Implementació d'un tauler de dades web de la Ciutat de Barcelona

Jose Luis Dolz Fernández
Grau en Enginyeria Informàtica
Business Intelligence

Humberto Andrés Sanz
Atanasi Daradoumis Haralabus

15 de juny de 2016

FITXA DEL TREBALL FINAL

Títol del treball:	*Implementació d'un tauler de dades web de la Ciutat de Barcelona*
Nom de l'autor:	*Jose Luis Dolz Fernández*
Nom del consultor/a:	*Humberto Andrés Sanz*
Nom del PRA:	*Atanasi Daradoumis Haralabus*
Data de lliurament:	*06/2016*
Titulació o programa:	*Grau en Enginyeria Informàtica*
Àrea del Treball Final:	*Business Intelligence*
Idioma del treball:	*Català*
Paraules clau	*dashboard, Barcelona, web*

Resum del Treball (màxim 250 paraules):

En aquesta memòria es descriu la implementació d'un tauler de dades com una aplicació web amb informació sobre la ciutat de Barcelona. L'objectiu és aconseguir una eina amb una interfície gràfica agradable, senzilla i intuïtiva que pugui fer servir un ventall ampli d'usuaris.

Per portar a terme aquest projecte s'ha fet servir el repositori obert de dades que l'ajuntament de la ciutat. Per tant, s'ha estudiat el format de les dades, els seus atributs i les seves relacions abans de seleccionar les més idònies per l'aplicació. Aquestes dades es troben separades en diferents arxius i diferents formats dins del repositori. En conseqüència, s'ha tingut que crear una base de dades on emmagatzemar cronològicament les mateixes dades de manera que es pugui conservar la seva evolució històrica per un posterior anàlisi.

En el cas concret d'aquesta aplicació, el desenvolupament es divideix en tres capes: una per la base de dades; una altra capa pel servidor, on es crea l'API

per servir les dades de la BD així com les pàgines web; i, finalment, una capa de client on es troben les vistes per l'usuari amb gràfics vectorials i els seus controladors.

D'aquesta manera, s'aconsegueix un tauler de dades interactiu, que s'actualitza de forma dinàmica i que serveix com a extracció de coneixement sobre l'estat de diferents àrees de la ciutat. Així mateix, l'aplicació queda oberta per la seva millora i ampliació amb l'aportació de nous components, noves dades, millor disseny, etc.

Abstract (in English, 250 words or less):

This report describes the implementation of a dashboard as a web application with information about the city of Barcelona. The main objective is to achieve a nice and simple intuitive solution that can be used by a wide range of users.

In order to accomplish this project, the city hall open data repository has been used. Therefore, a deep research of format, attributes and relationships of the data has been made before selecting the most suitable one for the application. These data are separated in different files and formats within the repository. Accordingly, a database has been created to store the same data chronologically so the historic evolution can be preserved for further analysis.

In the case of this application, the development has been divided in three layers: the database layer; the server layer, where API is created to serve the information from the database as well as web pages; and finally, a client layer with user views, with vector graphics and controllers.

This way, an interactive panel data has been obtained, which is dynamically updated and serves as knowledge extraction about the status of the different areas in the city. Also, the application remains opened for its improvement and expansion with new components, new data, better design, etc.

Índex

Agraïments 1
1 Introducció 2
1.1 Context i justificació del Treball 2
1.2 Objectius del Treball 4
1.3 Enfocament i mètode seguit 6
1.4 Planificació del Treball 11
1.5 Breu sumari de productes obtinguts 13
1.6 Breu descripció dels altres capítols de la memòria 14
2 Implementació base de l'aplicació 16
2.1 Dades: selecció i tractament 16
2.2 Base de dades 17
2.3 Servidor 18
2.4 Aplicació client 21
2.4.1 Bootstrap 21
2.4.2 Ember 23
2.4.3 Mapes amb D3 28
2.4.4 Gràfics amb D3 33
2.4.5 Tractament de les dades amb Crossfilter 35
2.4.6 Estils 39
2.5 Últims detalls i correccions 40
3 Implementació de noves característiques 43
3.1 Biblioteques 43
3.1.1 Base de dades 45
3.1.2 Servidor 46
3.1.3 Aplicació client 47
3.2 Nivell acadèmic 50
3.2.1 Base de dades 52
3.2.2 Servidor 54
3.2.3 Aplicació client 55
3.3 Dades creuades 59
3.3.1 Base de dades 59

3.3.2 Servidor 60
3.3.3 Aplicació client 60
4 Conclusions 64
5 Glossari 66
6 Bibliografia 69
7 Annexos 71
Annex I: Planificació del projecte 71
Annex II: Instal·lació de tecnologies de treball 72
Annex III: Scripts SQL 75
Creació taula poblacio 75
Creació taula biblioteques 78
Creació taula academic 78

Llista de figures

Figura 1. Estat del tauler Kanban a Trello durant la redacció d'aquesta memòria 7
Figura 2. Divisió per capes de les tecnologies del projecte i les seves relacions 10
Figura 3. Temporalització del projecte 12
Figura 4. Opcions de descarrega de dades 16
Figura 5. Dades de la taula *poblacio* 18
Figura 6. Resposta de l'API al cridar */api/v1/population* 21
Figura 7. Exemple de distribució dels elements d'una pàgina segons el dispositiu 22
Figura 8. Adaptació de la barra de navegació a un telèfon mòbil: 23
Figura 9. Exemple d'Ember mostrant una propietat del controlador 24
Figura 10. Interconnexió dels diferents mòduls per implementar la vista de població 25
Figura 11. Compilació de l'aplicació Ember.js i avisos de JSHint 27
Figura 12. Pantalla final de benvinguda a l'aplicació 28
Figura 13. Canvi del mapa al polsar el botó de canvi de vista 32
Figura 14. Diagrama de seccions segons sexe (sense seleccionar i amb selecció) 34
Figura 15. Gràfic de l'evolució de la població per anys 34
Figura 16. Gràfic de la població dividida per edats amb un rang seleccionat 35
Figura 17. Comparació entre codi CSS compilat i l'escrit per a SaSS 39
Figura 18. Versió final de la vista de població amb dades del padró 41
Figura 19. Disseny final de la vista d'informació. 42
Figura 20. Opcions d'importació d'un arxiu de full càlcul a LibreOffice 44
Figura 21. Dades de les visites a biblioteques a l'any 2014 45
Figura 22. Mostra dels primers registres de la taula *biblioteques* 46
Figura 23. Menú desplegable per les vistes de biblioteques 47
Figura 24. Vista de *Dades generals* de l'apartat *Biblioteques* 49
Figura 25. Vista de *Visites* a l'apartat *Biblioteques* 50
Figura 26. Dades de *Nivell acadèmic* sense tractar de l'any 2009 52
Figura 27. Primeres files de la taula *academic* a PostgreSQL 53
Figura 28. Component de gràfic de barres inicialitzat i amb selecció 56
Figura 29. Finestra d'ajuda a la llegenda del gràfic 58
Figura 30. Vista de *Nivell acadèmic* amb filtres seleccionats 58
Figura 31. Vista de nombre d'habitants per biblioteques 61
Figura 32. Vista de les visites a les biblioteques per habitant 62
Figura 33. Vista dels préstecs a les biblioteques per habitant 62
Figura 34. Vista de les visites per cada préstec a les biblioteques 63
Figura 35. Execució d'un *script* SQL a pgAdmin 72
Figura 36. Opcions per importar un arxiu CSV a una taula de PostgreSQL 73
Figura 37. Vista del codi a Sublime Text 74

Agraïments

A tots els familiars, amics, companys, coneguts, professors, mentors i un llarg etcètera que heu format part del camí que m'ha portat fins aquí.

A Humberto, consultor de l'assignatura, per resoldre tan ràpidament tots els meus dubtes.

Als companys de h4ckademy, per actualitzar el meu software. Especialment a Israel Gutiérrez per la seva meravellosa iniciativa.

A Agiledevers, per la seva paciència a l'hora de treballar i d'instruir-me com a desenvolupador front-end amb JavaScript, Crossfilter i D3.

Al meu germà, que sempre està present. Gràcies per aguantar-me la quantitat de converses sobre música, cinema, teatre, informàtica, etc.

A Beatriz, la meva parella i la millor amiga que m'ha donat la vida. Gràcies per sofrir gairebé tant com jo aquesta carrera i haver cuidat de mi quan no em quedava temps ni per respirar.

Per últim, però no menys importants, als meus pares. Gràcies per recolzar-me en totes les decisions que he pres a la vida, per estranyes que semblessin. Moltes gràcies per donar-m'ho tot i, sobretot, per donar-me la llibertat d'escollir el meu camí i la persona que sóc.

1 Introducció

Aquest Treball Final de Grau (TFG), en l'especialitat de *Business Intelligence* (BI), tractarà sobre la implementació d'un tauler de dades – *data dashboard* en anglès – en web per la ciutat de Barcelona. En aquest tauler es podran visualitzar les dades que l'ajuntament de la Ciutat Comtal posa a disposició del públic[1] d'una forma estructurada i fàcilment llegible. D'aquesta manera, les diferents dades disponibles en diferents arxius repartits en el repositori es podran veure de forma conjunta, històrica i es tindrà la possibilitat de creuar-la.

L'opció d'implementar aquesta aplicació en un format per navegadors d'Internet no és casual: per una part, l'augment de consum d'informació es deu a la proliferació dels dispositius mòbils i la seva ubiqüitat; per altra banda, el fet de crear el tauler com una pàgina web amb disseny responsiu dona la possibilitat de poder accedir a qualsevol dispositiu amb connexió a la Xarxa.

El treball a realitzar va més enllà de crear una vista per l'usuari on es mostrin les dades, el que en anglès es coneix com a *Front-end*. També es desenvoluparà la part del servidor i la seva connexió amb una base de dades (BD), conegut com a *Back-end* en anglès, que serviran les dades al client.

1.1 Context i justificació del Treball

A l'any 2011, IBM va publicar un informe on estimava que la humanitat generava 2'5 exabytes al dia [1]. Per fer una aproximació a aquesta quantitat, segons la Viquipèdia en espanyol, serien necessaris 0'5 Petabytes per poder gravar en alta definició la vida sencera d'una persona que visqui 100 anys [2].

[1] El repositori de dades de Barcelona es pot trobar en la següent direcció: http://opendata.bcn.cat/opendata/ca

Si 1024 Petabytes corresponen a 1 Exabyte, vol dir que cada dia es generaven l'equivalent a tenir en vídeo d'alta definició la vida de 5120 persones!

Tot i que més de la meitat d'aquesta informació és en format de vídeo i àudio, encara resta un bon munt de dades que es poden utilitzar per determinar el comportament de les persones. Aquest gran volum d'informació, comunament conegut com **Big Data**, està prenent gran rellevància a tot arreu, sobretot a les empreses privades. Aquestes volen millorar les experiències dels seus clients tractant de fer-les més personalitzades i augmentar així el seu volum de facturació. Tal és la seva importància que, en un article per la revista Hardvard Business de l'any 2012, Davenport i Patil definien el treball de científic de dades com "el treball més sexy del segle 21" [3].

Com es pot deduir, Barcelona no és aliena a aquesta creació de grans volums de dades. Certament, els milions de persones que viuen a la ciutat i en la seva àrea metropolitana són grans generadors de dades, als que s'hi poden sumar els milions de turistes que la visiten cada any. I no només les persones sinó també les empreses i tots els serveis amb que compta: autobusos, línies de metro, de ferrocarrils, estacions meteorològiques, institucions públiques, etc. Molta d'aquesta informació ja es pot obtenir al repositori de dades de l'ajuntament en diferents formats. Malgrat les diferents opcions ofertes, gairebé tota aquesta informació es troba en format de taula de dades o full de càlcul, moltes vegades separada segons la seva data d'actualització. Com a conseqüència d'aquesta divisió i formats, si un usuari volgués veure, per exemple, l'evolució demogràfica del districte de Nou Barris, hauria de descarregar els diferents arxius corresponents segons les dates del seu interès i creuar les dades ell mateix. En el cas de que volgués veure tendències i obtenir gràfics, també hauria de fer les seves pròpies creacions.

Si s'extrapola aquest problema a una institució com l'ajuntament de Barcelona, la càrrega de treball per obtenir dades de qualitat per a la presa de decisions pot créixer exponencialment. Per exemple, les passes a donar serien:

- Cercar les dades idònies que es vulguin creuar a la pàgina web del repositori .

- Descarregar, per cada data, els diferents fitxers segons la data de l'actualització.
- Obrir cadascun dels fitxers i creuar les dades corresponents amb ordre cronològic
- Si s'escau, crear els gràfics a partir d'aquestes dades.

En el cas exposat, no s'ha tingut en compte que les dades poden estar actualitzant-se en el moment de realitzar la tasca. Per tant, podria donar-se que, un cop enllestit l'informe per poder prendre decisions, les dades ja estiguin obsoletes. Un tauler de dades connectat a una BD permet aprofitar aquesta immediatesa de la informació, pot creuar-la d'una forma més senzilla i mostrar gràfics actualitzats per recolzar als usuaris en la presa de decisions.

Segurament, l'ajuntament disposa de un nombre suficient de treballadors en les diferents regidories i comissions que accedeixen a dades molt específiques. Tot i amb això, el creixement continu dels volums de dades farà que, inevitablement, arribi un dia que manejar tota aquesta informació sigui gairebé impossible amb fulls de càlcul, ja sigui per la mida de les dades o per la seva continua actualització. Consegüentment, el salt d'aquest repositori de fitxers cap a un tauler de dades –accessible tant per l'ajuntament com per al públic en general- serà una necessitat inevitable.

1.2 Objectius del Treball

El principal objectiu d'aquest TFG és implementar un tauler de dades o *dashboard*, mitjançant una aplicació web del tipus client-servidor. El tauler mostrarà les dades gràficament perquè siguin entenedores per a la majoria d'usuaris i de manera que els ajudi a extreure coneixement. Així, permetrà entreveure d'una forma més eficient els punts forts i febles de la ciutat.

Una de les principals tasques del treball es traslladar el major nombre de fitxers que es troben al repositori a una base de dades. Això permetrà a l'aplicació accedir a qualsevol dada immediatament d'una forma eficient i

seleccionar només la informació necessària que es vol mostrar. A més a més, aquesta tasca portarà directament a un altre objectiu: demostrar la idoneïtat de tenir les dades en una BD accessible per a tothom, fent servir, per exemple, una API REST o JSON. D'aquesta manera, qualsevol usuari podria rebre la informació que l'interessi mitjançant una senzilla direcció web, també coneguda com a URL (de l'anglès *Uniform Resource Locator*).

A continuació, una altra tasca serà dur a terme la visualització de les dades servides mitjançant gràfics adients: de barres, de línies, circulars, etc. Cal fer esment específic a l'exposició de les dades sobre un mapa. Dit d'una altra manera, es mostrarà la informació desitjada sobre el plànol de Barcelona classificada segons els diferents districtes de la ciutat i, si s'escau, dels diferents barris que la formen.

Es vol fer especial èmfasi a la part visual del projecte. D'entrada, es tractarà d'oferir una bona experiència d'usuari de manera que explorar i navegar per les diferents pàgines sigui el més intuïtiu possible. En darrer lloc, i si el temps ho permet, l'aplicació serà *Responsive*, de manera que es pugui adaptar als navegadors dels dispositius que els diferents usuaris facin servir. Això vol dir adaptar el disseny per a les pantalles dels telèfons mòbils, tauletes tàctils (comunament conegudes com *tablets*) i monitors.

Per últim, però no menys important, es tractarà de pujar aquesta aplicació a un domini d'Internet perquè sigui accessible per a tothom. Tanmateix, s'oferirà el resultat al servei tècnic de l'ajuntament de Barcelona, sempre que s'arribi a un mínim de qualitat i amb el vist i plau dels consultors i professors de l'assignatura.

Finalment, l'autor vol destacar que aquest TFG serà també una oportunitat per adquirir nous coneixements com si es tractés d'una nova assignatura, a banda de demostrar els que s'han assolit durant el transcurs del Grau.

1.3 Enfocament i mètode seguit

L'enfocament d'aquest treball és apropar-se a la gestió de qualsevol projecte d'enginyeria del programari, però emprant una metodologia més àgil. Dit d'una altra manera, no hi haurà tanta necessitat de fer una documentació formal feixuga –com ara els diagrames de components i classes en UML o de les restriccions en llenguatge OCL– a causa de la quantitat de mòduls, llibreries i marcs de treballs (*framework* en anglès) que es poden emprar en una aplicació web.

Tots els programes, llenguatges, llibreries i marcs de treball –que s'expliquen amb més detall més endavant– que es faran servir són de codi obert (*Open source*). És a dir, és programari que es distribueix i es desenvolupa lliurament sense cap tipus de llicència. Per tant, per raons òbvies, l'aplicació resultant d'aquest TFG serà també de codi obert i serà oferta a la comunitat.

El primer pas, com s'ha esmentat anteriorment, serà la **selecció de les dades d'interès** del repositori de l'ajuntament de Barcelona. Per raons de temps i agilitat, s'escolliran només aquelles dades d'actualització anual. En primer lloc es seleccionaran dades d'un caire més obvi, com la informació demogràfica, i s'avançarà, si el temps ho permet, cap a dades més detallades però igualment útils com, per exemple, nombre de guarderies per barri, comissaries, etc.

El següent pas serà construir un sistema **servidor** que es comuniqui amb la base de dades i ofereixi les dades en el format idoni –JSON en el cas d'aquest treball- als clients dels usuaris.

Per tant, l'últim pas serà implementar la **vista** dels usuaris. Cada vista serà una pàgina web amb el seu propi controlador que li donarà les dades i les actualitzarà en funció del comportament i les preferències de l'usuari. A tall d'exemple, si l'usuari posa el cursor o clica sobre un districte o barri de la ciutat, és mostrarà informació detallada d'aquest i/o s'actualitzarà la resta de gràfics mostrats per donar només les dades referides a aquesta secció.

Amb relació a la priorització del projecte, la primera fita a assolir serà aconseguir una versió simplificada i del tot funcional de l'aplicació. En altres paraules, el primer que s'haurà d'aconseguir és una versió on es mostrin unes primeres dades senzilles –cens per districtes, per exemple– en una versió per monitor d'ordinador on es comuniquin totes les capes: BD, servidor i client. Una vegada obtinguda aquesta primera aplicació, s'afegiran noves dades i les seves corresponents vistes.

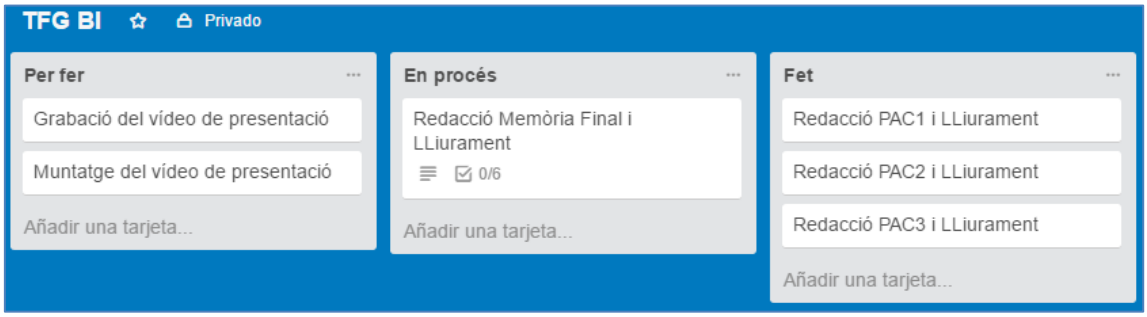


Figura 1. Estat del tauler Kanban a Trello durant la redacció d'aquesta memòria

El mètode de treball que s'emprarà serà àgil. Tot i que aquest TFG és un projecte individual, s'enfocarà com si es tractés d'un projecte creat per un equip professional que pugui treballar en remot, és a dir, des de diferents parts del planeta. Per una banda, s'utilitzarà un **tauler Kanban** [4] per organitzar la càrrega de treball i les diferents funcionalitats que s'han d'implementar. En el nostre cas es dividiran les targetes en tres columnes: Per fer, En procés i Fet, com es pot veure a la Figura 1. L'aplicació escollida per controlar aquest tauler és **Trello**[2]. Per altre costat, i amb la mateixa filosofia de treball en equip, s'utilitzarà un repositori web de versions de programari. Tenint en compte que aquest projecte és de codi obert, l'aplicació escollida és la més famosa d'aquest camp: **GitHub**[3].

A continuació es dona el llistat de les tecnologies –totes de codi obert– que presumiblement seran utilitzades per portar a terme aquest projecte i que es representen, posteriorment, a la il·lustració 1:

[2] La pàgina web de l'aplicació es pot visitar a https://trello.com/

[3] La pàgina principal de Github es pot trobar a https://github.com/

- **PostgeSQL**: Base de de dades relacional de llenguatge SQL (de l'anglès *Structured Query Language*) que servirà per emmagatzemar les dades del repositori obert de l'ajuntament de Barcelona. Es pot obtenir més informació a la direcció http://www.postgresql.org/

- **JavaScript (JS)**: llenguatge de programació interpretat, com a part dels navegadors web. És orientat a objectes, basat en prototipus, dinàmic i de tipatge dèbil. Per obtenir més informació es pot visitar el següent enllaç: https://developer.mozilla.org/es/docs/Web/JavaScript

- **Node.js**: creat per la Node.js Foundation, és un entorn d'execució en llenguatge JavaScript d'entrada i sortida, dirigit per esdeveniments i asíncron, que s'utilitza en el costat del servidor. Fa servir el motor V8 desenvolupat per Google, cosa que li permet executar codi en el servidor a grans velocitats. Més informació en https://nodejs.org/

- **Express.js**: creat també per la Node.js Foundation, és un marc de treball per a Node.js, que permet crear aplicacions web d'una o més pàgines. Aporta un senzill enrutador i funcions *middleware* (capa intermèdia de programari que assisteix al client, en el nostre cas, per comunicar-se amb el servidor). Més informació: http://expressjs.com/es/.

- **Massive.js**: creat per Rob Conery, és una petita llibreria que serveix d'ajuda per la connexió del servidor amb la base de dades. Més informació: https://github.com/robconery/massive-js

- **HTML**: acrònim de l'anglès *HyperText Markup Language*. Llenguatge de marcat per estructurar textos, famós per ser la base de les pàgines web a Internet.
  Més informació: https://developer.mozilla.org/es/docs/Web/HTML.

- **CSS** i **SaSS**: CSS correspon a les segles en anglès de *Cascading Style Sheets*. És un llenguatge per definir l'estil de les pàgines escrites en HTML. SaSS és l'acrònim anglès de *Syntactically Awesome Style Sheets* i va ser creat per Hampton Catlin, Natalie Weizenbaum, Chris Eppstein i altres. És una extensió de CSS que permet l'ús de variables i herència.

Fa possible la modularització en diferents fitxers i, per tant, augmenta la senzillesa del programari. El seu intèrpret tradueix el codi en un únic fitxer CSS. Més informació:
CSS: https://developer.mozilla.org/es/docs/Web/CSS
SaSS: http://sass-lang.com/

- **jQuery**: creada inicialment per John Resig, llibreria de JavaScript que simplifica la interacció amb els elements dels documents HTML així com manejar esdeveniments i afegir animacions. Més informació: https://jquery.com/

- **Bootstrap**: o Twitter Bootstrap, creada per Mark Otto i Jacob Thornton per Twitter, és un marc de treball de codi obert que conté plantilles de disseny amb formularis, fonts, botons, etc. Més informació: http://getbootstrap.com/

- **Ember.js**: creat per Tilde Inc., és un marc de treball de JavaScript per crear aplicacions web el costat del client. Permet introduir directives dins del codi HTML, com bucles i condicionals, i aplica l'arquitectura MVC (model-vista-controlador). A més, es farà servir **Ember-cli**, una interfície de la línia de comandes que facilita la creació de les diferents vistes, controladors, models, etc. Més informació:
  Ember.js: http://emberjs.com/
  Ember-cli: http://ember-cli.com/

- **Crossfilter**: creada per Square, és una llibreria JavaScript per explorar, manipular i filtrar grans conjunts de dades en un temps extremadament curt. Més informació: http://square.github.io/crossfilter/

- **D3.js**: o simplement D3 per l'acrònim anglès *Data-Driven Documents*, documents dirigits per dades. Va ser creada per Mike Bostock i és una llibreria JavaScript per crear gràfic dinàmics i interactius a partir de dades en el format vectorial escalable **SVG** (*Scalable Vector Graphics*). Més informació sobre D3 es pot trobar a https://d3js.org/ i sobre gràfics SVG https://developer.mozilla.org/es/docs/Web/SVG

A la Figura 2 es pot veure les interaccions entre les diferents capes i tecnologies:

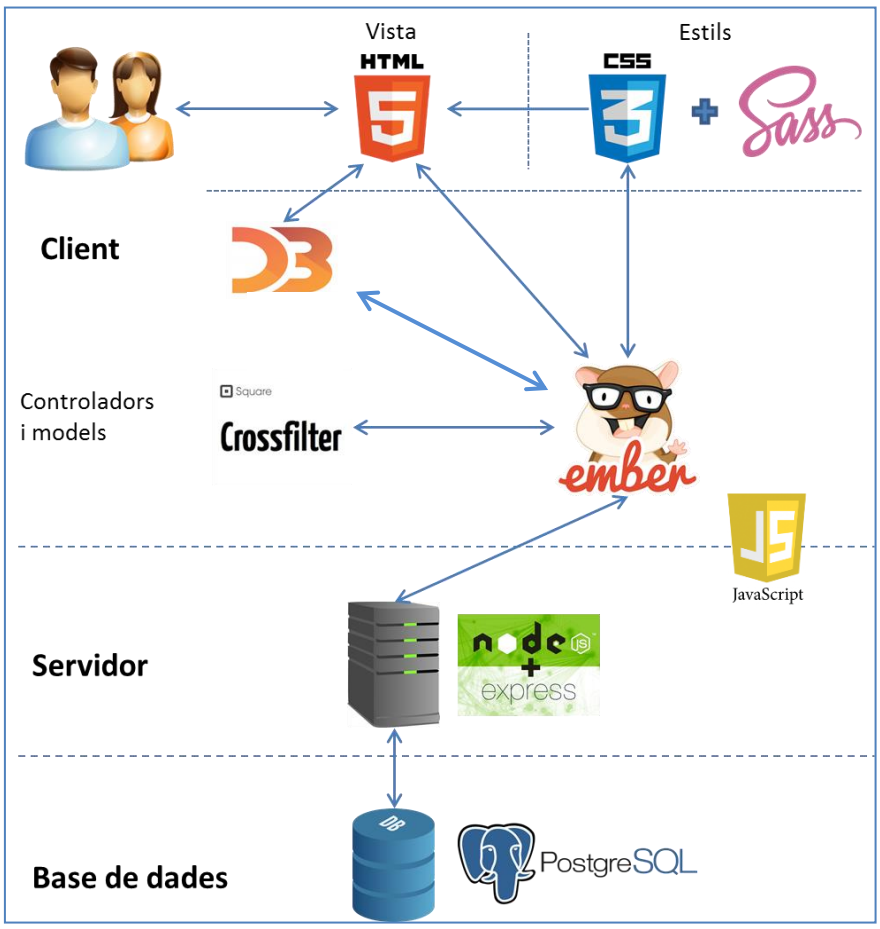


Figura 2. Divisió per capes de les tecnologies del projecte i les seves relacions

Gran part de les tecnologies esmentades en el punt anterior són familiars per l'autor. Altres suposen un nou repte d'aprenentatge doncs, com s'ha afirmat amb anterioritat, es vol que aquest projecte suposi una nova oportunitat per aprendre nous coneixements.

Amb relació al maquinari que es farà servir per implementar l'aplicació, es podria fer amb qualsevol tipus de ordinador personal. En el cas que ens ocupa, es farà servir un Apple Macbook amb OS X, per disposar de base UNIX i tenir instal·lat algunes de les tecnologies abans esmentades, i un ordinador personal amb Windows 10 per fer servir el programa **Microsoft Project 2010**[4]

[4] Per saber més sobre MS Project 2010:
http://www.microsoft.com/latam/gobierno/productos/project2010.aspx

per planificar la temporalització del projecte. Pel que fa als programes per desenvolupar el codi, s'utilitzarà la tercera versió de l'editor de text gratuït **Sublime Text**[5] que facilita enormement l'edició del codi i compte amb extensions per facilitar la visualització i la comprovació del mateix. En el cas de navegadors d'Internet, es faran proves de compatibilitat amb els més coneguts, però es treballarà sobre tot amb **Chrome**[6] de Google per constar d'eines potents per a desenvolupadors d'aplicacions web.

Per concloure amb aquest apartat, es vol recordar que totes les tecnologies a utilitzar són de codi obert i, per tant, de llicències permissives del tipus *Berkeley Software Distribution*, *Massachusetts Institute of Technology*, *General Public License*, *Creative Commons* i, pot ser la més restrictiva, *Apache Software License* [5]. En qualsevol cas, l'obligació màxima és la de reconèixer els seus autors i afegir arxius de llicència i avisos sobre la seva distribució. Per altre costat, el projecte emprarà les dades ofertes per l'ajuntament de Barcelona, sense fer cap menció a persones concretes. Per acabar, no es farà servir un sistema d'usuaris amb autenticació per la qual cosa no s'emmagatzemarà cap tipus d'informació personal de l'usuari.

1.4 Planificació del Treball

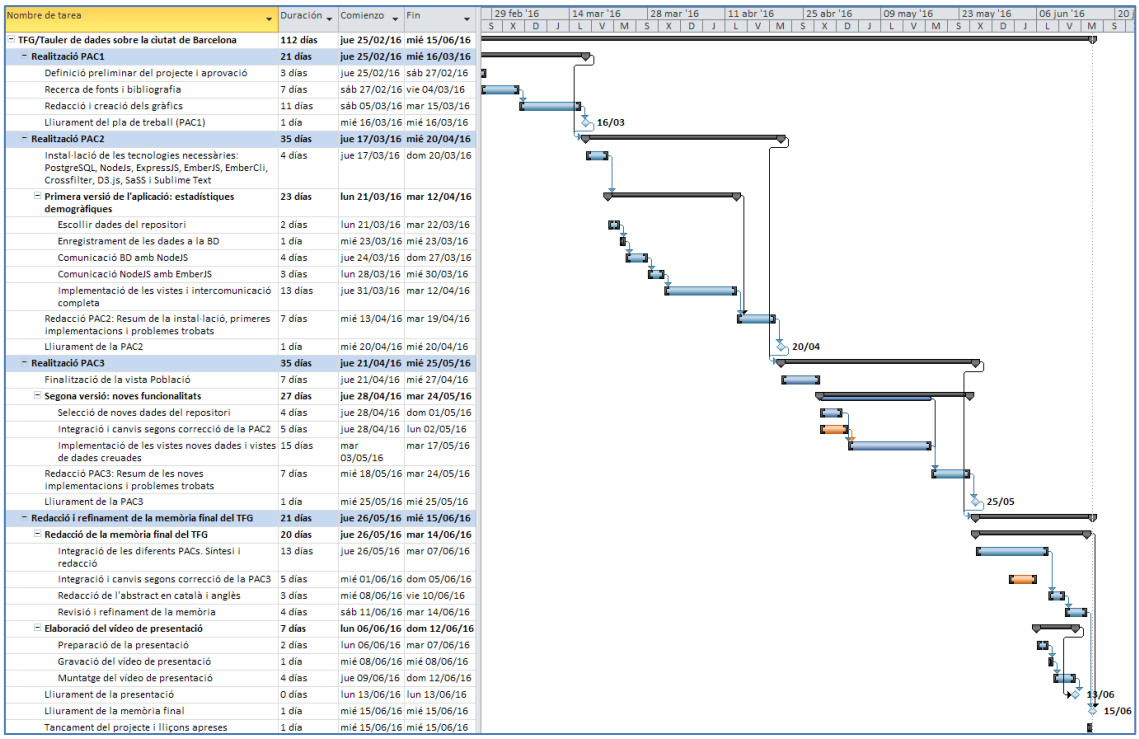
Un cop presentada la proposta del projecte es procedeix a temporalitzar les tasques i les fites per a la seva realització. Dins d'aquesta temporalització es respecta les dates de lliurament de les Proves d'Avaluació Continuada (PAC) pròpies de l'assignatura.

Per tant, i coincidint amb les entregues programades al pla docent, s'ha estructurat el TFG en quatre fases diferenciades. Aquestes fases s'han especificat en el diagrama de Gantt final[7] que és pot observar a la Figura 3.

[5] Més informació sobre Sublime Text: http://www.sublimetext.com

[6] Per obtenir més informació sobre les eines: https://developer.chrome.com/devtools

[7] Es pot trobar el mateix diagrama amb millor resolució a l'Annex I: Planificació del projecte



Nombre de tarea	Duración	Comienzo	Fin
TFG/Tauler de dades sobre la ciutat de Barcelona	**112 días**	**jue 25/02/16**	**mié 15/06/16**
Realització PAC1	**21 días**	**jue 25/02/16**	**mié 16/03/16**
Definició preliminar del projecte i aprovació	3 días	jue 25/02/16	sáb 27/02/16
Recerca de fonts i bibliografia	7 días	sáb 27/02/16	vie 04/03/16
Redacció i creació dels gràfics	11 días	sáb 05/03/16	mar 15/03/16
Lliurament del pla de treball (PAC1)	1 día	mié 16/03/16	mié 16/03/16
Realització PAC2	**35 días**	**jue 17/03/16**	**mié 20/04/16**
Instal·lació de les tecnologies necessàries: PostgreSQL, NodeJs, ExpressJS, EmberJS, EmberCli, Crossfilter, D3.js, SaSS i Sublime Text	4 días	jue 17/03/16	dom 20/03/16
Primera versió de l'aplicació: estadístiques demogràfiques	**23 días**	**lun 21/03/16**	**mar 12/04/16**
Escollir dades del repositori	2 días	lun 21/03/16	mar 22/03/16
Enregistrament de les dades a la BD	1 día	mié 23/03/16	mié 23/03/16
Comunicació BD amb NodeJS	4 días	jue 24/03/16	dom 27/03/16
Comunicació NodeJS amb EmberJS	3 días	lun 28/03/16	mié 30/03/16
Implementació de les vistes i intercomunicació completa	13 días	jue 31/03/16	mar 12/04/16
Redacció PAC2: Resum de la instal·lació, primeres implementacions i problemes trobats	7 días	mié 13/04/16	mar 19/04/16
Lliurament de la PAC2	1 día	mié 20/04/16	mié 20/04/16
Realització PAC3	**35 días**	**jue 21/04/16**	**mié 25/05/16**
Finalització de la vista Població	7 días	jue 21/04/16	mié 27/04/16
Segona versió: noves funcionalitats	**27 días**	**jue 28/04/16**	**mar 24/05/16**
Selecció de noves dades del repositori	4 días	jue 28/04/16	dom 01/05/16
Integració i canvis segons correcció de la PAC2	5 días	jue 28/04/16	lun 02/05/16
Implementació de les vistes noves dades i vistes de dades creuades	15 días	mar 03/05/16	mar 17/05/16
Redacció PAC3: Resum de les noves implementacions i problemes trobats	7 días	mié 18/05/16	mar 24/05/16
Lliurament de la PAC3	1 día	mié 25/05/16	mié 25/05/16
Redacció i refinament de la memòria final del TFG	**21 días**	**jue 26/05/16**	**mié 15/06/16**
Redacció de la memòria final del TFG	**20 días**	**jue 26/05/16**	**mar 14/06/16**
Integració de les diferents PACs. Síntesi i redacció	13 días	jue 26/05/16	mar 07/06/16
Integració i canvis segons correcció de la PAC3	5 días	mié 01/06/16	dom 05/06/16
Redacció de l'abstract en català i anglès	3 días	mié 08/06/16	vie 10/06/16
Revisió i refinament de la memòria	4 días	sáb 11/06/16	mar 14/06/16
Elaboració del vídeo de presentació	**7 días**	**lun 06/06/16**	**dom 12/06/16**
Preparació de la presentació	2 días	lun 06/06/16	mar 07/06/16
Gravació del vídeo de presentació	1 día	mié 08/06/16	mié 08/06/16
Muntatge del vídeo de presentació	4 días	jue 09/06/16	dom 12/06/16
Lliurament de la presentació	0 días	lun 13/06/16	lun 13/06/16
Lliurament de la memòria final	1 día	mié 15/06/16	mié 15/06/16
Tancament del projecte i lliçons apreses	1 día	mié 15/06/16	mié 15/06/16

Figura 3. Temporalització del projecte

Es mostra a continuació una possible estimació dels costos derivats de la implementació d'aquest projecte com si es tractés d'un producte real realitzat per un equip professional de graduats en Enginyeria informàtica.

Per un costat, aquest projecte compte amb unes 300 hores per poder realitzar-se. Es calcula, a continuació, l'estimació de les possibles despeses desglossades de la següent manera:

- Sou brut mensual: 1500€.
- Lloguer de lloc de treball a un espai de *coworking*: 220€/mes (preu de referència en el lloc de treball de l'autor). Aquest preu inclou connexió a Internet d'alta velocitat i consum elèctric.
- Cost del maquinari: amb un equip de potència mitjana es podria realitzar aquest projecte i s'estima en 1000€. Si es té compte que l'equip es renova cada 2 anys, el seu cost mensual seria, aproximadament, de 42€.

Per tant, amb la suma dels punts anteriors es tindria un cost mensual total de 1762€. Si es treballa a temps complet –això és, 40 hores setmanals–

amb 4 setmanes al mes, s'ha de dividir el cost mensual per 160 hores. El resultat és de 11'0125€, que s'arrodoneix a 12€ per hora per afegir despeses imprevistes. Aleshores, l'estimació del cost per implementar i documentar aquest projecte és de 3600 euros.

Per altra banda, s'ha de tenir en compte la inversió en un servei de *hosting* de qualitat on emmagatzemar la pàgina. A més d'oferir l'enregistrament d'un domini propi, hauria de tenir espai il·limitat per la base de dades i disposar de tràfic il·limitat. Es pren com a referència el preu mensual, sense ofertes, d'un servidor per rendiment d'una de les empreses[8] capdavanteres en aquest camp: 49'99€. Per tant, l'entitat que volgués oferir aquesta aplicació web hauria de fer una inversió afegida de 599'88 euros per any.

1.5 Breu sumari de productes obtinguts

Com a resultat del projecte, s'obté una aplicació dividida en diferents vistes que corresponen a diferents panels de dades dividits segons la seva àrea d'interès i propòsit. Cadascuna d'aquestes vistes reutilitza components gràfics comuns, de manera que l'usuari pot emprar-los de forma més intuïtiva gràcies a la repetició.

Per aconseguir aquesta aplicació s'han hagut d'implementar els següents elements:

- Una **base de dades** on s'emmagatzemen les dades històriques separades segons la seva àrea d'interès.
- Un **servidor** que serveixi les pàgines web al client i que, a més a més, sigui l'API per obtenir les dades de la BD.
- Un **servei de dades** centrals, en la part de client, accessible per a totes les vistes que ho requereixin.

[8] **1&1** (2016), Servidores gestionados para el rendimiento de su web.

- Diferents **vistes** accessibles des d'una barra de navegació. Cadascuna d'aquestes vistes té la seva **ruta** i **controlador** que serveixen com a lògica de negoci i permeten la manipulació de les dades i la comunicació amb l'API.

1.6 Breu descripció dels altres capítols de la memòria

Per poder seguir la resta de capítols d'aquesta memòria, es recomana la instal·lació de les tecnologies emprades en el desenvolupament així com una còpia del codi que es pot trobar a Github. En aquest document s'obvia el codi implementat per ser massa feixuc però es pot trobar en la següent direcció:

https://github.com/ElXaxe/Barcelona-Dashboard

Els comentaris i els noms de variables, funcions, etc. que s'utilitzen dins del codi són, en la seva majoria, en anglès. Això és així per tractar-se d'un codi que es comparteix amb la comunitat i perquè el pugui comprendre qualsevol programador. S'emprarà el català en els casos que siguin requerits per facilitat de comprensió o de correspondència entre les diferents tecnologies.

Si es desitja provar l'aplicació sense haver d'instal·lar les tecnologies necessàries i haver de compilar el codi, es pot fer si s'accedeix a la següent adreça web:

https://barcelona-dashboard.herokuapp.com/

En els següents capítols es descriuen les tasques portades a terme en les etapes planificades per complir amb els objectius proposats en aquest projecte.

L'apartat 2 d'aquesta memòria presenta tots els passos donats per aconseguir una primera versió de l'aplicatiu. Es detallaran algunes característiques de les tecnologies escollides, la metodologia que aporten i les seves exigències. Es descriuen també les diferents parts d'una aplicació web dividides en la capa de dades, la capa del servidor i la capa de client. A més, es comenten les decisions de disseny aplicades i els components creats que

seran reutilitzables en versions posteriors del programari. Aquesta primera versió servirà com a base.

En el següent apartat, s'explica les ampliacions i noves característiques implementades a partir de la base creada al punt anterior. Es comenten també les dificultats trobades per incloure noves dades del repositori i la impossibilitat d'implementar vistes més adients pels propòsits desitjats. Per altra banda, s'apunten certes modificacions necessàries respecte a l'aplicació base i la oportunitat de crear nous components.

En últim terme, es detallen les conclusions a les que s'arriben un cop finalitzat el projecte. S'aprofitarà per comentar també les línies futures de treball i les possibles millores sobre aquesta aplicació web amb vista a augmentar les seves possibilitats com a eina de Business Intelligence per l'extracció de coneixement i la presa de decisions.

2 Implementació base de l'aplicació

Desenvolupar una aplicació web completa és una tasca difícil i feixuga. Avui dia és normal que, dins del món de la programació, aquesta feina es divideixi en dos grups: els que s'encarreguen de la part del client, anomenats desenvolupadors *front-end*, i els que es fan càrrec de la part del servidor, coneguts com a desenvolupadors *back-end*. En empreses més grans, cada part es pot dividir en especialitzacions més específiques: expert en experiència d'usuari, responsable de dades, etc. Encarregar-se i desenvolupar les dos parts en que es divideix una aplicació client-servidor és el que es coneix com a desenvolupador ***full-stack***. Dit d'una altra manera, s'encarrega de programar tota la pila de capes que formen l'aplicació.

A continuació, s'explica tot el treball de desenvolupament que ha portat a terme l'autor en aquesta versió base de l'aplicatiu web.

2.1 Dades: selecció i tractament

En aquesta primera versió de l'aplicació, s'intenta tractar la població de Barcelona amb dades d'actualització anual. Si es trien aquestes opcions a la web d'OpenDataBCN de l'ajuntament de Barcelona, es pot veure que s'obtenen fins a 82 resultats diferents. En aquest cas, s'ha escollit les dades oficials del padró, amb les edats any a any, tant de la població masculina com de la població femenina.

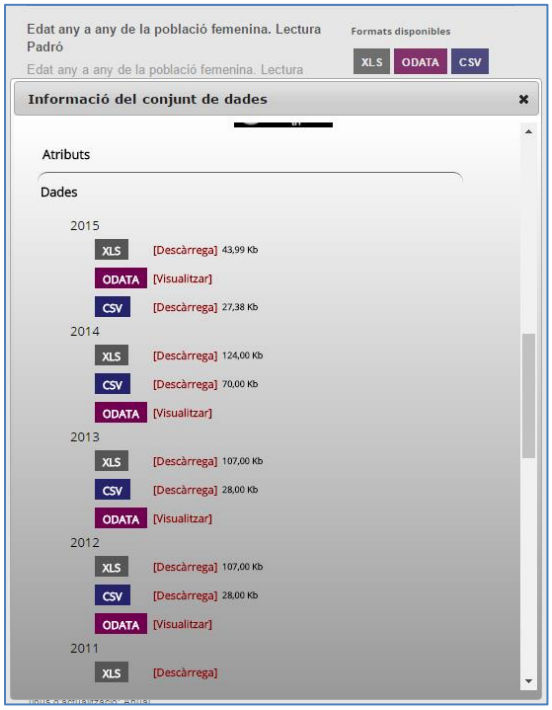


Figura 4. Opcions de descarrega de dades

Dins de cadascuna d'aquestes dues opcions es troben les dades en diferents formats, separades en fitxers diferents segons l'any, tal com es mostra a la Figura 4. Seria recomanable que aquestes

dades s'oferissin com una API (de l'anglès *Application Programming Interface*) que permetés adquirir la informació desitjada només amb una crida a una direcció web (també coneguda com URI, *Uniform Resource Identifier*, o URL, *Uniform Resource Locator*).

En el cas d'aquest TFG, es decideix descarregar tots els arxius CSV –valors separats per comes– amb les dades necessàries escollides de tots els anys disponibles per després ajuntar tota la informació en un mateix fitxer amb un programa de fulls de càlcul. Cada columna d'aquest fitxer serà un atribut de la taula població i són: any de les dades, districte, barri i una columna per edat tant per dones com per homes. Dit d'una altra manera, una columna per totes les dones amb 0 anys, una altra per les que tenen 1 any, etc., i el mateix per als homes. Resumint-ho en una línia d'especificació de bases de dades (BD) per a la taula *poblacio*:

poblacio: any, districte, barri, donesAny0, donesAny1, ..., donesAny95, homesAny0, ..., homesAny95

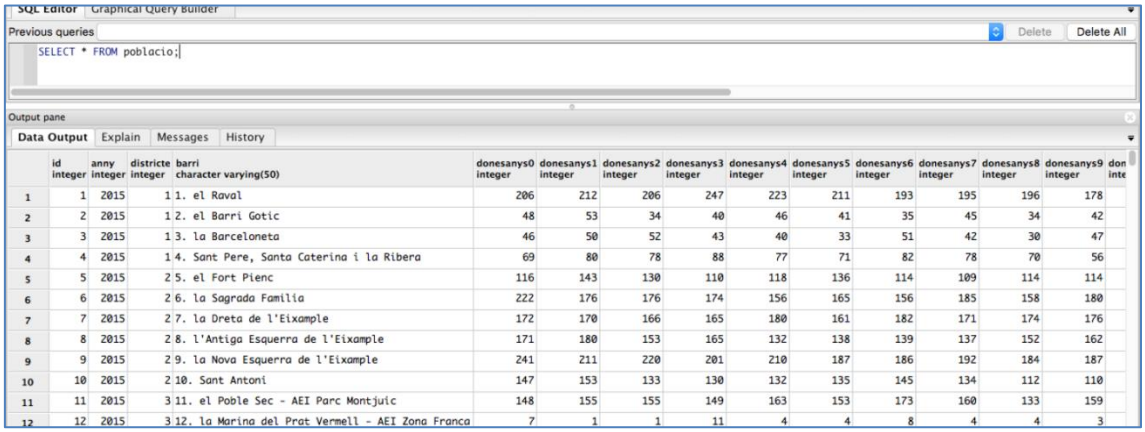
Una vegada recollida tota la informació, s'assegura que no hi hagi mancances ni dades errònies. Amb aquest fitxer es podrà importar tota la informació a la BD.

2.2 Base de dades

En **PostgreSQL**, el sistema gestor de bases de dades (SGBD) que s'ha triat per implementar l'aplicació, s'ha de crear la taula anomenant als atributs amb els mateixos noms que tenen les columnes a l'arxiu que s'importarà. Una vegada creat i executat el codi SQL per la creació de la taula *poblacio*[9], s'importa l'arxiu CSV indicant el tipus d'arxiu, el seu delimitador i que la primera

[9] Veure *Creació taula poblacio* a l'Annex III: Scripts SQL

fila és la capçalera. Es pot comprovar que s'han importat bé les dades fent una consulta sobre tots els registres, tal com es mostra en la Figura 5:



	id integer	anny integer	districte integer	barri character varying(50)	donesanys0 integer	donesanys1 integer	donesanys2 integer	donesanys3 integer	donesanys4 integer	donesanys5 integer	donesanys6 integer	donesanys7 integer	donesanys8 integer	donesanys9 integer	don inte
1	1	2015	1	1. el Raval	206	212	206	247	223	211	193	195	196	178	
2	2	2015	1	2. el Barri Gotic	48	53	34	40	46	41	35	45	34	42	
3	3	2015	1	3. la Barceloneta	46	50	52	43	40	33	51	42	30	47	
4	4	2015	1	4. Sant Pere, Santa Caterina i la Ribera	69	80	78	88	77	71	82	78	70	56	
5	5	2015	2	5. el Fort Pienc	116	143	130	110	118	136	114	109	114	114	
6	6	2015	2	6. la Sagrada Familia	222	176	176	174	156	165	156	185	158	180	
7	7	2015	2	7. la Dreta de l'Eixample	172	170	166	165	180	161	182	171	174	176	
8	8	2015	2	8. l'Antiga Esquerra de l'Eixample	171	180	153	165	132	138	139	137	152	162	
9	9	2015	2	9. la Nova Esquerra de l'Eixample	241	211	220	201	210	187	186	192	184	187	
10	10	2015	2	10. Sant Antoni	147	153	133	130	132	135	145	134	112	110	
11	11	2015	3	11. el Poble Sec - AEI Parc Montjuic	148	155	155	149	163	153	173	160	133	159	
12	12	2015	3	12. la Marina del Prat Vermell - AEI Zona Franca	7	1	1	11	4	4	8	4	4	3	

Figura 5. Dades de la taula *poblacio*

Ara ja es pot crear un servidor per servir les pàgines estàtiques als clients i perquè es connecti amb la BD i que ofereixi una API des d'on servir les dades.

2.3 Servidor

El servidor, com s'ha comentat abans, estarà escrit en JavaScript i s'executarà en **Node.js** amb el marc de treball **Express.js**, que facilita la tasca d'aixecar un servidor, i la llibreria **Massive.js** de Rob Conery, per poder connectar amb la base de dades PostgreSQL i manejar grans volums de dades. S'ha dividit el servidor en tres arxius :

1. **server.js**: arxiu principal del servidor. Es troben les comandes de connexió amb la BD i de creació del servidor.
2. **server/config.json**: arxiu de configuració on s'especifica les dades per connectar amb la BD de forma local.
3. **server/routes.js**: arxiu on s'emmagatzemen totes les rutes a l'API i l'execució de les seves respostes.

Amb referència al codi, es destaca a continuació les parts més importants i la seva explicació. Pel que fa a l'arxiu *server.js*, una vegada

carregat el fitxer de configuració, mitjançant Massive.js es crea una instància de connexió amb la base de dades i s'assigna aquesta a la variable *db* del servidor [6]:

```
var connection = "postgres://" +
    config.postgres.user + ":" +
    config.postgres.password + "@" +
    config.postgres.host + ":" +
    config.postgres.port + "/" +
    config.postgres.db;
// Connect to PostgreSQL Database
var massiveInstance = massive.connectSync({connectionString: connection});
app.set('db', massiveInstance);
```

Es defineixen els directoris dels arxius estàtics que es serviran al client quan es connecti i la resposta a totes les rutes dins de l'aplicació Ember.js [7]:

```
app.use(express.static(__dirname + '/public'));

app.get('/*', function(req, res) {
    res.sendFile(path.join(__dirname + '/public/' + '/index.html'));
});
```

Com es pot veure en aquestes cinc línies de codi, qualsevol petició al servidor –que no sigui a l'API– serà redireccionada a l'arxiu *index.html* que conté l'aplicació Ember. Serà aquesta la que s'encarregui de redirigir a la vista corresponent. Per exemple, per a la direcció:

https://barcelona-dashboard.herokuapp.com/poblacio

Ember servirà la vista *poblacio.hbs* que té compilada dins de la carpeta de serveis estàtics. Es veurà més endavant quan es parli d'Ember.js.

Per altra banda, dins de l'arxiu *routes.js*, es configura el *router* d'Express perquè serveixi crides a l'API dins de la ruta */api/v1* [8]. Dins de l'arxiu *routes.js* es troben les diferents crides que es poden fer [9]. En aquesta primera versió, es pot trobar la crida a les dades sobre població en el següent codi:

```
// Demography
router.get('/population', function(request, response) {
    db.poblacio.find({}, function(err, res){
        if (err) {
            handleError(err);
        }

        response.json({
            data: res.map( (el) => {
                var womenYears = [];
```

```
11.                 var menYears = [];
12.                 var womenTotal = 0
13.                 var menTotal = 0;
14.                 for (var i = 0; i <= 95; i++) {
15.                     var women = el['donesanys' + i];
16.                     var men = el['homesanys' + i];
17.                     womenTotal += women;
18.                     menTotal += men;
19.                     womenYears.push(women);
20.                     menYears.push(men);
21.                 }
22.                 return {
23.                     id: el.id,
24.                     type: 'poblacios',
25.                     attributes: {
26.                         year: el.anny,
27.                         district: el.districte,
28.                         neighbor: el.barri,
29.                         womenYears: womenYears,
30.                         menYears: menYears,
31.                         womenTotal: womenTotal,
32.                         menTotal: menTotal
33.                     }
34.                 };
35.             })
36.         });
37.     });
38. });
```

El que fa aquest codi es crear un parell d'*arrays* (vectors) i un parell de totals per cada registre de la BD i envia aquestes dades al client en format JSON . Cada registre que s'envia conté, per ordre (línies 23-32):

- **id**: identificador.
- **type**: tipus de dades. En aquest cas, és un model del tipus *poblacios*. Es necessari afegir una *S* si es vol emprar com a model d'Ember. Per exemple, si es disposés d'un model *cotxe*, Ember aniria a cercar les dades de tots els cotxes al plural del seu model per connotació anglesa, això és, afegint una *S* a la paraula.
- **attributes**: les dades en sí. Aquestes són:
    - **year**: l'any de les que s'obtenen les dades.
    - **district**: el districte de Barcelona on es troba el barri.
    - **neighbor**: el barri de Barcelona on s'originen les dades.
    - **womenYears**: *array* amb el nombre de dones segons l'edat. L'edat ve determinada per l'índex de l'*array*. És a dir, a womenYears[46] es troben el nombre de dones amb una edat de 46 anys per aquell any i barri.
    - **menYears**: *array* amb el nombre de dones segons l'edat.

- **womenTotal**: el nombre de dones total del barri a l'any indicat. Es fa aquest càlcul al servidor perquè és una dada que s'utilitzarà sovint i així s'alleugera els càlculs en client.
- **menTotal**: el nombre de dones total del barri a l'any indicat.

Així, cada vegada que es cridi a l'API amb la ruta */api/v1/population* es serviran les dades, com es mostra en la següent Figura 6:

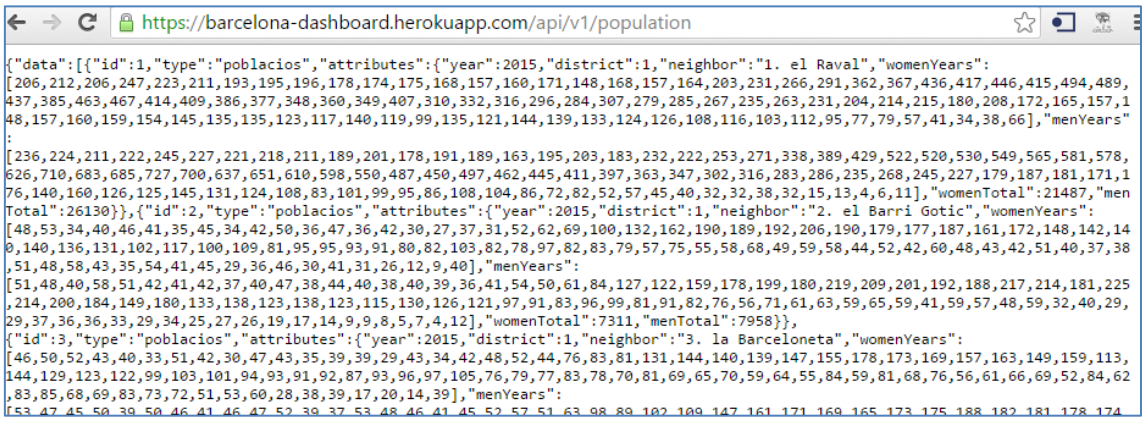


Figura 6. Resposta de l'API al cridar */api/v1/population*

Si es vol arrencar el servidor, s'ha d'executar la següent comanda dins d'un terminal i dins de l'arrel de la carpeta on estigui el projecte:

```
node server.js
```

2.4 Aplicació client

2.4.1 Bootstrap

El marc de treball Twitter Bootstrap o, simplement, **Bootstrap**, conté un conjunt d'estils tipografies, components i altres elements que fan més fàcil el disseny de les aplicacions web.

Un dels elements destacables de Bootstrap és el seu **sistema de graella** per realitzar un disseny responsiu de les pàgines d'una forma senzilla. La graella es divideix en files i columnes. Sobre aquestes últimes es poden aplicar diferents mides segons el dispositiu o la finestra de l'explorador que s'empri per accedir a la pàgina web. Aquest sistema permet dividir l'espai en 12

trossos i diferenciar entre 4 mides de pantalla: molt petita (*xs*), petita (*sm*), mitjana (*md*) i gran (*lg*). Si es declara només una classe de mida, totes les que siguin més gran agafaran la mateixa amplada. Es poden veure exemples de diferents mides de pantalla a la infografia de la Figura 7:

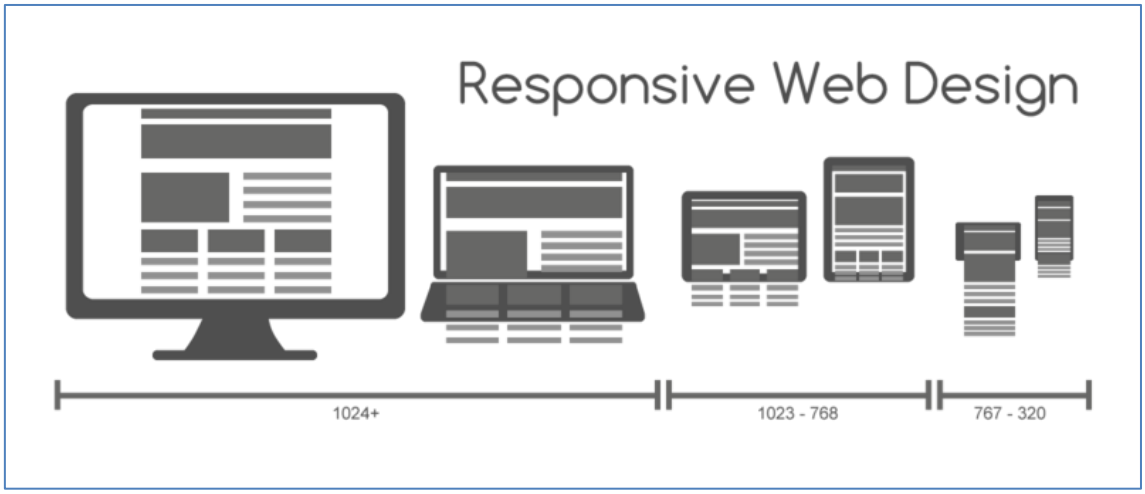


Figura 7. Exemple de distribució dels elements d'una pàgina segons el dispositiu[10]

Suposem, per exemple, que es tenen dos elements de tipus *div* que es vol que estiguin un al costat de l'altre, ocupant cadascú el 50% de la pantalla si es veu en una finestra gran, però que es vol que ocupin el 100% de l'amplada si es veu en una finestra mitjana, petita o molt petita. Llavors només s'ha de posar dos classes, *xs* i *lg*, de la següent manera:

```
1.  <div class="col-xs-12 col-lg-6"> Primer element </div>
2.  <div class="col-xs-12 col-lg-6"> Segon element </div>
```

Tot i que fer l'aplicació amb un disseny totalment responsiu queda fora de l'abast d'aquest projecte, especialment per telèfons mòbils, es mirarà d'aplicar el sistema de graella perquè la pàgina sigui accessible des de qualsevol dispositiu.

En la mateixa línia, es vol destacar el component de barra de navegació d'aquest marc de treball. Aquest component permet de forma simple crear una

[10] Rafizeldi, M. (2013). Responsive Web Design for Desktop, Notebook, Tablet and Mobile Phone [PNG]. Recuperat de https://commons.wikimedia.org/wiki/File:Responsive_Web_Design_for_Desktop,_Notebook,_Tablet_and_Mobile_Phone.png

barra de navegació amb llistes no ordenades del llenguatge de marcat HTML. A més, és un component responsiu i s'adapta al dispositiu emprat, tal com es pot veure en la Figura 8:

Figura 8. Adaptació de la barra de navegació a un telèfon mòbil:

2.4.2 Ember

Pels que estiguin familiaritzats amb el marc de treball JavaServer Faces (JSF) de Java, trobaran moltes similituds amb el desenvolupament fet amb **Ember.js**. En aquest projecte, el codi Ember es pot trobar, des de l'arrel, dins de la carpeta ***client-app/app/***.

Ember proporciona el patró Model-Vista-Controlador(-Component) per a la presentació. Les vistes són plantilles *handlebars* anomenades ***templates*** amb extensió HBS. Aquestes plantilles s'implementen en codi de marcat HTML

i permet incrustar expressions *handlebars*[11] que poden inserir altres vistes, components, *helpers* com blocs condicionals o bucles i propietats del controlador de la vista. Per exemple, es pot tenir una propietat anomenada *usuari* dins del controlador amb les dades de l'usuari. Amb *handlebars* es pot cridar a la propietat *nom* d'aquest objecte i, una vegada compilat el codi i carregat al navegador, es mostrarà a l'explorador, tal com es mostra a la següent taula de la Figura 9:

controller.js	**template.hbs**	**template.html** (una vegada compilat i executat a l'explorador)
`Usuari: {nom: 'Joan', Cognom: 'Ningu'}`	`<p> Hola, {{usuari.name}}! </p>`	`Hola, Joan!`

Figura 9. Exemple d'Ember mostrant una propietat del controlador

Pel que fa al **controlador**, aquest permet emmagatzemar accions (mètodes) i, com s'ha dit, propietats. Unes propietats molt interessants són els *observers* i les *computed properties*, que reaccionen al canviar les propietats que observen o en depenen. Continuant amb l'exemple anterior, es pot tenir una *computed property* que es digui *nomSencer*, que retorna la concatenació del nom i el cognom de l'usuari. Quan s'inicia l'aplicació, el seu valor és 'Joan Ningu' i es mantindrà així fins que canviï alguna de les propietats de la que depèn. Per tant, quan es canviï el cognom per 'Fuster', la propietat *nomSencer* passarà a contenir la cadena 'Joan Fuster'.

El **model** que es carrega al controlador i a la vista és configura al seu ***route*** (ruta en català) corresponent, que funciona d'alguna manera com la via d'entrada a la vista i permet la inicialització del controlador. També permet controlar transicions a altres rutes i accions abans i després de la carrega del model. El *route* controla que en el model es resolgui la promesa de carrega de dades abans de passar al controlador i la vista. És a dir, fins que el model no s'ha carregat del tot, no es mostrarà res més.

Els **components** poden funcionar com a vistes més petites i modulars. Es divideixen en una vista i un controlador. Tots els gràfics que

[11] Per obtenir més informació sobre Handlebars es pot visitar: http://handlebarsjs.com/

s'implementaran es desenvoluparan com a components. Modularitzant-los d'aquesta manera serà més senzilla la seva reutilització en altres vistes. A més, quan es crida al component es poden inicialitzar les propietats que es vulguin. A tall d'exemple, es pot haver implementat un component rellotge que mostri l'hora amb els minuts i els segons amb una animació. Des de la vista es pot cridar inicialitzant l'hora i els minuts d'aquesta manera:

```
{{rellotge hora=18 minut=04}}
```

Sobra dir que la vista o component i el seu controlador corresponent han de tenir el mateix nom. En aquest entrega es poden veure els diferents components, rutes (*routes)*, controladors (*controllers*) i plantilles (*templates*) dins de les carpetes del mateix nom.

Ember permet l'ús de **serveis** (*services*) amb patró *singleton*. Dit d'una altra manera, quan s'executa aquest servei és únic per a tota la instància de l'aplicació. S'utilitzarà un servei anomenat *data-service.js* per emmagatzemar les dades amb Crossfilter (que es veurà més endavant). Es farà així per controlar que les dades estiguin centralitzades i dimensionades en un mateix lloc per totes les vistes que utilitzin la mateixa informació. És recomanable fer la crida a l'API fora del servei. El problema que es presenta es que la promesa de carrega de dades pot trigar massa temps en complir-se i, llavors, es mostra la vista sense cap dada. Després de diferents implementacions, s'ha decidit que sigui el *route* qui s'encarregui de fer la crida per obtenir les dades i, una vegada carregat el model, inicialitzar les dades al servei tal com es mostra a la Figura 10.

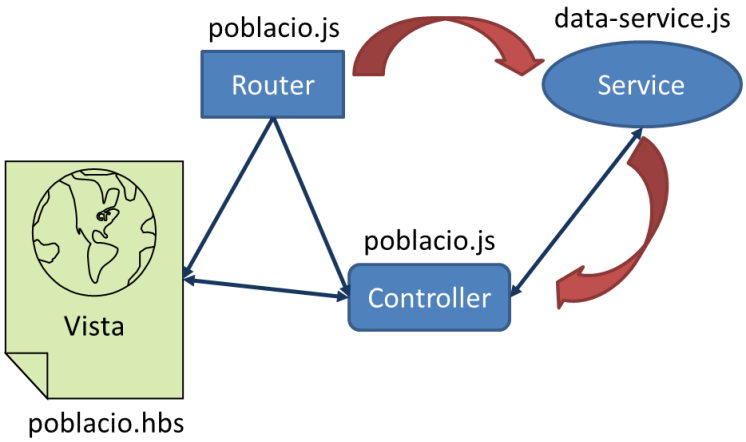


Figura 10. Interconnexió dels diferents mòduls per implementar la vista de població

Aquest servei s'injecta al controlador i permet usar les seves propietats. Reproduïm a continuació el codi de la ruta *poblacio.js* on es mostra la decisió d'implementació per resoldre tant la carrega de dades de l'API com els fitxers JSON on s'emmagatzemen les dades per dibuixar els mapes dels districtes i barris de Barcelona:

```
1.  import Ember from 'ember';
2.  import ajax from 'ic-ajax';
3.
4.  export default Ember.Route.extend({
5.      dataService: Ember.inject.service('data-service'),
6.
7.      model() {
8.          return Ember.RSVP.hash({
9.              population: ajax({
10.                 url: '/api/v1/population',
11.                 type: 'get'
12.             }),
13.
14.             district: new Promise((res, rej) => {
15.               d3.json('assets/districtes.json', function(err, data) {
16.                 err ? rej(err) : res(data);
17.               });
18.             }),
19.
20.             neighbor: new Promise((res, rej) => {
21.               d3.json('assets/barris.json', function(err, data) {
22.                 err ? rej(err) : res(data);
23.               });
24.             })
25.         });
26.     },
27.
28.     afterModel: function(model, transition) {
29.         this.get('dataService').initPopulation(model.population.data);
30.     },
31.
32.     setupController: function(controller, model) {
33.         this._super(controller, model);
34.     },
35.
36. });
```

Crida a l'API i carrega a *model.population*

Carrega dels districtes a *model.district*

Carrega dels barris a *model.neighbor*

Un cop resoltes les promeses del model, s'inicialitza les dades al servei

En els següents apartats s'amplia la implementació portada a terme en la part de client, però abans de continuar es vol comentar com es compila l'aplicació. Ember consta d'un arxiu *index.html* que serveix de base per a totes les aplicacions. En aquest arxiu es compilen totes les vistes que es cridin des del *template application.hbs.* Els arxius d'estil CSS, les llibreries i els marcs de treball JavaScript es compilen dintre de la carpeta *assets*. Tot es compila a la carpeta de destí, que en aquest cas es troba a l'arrel del projecte amb el nom de ***public/.*** A més, el compilador també fa una copia de la carpeta *assets* que

es troba dins de l'aplicació client. Aquesta carpeta serveix per desar altres fitxers d'interès per l'aplicació com ara els fitxers JSON amb la informació de dibuix dels mapes o les imatges que s'utilitzin.

Per arrencar la compilació de l'aplicació, cal executar la següent comanda en un terminal a l'arrel del client (en el cas d'aquest projecte, a la carpeta *client-app/*):

- `ember build`

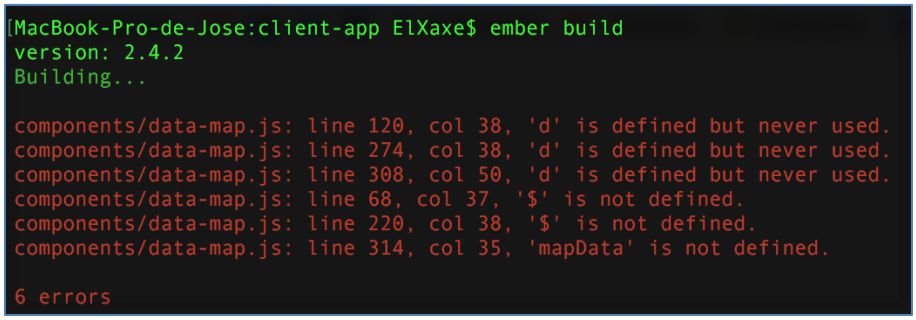


Figura 11. Compilació de l'aplicació Ember.js i avisos de JSHint

Si es disposa, a més, del programa afegit JSHint[12], es poden comprovar les errades o alguns avisos sobre el codi tals com faltes de punt i coma, variables no definides o no utilitzades, com es mostra a la Figura 11.

Per últim, abans de passar al següent apartat, es vol comentar que el codi JavaScript implementat utilitza la notació **EMACScript 2015** o ES6. Entre les seves novetats es destaquen:

- Utilització de *let* o *const* en comptes de *var* per declarar variables. La diferència radica en que *const* declara constants i, per tant, no són modificables, mentre que *let* es una variable normal.

- *Arrow functions*: En comptes de declarar les funcions de *callback* de JavaScript com:

```
reduce( function(a, b) { return a + b; })
```

Ara es pot fer de la següent manera:

[12] Més informació sobre JSHint es pot trobar a http://jshint.com/

```
reduce( (a, b) => a +b; )
```

A banda de escriure menys, això te l'avantatge de que també es passa el context des d'on es crida. És a dir, la partícula *this* dintre d'una *arrow function* no es refereix al context de dintre de la funció sinó al context superior que fa la crida.

La **primera pàgina** que es mostra de l'aplicació és de benvinguda. Ofereix una petita explicació i una barra de navegació que, de moment, ofereix la possibilitat d'accedir a la vista d'informació de la població de la ciutat de Barcelona (veure Figura 12).

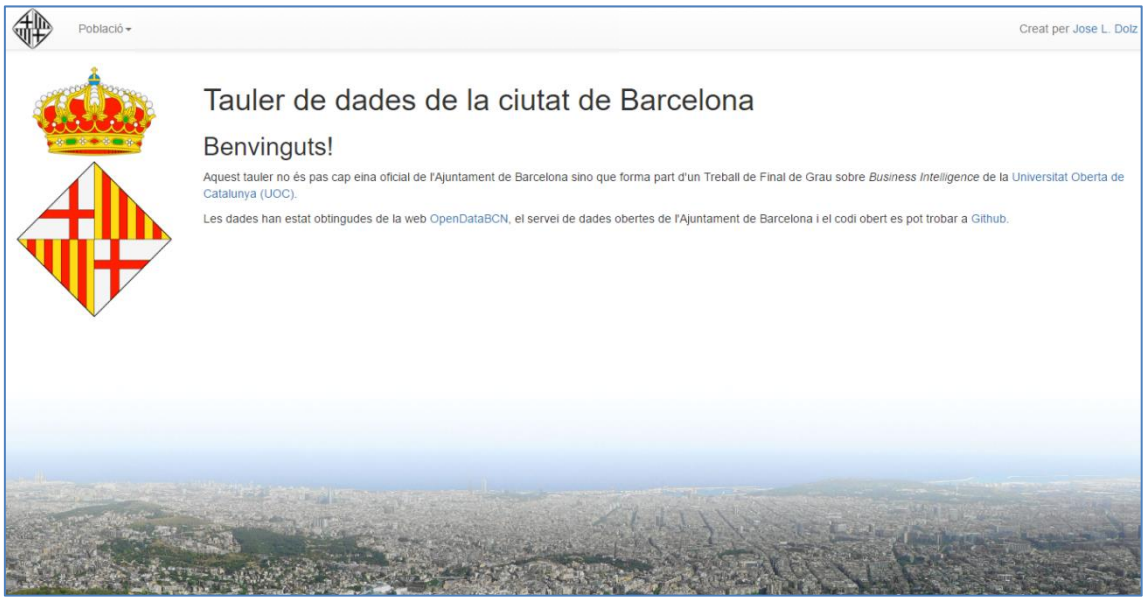


Figura 12. Pantalla final de benvinguda a l'aplicació

2.4.3 Mapes amb D3

Com ja s'ha comentat, els mapes s'han implementat com a components d'Ember.js i es dibuixen amb l'ajuda de la llibreria **D3.js** de Mike Bostock [10]. Com s'ha vist a l'apartat anterior, en el *route* es carrega el model amb la informació per dibuixar els mapes. Aquesta informació s'ha obtingut gràcies al treball desinteressat de Martín González que ofereix els fitxers JSON amb informació topogràfica (**TopoJSON**) de Barcelona de manera oberta [11]. Una

vegada carregat el model, des de la vista es crida al component amb el següent codi:

```
1.  {{data-map
2.  districtView = viewDistricts
3.  mapData = dataMap
4.  mapPaths = paths
5.  units='habitants'
6.  reseted=resetMap
7.  class = "col-xs-12"
8.  id = "map"
9.  }}
```

Com es pot veure, s'inicia amb un booleà que diu si es veu els districtes o els barris (línia 2) determinat per la propietat del controlador *viewDistricts*; a *mapData* es passa la informació que es vol que mostri i que s'ha obtingut de l'API; *mapPaths* conté la informació per dibuixar el mapa (*paths* són les formes que es dibuixaran, com es veurà més endavant, del districtes o barris); a la línia 5 es determinen les unitats que s'han de mostrar; un booleà que indica si el mapa s'ha de reiniciar (línia 6); i per últim, la mida de la secció del mapa dins de la seva fila i columna i un identificador.

Dins del component *data-map.js* –que es pot veure dins de la carpeta ***client-app/app/components/–*** es troben les propietats d'aquest. El component s'inicia amb el mètode *didInserElement()*, que es dispara quan el component s'insereix a la vista i es modifica quan es canvia el booleà que ens diu si la vista és de districtes o barris (anomenat *districtView*) i amb les dades que contenen la informació de la poblaci, dins de la propietat *mapData*.

D3 funciona, *grosso modo*, de la següent manera [12]:

- El gràfic és una jerarquia d'elements, on el pare o arrel és el gràfic SVG (de l'anglès *Scalable Vector Graphics* o Gràfics Vectorials Escalables).
- Els elements s'afegeixen a elements pares. La tècnica més normal és afegir primer un grup (element *<g>*) i afegir a aquest un conjunt d'elements iguals com, per exemple, els rectangles que serviran per fer un gràfic de barres.
- Es poden (i es deuen) crear escales per l'eix X i l'eix Y. Aquestes escales es poden mapejar d'un domini a un rang. Els dominis i els rangs

poden ser variats, però els més comuns son els linears i els ordinals. A tall d'exemple, es copia el següent tros de codi, que es troba dins del component *year-evolution.js*, que es comentarà més endavant:

```
1.  let x = d3.scale.ordinal()
2.      .domain( years.map(function(d) { return d; }) )
3.      .rangeRoundBands([0, width], 0.1);
```

Aquí, es declara X como una escala ordinal, que mapeja un vector de cadenes –en aquest cas, format pels anys que es mostraran– amb un rang que va des de 0 fins al nombre que es declara a la variable *width* (que correspondrà amb l'amplada del gràfic). És a dir, que si el primer element del vector és la cadena '2007', l'escala el mapejarà com un cero. Si el darrer element és '2015', el mapejarà amb el valor de *width*. Això serveix, entre d'altres coses, per ordenar els elements dins d'un eix.

- De la mateixa manera, es poden declarar escales de colors. Es declara un primer i un últim color i D3 s'encarregarà d'assignar el color corresponent dins del rang on calgui.

- Cada vegada que s'afegeix un element (o un conjunt), es poden declarar les seves propietats i atributs, com posició X i Y, la classe que li assignem, l'amplada i l'altura, els estils, les seves classes, accions, etc.

- Abans d'afegir els elements en un grup, se'ls si ha d'assignar les dades amb la funció *data()*. Normalment, aquestes dades es troben en forma d'*array* d'objectes (nombres, cadenes, tuples, etc.). Cada element gràfic d'un grup correspondrà a una dada i tindrà el seu índex. En cada propietat es pot cridar a una funció de *callback*, que té per paràmetres l'element actual de l'*array* i el seu índex. Per exemple, en el següent tros de codi es pot veure com es poden dibuixar unes barres, basades en rectangles, amb una altura que vindrà determinada per les dades:

```
1.      svg.selectAll('.bar')
2.          .data(myData)
3.          .enter()
4.          .append('rect')
5.          .attr('class', 'bar')
6.          .attr('x', function (d, i) { return xScale(i); })
7.          .attr('y', 0)
8.          .attr('width', 50)
9.          .attr('height', function (d, i) { return d; })
```

El que aquest codi expressa és que es seleccionaran tots els elements que s'anomenaran amb la classe *bar* i se'ls hi assigna les dades de l'*array myData* (línies 1 i 2). Aquest grup serà de rectangles (*rect*) que cadascú tindrà la classe *bar* (línies 4 i 5). La seva posició al eix X vindrà determinada per l'índex que té a l'*array* i segons l'escala que s'hagi creat abans per a l'eix X (línia 6). La seva posició en l'eix Y serà a l'inici i la seva amplada de 50 píxels (línies 7 i 8). Per últim, la seva altura vindrà determinada per l'element actual de l'*array* de valors.

- Les accions són *listeners* (escoltadors) d'esdeveniments. A un element se li pot assignar que reaccioni quan fan clic sobre ell, o quan passen el cursor del ratolí per sobre, etc.

En el component *data-map* el primer que es fa es crear el gràfic SVG amb unes mides determinades. Després s'afegeixen els *paths* o formes descrites a l'arxiu TopoJSON carregat i a cada forma s'afegeix un *listener* perquè quan es passi per damunt es mostri una petita finestra amb informació i, alhora, canviï el color de la forma.

Una vegada es polsa sobre el botó *Mostrar barris* o es canvia la informació, s'executa la *computed property changeMap*, que carrega la nova informació dels *paths* o formes dins del gràfic SVG, com es veu a la Figura 13. A més, el botó també canviarà al seu text a *Mostrar districtes* si es prem, com s'ha declarat al codi de la vista *poblacio.hbs*.

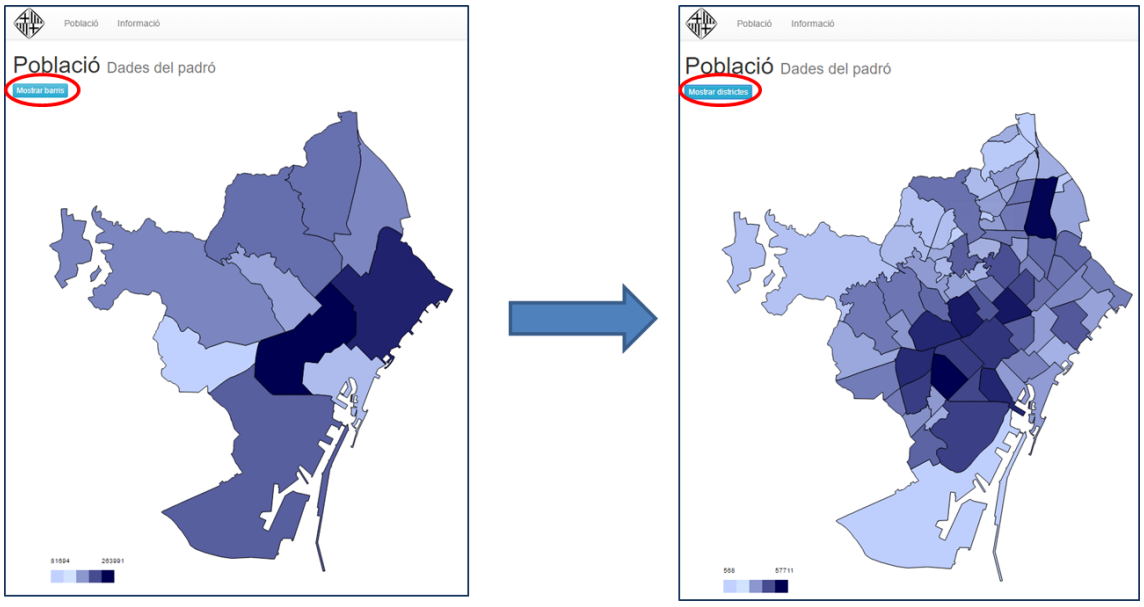


Figura 13. Canvi del mapa al polsar el botó de canvi de vista

Quan s'actualitza un gràfic a D3, només cal modificar les propietats indicades dels elements que es volen. Per exemple, si es modifica el nombre d'habitants d'un any mostrat en una barra d'un gràfic, només s'hauria d'actualitzar la seva altura. A més, D3 ofereix el mètode *transition()* que permet que aquest canvi sigui animat i es pot assignar una durada i el tipus d'animació que es vol.

El gràfic rep dades de la població a cadascun dels districtes o barris i els pinta segons el seu rang dins de l'escala de colors declarada. El tractament de les dades que es passen al mapa es veurà a l'apartat *Tractament de les dades amb Crossfilter*.

Tractar de crear un gràfic D3 com un component Ember ha estat una tasca bastant difícil. Normalment, el codi D3 es troba o es crida en el codi JavaScript d'una pàgina HTML que permet tenir variables globals o compartides entre les diferents parts del codi. Però aquesta no és la manera de treballar amb mòduls i, sobretot, amb Ember. Cada controlador només pot tenir accés a les seves propietats i a les propietats dels serveis que s'injecten. A més, per emprar una propietat dins d'una altra s'ha de cridar al mètodes *this.get('propietat')*. Per tant, quan es crea l'element SVG, aquest s'ha d'assignar a una propietat del controlador del component dins del mètode *didInsertElement*.

En aquest cas, es pot cercar l'element SVG dins del DOM (*Document Object Model*) de la vista però es prou difícil. No es pot seleccionar l'element pel nom de la seva etiqueta doncs poden existir altres elements SVG a la vista i es podria selecciona un element incorrecte. A més, Ember dona, per defecte, un identificador diferent cada vegada que insereix un component [13]. Això s'ha solucionat amb la següent línia de codi:

```
let svg = d3.select('#'+this.get('elementId'));
```

Es posa dins d'una variable local l'element amb l'identificador indicat per la propietat *elementId* del component actual i així es podrà estar segur que s'està treballant sobre el component adequat.

Aquest gràfic serveix com a filtre per les dades. Si es selecciona un districte o barri, segons la vista, es seleccionarà només les seves dades concretes per mostrar-les en els altres components gràfics que es trobin a la vista.

2.4.4 Gràfics amb D3

A continuació es veurà els diferents gràfics que s'han creat com a components i que formen part de la vista 'Població: dades del padró'. D3 ha d'adaptar algunes dades depenent del tipus de gràfic que es dibuixa. Així, quan es tracta d'un gràfic de línia, hi ha un mètode *d3.svg.line()* que permet adaptar-les. El mateix passa per als diagrames de sectors però no cal, per exemple, per als gràfics de barres. Detallar les diferències de cada gràfic de D3 queda fora de l'abast d'aquest projecte.

Els primers dels gràfics (veure Figura 14) és un diagrama de sectors on es divideix la població de Barcelona per sexes. Es troba al component ***pie-chart.js*** i rep un vector de parells claus-valor. En aquest cas el nombre de dones i homes al padró de Barcelona per l'any 2015 i totes les edats, de 0 a 95 anys. Més endavant rebrà les dades que l'usuari hagi seleccionat en els altres gràfics.

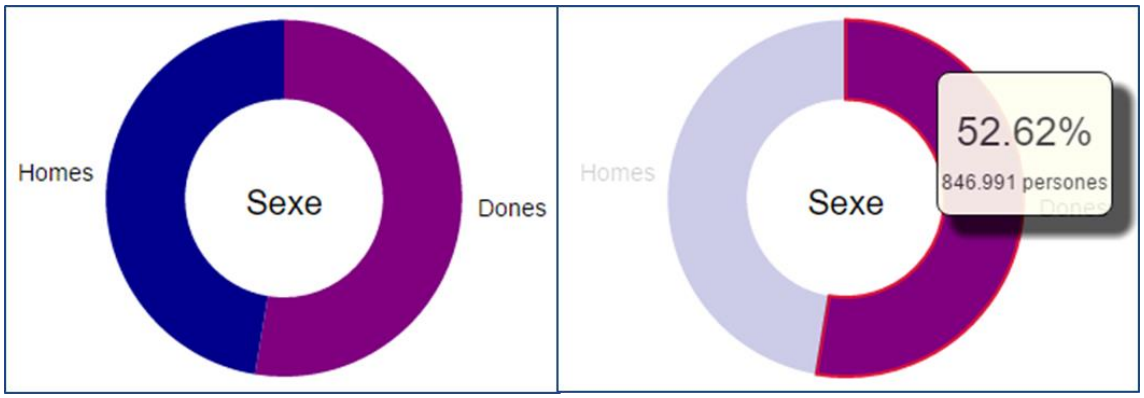


Figura 14. Diagrama de seccions segons sexe (sense seleccionar i amb selecció)

Aquest gràfic serveix també de filtre per seleccionar les dades que mostren en els altres. En altres paraules, es pot seleccionar, fent clic, només als homes o les dones i els altres gràfics s'actualitzaran en conseqüència. Si es clica sobre el títol del gràfic –Sexe- es seleccionaran tots dos gèneres.

El següent gràfic que es troba en aquesta vista és un gràfic híbrid de barres i línia que es troba en el component ***year-evolution.js***. Les barres només serveixen per assenyalar l'any de les dades que es mostren a la resta de gràfics i poder canviar d'any. La línia del gràfic mostra l'evolució de la població en els diferents anys i si es passa el cursor per sobre dels cercles es pot veure una finestra que indica el nombre d'habitants, com es mostra en la següent Figura 15:

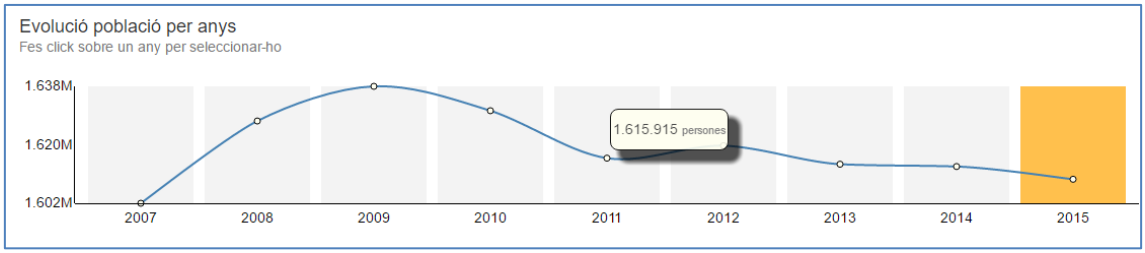


Figura 15. Gràfic de l'evolució de la població per anys

Si es clica, per exemple, sobre el diagrama de sectors del sexe de la població, es pot veure com aquest gràfic s'actualitza amb una animació, tant la línia d'evolució com el eix Y a l'esquerra del gràfic.

Per últim, el darrer dels gràfics mostra la quantitat d'habitants dividits per edat per l'any 2015 a la ciutat de Barcelona (veure Figura 16). Com els altres gràfics, s'actualitzarà segons els filtres que l'usuari hagi escollit. Es pot

comprovar que s'anima i actualitza, tant el gràfic com l'eix Y, si es selecciona un filtre concret en els altres components de la vista. A més, aquest gràfic serveix per seleccionar un rang d'edat de la població si s'arrossega el cursor per sobre de les edats desitjades.

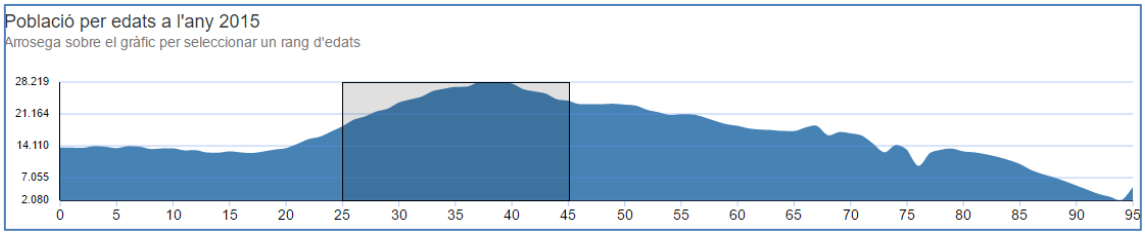


Figura 16. Gràfic de la població dividida per edats amb un rang seleccionat

Tots els elements de la vista s'han tractat amb el sistema de graella responsiu que s'ha comentat a l'apartat *Bootstrap*. Com ja s'ha comentat, queda fora de l'objectiu del projecte adaptar-ho per dispositius mòbils encara que s'ha fet el millor per tenir-los en compte. Aquest projecte està orientat a gaudir-se en pantalles d'ordinador. Per això, es possible que l'experiència d'usuari quedi malmesa quan s'accedeix a l'aplicació amb dispositius amb una pantalla petita.

2.4.5 Tractament de les dades amb Crossfilter

Crossfilter, de la companyia Square Inc., permet tractar grans volums de dades de forma ràpida i eficient. La forma de fer-ho es declarar dimensions sobre la informació. Aquestes dimensions permetran filtrar, reduir i tractar les dades.

En el cas de la població, es pot veure en el servei *data-service.js* que s'han declarat tres dimensions: una per l'any, una altra pels districtes i altra pels barris. No ha calgut fer una pels gèneres al tractar-se només de dos elements que es poden controlar amb uns senzills condicionals. Per altra banda, tampoc s'ha fet una dimensió per les edats al tractar-se d'*arrays* que es manipulen amb bucles. Un cop es seleccionin les edats només caldrà refer els càlculs amb els índex indicats per les propietats *minAge* i *maxAge* del controlador *poblacio.js*.

Es comenta a continuació el tractament de les dades que es passen als quatre components gràfics de la vista Població. Tots els tractaments es fan al controlador *poblacio.js*. En primer lloc, el codi que correspon a les dades que es passen al mapa de la ciutat de Barcelona, que es troba a la *computed property dataMap*:

```
yearDim.filter(year);
if (isDistrict) {
    group = districtDim.group(function(d) { return d; });
} else {
    group = neighborDim.group(function(d) { return d; });
}

data = group.reduceSum( function(d) {
    let w, m;

    if (minAge === 0 && maxAge === 95) {
        w = d.attributes.womenTotal;
        m = d.attributes.menTotal;
    } else {
        w = d.attributes.womenYears.slice(minAge, maxAge).reduce(getSum);
        m = d.attributes.menYears.slice(minAge, maxAge).reduce(getSum);
    }

    if (gender === 'Dones'){
        return w;
    } else if (gender === 'Homes') {
        return m;
    }
        return w + m;
    });

return data.all();
```

En primer lloc, es filtren les dades per la dimensió de l'any indicat a la variable *year*. Si la vista es de districtes, les dades s'agrupen per la dimensió districte i, si no, per la dimensió barri. Una vegada agrupades, es sumen els habitants segons el rang d'edat escollit: si no s'ha triat cap, s'utilitzen els totals calculats a l'API; per contra, si s'ha escollit un rang d'edats, es selecciona el tros de l'*array* indicat per *minAge* i per *maxAge* i es redueix sumant els seus valors segons el sexe. Per acabar, es sumen els habitants de cada grup segons s'hagi seleccionat a les dones, als homes o a cap dels dos.

Les dades que es passen al diagrama de sectors de gènere es troben a la propietat computada *genderData*:

```
yearDim.filter(year);

if (!isDistrict) {
    group = districtDim.group();
} else {
    group = neighborDim.group();
}
women = $.map(group.reduceSum(function(d) {
   if ( minAge === 0 && maxAge === 95) {
      return d.attributes.womenTotal;
   } else {
     return d.attributes.womenYears.slice(minAge, maxAge).reduce(getSum);
   }
}).all(), function(el) { return el.value; });

men = $.map(group.reduceSum(function(d) {
   if ( minAge === 0 && maxAge === 95) {
      return d.attributes.menTotal;
   } else {
      return d.attributes.menYears.slice(minAge, maxAge).reduce(getSum);
   }
}).all(), function(el) { return el.value; });

data.pushObject({
    key: 'Dones',
    value: women.reduce(getSum)
});

data.pushObject({
    key: 'Homes',
    value: men.reduce(getSum)
});

return data;
```

Es filtra primer per l'any i s'agrupen segons si son districtes o barris. Llavors es suma el total de dones de cada districte o barri segons el rang d'edat, tal com hem vist en el punt anterior. Es fa el mateix amb les dades dels homes. Després, aquests *arrays* es redueixen sumant tots els seus elements i es passen al objecte *data*, que s'enviaran al diagrama de sectors, assignant-li a cada valor una clau: *Dones* o *Homes*.

Les dades pel gràfic d'evolució de la població, que es troben a la propietat *yearData*, són més senzilles de calcular:

```
data = yearDim.group()
    .reduceSum( function(d) {
        const women = d.attributes.womenYears.slice(minAge, maxAge);
        const men = d.attributes.menYears.slice(minAge, maxAge);
        const w = women.reduce(getSum);
        const m = men.reduce(getSum);

        if (gender === 'Dones') {
            return w;
        } else if (gender === 'Homes') {
            return m;
        }
```

```
        return w + m;
        });

return data.all();
```

Les dades s'agrupen per anys i per cada any es redueixen els *arrays* d'habitants, tallats segons el rang d'edats, amb una suma. Es calcula la suma tant per dones com per homes i després es retorna el valor segons el gènere que estigui seleccionat o la suma de tots dos si no s'ha seleccionat cap.

A *rangeData* es pot trobar el càlcul de les dades que es passen al gràfic d'edats. Pot ser és el càlcul més complex a simple vista:

```
yearDim.filter(year);
aux = yearDim.top(Infinity);

aux.forEach( (d, i) => {
 for( var j = 0; j < d.attributes.womenYears.length;  j++) {
   old = data[j] ? data[j] : 0;
   if (gender === 'Dones') {
      data[j] = d.attributes.womenYears[j] + old;
   }
   else if ( gender === 'Homes') {
       data[j] = d.attributes.menYears[j] + old;
   }
   else {
       data[j] = d.attributes.womenYears[j] + d.attributes.menYears[j] + old;
   }
 }
});

return data;
```

Un altre cop, es torna a filtrar per any. La comanda *top(x)* permet agafar els primers X elements de la dimensió. En aquest cas, amb el paràmetre *Infinity*, es poden agafar tots. Amb l'array resultant es fa un recorregut per cada element que, recordem, és un *array* de 0 a 95 on cada element són el nombre de habitants amb aquella edat. Es fa servir la variable *old* com acumulador i es va sumant els habitants de cada edat –segons el gènere escollit o tots dos si no s'ha triat cap– dins d'un *array* que seguirà la mateixa estructura i que serà el que s'enviarà al component *range-selector.js*.

2.4.6 Estils

Els estils es poden trobar dins de la carpeta ***client-app/app/styles***, al fitxer *app.scss*. Aquesta extensió es la que cercarà **SaSS** a l'hora de compilar-ho tot en un únic fitxer **CSS** que, en aquest cas, portarà el nom d'*app.css.*

Dins dels avantatges de SaSS es troba la possibilitat d'utilitzar variables per CSS i un sistema d'herència dels elements, identificadors i classes. El mètode clàssic seria:

```
body .main-content .population #lineYears {   height: 15vh;   }
```

Això assigna una alçada al gràfic identificat com *lineYears*, que es troba dins de la classe *population*, que es troba, alhora, dins de la classe *main-content* que està dintre de l'element *body*. Amb SaSS, es pot declarar com una jerarquia de la següent manera:

```
body {
	.main-content {
		#lineYears { height: 15vh; }
	}
}
```

Pot semblar igual que en el mètode clàssic però és perquè aquí només s'ha definit una propietat d'un element. Quan els elements comencen a créixer, definir els estils en una jerarquia es molt més eficient i guanya molt visualment, com es pot comprovar en la següent comparació de la Figura 17:

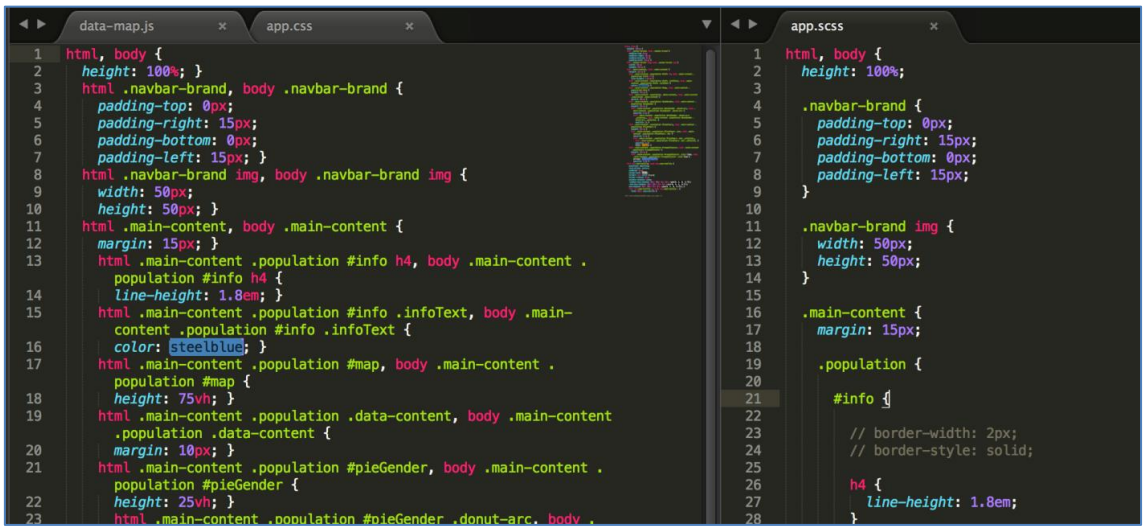


Figura 17. Comparació entre codi CSS compilat i l'escrit per a SaSS

2.5 Últims detalls i correccions

Abans de donar per finalitzada la vista Població i l'aplicació que es farà servir com a base del projecte, s'han d'afinar alguns punts per tal de tenir un desenvolupament i un programari de qualitat.

Primerament, configurarem el servidor perquè treballi amb entorns diferents. En el marc del desenvolupament de programari és normal trobar diferents entorns segons l'àmbit o utilització del codi. Normalment es poden trobar quatre diferents:

1. **Desenvolupament**: és l'entorn emprat pels desenvolupadors on poden modificar el codi, corregir-ho i implementar noves característiques.

2. **Proves**: també anomenat entorn de test, és l'entorn destinat a fer les proves sobre el codi i aplicació. Aquestes proves poden ser unitàries, d'integració, funcionals, de rendiment, etc.

3. **Intermedi**: o de *staging*, és un entorn semblant a l'entorn de producció que és utilitzat pel client per provar l'aplicació i informar sobre els possibles errors.

4. **Producció**: és l'entorn on s'executa la versió de l'aplicació que utilitzaran els usuaris finals.

En el cas d'aquest projecte s'ha fet necessari separar l'entorn del servidor en desenvolupament i producció. Aquesta separació esdevé imprescindible per la diferència de configuració entre la màquina on es desenvolupa el programari i el servidor web on està allotjada l'aplicació perquè sigui accessible a tothom. Més concretament, es tracta de diferenciar la configuració de connexió del servidor Node.js de l'aplicació a la BD de PostgreSQL. El codi de configuració, que es mostra a continuació, es troba al fitxer *server.js* a l'arrel del projecte:

```
1.  // Change connection string depending on the environment
2.  if ( process.env.NODE_ENV === 'development') {
3.    connection = "postgres://" +
4.                 config.postgres.user + ":" +
5.                 config.postgres.password + "@" +
```

```
            config.postgres.host + ":" +
            config.postgres.port + "/" +
            config.postgres.db;
}
else if ( process.env.NODE_ENV === 'production') {
  connection = "postgres://crwprravrqaurn:OygTkfp6dgaztg2wS7VpF7YZuf@ec2-23-21-215-184.compute-1.amazonaws.com:5432/d42fo6nihcr2iu";
}
else {
    console.log("Error on Node Environment: " + process.env.NODE_ENV);
}
```

Per executar l'entorn de desenvolupament a la màquina on es treballa, cal que es declari a la variable *NODE_ENV* abans d'executar el servidor Node amb la següent sentència:

- `NODE_ENV=development node server`

D'altra banda, l'entorn de producció serà l'escollit per defecte per Heroku[13], el servei gratuït pel desplegament d'aplicacions web seleccionat per allotjar aquesta aplicació. Gràcies a aquest servei, ja es pot tenir la primera vista de l'aplicació accessible per a tothom, tal com es mostra a la Figura 18:

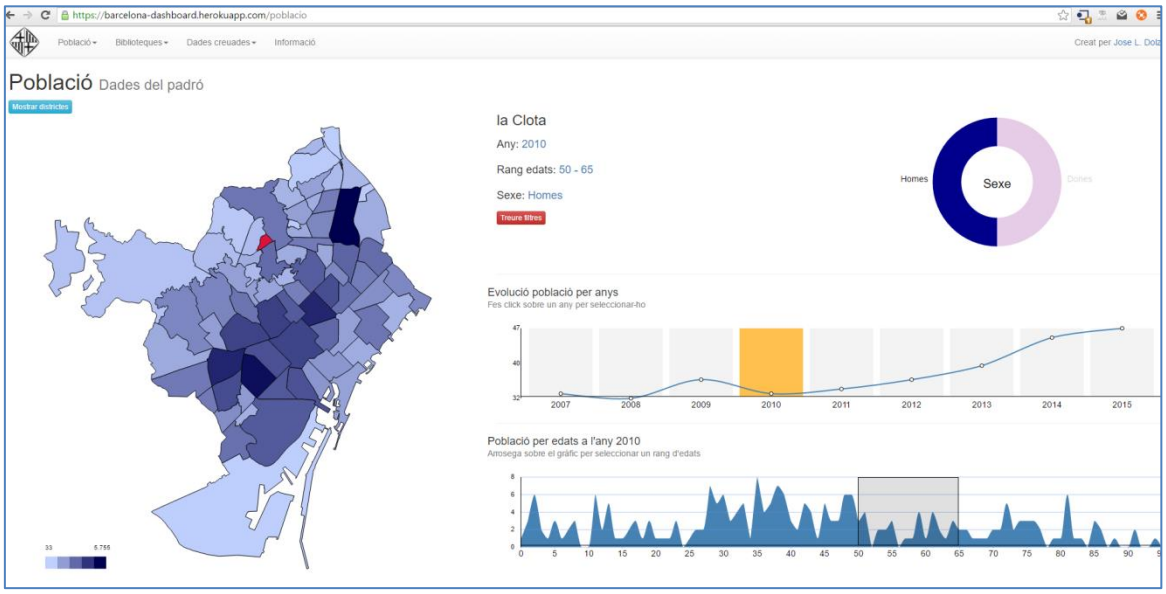


Figura 18. Versió final de la vista de població amb dades del padró

Com es pot observar, s'ha afegit a la vista una petita taula d'informació o es mostra textualment els filtres seleccionats de zona (districte o barri), any, rang d'edats de la població i el sexe. A més, es proporciona un nou botó que

[13] Més informació a http://www.heroku.com

apareix només quan hi ha cap filtre seleccionar i permet, al fer clic sobre ell, treure'ls tots.

Per acabar, es crea una vista d'informació on es llisten les tecnologies i els recursos utilitzats en l'elaboració de l'aplicació amb enllaços a la seves pàgines web corresponent perquè l'usuari pugui ampliar la informació (veure Figura 19) i conèixer al seus autors.

Figura 19. Disseny final de la vista d'informació.

3 Implementació de noves característiques

Aquest punt tracta sobre les noves característiques implementades a la aplicació. Dit d'una altra manera, es parlarà de la implementació de noves vistes referents a noves dades tretes del repositori de l'Ajuntament.

La selecció de nova informació ha estat una tasca estudiada i precisa. Tot i que en el repositori es poden trobar dades sobre temes interessants, tenen una implementació heterogènia. Dit amb altres paraules, no totes presenten una estructura semblant amb atributs d'any i de districte o barri. Fins i tot, els fitxers que es refereixen a les mateixes dades però d'anys diferents es poden trobar amb una codificació interna diferent, cosa que augmenta la dificultat del seu tractament.

En els següents apartats s'explica l'elecció i la implementació de les noves vistes de dades.

3.1 Biblioteques

Abans de començar la recerca de noves dades, es va escollir que una de les condicions que havien de complir era la seva possibilitat de creuament amb les dades sobre la població. D'aquesta manera, es va determinar que seria ideal trobar dades sobre serveis que oferia la ciutat als seus habitants. Així es podria mostrar, per exemple, el nombre d'habitants per un determinat servei, el seu ús, etc.

Després de comparar les dades sobre diferents serveis del repositori, es va decidir tractar les **biblioteques municipals**. La decisió d'escollir aquest servei ha estat en com estan configurades les seves dades amb atributs adients per la representació en l'aplicació. Altres serveis oferien una distribució d'atributs diferents, com ja s'ha comentat en el punt anterior, i tampoc donaven possibilitats d'agrupar un camp en poques categories o fer-ne sumatoris.

Tot i que aquestes dades eren la millor elecció, també presentaven problemes. Segons l'any de la seva publicació, la codificació del fitxer canviava. En alguns es trobaven codificacions d'idioma diferents, com UTF-8 o ISO 8859-1 (alfabet llatí), i les lletres amb accent gràfic es mostraven amb caràcters estranys o s'havien desat com a fitxers de dades separats per comes amb opcions diferents, com ara posar les cadenes de caràcters entre cometes. Per superar aquests obstacles s'ha fet ús del programari LibreOffice[14] de codi obert, mantingut per una comunitat de desenvolupadors de programari lliure. Com es pot veure a la Figura 20, l'aplicació permet escollir una sèrie d'opcions abans d'obrir un fitxer com ara la codificació, els separadors i els limitadors.

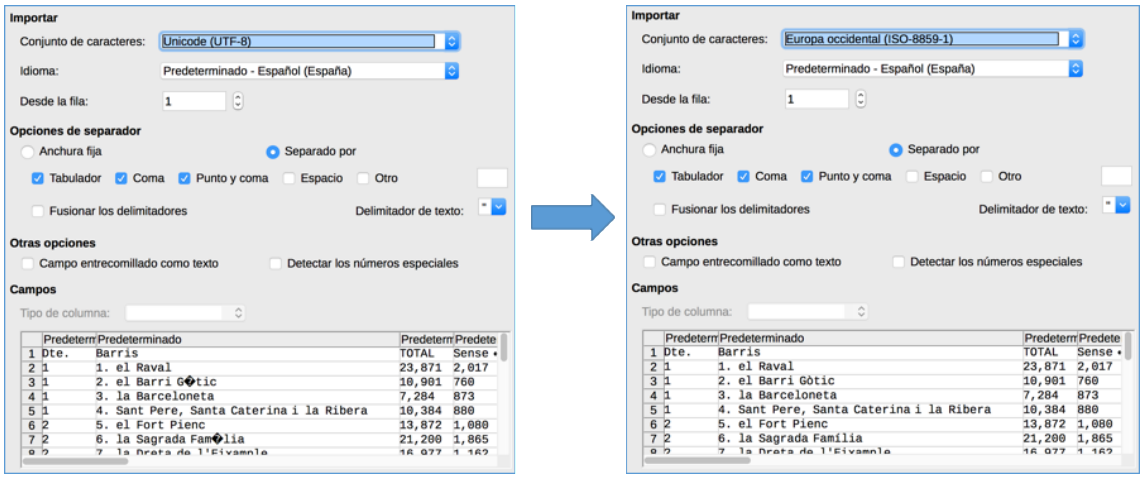


Figura 20. Opcions d'importació d'un arxiu de full càlcul a LibreOffice

De manera paral·lela, s'han revisat que les dades numèriques tinguessin el mateix format per a la base de dades: sense separadors de milers i que el separador dels decimals fos un punt. Per acabar, es crea un full de càlcul amb les dades de les biblioteques des de l'any 2010 fins l'any 2014 i es desa com un fitxer CSV (de l'anglès *Comma Separated Values*).

[14] Per obtenir més informació i/o descarregar LibreOffice es pot visitar la seva pàgina web a https://es.libreoffice.org/

3.1.1 Base de dades

En el repositori de l'Ajuntament de Barcelona es poden trobar les dades sobre biblioteques diferenciades entre les visites i els préstecs. Com la clau identificadora corresponent a la BD és igual en tots dos arxius –composta pels camps any i districte- es decideix combinar els préstecs i les visites en la mateixa taula, tai i com es mostra a la Figura 21.

	A	B	C	D	E	F	G	
1	Any	Ambit	Titularitat	TipusEquipament	Equipament	Districte	Prestecs	Nota
2	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Barceloneta - La Fraternitat	01. Ciutat Vella	76226	
3	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Francesca Bonnemaison	01. Ciutat Vella	147954	
4	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Gòtic - Andreu Nin	01. Ciutat Vella	66009	
5	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Sant Pau i Santa Creu	01. Ciutat Vella	101266	
6	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Esquerra de l'Eixample-Agustí Centelles	02. Eixample	223430	
7	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Fort Pienc	02. Eixample	113874	
8	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Joan Miró	02. Eixample	64210	La Biblioteca Joan Miró ha tancat per obres el 07/07/2014,
9								
10	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Sagrada Família	02. Eixample	268064	
11	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Sant Antoni - Joan Oliver	02. Eixample	103949	
12	2014	Lletres	Consorci o fundació amb presència municipal	Biblioteques	Biblioteca Sofia Barat	02. Eixample	72131	

Figura 21. Dades de les visites a biblioteques a l'any 2014

Així mateix, es decideix prescindir dels camps *Àmbit, Titularitat, TipusEquipament* i *Nota*. Aquests camps es consideren no pertinents a l'oferir informació poc rellevant i repetida. A més, es retalla el camp districte per mostrar només el nombre del districte doncs el nom ja ens ho ofereixen els arxius TopoJSON que s'utilitzen per dibuixar el mapa dels districtes. En conseqüència, la signatura de la nova taula *biblioteques* de la BD queda de la següent manera:

biblioteques: anny, districte, nom, visites, prestecs

Cal recordar que la paraula *any* es una paraula clau reservada a PostgreSQL i, per tant, no es pot fer servir i es decideix afegir una *n* per indicar l'any de les dades. Després de crear la nova taula a la BD del Sistema Gestor de Bases de Dades escollit[15], s'importen les dades des del fitxer CSV creat i es verifiquen que s'han desat correctament com indica la Figura 22.

[15] Veure *Creació taula biblioteques* a l'*Annex III: Scripts SQL*.

	id [PK] serial	anny integer	districte integer	nom character varying(50)	visites integer	prestecs integer
1	5	2014	1	Biblioteca Barceloneta - La Fraternitat	85968	76226
2	6	2014	1	Biblioteca Francesca Bonnemaison	162900	147954
3	7	2014	1	Biblioteca Gòtic - Andreu Nin	94148	66009
4	8	2014	1	Biblioteca Sant Pau i Santa Creu	194983	101266
5	9	2014	2	Biblioteca Esquerra de l'Eixample-Agustí Centelles	325978	223430
6	10	2014	2	Biblioteca Fort Pienc	200049	113874
7	11	2014	2	Biblioteca Joan Miró	98453	64210
8	12	2014	2	Biblioteca Sagrada Família	351177	268064
9	13	2014	2	Biblioteca Sant Antoni - Joan Oliver	139796	103949
10	14	2014	2	Biblioteca Sofia Barat	99842	72131
11	15	2014	3	Biblioteca Francesc Candel	132695	101905
12	16	2014	3	Biblioteca Poble Sec - Francesc Boix	99009	74196
13	17	2014	3	Biblioteca Vapor Vell	295238	175356

Figura 22. Mostra dels primers registres de la taula *biblioteques*

Ara ja es pot afegir el codi necessari a l'API del servidor per demanar les dades d'aquesta taula.

3.1.2 Servidor

Amb referència al servidor, i com es va fer amb les dades de població, s'ha d'implementar un nou codi perquè l'API pugui servir les dades sobre les biblioteques. Aquest cop el desenvolupament és més senzill doncs no cal fer cap càlcul sobre totals. Es destaca a continuació el codi afegit a l'arxiu *routes.js*, on s'enregistra el codi de crides a l'API:

```
1.  // Libraries
2.  router.get('/libraries', function(request, response) {
3.      db.biblioteques.find({}, function(err, res) {
4.          if (err) {
5.              handleError(err);
6.          }
7.
8.              response.json({
9.                  data: res.map( (el) => {
10.
11.                 return {
12.                     id: el.id,
13.                     type: 'librarys',
14.                     attributes: {
15.                         year: el.anny,
16.                         district: el.districte,
17.                         libraryName: el.nom,
18.                         visits: el.visites,
19.                         loans: el.prestecs
20.                     }
21.                 }
22.
23.             })
24.         });
25.     });
26. });
```

3.1.3 Aplicació client

El següent punt a considerar és la vista del client. Es vol mostrar les visites a les biblioteques i els préstecs de llibres en aquestes, però també resulta interessant saber el seu nombre i com es reparteixen per la ciutat. Aquestes tres dades han d'anar separades doncs és poc recomanable mostrar-les totes de cop en el mateix mapa i en els diferents components gràfics amb el risc de fer-les il·legibles. En un principi es va considerar afegir un botó per poder escollir les dades a mostrar-se, però després de fer proves amb diferents usuaris va resultar molt millor oferir un menú desplegable com el que es mostra a la Figura 23.

Figura 23. Menú desplegable per les vistes de biblioteques

Seguidament es generen els arxius Ember necessaris –plantilla HBS, ruta i controlador- per cadascuna de les vistes a les que s'anomena *libraries*, *libraries-visits* i *libraries-loans* respectivament. El codi de la ruta és el mateix per tots tres: per una banda, es carrega al model les dades per dibuixar els districtes i la informació de les biblioteques. D'altra banda, un cop es carrega la informació, s'inicialitzen aquestes dades en el servei *data-service*, tal com es mostra a continuació:

```
model() {
    return Ember.RSVP.hash({
        libraries: ajax({
            url: '/api/v1/libraries',
            type: 'get'
        }),

        district: new Promise((res, rej) => {
          d3.json('assets/districtes.json', function(err, data) {
            err ? rej(err) : res(data);
          });
        }),

    });
},

afterModel: function(model, transition) {
    this.get('dataService').initLibraries(model.libraries.data);
},
```

Tot i que sembla un codi redundant s'ha de recordar que les dades al servei només es carregaran si no s'han inicialitzat abans. És a dir, es carregaran quan l'usuari entri per primer en cop en una vista amb dades de biblioteques i cap més. Així es millora l'eficiència de l'aplicació. En el servei de dades s'afegeix el codi necessari per utilitzar Crossfilter amb les noves dades i es declaren les noves dimensions:

```
1.  initLibraries(libraries) {
2.      if( isEmpty(this.get('libYearDim')) ) {
3.          const crossfilter = this.get('librariesCF');
4.          crossfilter.add(libraries);
5.
6.          const libYearDim = crossfilter.dimension( (d) => d.attributes.year);
7.          const years = libYearDim.group().all();
8.
9.          this.setProperties({
10.             libYearDim: libYearDim,
11.             libDistrictDimension: crossfilter.dimension( (d) => d.attributes.d
    istrict),
12.             libNameDimension: crossfilter.dimension( (d) => d.attributes.libra
    ryName),
13.             libYear: years[years.length - 1].key
14.         });
15.     }
16. },
17. libYearDim: null,
18. libDistrictDimension: null,
19. libNameDimension: null,
20. libYear: null,
```

Com es pot observar al codi anterior, es defineixen l'any, el districte i el nom de la biblioteca com dimensions de les dades (línies de la 17 a la 19). La dimensió del nom només s'utilitzarà per mostrar el nom de les biblioteques d'un districte. Per altra banda, es decideix crear una nova propietat *libYear* que emmagatzemarà l'últim any disponible a les dades i determinarà l'any seleccionat i filtrat a l'inici de la vista. Aquesta propietat serà compartida per totes les vistes amb dades sobre biblioteques. Així, si s'ha seleccionat, per exemple, l'any 2011 per veure les Visites, quan es canviï a la vista de Préstecs, es veuran les dades del mateix any. Per aquest motiu, s'afegeix també la propietat *populYear* per emmagatzemar el darrer any de les dades de població i que es podrà utilitzar en noves vistes sobre la població que s'afegeixin en un futur.

Una vegada el servei de dades està complet, ja es pot injectar en els diferents controladors de les vistes. En els següents subapartats s'explica les

decisions presses a l'hora d'escollir els elements que es mostren als usuaris en les tres vistes sobre dades de biblioteques.

Dades generals

Aquesta vista mostra la distribució de les biblioteques a la ciutat de Barcelona. En primer lloc, es reutilitza el component de mapa –com s'ha decidit que es farà en totes les vistes– però aquest cop es fa servir només la vista de districtes. En segon terme, es reutilitza el component *year-evolution* per mostrar l'evolució del nombre de biblioteques. Finalment, en la part inferior dreta de la vista, al no existir més dades importants dins d'aquest apartat que permetin mostrar un gràfic, es decideix mostrar el noms de les biblioteques que componen un districte. El resultat és la pantalla que es pot veure en la captura de la Figura 24.

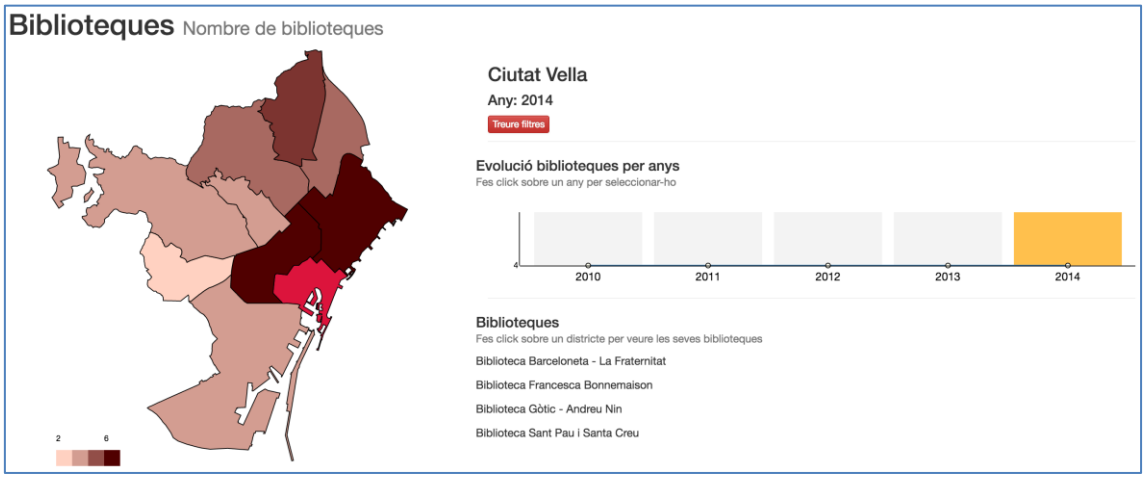


Figura 24. Vista de *Dades generals* de l'apartat *Biblioteques*

Visites i Préstecs

Les dos següents vistes de l'apartat Biblioteques s'han implementat amb els mateixos components. En primer terme, el component del mapa de la ciutat amb la vista per districtes; en segon lloc, la evolució anual de les visites o els préstecs, segons la pantalla; per últim, es reutilitza el component de diagrama de seccions. Aquest component es mostra una vegada es selecciona un districte i mostra les proporcions de visites –o préstecs– a les seves biblioteques, com es pot observar a la Figura 25.

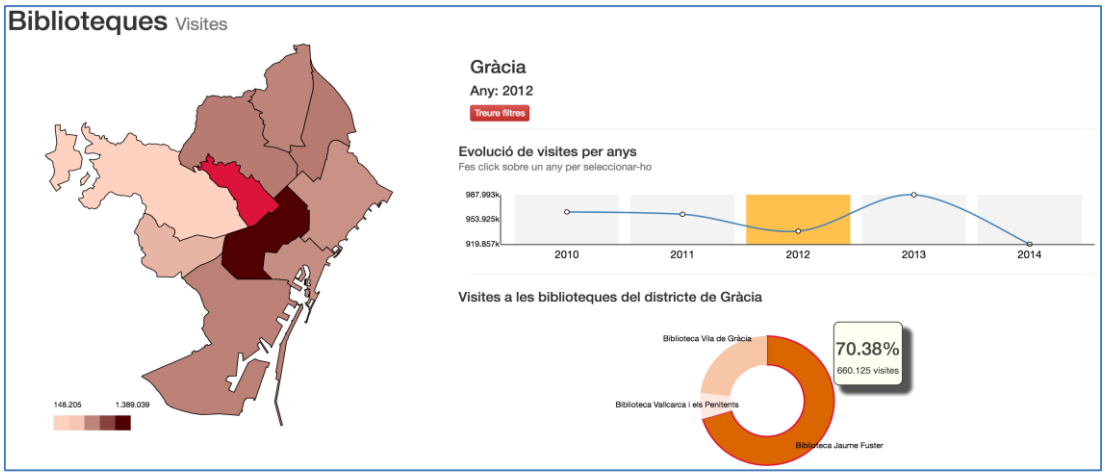


Figura 25. Vista de *Visites* a l'apartat *Biblioteques*

3.2 Nivell acadèmic

Com ja s'ha comentat abans, es volia implementar el major nombre de vistes sobre diferents serveis de Barcelona i, d'aquesta manera, creuar les dades en altres pantalles per poder fer comparatives. Malauradament, només les dades sobre biblioteques eren suficientment homogènies per aquest propòsit.

Encara que no es troben més dades per creuar amb les dades de població, es decideix cercar nova informació per ampliar aquest apartat. Amb els mateixos requisits d'homogeneïtat que abans, es troben les dades sobre el nivell acadèmic dels habitants de la ciutat. Es descarregarà les dades separades per gènere i així es podrà aprofitar per filtrar per aquest atribut. Un cop descarregats tots els arxius corresponents, es passa a fer el tractament de les dades.

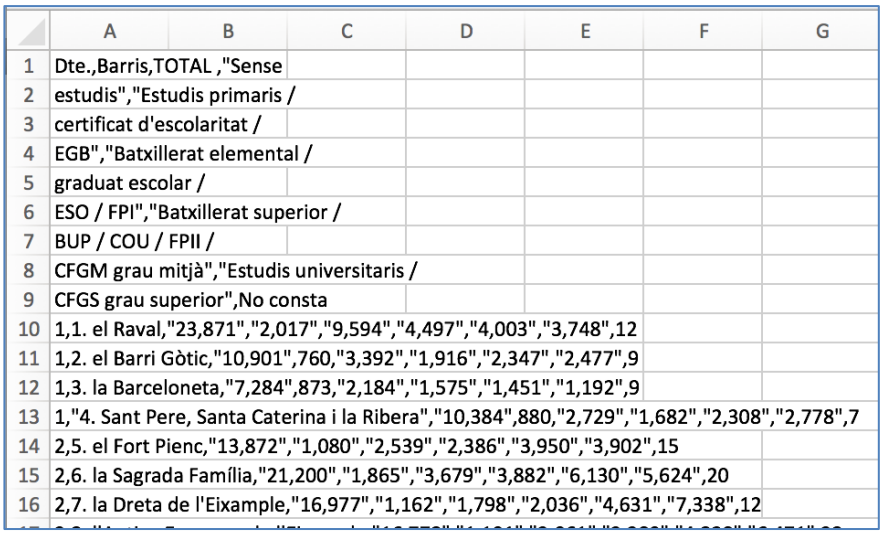
El primer pas que es realitza és comparar els totals de població de les noves dades amb els de la vista Població i s'obté un nombre inferior. Això és així perquè en la vista Població s'han obtingut les dades del padró i que una persona estigui empadronada a Barcelona no vol dir que visqui en ella necessàriament. Consegüentment, en aquesta nova vista es tindrà un nombre inferior de persones.

El segon pas es veure els atributs (columnes) de les que es disposa sobre el nivell acadèmic, que són:

- Sense estudis
- Estudis Primaris / Certificat d'Escolaritat / EGB
- Batxillerat Elemental / Graduat Escolar / ESO / FP I
- Batxillerat Superior / BUP / COU / FP II / CFGM Grau Mitjà
- Estudis Universitaris / CFGS Grau Superior
- No consta

Com es pot comprovar, coexisteixen dins dels mateixos atributs nivells de diferents sistemes educatius. Abans de crear la taula a la base de dades s'haurà de cerca un nom adient per cada atribut que resumeixi de forma equitativa tots els tipus d'estudis que es troben junts. En aquest projecte es descartarà la columna *No consta*. La primera raó es que no es pot determinar la classe d'estudis de les persones d'aquest atribut i, per tant, no és pertinent. La segona raó és que la quantitat de persones dins d'aquest atribut és molt petita en comparació amb el total. Com tampoc és té les dades del nivell acadèmic sobre les dades del padró, es descarten els valors d'aquest atribut al considerar-se com petits *outliers* (estranys) dins de l'estadística.

L'últim pas es comprovar les dades numèriques, tal com s'ha fet amb les dades de Biblioteques. Com ja s'ha comentat, es troben codificacions diferents en arxius corresponents a anys diferents. Així es té que, en anys més llunyans, la codificació dels nombres es troba entre cometes, com si fossin cadenes de text, i hi ha una coma com a separadors de milers (veure Figura 26).



	A	B	C	D	E	F	G
1	Dte.,Barris,TOTAL ,"Sense						
2	estudis","Estudis primaris /						
3	certificat d'escolaritat /						
4	EGB","Batxillerat elemental /						
5	graduat escolar /						
6	ESO / FPI","Batxillerat superior /						
7	BUP / COU / FPII /						
8	CFGM grau mitjà","Estudis universitaris /						
9	CFGS grau superior",No consta						
10	1,1. el Raval,"23,871","2,017","9,594","4,497","4,003","3,748",12						
11	1,2. el Barri Gòtic,"10,901",760,"3,392","1,916","2,347","2,477",9						
12	1,3. la Barceloneta,"7,284",873,"2,184","1,575","1,451","1,192",9						
13	1,"4. Sant Pere, Santa Caterina i la Ribera","10,384",880,"2,729","1,682","2,308","2,778",7						
14	2,5. el Fort Pienc,"13,872","1,080","2,539","2,386","3,950","3,902",15						
15	2,6. la Sagrada Família,"21,200","1,865","3,679","3,882","6,130","5,624",20						
16	2,7. la Dreta de l'Eixample,"16,977","1,162","1,798","2,036","4,631","7,338",12						

Figura 26. Dades de *Nivell acadèmic* sense tractar de l'any 2009

Després de separar les dades per columnes i fer servir les cometes com contenidors de cadenes de caràcters, es troben nombres separats per comes que es podrien considerar unitats i decimals. Des del principi del projecte s'ha optat per ometre els separadors de milers en les dades i utilitzar el punt com a separador de decimals. Per obtenir una homogeneïtat en aquestes dades, s'elimina la coma de tots els nombres on aparegui.

Per acabar, es fusionen totes les dades de tots els anys respectant l'identificador *any-districte-barri* i copiant primer les columnes per la població femenina i després la masculina abans de prendre una decisió sobre la creació de la taula.

3.2.1 Base de dades

Abans de crear la taula a la BD, s'ha de prendre una decisió sobre els noms dels atributs referents al nivell d'estudis. Aquests noms s'han de retallar per facilitar la seva manipulació dins del codi i, a més, s'ha d'afegir un sufix per identificar les dades segons el gènere. Per aquest fet, es decideix fer la següent conversió en els noms dels atributs:

- Sense estudis → **sense**
- Estudis Primaris / Certificat d'Escolaritat / EGB → **primaris**

- Batxillerat Elemental / Graduat Escolar / ESO / FP I → **secundaris**
- Batxillerat Superior / BUP / COU / FP II / CFGM Grau Mitjà → **mitjans**
- Estudis Universitaris / CFGS Grau Superior → **superiors**

A més, s'afegirà el sufix *Dones* o *Homes* on escaigui. Cal insistir en que aquests noms s'escullen per simplificar el desenvolupament. Una vegada es mostrin les dades a l'usuari s'haurà de decidir les seves etiquetes dins de la interfície.

Un cop determinat els noms dels atributs, la signatura de la taula *academic* a la base de dades és la següent:

academic: anny, districte, barri, senseDones, primarisDones, secundarisDones, mitjansDones, superiorsDones, primarisHomes, secundarisHomes, mitjansDones, superiorsHomes

Després d'executar les comandes necessàries a SQL per la creació de la taula[16], s'importen les dades des del fitxer CSV, tal com s'ha fet en punts anteriors, i es comprova la seva correcció mostrant les primeres files al SGBD, com es mostra a la Figura 27:

	id [PK] serial	anny integer	districte integer	barri character varying(50)	sensedones integer	primarisdones integer	secundarisdones integer	mitjansdones integer	superiorsdones integer	sensehomes integer	primarishomes integer	secundarishomes integer	mitjanshomes integer	superiorsH integer
1	1	2015	1	1. el Raval	880	5932	3732	3300	4563	443	9031	4966	3909	4455
2	2	2015	1	2. el Barri Gotic	268	1170	977	1515	2721	124	1567	1323	1713	2534
3	3	2015	1	3. la Barceloneta	553	1786	1444	1259	1839	244	1782	1726	1438	1560
4	4	2015	1	4. Sant Pere, Santa Caterina i la Ribera	470	2020	1464	2067	4002	190	2225	1807	2263	3480
5	5	2015	2	5. el Fort Pienc	571	2490	2690	3655	5462	290	1767	2341	3808	4637
6	6	2015	2	6. la Sagrada Familia	1125	4698	4728	5887	8576	539	2861	4103	6112	6992
7	7	2015	2	7. la Dreta de l'Eixample	615	2345	2716	5113	9882	279	1338	2106	4670	8727
8	8	2015	2	8. l'Antiga Esquerra de l'Eixample	647	2568	2898	5039	8817	313	1452	2352	4845	7884
9	9	2015	2	9. la Nova Esquerra de l'Eixample	953	4475	4704	6937	10664	444	2679	3922	7009	9341
10	10	2015	2	10. Sant Antoni	866	3598	3397	4159	6046	361	2557	3105	4570	5306

Figura 27. Primeres files de la taula *academic* a PostgreSQL

A continuació s'ha de crear la ruta de l'API en el costat del servidor per poder obtenir les dades d'aquesta nova taula de la BD.

[16] Veure *Creació taula academic* a l'*Annex III: Scripts SQL*.

3.2.2 Servidor

Com en dades anteriors, s'afegeix nou codi a l'arxiu *routes.js* per retornar les dades necessàries quan es cridi a la ruta *acadèmic* de l'API del servidor Node. En aquest cas, s'afegiran uns càlculs previs sobre els totals de la població separats per gèneres. D'aquesta manera, i tal com es va fer amb la vista Població, s'alleugera la càrrega de treball en el costat del client. Es decideix emprar una jerarquia de dades per separar les que pertanyen a dones i les que pertanyen a homes. El nou codi per l'API queda de la següent manera:

```
1.  // Academic levels
2.  router.get('/academic', function(request, response) {
3.      db.academic.find({}, function(err, res) {
4.          if (err) {
5.              handleError(err);
6.          }
7.
8.          response.json({
9.          data: res.map( (el) => {
10.          var womenTotal = el.sensedones + el.primarisdones + el.secundarisdones +
                        el.mitjansdones + el.superiorsdones;
11.          var menTotal = el.sensehomes + el.primarishomes + el.secundarishomes +
                    el.mitjanshomes + el.superiorshomes;
12.
13.                 return {
14.                     id: el.id,
15.                     type: 'academics',
16.                     attributes: {
17.                         year: el.anny,
18.                         district: el.districte,
19.                         neighbor: el.barri,
20.                         women: {
21.                             none: el.sensedones,
22.                             primary: el.primarisdones,
23.                             secondary: el.secundarisdones,
24.                             average: el.mitjansdones,
25.                             superior: el.superiorsdones,
26.                             total: womenTotal
27.                         },
28.                         men: {
29.                             none: el.sensehomes,
30.                             primary: el.primarishomes,
31.                             secondary: el.secundarishomes,
32.                             average: el.mitjanshomes,
33.                             superior: el.superiorshomes,
34.                             total: menTotal
35.                         }
36.                     }
37.                 };
38.             })
39.         });
40.     });
41.   });
```

3.2.3 Aplicació client

Pel que fa a la part del client d'aquestes noves dades, es prenen els mateixos passos que en vistes anteriors: es generen amb Ember la vista, la ruta i el controlador. Com les dades del nivell acadèmic formen part de la població, es decideix fer un menú desplegable sota el camp Població. Així, la vista Població que ja existia es troba sota el nom de 'dades del padró' i les noves sota el nom de 'nivell acadèmic'.

El següent pas és carregar les dades TopoJSON de districtes i barris, així com les dades del nivell acadèmic, a través del model a la ruta, i passar aquestes últimes al servei *data-service*. El codi del model i la crida a la inicialització del servei resta com:

```
model() {
    return Ember.RSVP.hash({
        academic: ajax({
            url: '/api/v1/academic',
            type: 'get'
        }),

        district: new Promise((res, rej) => {
          d3.json('assets/districtes.json', function(err, data) {
            err ? rej(err) : res(data);
          });
        }),

        neighbor: new Promise((res, rej) => {
          d3.json('assets/barris.json', function(err, data) {
            err ? rej(err) : res(data);
          });
        })
    });
},
```

A la part del servei de dades introduïm el següent tros de codi:

```
initAcademics(academics) {
   if ( isEmpty(this.get('acadYearDim')) ) {
       const crossfilter = this.get('academicsCF');
       crossfilter.add(academics);

       const acadYearDim = crossfilter.dimension( (d) => d.attributes.year);
       const years = acadYearDim.group().all();

       this.setProperties({
          acadYearDim: acadYearDim,
          acadDistrictDimension: Crossfilter
                 .dimension( (d) => d.attributes.district),
          acadNeighborDimension: Crossfilter
                 .dimension( (d) => +(d.attributes.neighbor)
                 .slice(0, (d.attributes.neighbor).indexOf('.')) ),
          acadYear: years[years.length - 1].key
       });
```

```
18.
19.     }
20. },
21. acadYearDim: null,
22. acadYear: null,
23. acadDistrictDimension: null,
24. acadNeighborDimension: null,
```

Com es pot veure, les dimensions escollides per filtrar les dades són l'any, el districte i el barri (línies de la 21 a la 24). També s'ha afegit un a propietat *acadYear* per inicialitzar la vista amb l'últim any present a les dades. Per filtrar per gènere es farà des del controlador del client.

Una vegada el servei de dades està complet, ja es pot injectar en els diferents controladors de les vistes. Es reutilitzarà els mateixos components que la vista de dades del padró menys el component de selector de rang. Es podria fer servir un selector de rang per escollir el nivell acadèmic de la població i mostrar alhora, per exemple, estudis secundaris i estudis mitjans. Però després de fer una consultar a experts en interfícies i algunes pedagogues, sembla més convenient poder triar només un nivell acadèmic per fer de filtre. A més, el component de selecció de rang funciona bé amb nombres però no tant amb atributs categòrics. Per això, es decideix crear un nou component D3 com un **gràfic de barres** per fer de selector del nivell acadèmic. El codi es pot trobar a l'aplicació Ember a la ruta ***components/bar-chart.js***. El gràfic s'inicialitza amb tots els nivells seleccionats i permet polsar sobre una barra per triar el nivell, tal com es mostra a la següent Figura 28:

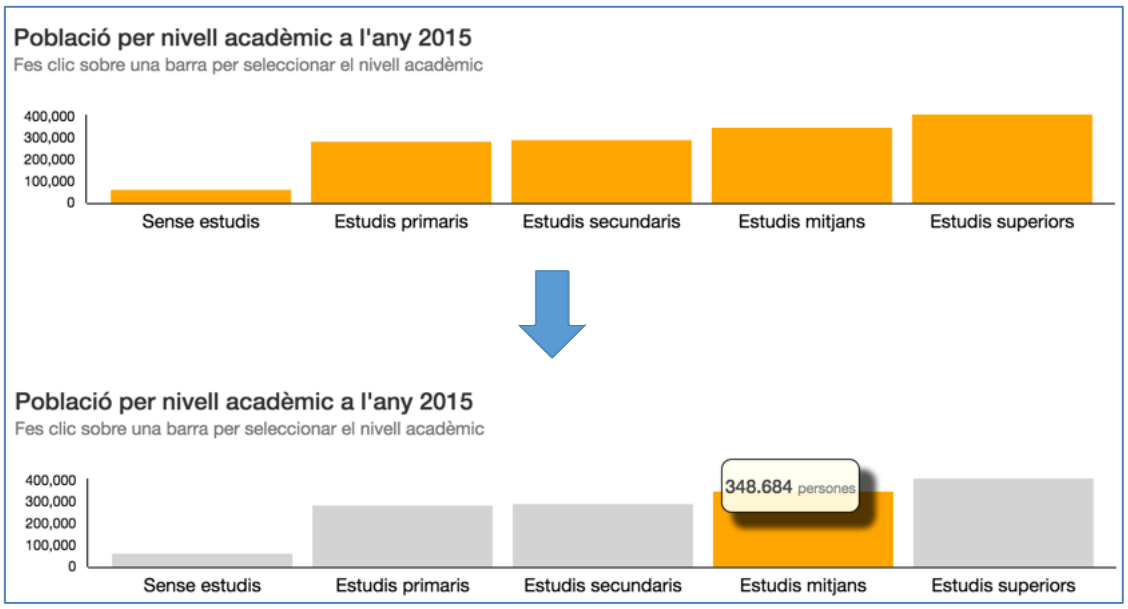


Figura 28. Component de gràfic de barres inicialitzat i amb selecció

Com es pot veure, el gràfic es comporta de manera similar al gràfic d'evolució: quan es filtrin o actualitzin les dades, també ho faran les barres i l'eix Y. A més, mostra el valor de la barra al passar el cursor per sobre. Les dades que es passen al component es trobar al controlador *academic.js* i a continuació es pot veure un tros del seu codi:

```
1.  yearDim.filter(year);
2.  aux = yearDim.top(Infinity);
3.  aux.forEach( (d, i) => {
4.      if (gender === 'Dones') {
5.          none += d.attributes.women.none;
6.          primary += d.attributes.women.primary;
7.          secondary +=  d.attributes.women.secondary;
8.          average += d.attributes.women.average;
9.          superior += d.attributes.women.superior;
10.     } else if (gender === 'Homes') {
11.         none += d.attributes.men.none;
12.         primary += d.attributes.men.primary;
13.         secondary += d.attributes.men.secondary;
14.         average += d.attributes.men.average;
15.         superior += d.attributes.men.superior;
16.     } else {
17.         none += d.attributes.women.none + d.attributes.men.none;
18.         primary += d.attributes.women.primary + d.attributes.men.primary;
19.         secondary += d.attributes.women.secondary + d.attributes.men.secondary
    ;
20.         average += d.attributes.women.average + d.attributes.men.average;
21.         superior += d.attributes.women.superior + d.attributes.men.superior;
22.     }
23.
24.     });
25.
26. data.pushObject({key: 'Sense estudis', value: none});
27. data.pushObject({key: 'Estudis primaris', value: primary});
28. data.pushObject({key: 'Estudis secundaris', value: secondary});
29. data.pushObject({key: 'Estudis mitjans', value: average});
30. data.pushObject({key: 'Estudis superiors', value: superior});
31.
32. return data;
```

Com es pot comprovar, primer es filtre per any i s'obtenen tots els nivells de l'any seleccionat. Per aquest cas, es millora a l'hora de filtrar per zones (districtes o barris) doncs aquesta dimensió es filtrarà un cop es reculli la selecció del component i així s'estalvien els condicionals a cada propietat computada. Després, per cada registre es calcula la suma del nombre de persones per nivell segons s'hagi seleccionat el filtre de sexe: dones, homes o tots dos si no ha triat cap.

Malgrat tot, aquest gràfic té un problema afegit: si la llegenda a l'eix X és massa llarga, s'encavalcarà i serà poc llegible. Com s'ha vist abans, s'ha tingut que substituir els noms llargs dels nivells acadèmics pels noms que s'han

proposat per la base de dades. Això pot ser un problema pels usuaris que utilitzin l'aplicació i es preguntin a què es refereix la vista quan es parla d'estudis secundaris o d'estudis mitjans.

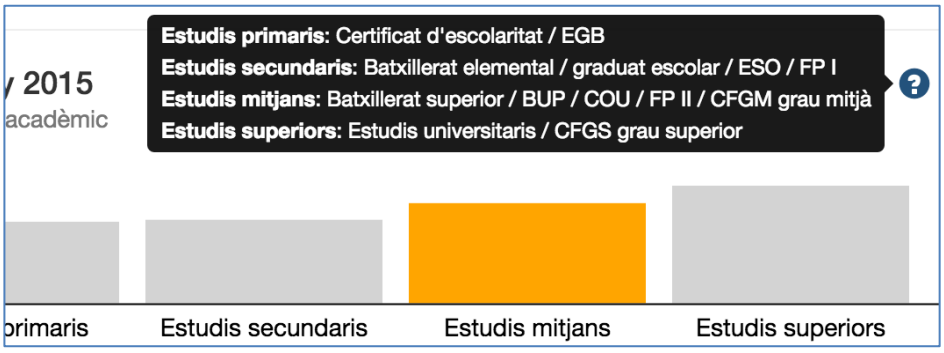


Figura 29. Finestra d'ajuda a la llegenda del gràfic

Per solucionar aquest conflicte es decideix crear una icona d'ajuda, com es pot observar a la Figura 29. Aquesta es col·loca a la part superior dreta del gràfic i, quan l'usuari passi el punter per sobre, mostrarà una finestra on s'indica els nivells d'estudis acadèmics englobats en cadascuna de les barres.

Un cop afegit el nou component i la petita icona d'ajuda, es dona per finalitzada la vista de dades sobre el nivell acadèmic que queda com es mostra a continuació en la Figura 30:

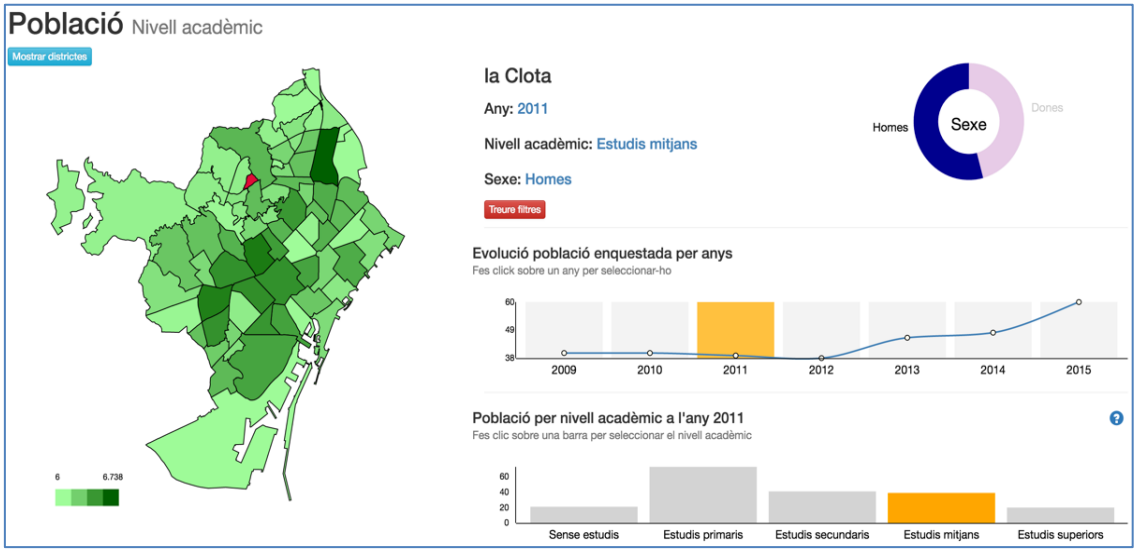


Figura 30. Vista de *Nivell acadèmic* amb filtres seleccionats

3.3 Dades creuades

El darrer pas en la implementació del projecte és crear vistes amb les dades creuades de les vistes ja implementades. Per fer això s'ha d'escollir molt bé quines són les dades que es poden creuar i quines no.

Atès les dades de les que es disposen, sembla clar que el primer creuament obvi és entre les visites a les biblioteques i el número de préstecs que tenen. En segon terme, es poden crear les dades de població –nombre d'habitants– per la quantitat de biblioteques, visites a aquestes i els préstecs de llibres. Per acabar, s'estudia creuar dades amb el nivell acadèmic. Aquestes dades ja formen part de la població, així que es descarta creuar-les amb les dades del padró. Tampoc es poden creuar amb les dades de les biblioteques perquè no es pot saber quins són els usuaris d'aquestes. Si a les visites o als préstecs de llibres es tingués una atribut de nivell acadèmic seria factible emprar-ho pel creuament. Però creuar les dades sense més informació donaria com a resultat un tauler esbiaixat i amb dades inconsistents. Dit això, es descarta creuar les dades de nivell acadèmic de la població amb cap altre.

3.3.1 Base de dades

Per portar a terme la implementació d'aquestes noves vistes de dades creuades no cal crear noves taules ni creuar les ja existents en una nova vista a la base de dades. En el cas d'aquest projecte, la manipulació és poc costosa i es pot fer en el costat del client.

Si es tingués en consideració una BD molt més complexa, segurament s'hauria d'implementar vistes de taula que, mitjançant disparadors (*triggers*), s'actualitzessin en el cas de que les taules d'origen canviessin les seves dades. Això és innecessari en el cas d'aquest projecte i, per tant, no s'afegirà res més.

3.3.2 Servidor

Tal com s'ha comentat al subapartat anterior, no es crearan més taules i les manipulacions es faran al costat del client. En conseqüència, no fa falta afegir cap codi nou a l'API del servidor.

3.3.3 Aplicació client

En el costat del client hem de repetir les mateixes passes donades en la implementació de vistes anteriors. Caldrà que les rutes carreguin els models necessaris que, en aquest cas, seran les dades de població i de biblioteques. La ruta més complicada que es pot trobar és aquella que necessiti carregar les dues taules de dades i inicialitzar-les al servei *data-service*, com es mostra a continuació:

```
model() {
    return Ember.RSVP.hash({
        population: ajax({
            url: '/api/v1/population',
            type: 'get'
        }),

        libraries: ajax({
            url: '/api/v1/libraries',
            type: 'get'
        }),

        district: new Promise((res, rej) => {
          d3.json('assets/districtes.json', function(err, data) {
            err ? rej(err) : res(data);
          });
        }),

    });
},

afterModel: function(model, transition) {
    this.get('dataService').initLibraries(model.libraries.data);
    this.get('dataService').initPopulation(model.population.data);
},
```

En la part del servei de dades no cal fer cap modificació doncs tots els tractaments amb Crossfilter i la definició de les dimensions pel filtratge de la informació ja s'han desenvolupament anteriorment. Cal insistir que en cas de que el servei ja tingui inicialitzades les dades, pel fet d'haver visitat una altra

vista que les necessités, no les tornarà a carregar a inicialitzar, i així s'alleugera la càrrega del client.

El següent pas es crear la vista de cada creuament de dades. Abans, s'afegeix un altre apartat a la barra de navegació amb el nom de Dades creuades. Sota aquest apartat aniran anidades totes les vistes que es comenten en el següents subapartats. Com a components gràfics es prendran com a base els que s'han fet servir en les vistes d'origen. Si en alguna vista no es pot emprar algun component o s'ha de canviar per un altre, es comentarà on escaigui. En totes les vistes només es mostraran els anys que estiguin disponibles en la intersecció de les dades.

Densitat de població per biblioteques

Vista similar al nombre de biblioteques. Es fa ús, com en totes, del mapa de dades i de l'evolució per anys. Es mostra també el nom de les biblioteques quan es selecciona un districte. S'utilitza el color grana per designar les zones on la quantitat d'habitants per biblioteca és molt alta i en verd on és més baixa, com es pot observar a la Figura 31.

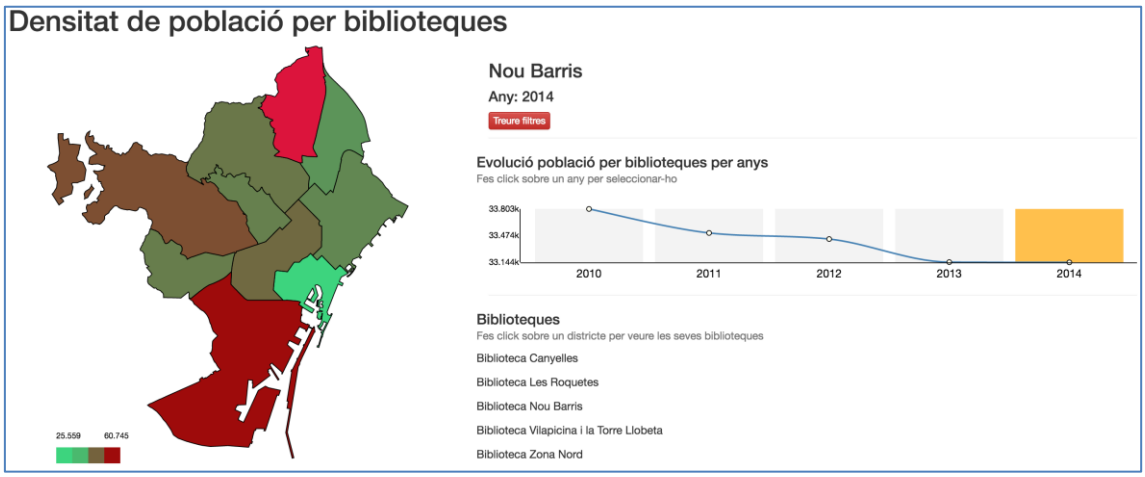


Figura 31. Vista de nombre d'habitants per biblioteques

Població i visites a biblioteques

Aquesta vista fa ús dels mateixos elements que a la vista del subapartat anterior. En aquest cas, els districtes amb els habitants que menys visiten les seves biblioteques s'omplen de gris fosc i aquells districtes on la seva població fa més visites s'acoloreixen de verd clar (veure Figura 32).

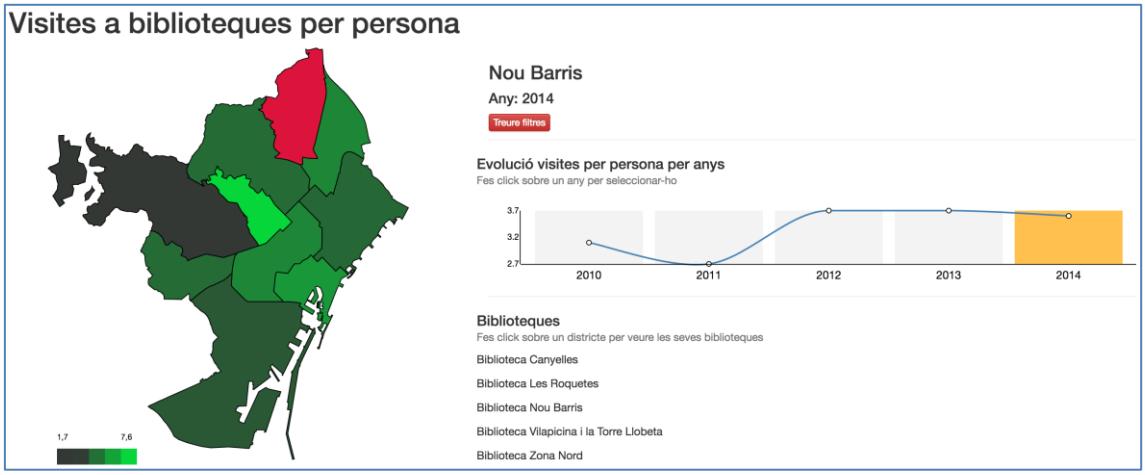


Figura 32. Vista de les visites a les biblioteques per habitant

Població i préstecs a les biblioteques

Igual que en les dues vistes anteriors, en aquesta plantilla s'utilitzen els mateixos elements. Les zones més fosques designen els districtes on la seva població realitza menys préstecs de llibres a les biblioteques mentre que les que s'omplen en blau clar són aquelles que els seus habitants demanen més préstecs, com es mostra a la següent Figura 33.

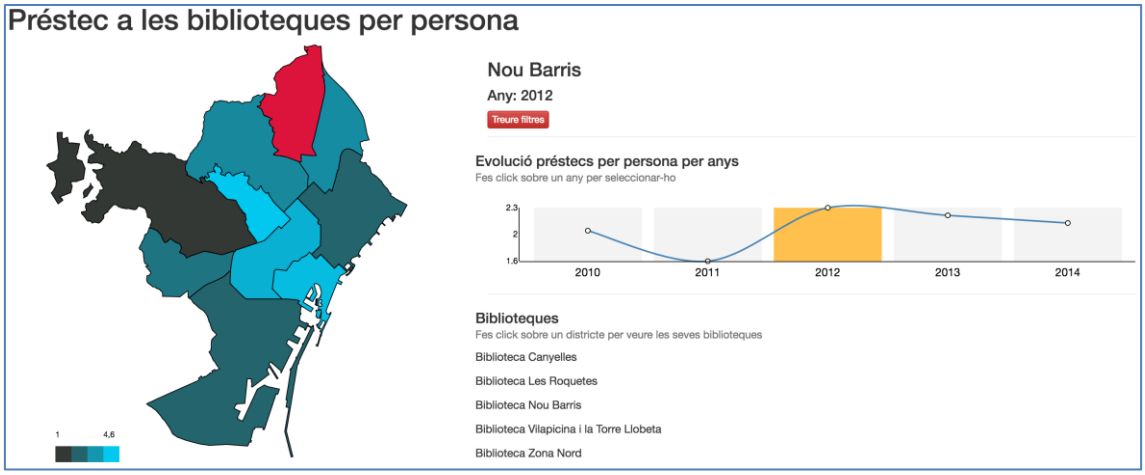


Figura 33. Vista dels préstecs a les biblioteques per habitant

Biblioteques: taxa de visites per préstec

En aquest cas, en comptes de llistar les biblioteques, es fa servir un diagrama de seccions per poder mostrar els ràtios de visites/préstecs de cada entitat quan es selecciona un districte. Per aquest cas particular, el color més clar correspon als districtes amb habitants que realitzen menys visites per préstec mentre que els de color més fosc en necessiten més. Es pot veure una captura d'aquesta darrera vista a la següent Figura 34:

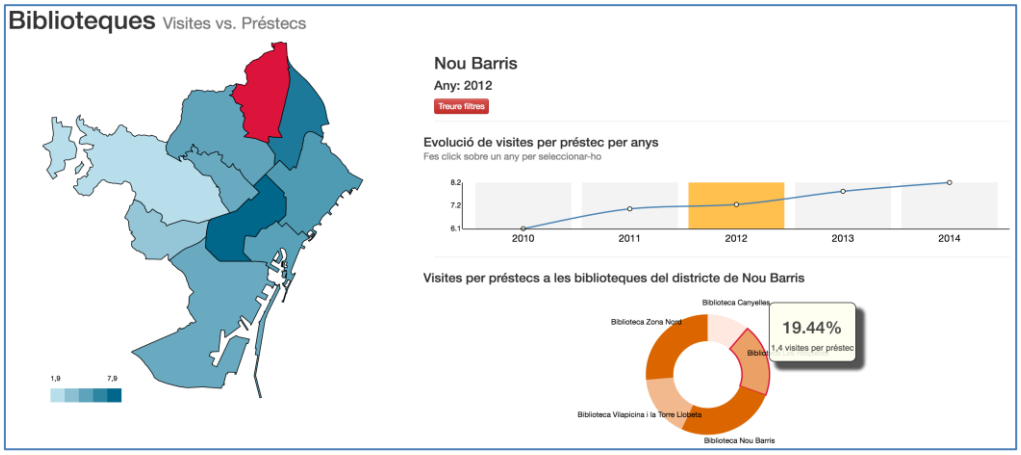


Figura 34. Vista de les visites per cada préstec a les biblioteques

4 Conclusions

Després de la finalització d'aquest projecte, es valora molt positivament l'experiència de crear una aplicació web completa o, dit d'una altra manera, d'exercir com desenvolupador *full-stack*. És la primera vegada que l'autor afronta un projecte d'aquestes dimensions i, tot i que de vegades el treball ha semblat una tasca titànica, la comesa ha estat molt enriquidora. Per altra banda, el temps disponible en el semestre per la implementació ha estat reduït i, en conseqüència, l'aplicació te menys vistes que les desitjades en un primer moment. Tot i això, el programari obtingut té una base de qualitat, és intuïtiu i pot ajudar a extreure coneixement i, en conseqüència, a la presa de decisions, com és l'objectiu de una aplicació de Business Intelligence.

Pel que fa a la planificació, es va haver de redefinir les tasques del temps destinat a implementar noves funcionalitats. Es va treure una tasca de documentació UML per la base de dades per ser innecessària un cop repensada la planificació i es va augmentar en 7 dies la finalització de la primera vista de població per deixar-la completa. Els objectius i la resta de tasques s'han complit sense gaires problemes, fins i tot després d'haver d'afegir un nou component gràfic com és el diagrama de barres. Pot ser, i per ser del tot honestos, es va trigar un dia més en seleccionar noves dades en el repositori. Això és així pels problemes que s'han comentat al llarg d'aquesta memòria amb la revisió de la homogeneïtat de les dades del repositori d'OpenDataBCN i la seva idoneïtat amb el projecte.

De manera paral·lela, es pot assegurar que els objectius que es plantejaven al principi del projecte han estat assolits. En primer lloc, s'ha obtingut una aplicació de qualitat, encara que reduïda, que serveix com una bona base per a la seva ampliació i millora. En segon lloc, s'ha complert amb el propòsit de posar a l'abast de tothom tant el codi com una versió funcional de l'aplicació. Finalment, també s'ha aconseguit l'objectiu de demostrar els coneixements adquirits al llarg del Grau en Enginyeria Informàtica. A més a més, ha estat una oportunitat d'adquirir-ne de nous a l'afrontar per primera vegada un desenvolupament complet d'una aplicació. Crear una API i convertir

gràfics en components són dos bons exemples de dificultats que s'han hagut de resoldre mitjançant estudis i anàlisi exhaustius, convertint-se en noves lliçons apreses.

La metodologia i les tecnologies utilitzades han demostrat ser les adequades. Podria haver estat més senzill fer servir JavaScript directament a la part de client en comptes de fer servir un marc de treball com Ember però aquest marc de treball permet obtenir una arquitectura de programari robust, modularitzat i amb patrons, tal com s'ha aprés a l'itinerari d'Enginyeria del Programari. Per altra banda, s'han creat components gràfics nous quan ha estat necessari mostrar dades que es podien haver quedat ocultes o per mostrar-les d'una altra manera.

Per acabar, aquest programari resta com una base perfecte per a desenvolupar un tauler de dades web més enriquit i amb més dades que permetin, tant a entitats públiques com privades –així com a habitants–, a la presa de decisions. Perquè la tasca d'ampliació o millora de l'aplicació sigui més senzilla seria bo que l'ajuntament de Barcelona adaptés el seu repositori de dades i creés una API pròpia com les que ja existeixen en altres institucions com, per exemple, el *Instituto Nacional de Estadística* de l'estat espanyol[17]. D'entre les possibles millores es destaquen, per un costat, la possibilitat d'afegir una finestra flotant que mostri les dades filtrades de la vista actual; per altre costat, afegir atributs de geo-posició d'algunes dades per posicionar-les de forma exacta sobre un mapa més detallat i veure, per exemple, les zones calentes d'accidents, multes o utilització de serveis.

[17] Es pot obtenir més informació sobre l'API JSON de l'INE a la direcció http://www.ine.es/dyngs/DataLab/manual.html?cid=45

5 Glossari

API *f* acrònim d'*Application Programming Interface*. Interfície que dona la possibilitat de comunicar-se amb serveis d'un tercer a través d'un enllaç d'Internet.

Array *m* tipus de dades estructurat que emmagatzema elements del mateix tipus, un darrera d'un altre, i que són accessibles mitjançant un índex.

Back-end *m* també conegut com capa d'accés a dades, és la part del programari que processa l'entrada del *Front-end*.

Booleà *m* Tipus de dada que només pot prendre els valors cert (*true*) o fals (*false*).

Business Intelligence *f* nom anglès per designar la intel·ligència de negoci o empresarial. Es refereix a la capacitat de transformar dades en informació i aquesta en coneixement pel procés de presa de decisions.

Callback *f* funció que es passa com a paràmetre d'una altra i que aquesta darrera executa.

Codi obert *m* programari que pot ser distribuït i desenvolupat lliurement.

CSS *f* sigles de *Cascading Style Sheets*, llenguatge per definir l'estil de les pàgines escrites en HTML.

CSV *m* acrònim de l'anglès *Comma Separated Values*. Format senzill per emmagatzemar dades en forma de taula amb les columnes separades per comes (o punt i comes).

Disseny responsiu *m* disseny que 'respon' i s'adapta a la mida del navegador.

Exabyte *m* unitat d'emmagatzematge equivalent a 10^{18} bytes o 1024 Petabytes.

Framework *m* veure Marc de treball.

Front-end *m* part del programari que interactua amb l'usuari. També es pot conèixer com capa de presentació.

Full-stack **(desenvolupador)** *m* Terme que es tradueix com 'pila completa', es refereix al programador que s'encarrega de desenvolupar tant la part de *Front-end* com de *Back-end*.

HTML *m* sigles de *HyperText Markup Language*. Llenguatge de marcat per estructurar textos.

JavaScript *m* llenguatge de programació interpretat, com a part dels navegadors web

JSON *f* acrònim de *JavaScript Object Notation*, una sintaxi lleugera i senzilla per emmagatzemar i intercanviar dades.

Marc de treball *m* entorn de treball per desenvolupament que acostuma a integrar components que ho faciliten.

Maquinari *m* traducció al català de la paraula anglesa *Hardware* que denota el conjunt d'elements i materials físics que constitueixen una computadora o ordinador.

Middleware *m* o capa intermèdia, és el programari que ajuda a una aplicació a comunicar-se amb altres aplicacions, components, paquets, xarxes, maquinari o sistemes operatius

Model-Vista-Controlador (MVC) *m* patró d'arquitectura de programari que separa les dades i la lògica de negoci d'una aplicació de la interfície d'usuari.

OCL *m* sigles d'*Object Constraint Language.* Llenguatge per la descripció formal d'expressions i restriccions sobre un model UML.

Open Source *m* veure Codi obert.

Petabyte *m* unitat d'emmagatzematge equivalent a 10^{15} bytes o 1024 Terabytes.

Programari *m* traducció al català de la paraula anglesa *Software* que descriu el conjunt de programes informàtics.

Repositori *m* lloc on s'emmagatzemen dades, codi, imatges, etc.

Responsive Design *m* veure Disseny Responsiu.

REST *m* acrònim de *Representational State Transfer*, estil d'arquitectura de programari per sistemes hipermèdia normalment representats per HTML o XML.

Singleton *m* aquest patró de disseny restringeix la creació dels objectes d'una classe d'un tipus a un únic objecte.

SQL *m* sigles de *Structured Query Language*, llenguatge estàndard per la definició, manipulació, control i consulta de bases de dades.

SVG *m* sigles de l'anglès *Scalable Vector Graphics*. Són gràfics vectorials escalables oberts i basats en gràfics XML.

UML *m* acrònim d'*Unified Modeling Language*, estàndard internacional per crear diagrames i documentació pel desenvolupament de programari.

URL *m* sigles d'*Uniform Resource Locator.* Seqüència de caràcters que s'utilitza per localitzar un recurs a Internet. Conegut de forma comú com direcció o enllaç web.

XML *m* acrònim d'*eXtensible Markup Language*. Es tracta d'un llenguatge de marques per emmagatzemar dades de forma llegible.

6 Bibliografia

[1] (2011, 21 d'octubre) "Every Day We Create 2.5 Quintillion Bytes of Data". *StorageNewsletter.com* [article en línia]. [Data de consulta: 7 de febrer de 2016].
http://www.storagenewsletter.com/rubriques/market-reportsresearch/ibm-cmo-study/

[2] (2016, 1 de febrer) "Petabyte*". Wikipedia, la enciclopedia libre* [article en línia]. [Data de consulta: 7 de febrer de 2016].
https://es.wikipedia.org/wiki/Petabyte

[3] **Davenport, T. H.; Patil, D.J.** (2012, octubre) "Data Scientist: The Sexiest Job of the 21st Century". *Hardvard Business Review* [article en línia]. [Data de consulta: 7 feb. 2016].
https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century

[4] **Padilla, Marta**. (2012, 13 de febrer) "Kanban". *The Project THP* [article en línia] [Data de consulta: 9 de febrer 2016]
http://www.theproject.ws/es/node/840

[5] (2016, 5 de març) "Lista de licencias con comentarios". *El sistema operativo GNU* [article en línia] [Data de consulta: 11 feb. 2016]
http://www.gnu.org/licenses/license-list.ca.html

[6] **Connery, Rob**. (2016, 17 d'abril) "Massive.js - Massive 2.0: A Postgres-centric Data Access Tool". [article en línia] [Data de consulta: 19 de març de 2016] http://github.com/robconery/massive-js/

[7] **Hanchett, Erik**. (2015, 6 d'abril) "How To Setup Your Ember Project With Node or io.js And Express". *ProgramWithErik.com* [article en línia] [Data de consulta: 21 de març de 2016]
http://www.programwitherik.com/setup-your-ember-project-with-node/

[8] **Anton, Cesar**. (2015, 20 de juliol) "Cómo crear un API JSON en PostgreSQL con NodeJS" *Platzi.com* [article en línia] [Data de consulta: 21 de març de 2016) https://platzi.com/blog/postgresql-nodejs/

[9] (2015, 1 de gener) "Roll your own Node.js API for Ember" *TheTechCoufounder.com* [article en línia] [Data de consulta: 22 de març de 2016] http://thetechcofounder.com/roll-your-own-node-js-api-for-ember/

[10] **Bostock, Mike**. (2012, 30 de desembre) "Let's Make a Map". *Mike Bostock's blog* [article en línia] [Data de consulta: 25 de març de 2016] https://bost.ocks.org/mike/map/

[11] **González, Martín**. (2015, 19 de maig) "Barcelona Geodata". *Github.com* [article en línia] [Data de consulta: 25 de març de 2016] https://github.com/martgnz/bcn-geodata

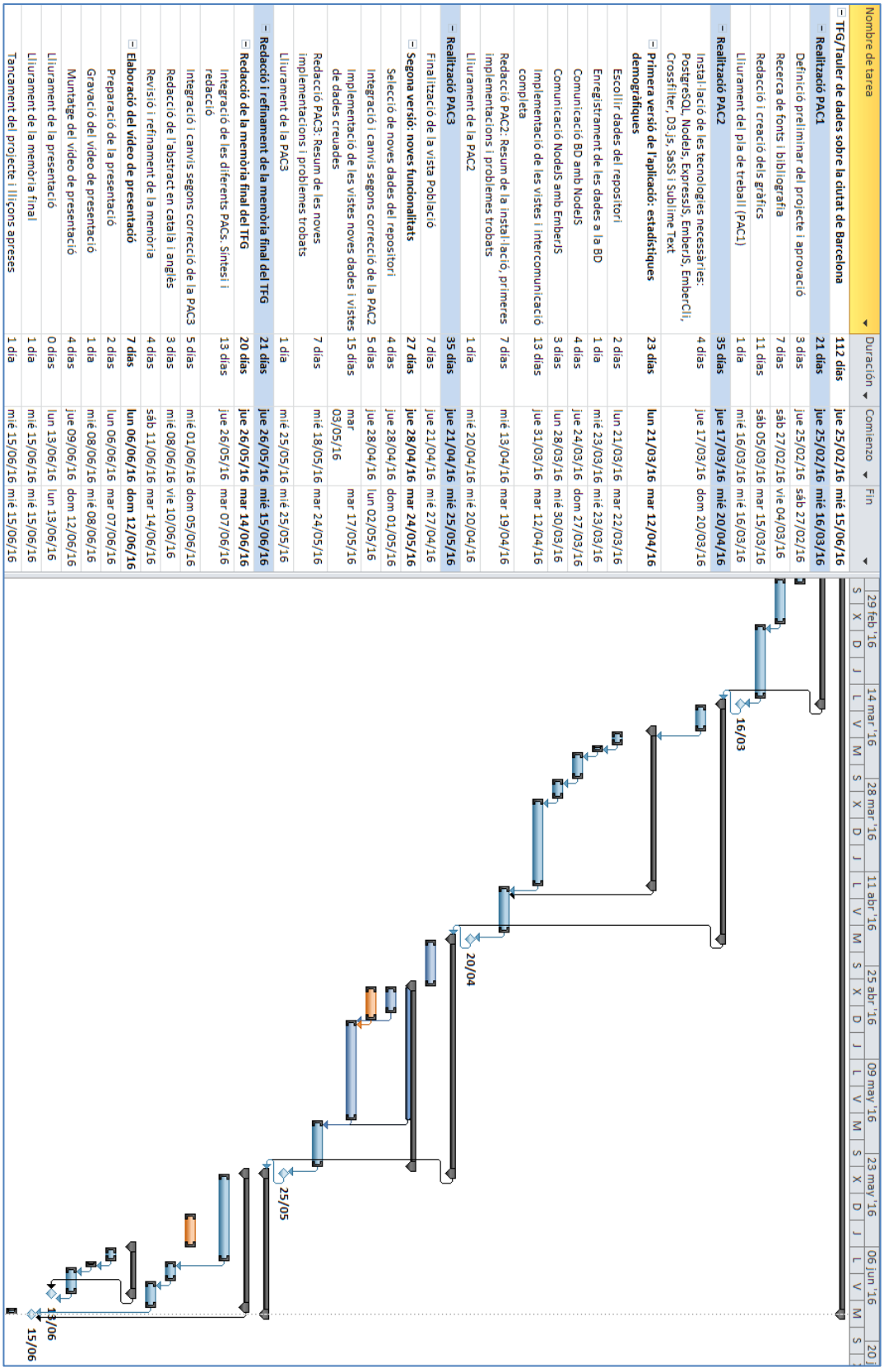
[12] **Bostock, Mike**. (2016, 1 de març) "D3 wiki" *Github.com* [article en línia] [Data de consulta: 25 de març de 2016] https://bost.ocks.org/mike/map/

[13] (2016, 15 d'abril) "Ember API". *EmberJS.com* [article en línia] [Data de consulta: 16 d'abril de 2016] http://emberjs.com/api/

[14] (2014, 21 de setembre) "Crossfilter API Reference". *Square Inc.* [article en línia] [Data de consulta: 28 de març de 2016] https://github.com/square/crossfilter/wiki/API-Reference

7 Annexos

Annex I: Planificació del projecte



Nombre de tarea	Duración	Comienzo	Fin
TFG/Tauler de dades sobre la ciutat de Barcelona	**112 días**	**jue 25/02/16**	**mié 15/06/16**
Realització PAC1	**21 días**	**jue 25/02/16**	**mié 16/03/16**
Definició preliminar del projecte i aprovació	3 días	jue 25/02/16	sáb 27/02/16
Recerca de fonts i bibliografia	7 días	sáb 27/02/16	vie 04/03/16
Redacció i creació dels gràfics	11 días	sáb 05/03/16	mar 15/03/16
Lliurament del pla de treball (PAC1)	1 día	mié 16/03/16	mié 16/03/16
Realització PAC2	**35 días**	**jue 17/03/16**	**mié 20/04/16**
Instal·lació de les tecnologies necessàries: PostgreSQL, NodeJs, ExpressJS, EmberJS, EmberCli, Crossfilter, D3.js, SaSS i Sublime Text	4 días	jue 17/03/16	dom 20/03/16
Primera versió de l'aplicació: estadístiques demogràfiques	**23 días**	**lun 21/03/16**	**mar 12/04/16**
Escollir dades del repositori	2 días	lun 21/03/16	mar 22/03/16
Enregistrament de les dades a la BD	1 día	mié 23/03/16	mié 23/03/16
Comunicació BD amb NodeJS	4 días	jue 24/03/16	dom 27/03/16
Comunicació NodeJS amb EmberJS	3 días	lun 28/03/16	mié 30/03/16
Implementació de les vistes i intercomunicació completa	13 días	jue 31/03/16	mar 12/04/16
Redacció PAC2: Resum de la instal·lació, primeres implementacions i problemes trobats	7 días	mié 13/04/16	mar 19/04/16
Lliurament de la PAC2	1 día	mié 20/04/16	mié 20/04/16
Realització PAC3	**35 días**	**jue 21/04/16**	**mié 25/05/16**
Finalització de la vista Població	7 días	jue 21/04/16	mié 27/04/16
Segona versió: noves funcionalitats	**27 días**	**jue 28/04/16**	**mar 24/05/16**
Selecció de noves dades del repositori	4 días	jue 28/04/16	dom 01/05/16
Integració i canvis segons correcció de la PAC2	5 días	jue 28/04/16	lun 02/05/16
Implementació de les vistes noves dades i vistes de dades creuades	15 días	mar 03/05/16	mar 17/05/16
Redacció PAC3: Resum de les noves implementacions i problemes trobats	7 días	mié 18/05/16	mar 24/05/16
Lliurament de la PAC3	1 día	mié 25/05/16	mié 25/05/16
Redacció i refinament de la memòria final del TFG	**21 días**	**jue 26/05/16**	**mié 15/06/16**
Redacció de la memòria final del TFG	**20 días**	**jue 26/05/16**	**mar 14/06/16**
Integració de les diferents PACs. Síntesi i redacció	13 días	jue 26/05/16	mar 07/06/16
Integració i canvis segons correcció de la PAC3	5 días	mié 01/06/16	dom 05/06/16
Redacció de l'abstract en català i anglès	3 días	mié 08/06/16	vie 10/06/16
Revisió i refinament de la memòria	4 días	sáb 11/06/16	mar 14/06/16
Elaboració del vídeo de presentació	**7 días**	**lun 06/06/16**	**dom 12/06/16**
Preparació de la presentació	2 días	lun 06/06/16	mar 07/06/16
Gravació del vídeo de presentació	1 día	mié 08/06/16	mié 08/06/16
Muntatge del vídeo de presentació	4 días	jue 09/06/16	dom 12/06/16
Lliurament de la presentació	0 días	lun 13/06/16	lun 13/06/16
Lliurament de la memòria final	1 día	mié 15/06/16	mié 15/06/16
Tancament del projecte i lliçons apreses	1 día	mié 15/06/16	mié 15/06/16

Annex II: Instal·lació de tecnologies de treball

Còpia del codi

Els sistemes OSX i Linux venen amb Git per defecte. Per saber si es té instal·lat, cal executar la següent comanda a un terminal:

- `git –version`

Si no surt res, cal instal·lar Git des de la seva pàgina web a la direcció git-scm.com. Si es vol copiar el codi en una carpeta, s'ha d'executar la següent comanda en un terminal dins de la carpeta de destí:

- `git clone https://github.com/ElXaxe/Barcelona-Dashboard.git`

Base de dades: PostgreSQL

La instal·lació s'ha de fer des de la pàgina web de PostgreSQL. Una vegada instal·lat, s'ha de crear la taula amb l'*script* SQL i després importar les dades de l'arxiu *poblacio.csv* de la carpeta *database*.

Per facilitar l'ús de PostgreSQL es pot optar per instal·lar la interfície gràfica pgAdmin[18]. Amb aquesta aplicació, per executar l'*script*, s'ha de fer clic a la icona de la lupa amb la paraula SQL. En la nova finestra es pot copiar el codi de l'*script* i executar-ho amb el botó de triangle verd, tal com es mostra en la Figura 35.

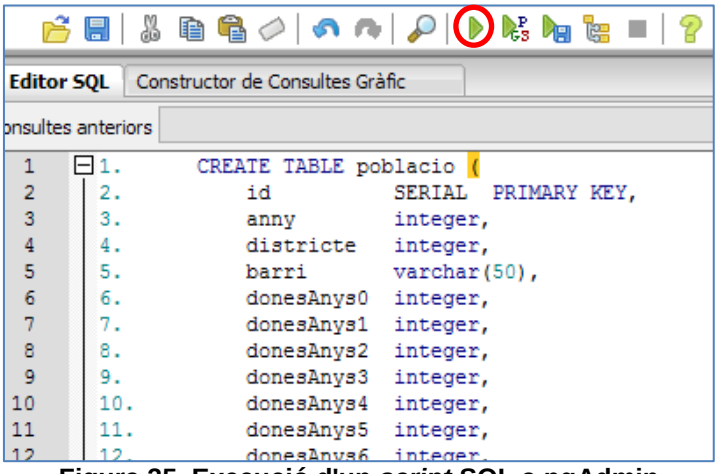


Figura 35. Execució d'un *script* SQL a pgAdmin

[18] Més informació a https://www.pgadmin.org/

Una vegada creada, es clica sobre el botó dret en la **base de dades → esquema → public → taules** i es clica en **Refresca**. Sortirà la nova taula on es torna a clicar amb el botó dret i es triar **Importar**. S'escull l'arxiu CSV que es vol amb el botó **Browse**, en **Format** s'escull CSV i a la pestanya **Opcions Misc.** s'escull el punt i coma com a **Delimitador**, marcant la casella de **Capçalera**, tal com es mostra a la Figura 36.

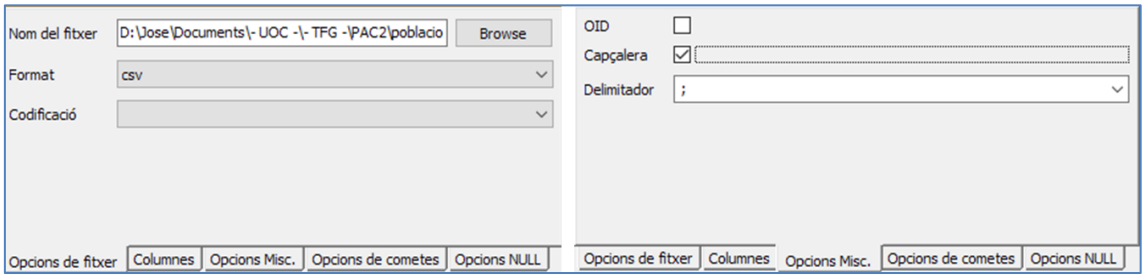


Figura 36. Opcions per importar un arxiu CSV a una taula de PostgreSQL

Servidor: Node.js i Express

En el cas de tenir OS X com a sistema operatiu, es recomanable emprar Homebrew per instal·lar Node.js. En altres casos, es pot seguir un tutorial adient en la pàgina web Node.js.

Un cop instal·lat Node, s'han de resoldre totes les dependències que té, executant la següent comanda des de l'arrel on s'ha copiat el projecte:

- `npm install`

Amb això s'instal·larà Express, Massive i Body-parser. Ara cal modificar la configuració del servidor perquè es connecti a la base de dades que s'ha configurat a la maquina on es treballi. Les línies a configurar són les marcades en vermell, corresponents a la base de dades, usuari i contrasenya, respectivament:

```
{
    "express" : {
        "port"      : 80
    },

    "postgres": {
        "db"        : "test",
        "user"      : "ElXaxe",
        "password"  : "barcelona",
        "host"      : "localhost",
        "port"      : 5432
    }
}
```

Client: Ember.js i altres llibreries

Per a instal·lar Ember, s'ha d'executar la següent comanda des de l'arrel del projecte:

- `npm install -g ember-cli`

El codi de l'aplicació de client es trobar a la carpeta *client-app*, on s'ha d'entrar i executar les següents comandes:

- `npm install`
- `bower install`

Després d'una bona estona, s'instal·laran totes les dependències: jQuery, Bootstrap, TopoJson, D3, Crossfilter, etc.

Per instal·lar Sass dependrà del sistema operatiu on es treballi. Es per això que és recomanable seguir les instruccions de la seva pàgina web.

Editor opcional: Sublime Text

Es recomanable emprar un editor de codi que permeti identificar visualment els diferents elements del codi. Sublime Text es pot descarregar des de la seva pàgina web i afegir els llenguatges necessaris, com es pot veure a la Figura 37.

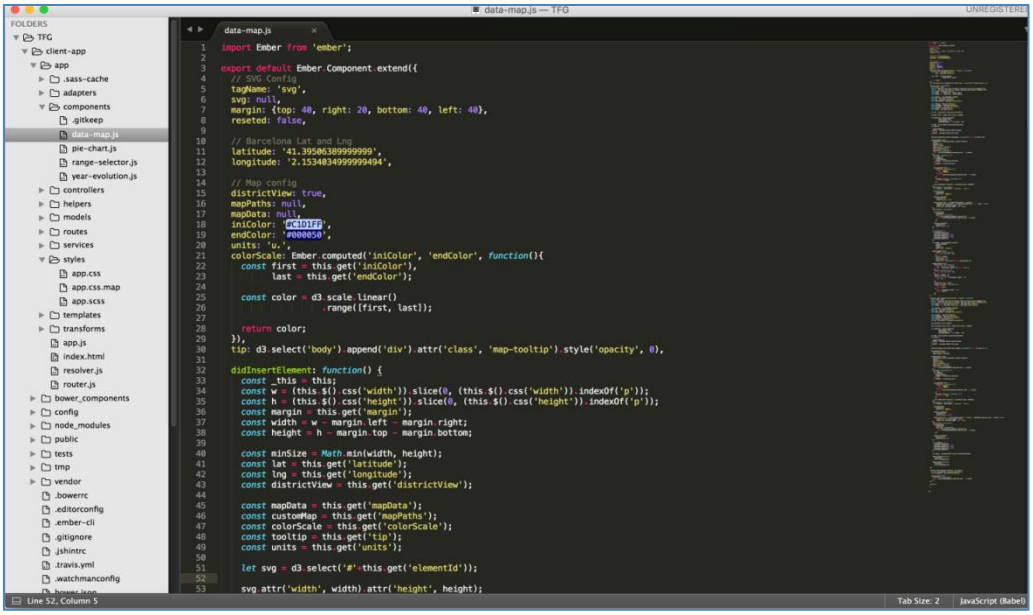


Figura 37. Vista del codi a Sublime Text

Annex III: Scripts SQL

Creació taula poblacio

```sql
CREATE TABLE poblacio (
    id          SERIAL  PRIMARY KEY,
    anny        integer,
    districte   integer,
    barri       varchar(50),
    donesAnys0  integer,
    donesAnys1  integer,
    donesAnys2  integer,
    donesAnys3  integer,
    donesAnys4  integer,
    donesAnys5  integer,
    donesAnys6  integer,
    donesAnys7  integer,
    donesAnys8  integer,
    donesAnys9  integer,
    donesAnys10 integer,
    donesAnys11 integer,
    donesAnys12 integer,
    donesAnys13 integer,
    donesAnys14 integer,
    donesAnys15 integer,
    donesAnys16 integer,
    donesAnys17 integer,
    donesAnys18 integer,
    donesAnys19 integer,
    donesAnys20 integer,
    donesAnys21 integer,
    donesAnys22 integer,
    donesAnys23 integer,
    donesAnys24 integer,
    donesAnys25 integer,
    donesAnys26 integer,
    donesAnys27 integer,
    donesAnys28 integer,
    donesAnys29 integer,
    donesAnys30 integer,
    donesAnys31 integer,
    donesAnys32 integer,
    donesAnys33 integer,
    donesAnys34 integer,
    donesAnys35 integer,
    donesAnys36 integer,
    donesAnys37 integer,
    donesAnys38 integer,
    donesAnys39 integer,
    donesAnys40 integer,
    donesAnys41 integer,
    donesAnys42 integer,
    donesAnys43 integer,
    donesAnys44 integer,
    donesAnys45 integer,
    donesAnys46 integer,
    donesAnys47 integer,
    donesAnys48 integer,
    donesAnys49 integer,
    donesAnys50 integer,
    donesAnys51 integer,
    donesAnys52 integer,
    donesAnys53 integer,
```

```
60.     donesAnys54 integer,
61.     donesAnys55 integer,
62.     donesAnys56 integer,
63.     donesAnys57 integer,
64.     donesAnys58 integer,
65.     donesAnys59 integer,
66.     donesAnys60 integer,
67.     donesAnys61 integer,
68.     donesAnys62 integer,
69.     donesAnys63 integer,
70.     donesAnys64 integer,
71.     donesAnys65 integer,
72.     donesAnys66 integer,
73.     donesAnys67 integer,
74.     donesAnys68 integer,
75.     donesAnys69 integer,
76.     donesAnys70 integer,
77.     donesAnys71 integer,
78.     donesAnys72 integer,
79.     donesAnys73 integer,
80.     donesAnys74 integer,
81.     donesAnys75 integer,
82.     donesAnys76 integer,
83.     donesAnys77 integer,
84.     donesAnys78 integer,
85.     donesAnys79 integer,
86.     donesAnys80 integer,
87.     donesAnys81 integer,
88.     donesAnys82 integer,
89.     donesAnys83 integer,
90.     donesAnys84 integer,
91.     donesAnys85 integer,
92.     donesAnys86 integer,
93.     donesAnys87 integer,
94.     donesAnys88 integer,
95.     donesAnys89 integer,
96.     donesAnys90 integer,
97.     donesAnys91 integer,
98.     donesAnys92 integer,
99.     donesAnys93 integer,
100.            donesAnys94 integer,
101.            donesAnys95 integer,
102.            homesAnys0  integer,
103.            homesAnys1  integer,
104.            homesAnys2  integer,
105.            homesAnys3  integer,
106.            homesAnys4  integer,
107.            homesAnys5  integer,
108.            homesAnys6  integer,
109.            homesAnys7  integer,
110.            homesAnys8  integer,
111.            homesAnys9  integer,
112.            homesAnys10 integer,
113.            homesAnys11 integer,
114.            homesAnys12 integer,
115.            homesAnys13 integer,
116.            homesAnys14 integer,
117.            homesAnys15 integer,
118.            homesAnys16 integer,
119.            homesAnys17 integer,
120.            homesAnys18 integer,
121.            homesAnys19 integer,
122.            homesAnys20 integer,
123.            homesAnys21 integer,
124.            homesAnys22 integer,
125.            homesAnys23 integer,
```

```
126.            homesAnys24 integer,
127.            homesAnys25 integer,
128.            homesAnys26 integer,
129.            homesAnys27 integer,
130.            homesAnys28 integer,
131.            homesAnys29 integer,
132.            homesAnys30 integer,
133.            homesAnys31 integer,
134.            homesAnys32 integer,
135.            homesAnys33 integer,
136.            homesAnys34 integer,
137.            homesAnys35 integer,
138.            homesAnys36 integer,
139.            homesAnys37 integer,
140.            homesAnys38 integer,
141.            homesAnys39 integer,
142.            homesAnys40 integer,
143.            homesAnys41 integer,
144.            homesAnys42 integer,
145.            homesAnys43 integer,
146.            homesAnys44 integer,
147.            homesAnys45 integer,
148.            homesAnys46 integer,
149.            homesAnys47 integer,
150.            homesAnys48 integer,
151.            homesAnys49 integer,
152.            homesAnys50 integer,
153.            homesAnys51 integer,
154.            homesAnys52 integer,
155.            homesAnys53 integer,
156.            homesAnys54 integer,
157.            homesAnys55 integer,
158.            homesAnys56 integer,
159.            homesAnys57 integer,
160.            homesAnys58 integer,
161.            homesAnys59 integer,
162.            homesAnys60 integer,
163.            homesAnys61 integer,
164.            homesAnys62 integer,
165.            homesAnys63 integer,
166.            homesAnys64 integer,
167.            homesAnys65 integer,
168.            homesAnys66 integer,
169.            homesAnys67 integer,
170.            homesAnys68 integer,
171.            homesAnys69 integer,
172.            homesAnys70 integer,
173.            homesAnys71 integer,
174.            homesAnys72 integer,
175.            homesAnys73 integer,
176.            homesAnys74 integer,
177.            homesAnys75 integer,
178.            homesAnys76 integer,
179.            homesAnys77 integer,
180.            homesAnys78 integer,
181.            homesAnys79 integer,
182.            homesAnys80 integer,
183.            homesAnys81 integer,
184.            homesAnys82 integer,
185.            homesAnys83 integer,
186.            homesAnys84 integer,
187.            homesAnys85 integer,
188.            homesAnys86 integer,
189.            homesAnys87 integer,
190.            homesAnys88 integer,
191.            homesAnys89 integer,
```

```
            homesAnys90 integer,
            homesAnys91 integer,
            homesAnys92 integer,
            homesAnys93 integer,
            homesAnys94 integer,
            homesAnys95 integer
        );
```

Creació taula biblioteques

```
CREATE TABLE biblioteques (
    id          SERIAL  PRIMARY KEY,
    anny        integer,
    districte   integer,
    nom         varchar(50),
    visites     integer,
    prestecs    integer
);
```

Creació taula academic

```
CREATE TABLE academic (
    id          SERIAL  PRIMARY KEY,
    anny            integer,
    districte       integer,
    barri           varchar(50),
    senseDones      integer,
    primarisDones   integer,
    secundarisDones integer,
    mitjansDones    integer,
    superiorsDones  integer,
    senseHomes      integer,
    primarisHomes   integer,
    secundarisHomes integer,
    mitjansHomes    integer,
    superiorsHomes  integer
);
```