UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

GENERACIÓ SEMI-AUTOMÀTICA D'APIS GRAPHQL

Autor:
Robert Almar Graupera

Director:
Carles Farré Tost - ESSI

Co-Director:
Jovan Varga - ESSI

Data de defensa:
24/01/2018

Grau en Enginyeria Informàtica
Especialitat d'Enginyeria del software

Resum

El projecte GENESIS s'ha proposat cinc reptes, un dels quals vol donar solució a la generació manual d'APIs perquè aquesta resulta una tasca molesta per moltes persones.

Per assolir aquest repte, els integrants del projecte GENESIS han desenvolupat un procés per etapes que consisteix en generar una API de la forma més automàtica possible a partir d'unes fonts de dades. L'objectiu del present Treball de Fi de Grau serà completar l'última etapa d'aquest procés.

Resumen

El proyecto GENESIS se ha propuesto cinco retos, uno de los cuales quiere dar solución a la generación manual de APIs porque esta resulta una tarea molesta para muchas personas.

Para alcanzar este reto, los integrantes del proyecto GENESIS han desarrollado un proceso por etapas que consiste en generar una API de la forma más automática posible a partir de unas fuentes de datos. El objetivo del presente Trabajo de Fin de Grado será completar la última etapa de este proceso.

Abstract

The GENESIS project has proposed five challenges, one of them wants to solve the manual generation of APIs because it is an annoying task for many people.

To achieve this challenge, the members of the GENESIS project have developed a process by stages that consists of generating an API in the most automatic way possible from data sources. The objective of the present Final Project will be to complete the last stage of this process.

Índex

Índex de figures

Índex de taules

1.Introducció i contextualització

1.1. Context

1.1.1. Introducció

Aquest Treball de Fi de Grau de modalitat A de l'especialitat Enginyeria del Software de la Facultat d'Informàtica de Barcelona ajudarà al projecte GENESIS[1], el qual hi participen els membres dels equips de recerca GESSI i DTIM, a assolir un dels seus reptes.

Ens centrarem en el repte que vol donar solució a la tasca molesta de fer APIs[1] manualment. Per assolir-lo, integrants dels departaments GESSI i DTIM han desenvolupat un procés per etapes [2] que consisteix en generar una API GraphQL de la forma més automàtica possible, aquest procés consta de 5 etapes i en l'última d'aquestes s'explica que a partir d'una ontologia[2] es pot generar una API GraphQL la qual diuen que està constituïda per un esquema GraphQL i un servidor GraphQL .

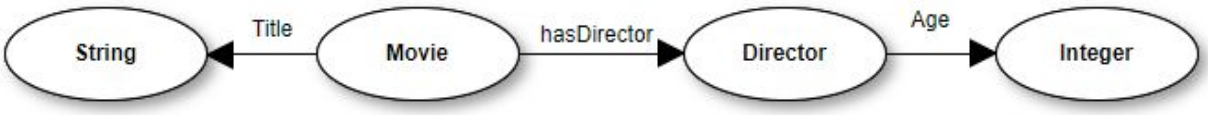


Figura 1. Primer exemple d'una ontologia

Les cinc etapes que té el procés són les següents:

1. Extracció d'ontologies a partir d'unes fonts de dades
2. Reestructuració de les ontologies

[1] Una API és un conjunt de funcions que permeten interactuar amb un sistema, per exemple un sistema de gestió de base de dades, i que podran ser utilitzades per un software.
[2] Una ontologia és una representació d'uns conceptes i de les relacions que hi ha entre aquests conceptes.

3. Construcció de l'ontologia global
4. Reestructuració de l'ontologia global
5. Generació de l'API GraphQL a partir de l'ontologia global.

El nostre TFG es centrarà en completar l'última etapa i per fer-ho es necessita una ontologia, la qual vindrà definida per l'etapa anterior. S'ha de tenir en compte que aquesta ontologia serà variable, és a dir, serà diferent per cada conjunt de fonts de dades que s'utilitzi en la primera etapa, per tant el nostre software ha d'estar preparat per generar una API GraphQL [3] a partir d'una ontologia que és variable.

1.1.2. Parts interessades

En aquest apartat es defineixen els actors principals (stakeholders), és a dir, aquelles persones o organitzacions afectades pel nostre projecte. En el nostre cas seran les següents:

- Director i co-director

  El director, Carles Farré, i el co-director, Jovan Varga, tenen la funció de supervisar el projecte i guiar la correcta evolució d'aquest. També es reuniran amb el desenvolupador, Robert Almar, per avaluar l'estat del projecte i si creuen convenient proposar fer canvis o millores en el projecte.

- Cap de projecte, dissenyador, desenvolupador i tester

  En ser un treball de fi de grau, aquestes parts interessades les farà la mateixa persona, que és l'autor del projecte. Haurà de documentar-lo i és l'encarregat de dissenyar, implementar el projecte i comprovar el correcte funcionament d'aquest.

  També organitzarà reunions amb el director i co-director per resoldre els dubtes que se li presentin durant el projecte.

- Usuaris finals

  Aquest grup està format per totes les persones que vulguin utilitzar el software desenvolupat durant el projecte.

1.2. Estat de l'art

L'objectiu principal d'aquest projecte és implementar un software que generi una API GraphQL a partir d'una ontologia, s'ha de dir que no s'ha trobat cap solució existent que implementi aquest software. A continuació es veuen alguns estudis que mostren com han generat la seva API i eines que han pogut generar APIs a partir d'alguna font de dades.

Hi ha un estudi [4] que van fer dos grups de recerca de Tailàndia els quals van agafar diverses dades amb diferents formats (CVS, XLS) extretes de la pàgina oficial del govern de Tailàndia les quals les convertien a format RDF/XML i les escrivien en un document, volien aconseguir que l'usuari pugues fes consultes sobre aquest document sense que l'usuari tingues cap coneixement sobre les tecnologies que s'han d'utilitzar per consultar aquest document. Per tant, s'utilitzava d'una interfície gràfica on l'usuari ficava les condicions que havia de tenir la consulta i en executar la consulta es mostrava per pantalla el resultat de la consulta.

Un altre estudi [5] detalla com s'ha creat l'aplicació EMF-REST [6] que a partir d'un model EMF genera una API RESTful. Pot retornar els resultats en format JSON o XML, proporciona el fet de validar el model a través d'OCL, ofereix un control d'accés a l'API mitjançant autentificació i seguretat amb encriptació, també genera una llibreria implementada amb JavaScript per gestionar l'API a la part del client.

S'ha trobat una eina de software lliure [7] que ens permet crear una API RESTFul automàticament de la teva base de dades, actualment només retorna la informació en

format JSON. Pots triar quines taules són publiques i quines no, això permetrà que hi hagi o no punts d'entrada de l'API sobre aquestes taules.

Una altra eina és Sails.js [8] que és un dels frameworks més populars per Node.js el qual utilitza el patró MVC. Ofereix la possibilitat de crear automàticament uns punts d'entrada a una API REST només executant una línia de comanda. Per exemple, amb aquesta comanda "run sails generate api dentista" es podrà buscar, ordenar, fer un filtre, crear, esborrar, modificar dentistes entre altres opcions.

JSON Server [9] és una eina de software lliure que ens permet fer una API REST de forma automàtica a partir d'un arxiu JSON. Aquesta eina llegeix l'arxiu JSON i identifica quins són els models que hi ha, si hi ha claus foranes entre si i ofereix la possibilitat de fer peticions GET, PUT, POST, DELETE, PATCH, filtrar i ficar un límit de files a retornar.

Per últim, DreamFactory [10] és una plataforma d'API REST la qual en crea una a partir de qualsevol base de dades SQL o NoSQL i documenta cada punt d'entrada de l'API. Una de les seves característiques més interessants és que permet amb una consulta consultar en més d'una base de dades.

En conclusió, s'ha vist que moltes eines han solucionat el problema de generar APIs REST a partir d'una font de dades però cap d'elles s'adapta al nostre problema el qual necessita que una eina que generi una API GraphQL a partir d'una ontologia. Per tant, s'haurà de crear una solució des de zero que pugui assolir aquest objectiu.

2. Abast del projecte

Tal com hem comentat en l'apartat anterior, el nostre TFG es centrarà en completar l'última etapa del procés per etapes que han desenvolupat els integrants del projecte GENESIS. Aquesta etapa s'haurà completat quan s'hagin assolit els objectius que té aquest TFG, que són els següents.

- Implementar un algoritme que permeti generar un esquema GraphQL a partir d'una ontologia que és variable.
- Implementar un algoritme que permeti generar un servidor GraphQL a partir d'una ontologia que és variable.
- Desenvolupar una aplicació web que permeti executar els algoritmes.
- Desenvolupar una aplicació web que permeti fer peticions a una API GraphQL generada pels algoritmes. També tindrà altres funcionalitats que es descriuran a l'apartat 6.

2.1. Possibles obstacles

- Temps limitat

  El treball de fi de grau té una data fixada per presentar-lo, per tant, els inconvenients que es puguin presentar durant el desenvolupament del projecte s'han de solucionar adequadament i el més ràpid possible.

- Desconeixement de la tecnologia

  L'evolució del projecte anirà molt lligat a l'aprenentatge autònom que s'adquireixi de GraphQL i a les llibreries de Java que s'utilitzin.

2.2. Metodologia i rigor

Les metodologies àgils són una opció a tenir en compte quan desenvolupem un projecte de curt termini, però moltes d'aquestes van orientades per fer treballs en grup. En aquest cas, s'ha triat SCRUM[11] com a metodologia a seguir però s'haurà

d'adaptar a les característiques d'un treball de fi de grau, ja que es farà de forma individual.

2.2.1. Eines de seguiment

Al llarg del projecte, el nostre software anirà evolucionant per tant utilitzar un control de versions i un repositori en línia, com poden ser Git i Github, serà el més adequat pel projecte. Git s'utilitzarà per guardar el codi en el nostre repositori en línia i descriure quin objectiu s'ha completat. Per altra banda, Github allotjarà el repositori Git i ens permetrà accedir al nostre codi des de qualsevol màquina.

2.2.2. Mètode de validació

Les reunions que es faran regularment amb el director o co-director del projecte permetran veure el funcionament del software desenvolupat i comentar què és el que s'ha fet des de l'última reunió, llavors el director o co-director donarà feedback per tal que el desenvolupador sàpiga si el projecte s'està desenvolupant de forma correcta i en cas que no, doncs modificar els errors que han vist.

També es ficarà a prova el software desenvolupat, al final de cada iteració de forma que es garantirà que el software vagi complint els requisits del projecte.

3.Planificació

3.1 Duració del projecte

El projecte té una duració estimada de 4 mesos, del 18 setembre fins al 22 de gener del 2018 com a data límit.

3.2 Fases del projecte

3.2.1 Planificació del projecte

Aquesta primera fase és una de les més importants del projecte, ja que és quan definim els aspectes bàsics d'aquest, com són el context, abast, objectius, planificació, gestió econòmica i la sostenibilitat.

Aquesta fase dura aproximadament un mes i mig, del 18 de setembre fins a la setmana del 30 d'octubre de 2017 i no té cap dependència.

3.2.2 Iteració 1

S'escriuen les històries d'usuari que implicarien a la generació de l'esquema GraphQL i servidor. Aquestes ens ajudaran a saber que és el que es generarà i què no. També es veu el disseny l'esquema GraphQL i el servidor GraphQL , és a dir, quina estructura haurien de tenir i es fa el disseny d'un metamodel de les ontologies[3] el qual es basaran els algoritmes amb l'objectiu de crear un esquema GraphQL i un servidor GraphQL per qualsevol ontologia donada que segueixi aquest metamodel.

Pel que fa a la implementació s'ha fet una primera versió dels algoritmes que hauran de generar l'esquema GraphQL i el servidor GraphQL.

[3] El metamodel de les ontologies especifica quins són els elements vàlids que poden tenir les ontologies i quina relació hi ha entre aquests elements.

Per últim, l'apartat de proves on s'haurà de veure que les generacions de l'esquema GraphQL i el servidor GraphQL siguin correctes i que en conjunt puguin fer una API GraphQL funcional. Aquestes proves estaran presents en les iteracions 1,2 i 3, estan més detallades a l'apartat 12 d'aquest document.

Aquesta iteració es comença quan s'ha acabat de fer la primera fase i té una durada de 3 setmanes aproximadament.

3.2.3 Iteració 2

Es torna a dissenyar el metamodel el qual tindrà nous elements i noves relacions que abans no tenia, i això provoca que les ontologies puguin utilitzar aquests nous elements i relacions. Aquests nous elements del metamodel de les ontologies s'hauran de tenir en compte a l'hora de generar el servidor GraphQL i l'esquema GraphQL, per tant es readapta el codi al nou disseny del metamodel.
També es fan proves per tal de veure que els resultats són els esperats.

Aquesta iteració comença quan acaba la primera iteració i té una durada de 2 setmanes aproximadament.

3.2.4 Iteració 3

Aquesta iteració és igual que la segona iteració.

Aquesta iteració comença quan acaba la segona iteració i té una durada de 2 setmanes aproximadament.

3.2.5 Iteració 4

En aquesta iteració es defineixen els requisits funcionals de les aplicacions web, aquestes s'implementen i també es fa el disseny de les pantalles i les navegacions

entre elles. A més a més es faran proves per tal d'assegurar el correcte funcionament, aquestes proves consistirien en reproduir els passos que faria l'usuari final per tal d'utilitzar les funcionalitats disponibles.

Aquesta iteració comença quan acaba la tercera iteració i té una durada de 2 setmanes aproximadament.

3.2.6 Documentació i preparació de la defensa

Un cop feta tota la part pràctica del treball de fi de grau s'haurà de redactar una memòria i preparar-se la defensa davant del tribunal.

3.3 Estimació d'hores per tasca

A continuació es mostrarà en una taula les hores dedicades a cada tasca que s'ha identificat i a quina iteració pertany.

Tasca	**Iteració**	**Temps dedicat (Hores)**
Planificació del projecte		75.00
Abast i contextualització	-	24.50
Planificació temporal	-	8.25
Gestió econòmica i sostenibilitat	-	9.25
Presentació preliminar	-	6.25
Document especialitats	-	8.50
Presentació i document final	-	18.25
Generació de l'esquema GraphQL		170.00
Anàlisi de requisits	1	10.00
Disseny esquema i metamodel	1	20.00
Implementació	1	50.00
Proves	1	10.00

Disseny metamodel	2	5.00
Implementació	2	25.00
Proves	2	10.00
Disseny metamodel	3	5.00
Implementació	3	25.00
Proves	3	10.00
Generació del servidor GraphQL		180.00
Anàlisi de requisits	1	10.00
Disseny servidor i metamodel	1	20.00
Implementació	1	35.00
Proves	1	10.00
Disseny metamodel	2	5.00
Implementació	2	35.00
Proves	2	10.00
Disseny metamodel	3	5.00
Implementació	3	40.00
Proves	3	10.00
Desenvolupament de les aplicacions web		80.00
Anàlisi de requisits	4	5
Disseny	4	10
Implementació	4	50
Proves	4	15
Documentació i presentació		80.00
Documentació final	-	55.00
Preparar la defensa	-	25.00

Total	**585.00**

Taula 1. Estimació en hores de les diferents tasques del projecte

3.4 Recursos

Per poder realitzar aquest projecte s'han utilitzat tres tipus de recursos: els materials, els de software i els personals.

3.4.1 Recursos humans

Una persona que dediqui 35 hores setmanals aproximadament al projecte el qual s'encarregarà de fer totes les tasques descrites anteriorment.

3.4.2 Recursos materials

- Un ordinador portàtil amb el qual s'escriurà el codi i es documentarà el projecte.
- Un lloc de treball perquè es pugui dur a terme el projecte amb els recursos indirectes que comporta tenir-lo, com pot ser l'electricitat.

3.4.3 Recursos de software

- Windows 10 Home, sistema operatiu de l'ordinador.
- Git i Github, control de versions i servidor on es guardarà el codi, respectivament.
- Google Docs, per realitzar la documentació del projecte.
- GanttProject, per fer els diagrames de Gantt

La resta de software utilitzat s'explicarà a l'apartat 10.

3.5 Valoració d'alternatives i pla d'acció

Durant el transcurs del projecte pot ser que s'endarrereixi la planificació a causa de les desviacions que poden sorgir. Per això s'han d'analitzar i pensar en una solució perquè aquestes ens afectin el menys possible.

Una de les més crítiques i importants és que alguna implementació ens requereixi més hores del normal, ja sigui pel desconeixement de les tecnologies o perquè el desenvolupador es troba davant d'un problema que no ha resolt abans. En aquest cas, s'hauria de dedicar més hores en estudiar la tecnologia o si el problema li és difícil de resoldre, reunir-se amb el tutor per trobar alternatives per solucionar-lo.

Una altra desviació seria que trobar-se amb errors d'instal·lació amb el software que s'utilitzarà en el projecte, això pot ser donat per falta de programari requerit o versió del sistema operatiu, això fa endarrerir tot el projecte, ja que és necessari tenir llest el software per començar el projecte. El que caldrà fer serà dedicar més hores en solucionar-ho o buscar una alternativa equivalent que no ens doni errors.

Donat que a cada tasca del projecte li hem assignat més hores del que pertoca les quals en principi no són necessàries, les podem fer servir per a les desviacions que hem valorat. Altrament, si la tasca s'acaba abans de la data fixada, es començarà amb la següent tasca i el temps sobrant s'invertirà amb les següents tasques. D'aquesta forma n'assegurem que el projecte no s'endarrereixi més enllà de la setmana del 22 de gener de 2018.

3.6 Canvis respecte a la planificació inicial

Hi ha hagut canvis respecte a la planificació inicial, les tasques de les generacions s'han pogut fer de forma paral•lela. També, a mesura que es desenvolupava el projecte s'anaven trobant elements que no s'havien representat en el disseny del metamodel de les ontologies, per tant s'ha anat canviant el disseny d'aquest al final de cada iteració per tal que aquest sigui més complet. Això ha provocat que es dediques més temps al disseny i que cada vegada s'hagués d'adaptar el codi al nou disseny. També s'ha de dir que cada cop hi havia un disseny més complex però al final s'ha anat simplificant bastant i ha quedat semblant a l'inicial.

Respecte a les proves, s'ha dedicat més temps, ja que s'ha cregut convenient provar el codi dels algoritmes de forma més sovint, ja que així s'eliminarien erros més fàcilment.

També, s'han desenvolupat les aplicacions web que al principi del projecte no s'havien tingut en compte però no ha suposat un gran inconvenient fer-les, ja que gràcies a fer les tasques que generen l'esquema GraphQL i servidor GraphQL de forma paral·lela hem tingut temps per desenvolupar-les.

Tots aquests canvis afectaran al pressupost del projecte.

3.7 Diagrames de Gantt

A l'annex A es trobaran els diagrames de Gantt respecte a la planificació inicial i la planificació final.

Les hores de disseny del metamodel de les ontologies estan incloses a les tasques "Generació de l'esquema GraphQL" i "Generació del servidor GraphQL", ja que el disseny del metamodel influeix a l'hora de fer les dues tasques.

4. Gestió econòmica

En aquest apartat es proporcionarà una estimació del cost del projecte a partir dels recursos que s'han esmentat anteriorment, com poden ser els de hardware, software i humans, els quals tenen unes despeses.

4.1 Pressupost de recursos humans

Aquest projecte només serà desenvolupat per una persona, per tant, exercirà els rols de cap de projecte, analista, dissenyador i tester. Veient les hores proposades per cada tasca en l'apartat 3.3 i els salaris mitjans de cada un dels rols podem calcular el pressupost.

A continuació es mostren els salaris que s'assignaran a cada rol:

- Cap de projecte: 40 €/h
- Analista: 35 €/h
- Arquitecte: 35 €/h
- Programador: 30 €/h
- Tester: 25 €/h

Tasca	Temps de dedicació en hores					Cost
	Cap de projecte	Analista	Arquitecte	Programador	Tester	
Planificació del projecte	**75**	**0**	**0**	**0**	**0**	**3.000 €**
• Abast i contextualització	24.50	0	0	0	0	980 €
• Planificació temporal	8.25	0	0	0	0	330 €

• Gestió econòmica i sostenibilitat	9.25	0	0	0	0	370 €
• Presentació preliminar	6.25	0	0	0	0	250 €
• Document especialitats	8.50	0	0	0	0	340 €
• Presentació i document final	18.25	0	0	0	0	730 €
Generació de l'esquema GraphQL	**0**	**10**	**30**	**100**	**30**	**5.150 €**
• Anàlisi de requisits	0	10	0	0	0	350 €
• Disseny	0	0	30	0	0	1.050 €
• Implementació	0	0	0	100	0	3.000 €
• Proves	0	0	0	0	30	750 €
Generació del servidor GraphQL	**0**	**10**	**30**	**110**	**30**	**5.450 €**
• Anàlisi de requisits	0	10	0	0	0	350 €
• Disseny	0	0	30	0	0	1.050 €
• Implementació	0	0	0	110	0	3.300 €
• Proves	0	0	0	0	30	750 €
Desenvolupament de les aplicacions web	**0**	**5**	**10**	**50**	**15**	**2.400 €**
• Anàlisi de requisits	0	5	0	0	0	175 €
• Disseny	0	0	10	0	0	350 €
• Implementació	0	0	0	50	0	1.500 €
• Proves	0	0	0	0	15	375 €
Documentació i presentació	**80**	**0**	**0**	**0**	**0**	**3.200 €**

• Documentació final	55	0	0	0	0	2.200 €
• Preparar la defensa	25	0	0	0	0	1.000 €
Total						**19.200 €**

Taula 2. Pressupost recursos humans

4.2 Pressupost de recursos de hardware

Per dur a terme el projecte seran necessaris uns elements de software, que en el nostre cas serà l'ordinador portàtil el qual se li assignarà una vida útil i un temps de 4 mesos d'amortització que és la durada del TFG.

Producte	Preu	Unitats	Vida útil (anys)	Amortització
HP Pavilion 15-cc510ns	899 €	1	5	59,11 €

Taula 3. Pressupost de recursos de hardware

4.3 Pressupost de recursos de software

L'ordinador portàtil ja inclou una distribució de Windows 10 Home per tant el seu preu i la seva amortització va inclosa en la taula de l'apartat anterior i la resta de software és gratuït per tant no hi haurà cap cost associat.

4.4 Altres despeses

A part de les despeses comentades anteriorment, n'hi ha d'altres que són indirectes les quals estaran presents durant el projecte, com pot ser la fibra òptica, electricitat, el transport per fer les reunions amb el director i co-director o el lloguer del lloc de treball.

Producte	Preu (€/mes)	Dedicació %	Cost
Fibra òptica	28,21	60	67,70 €

Electricitat	40	30	48 €
Transport (T-10 3 zones)	26,75	100	107 €
Lloc de treball	150	20	120 €
Total			**340,70 €**

Taula 4. Pressupost d'altres despeses

4.5 Contingències

Per cobrir el cost de les contingències es dedicarà el 15% del preu de les despeses directes i indirectes.

Producte	Percentatge	Preu	Cost
Recursos humans	15%	19.200 €	22.080 €
Recursos hardware	15%	59,11 €	67,97 €
Recursos software	15%	0 €	0 €
Altres despeses	15%	340,70 €	391,81 €
Total			**22.539,78 €**

Taula 5. Pressupost amb contingències

4.6 Imprevistos

En aquest apartat es veuran els diferents imprevistos que ens poden sorgir durant el projecte, en el nostre cas seran aquests dos:

- Es podria produir una avaria l'ordinador portàtil el qual s'hauria de reparar o substituir per un de nou per continuar el projecte sense problema. Degut el temps limitat que tenim la millor opció seria comprar-ne un de nou que suposaria una despesa de 900 euros. A aquest imprevist se li assigna un 5% de probabilitats que passi.

- Podria ser que en una tasca es dediqui més temps de l'esperat i hauríem de fer hores extres les quals es pagarien amb el salari corresponent depenent del rol que s'exercís per fer la tasca. Es suposarà que es fan 10 h més de cada rol, per tant s'haurà de pagar 400 + 350 + 350 + 300 + 250 = 1.650. Aquest imprevist se li assigna un 20% de probabilitats que passi.

Imprevist	Probabilitat	Preu
Avaria ordinador portàtil	5%	900 €
Hores extres	20%	1.650 €
Total		**2.550 €**

Taula 6. Despeses dels imprevistos

4.7 Pressupost final

Aquí es mostrarà les despeses que tindrà el projecte en una sola taula:

Tipus	Preu
Despeses amb contingències	22.539,78 €
Imprevistos	2.550 €
Total	25.089,78 €
IVA	21%
Total amb IVA	**30.358,63 €**

Taula 7. Pressupost final

4.8 Control de gestió

Els recursos de software, hardware van ser adquirits fa temps i no es pot fer un control sobre ells perquè no se sap quin consum real hem tingut sobre ells. Per tant l'únic que podem controlar són els recursos humans.

Al final de cada tasca es mirarà quantes hores s'han dedicat a fer-la i quantes n'hem estimat, l'ideal seria que aquesta diferència no fos molt gran. Llavors al final del projecte podrem saber si les hores realitzades corresponen a les totals del projecte.

Per calcular aquestes desviacions podem fer servir les següents fórmules:

- Desviació recursos humans = (cost estimat - cost real) * hores realitzades
- Desviació total en recursos = cost total estimat en recursos - cost total real en recursos
- Desviació completar tasca = (cost estimat - cost real) * hores realitzades
- Desviació total en completar tasques = cost total estimat en realitzar tasques - cost total real en realitzar tasques

5. Sostenibilitat i compromís social

5.1 Dimensió econòmica

Aquest projecte s'ha plantejat d'una forma bastant eficient pel que fa als recursos utilitzats, ja que només s'utilitzarà els recursos essencials per fer cada una de les tasques identificades. Per tant, el projecte no es podria realitzar utilitzant menys recursos. A més, s'ha procurat utilitzar abans software lliure que no un de pagament, així no s'augmenta el cost del projecte.

També dir que el projecte s'hagués pogut fer en menys temps si haguéssim tingut més recursos humans però com tot el projecte l'ha realitzat una persona sola, el temps que s'ha tardat en fer-lo és superior.

Aquest projecte no està pensat amb l'objectiu que surti al mercat, per tant no hi haurà riscos econòmics, ja que no existeix un desig lucratiu.

5.2 Dimensió social

El meu projecte anirà destinat principalment als integrants del projecte GENESIS, ja que necessiten el meu software per realitzar la última etapa del procés que han proposat per generar una API GraphQL.

A més, el meu software permet estalviar-se temps, ja que si no estigués implementat s'hauria de fer manualment la generació de l'API GraphQL a partir d'una ontologia.

A nivell personal, he après una sèrie de coneixements que tot enginyer informàtic hauria de tenir, com pot ser com gestionar un projecte i d'altres de més específics com pot ser la tecnologia GraphQL.

5.3 Dimensió ambiental

La realització d'aquest projecte no té un gran impacte ambiental més enllà del consum elèctric de l'ordinador portàtil. Aquest consum és inevitable perquè necessitem un ordinador portàtil per produir el codi i la documentació. Com que la documentació del projecte es fa en línia, no es consumirà cap paper ni cap altre material per fer-la.

Durant el transcurs del projecte s'ha fet una bona gestió de l'electricitat, com per exemple no deixant encès l'ordinador si no el fem servir, ja que per produir-la deixa petjada al medi ambient.

L'impacte ambiental del desenvolupament del projecte fins a l'estat actual, es pot quantificar mesurant el consum energètic dels recursos i es farà amb la següent formula:

15 W x 390 h = 5.850 Wh = 5,85 kW h

Segons l'informe de Megan Bray[12] un portàtil consumeix de mitjana 15 W quan està actiu i s'ha agafat les hores que s'han dedicat a la implementació, proves i redacció de la memòria final que ens dóna un total de 390 h.

5.4 Matriu de sostenibilitat

Aquí podem veure la matriu de sostenibilitat on els seus valor s'han posat a partir de les valoracions que hem fet anteriorment.

	PPP	Vida útil	Riscos
Ambiental	Consum de disseny	Petjada ecològica	Riscos ambientals
	8	16	-1
Econòmic	Factura	Pla de viabilitat	Riscos econòmics
	7	16	0
Social	Impacte personal	Impacte social	Riscos socials
	6	16	0
Rang de sostenibilitat	21	48	-1
	68		

Taula 8. Matriu de sostenibilitat

6. Anàlisi de requisits

En aquest apartat es definiran els requisits funcionals i no funcionals de les nostres aplicacions web, s'han obtingut a partir de les reunions que s'han anat fent amb el director i co-director.

Aquests requisits s'han especificat amb la plantilla Volere [13], en la qual podrem trobar el número de requisit, el tipus de requisit segons Volere, descripció d'aquest, justificació i el criteri d'acceptació. També es mostra el nivell de satisfacció del client si el requisit ha sigut implementat i el nivell d'insatisfacció si aquest no ho ha estat, el 5 serà la màxima satisfacció o insatisfacció i l'1 la mínima. Per últim es mostra la prioritat de cada requisit, on el número 1 serà la màxima prioritat i el 5 la mínima.

6.1 Requisits funcionals

S'ha de dir que en el nostre projecte es fan servir dues aplicacions web: l'aplicació web principal amb la qual es podrà fer les generacions i la segona aplicació web és creada per la principal i conté l'API GraphQL, per tant es diferenciaran els requisits funcionals entre aquestes.

Els criteris d'acceptació vinculats a GraphQL dels casos d'ús #1 i #2 corresponen a l'anàlisi de requisits i disseny de les tasques "Generació de l'esquema GraphQL" i "Generació del servidor GraphQL", ja que ens diuen què podran generar i quina estructura han de seguir els elements generats.

6.1.1 Aplicació web principal

 # Requisit: 1
Tipus de requisit: 9 (Requisit funcional)
Descripció: L'usuari vol generar un esquema GraphQL.
Justificació: Necessari per complir amb els objectius del projecte
Criteri d'acceptació:

- El sistema li mostra un formulari que l'usuari haurà d'omplir amb una connexió vàlida a una base de dades que contingui una ontologia.
- El sistema haurà de poder generar un esquema GraphQL que pugui tenir els elements que estan descrits de l'apartat 7.2.

Satisfacció pel client: 5
Insatisfacció pel client: 5
Prioritat: 1

 # Requisit: 2
Tipus de requisit: 9 (Requisit funcional)
Descripció: L'usuari vol generar un servidor GraphQL.
Justificació: Necessari per complir amb els objectius del projecte
Criteris d'acceptació:

- El sistema li mostra un formulari que l'usuari haurà d'omplir amb una connexió vàlida a una base de dades que contingui una ontologia.
- El sistema haurà de poder generar un servidor GraphQL que tingui les classes de l'apartat 7.3.

Satisfacció pel client: 5
Insatisfacció pel client: 5
Prioritat: 1

6.1.2 Aplicació web API GraphQL

 # Requisit: 3
Tipus de requisit: 9 (Requisit funcional)
Descripció: L'usuari vol descarregar-se l'aplicació web que conté l'API GraphQL.
Justificació: Es podrà fer portable l'aplicació web que conté l'API GraphQL i distribuir-la als usuaris que els hi interessi enviar peticions a l'API GraphQL generada però no tinguin l'ontologia a mà per generar aquesta API GraphQL.
Criteri d'acceptació:

- El sistema comprimeix l'aplicació web de l'API GraphQL i es descarrega a la part del client.

Satisfacció pel client: 3
Insatisfacció pel client: 3
Prioritat: 3

 # Requisit: 4
Tipus de requisit: 9 (Requisit funcional)
Descripció: L'usuari vol fer peticions a l'API GraphQL.
Justificació: Es podrà obtenir la informació desitjada fent peticions a l'API GraphQL.
Criteris d'acceptació:

- El sistema mostrarà a l'usuari una interfície gràfica amb la qual l'usuari podrà definir les seves consultes i enviar peticions a l'API GraphQL.
- El sistema retorna la resposta de la petició enviada per l'usuari.

Satisfacció pel client: 5
Insatisfacció pel client: 5
Prioritat: 1

 # Requisit: 5
Tipus de requisit: 9 (Requisit funcional)
Descripció: L'usuari vol editar els paràmetres de connexió a la base de dades.
Justificació: Es podrà modificar els paràmetres d'accés a una altra base de dades
Criteris d'acceptació:

- El sistema li mostra un formulari que l'usuari haurà d'omplir amb una connexió vàlida a una base de dades.
- El sistema mostra els paràmetres de connexió actual que hi ha amb la base de dades.
- El sistema emmagatzema els nous paràmetres

Satisfacció pel client: 4
Insatisfacció pel client: 3
Prioritat: 2

 # Requisit: 6
Tipus de requisit: 9 (Requisit funcional)
Descripció: L'usuari vol veure l'esquema GraphQL.
Justificació: Es podrà veure l'esquema GraphQL que va lligat a l'aplicació web que conté l'API GraphQL.
Criteri d'acceptació: El sistema mostrarà a l'usuari l'esquema GraphQL generat.
Satisfacció pel client: 3
Insatisfacció pel client: 3
Prioritat: 3

6.2 Requisits no funcionals

Els requisits no funcionals seran els mateixos per les dues aplicacions web.

 # Requisit: 7
Tipus de requisit: 10a (Aparença)
Descripció: El sistema ha de ser atractiu pels usuaris.
Justificació: L'aparença del nostre sistema haurà de ser atractiva perquè els usuaris utilitzin el sistema. És important ja que l'aparença és la part del nostre sistema que interacciona amb l'usuari.
Criteri d'acceptació: Un 75% dels usuaris asseguren que en la primera presa de contacte amb la interfície del sistema ha estat agradable i satisfactòria.
Satisfacció pel client: 3
Insatisfacció pel client: 2
Prioritat: 4

 # Requisit: 8
Tipus de requisit: 10b (Estil)
Descripció: El sistema ha de tenir un disseny simple i minimalista
Justificació: Aquest tipus de disseny és el que es porta entre la majoria de plataformes similars
Criteri d'acceptació: Un 75% dels usuaris que l'utilitzen consideren que és simple i minimalista.
Satisfacció pel client: 3
Insatisfacció pel client: 2
Prioritat: 4

 # Requisit: 9
Tipus de requisit: 11d (Comprensió i cortesia)
Descripció: El llenguatge usat al sistema ha de ser clar i concís per evitar ambigüitats
Justificació: Necessari per evitar que cap usuari pateixi un malentès.
Criteri d'acceptació: Un 75% dels usuaris que l'utilitzen consideren que llenguatge és simple i concís
Satisfacció pel client: 3
Insatisfacció pel client: 2
Prioritat: 4

 # Requisit: 10
Tipus de requisit: 15b (Integritat)
Descripció: El sistema no acceptarà dades que puguin ser incorrectes, ja sigui pel seu tipus o pel seu contingut.
Justificació: És molt important que les dades processades pel sistema siguin íntegres. Això vol dir que siguin vàlides (certes) i completes.
Criteri d'acceptació:El sistema no permet que s'introdueixin dades no vàlides.
Satisfacció pel client: 3
Insatisfacció pel client: 2
Prioritat: 4

6.3 Casos d'ús

Un cop sabuts els requisits funcionals que s'implementaran s'ha fet un diagrama de casos d'ús sobre l'actor que els realitzarà i també s'han especificat.

6.3.1 Diagrama de casos d'ús

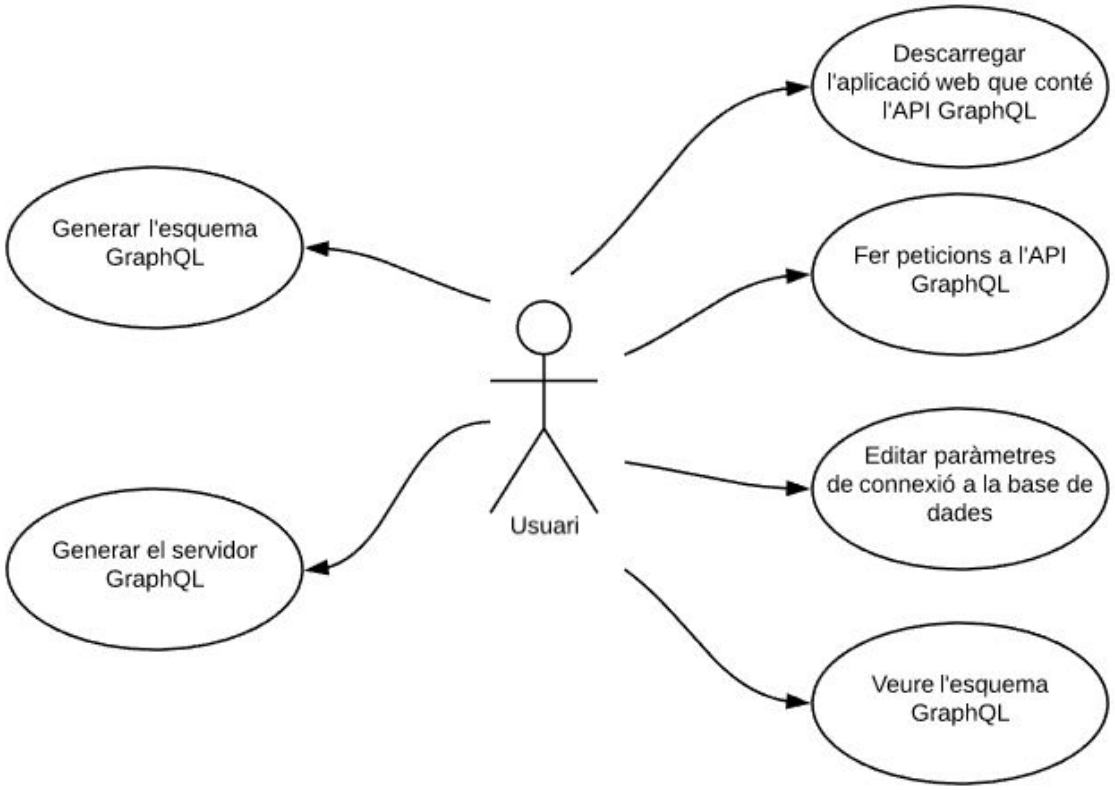


Figura 2. Diagrama de casos d'ús

6.3.2 Especificació dels casos d'ús

Cas d'ús 1: Generar l'esquema GraphQL
Actor principal: Usuari
Precondició: -
Disparador: L'usuari vol generar l'esquema GraphQL

Escenari principal d'èxit:

1.El sistema mostra un formulari a l'usuari tal que pugui introduir la informació necessària per poder generar l'esquema GraphQL.
2. L'usuari introdueix la informació mínima necessària (Adreça de connexió al servidor que conté l'ontologia, usuari i contrasenya per connectar-se al servidor i el nom de la base de dades que es vol accedir)
3. L'usuari confirma les dades introduïdes.
4. El sistema valida les dades.
5. El sistema genera l'esquema GraphQL.

Extensions del cas d'ús:

4a. No s'ha introduït tota la informació mínima necessària o aquesta és errònia.
4a1. El sistema notifica a l'usuari que hi ha hagut un error.
4a2. El sistema retorna l'usuari al punt 2.

Cas d'ús 2: Generar el servidor GraphQL
Actor principal: Usuari
Precondició: -
Disparador: L'usuari vol generar el servidor GraphQL

Escenari principal d'èxit:

1.El sistema mostra un formulari a l'usuari tal que pugui introduir la informació necessària per poder generar el servidor GraphQL.
2. L'usuari introdueix la informació mínima necessària (Adreça de connexió al servidor que conté l'ontologia, usuari i contrasenya per connectar-se al servidor i el nom de la base de dades que es vol accedir)
3. L'usuari confirma les dades introduïdes.
4. El sistema valida les dades.
5. El sistema genera el servidor GraphQL.

Extensions del cas d'ús:

4a. No s'ha introduït tota la informació mínima necessària o aquesta és errònia.
4a1. El sistema notifica a l'usuari que hi ha hagut un error.

4a2. El sistema retorna l'usuari al punt 2.

Cas d'ús 3: Descarregar l'aplicació web que conté l'API GraphQL
Actor principal: Usuari
Precondició: S'ha generat el servidor i l'esquema GraphQL
Disparador: L'usuari vol descarregar-se l'aplicació web que conté l'API GraphQL

Escenari principal d'èxit:

1.El sistema comprimeix l'aplicació web que conté l'API GraphQL
2.El sistema descarrega l'arxiu comprimit a la part del client.

Cas d'ús 4: Fer petició a l'API GraphQL
Actor principal: Usuari
Precondició: S'ha generat el servidor i l'esquema GraphQL
Disparador: L'usuari vol fer una petició a l'API GraphQL

Escenari principal d'èxit:

1.El sistema mostra una interfície gràfica a l'usuari tal que pugui definir la consulta desitjada.
2. L'usuari defineix la consulta a fer.
3. L'usuari fa la petició a l'API GraphQL.
4. El sistema valida la consulta feta per l'usuari.
5. El sistema retorna la informació desitjada per l'usuari.

Extensions del cas d'ús:

4a. La consulta definida és errònia.
4a1. El sistema notifica a l'usuari que hi ha hagut un error.
4a2. El sistema retorna l'usuari al punt 2.

Cas d'ús 5: Editar paràmetres de connexió a la base de dades
Actor principal: Usuari
Precondició: S'ha generat el servidor i l'esquema GraphQL
Disparador: L'usuari vol editar els paràmetres de connexió de la base de dades
Escenari principal d'èxit:

1.El sistema mostra un formulari a l'usuari tal que pugui introduir la informació necessària per poder editar els paràmetres de connexió a la base de dades i també mostra els paràmetres actuals de connexió a la base de dades.
2. L'usuari introdueix la informació mínima necessària (Adreça de connexió al servidor que conté l'ontologia, usuari i contrasenya per connectar-se al servidor i el nom de la base de dades que es vol accedir)

3. L'usuari confirma les dades introduïdes.
4. El sistema valida les dades.
5. El sistema edita els paràmetres de connexió a la base de dades.

Extensions del cas d'ús:

4a. No s'ha introduït tota la informació mínima necessària o aquesta és errònia.
4a1. El sistema notifica a l'usuari que hi ha hagut un error.
4a2. El sistema retorna l'usuari al punt 2.

Cas d'ús 6: Veure esquema GraphQL
Actor principal: Usuari
Precondició: S'ha generat el servidor i l'esquema GraphQL
Disparador: L'usuari vol veure l'esquema GraphQL generat

Escenari principal d'èxit:

1.El sistema mostra l'esquema GraphQL generat a l'usuari.

7. APIs GraphQL

GraphQL és un llenguatge de consultes que permet definir quines dades es volen demanar a una API GraphQL. Llavors, una API GraphQL és una API que tindrà la capacitat de retornar una resposta a les consultes GraphQL que se li enviïn.

Algunes de les característiques de les API GraphQL són que retornen la informació que a l'usuari l'interessa i que només tindrà un punt d'entrada el qual tindrà la capacitat de respondre totes les peticions que se li facin.

Aquestes característiques resolen alguns problemes que tenen les API REST, per exemple:

- Retorn d'informació innecessària: l'API REST pot retornar informació que no l'interessa al client, en canvi l'API GraphQL només retornarà la informació que vol el client.
- Múltiples peticions: A l'API REST, a vegades li hem d'enviar diverses peticions per obtenir la informació que ens interessa, en canvi l'API GraphQL aquesta informació la pots obtenir fent una sola petició.

A la següent imatge, es poden veure aquestes diferències que hi ha entre una API GraphQL i una API REST. Imaginem que el client vol obtenir el nom d'un suburbi concret i nom de les parades que proporciona.

Figura 3. Diferències entre una API GraphQL i una API REST

Es veu que l'API GraphQL retorna la informació que vol l'usuari només fent-li una petició, en canvi l'API REST per aconseguir la mateixa informació se li ha de fer dues peticions i a més la resposta d'aquestes peticions contenen informació addicional que no vol el client (habitants i stopAddress).

En una API GraphQL, a part de demanar dades, es poden modificar o crear, però en aquest projecte només ens centrarem en els aspectes que influeixen en una consulta, ja que els altres no formen part de l'abast del projecte.

En aquest apartat s'explicarà alguns elements bàsics que pot tenir una consulta GraphQL i els components que ha de tenir una API GraphQL funcional, l'esquema i el servidor.

7.1 Consultes amb GraphQL

A continuació es mostren els elements més comuns que poden tenir les consultes amb GraphQL, aquestes consultes les haurà de definir el client.

7.1.1 Camps

La consulta més simple és demanar que ens retornin el valor dels camps específics d'un objecte.

```
{
  allBicingStations{
    idInstance
    stationID
  }
}
```

```
{
  "data": {
    "allBicingStations": [
      {
        "idInstance": "http://www.example.com/EstacioBici",
        "stationID": "1"
      },
      {
        "idInstance": "http://www.example.com/EstacioBici2",
        "stationID": "2"
      },
      {
        "idInstance": "http://www.example.com/EstacioBici3",
        "stationID": "3"
      }
    ]
  }
}
```

Figura 4. Consulta dels camps d'un objecte

En aquest cas, de totes les estacions de bicis es consulten els camps “idInstace” i “stationID”

7.1.2 Paràmetres

A un camp li podem ficar paràmetres, sempre que es requereixi.

Figura 5. Consulta amb paràmetre

L’objecte amb id : “http://www.example.com/EstacioBici2” se li consulten els camps “stationID” i “stationBikesNumber”

7.1.3 Fragments

Els fragments permeten construir una col·lecció de camps i després incloure’ls a les consultes que vulguis. Aquests camps sempre han d’estar presents en els objectes sinó la consulta retornarà un error.

Figura 6. Consulta utilitzant un fragment

7.1.4 Variables

A l’apartat 7.1.2 hem vist que es pot posar un String com a paràmetre en un camp que el requereix, però aquest String el podem passar com una variable també.

Figura 7. Consulta utilitzant una variable

Com es pot veure a la variable se li ha de dir de quin tipus és, en el nostre cas als únics camps que se li podrà posar un paràmetre seran els que retornin un objecte identificat per un id i aquest paràmetre serà del tipus “String!” sempre.

7.1.5 Inline Fragments

Si s’està consultant un camp que retorna una “Interface”, explicat a l’apartat 7.2.6, s’haura d’utilitzar els “Inline Fragments” si volem consultar els camps d’un object type que implementa aquella “Interface”.

Imaginem que tenim la següent herència:

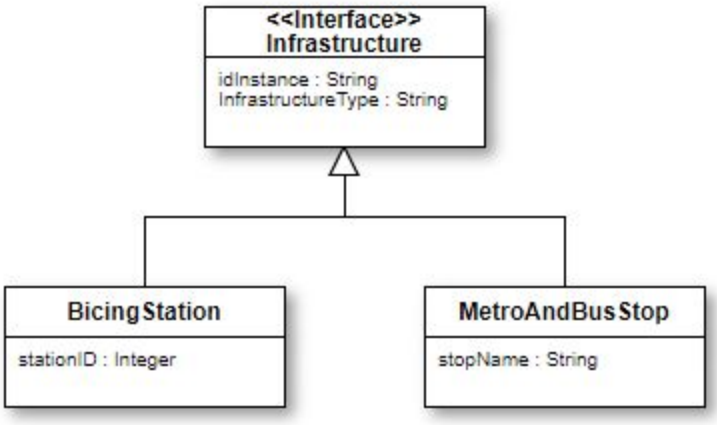


Figura 8. Herència per explicar els Inline Fragments

I fem una consulta a un camp el qual retorna una llista d'Infrastructure, les quals poden ser del tipus BicingStation o MetroAndBusStop, llavors es consulten els camps que existeixen en qualsevol Infrastructure sigui el tipus que sigui i per últim, si es vol consultar els camps específics d'una Infrastructure que sigui un BicingStation o un MetroAndBusStop s'haurà d'especificar amb un "Inline Fragment".

Figura 9. Consulta amb Inline Fragments

7.2 Esquema GraphQL

Un cop vist el format de les consultes que es poden fer, podem veure que es consulten certs camps els quals poden retornar objectes o escalars. L'usuari d'alguna forma haurà de saber quins camps pot consultar, quin tipus d'objecte retornen, quins camps es pot consultar d'aquest objecte retornat i és l'esquema GraphQL que tindrà aquesta informació. A l'annex B es pot veure un exemple d'un esquema GraphQL.

Per tant, l'esquema defineix quines consultes pot fer el client i quin serà el tipus de retorn d'aquests i estarà constituit per varis type que explicaré a continuació.

S'ha de dir que només s'expliquen els elements que es tenen en compte a l'hora de generar un esquema GraphQL. El llenguatge que s'utilitzarà per escriure l'esquema serà el "GraphQL schema language" que han creat els mateixos integrants de GraphQL.

7.2.1 Object type i camps

El component més bàsic d'un esquema GraphQL són els Object type, que representen un tipus d'objecte el qual pots obtenir fent peticions a l'API GraphQL.

```
type Suburb {
    suburbName: String!
    providesStop: [MetroAndBusStop]!
}
```

Figura 10. Object type

En la imatge es pot veure que:

- Suburb és un Object type el qual té alguns camps.
- suburbName i provideStop camps del type Suburb. Això vol dir que suburbName i provideStop són els únics camps que podem fer servir quan operem amb el type Suburb.
- String és un Scalar type, és el tipus del valor que retornarà el camp.
- String!, l'exclamació vol dir que el camp retornarà un valor no nul, és a dir, no podrà retornar un valor nul per aquest camp.
- [MetroAndBusStop]!, en aquest cas representa que el camp retornarà un valor que serà una llista no nul·la d'objectes MetroAndBusStop. El fet de retornar una llista no nul·la vol dir que només podrà retornar una llista amb mida 0 o més.

7.2.2 Paràmetres

Cada camp pot tenir 0 o més paràmetres. Per exemple:

```
getDistrict(id: String!): District
```

Figura 11. Camp amb un paràmetre

Tots els paràmetres tenen un nom. En aquest cas el camp getDistrict té un paràmetre am el nom “id” que és de tipus String!

7.2.3 Query Type

El query type ens permet definir els camps “arrel”, és a dir, tota consulta GraphQL haurà de començar amb un d’aquests camps i també li haurem de dir que l’esquema contingui aquest query type, sinó no podrem consultar aquests camps “arrel”.

```
type Query {
    allGeographicalCoordinates: [GeographicalCoordinate]
    getGeographicalCoordinate(id: String!): GeographicalCoordinate
    allDistricts: [District]
    getDistrict(id: String!): District
    allMetroAndBusStops: [MetroAndBusStop]
    getMetroAndBusStop(id: String!): MetroAndBusStop
    allBicingStations: [BicingStation]
    getBicingStation(id: String!): BicingStation
    allSuburbs: [Suburb]
    getSuburb(id: String!): Suburb
}

schema {
    query: Query
}
```

Figura 12. Query Type

7.2.4 Scalar Type

A continuació es mostren quins valors escalars pot retornar un camp:

- Int : un enter de 32-bits amb signe
- Float: un valor de coma flotant de doble precisió amb signe
- String: una seqüència de caràcters en format UTF-8
- Boolean: true o false
- ID: representa un identificador únic.

7.2.5 Llistes i no nuls

Podem definir que el valor que retorni un camp pugui ser una llista o un no nul.

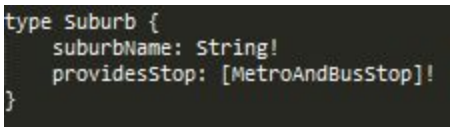


Figura 13. Llistes i no nuls

Com podem veure a la imatge per representar que el valor que retorni un camp és no nul s'haurà d'afegir al final del tipus d'aquest valor una exclamació "!". Això voldrà dir que el camp sempre retornarà un valor no nul i en el cas que es retornés un nul es dispararà una excepció de GraphQL.

Per representar que el valor que retorni un camp és una llista haurem de ficar el tipus de retorn entre claudàtors "[" i "]".

Les llistes i els no nuls es poden combinar entre si per tal de formar, per exemple llistes no nules "camp : [String]!".

7.2.6 Interfaces

Una interface és un type polimòrfic, és a dir, un type que pot ser de més d'un tipus. Aquest type pot ser implementat per un o més Object Type, el que implica aquesta implementació és que l'Object type haurà d'heretar els camps de l'interface i que l'Interface podrà ser del tipus de l'Object type que l'implementa.

Per exemple, podem tenir una interface amb nom Infrastructure que representa totes les infrastructures que estan en un barri, aquest serà el nostre type polimòrfic.

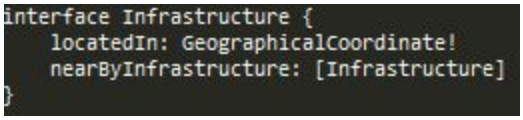


Figura 14. Interface

I aquí tenim alguns Object types que han implementat l'interface.

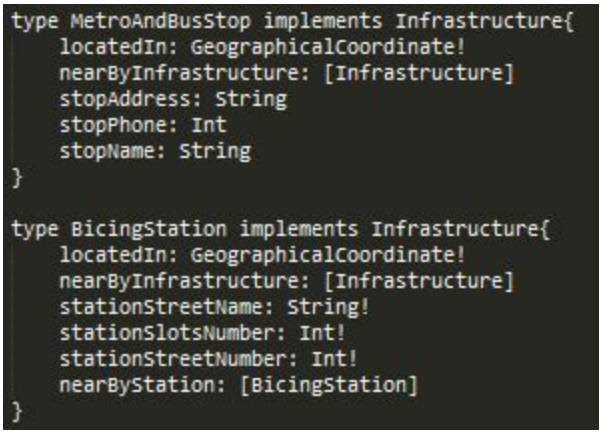


Figura 15. Objects type implementant una Interface

Es pot veure que els dos Object type tenen els camps de l'interface, però també tenen altres camps addicionals que són específics pel seu type, a més sabrem que en aquest cas, una Infrastructure podrà ser del tipus BicingStation o MetroAndBusStop.

7.3 Servidor GraphQL

El servidor GraphQL serà qui agafi les consultes GraphQL i les resolgui de tal forma que pot retornar la informació desitjada per l'usuari.

Aquest servidor pot ser implementat en diversos llenguatges de programació, en el nostre cas farem servir Java.

Degut el nul coneixement que es tenia per fer un servidor GraphQL implementat en java s'ha anat seguint recomanacions que proposen experts en el tema [14]

Les classes que hauria de tenir serien els següents:

1. Un servlet que serà una classe que permetrà que l'usuari interactui amb el servidor GraphQL i es qui rebrà les consultes GraphQL.
2. Una classe que tingui uns mètodes que representin els camps "arrel" del type query.

Per cada Object type:

3. Una classe la qual implementi l'obtenció d'una instància o totes les instàncies d'aquest Object type.
4. Una classe que implementi l'obtenció de valor dels camps d'aquest Object type i en cas que implementi una Interface, indicar-ho.

Per cada Interface:

5. Una interface que tingui uns mètodes que representin els camps de l'Interface de l'esquema GraphQL.

Les classes seran generades per l'algoritme i les llibreries que s'utilitzin gestionaran aquestes de tal forma que executin de forma correcta les consultes que arribin al servidor GraphQL i retornaran una resposta JSON.

8. Generacions de l'esquema GraphQL i servidor GraphQL

Un cop explicats els continguts que hauria de tenir l'esquema GraphQL i el servidor GraphQL, s'explicarà de quina forma podem aconseguir generar-los.

8.1 Disseny del metamodel de les ontologies

Com hem dit l'apartat 2 el nostre software ha d'estar preparat per generar una API GraphQL a partir d'una ontologia que és variable, això vol dir que pot ser qualsevol ontologia. Tenir un metamodel de les ontologies ens ha permès que els algoritmes puguin generar una API GraphQL a partir de qualsevol ontologia que utilitzi els elements i relacions que permet el metamodel. Llavors si l'ontologia que ens donen com a base per crear l'API GraphQL utilitza els elements i relacions que permet el metamodel, podrem crear l'API GraphQL corresponen a aquella ontologia.

S'ha de dir que s'ha necessitat l'ajuda del co-director per crear el metamodel i les seves respectives versions.

8.1.1 Descripció del metamodel de les ontologies

A continuació es mostra el metamodel final i una explicació sobre aquest. Aquest metamodel estarà definit amb el llenguatge RDF i RDFS, les ontologies també vindran definides d'aquesta forma.

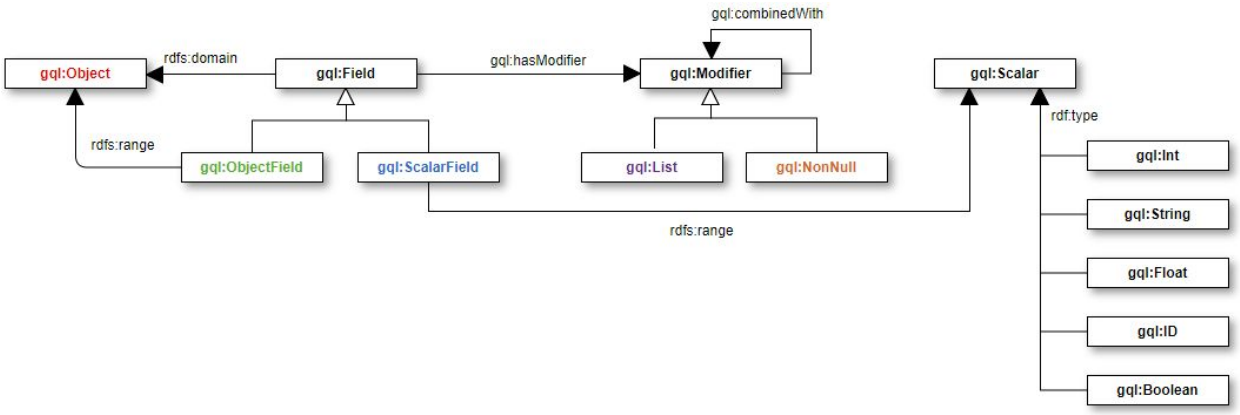


Figura 16. Metamodel final utilitzat per les generacions

Aquest metamodel està definit en un fitxer a part el qual podrem trobar a l'annex C i farà servir conceptes que ja estan definits pels llenguatges RDF [15] o RDFS [16].

En el metamodel es pot veure que representa elements de l'esquema GraphQL que hem vist anteriorment. El que ens explica el metamodel bàsicament és que un Field (camp) forma part d'un Object (Object Type), i aquest camp pot ser del tipus ObjectField que això vol dir que el valor que retorna el camp és un Object o també ScalarField que això vol dir que el valor que retorna el camp és un Scalar que pot ser un Int, String, Float, ID o Boolean. Per altra banda, un Field pot tenir un Modifier que pot ser del tipus List o NonNull, a més aquest Modifier es podrà combinar amb d'altres per tal que el camp pugui retornar llistes no nul·les per exemple.

8.1.2 Exemple d'una ontologia

A continuació es mostra una ontologia de forma gràfica descrita amb llenguatge RDF.

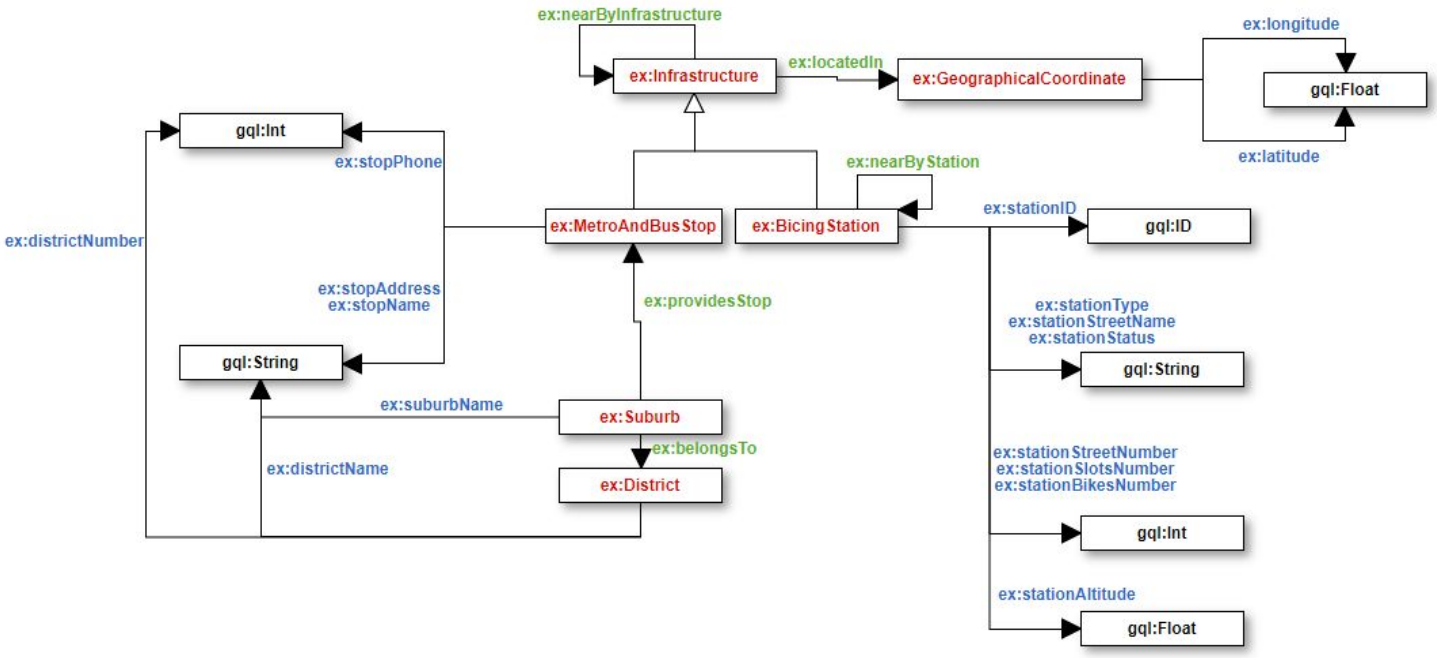


Figura 17. Segon exemple d'una ontologia

Els colors que hi ha en l'ontologia estan relacionats amb el metamodel, és a dir, que el color vermell correspon a un Object, el blau a un ScalarField, el verd a un ObjectField i el negre correspon a un escalar.
S'ha de tenir en compte que no es mostren els Modifier corresponents a cada Field per tal de no tenir una imatge molt carregada d'informació però a l'annex D es veurà com s'ha definit l'ontologia amb tota la seva informació.

Es pot veure que l'ontologia segueix les regles del metamodel presentat anteriorment.

8.1.3 Dissenys anteriors del metamodel

Per aconseguir el metamodel final que s'ha presentat, s'han hagut de fer modificacions en aquest cada cop que es trobava un obstacle que no es podia resoldre amb el disseny que es tenia.

A continuació es mostren els dissenys anteriors en els quals explicarem què es va canviar entre les versions i per què.

8.1.3.1 Primer disseny

Aquest és el disseny inicial del metamodel.

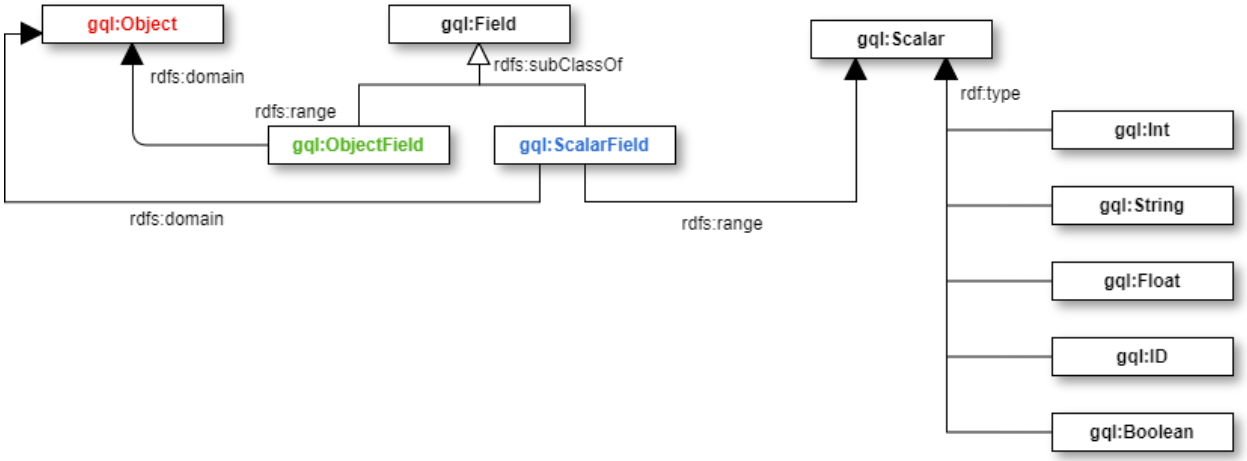


Figura 18. Primer disseny del metamodel

Com es pot veure té una certa semblança al metamodel final però tot i això li falten elements i regles per afegir.

8.1.3.2 Segon disseny

A continuació es mostra el segon disseny que s'ha tingut.

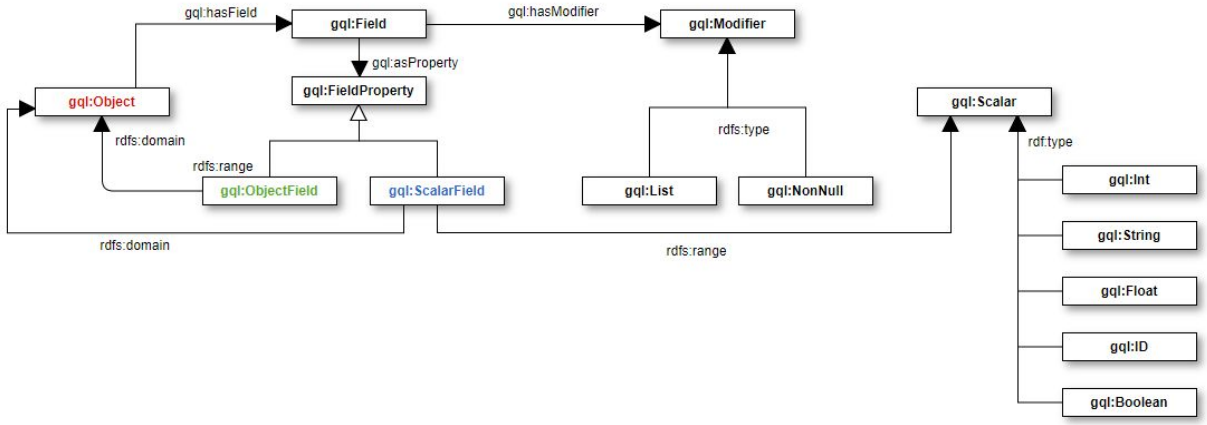


Figura 19. Segon disseny del metamodel

S'ha canviat de nom el concepte Field de l'anterior disseny a FieldProperty i s'ha afegit un nou concepte Field el qual farà la funció de guardar la informació d'un FieldProperty i també si té vinculat algun Modifier.

8.1.3.3 Tercer disseny

A continuació es mostra el tercer disseny que s'ha tingut.

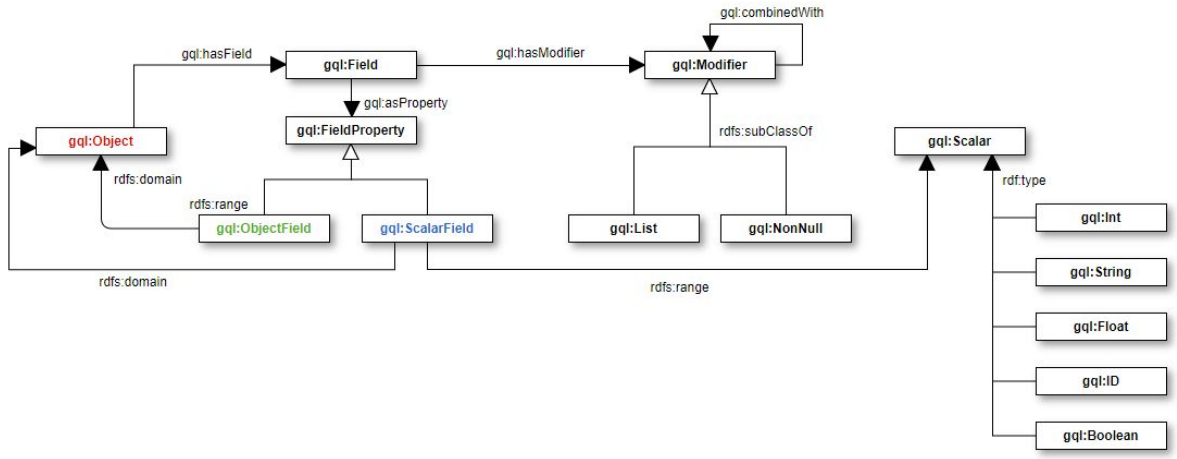


Figura 20. Tercer disseny del metamodel

En aquest disseny s'ha modificat la part del Modifier, abans es tenien definits els valors que podia tenir però ara s'han convertit en classes i es podran combinar entre si. De tal forma que podrem crear llistes no nul·les per exemple.

Per arribar al metamodel final s'ha decidit modificar la part del Field i FieldProperty perquè s'ha vist que era innecessari tenir-ho perquè amb un metamodel més simple es podia arribar a fer el mateix.

8.2 Exemple pràctic

A l'annex E es mostra la traducció dels conceptes i relacions d'aquesta ontologia, que té els elements i relacions del metamodel, a l'esquema GraphQL i al servidor GraphQL. Es farà servir una porció de l'ontologia d'exemple de l'apartat 8.1.2 per fer-ho més entenedor.

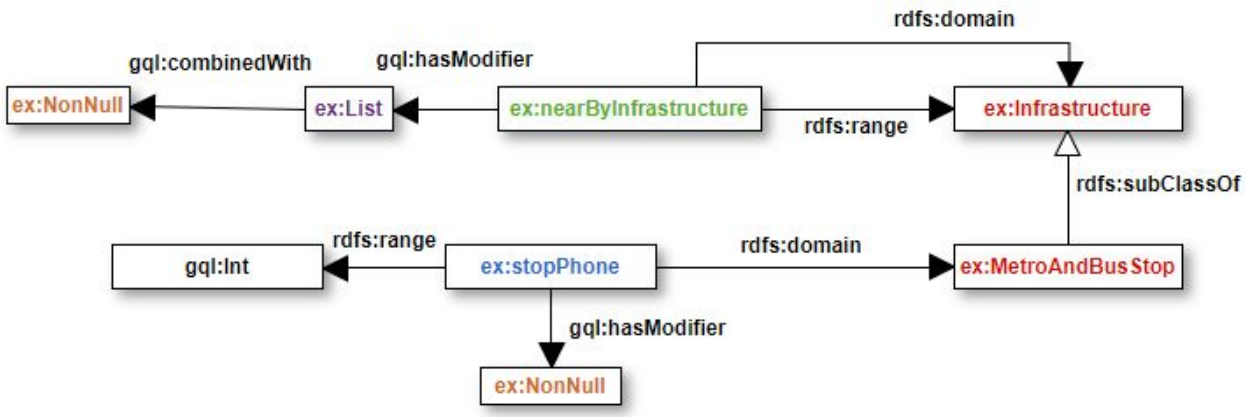


Figura 21. Tercer exemple d'una ontologia

8.2.1 Explicació de l'exemple

Com es pot veure a la imatge, els conceptes Infrastructure i MetroAndBusStop són del tipus "gql:Object", per tant es crea un Type per cadascú a l'esquema GraphQL i una classe java al servidor GraphQL per representar aquests dos conceptes.

També es pot veure que el concepte Infrastructure té un “gql:ObjectField”, és a dir un camp que retorna un Object i a aquest camp també se li diu que haurà de retornar una llista no nu·la, per tant, en aquest cas retornarà una llista no nul·la d’objectes Infrastructure. Per altra banda, el concepte MetroAndBusStop retorna un “gql:ScalarField” és a dir un camp que retorna un Scalar i a aquest camp se li diu que ha de ser no nul, per tant, en aquest cas retornarà un Integer no nul. Llavors als Types de l’esquema GraphQL se’ls hi afegeix aquests camps amb el valor que retornen i a les classes java se’ls hi afegeix uns mètodes que representaran aquests camps i el valor que retornen.

També veiem que el concepte MetroAndBusStop és subclasse d’Infrastructure, per tal que hereti els camps d’Infrastructure no ho podem fer directament del Type Infrastructure, sinó que s’haurà de crear una Interface del concepte pare i que el fill l’implementi, per tant es crea una Interface del pare a l’esquema GraphQL i al servidor GraphQL amb els camps que té i fem que el fill implementi aquesta Interface i hereti els seus camps.

8.3 Afegiments que s’hauran de generar a l’esquema GraphQL i servidor GraphQL

Com s’ha vist a l’anterior apartat, donada una ontologia podem arribar a generar una part de l’esquema GraphQL i servidor GraphQL però no arriba a ser suficient per satisfer les necessitats del client i tenir una API GraphQL funcional.

A continuació es mostren els afegiments que es generen a l’esquema i al servidor GraphQL:

Esquema GraphQL:

- S’afegeix el type query el qual conté el camps “arrel”, en el nostre cas s’ha decidit que per cada Object type es faci un camp “arrel” que pugui obtenir totes les seves instàncies i un altre per obtenir una instància a partir del seu identificador.

- S’afegeix un camp “idInstace” a cada Object type de tal forma que puguem saber l’identificador de cada instància d’aquest Object Type.
- Qualsevol Interface creada d’un concepte pare ha de ser implementada per l’Object Type del concepte pare, això provoca que quan consultem un camp que retorni aquesta Interface, aquesta podrà ser del tipus del pare o del fill .
- Cap camp tindrà com a valor de retorn un concepte pare, sempre que es consulti un concepte pare s’estarà consultant la seva Interface.
- S’afegeix un camp als Object Type que implementin una Interface de tal forma que puguem saber de quin tipus és i puguem diferenciar-los entre ells. El nom d’aquest camp seguirà el següent format: <nomInterface> + “Type”

Servidor GraphQL:

- Els mateixos afegiments que hem fet a l’esquema GraphQL també hauran d’estar representats en el servidor.
- S’han de generar les classes restants per tenir un servidor GraphQL funcional, descrites en l’apartat 7.3.
- Generar la lògica de com obtenir el valor de retorn de cada camp.

9. Disseny de les aplicacions web

En aquest apartat es podrà veure el mapa de navegació de les pantalles, el disseny de les pantalles i l'aplicació del patró model-vista-controlador.

9.1 Mapa de navegació

A continuació es mostra el mapa de navegació de les pantalles:

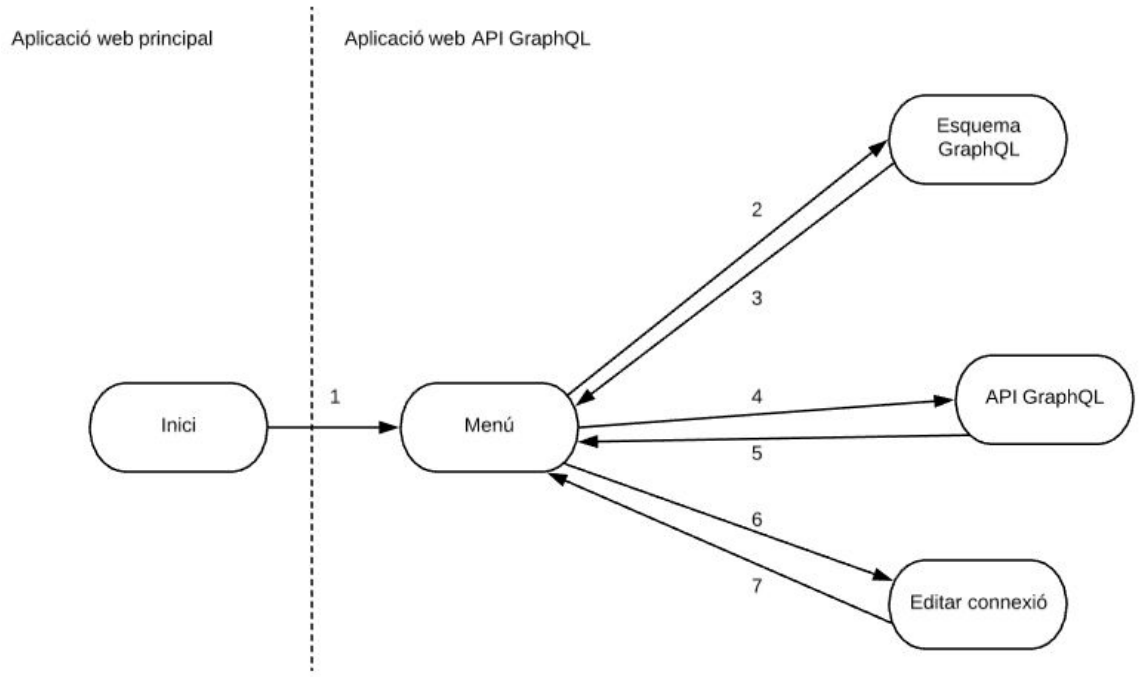


Figura 22. Mapa de navegació

Les accions que permeten navegar entre les pantalles són les següents:

1. Fer clic al botó que permet generar de forma automàtica l'esquema i servidor GraphQL
2. Fer clic al botó que permet veure l'esquema GraphQL
3. Fer clic al botó que permet tornar al menú
4. Fer clic al botó que permet accedir a fer peticions a l'API GraphQL
5. Fer clic al botó que permet tornar al menú
6. Fer clic al botó que permet editar els paràmetres de connexió a la base de dades.

7. Fer clic al botó que permet tornar al menú

9.2 Disseny de les pantalles

En aquest apartat s'inclouen captures de les pantalles i una breu descripció dels elements que tenen.

9.2.1 Inici

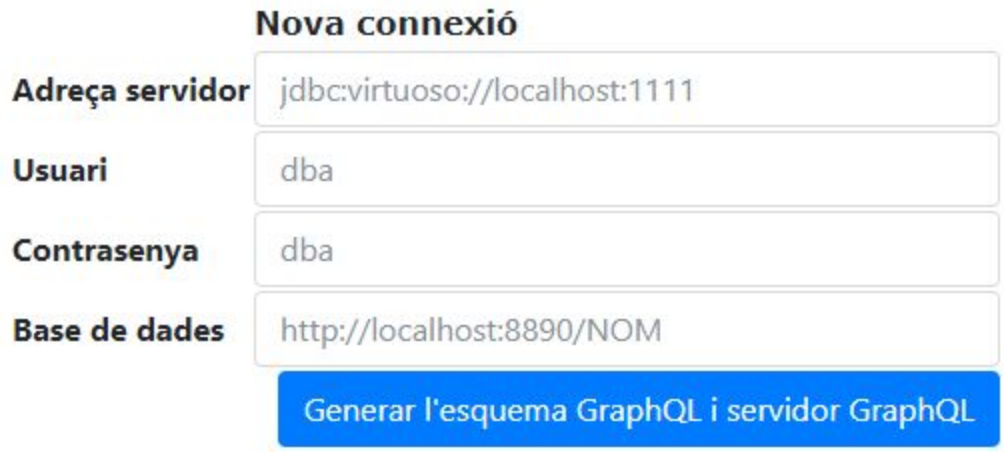


Figura 23. Pantalla inici

En aquesta pantalla es veu un formulari el qual l'usuari ha d'omplir ficant uns camps vàlids i complets amb l'objectiu de generar un esquema i servidor GraphQL, en el cas que aquests no ho siguin es mostrarà un missatge d'error. També es veu que hi ha "placeholders" per tal que l'usuari tingui una idea de què ficar a cada camp.

9.2.2 Menú

Figura 24. Pantalla menú

En aquesta pantalla s'ofereix a l'usuari triar una de les diferents funcionalitats que té l'aplicació web que conté l'API GraphQL.

9.2.3 Esquema GraphQL

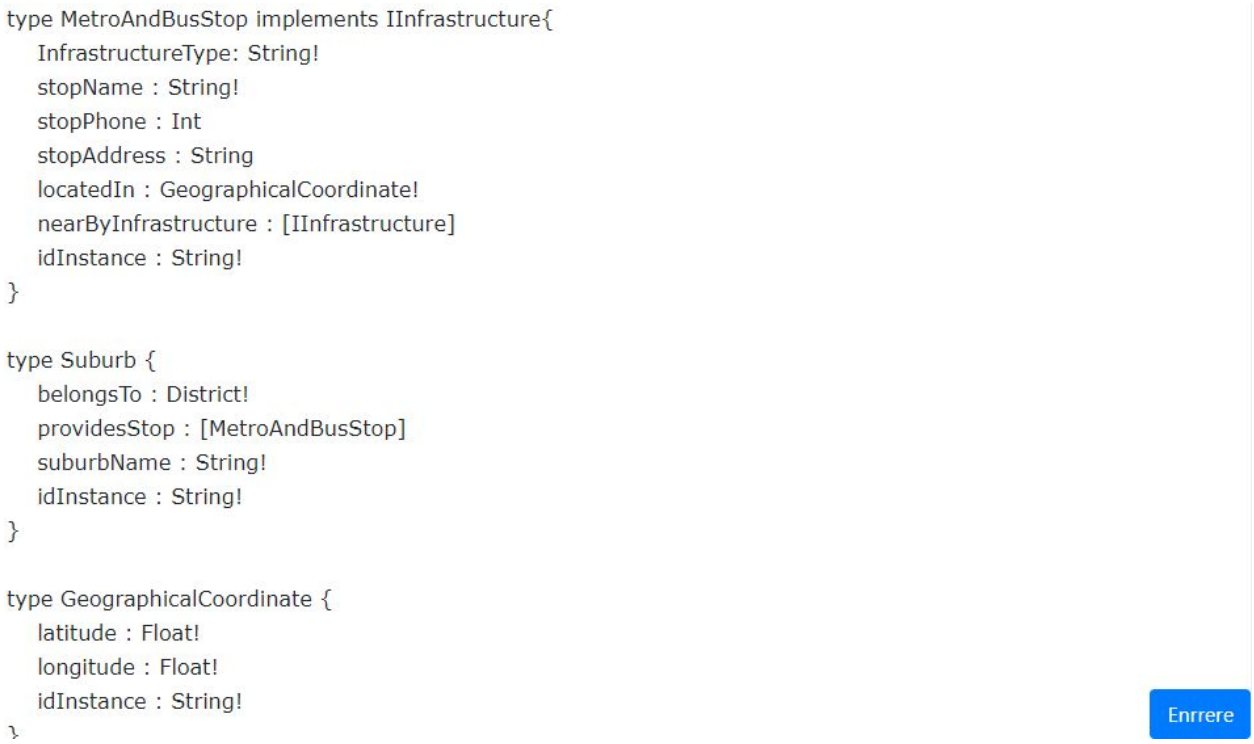


Figura 25. Pantalla esquema GraphQL

En aquesta pantalla es veu l'esquema GraphQL generat automàticament a partir d'una ontologia.

9.2.4 API GraphQL

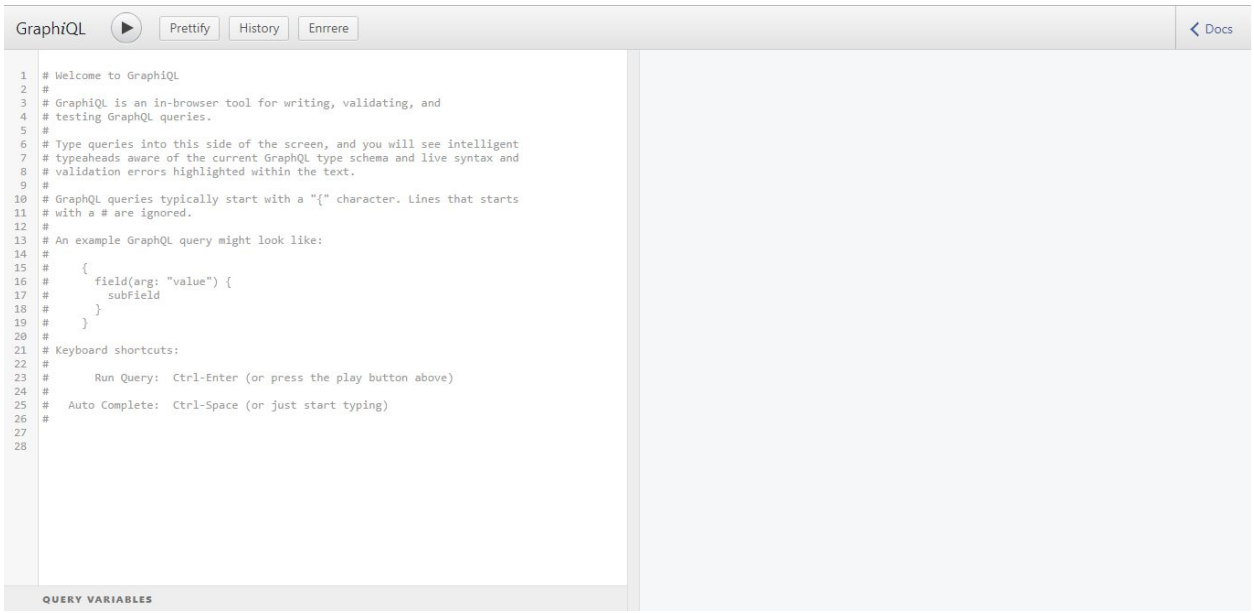


Figura 26. Pantalla API GraphQL

En aquesta pantalla l'usuari pot definir les seves consultes GraphQL i fer peticions a l'API GraphQL i veu el resultat d'aquestes en la mateixa pantalla. S'ha de dir que aquesta pantalla s'ha agafat del repositori de Github oficial de GraphQL[17].

9.2.5 Editar connexió

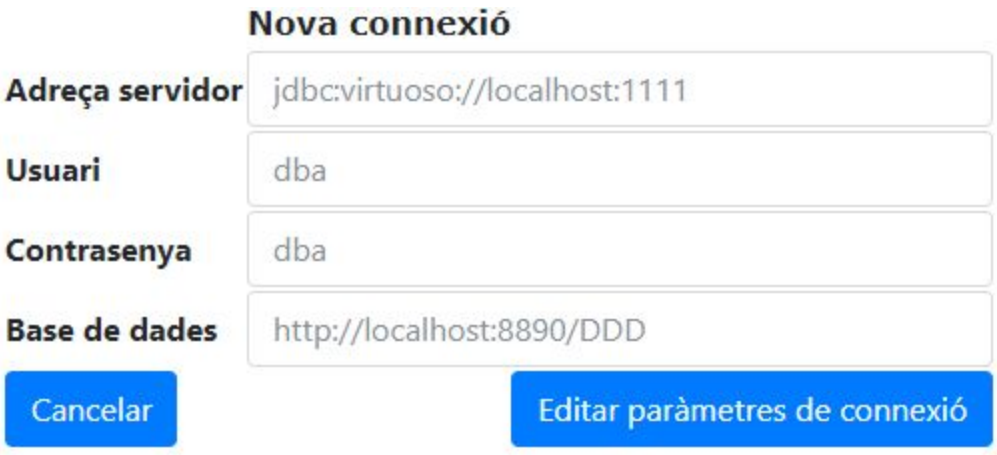


Figura 27. Pantalla editar connexió

Aquesta pantalla és similar a la “pantalla Inici” però el canvi més significatiu és que en el “placeholder” dels camps es poden veure els paràmetres actuals de connexió a la base de dades.

9.3 Patró Model-Vista-Controlador

En l’aplicació web que conté l’API GraphQL s’ha utilitzat el patró model-vista-controlador (MVC) en les pantalles d’editar els paràmetres de connexió a la base de dades i veure l’esquema GraphQL.

Aquest patró permet separar l’aplicació en tres components: el controlador, el model i la vista.

Model: El model representa la informació que l’usuari pot visualitzar per pantalla. Serà l’encarregat, en el nostre cas, de consultar l’arxiu que té guardat l’esquema GraphQL i de consultar i modificar l’arxiu que té guardat els paràmetres de connexió a la base de dades.
Vista: És la interfície d’usuari de la nostra aplicació web. S’encarregarà de presentar les dades obtingudes del controlador.
Controlador: És l’enllaç entre la vista i el model. Manipularà el model per tal de respondre a les accions que fa l’usuari sobre la vista.

El flux de control que s’ha seguit és el següent:

1. L’usuari realitza una acció a la interfície d’usuari i el controlador rep aquesta petició feta per l’usuari.
2. El controlador manipula el model perquè aquest resolgui la petició.
3. Un cop resolta, el model retornarà el resultat de la petició al controlador.
4. El controlador actualitzarà la vista passant-li la informació que li ha proporcionat el model.

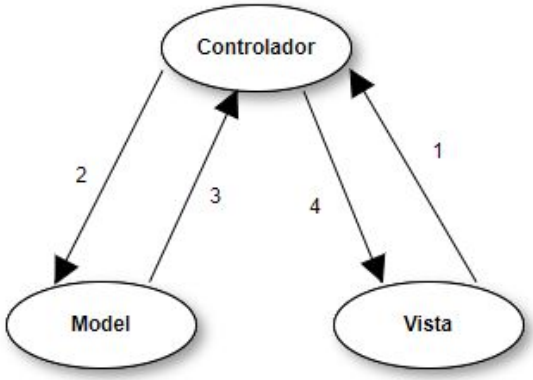


Figura 28. Patró MVC

9.3.1 Exemple diagrama de seqüència

A continuació es mostra el cas d'ús de veure l'esquema GraphQL en un diagrama de seqüència en el qual s'utilitza el patró MVC.

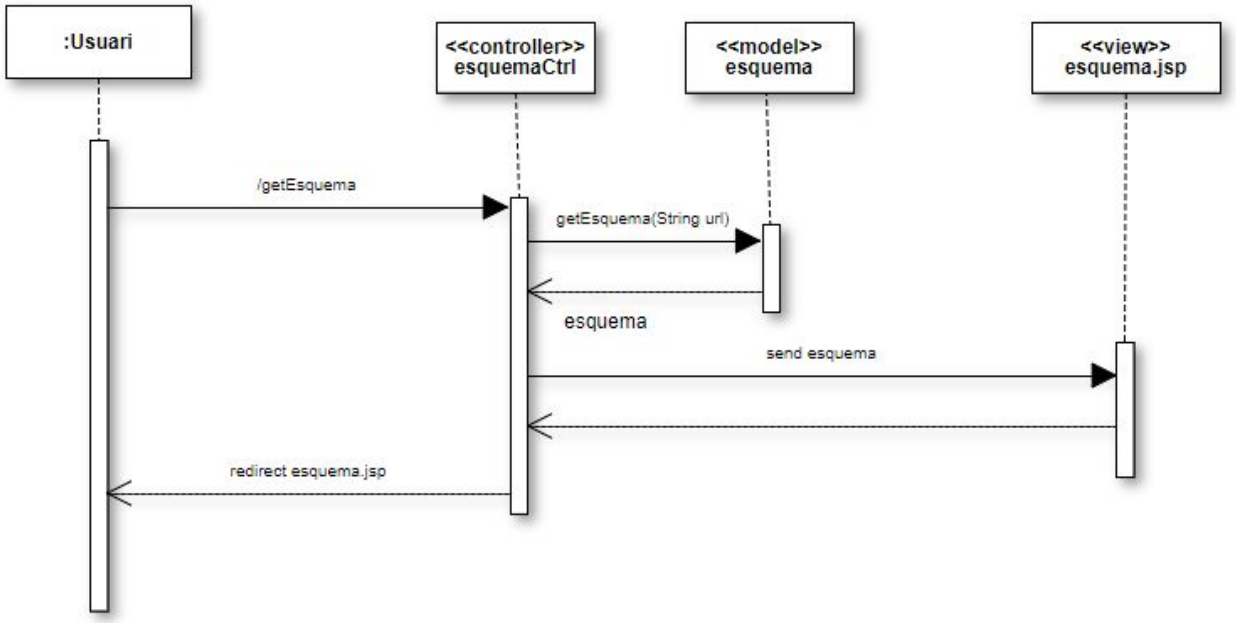


Figura 29. Diagrama de seqüència pel cas d'ús veure esquema GraphQL

El controlador rebrà la petició "/getEsquema" que ha generat l'usuari mitjançant la vista, després aquest controlador manipularà el model per tal que aquest resolgui la petició i

retorni l'esquema GraphQL, per arribar a fer-ho el controlador li passarà un string el qual contindrà la localització de l'esquema GraphQL, ja que està guardat en un arxiu.

Un cop el model retorni l'esquema GraphQL, el controlador li passarà a la vista i aquesta s'actualitzarà per tal de mostrar l'esquema que se li ha passat.

10. Implementació

Pel desenvolupament de les aplicacions web i les generacions automàtiques del servidor GraphQL i esquema GraphQL s'han utilitzat diverses tecnologies que ens han permès assolir els objectius del projecte. A continuació es fa una breu descripció de cada una i es justifica perquè s'ha utilitzat.

HTML

Descripció

HyperText Markup Language [18] (HTML) és un llenguatge de marcatge utilitzat per descriure l'estructura i el contingut, en forma de text, que poden tenir les pantalles. Per fer-ho s'utilitza etiquetes que permeten definir els elements que ens proporciona HTML. Actualment, World Wide Web Consortium (W3C) considera aquesta eina com un estàndard a l'hora de crear pàgines web.

Justificació

Aquesta eina es necessita per descriure l'estructura i contingut de les pantalles de les nostres aplicacions web.

CSS

Descripció

Cascading Style Sheets [18] (CSS) és un llenguatge d'estil que s'encarrega de la presentació dels elements descrits en un llenguatge de marcatge, alguns exemples poden ser el posicionament, el color que es doni als elements o el tipus de lletra que s'utilitzi, entre altres opcions. W3C, també considera aquesta eina com un estàndard a l'hora de crear pàgines web.

Justificació

Aquesta eina es necessita per posicionar els elements de les pantalles descrits amb HTML.

JavaScript

Descripció

JavaScript [19] és un llenguatge de programació el qual no necessita ser compilat i s'executa al navegador quan la pàgina és descarregada o l'usuari fa una acció sobre ella. S'usa per fer més dinàmiques les pàgines web i sense recarregar la pàgina pot ficar nou contingut en ella.

Justificació

S'usa quan l'usuari faci una petició a l'API GraphQL i s'ha de mostrar a la pàgina la informació que retorna en format JSON.

GraphQL schema language

Descripció

GraphQL schema language [20] és un llenguatge que s'utilitza per fer un esquema GraphQL.

Justificació

S'utilitzarà a l'hora de generar automàticament un esquema GraphQL.

Java

Descripció

Java [21] és un llenguatge de programació orientat a objectes desenvolupat per Sun Microsystems.

Justificació

S'ha decidit utilitzar aquest llenguatge de programació per fer el projecte perquè és amb el qual tinc més experiència i em sento més còmode. Es farà servir per fer les generacions automàtiques del servidor GraphQL i esquema GraphQL, a més s'utilitzarà per implementar el patró MVC.

Java Servlets

Descripció

Els java servlets [22] són classes java les quals donen resposta a peticions HTTP que fa el client.

Justificació

Cada funcionalitat tindrà un servlet associat el qual permetrà a l'usuari fer servir les funcionalitats disponibles que tenim a les aplicacions web.

.JSP

Descripció

JavaServer Pages [23] (JSP) és una tecnologia que permet combinar els llenguatges HTML, CSS i JavaScript amb llenguatge Java.

Justificació

En el patró MVC farà la funció de vista i ens permetrà agafar les dades que ens envia el Java Servlet que ha cridat l'usuari.

RDF i RDFS

Descripció

Resource Description Framework [24] (RDF) i RDF Schema [25] (RDFS) són llenguatges que permeten definir ontologies i instàncies dels conceptes de les ontologies amb tripletes, a més estan reconeguts com a estàndards pel W3C.

Format de les tripletes

L'element bàsic de les ontologies és la tripleta, que la podríem definir de la següent forma si ho pensem com si fos un graf.

Un node (el subjecte) està relacionat amb un altre node (destinatari) mitjançant un enllaç (predicat). Llavors el format seria el següent:

<subjecte> <predicat> <destinatari>

S'ha de dir que els tres components de la tripleta estan identificats amb una URI.

Exemple

Si volem representar una ontologia amb 2 conceptes (nodes) i una relació entre ells, com per exemple “Una persona coneix a una persona” ho haurem de fer de la següent manera:

1. Definim uns prefixos els quals ens escorçaran la llargada de les tripletes
   @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
   @prefix ex: <http://www.exemple.com/> .
   @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

2. Definim un concepte Persona amb la propietat rdfs:Class
   ex:Persona a rdfs:Class ;

3. Definim l’enllaç entre dues “ex:Persona” amb la propietat rdf:Property.
   ex:coneix a rdf:Property.
   ex:coneix rdfs:domain ex:Persona. (fa referència al subjecte de la tripleta)
   ex:coneix rdfs:range ex:Persona . (fa referència al destinatari de la tripleta)

Després si volem fer instàncies d’aquests conceptes, com per exemple “l’Albert coneix a la Maria” ho hauríem de fer de la següent manera:

Li donem un nom (URI) a les instàncies, diem que són persones i després les relacionem entre elles.

4. Fer instàncies dels conceptes
   ex:Maria a ex:Persona.
   ex:Albert a ex:Persona .
   ex:Albert ex:coneix ex:Maria .

Aquestes tripletes no formarien part d’una ontologia, ja que estem fent instàncies dels conceptes.

Justificació

Aquest llenguatge s’utilitzarà per el metamodel des les ontolgoies, ontologies i instàncies dels conceptes de l’ontologia.

Servidor virtuoso

Descripció

Virtuoso [26] és un servidor el qual permet emmagatzemar dades descrites amb els llenguatges RDF i RDFS.

Justificació

L'usuari el farà servir per a emmagatzemar l'ontologia i instàncies dels conceptes de l'ontologia, les quals estaran descrites amb el llenguatge RDF i RDFS.

SPARQL

Descripció

SPARQL [27] és un llenguatge de consultes RDF el qual permet obtenir dades emmagatzemades descrites amb els llenguatges RDF i RDFS en un servidor. També és un estàndard del W3C.

Justificació

A l'hora de fer la generació automàtica de l'esquema i servidor GraphQL s'haurà d'escriure consultes per tal d'obtenir informació de l'ontologia que ha proporcionat l'usuari per tal de crear l'esquema, a més, per la part del servidor GraphQL, haurà d'escriure consultes per obtenir els valors dels camps de les instàncies dels conceptes de l'ontologia que ha proporcionat l'usuari.

Virtuoso Jena Provider

Descripció

Virtuoso Jena Provider [28] és una llibreria de Java que permet realitzar una connexió al servidor Virtuoso i fer consultes sobre aquest.

Justificació

En el nostre codi Java l'haurem d'utilitzar amb la finalitat d'executar les consultes definides amb SPARQL i poder obtenir la informació retornada.

Servidor Apache

Descripció

Apache [29] és un servidor web de codi obert que s'usa per al desplegament d'aplicacions web.

Justificació

En el nostre projecte s'utilitzarà per desplegar les nostres aplicacions web.

GraphQL-Java i GraphQL-Java-Tools

Descripció

GraphQL-Java [30] és una llibreria de java que implementa l'especificació de GraphQL.

GraphQL-Java-Tools [31] és una llibreria de java que es basa en la llibreria GraphQL-Java però ens ofereix una sintaxi més simple per implementar el servidor GraphQL.

Justificació

S'utilitzen per implementar el servidor GraphQL.

Javapoet

Descripció

Javapoet [32] és una llibreria de java la qual crea classes .java en temps d'execució.

Justificació

S'utilitza en la generació de les classes que componen el servidor GraphQL.

Eclipse

Descripció

Eclipse [33] és una IDE usada per la programació normalment pel llenguatge Java

Justificació

S'utilitza per escriure el codi Java .

11. Diagrama de desplegament

A continuació es mostra el diagrama de desplegament que ens exposa l'arquitectura del sistema:

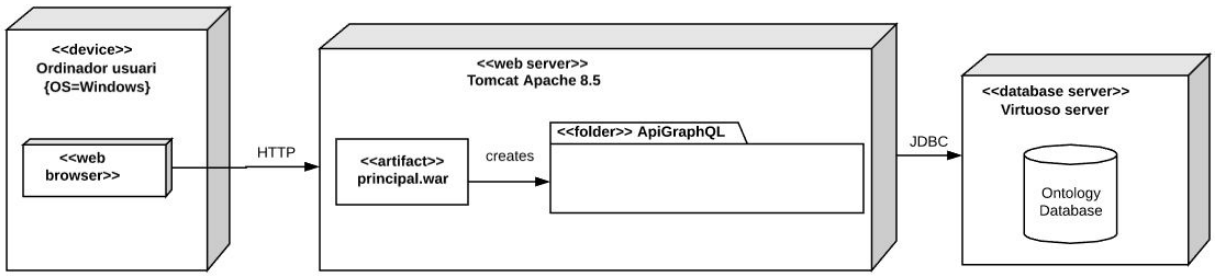


Figura 30. Diagrama de desplegament

Com es pot veure l'usuari, a través del seu navegador web, fa peticions HTTP al servidor Apache per tal d'utilitzar les aplicacions web que conté.

Per utilitzar les funcionalitats que té l'aplicació web que conté el codi per generar el servidor i l'esquema GraphQL tan sols haurem de desplegar l'arxiu "principal.war" al servidor Apache. Després, quan fem les generacions a partir de l'aplicació web principal, es crearà l'aplicació web que conté l'API GraphQL. El problema que hi ha és que aquesta no s'ha creat a partir del desplegament d'un arxiu .war, llavors el servidor Apache no reconeix els arxius Java compilats (.class) que fa servir l'aplicació.

Per solucionar això s'ha de configurar el servidor Apache de tal forma que estigui en mode "hot deploy", aquest mode s'aconsegueix modificant l'arxiu "context.xml" que està a la carpeta <DirectoriInstalacioApache>/<VersioTomcat>/conf/context.xml.

En aquest fitxer s'ha de localitzar la línia següent: <Context>
I canviar-la per la següent línia: <Context reloadable="true">

El que provocarà això serà que Apache revisi els directoris WEB-INF/classes i WEB-INF/lib de l'estructura de carpetes que tenen les aplicacions web per si hi ha hagut algun canvi i si és així, tornar a recarregar l'aplicació web.

Com que a l'hora de generar la nova aplicació web haurem creat aquests dos directoris i haurem afegit fitxers dins d'ells, el servidor Apache ho detectarà i començarà a recarregar l'aplicació web per tal que es puguin utilitzar les classes Java compilades.

Per últim, les aplicacions web a través d'una connexió JDBC podran accedir a la base de dades que contingui l'ontologia.

En el següent repositori de github es pot veure l'aplicació web principal: https://github.com/rodergas/TFG

En el següent repositori de github es pot veure l'aplicació web ApiGraphQL generada a partir de l'ontologia de l'apartat 8.1.2:
https://github.com/rodergas/ExempleAPIGraphQL

12. Proves

Aquest apartat té l'objectiu de mostrar com s'han provat els requisits funcionals i no funcionals.

Pel que fa als requisits funcionals tenim els següents:

1 - L'usuari vol generar automàticament un esquema GraphQL.

2 - L'usuari vol generar automàticament un servidor GraphQL.

3 - L'usuari vol fer peticions a l'API GraphQL.

4 - L'usuari vol descarregar-se l'aplicació web de l'API GraphQL.

5 - L'usuari vol editar els paràmetres de connexió a la base de dades.

6 - L'usuari vol veure l'esquema GraphQL.

Els tres primers requisits funcionals s'han provat conjuntament seguint els següents passos:

- Mitjançant el metamodel que es té es mira quins són els elements que podria tenir una ontologia.
- Es crea una ontologia la qual tingui tots els elements i regles que especifica el metamodel i es creen instàncies dels conceptes de l'ontologia.

Imaginem que tenim el primer disseny del metamodel de les ontologies (Figura 17), llavors podríem crear aquesta ontologia i aquestes instàncies dels conceptes de l'ontologia .

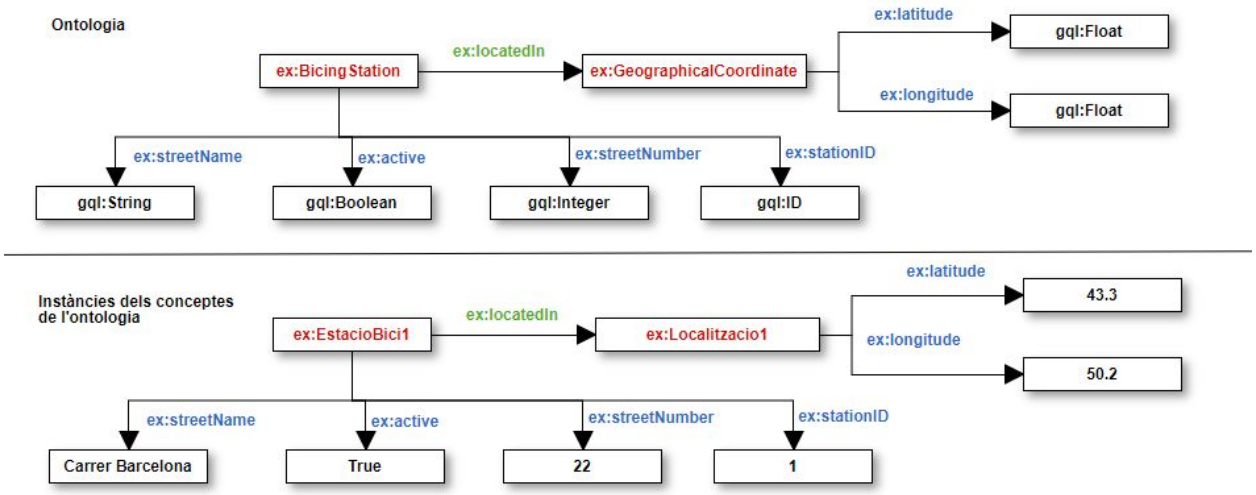


Figura 31. Instàncies dels conceptes d'una ontologia

- Es guarda l'ontologia i les instàncies en una base de dades.
- Els algoritmes, basant-se en les regles i elements del metamodel, generaran un esquema GraphQL i un servidor GraphQL per l'ontologia donada, la qual estarà guardada en una base de dades. Per tant hauran d'utilitzar uns paràmetres de connexió a la base de dades els quals l'usuari proporcionarà.
- La generació de l'esquema GraphQL i servidor GraphQL permetran formar l'API GraphQL.
- Es defineixen totes consultes possibles que l'API GraphQL pot contestar i s'envien mitjançant peticions, és a dir, es provarà que podem extreure tots els valors de les instàncies dels conceptes de l'ontologia els quals estan guardats a la base de dades.
- Si el resultat és el desitjat, podem afirmar les següents coses:
    - Que s'han pogut fer peticions a l'API GraphQL i aquestes retornen el resultat esperat (requisit funcional 3)
    - Si es retorna el resultat esperat llavors la generació automàtica de l'esquema GraphQL i servidor GraphQL per aquella ontologia són correctes.
    - Si el funcionament ha sigut correcte per l'ontologia donada, també serà correcte per qualsevol ontologia que contingui els elements i regles del metamodel que s'han provat, ja que els algoritmes es basen en el metamodel per fer les generacions automàtiques del servidor i esquema GraphQL. (requisits funcionals 1 i 2) i com que aquestes generacions són correctes es podran fer peticions a una API GraphQL i aquesta retornarà els resultats esperats (requisit funcional 3)

D'una forma més senzilla es mostren les proves que s'han fet als altres requisits:

- Pel que fa al quart requisit tan sols s'haurà de veure que es descarrega un arxiu comprimit de l'aplicació web que conté l'API GraphQL i després tornar-lo a desplegar al servidor web per tal de confirmar que s'ha descarregat correctament.
- Pel que fa al cinquè requisit tan sols s'haurà de veure que l'esquema generat correspon a l'ontologia donada.
- Pel sisè requisit quan s'editin els paràmetres, s'ha de veure que el model agafi els nous paràmetres i que la vista els mostri per pantalla, si és així, voldrà dir que s'han editat correctament.

Per altra banda tenim els requisits no funcionals que són els següents:
1 - El sistema ha de ser atractiu pels usuaris.

2 - El sistema ha de tenir un disseny simple i minimalista

3 - El llenguatge usat al sistema ha de ser clar i concís per evitar ambigüitats

4 - El sistema no acceptarà dades que puguin ser incorrectes, sigui pel seu tipus o pel seu contingut.

Per veure que s'han completat els tres primers requisits no funcionals s'ha fet una enquesta on-line la qual han contestat un total de 19 persones, no ha donat temps a què contestes més gent, ja que l'enquesta s'ha obert uns dies abans que finalitzes el projecte. Els resultats han sigut els següents:

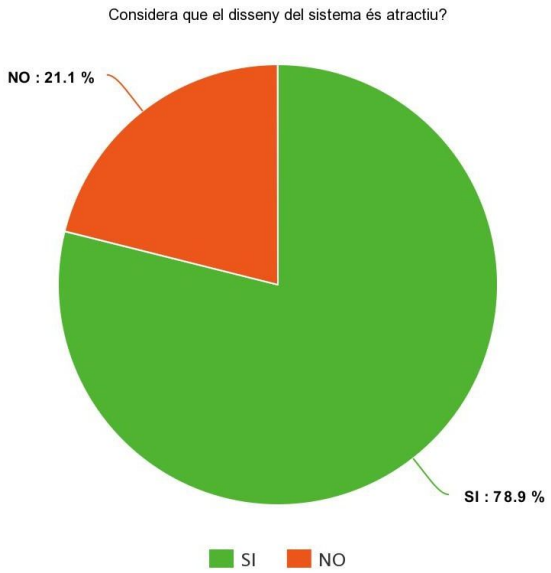


Figura 32. Requisit no funcional #1

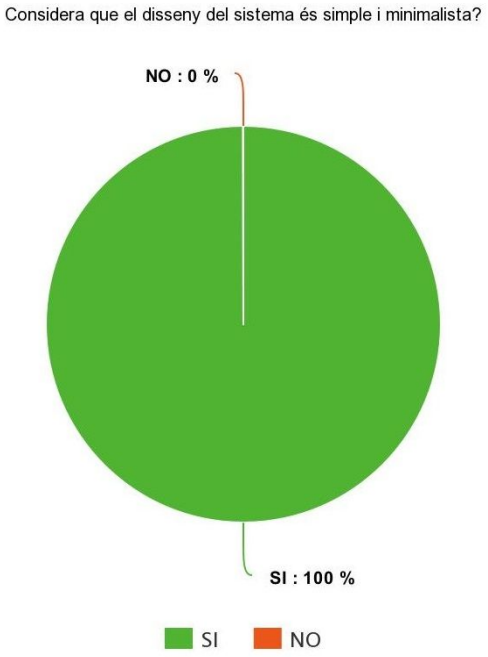


Figura 33. Requisit no funcional #2

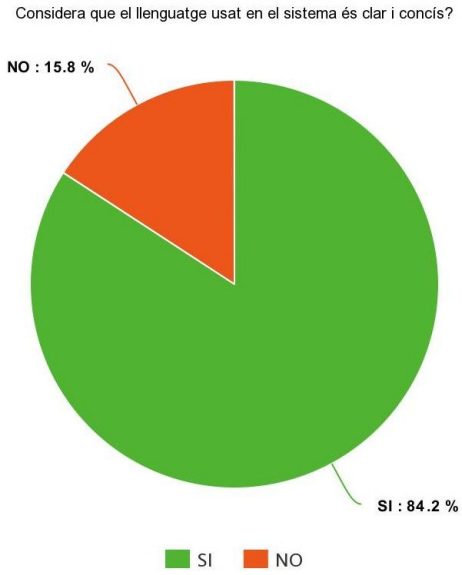


Figura 34. Requisit no funcional #3

Com es pot veure els tres primers requisits no funcionals compleixen amb el criteri d'acceptació el qual diu que s'ha d'obtenir com a mínim un 75% de valoracions positives de cada un d'ells.

Pel que fa al quart requisit no funcional també es compleix, ja que tots els formularis comproven la integritat de les dades, és a dir, han de ser vàlides i completes, també hi haurà un control d'errors a l'hora d'enviar una consulta GraphQL, es validarà que la consulta GraphQL sigui sintàcticament i semànticament correcte.

13.Conclusions

Un cop acabat el projecte es pot veure que s'han completat els objectius principals que s'havien plantejat inicialment, ja que s'ha implementat els algoritmes que generen l'esquema i servidor GraphQL i també s'han creat les aplicacions web amb les seves respectives funcionalitats. Per tant, també s'haurà completat una etapa del procés que han proposat els integrants del projecte GENESIS.

Pel que fa al desenvolupament del projecte, m'ha aportat nous coneixements. Com pot ser el fet de treballar certs aspectes de la gestió de projectes els quals quasi no havia tractat com poden ser la gestió econòmica o la sostenibilitat.

També m'ha permès adquirir nous coneixements de la majoria de tecnologies utilitzades, ja que moltes d'elles les aprenia de zero com pot ser GraphQL, el llenguatge RDF, SPARQL, entre d'altres… El fet de no tenir experiència en aquestes tecnologies feia que em trobes amb problemes bastant sovint i això em dificultava avançar en el projecte però tot i això, gràcies als consells del director i co-director, veia que anava resolent-los a poc a poc .

13.1 Treball futur

Encara que s'hagin assolit els objectius, hi ha coses millorables en el projecte:

- Que els algoritmes puguin generar, a partir d'una ontologia que sigui variable, un esquema GraphQL i un servidor GraphQL amb tots els elements de la tecnologia GraphQL, això suposaria canviar el metamodel de les ontologies ja que hauria de tenir representats tots els elements de la tecnologia GraphQL, tal que així les ontologies puguin fer servir aquests elements.

- El disseny de les pantalles és bastant millorable, a causa del poc temps que he tingut per fer-les no s'han pogut fer més atractives.

13.2 Assoliment de les competències tècniques

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

El nostre sistema no és molt complex ni molt crític, per això es té en compte només una mica. Es pot comprovar el seu assoliment, ja que hem desenvolupat correctament el sistema.

CES1.2: Donar solució a problemes d'integració en funció de les estratègies, dels estàndards i de les tecnologies disponibles. [En profunditat]

Les diverses tecnologies usades durant el projecte , que s'han descrit a l'apartat 10, s'han hagut d'integrar de forma conjunta per tal de poder fer el sistema desenvolupat.

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

S'han avaluat diversos riscos a l'apartat "3.5 Valoració d'alternatives i pla d'acció"

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

S'han desenvolupat dues aplicacions web distribuïdes en xarxa les quals s'han anat desenvolupant durant gran part del projecte.

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

Al final de cada iteració s'ha hagut de fer proves per tal de veure que el software desenvolupat funciona correctament.

14. Bibliografia

[1] Catalunya, U. *Generation and Evolution of Smart APIs. GENESIS — UPC. Universitat Politècnica de Catalunya*. [online]. Available at: https://genesis.upc.edu/en [Accessed 21 Sep. 2017].

[2] Costal, D., Farré,C., Gómez,C., Jovanovic,P., Romero,O. & Varga, J. (2017, September). Semi-automatic Generation of Data-Intensive APIs. [online]. Available at: http://www.essi.upc.edu/~farre/preprints/sagdiapis-2017.pdf [Accessed 21 Sep. 2017].

[3] Graphql.org. *GraphQL: A query language for APIs.* [online] Available at: http://graphql.org/ [Accessed 21 Sep. 2017].

[4] Krataithong, P., Buranarach, M., Hongwarittorrn, N. & Supnithi, T. An RDF Platform for Generating Web API for Open Government Data.[online] Available at: http://ceur-ws.org/Vol-1741/jist2016pd_paper3.pdf [Accessed 23 Sep. 2017]

[5] Ed-douibi, H., Cánovas,J., Gómez, A., Tisi, M. & Cabot, J. EMF-REST: Generation of RESTful APIs from Models. Available at: http://hal.univ-nantes.fr/hal-01394402/document [Accessed 23 Sep. 2017]

[6] Som-research.uoc.edu. *EMF-REST*. [online] Available at: https://som-research.uoc.edu/tools/emf-rest/ [Accessed 21 Oct. 2017].

[7] GitHub. *GeekyTheory/Automatic-API-REST*. [online] Available at: https://github.com/GeekyTheory/Automatic-API-REST [Accessed 24 Sep. 2017].

[8] Company, T. *Sails.js | Realtime MVC Framework for Node.js*. [online] Sailsjs.com. Available at: http://sailsjs.com/ [Accessed 21 Oct. 2017].

[9] GitHub. *typicode/json-server*. [online] Available at: https://github.com/typicode/json-server [Accessed 21 Oct. 2017].

[10] DreamFactory. *DreamFactory*. [online] Available at: https://www.dreamfactory.com/ [Accessed 21 Oct. 2017].

[11] En.wikipedia.org. *Scrum (software development)*. [online] Available at: https://en.wikipedia.org/wiki/Scrum_(software_development) [Accessed 21 Oct. 2017].

[12] Limited, D. *Review of Computer Energy Consumption and Potential Savings*. [online] Dssw.co.uk. Available at: https://www.dssw.co.uk/research/computer_energy_consumption.html#_Toc29375323 [Accessed 28 Dec. 2017].

[13] Robertson, S. and Robertson, J. *Mastering the requirements process*. [online] Available at: http://www.volere.co.uk/pdf%20files/template_es.pdf [Accessed 1 Nov. 2017].

[14] Howtographql. *Building a GraphQL Server with Java Backend Tutorial*. [online] Available at: https://www.howtographql.com/graphql-java/0-introduction/ [Accessed 10 Nov. 2017].

[15] W3. *RDF syntax*. [online] Available at: https://www.w3.org/1999/02/22-rdf-syntax-ns# [Accessed 4 Nov. 2017].

[16] W3. *RDF Schema*. [online] Available at: https://www.w3.org/2000/01/rdf-schema# [Accessed 4 Nov. 2017].

[17] GitHub. *graphql/graphiql*. [online] Available at: https://github.com/graphql/graphiql/blob/master/example/index.html [Accessed 21 Nov. 2017].

[18] W3. *HTML & CSS - W3C*. [online] Available at: https://www.w3.org/standards/webdesign/htmlcss [Accessed 21 Nov. 2017].

[19] W3. *JavaScript Web APIs - W3C*. [online] Available at: https://www.w3.org/standards/webdesign/script.html [Accessed 21 Nov. 2017].

[20] Graphql. *GraphQL: A query language for APIs.*. [online] Available at: http://graphql.org/learn/schema/ [Accessed 10 Nov. 2017].

[21] Wikipedia. *Java (programming language)*. [online] Available at: https://en.wikipedia.org/wiki/Java_(programming_language) [Accessed 10 Nov. 2017].

[22]Wikipedia. *Java servlet*. [online] Available at: https://en.wikipedia.org/wiki/Java_servlet [Accessed 10 Nov. 2017].

[23] Wikipedia. *JavaServer Pages*. [online] Available at: https://en.wikipedia.org/wiki/JavaServer_Pages [Accessed 19 Des. 2017].

[24] W3. *RDF Primer*. [online] Available at: https://www.w3.org/TR/rdf-primer/ [Accessed 4 Nov. 2017].

[25] W3. *RDF Schema 1.1*. [online] Available at: https://www.w3.org/TR/rdf-schema/ [Accessed 4 Nov. 2017].

[26]Openlink. *Servidor Virtuoso*. [online] Available at: http://vos.openlinksw.com/owiki/wiki/VOS/VOSIntro [Accessed 10 Nov. 2017].

[27]W3. *SPARQL 1.1 Query Language*. [online] Available at: https://www.w3.org/TR/sparql11-query/ [Accessed 10 Nov. 2017].

[28]Openlink. *Virtuoso Jena Provider*. [online] Available at: http://vos.openlinksw.com/owiki/wiki/VOS/VirtJenaProvider [Accessed 10 Nov. 2017].

[29]Wikipedia. *Servidor HTTP Apache*. [online] Available at: https://es.wikipedia.org/wiki/Servidor_HTTP_Apache [Accessed 30 Des. 2017].

[30]GitHub. *graphql-java/graphql-java*. [online] Available at: https://github.com/graphql-java/graphql-java [Accessed 10 Nov. 2017].

[31]GitHub. *graphql-java/graphql-java-tools*. [online] Available at: https://github.com/graphql-java/graphql-java-tools [Accessed 10 Nov. 2017].

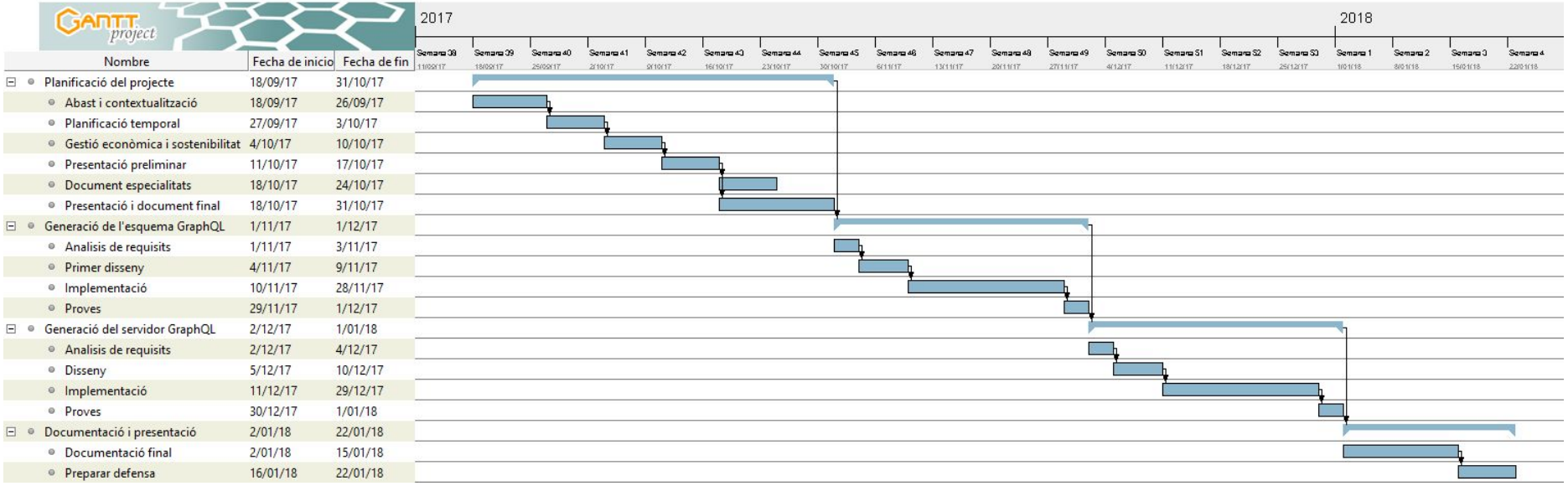
[32]GitHub. *square/javapoet*. [online] Available at: https://github.com/square/javapoet [Accessed 10 Nov. 2017].

[33]Wikipedia . *Eclipse (software)*. [online] Available at: https://en.wikipedia.org/wiki/Eclipse_(software) [Accessed 10 Nov. 2017].

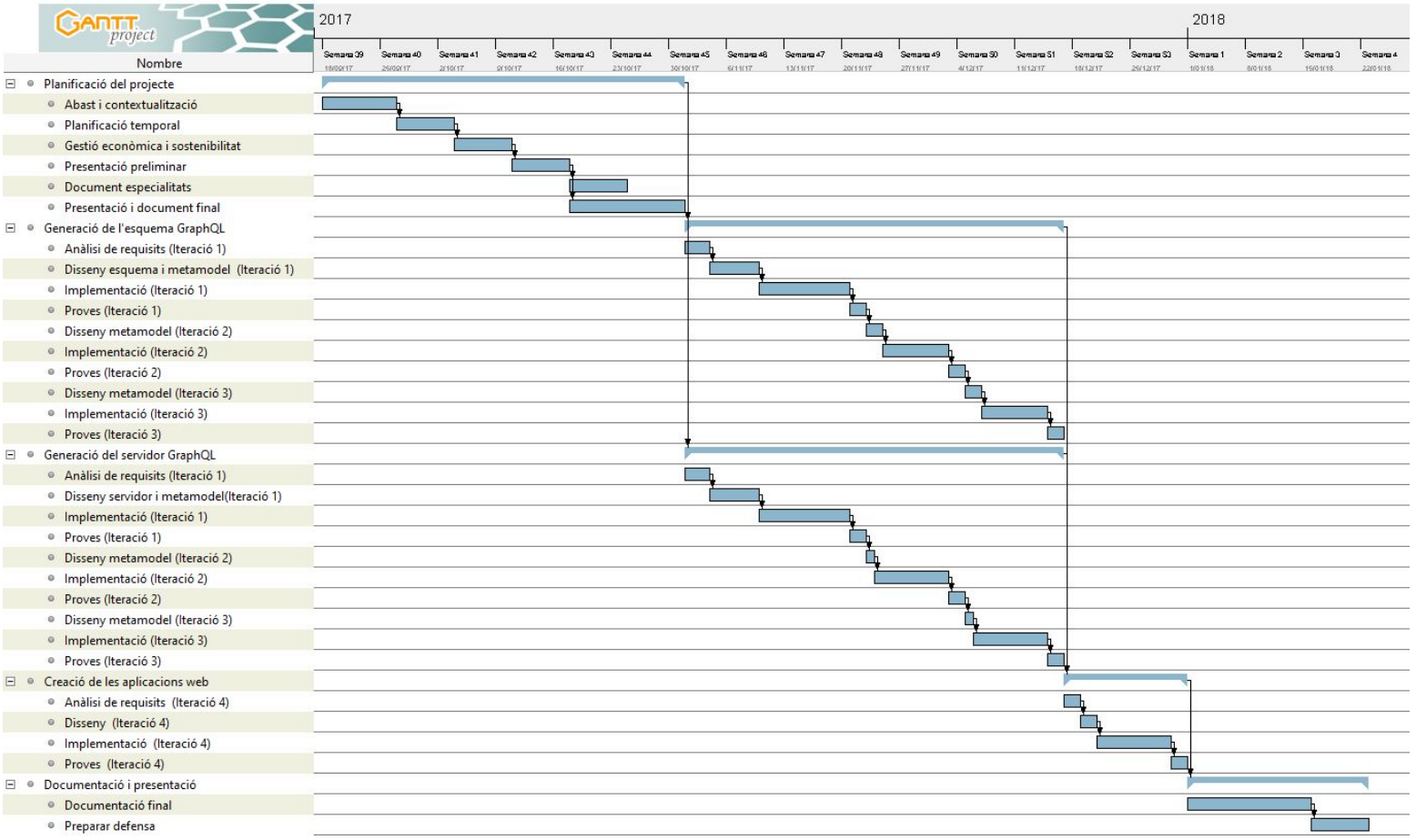
Annexos

Annex A. Diagrames de Gantt

Annex A.1 Diagrama de Gantt inicial



Annex A.2 Diagrama de Gantt final



Annex B. Esquema GraphQL

```
type Suburb {
    belongsTo : District!
    providesStop : [MetroAndBusStop]
    suburbName : String!
    idInstance : String!
}

type GeographicalCoordinate {
    latitude : Float!
    longitude : Float!
    idInstance : String!
}

type BicingStation implements IInfrastructure{
    InfrastructureType: String!
    stationBikesNumber : Int!
    nearByStation : [BicingStation]
    stationStreetNumber : Int!
    stationType : String!
    stationStatus : String!
    stationSlotsNumber : Int!
    stationID : ID!
    stationAltitude : Float!
    stationStreetName : String!
    locatedIn : GeographicalCoordinate!
    nearByInfrastructure : [IInfrastructure]
    idInstance : String!
}

type District {
    districtNestedList : String
    districtName : String!
    districtNumber : Int!
    idInstance : String!
}

interface IInfrastructure {
    InfrastructureType: String!
    idInstance : String!
    locatedIn : GeographicalCoordinate!
    nearByInfrastructure : [IInfrastructure]
}

type Infrastructure implements IInfrastructure {
    InfrastructureType: String!
    locatedIn : GeographicalCoordinate!
    nearByInfrastructure : [IInfrastructure]
    idInstance : String!
}

type Query {
    allMetroAndBusStops: [MetroAndBusStop]
    getMetroAndBusStop(id: String!): MetroAndBusStop
    allSuburbs: [Suburb]
    getSuburb(id: String!): Suburb
    allGeographicalCoordinates: [GeographicalCoordinate]
    getGeographicalCoordinate(id: String!): GeographicalCoordinate
    allBicingStations: [BicingStation]
    getBicingStation(id: String!): BicingStation
    allDistricts: [District]
    getDistrict(id: String!): District
    allInfrastructures: [IInfrastructure]
    getInfrastructure(id: String!): IInfrastructure
}
schema {
    query: Query
}
```

Annex C. Metamodel de les ontologies descrit amb llenguatge RDF

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix gql: <http://www.essi.upc.edu/~jvarga/gql/> .

###   Classes    ###

gql:Object a rdfs:Class ;

        rdfs:label "GraphQL Object Type"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:Field a rdfs:Class ;
        rdfs:subClassOf rdf:Property ;
        rdfs:domain gql:Object ;
        rdfs:label "GraphQL Field "@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:ObjectField a rdfs:Class ;
        rdfs:subClassOf gql:Field ;
        rdfs:subClassOf rdf:Property ;
        rdfs:domain gql:Object ;
        rdfs:range gql:Object ;
        rdfs:label "GraphQL Field linking to Object"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:ScalarField a rdfs:Class ;
        rdfs:subClassOf gql:Field ;
        rdfs:subClassOf rdf:Property ;
        rdfs:domain gql:Object ;
        rdfs:range gql:Scalar ;
        rdfs:label "GraphQL Field linking to Scalar"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:Modifier a rdfs:Class ;
        rdfs:label "GraphQL Modifier"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:List a rdfs:Class ;
        rdfs:subClassOf gql:Modifier ;
        rdfs:label "GraphQL List Modifier"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:NonNull a rdfs:Class ;
        rdfs:subClassOf gql:Modifier ;
        rdfs:label "GraphQL Non-null Modifier"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:Scalar a rdfs:Class ;
        rdfs:label "GraphQL Scalar"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .
```

```
###  Properties  ###

gql:hasModifier  a  rdf:Property ;
        rdfs:label "Field has modifier"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> ;
        rdfs:domain gql:Field ;
        rdfs:range gql:Modifier .

gql:combinedWith  a  rdf:Property ;
        rdfs:label "Modifier combined with another modifier"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> ;
        rdfs:domain gql:Modifier ;
        rdfs:range gql:Modifier .

###  Scalar types  ###

gql:Int a gql:Scalar ;
        rdfs:label "GraphQL Integer"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:Boolean a gql:Scalar ;
        rdfs:label "GraphQL Boolean"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:Float a gql:Scalar ;
        rdfs:label "GraphQL Float"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:ID a gql:Scalar ;
        rdfs:label "GraphQL ID"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .

gql:String a gql:Scalar ;
        rdfs:label "GraphQL String"@en ;
        rdfs:isDefinedBy <http://www.essi.upc.edu/~jvarga/gql> .
```

Annex D. Ontologia d'exemple descrita amb llenguatge RDF

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix gql: <http://www.essi.upc.edu/~jvarga/gql/> .
@prefix ex: <http://www.example.com/> .

###   Classes    ###

ex:GeographicalCoordinate  a  gql:Object .

ex:Infrastructure  a  gql:Object .


ex:MetroAndBusStop  a  gql:Object ;
        rdfs:subClassOf ex:Infrastructure .

ex:BicingStation  a  gql:Object ;
        rdfs:subClassOf ex:Infrastructure .

ex:Suburb  a  gql:Object .

ex:District  a  gql:Object .


###  Properties  ###

ex:longitude  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:GeographicalCoordinate ;
        rdfs:range gql:Float .

ex:latitude  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:GeographicalCoordinate ;
        rdfs:range gql:Float .

ex:locatedIn  a  rdf:Property, gql:ObjectField ;
        rdfs:domain ex:Infrastructure ;
        rdfs:range ex:GeographicalCoordinate .

ex:nearByInfrastructure  a  rdf:Property, gql:ObjectField ;
        rdfs:domain ex:Infrastructure ;
        rdfs:range ex:Infrastructure .

#stop properties
ex:stopName  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:MetroAndBusStop ;
        rdfs:range gql:String .

ex:stopAddress  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:MetroAndBusStop ;
        rdfs:range gql:String .

ex:stopPhone  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:MetroAndBusStop ;
        rdfs:range gql:Int .

#station properties
ex:stationID  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range gql:ID .

ex:stationType  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range gql:String .

ex:stationStreetName  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range gql:String .

ex:stationStatus  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range gql:String .

ex:stationStreetNumber  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range gql:Int .

ex:stationSlotsNumber  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range gql:Int .

ex:stationBikesNumber  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range gql:Int .

ex:stationAltitude  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range gql:Float .

ex:nearByStation  a  rdf:Property, gql:ObjectField ;
        rdfs:domain ex:BicingStation ;
        rdfs:range ex:BicingStation .

#suburb properties
ex:providesStop  a  rdf:Property, gql:ObjectField ;
        rdfs:domain ex:Suburb ;
        rdfs:range ex:MetroAndBusStop .

ex:belongsTo  a  rdf:Property, gql:ObjectField ;
        rdfs:domain ex:Suburb ;
        rdfs:range ex:District .

ex:suburbName  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:Suburb ;
        rdfs:range gql:String .
```

```
#district properties
ex:districtName  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:District ;
        rdfs:range gql:String .

ex:districtNumber  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:District ;
        rdfs:range gql:Int .

ex:districtNestedList  a  rdf:Property, gql:ScalarField ;
        rdfs:domain ex:District ;
        rdfs:range gql:String .

## Modifiers ##

ex:districtName  gql:hasModifier ex:nn2 .
ex:nn2 a gql:NonNull .

ex:districtNumber  gql:hasModifier ex:nn3 .
ex:nn3 a gql:NonNull .

ex:providesStop  gql:hasModifier ex:l2 .
ex:l2 a gql:List .

ex:belongsTo  gql:hasModifier ex:nn4 .
ex:nn4 a gql:NonNull .

ex:suburbName gql:hasModifier ex:nn5 .
ex:nn5 a gql:NonNull .

ex:stationID  gql:hasModifier ex:nn6 .
ex:nn6 a gql:NonNull .

ex:stationType gql:hasModifier ex:nn7 .
ex:nn7 a gql:NonNull .

ex:stationStreetName gql:hasModifier ex:nn8 .
ex:nn8 a gql:NonNull .

ex:stationStatus gql:hasModifier ex:nn9 .
ex:nn9 a gql:NonNull .

ex:stationStreetNumber gql:hasModifier ex:nn10 .
ex:nn10 a gql:NonNull .

ex:stationSlotsNumber gql:hasModifier ex:nn11 .
ex:nn11 a gql:NonNull .

ex:stationBikesNumber gql:hasModifier ex:nn12 .
ex:nn12 a gql:NonNull .

ex:stationAltitude gql:hasModifier ex:nn13 .
ex:nn13 a gql:NonNull .

ex:nearByStation gql:hasModifier ex:l4 .
ex:l4 a gql:List .

ex:stopName gql:hasModifier ex:nn14 .
ex:nn14 a gql:NonNull .

ex:longitude gql:hasModifier ex:nn15 .
ex:nn15 a gql:NonNull .

ex:latitude gql:hasModifier ex:nn16 .
ex:nn16 a gql:NonNull .

ex:locatedIn gql:hasModifier ex:nn17 .
ex:nn17 a gql:NonNull .

ex:nearByInfrastructure gql:hasModifier ex:l5 .
ex:l5 a gql:List .
```

Annex E. Traducció dels conceptes i relacions d'una ontologia a l'esquema i servidor GraphQL.

Concepte i relacions de l'exemple (RDF)	Esquema GraphQL (Llenguatge GraphQL)	Servidor GraphQL (Java)
ex:Infrastructure a gql:Class	Type Infrastructure	Public class Infrastructure
ex:MetroAndBusStop a gql:Class	Type MetroAndBusStop	Public class MetroAndBusStop

ex:nearByInfrastructure a rdf:Property, gql:ObjectField ;
rdfs:domain ex:Infrastructure ;
rdfs:range gql:Infrastructure .
 	
ex:NonNull a gql:NonNull .
ex:List a gql:List
ex:districtName gql:hasModifier ex:List .
ex:List gql:combinedWith ex:NonNull .
	Type Infrastructure {
	nearByInfrastructure : [Infrastructure]!
	}
		Public class Infrastructure{
		Public ArrayList<Infrastructure>nearByInfrastructure(){
		...
		}
		}

ex:stopPhone a rdf:Property, gql:ScalarField ;
rdfs:domain ex:MetroAndBusStop ;
rdfs:range gql:Int .
 	
ex:NonNull a gql:NonNull .
ex:districtName gql:hasModifier ex:NonNull .
	Type MetroAndBusStop {
	stopPhone : Int!
	}
		Public class MetroAndBusStop{
		Public Integer stopPhone(){
		...
		}
		}

ex:MetroAndBusStop rdfs:subClassOf ex:Infrastructure	Interface IInfrastructure {
	nearByInfrastructure : [Infrastructure]!
	}
	
	Type MetroAndBusStop implements IInfrastructure {
	stopPhone : Int!
		Public interface IInfrastructure {
		ArrayList<Infrastructure> nearByInfrastructure;
		}
		
		Public class MetroAndBusStop implements IInfrastructure{
		Public Integer stopPhone(){

	nearByInfrastructure : [Infrastructure]!
	}
		...
		}
		Public ArrayList<Infrastructure>nearByInfrastructure(){
		...
		}
		}