multiple ... ",
            "in": "query",
            "name": "node",
            "type": "string"
          },
          {
            "collectionFormat": "multi",
            "description": "Filter by individual project(s). ... ",
            "in": "query",
            "name": "project",
            "type": "string"
          }
        ],
        { ... }
      },
      "responses": {
        "200": {
          "description": "List of available calls",
          "schema": {
            "$ref": "#/definitions/api_calls_calls_get_ResponseCall"
          }
        },
        "400": {
          "description": "Invalid parameters format"
        },
        "401": {
          "description": "Resource requires authorization"
        }
      }
    },
    "summary": "Call API"
  }
},
}
"swagger": "2.0"
}

```

## 5.2.1 Atributs principals

Atribut Swagger	Descripció
<code>definitions</code>	Un objecte que contindrà totes les definicions de tipus de dades utilitzades en les operacions de l'API. Cada objecte dels aquí definits pot ser referenciat en altres parts de l'especificació.
<code>paths</code>	Camp obligatori. Cadascuna de les rutes de l'API REST està definida aquí juntament amb els mètodes i paràmetres HTTP acceptats (GET, POST, etc.).
<code>info</code>	Camp obligatori. Informació genèrica descriptiva de l'API com el nom, versió, etc.
<code>swagger</code>	Camp obligatori. Versió de l'especificació Swagger que s'està usant (2.0 en el nostre cas).

## 5.3 Aplicació

Com que aquesta especificació compleix amb els requisits requerits per a la nostra API, l'aplicarem en el codi font mitjançant l'ús d'extensions per al framework Flask que en permeten una fàcil integració i generació de l'arxiu d'especificació final.

Un altre aspecte on farem ús de la pròpia documentació estructurada és en la construcció de tests per l'API. Diversos tests verificaran que tots els camps i valors especificats es corresponen realment amb la sortida de cada recurs equivalent.

### 5.3.1 Integració en el codi

Existeix una extensió per al framework Flask (Flasgger<sup>[47]</sup><sup>2</sup>) que permet generar la documentació Swagger a partir d'arxius separats per a cada recurs en format YAML. El format YAML comparteix algunes característiques amb el XML o el JSON però amb la sintaxi simplificada i per tant més pràctic de mantenir.

L'extensió Flasgger proveeix un decorador Python que permet especificar quin arxiu conté la documentació per a cada recurs independentment. L'arxiu JSON final es genera automàticament.

Exemple de codi amb el decorador `@swag_from` indicant on trobar i processar la part de l'especificació Swagger per al recurs `/`:

<sup>2</sup> Durant la programació d'aquest projecte s'ha contribuït a la millora d'aquesta extensió afegint-hi algunes parts de l'especificació Swagger que faltaven:

<https://github.com/rochacbruno/flasgger/pull/25>

```

from flask import Flask
from flask_restful import Resource, Api
from flasgger.utils import swag_from

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    @swag_from('swagger_specs.yml')
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, '/')

```

L'arxiu `swagger_specs.yml` al seu torn podria ser quelcom similar a:

```

Hello API
---
definitions:
  - schema:
      id: Hello
      properties:
        hello:
          type: string
          description: 'Hello attribute, returns "world"'
responses:
  200:
    description: Returns hello:world object
    schema:
      $ref: '#/definitions/api_hello_get_Hello

```

El resultat final en format JSON es pot accedir com un recurs addicional en la ruta generada automàticament `/spec`, per exemple usant la comanda `curl`:

```

$ curl http://127.0.0.1:5000/spec
{
  "basePath": "/v1",
  "definitions": {
    "api_auth_token_get_Token": {
      "properties": {
...
$

```

### 5.3.2 Integració en els tests

La decisió de cenyir-se a un format estricte com JSON (o YAML) permet integrar de manera senzilla la validació entre la coherència de la documentació i el resultat proveït per l'API. Per exemple, amb l'escriptura de tests que comprovin que tots els camps especificats són presents en cadascun dels recursos ens assegurem que en cada modificació del codi estem també obligats a actualitzar-ne la documentació.

Un exemple de test que validi que tots els camps llegits es tan senzilla com (arxiu `goteoapi/tests/testcalls.py:118`):

```
def test_call():
    rv = test_app.get('/calls/test-call')
    resp = get_json(rv)
    ...
    # Swagger test
    fields = get_swagger(DIR + 'swagger_specs/call_item.yml', 'CallFull')
    eq_(set(resp.keys()), set(fields.keys()))
```

on la funció `get_swagger` (definida a `goteoapi/tests/__init__.py`) converteix una definició en format YAML a un objecte Python de tipus diccionari (amb claus i valors). La sentència `eq_` valida que la resposta obtinguda d'una crida al recurs `/calls/` sigui igual a l'especificació de la documentació.

### 5.3.3 Auto generació de la documentació

Un altre aspecte interessant d'aquesta documentació fortament estructurada és la possibilitat d'incorporar eines que generin una aplicació que consumeixi dades de l'API de manera automatitzada. Per una banda es pot programar un generador de documentació més "tradicional", el resultat del qual seran documents maquetats per la seva impressió o consulta en línia. De l'altra, es poden generar eines que serveixin de test a qualsevol programador que vulgui provar l'API.

El mateix projecte Swagger proporciona una eina ja preparada per aquesta tasca, *Swagger-UI*[48] (Fig. 16). Programada en HTML i Javascript, és bàsicament una pàgina web que llegeix l'especificació de la documentació JSON i en genera un seguit de formularis que permeten fer crides amb diferents paràmetres per tal de provar els diferents recursos que l'API ofereix.

La documentació final serà doncs un seguit de documents en HTML (pensats per estar disponibles a la web) generats a partir de l'especificació Swagger juntament amb l'aplicació Swagger-UI.

El resultat final de la documentació generada (Fig. 17) s'inclou en l'apèndix de documentació. També es pot consultar a l'adreça <http://developers.goteo.org/>.

Tanmateix, tant el codi font com la resta d'arxius que componen l'aplicació creada per generar aquesta documentació "oficial" són part d'un projecte a part i no s'inclouen en aquesta memòria. De fet, un dels objectius del projecte és, precisament, tenir una clara separació entre contingut i disseny de la documentació. El contingut s'inclou en l'API però no el disseny (que és bàsicament el que realitza l'aplicació de formatació).

De totes maneres, per a qui pugui interessar, el codi font de l'aplicació de formatació està programat en llenguatge Ruby utilitzant l'aplicació Jekyll, i disponible lliurement a la plataforma Github[49].

The screenshot shows the Swagger-UI interface for the Goteo.org API. At the top, there is a green header with the Swagger logo, the URL `https://api.goteo.org/v1/spec`, and buttons for 'Authorize' and 'Explore'. Below the header, the title 'Goteo.org Api' is displayed, followed by 'Goteo.org Api' and navigation links for 'auth', 'Show/Hide', 'List Operations', and 'Expand Operations'.


The 'auth' section is expanded, showing the 'login' endpoint (GET /login). It includes 'Implementation Notes' stating that the resource returns login information and a link to the developer documentation. Below this, the 'Response Class (Status 200)' is shown as 'Bearer token data ready for an OAuth Implicit authorization'. A 'Model' tab is selected, displaying an example JSON response:

```
{
  "access_token": "string",
  "expires_in": "string",
  "token_type": "string"
}
```

The 'Response Content Type' is set to 'application/json'. Below this, the 'Response Messages' section is visible, showing a table with columns for HTTP Status Code, Reason, Response Model, and Headers. A message with status code 401 and reason 'Resource requires authorization' is listed. A 'Try it out!' button and a 'Hide Response' link are also present.

At the bottom of the interface, a list of other API endpoints is shown: 'call', 'call\_projects', 'calls', 'categories', 'digests', and 'invest', each with its own 'Show/Hide', 'List Operations', and 'Expand Operations' links.

Figura 16: Swagger-UI, aplicació de test d'APIs tipus REST



**API DOCS**

Introduction

Responses

Security

Filters

Live Console

**ENDPOINTS**

**Projects**

Users

Categories

Licenses

Invests

Matchfunding calls

Reports

Digests

## Projects endpoint

This section gives you information about projects in [Goteo.org](https://goteo.org)

---

### /projects/

Obtains a list of projects.

```
curl -i --basic --user "user:key" https://api.goteo.org/v1/projects/
```

**Filters available:**

The **standard set of filters** applies to this endpoint with these particulars:

Attribute	Type	Description
<b>node</b>	string	Filter by individual node(s). Multiple nodes can be specified. Restricts the list to the projects originally created in that node(s)
<b>project</b>	string	Filter by individual project(s). Multiple projects can be specified
<b>lang</b>	string	Get results by specified lang. Multiple langs can be specified
<b>from_date</b>	string	Filter from date. Ex. "2013-01-01". Restricts the list to the projects created in that range
<b>to_date</b>	string	Filter until date.. Ex. "2014-01-01". Restricts the list to the projects created in that range
<b>category</b>	integer	Filter by project category. Multiple categories can be specified. Restricts the list to the projects that have interests in that category(ies)
<b>location</b>	number	Filter by project location (Latitude,longitude,Radius in Km). Restricts the list to the projects used in projects geolocated in that area
<b>page</b>	integer	Page number (starting at 1) if the result can be paginated
<b>limit</b>	integer	Page limit (maximum 50 results, defaults to 10) if the result can be paginated

**Response values:**

**ResponseProject**

Attribute	Type	Description
<b>items</b>	array	... <a href="#">Project</a>
<b>meta</b>		... <a href="#">MetaProject</a>

**Project**

Attribute	Type	Description
<b>amount</b>	number	Currently achieved amount for the project
<b>date-created</b>	string	Date when the project was created RFC822 format

Figura 17: Aspecte de la documentació de l'API un cop formatada

## 6. Desplegament

La fase de desplegament es produeix un cop acabada la programació de la primera versió de l'API, es tracta de posar el software en producció en un servidor real i preparat per acceptar i respondre peticions.

En els exemples que s'han mostrat fins ara s'ha utilitzat un servidor web intern del llenguatge Python pensat per a la fase del desenvolupament. No obstant això, no és un servidor pensat per producció doncs li falten moltes de les característiques desitjables. Entre elles la capacitat d'acceptar múltiples peticions concurrents o generar registres d'accés per a cada visitant.

Hi ha diversos servidors web disponibles, en el nostre cas s'ha triat el servidor Nginx per la seva lleugeresa i per compatibilitat amb altres webs de la *Fundación Goteo*. És un servidor que proporciona molt bona capacitat de resposta però, que per si sol, no és capaç d'oferir una interpretació a un llenguatge dinàmic com és Python. En el seu lloc es delega el procés a una aplicació secundària que es comunica mitjançant una interfície definida (*Fig. 18*). En el cas de Python i Nginx farem servir la més popular, uWSGI[50] (*Unified Web Server Gateway Interface*).

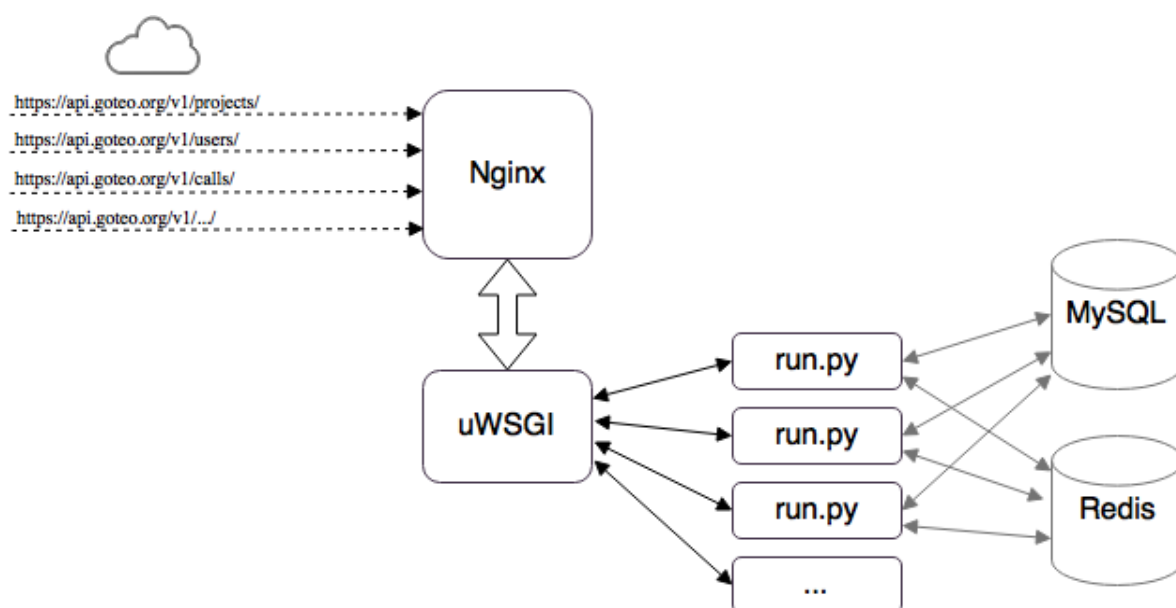


Figura 18: Esquema de la configuració en producció de l'API

L'aplicació intermèdia uWSGI s'encarrega de gestionar el nombre d'instàncies del codi Python de l'API que corren en paral·lel per tal poder respondre a més d'una petició alhora. També gestiona altres aspectes com la reinicialització en cas de caiguda del servei o el prefix que s'ha d'incorporar als URLs de l'API (les rutes de l'API en producció sempre contenen el prefix `/v1/`, per diferenciar-ho de possibles versions futures).

Finalment, cal associar un nom de domini a Internet que apunti al servidor així com els protocols implementats. El nom de domini final és `api.goteo.org` i els protocols implementats son `http://` i `https://`.

Així, algunes de les rutes de l'API quedarien amb l'URL completa:

<https://api.goteo.org/v1/projects/>  
<https://api.goteo.org/v1/users/>  
<https://api.goteo.org/v1/projects/la-trapa/donors/>  
etc,

Aquest sistema permet escalar el sistema quan el trànsit és alt, ja sigui en "vertical" (canviant el servidor per un altre de més potent) o en horitzontal (afegint més servidors configurats exactament igual en paral·lel per distribuir la càrrega).

Pel que fa a la documentació, d'una banda tindrem el recurs de l'especificació Swagger en format JSON, en l'adreça:

<https://api.goteo.org/v1/spec>

De l'altra, la documentació preparada en una presentació HTML en línia:

<https://developers.goteo.org/doc/>

# 7. Exemples i proves de funcionament

En aquest últim capítol es farà un recull d'eines i aplicacions que ja fan ús de l'API en la seva versió en producció.

## 7.1 Goteo Stats

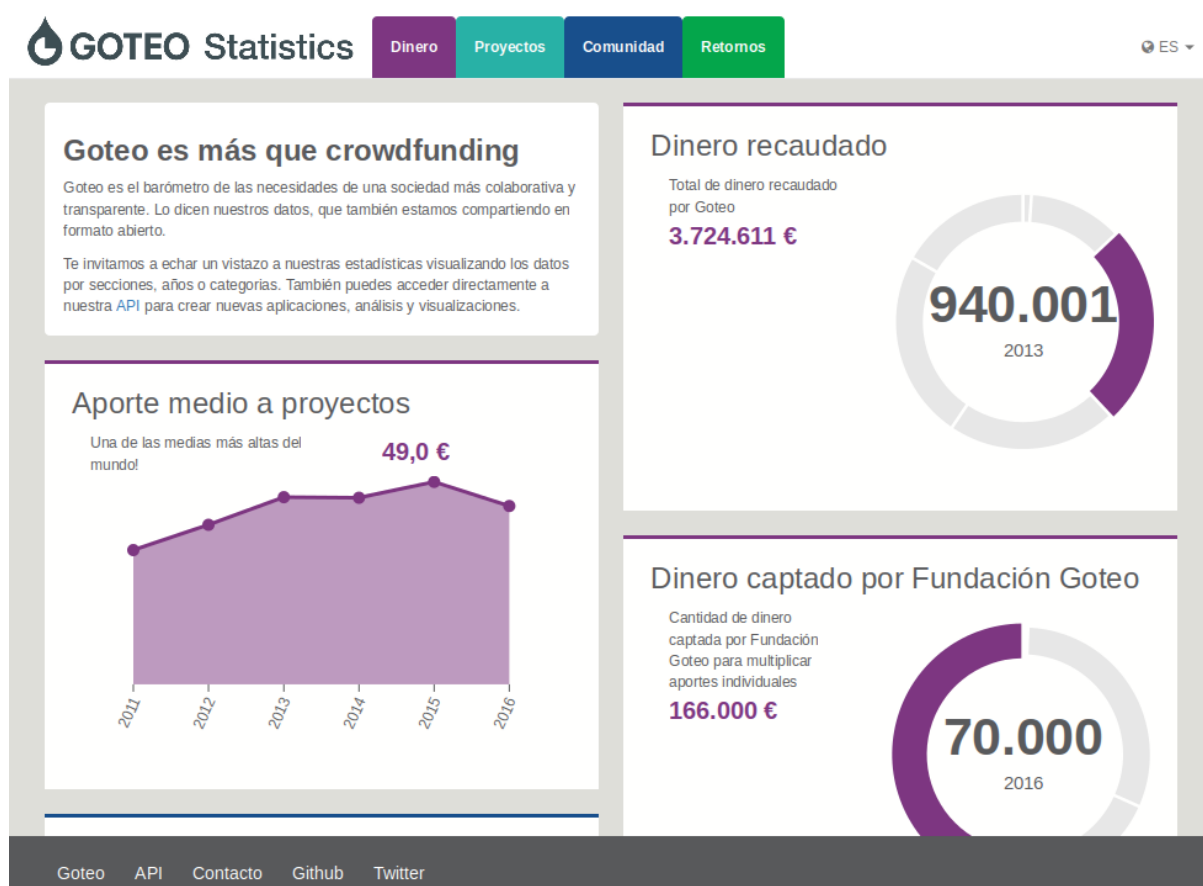


Figura 19: Aplicació Goteo Stats

**Tecnologies:** HTML, Javascript (Framework Angular)

**URL:** <https://stats.goteo.org/>

**Desenvolupadors:** Outliers Collective (<http://outliers.es/>)

Es tracta d'una aplicació en línia que permet visualitzar multitud d'estadístiques entorn de la plataforma Goteo. Organitzada en seccions o àmbits d'interès, anys i categories (Fig. 19). Tot tipus d'informació rellevant pot ser consultada aquí de manera gràfica. Per exemple, els 10 projectes més exitosos de la plataforma, els 10

projectes amb les donacions més altes, el nombre de projectes publicats per any, mitjanes de donacions, etc.

Està adaptada als navegadors de dispositius mòbils i fa ús especialment dels recursos proporcionats pel mòdul `goteapi_reports`.

## 7.2 Mapes dinàmics



Figura 20: Exemple de geolocalització usant Google Maps

**Tecnologies:** HTML, PHP, Javascript, Google Maps

**URLs:**

- <https://experiments.goteo.org/madrid-projects/>
- <https://experiments.goteo.org/barcelona-projects/>

**Desenvolupador:** Javi Carrillo (Fundación Goteo)

Un parell d'aplicacions en línia per demostrar la capacitat de geolocalització de l'API. Enfocada en els 2 nuclis urbans més grans d'Espanya (Barcelona, Fig. 20 i Madrid), mostra els projectes localitzats en un mapa que s'han finançat a Goteo al voltant d'aquestes ciutats.

De l'API fa ús del recurs `https://api.goteo.org/v1/projects/` amb el filtre de localització `location`.

## 7.3 Demo ECF Labs

### Goteo API usage in Angular JS


This example was developed during the [Goteathon event](#) organised by [Goteo.org](#) in [MediaLab Prado](#) in Madrid.

One of the main topics was Goteo.org API so two of us being among attendees wanted to contribute in practice, to test and implement API within the specific framework that we are both in high favour as a developers - [AngularJS](#). We thought it may be useful as an addition to existing [API docs](#) and usage examples.


Our example and contribution you can see on the right side, it also shows a simple tab-like navigation in Angular JS. Content is loaded by doing a request to Goteo API. All data rendered in tab panels is a response from API url that is parsed and prepared for tabs

Note that explanations are given within the code itself, as a code docu.


TOP 10 COLLABORATIONS
TOP 10 DONATIONS
FAVORITE R




¡Rebelaos! Publicación por la autogestión




FoldaRap, Peer-to-Peer Edition




#CrowdfundPaRato



Cerca de tu casa



Smart Citizen - Sensores ciudadanos



BHOREAL

Code Example (check it on [GitHub](#))

```

1 'use strict';
2
3 // Example Angular JS usage
4 angular.module('goteo', ['angular-loading-bar', 'ngAnimate'])
5
6 // Common API data
7 .value('GoteoApiKeys', {
8   user: '[YOUR_GOTEO_USERNAME]',
9   key: '[YOUR_GOTEO_KEY]',
10  // For this example we are choosing "Reports" section
11  baseUrl: 'https://api.goteo.org/v1/reports/summary/',
12  cache: true
13 })
14
15 // Define Angular service to send data request to Goteo API

```

Figura 21: Demostració de com programar l'accés a l'API usant el framework AngularJS

**Tecnologies:** HTML, Javascript (Framework Angular)

**URLs:**

- <http://ecflabs.org/lab/innovacion/goteo-api-usage-angularjs-demo-ecf-labs-team>
- <https://experiments.goteo.org/GoteoApiAngularJS/angular.html>

**Desenvolupador:** Gianfranco Pooli (ECF Labs, <http://ecflabs.org>)

És una simple demostració de com desenvolupar una aplicació (*Fig. 21*) usant el *framework* Javascript Angular i l'API de Goteo. Vas ser programat durant una trobada que es va fer a Madrid patrocinada pel Medialab Prado[51] com a presentació de l'API de Goteo.

## 7.4 Matchfunding visualizations

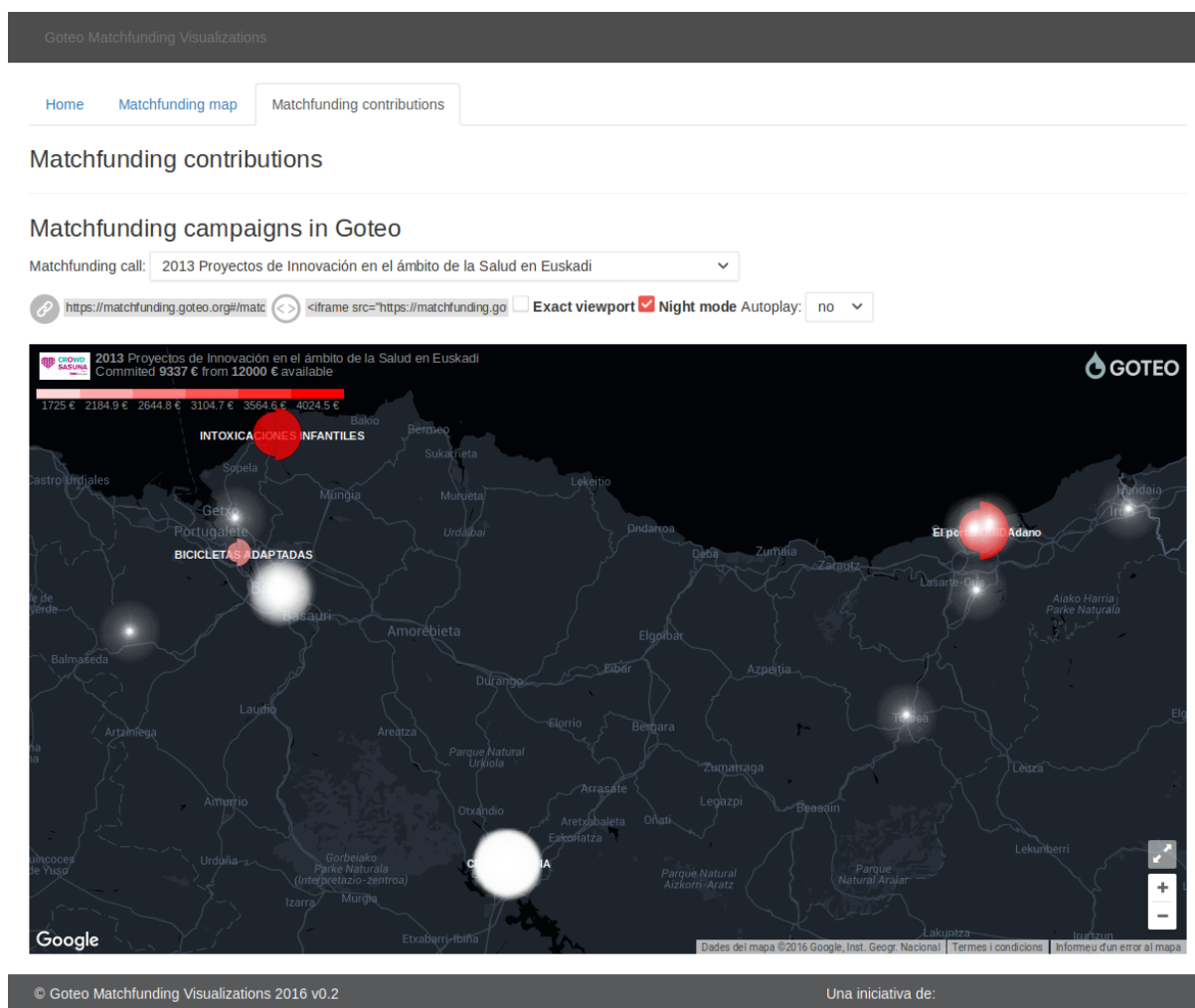


Figura 22: Aplicació animada de les aportacions Capital Regadiu

**Tecnologies:** HTML, Javascript (Framework Angular, Framework D3), Google Maps.

**URL:** <https://matchfunding.goteo.org/>

**Desenvolupador:** Ivan Vergés (Fundación Goteo)

Es tracta d'una aplicació pensada per compartir representacions animades de les aportacions d'usuaris relacionades amb convocatòries de capital regadiu. Es poden veure mapes amb totes les convocatòries que s'han promogut a la plataforma Goteo, les quantitats monetàries i el detall dels projectes i localitzacions de les aportacions individuals (Fig. 22).

## 7.5 Telegram @GoteoBot



Figura 23: Bot experimental per el programa de missatgeria instantània Telegram

**Tecnologies:** Python 3, Telegram API.

**URLs:**

- <https://web.telegram.org/#/im?p=@GoteoBot>
- <https://github.com/microstudi/goteobot>

**Desenvolupador:** Ivan Vergés

Un senzill bot per l'aplicació de missatgeria instantània *Telegram* (similar al *Whatsapp*). El bot és un agent automàtic que envia missatges (Fig. 23) a l'usuari subscrit cada vegada que hi ha una aportació en un projecte. El projecte està en fase experimental.



# Conclusions

En aquesta memòria ens hem proposat descriure el que ha suposat la creació d'una aplicació que actuarà com a una interfície d'accés (anomenada API) capaç d'exposar les dades més importants de la base de dades de la web de Goteo de manera pública.

L'objectiu primordial era que altres clients (que no fossin la web original) fossin capaços d'utilitzar i manipular les dades ofertes en maneres originals o diferents, de manera que aportin un valor afegit a la plataforma. Així, s'han pogut desenvolupar de manera exitosa aplicacions que fan ús d'aquestes dades i les presenten a l'usuari en formes variades i útils.

Per fer-ho, s'han estudiat i descrit, tant els formats i protocols necessaris per a la distribució de les dades de forma estructurada com el mateix codi de l'aplicació. En ambdós casos un dels objectius era aconseguir la màxima universalitat i potencialitat en la seva difusió. En el cas de les dades, mitjançant l'ús de formats i estàndards ben establerts actualment en l'àmbit d'Internet. En el cas de la mateixa aplicació, intentant que aquesta segueixi uns criteris d'eficiència, capacitat de test i extensibilitat mitjançant l'ús extensiu d'eines existents de codi lliure a més de publicar-se ella mateixa com a codi obert sota la llicència *Pública General Affero de GNU[52]*, validada per la *Free Software Foundation, Inc[53]*.

A més, per potenciar la difusió de l'API, també s'ha definit i creat una documentació sistemàtica i global seguint els mateixos paràmetres d'universalitat de manera que és possible per a qualsevol que ja utilitzi eines similars pugui crear-ne una que sigui capaç de consumir les dades ofertes de manera ràpida i natural.

Pel que fa a la realització tècnica del projecte, es pot concloure que s'ha dut a terme de manera exitosa doncs s'han complert tots els objectius inicials de:

- Crear un programari amb una base sòlida, modular i extensible que implementi una interfície d'accés a les dades internes de la plataforma Goteo en mode de només lectura.
- Que el programari estigui preparat per la seva evolució de manera que afegir futures funcionalitats no requereixi canviar substancialment el codi actual.
- Oferir un entorn amigable i documentat per a qualsevol que vulgui contribuir al codi, alhora, compartir el codi públicament amb una llicència de codi obert.
- Capacitat per testejar el codi i assegurar-ne la solidesa.
- Posada en producció de l'aplicació amb dades reals en servidors públics.
- Generació i presentació de la documentació pública.

No obstant això, cal dir que l'aplicació generada fins al moment és només una aplicació de lectura, és a dir una aplicació pensada per presentar les dades, no per introduir-ne de noves. De totes maneres, tant l'estructura del codi com el format de presentació de les dades permetran una evolució futura del codi cap a un sistema més complet on es puguin donar els dos tipus d'accés (lectura i escriptura).

De fet, l'objectiu final de la *Fundación Goteo* és, precisament, poder disposar d'un conjunt complet que permeti substituir l'actual web per un sistema més modular i més fàcil de mantenir on l'API formarà l'element central en el model de consum i generació de dades.

# Apèndix

Acompanyen a aquesta memòria dos arxius adjunts:

- El codi font (Python) de l'aplicació:

Arxiu: **codi\_api.zip**

- La documentació generada per la seva visualització off-line en HTML:

Arxiu: **documentacio.zip**

Adicionalment, el resultat del treball descrit en aquesta memòria es pot trobar publicat en el repositori *Github*[54] de la *Fundación Goteo* en les adreces:

- Codi font de l'aplicació API:

<https://github.com/GoteoFoundation/goteo-api>

- Documentació on-line de l'API:

<https://developers.goteo.org>

- Codi font de l'aplicació generadora de la documentació HTML de l'API a partir de l'original en format JSON inclosa en la API:

<https://github.com/GoteoFoundation/goteo-api-doc>



# Referències

- [1] «Goteo.org :: Crowdfunding the commons». [En línia]. Disponible a: <https://www.goteo.org/>. [Accedit: 11-jul-2016].
- [2] «W3C HTML». [En línia]. Disponible a: <https://www.w3.org/html/>. [Accedit: 11-jul-2016].
- [3] «Cascading Style Sheets». [En línia]. Disponible a: <https://www.w3.org/Style/CSS/>. [Accedit: 11-jul-2016].
- [4] «PHP: Hypertext Preprocessor». [En línia]. Disponible a: <http://php.net/>. [Accedit: 11-jul-2016].
- [5] «SQL - Wikipedia, the free encyclopedia». [En línia]. Disponible a: <https://en.wikipedia.org/wiki/SQL>. [Accedit: 11-jul-2016].
- [6] «MySQL». [En línia]. Disponible a: <http://www.mysql.com/>. [Accedit: 11-jul-2016].
- [9] «BOE.es - Documento BOE-A-2002-25180». [En línia]. Disponible a: <https://www.boe.es/buscar/doc.php?id=BOE-A-2002-25180>. [Accedit: 11-jul-2016].
- [7] «Crowdfunding - Wikipedia, the free encyclopedia». [En línia]. Disponible a: <https://en.wikipedia.org/wiki/Crowdfunding>. [Accedit: 11-jul-2016].
- [8] «Sobre Goteo :: Goteo.org». [En línia]. Disponible a: <https://www.goteo.org/about>. [Accedit: 11-jul-2016].
- [10] «Capital riego :: Goteo.org». [En línia]. Disponible a: <https://www.goteo.org/service/resources>. [Accedit: 11-jul-2016].
- [11] «RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1». [En línia]. Disponible a: <https://tools.ietf.org/html/rfc2616>. [Accedit: 11-jul-2016].
- [12] «RFC 1738 - A Gopher URL Format». [En línia]. Disponible a: <https://tools.ietf.org/html/rfc1738>. [Accedit: 11-jul-2016].
- [13] «Extensible Markup Language (XML)». [En línia]. Disponible a: <https://www.w3.org/XML/>. [Accedit: 11-jul-2016].
- [14] «The Official YAML Web Site». [En línia]. Disponible a: <http://yaml.org/>. [Accedit: 11-jul-2016].
- [15] «ECMA-404 The JSON Data Interchange Standard.» [En línia]. Disponible a: <http://www.json.org/>. [Accedit: 11-jul-2016].
- [16] «Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)». [En línia]. Disponible a: [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm). [Accedit: 11-jul-2016].
- [17] «RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax». [En línia]. Disponible a: <https://tools.ietf.org/html/rfc3986>. [Accedit: 11-jul-2016].

- [18] «World Wide Web - Wikipedia, the free encyclopedia». [En línia]. Disponible a: [https://en.wikipedia.org/wiki/World\\_Wide\\_Web](https://en.wikipedia.org/wiki/World_Wide_Web). [Accedit: 11-jul-2016].
- [19] «HATEOAS - Wikipedia, the free encyclopedia». [En línia]. Disponible a: <https://en.wikipedia.org/wiki/HATEOAS>. [Accedit: 11-jul-2016].
- [20] «JavaScript | MDN». [En línia]. Disponible a: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. [Accedit: 11-jul-2016].
- [21] «OSI model - Wikipedia, the free encyclopedia». [En línia]. Disponible a: [https://en.wikipedia.org/wiki/OSI\\_model](https://en.wikipedia.org/wiki/OSI_model). [Accedit: 11-jul-2016].
- [22] «Internet protocol suite - Wikipedia, the free encyclopedia». [En línia]. Disponible a: [https://en.wikipedia.org/wiki/Internet\\_protocol\\_suite](https://en.wikipedia.org/wiki/Internet_protocol_suite). [Accedit: 11-jul-2016].
- [23] «RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content». [En línia]. Disponible a: <https://tools.ietf.org/html/rfc7231>. [Accedit: 11-jul-2016].
- [24] «RFC 6585 - Additional HTTP Status Codes». [En línia]. Disponible a: <https://tools.ietf.org/html/rfc6585>. [Accedit: 11-jul-2016].
- [25] «RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication». [En línia]. Disponible a: <https://tools.ietf.org/html/rfc2617>. [Accedit: 11-jul-2016].
- [26] «RFC 6749 - The OAuth 2.0 Authorization Framework». [En línia]. Disponible a: <https://tools.ietf.org/html/rfc6749>. [Accedit: 11-jul-2016].
- [27] «Hypermedia - Wikipedia, the free encyclopedia». [En línia]. Disponible a: <https://en.wikipedia.org/wiki/Hypermedia>. [Accedit: 11-jul-2016].
- [28] «Ecma International». [En línia]. Disponible a: <http://www.ecma-international.org/>. [Accedit: 11-jul-2016].
- [29] «Python.org». [En línia]. Disponible a: <https://www.python.org/>. [Accedit: 11-jul-2016].
- [30] «Redis». [En línia]. Disponible a: <http://redis.io/>. [Accedit: 11-jul-2016].
- [31] «Virtualenv — virtualenv 15.0.2 documentation». [En línia]. Disponible a: <https://virtualenv.pypa.io/en/stable/>. [Accedit: 11-jul-2016].
- [32] «pip — pip 8.1.2 documentation». [En línia]. Disponible a: <https://pip.pypa.io/en/stable/>. [Accedit: 11-jul-2016].
- [33] «PyPI - the Python Package Index : Python Package Index». [En línia]. Disponible a: <https://pypi.python.org/pypi>. [Accedit: 11-jul-2016].
- [34] «Flask (A Python Microframework)». [En línia]. Disponible a: <http://flask.pocoo.org/>. [Accedit: 11-jul-2016].
- [35] «Werkzeug (The Python WSGI Utility Library)». [En línia]. Disponible a: <http://werkzeug.pocoo.org/>. [Accedit: 11-jul-2016].
- [36] «SQLAlchemy - The Database Toolkit for Python». [En línia]. Disponible a: <http://www.sqlalchemy.org/>. [Accedit: 11-jul-2016].
- [37] «Flask-RESTful — Flask-RESTful 0.2.1 documentation». [En línia]. Disponible a: <http://flask-restful-cn.readthedocs.io/en/0.3.4/>. [Accedit: 11-jul-2016].

- [38] «Nose 1.3.7 documentation». [En línia]. Disponible a: <http://nose.readthedocs.io/en/latest/>. [Accedit: 11-jul-2016].
- [39] «Ubuntu PC operating system | Ubuntu». [En línia]. Disponible a: <http://www.ubuntu.com/desktop>. [Accedit: 11-jul-2016].
- [40] «Vagrant by HashiCorp». [En línia]. Disponible a: <https://www.vagrantup.com/>. [Accedit: 11-jul-2016].
- [41] «Oracle VM VirtualBox». [En línia]. Disponible a: <https://www.virtualbox.org/>. [Accedit: 11-jul-2016].
- [42] «Flask-Cache 0.13 documentation». [En línia]. Disponible a: <https://pythonhosted.org/Flask-Cache/>. [Accedit: 11-jul-2016].
- [43] «Haversine formula - Wikipedia, the free encyclopedia». [En línia]. Disponible a: [https://en.wikipedia.org/wiki/Haversine\\_formula](https://en.wikipedia.org/wiki/Haversine_formula). [Accedit: 11-jul-2016].
- [44] «Spherical law of cosines. - Wikipedia, the free encyclopedia». [En línia]. Disponible a: [https://en.wikipedia.org/wiki/Spherical\\_law\\_of\\_cosines](https://en.wikipedia.org/wiki/Spherical_law_of_cosines). [Accedit: 11-jul-2016].
- [45] «Open API Initiative». [En línia]. Disponible a: <https://openapis.org/>. [Accedit: 11-jul-2016].
- [46] «The Linux Foundation». [En línia]. Disponible a: <http://www.linuxfoundation.org/>. [Accedit: 11-jul-2016].
- [47] «Creates Swagger 2.0 API documentation for all your Flask views extracting specs from docstrings or referenced files». [En línia]. Disponible a: <https://github.com/rochacbruno/flasgger>. [Accedit: 11-jul-2016].
- [48] «Swagger Editor – Swagger». [En línia]. Disponible a: <http://swagger.io/swagger-editor/>. [Accedit: 11-jul-2016].
- [49] «Documentation for the goteo API». [En línia]. Disponible a: <https://github.com/GoteoFoundation/goteo-api-doc>. [Accedit: 11-jul-2016].
- [50] «The uWSGI project — uWSGI 2.0 documentation». [En línia]. Disponible a: <https://uwsgi-docs.readthedocs.io/en/latest/>. [Accedit: 11-jul-2016].
- [51] «Goteathon: hackathon de Datos abiertos y crowdfunding organizado por la Fundación Goteo - Medialab-Prado Madrid». [En línia]. Disponible a: <http://medialab-prado.es/article/goteathon-hackathon-datos-abiertos-crowdfunding>. [Accedit: 11-jul-2016].
- [52] «GNU Affero General Public License - GNU Project - Free Software Foundation». [En línia]. Disponible a: <https://www.gnu.org/licenses/agpl-3.0.en.html>. [Accedit: 11-jul-2016].
- [53] «Free Software Foundation — working together for free software». [En línia]. Disponible a: <https://www.fsf.org/>. [Accedit: 11-jul-2016].
- [54] «Build software better, together», GitHub. [En línia]. Disponible a: <https://github.com>. [Accedit: 11-jul-2016].
- [55] «Fundación Goteo». [En línia]. Disponible a: <https://fundacion.goteo.org/>. [Accedit: 11-jul-2016].