Universitat de les Illes Balears

escola politècnica superior

Treball Final de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Gestió i control intern de dades empresarials

ALEXANDRE RODRÍGUEZ GARAU

Tutor
Miquel Mascaró Portells

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 1 de setembre de 2017

SUMARI

ÍNDEX DE FIGURES

LISTINGS

ACRÒNIMS

UIB Universitat de les Illes Balears

TFG Treball de Final de Grau

D&A Data and Analytics

SQL Structured Query Language

API Application Programming Inteface

AWS Amazon Web Service

S3 Amazon Simple Storage Services

B2B Business-to-Business

B2C Business-to-Consumer

IU Interficies d'Usuari

CSV Comma-Separated Values

OLAP Online Analytical Processing

KPI Key Performance Indicator

BBDD Bases de Dades

FK Foreign Key

HTML Hyper Text Markup Language

CSS Cascading Style Sheets

JS JavaScript

DOM Document Object Model

AJAX Asynchronous JavaScript and XML

JSON JavaScript Object Notation

XML Extensible Markup Language

MVC Model Vista Controlador

SGBD Sistema Gestor de Bases de Dades

MVCC MultiVersion Concurrency Control

RDS Relational Database Service

BI Business Intelligence

PK Primary Key

EC2 Elastic Compute Cloud

EMR Elastic MapReduce

ReST Representational State Transfer

SOAP Simple Object Access Protocol

URL Uniform Resource Locator

SDK Software Development Kit

RDD Resilient Distributed Dataset

HDFS Hadoop Distributed File System

AMI Amazon Machine Image

HDD Hard Disk Drive

SSD Solid-State Drive

SVG Scalable Vector Graphics

VML Vector Markup Language

D3 Data-Driven Documents

SPICE Super-fast Parallel In-memory Calculation Engine

HTTP HyperText Transfer Protocol

URI Uniform Resource Identifier

ORM Object Relational Mapper

KAM Key Account Manager

Resum

Durant la meva estada a la Universitat de les Illes Balears (UIB) he tengut l'oportunitat de poder provar moltes àrees del gran món que és el de la Informàtica. Però després d'haver completat els tres primers cursos, amb el petit bagatge que havia aconseguit durant aquests anys, em trobava en situació de poder dir que el què més m'atreia de tots els camps de la Informàtica era, en general, el desenvolupament de software.

Això coincideix amb la recepció d'una oferta de realitzar pràctiques en empresa a *Hotelbeds*, una de les majors —si no la major— empreses amb una gran base informàtica on podia ampliar el meu coneixement en disseny i desenvolupament de software. Després de rebre diverses recomanacions, vaig acceptar i realitzar més de 1000 hores de feina. Una de les raons per les que vaig decidir realitzar les pràctiques en aquesta empresa és perquè el departament on faria feina era el departament de D&A, el que em va atreure degut a la seva proximitat al *Data Science* i *Data Mining*. Va ser durant aquest període que vaig decidir que seria allà on realitzaria el TFG.

Un cop vaig estar situat dins l'ambient laboral va ser quan vaig començar a treballar en el que últimament acabaria sent una plataforma per l'administració, control i explotació de les dades de l'empresa. Per a realitzar aquesta tasca vaig haver d'aprendre diverses tecnologies que m'han fet millorar com a professional del sector, com ara programació web —*Front End*—, programació d'API —*Back End*— i la interacció amb múltiples components de plataformes com ara Amazon Web Service (AWS), que impliquen bases de dades no-Structured Query Language (SQL), PostgreSQL i Amazon Simple Storage Services (S3), entre d'altres.

En aquest document es detallaran quines són les tasques que vaig dur a terme, la problemàtica que vaig trobar durant el desenvolupament i, quan calgui, una justificació de les decisions preses.

CAPÍTOL 1

INTRODUCCIÓ

1.1 Hotelbeds

Per a imaginar la magnitud real d'*Hotelbeds* basta en dir que és el *bedbank* més gran del món. En general, *Hotelbeds* és un negoci Business-to-Business (B2B), és a dir, enlloc d'interactuar amb el que serà l'usuari final del producte, actua com a intermediari entre el negoci que ofereix el servei i un altre negoci que el presenta al públic. Amb altres paraules, *Hotelbeds* ofereix un *booking engine* que empren agències de viatges minoristes arreu del món. Tot això implica la creació de processos i bases de dades que permetin el funcionament del negoci.

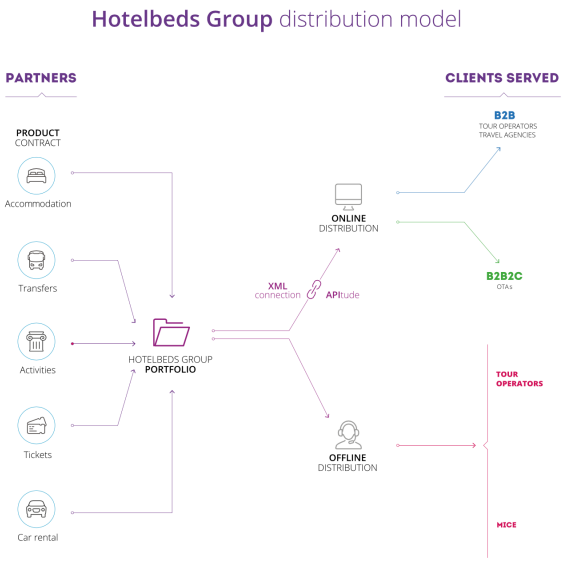


Figura 1.1: Model de distribució d'*Hotelbeds*

A grans trets l'empresa està dividida en 4 grans departaments o marques:

- **Bedbank**: és la part del negoci encarregada de contractar, connectar i distribuir de manera eficient tota una cartera de més de 130.000 hotels entre agències de viatges minoristes, agències de viatges online, Tour operadors i aerolínies.
- **Transfer and Activity Bank**: a diferència del Bedbank, T&AB actua tant en l'àmbit del B2B com del Business-to-Consumer (B2C) proporcionant rutes de transport entre localitzacions, activitats com ara excursions, teatre, o lloguer de cotxes.
- **Destination Management**: formada per 3 marques a nivell global, aquesta unitat de negoci s'encarrega de proporcionar serveis com ara recollida i transport en ports per a turistes de creuers —*Intercruises*— o per a fer quedades en altres països —*PacificWorld*—.
- **New Ventures**: la unitat de New Ventures té com a funció la compra de petites empreses creixents amb molt de potencial i les incorpora al negoci. Per exemple, empreses com *CARNECT*, que proporciona una plataforma per a reserves de cotxes de lloguer B2B, *TTServices*, un proveïdor de solucions de processament de visats per a governs arreu del món o *ROIBACK* són les que conformen aquesta unitat de negoci.

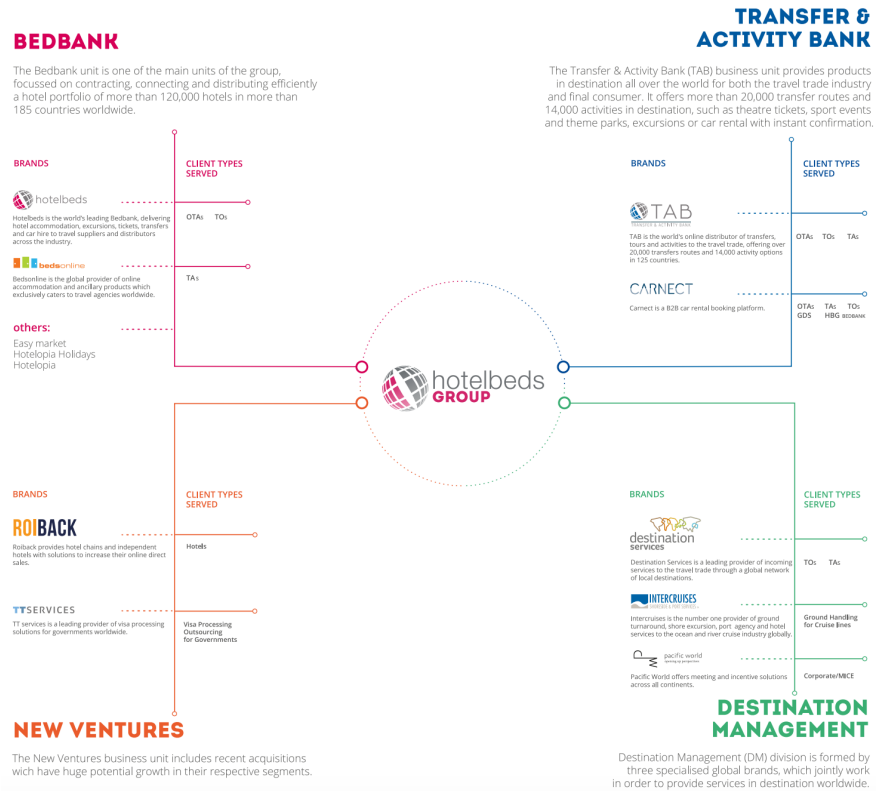


Figura 1.2: Unitats de Negoci dins *Hotlebeds*

1.2 D&A dins Hotelbeds

El departament de D&A va aparèixer dins *Hotelbeds* com a una unitat dedicada exclusivament al desenvolupament d'un gran projecte. Aquest consisteix en efectuar una migració de totes les dades de la plataforma actual d'emmagatzematge i ingestió de dades a una nova basada en el *cloud* emprant els serveis d'AWS. Evidentment, el flux de dades que arriben a les bases de dades de l'empresa és enorme i per tant tots els processos existents tant d'extracció, anàlisi ... de dades s'han de tornar a fer per a poder-se aplicar a la nova plataforma. A més de tota aquesta feina, també s'han de crear nous processos per a efectuar la pròpia migració de dades.

Però a part de tota aquesta branca de refer i migrar, el projecte tenia una sèrie de serveis que es volien crear com a complement a la nova plataforma. Aquests serveis s'ajunten en una suite d'aplicacions que, en general permeten fer una gestió de les dades, després de ser processades o no, de les taules de les bases de dades i eines per a poder fer *reporting*.

1.3 *Data Governance* com a TFG

El nom que se li va donar a tot el conjunt d'aplicacions va ser *Data Governance*. És en aquesta part del projecte on vaig tenir la major part de la meva feina. Com s'ha comentat abans, aquesta plataforma està formada per vàries aplicacions que tenen funcions diferents i que estan orientades a diversos perfils de professionals dins l'empresa. Per exemple, hi ha eines per a elaborar *dashboards* per a treballadors centrats en el negoci i no en la part tècnica. A partir d'aquests *dashboards* es poden prendre decisions importants per a millorar les ventes, per exemple.

D'altra banda també existeixen eines per a documentar i catalogar totes les taules de les bases de dades, de manera que queden registrades totes les modificacions que es fan a l'estructura d'una taula juntament amb les descripcions de les funcions de cada una d'elles.

Un cop se'm va explicar en que consistia el *Data Governance* vaig decidir que seria un bon desenvolupament en el qual basar-se per a elaborar el TFG ja que inclou moltes de les tecnologies i paradigmes de programació que sempre m'han atret des de que vaig començar a cursar els estudis d'Informàtica. A més sabia que era una oportunitat única ja que en cap altra banda podria tenir a la meva disposició tantes eines com les que ofereix AWS, ja que el seu preu resulta prohibitiu per a la majoria d'empreses. Aquest projecte inclou programació web i d'Interficies d'Usuari (IU) emprant noves eines emergents, programació d'APIs, fer servir llenguatges d'*scripting* com ara *Python* per a interactuar amb BBDD, tant SQL com no-SQL i fins i tot fer analisi multidimensional de les dades de negoci, o OLAP. Finalment també s'han emprat eines de *cloud computing*, concretament *SPARK Streaming* també amb *Python* per a fer càlculs distribuïts entre diverses màquines amb quantitats massives de dades.

1.4 En què consisteix

Dins les bases de dades d'*Hotelbeds* existeixen moltes taules de bases de dades, tantes que pot resultar complicat seguir el ritme en el que es creen noves taules. És per això que resulta interessant tenir una aplicació que permeti als usuaris d'arquitectura tenir informació de les taules de la base de dades. Això és el diccionari de dades, anomenat després *Technical Dictionary*, ja que és una eina que només l'empraran usuaris tècnics, familiaritzats amb els aspectes més concrets de les bases de dades.

D'altra banda, dins *Hotelbeds* existeixen molts departaments que fan feina en diferents projectes. En cada projecte és possible que s'obtenguin dades de manera regular, per exemple, un resum de les ventes d'una certa cadena hotelera en una regió cada més. Moltes vegades els treballadors que obtenen aquests resultats no són tècnics i les dades es tracten amb programes com *excel* o amb formats com Comma-Separated Values (CSV). Per això a vegades resulta complicat consolidar aquestes dades en les bases de dades. El *Data Entry* permet a usuaris d'aquests tipus és pujar fitxers en format CSV de manera regular —ja sigui diari, mensual o aperiòdicament— seguint unes directrius que defineixen el format que han de tenir els fitxers que es volen pujar.

Finalment, el motor OLAP, o *OLAP Engine* és una aplicació que permet definir KPI a usuaris de negoci que es calculen a partir de camps de les taules de la base de dades. Emprant tecnologies de computació distribuïda es fan agregacions que finalment permetran que altres eines per representar dades creïn gràfiques molt útils pels usuaris de negoci.

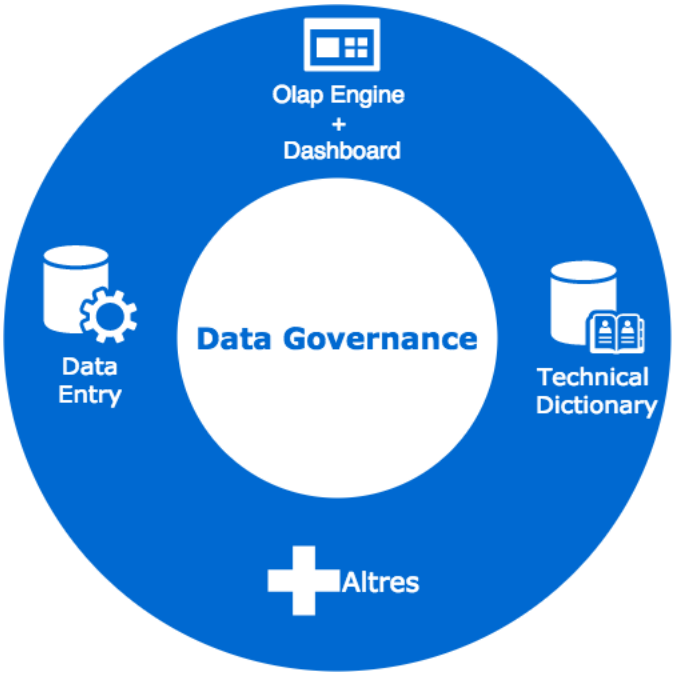


Figura 1.3: La plataforma del *DataGovernance*

1.5 Resum de la meva feina

De la plataforma final del *Data Governance* vaig acabar desenvolupant 3 aplicacions. En una primera iteració les vaig realitzar completament pel meu compte. A continuació s'explicarà breument en que consisteix cada una d'elles:

1.5.1 *Technical Dictionary*

D'aquesta plataforma vaig començar desenvolupant un prototip del que després es denominaria *Diccionari Tècnic*, o *Technical Dictionary*. Aquest servei serveix per a documentar totes les taules de les bases de dades existents, detallant, a grans trets, informació separada en 3 apartats:

- *Taula*: ofereix informació a nivell de la taula, com ara a quina base de dades correspon, qui la va crear, quan es va crear i una descripció sobre la seva funció entre d'altres.
- *Columnes*: de la mateixa manera que amb la informació sobre una taula, es detalla informació sobre cada columna de la taula seleccionada, per exemple: nom de la columna, el tipus de la columna, la longitud de la columna si aplica, etc.
- *FK*: informació sobre totes aquelles columnes que són FK en aquesta taula.

1.5.2 *Data Entry*

Aquest mòdul va ser ideat per a poder fer ingestes de dades sense estar planificades o automatitzades, és a dir, permetre a un usuari guardar dades de negoci que no provenen d'un procés automàtic en una taula d'alguna de les BBDD de l'empresa. D'aquesta manera s'elimina la necessitat de crear processos —*Pipelines*— per a moure dades d'un repositori a un altre.

Evidentment aquesta eina s'ha d'emprar amb precaució ja que si s'en fa un ús abusiu o incorrecte es poden crear moltes taules innecessàries. És per això que s'implementa un sistema de *plantilles* que permeten definir diversos paràmetres que el document que conté les dades ha de complir. Per exemple, es poden definir restriccions de nom de fitxer, de columnes que ha de contenir el fitxer, tipus de dades de les columnes, longituds ...

1.5.3 *OLAP Engine*

Finalment, el darrer component que vaig desenvolupar és el Motor OLAP, una eina ideada per a agrupar dades en *Cubs* sobre els quals es realitzen posteriors operacions per a poder extreure mètriques que poden servir com a indicadors de negoci, comunament coneguts com a KPI. A partir d'aquests KPIs es poden elaborar gràfiques de múltiples tipus, que ajuden a visualitzar les evolucions de les mètriques per a poder prendre decisions de negoci.

1.5.4 Estructura del document

Aquesta memòria s'estructura en una sèrie de capítols. A continuació s'explica breument en que consisteix cada capítol:

- *Introducció*: es posa en situació al lector amb una explicació del context de l'empresa i s'enumeren les tasques que es varen completar juntament amb els objectius.
- *Tecnologies*: s'enumeren un conjunt de solucions existents similars al que s'ha desenvolupat i s'explica la tecnologia que s'ha fet servir per a poder completar el desenvolupament.
- *Technical Dictionary*: s'explica com s'ha desenvolupat el diccionari tècnic, juntament amb els seus objectius i una explicació del seu disseny i arquitectura.
- *Data Entry*: s'explica com s'ha implementat el *Data Entry*, juntament amb els seus objectius i una explicació del seu disseny i arquitectura.
- *OLAP Engine*: s'explica com s'ha desenvolupat l'*OLAP Engine*, juntament amb els seus objectius i una explicació del seu disseny i arquitectura.
- *Conclusions*: s'exposen els resultats i les opinions personals després del desenvolupament.

A patir d'ara, quan es faci una referència a "*la plataforma*", ens referirem al *Data Governance*. En els capítols corresponents es farà una explicació detallada del desenvolupament de cada component.

CAPÍTOL 2

TECNOLOGIES

A continuació es recullen les diferents tecnologies que s'han emprat durant tot el desenvolupament dels mòduls de la plataforma. Per a cada una es farà una explicació explicant en que consisteix i per què ha resultat important. Quan sigui necessari, es justificarà l'elecció d'una eina respecte d'una altra amb funcionalitats diferents.

Degut a que tota la IU del *Data Governance* està en forma d'aplicació web predominen les tecnologies de desenvolupament d'aplicacions Web:

2.1 Tecnologies Web

- **Hyper Text Markup Language (HTML) 5**: es tracta de la cinquena iteració del llenguatge d'etiquetat per a la creació de pàgines web. HTML5 es va convertir en l'estàndard de la programació web l'any 2008 i va incorporar diverses millores a les anteriors versions:
  - Nous elements semàntics: noves etiquetes que simplifiquen l'estructura del document HTML com ara *header* o *footer*, millorant la llegibilitat i desincentivant l'abús de l'element *div*. També incorpora elements multimèdia com ara *video* i *audio* i elements gràfics —*svg* o *canvas*—.
  - Noves APIs HTML: s'han afegit noves eines que permeten obtenir informació del client, com per exemple l'API de geolocalització. També eines importantíssimes com ara el *Local Storage* i el *Session Storage*, que permeten al programador guardar informació en format *String*.
  - *Drag & Drop*: en HTML5 la funcionalitat *Drag & Drop* és estàndard, és a dir, qualsevol element pot esser arrossegable [4].

- **Cascading Style Sheets (CSS)**: llenguatge emprat per escriure fulles d'estil que s'empren per a definir i descriure l'aspecte estètic de, en aquest cas, les pàgines web. Juntament amb *JavaScript* (2.1) i HTML (2.1), CSS és una pedra angular que s'empra per la majoria de les webs per crear pàgines visualment atractives i IU per a aplicacions Web com per aplicacions mòbil [5].

  Actualment CSS es troba en la seva tercera versió des de la seva creació el 1996. En el pas de CSS2 a CSS3 es va separar l'especificació en petits documents anomenats *mòduls*, és a dir es va passar de tenir un gran document en el que es defineixen totes les funcionalitats de CSS a tenir *mòduls* que estenen o afegeixen funcionalitats ja existents en CSS2, aconseguint així una compatibilitat entre la versió 2 i 3.

- **JS, AJAX i JQuery**: JS és un llenguatge d'alt nivell, orientat a objectes, *no-tipat* –o tipat dinàmic– i interpretat. Com ja s'ha comentat abans, és un dels tres nuclis tecnològics essencials per a la majoria de pàgines web. Gràcies a JS les pàgines web poden ser interactives, ja que no només ofereix eines que qualsevol esperaria trobar en un llenguatge de programació, sinó que també té la possibilitat d'editar elements del Document Object Model (DOM) [6].

  Avui en dia tots els navegadors ofereixen suport a JS, que consta de moltes llibreries de codi lliure que es basen en JS i el complementen, abastant així una gran quantitat d'àmbits de la programació. Entre aquestes llibreries en destacarem 2:

  – *AJAX*: tecnologia que va aparèixer el 2005, utilitzada en el client per a crear aplicacions asíncrones. És a dir, emprant aquesta tècnica, les aplicacions poden enviar i rebre dades a un servidor sense la necessitat de recarregar la pàgina per a cada petició. Tot i que normalment s'entén AJAX com a una sola tecnologia, és en realitat un conjunt de tecnologies:

    * HTML i CSS: emprats per a la presentació de la pàgina.
    * DOM: per a mostrar dinàmicament la interacció amb les dades.
    * JavaScript Object Notation (JSON) o Extensible Markup Language (XML): per a l'intercanvi de dades.
    * L'objecte *XMLHttpRequest*: API en forma d'objecte que conté els mètodes per a realitzar la transferència de dades entre el client i el servidor.
    * JS: tecnologia que posa totes les altres en comú.

    Tot i que inicialment era un requisit, XML ha passat en un segon pla mentre que JSON s'ha convertit en el format més usat, tot i que altres formats també és poden emprar. Per a facilitar la creació d'aplicacions AJAX existeixen llibreries que ofereixen una abstracció per a realitzar peticions, com ara *JQuery*[7].

  – *JQuery*: com s'ha mencionat anteriorment, JQuery és una llibreria de JS dissenyada per facilitar la navegació del document, seleccionar elements del DOM, crear animacions, gestionar esdeveniments i desenvolupar aplicacions AJAX. En general, el desenvolupament amb *JQuery* té 4 principis:

    * Separació JS i HTML: la sintaxi per afegir gestors d'esdeveniments al DOMque proveeix *JQuery*, cosa que incentiva al programador a fer-

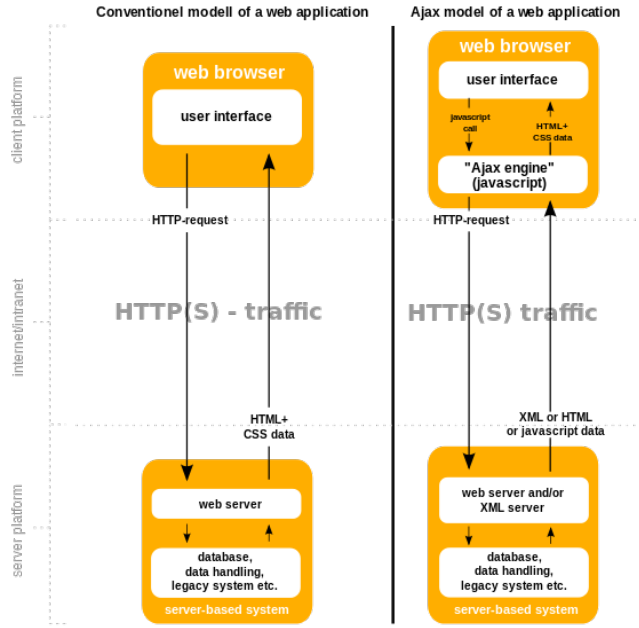


Figura 2.1: Funcionament d'una aplicació AJAX

ho així enlloc d'afegir atributs d'esdeveniments HTML per a cridar funcions de JS. Per exemple:

Listing 2.1: Esdeveniments amb HTML

```
<!-- enlloc de fer servir -->
<p id="element" onclick="doSomething();">Click me</p>
```

Listing 2.2: Esdeveniments amb JS o JQuery

```
//fer servir
document.getElementById('element').onclick = doSomething;
//o bé, amb JQuery
$( "#element" ).click(doSomething);
```

D'aquesta manera tenim el codi JS completament separat del codi HTML.

* Brevetat i claredat: *JQuery* promou la claredat amb trets distintius com ara l'encadenament de funcions.
* Eliminació d'incompatibilitats entre navegadors: els intèrprets de JavaScript dels diferents navegadors poden tenir algunes diferencies, que poden provocar que codi que funciona a un navegador deixi de funcionar en un altre. *JQuery* gestiona aquestes inconsistències.
* Extensibilitat: *JQuery* permet una manera fàcil de programar nous esdeveniments, elements i mètodes que després podran ser reutilitzats com a *plugins* [8].

- **Angular JS, Angular 2 i *Typescript***: Com s'ha comentat anteriorment, una versió preliminar va ser desenvolupada per mi emprant AngularJS. Després, quan es va fer el pas a la plataforma final es va fer el salt a Angular2.

  AngularJS és un *framework* per al desenvolupament d'aplicacions web basat en JS. Actualment mantingut per Google, va ser dissenyat específicament pel desenvolupament d'aplicacions d'una sola pàgina o *Single-Page Applications* — pàgines web que intenten emular una aplicació d'escriptori, on els recursos de la pàgina es carreguen dinàmicament en funció de les accions de l'usuari — i per a simplificar el desenvolupament d'aplicacions que segueixen el patró MVC.

  Aquest *framework* adapta i estén el codi tradicional HTML per permetre l'anomenat *two-way binding*, que permet la sincronització automàtica de models i vistes. Així es simplifica encara més la gestió dels elements del DOM i es millora el rendiment [9].

  Principalment, els objectius que es volien assolir en el moment en el que es va dissenyar AngularJS eren:

  - Separar la manupulació del DOM de la lògica del programa. La problemàtica resideix principalment en l'estructuració del codi.
  - Separar totalment el desenvolupament del codi de la banda del client i el codi de la banda del servidor, aconseguint així la possibilitat d'un desenvolupament en paral·lel.
  - Proporcionar una estructura pel procés de desenvolupar l'aplicació, començant pel disseny de l'IU passant per escriure la lògica del programa fins al *testing*.

  AngualrJS implementa el patró MVC per separar les dades de la presentació i els components lògics. Els conceptes de *Vista* i *Controlador* es veuen fortament reflectits en el concepte de `$scope`. En AngularJS, `$scope` és un tipus d'objecte i pot estar dins o fora de l'entorn en qualsevol part d'un programa, ja que segueix les normes d'entorn establertes per a qualsevol objecte de JS. Com a part de l'arquitectura MVC, `$scope` conforma el *Model* i totes les variables definides dins l'`$scope` poden ser accedides per tant la *Vista* com el *Controlador*. L'`$scope` actua com a la cola que junta la *Vista* i el *Controlador*.

  Un exemple d'aquest fet pot demostrar amb dues directives d'AngularJS, que s'empren com a atributs d'elements HTML:

  Suposant que tenim una variable declarada a l'`$scope` *name*:

Listing 2.3: Exemple de Controlador amb AngularJS

```
1     var app = angular.module('myApp', []);
2     app.controller('myCtrl', function($scope) {
3     $scope.name = "John Doe";
4     });
```

- `ng-bind`:

Listing 2.4: Exemple de funcionament de `ng-bind`

```
<!-- exemple ng-bind -->
<span ng-bind="name"></span>
```

En aquest cas es mostrarà el valor de la variable declarada en l'`$scope`, però de cap manera podrem canviar el seu valor.

- `ng-model`:

Listing 2.5: Exemple de funcionament de `ng-model`

```
<!-- exemple ng-model -->
<input ng-model="name">
```

Ara apareixerà un camp de text que tendrà com a valor *John Doe*. Com que es tracta d'`ng-model`, tenim *two-way binding* i qualsevol canvi que facem dins aquest camp de text implicarà que la variable també canviï de valor.

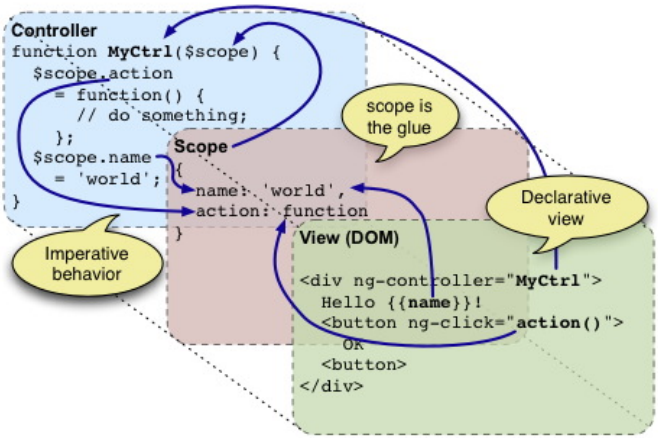


Figura 2.2: Exemple del MVC dins AngularJS [1]

D'altra banda tenim Angular 2. Aquest *framework*, tot i tenir un nom que sona a continuació de la primera versió —AngularJS—, va ser reescrit des de zero. A diferència de la versió original, per a programar una aplicació Angular 2 s'ha de programar amb *Typescript*. A continuació es farà una breu explicació de en que consisteix aquest llenguatge:

Typescript és un llenguatge desenvolupat i mantingut per *Microsoft* de codi lliure. És un *superset* de JS, de manera que és completament retrocompatible amb programes escrits purament en JS.

Enlloc de ser un llenguatge interpretat, *Typescript* fa una *transcompilació* del codi font i el transforma en codi JS, és a dir fa una compilació *source-to-source*. Així, qualsevol navegador que pugui executar JS podrà executar una aplicació hagi estat originalment desenvolupada en *Typescript* [10].

Typescript també ofereix la possibilitat d'incloure fitxers de capçalera, com ara llenguatges com C o C++, que descriuen l'estructura de fitxers objecte. Hi ha fitxers de capçalera de llibreries com *JQuery*, *MongoDB* o *D3.js*.

A més, *Typescript* afegeix la possibilitat de dotar de tipus a les variables i a les funcions —valor de retorn—. Això permet que es faci una comprovació de tipus en temps de compilació, tot i que pot ser perfectament ignorat i es pot usar el tipat dinàmic de JS [11].

Angular 2 presenta moltes diferències i millores respecte el seu antecessor AngularJS [2]:

- Elimina el concepte de `$scope` i els controladors, emprant una jerarquia de components com a element estructural.
- Té una sintaxi més simple, centrant-se amb l'ús de `" [ ] "` per a *bindings* de propietats i `" ( ) "` per a *bindings* d'esdeveniments.
- S'ha millorat la modularitat separant moltes de les funcions que abans estaven en el *nucli* de AngularJS en altres mòduls, millorant el rendiment.
- S'ha donat el salt a *Typescript*, que ha incorporat funcionalitats com programació orientada a objectes amb classes, tipat estàtic i genèrics.
- S'han substituït els controladors i l'`$scope` per components i directives. Un component és una directiva amb un *template* HTML.

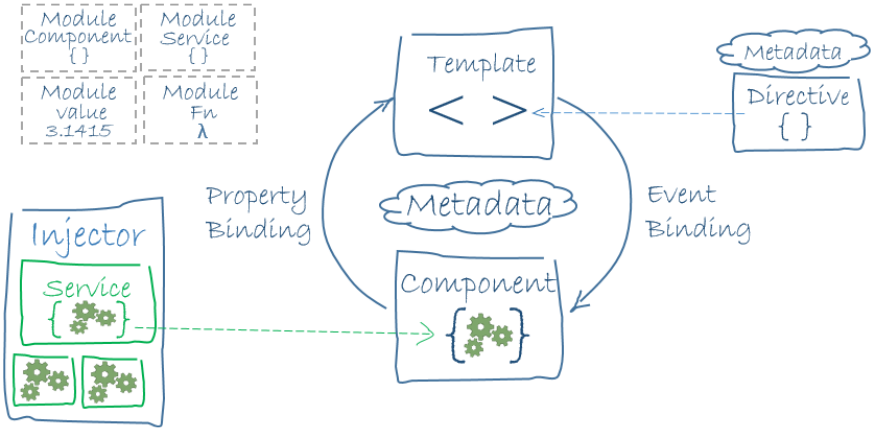


Figura 2.3: Exemple de l'arquitectura d'Angular 2 [2]

2.2 BBDD i emmagatzematge

A continuació s'exposaran les tecnologies que s'han emprat per a emmagatzemar les dades que després s'han emprat tant per mostrar per pantalla com les que s'han utilitzat com a eina auxiliar. Cal comentar que degut a que la majoria de càlculs es fan al *cloud*, moltes d'aquestes tecnologies formen part dels AWS.

- **PostgreSQL i *Redshift***: PostgreSQL, o Postgres, és un Sistema Gestor de Bases de Dades (SGBD) objecte-relacional. Com la majoria de BBDD, les seves funcions són guardar i retornar dades segons les peticions provinents d'altres aplicacions software. Tot i que comparteix característiques amb altres SGBD, consta d'algunes funcionalitats exclusives. Les característiques principals de Postgres són les següents [12]:

  - *Alta Concurrència*: Mitjançant un sistema anomenat MultiVersion Concurrency Control (MVCC), assigna a cada transacció una *foto* de la base de dades, de manera que altres canvis que s'hagin pogut fer durant aquesta transacció no siguin visibles fins s'ha completat. Això elimina la necessitat de tenir *read locks* —bloquejar les dades mentre s'estan llegint—.
  - *Varietat de tipus natius*: a més dels tipus suportats en qualsevol SGBD, Postgres suporta nombres de precisió arbitrària, texts de longitud il·limitada, Vectors o fins i tot figures geomètriques. A més, un usuari pot crear-se el seu propi tipus de dades.
  - *Funcions*: Postgres permet definir funcions amb diversos llenguatges, permetent així les avantatges de la programació funcional o fins i tot orientada a objectes.
  - *Altres característiques*: com ara l'existència de FK, *triggers*, vistes, herència de taules...

  Redshift, d'altra banda, és el servei que ofereix Amazon amb els seus AWS com a producte de *data warehousing*. Aquest producte està directament basat en PostgreSQL 8.0.2, per tant totes les noves funcionalitats afegides en les version 9.x no es troben implementades dins *Redshift*[13]. *Redshift* es diferencia de l'altre producte de base de dades d'AWS Relational Database Service (RDS) en que es troba especialment optimitzat per a operacions de grans conjunts de dades. De fet, fa ser desenvolupat per a processos OLAP i de Business Intelligence (BI). Degut a que abasten problemes tant diferents l'esquema d'emmagatzematge de dades i el motor d'execució de *queries*. Per exemple, mentre que un SGBD normal emmagatzemaria les dades en files, *Redshift* ho fa per columnes , emprant sistemes de compressió per aconseguir una utilització de memòria i E/S de disc[14].

  Tot i tenir aquestes avantatges respecte Postgres, *Redshift* també manca vàries funcionalitats que per defecte existeixen en Postgres. Algunes d'aquestes són importants sacrificis que s'han hagut de fer per aconseguir una eina eficient ideada per al *Data Warehousing*:

  - Tot i que és cert que està basat en PostgreSQL, existeixen petites diferències en segons quines comandes SQL.

  - Les restriccions `UNIQUE`, `FOREIGN KEY` i `PRIMARY KEY` es permeten, però són purament informatives, no són obligatories ni impliquen la creació d'indexs.
  - No existeix el concepte d'Herència.
  - No existeixen cap mena d'índexs.
  - No existeixen *triggers*.
  - No existeixen funcions o *Stored Procedures*[15].

- **DynamoDB**: Així com *Redshift*, tot i ser una Base de Dades relativament diferent, és SQL, DynamoDB és el servei de Base de Dades no-SQL que ofereix AWS. DynamoDB funciona amb un sistema de taules, en la qual s'ha de definir forçosament una *Partition Key*. Aquesta es farà servir per distribuir les dades per totes les particions per a millorar l'escalabilitat. Per tant ha de ser única. En general, la Primary Key (PK) d'una taula està formada per la *Partition Key* i una *Sort Key* sent aquesta darrera opcional. La *Sort Key* s'empra quan hi pot haver repetició de *Partition Key*.

  Per exemple, si guardam informació sobre música podríem posar el nom de l'artista. Com que un artista pot tenir més d'una cançó, haurem d'emprar la *Sort Key*, que serà el nom de cada cançó. Cada element d'una taula s'anomena *Item*, i tendrà, forçosament al manco un camp. La informació en els *Items* s'emmagatzema en forma de parelles Clau : Valor, com un objecte JSON. Si tenguessim un *Item* pel·lícula podria, per exemple, tenir la següent estructura:

Listing 2.6: Exemple d'un Item de DynamoDB

```
{
"year": "2017",
"title": "John Doe Returns",
"info": {
   "plot": "Una sinòpsis de longitut arbitrària",
   "rating": 10
   }
}
```

  En aquest cas podriem suposar que la PK esta formada pels atributs *year* i *title*, però no hem dit res de l'atribut info. Aquesta manca d'esquema fa que sigui no-SQL i ens sigui molt còmode afegir dades quan podem tenir un nombre indeterminat d'elements amb atributs que a vegades existiran i a vegades no. També podem disposar de les facilitats que ens dóna el poder emprar JSON com a format de dades, ja que és molt bo d'utilitzar amb llenguatges com *Python*.

- **S3**: Com cal esperar d'una tecnologia dins aquesta categoria, S3 és un servei d'emmagatzematge a través d'interfícies de serveis web, com ara ReST o Simple Object Access Protocol (SOAP). Els clients utilitzen S3 com a contenidor de les seves aplicacions basades en el cloud, com a repositori massiu o com a *data lake*[16].

  La idea darrera S3 no és la de tenir la funcionalitat d'una base de dades, ja que no existeixen ni esquemes ni taules. És un sistema que permet a l'usuari guardar qualsevol tipus de fitxer. Utilitza una arquitectura de *object storage*, una arquitectura que tracta les dades com a objectes, al contrari que altres arquitectures com ara sistemes de fitxers, que organitzen les dades segons jerarquies de fitxers o emmagatzematge per blocs, que maneja pistes i sectors. Cada objecte conté les seves pròpies dades, un petita part de *metadata* i un identificador únic[17].

  En el cas concret de S3, els objectes —qualsevol tipus de fitxer— poden tenir una mida màxima de 5 *terabytes*, cada un acompanyat per un màxim de 2 *kilobytes* de *metadata*. Amazon va decidir organitzar els objectes en *Buckets*, cada un dels quals posseït per un compte d'AWS i identificat per una clau assignada per l'usuari.

  Un *Bucket* representa el nivell més alt del que es podria entendre com un sistema de directoris en el que, dins cada nivell podem guardar qualsevol objecte. Els *Buckets* poden ser creats, llistats i recuperats emprant una interfície ReST. Els *Buckets* i els directoris poden ser accedits a partir d'una Uniform Resource Locator (URL), per exemple

  ```
  s3n://hbg-cdr-data-architecture/temp_olap_engine
  ```

  accediria al bucket `hbg-cdr-data-architecture` i al directori contingut dins aquest anomenat `temp_olap_engine`.

2.3 Altres

Finalment es descriuran altres eines que han estat imprescindibles per a completar el desenvolupament de la feina:

- **Python com a llenguatge d'*scripting***: Dins tot el departament de D&A el llenguatge que s'empra per a programar tots els processos, sobretot aquells que facin servir AWS —que són quasi tots—. Python és un llenguatge interpretat de propòsit general. Python premia molt la llegibilitat emprant indentació per marcar blocs de codi i una sintaxi molt simple que permet expressar conceptes amb menys línies que altres llenguatges de programació.

  És un llenguatge amb tipat dinàmic i gestor de memòria automàtic que admet molts paradigmes de programació com la programació orientada a objectes, imperativa, funcional i procedural entre moltes d'altres. Python no és un llenguatge ràpid per si mateix, però la majoria dels processos que s'escriuen no requereixen un temps d'execució mínim.

  En general, degut a la rica comunitat que té aquest llenguatge, disposa de moltíssims paquets que permeten resoldre tot tipus de problemàtiques, independentment de l'àmbit en el que s'està programant. Degut a la seva llegibilitat, juntament amb els comentaris del codi, normalment no és necessari documentar el codi. Una altra raó per la qual s'empra Python és degut a la seva curva d'aprenentatge. Resulta molt fàcil iniciar-se en la programació amb Python, encara més si ja es té una experiència amb altres llenguatges. El codi produït sol esser bastant fàcil de mantenir i el desenvolupament de programes amb Python és significativament més ràpid que amb altres llenguatges, el que implica menys hores de programació i, per tant, menys despeses en salaris de programadors.

  En el cas d'*Hotelbeds*, com que fa feina amb AWS i el llenguatge establert és Python, s'utilitza un mòdul anomenat `boto3`. Es tracta de l'Software Development Kit (SDK) per Python que proporciona AWS, que permet als desenvolupadors de Python escriure software que faci ús dels serveis d'Amazon, com *Redshift*, DynamoDB, S3 i altres que es comentaran a continuació. Aquest SDK proporciona una API orientada a objectes[18] fàcil d'utilitzar.

  Per a fer servir qualsevol servei d'AWS sense emprar la seva consola web cal tenir configurada la màquina amb una clau s'accés que serveix com a identificador i una clau secreta que serveix com a contrasenya. Per fer això cal tenir `AWS cli` instal·lat i configurar els credencials amb:

  ```
  aws configure
  ```

  i introduir els credencials:

  ```
  [default]
  aws_access_key_id = YOUR_ACCESS_KEY
  aws_secret_access_key = YOUR_SECRET_KEY
  ```

- **Spark**: Spark és un *framework* de codi lliure per a computació distribuïda. Tot i que va ser ideada a l'universitat de Berkeley, actualment és una propietat d'Apache. Spark proveeix una API que es centra principalment en una estructura de dades anomenada *Resilient Distributed Dataset (RDD)*, un conjunt de dades *read-only* que es distribueix sobre un *cluster* de màquines, mantingut de tal manera que és tolerant a fallades. Spark va ser desenvolupat per a combatre les limitacions que tenia *MapReduce*.

  Spark necessita dos components essencials: un *cluster manager* —o gestor de *clusters*— i un sistema d'emmagatzematge distribuït. Com a gestor Spark pot funcionar amb diverses opcions, com ara *Hadoop YARN*, *Apache Mesos* o fins i tot la versió pròpia independent de Spark. De banda del sistema d'emmagatzematge també accepta diverses opcions com per exemple Hadoop Distributed File System (HDFS), S3 o *Cassandra*, tot i que en aquest cas es va fer servir un sistema en memòria en un cluster de màquines virtuals proveïdes per Amazon. A més, Spark dóna la possibilitat d'executar en un mode *pseudodistribuït*, emprat per desenvolupar i provar. Aquest mode utilitza el sistema de fitxers de la màquina local i fa servir un executor per a cada nucli de la CPU.

  Spark es separa en una sèrie de components, dels quals ens fixarem en 2:

  - *Spark Core*: és el nucli de tot el projecte Spark i proveeix al programador amb funcionalitats d'enviament de tasques, programació de tasques, i operacions E/S, tot això gràcies a les APIs que té per als llenguatges Java, Scala, R i Python, que es centren en l'ús de RDDs. La manera de funcionar sol esser del tipus: el programa principal invoca operacions paral·leles com ara *map*, *filter* o *reduce* sobre una RDD passant una funció a Spark, que després l'orquestra per que s'executi en paral·lel en el *cluster*. Aquestes operacions són *mandroses* i agafen una RDD com a *input* i treuen una RDD com a *output*. Les RDD són immutables i la tolerància a fallades s'aconsegueix gràcies a que cada RDD té un historial d'operacions que se li han realitzat de manera que en qualsevol moment es pugui tornar enrere per ser reconstruïda en cas de que es perdin dades. Poden contenir qualsevol tipus d'objecte Python, Java o Scala.
  - *Spark SQL*: és un component sobre el Core de Spark que introdueix l'abstracció de dades *DataFrame*, que permet manejar dades estructurades i semi-estructurades. A més, incorpora un mètodes per a poder manipular els *DataFrames* així com suport per a emprar sentències SQL, que es poden aplicar directament sobre els *DataFrames* com si es tractessin de taules SQL.

    En el nostre cas ens podem trobar exemples com el següent, que llegeix les columnes desitjades d'una taula sencera i la carrega en un *DataFrame*, per a posterior manipulació:

```
query = "SELECT {} FROM {}".format(columns, table_name)

df = Spark_SqlContext.read                                    \
.format("com.databricks.spark.redshift")                      \
.option("url", url)                                           \
.option("user", REDSHIFT_USER)                                \
.option("password", REDSHIFT_PASS)                            \
.option("query", query)                                       \
.option("temporary_aws_access_key_id", access)                \
.option("temporary_aws_secret_access_key", secret)            \
.option("temporary_aws_session_token", token)                 \
.option("tempdir", s3url)                                     \
.load()


```

Listing 2.7: Exemple Spark SQL

- **Elastic Compute Cloud (EC2)**: Aquest servei d'AWS permet als clients llogar màquines virtuals en les quals es poden executar les seves pròpies aplicacions. EC2 permet a l'usuari un desplegament escalable les seves aplicacions proporcionant un servei web a partir del qual els clients poden configurar i arrencar Amazon Machine Image (AMI) per a crear les seves màquines virtuals. Aquestes màquines s'anomenen *instàncies*, i poden venir equipades amb qualsevol software que es desitgi. La gràcia d'aquest servei es que els usuaris poden crear, llançar i matar les instàncies com vulguin pagant només per el temps que es troben en marxa, vinent d'aquí el terme *elàstic*[19].

  Normalment les instàncies funcionen sobre Linux tot i que també existeix l'opció de fer servir Windows Server. En general existeixen diverses classes de màquines optimitzades per a segons quins tipus de tasques han de dir a terme[20]:

  - *Propòsit general*: amb un equilibri entre CPU, memòria, disc i xarxa, estan ideades per dur a terme un ampli espectre de tasques.
  - *Compute Optimized*: ofereixen una gran potencia centrada en la CPU i s'empren per processament en lots, servidors web o analítica distribuïda, entre d'altres.
  - *Memory Optimized*: amb grans quantitats de memòria, aquestes màquines estan centrades en bases de dades d'alt rendiment, mineria i anàlisi de dades, bases de dades en memòria i altres.
  - *Accelerated Computing*: ofereixen GPUs i CPUs molt avançades i s'empren per *Machine Learning*, física de fluids, modelatge mol·lecular i altres.
  - *Storage Optimized*: màquines amb alta capacitat de disc tant en Hard Disk Drive (HDD) com en Solid-State Drive (SSD). Ideades per aplicacions de computació distribuïda com Hadoop, *Data Warehousing* o, en el cas de SSD bases de dades no-SQL com MongoDB o bases de dades en memòria.

- **Elastic MapReduce (EMR)**: EMR és un altre servei d'AWS que proporciona *frameworks* com Hadoop, Spark, HBase o Presto que facilita i accelera el processament de grans quantitats de dades utilitzant les instàncies EC2 d'Amazon. A més permet la interacció amb dades d'altres serveis d'Amazon com ara *Redshift*, S3 o DynamoDB.

  Concretament, en aquest projecte s'ha fet servir EMR juntament amb Apache Spark, emprant la funcionalitat de Spark SQL. Diverses operacions es realitzen sobre gran *DataFrame* de manera que cada màquina virtual EC2 del *cluster* s'encarrega d'una petita part de la tasca.

- **AWS Lambda**: Lambda ofereix un servei que, a simple vista, pot semblar molt semblant al que pot oferir EC2. Lambda permet executar codi sense tenir cap servidor ni màquina sobre la que executar. Simplement s'ha de pujar el codi que es vol executar i el propi servei s'encarrega de proveir tot el que sigui necessari per a que funcioni. Els programes poden estar escrits en Node.js —JS—, Python, Java i C#. A més es pot definir *triggers* des d'altres serveis d'AWS que dispararan els programes que s'han pujat o es poden cridar directament des de qualsevol aplicació web.

  Aquest és el concepte de *serverless computing*, que permet desenvolupar i executar aplicacions i serveis sense haver de pensar amb servidors. D'aquesta manera les aplicacions s'executen en servidors però totes les gestions es fan automàticament per AWS.

  Per tant, què diferencia Lambda de EC2? Principalment la gestió dels recursos que obliga a fer EC2. Ens fa responsables de definir la capacitat i potència de la màquina mentre que Lambda ens allibera de qualsevol tipus de gestió.

  Tot i la seva semblança, aquests dos serveis se solen emprar per a tasques completament diferents. Lambda resulta molt útil quan s'han d'executar petites rutines en segons quins esdeveniments. En general no és un servei ideat per executar programes molt pesats o exigents. De fet, presenta certes limitacions, per exemple, una sola execució d'una petició no pot durar més de 300s, és a dir 5 minuts [21]. Això ens dóna una idea de quines tasques podem dur a terme amb aquest servei. En general està pensat per a fer de *back-end* d'aplicacions web i per a fer transformacions d'objectes quan es pugen a S3, per exemple. EC2 està pensat per tasques més exigents, tot i que també es fa servir de manera molt sovint com a servidor per a aplicacions web[22].

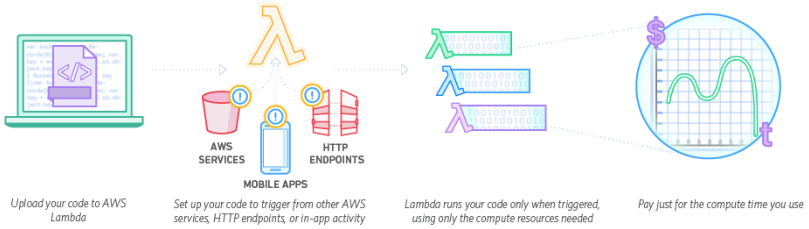


Figura 2.4: Esquema de Funcionament de Lambda [3]

2.4 Solucions i Eines Existents

En aquesta secció es parlarà de solucions ja existents que fan una funció similar al producte que s'ha desenvolupat o tecnologies que no s'han arribat a emprar en la implementació de la solució final. En general són solucions alternatives per a la representació de dades mitjançant gràfiques. Per aquestes alternatives identificam 3 nivells:

- **Llibreries**: com que es vol crear una eina pròpia i independent pot resultar interessant desenvolupar un *dashboard* emprant llibreries de visualització de dades:
  - *Google Charts*: una llibreria de Google totalment gratuïta amb una gran varietat de diferents gràfics molt fàcils d'emprar i de personalitzar. L'inconvenient que presenta emprar aquesta llibreria és que Google no permet als usuaris de *Google Charts* tenir la llibreria en local en un servidor, és a dir que una empresa que tracta amb dades confidencials probablement no sigui la millor eina per a visualitzar dades.
  - *Highcharts*: de la mateixa manera que la llibreria de Google, *Highcharts* ofereix una gran ventall de tipus de gràfiques i a més està basada amb Scalable Vector Graphics (SVG) i Vector Markup Language (VML), eines noves de HTML5, de manera que no necessita cap *plugin*. A més ofereix una interfície anomenada *Highcharts Cloud*, que permet l'ús de gràfiques interactives. Aquesta és d'ús gratuït per a l'ús personal i requereix la compra d'un llicència en cas d'ús comercial.
  - *Data-Driven Documents (D3)*: probablement sigui la llibreria més versàtil i potent que existeix en el camp de la visualització de dades. És de codi lliure i permet crear efectes visuals que actualitzen dinàmicament l'arbre DOM. És compatible amb tots els navegadors i és l'eina preferida pels experts que empren llibreries d'aquest tipus. Tot i la seva potència, D3 no té gràfiques predefinides, cosa que complica la tasca de creació de gràfiques des de 0 i presenta una corba d'aprenentatge molt inclinada.
  - *Superset*: anteriorment anomenat *Panoramix* i *Caravel*, *Superset* és una plataforma de representació visual de intuïtiva i interactiva. Està integrada amb les millors llibreries de visualització de dades i permet la creació de *dashboards* amb facilitat a més de tenir integració amb quasi qualsevol tipus de base de dades relacional. Principalment està escrit amb Python, així que pot fer ús de les avantatges que suposa tenir tots els paquets com *SqlAlchemy* per a la connexió amb les bases de dades. *Superset* permet incloure les gràfiques dins pàgines HTML com a *iframes*, permetent així els *dashboards* en les pàgines web. *Superset* encara es troba en desenvolupament i és de codi lliure, mantingut per Apache Incubator[23].

  Evidenment, el fet de desenvolupar el teu propi *dashboard* té certes desavantatges. Tot el desenvolupament s'ha de fer per part de l'empresa, des del tractament de dades fins a la representació en les gràfiques. Això implica més feina i més temps de desenvolupament. Tot i tenir certs inconvenients també presenta avantatges que cal tenir en compte: com que no depenem de cap marca i ens desenvolupam

(a) Google Charts (b) Highcharts (c) D3 (d) Superset

Figura 2.5: Les llibreries de representació de dades més rellevants

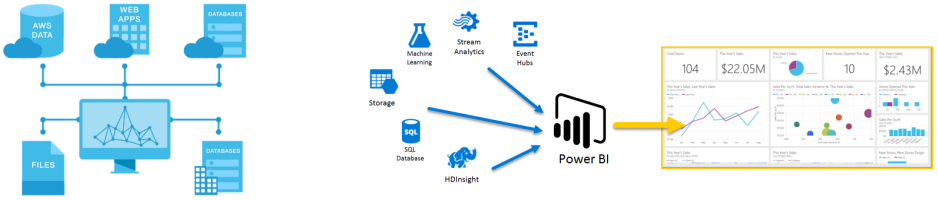
els mètodes per obtenir les dades, les podem tenir guardades en la configuració que vulguem, ja sigui en la propia casa en qualsevol tipus de base de dades o bé en el *cloud*. A més, aquestes llibreries solen ser gratuïtes o demanen unes llicències bastant assequibles, per tant retallam despeses per aquesta part, possiblement compensant el temps extra de desenvolupament que hem hagut d'assumir.

- **Tableau**: Tableau és una eina d'elaboració de *dashboards*. A partir d'una font de dades i una petita configuració ens permet elaborar gràfiques de tot tipus. En el nostre cas ens podria interessar fer servir Tableau per a representar els KPIs calculats en gràfiques, ja que simplement hauríem d'indicar quina classe de gràfica volem i de quina taula venen les dades. Tableau permet múltiples orígens de dades, des de Google Cloud o MS Azure fins a AWS.

Figura 2.6: Fonts *Cloud* de Tableau [3]

Cal comentar que Tableau és una eina molt usada dins el departament de D&A, ja que implica no haver de programar res, i que, per tant, accelera el desenvolupament de la plataforma. No obstant, també té certes desavantatges, la primera i més evident és la del preu: Tableau requereix que els usuaris comprin una llicència per fer-lo servir. A més, Tableau obliga en certa manera als usuaris a emmagatzemar les dades al *cloud* enlloc de poder tenir-les guardades en un servidor propi. Tampoc pot connectar-se a cubs OLAP. Tot i que és vera que existeix una altra versió de Tableau anomenada *Tableau Server* que soluciona aquests problemes, resulta ser encara més cara que la versió normal[24].

- **AWS QuickSight i altres**: Finalment la darrera alternativa és fer servir software com *QuickSight* o *Azure Power Bi*. Aquests són programes desenvolupats per les mateixes marques que ofereixen totes les eines d'emmagatzematges i computació al *cloud*. Degut a la integració amb aquests serveis es poden realitzar informes en temps real i crear gràfiques i *dashboards* de manera intuïtiva i ràpida. A més, en el cas de *AWS QuickSight* ofereix un motor de càlcul Super-fast Parallel In-memory Calculation Engine (SPICE) que fa tots els càlculs en el *cloud* i obté respostes ràpides.



(a) *AWS Quicksight*

(b) Azure Power BI

Figura 2.7: Eines de representació de dades de Amazon i Microsoft

La gràcia que tenen aquests serveis és, a part de la facilitat per crear *dashboards*, la flexibilitat que presenten quan s'ha de triar una font de dades. *AWS QuickSight* permet usar fitxers CSV, excel, emprar fonts com Amazon *Redshift*, Amazon RDS, Amazon S3 o Amazon EMR amb Spark.

Per tant, a grans trets, el que ens permeten aquests programes és l'elaboració molt ràpida de *dashboards*, sense invertir ni un minut programant. A més solen estar optimitzats per els seus propis sistemes i permeten un càlcul molt ràpid. A canvi d'això, però, els usuaris estan obligats a tenir contractats i usar l'emmagatzematge d'aquestes companyes, que no sempre pot resultar la millor opció. A més, tot i que ara comencen a sorgir solucions que permeten l'emmagatzematge local —*on premise*—, tot s'ha de fer forçosament en el *cloud*. No cal dir que també són eines que tenen un preu més elevat de caràcter mensual.

CAPÍTOL 3

DICCIONARI DE DADES

3.1 Descripció i Funcionament

No existeix una definició exacta del que és un diccionari de dades, tot i que grans empreses han fet les seves pròpies definicions. IBM ho defineix com *un repositori centralitzat d'informació sobre les dades com ara origen de les dades, relacions amb altres dades, utilització i format*[25], mentre que Oracle ho defineix com una col·lecció de taules amb metadades.

En el nostre cas particular la definició es correspon a un híbrid de les dues coses. El diccionari de dades o *Technical Dictionary* és una eina interactiva que permet veure totes les taules de la base de dades. En general, la IU s'estructura en 2 elements: Una caixa desplegable que conté una sèrie de possibles filtres que es poden aplicar i una taula en la que cada fila és una taula de la base de dades. Cada fila té la opció d'ampliar la informació permetent veure-la dividida en 3 subapartats: *Overview* —Visió General— , *Columns* —Columnes— i *Foreign Keys*. La informació que mostra, però, es troba emmagatzemada dins 3 taules de DynamoDB. Degut a que hi ha més de 3000 taules el diccionari consta d'un cercador amb el qual es pot filtrar segons diferents criteris, que es corresponen a 6 atributs de les taules. Concretament s'empren:

1. **`dwc_slot`**: indica a quina categoria correspon la taula dins del *DataWareHouse* corporatiu. Pot ser ACQ —*Acquisition*—, DWC —*DataWarehouse*—, CAL —Calculation— o EXP —Exploitation—.
2. **`table_name`**: fa referència al nom de la taula en qüestió.
3. **`data_area`**: es refereix a la temàtica de la taula, per exemple si conté informació sobre clients, circuits turístics, *bookings*...
4. **`table_schema`**:
5. **`creation_date_time`**: indica la data en la que es va guardar el registre. Es fa servir un *timestamp* de la forma `YYYY/MM/DD-HH:MM:SS`.

6. `database`: indica a quina base de dades pertany la taula. Actualment totes són de `hbgdwc`.

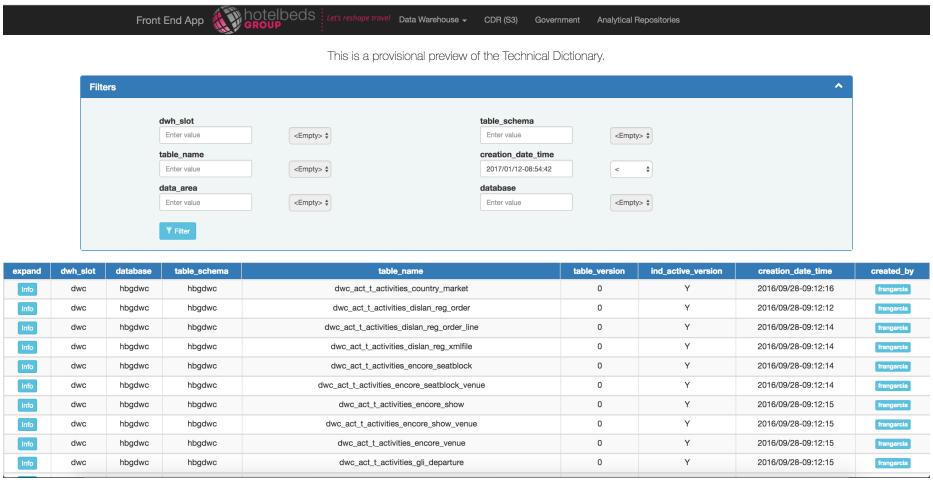


Figura 3.1: Vista principal del diccionari de dades

Tots els camps es poden filtrar per igual —=— i diferent —≠—, i la majoria permeten el filtratge per major —>—, menor —<—, major o igual —≥—, menor o igual —≤—. Pels camps `table_name` i `data_area` tenim la possibilitat de també aplicar el filtre *Contains*, que permet obtenir aquelles taules que continguin el que s'ha introduït dins el camp de text en el nom.

Un cop s'han introduït tots els filtres que es desitgen, l'usuari pot fer clic al botó *Filter* per a obtenir només aquells registres de les taules que compleixen els requisits. Això dispara una crida AJAX a un mètode de l'API que rep els filtres i retorna la informació pertinent. La resposta de l'API s'assigna a una variable de l'`$scope` anomenada `table_rows`, que és la que es fa servir per pintar les dades sobre la taula. Gràcies a que AngularJS està constantment comprovant si s'han actualitzat elements de l'`$scope` no hem de fer res més que assignar les noves dades a `table_rows` quan l'API ens ha contestat, així la taula s'actualitza visualment de manera automàtica, sense la necessitat d'emprar cap *script* de JS.

Quan es va dissenyar el diccionari de dades es va voler implementar un sistema de versions, de manera que segons quins canvis a les taules creassin noves versions. Distingim dos tipus de versions:

- **Versió**: direm que hi ha canvi de versió en una taula quan s'alteri l'estructura que tenia abans, per exemple si afegim, eliminam o canviam una columna d'una taula o modificam la PK.

- **Versió de Modificació**: El *Technical Dictionary* permet l'edició de segons quins camps, per exemple els de descripcions o que són de text lliure. Quan es fa un canvi d'un o més camps d'aquest tipus —venen marcats per una etiqueta que indica que un camp és editable— es crea un nou *item* amb els camps canviats i es puja a DynamoDB.

Els registres que es mostren a la taula són els que tenen la versió i la versió de modificació més alta. Tot i això, es poden consultar les versions anteriors emprant dos menús desplegables un cop hem entrat dins un registre de la taula. Un canvi en qualsevol de les dues versions actualitzarà els valors que tenia aquest registre en la versió seleccionada.

Quan es fa clic sobre el botó *info* d'un registre d'una taula s'obre un menú amb 3 pestanyes:

1. *Informació General*: Mostra tots els aspectes importants de la taula seleccionada. Això inclou els 6 camps que s'han detallat anteriorment i alguns altres. Tots els camps que són te text lliure són editables.

Figura 3.2: Pestanya de la informació general

2. *Columnes*: Detalla informació de totes les columnes que té la taula seleccionada. Com amb la informació general, també té alguns camps que són editables.

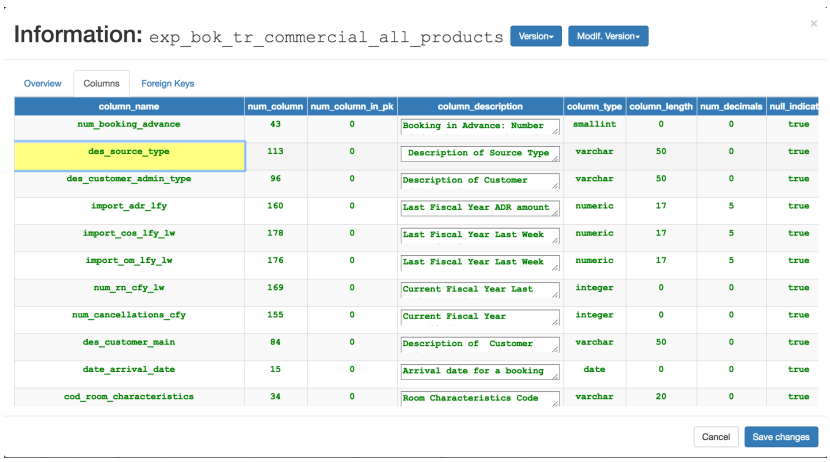


Figura 3.3: Pestanya amb la informació de les columnes

3. *FK*: Finalment aquesta pestanya mostra informació sobre les FKs de la taula en qüestió.

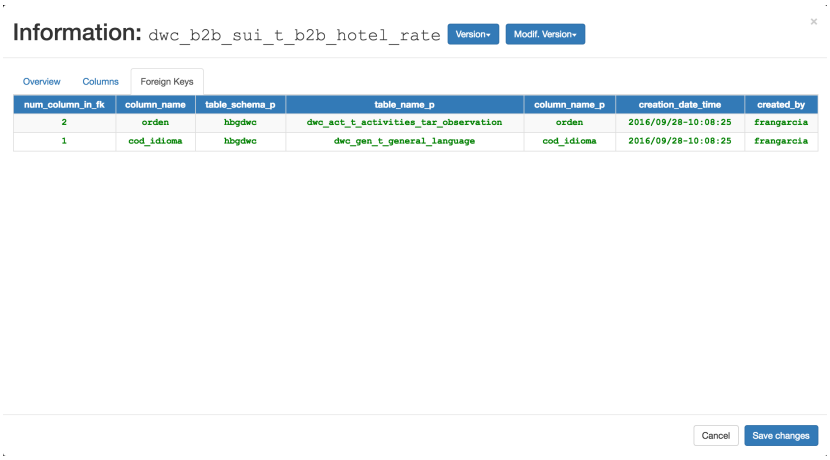


Figura 3.4: Pestanya amb la informació de les FKs

3.2 Connexió amb el *Back-End*

Per a aconseguir connectar-se des del client al servidor per a cridar algun mètode de l'API s'utilitzen cridades AJAX. Com que aquestes cridades es fan des del context d'AngularJS, empram el servei `$http`. Aquest servei facilita la comunicació amb servidors remots via HyperText Transfer Protocol (HTTP). En aquesta aplicació el servei es crida utilitzant una drecera depenent del que es vulgui aconseguir —*Get, Put, Post*—. Els mètodes del *Back-end* es criden mitjançant URLs. Aquesta indica a on s'envia la petició, que normalment és la direcció en la que es troba escoltant l'API. Suposant que l'API es troba en marxa a `http://localhost:5000`, direm que aquesta és la URL base. Per a cridar un mètode amb una cridada AJAX haurem d'especificar el nom del mètode que volem cridar.

Per exemple, si volem cridar el mètode `get_columns()` haurem de construir una URL que sigui `http://localhost:5000/getcols`. Com podem veure, el nom del mètode dins el codi de l'API no té per què coincidir amb el nom que li hem posat en la URL, sempre i quan estigui correctament configurat. En la secció corresponent es detallarà com es fa.

Finalment, per emprar paràmetres també es poden indicar a partir de la URL, i s'indiquen després de la URL base i el mètode, emprant ? com a separador per indicar que ara venen els paràmetres. Aquests s'especifiquen mitjançant un nom seguit d'un signe = i el valor del paràmetre. Si es volen enviar múltiples paràmetres es fa de la mateixa manera per a cada un concatenant un & entre paràmetre i paràmetre. Per tant, si volem cridar el mètode `get_columns()` amb 3 paràmetres, la URL tendrà la forma: `http://localhost:5000/getcols?table_name=name&version=0&slot=dwc`.

És possible que en una petició, enlloc de voler rebre dades en vulguem enviar, ja sigui per actualitzar un element de la base de dades —PUT— o per crear un nou

element —POST—. Per això, si volem enviar un JSON no ho farem com a paràmetre, sinó com a argument de la cridada del mètode POST o PUT, que és opcional.

Listing 3.1: Exemple de Petició amb AngularJS

```
var url = "http://localhost:5000/put_var/var";
var  = {foo : bar};
var responsePromise = $http.put(url, data);
responsePromise.success(
   function (response) {
      $scope.content = response.data;
   }
);
responsePromise.error(
   function () {
      $scope.content = "Something went wrong";
   }
);
```

Com es pot observar, un cop es fa la cridada es pot gestionar el que passarà un cop s'hagi completat la operació del *back-end*. En cas de que l'operació hagi estat completada correctament —codi 2xx— s'executarà la funció dins de `.success(...)`, en cas contrari —4xx, 5xx— s'executarà el codi de `.error()`. Cal comentar que, en cas de que el servidor retorni una resposta, aquesta pot ser accedida, ja que es el paràmetre de la funció anònima —*callback function*— que es crida tant en cas d'error com de èxit.

3.3 *Back-End*

En la gran majoria d'aplicacions web existeix un *Back-End* que s'encarrega de gestionar la interacció de les dades amb la base de dades amb les que treballa el *Front-End* —el que veu l'usuari—. En aquest cas, enlloc de fer una API tradicional SOAP es va decidir que es desenvoluparia una API tipus ReST.

3.3.1 ReST

Què és una API ReST? A grans trets, una API actua com a intermediari entre un usuari i un servei de qualsevol tipus, normalment Bases de Dades. Concretament, en el cas de la nostra aplicació, l'API té la funció de guardar dades a la base de dades provinents del front i retorna les dades que són sol·licitades per l'usuari.

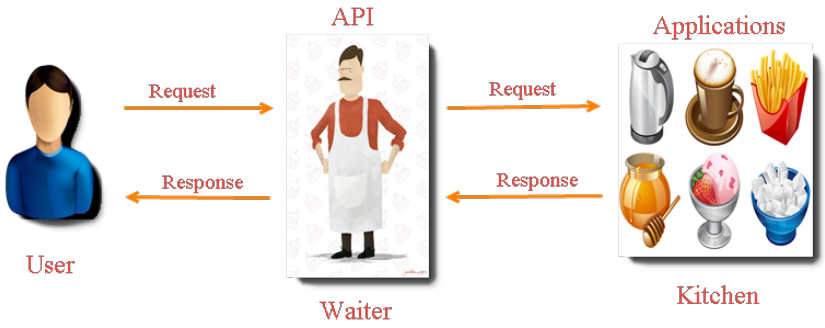


Figura 3.5: Abstracció que representa el funcionament d'una API [2]

En una API ReST les peticions que es fan a les Uniform Resource Identifier (URI) dels recursos sempre generen i retornen una resposta que estarà en JSON, XML, HTML o qualsevol altre format que estigui definit. Normalment, en una API d'aquest tipus la connexió es fa emprant el protocol HTTP[26]. Per tant, una API ReST no és més que un conjunt de recursos o funcions que són cridades per usuaris i retornen respostes. A grans trets s'utilitzen 4 tipus d'operacions diferents:

- **GET**: Proveeix accés de només-lectura —*readonly*— a un recurs.
- **POST**: Usat per crear un nou recurs o actualitzar recurs existent.
- **PUT**: Usat per crear un nou recurs o actualitzar recurs existent.
- **DELETE**: S'empra per eliminar un recurs existent.

Com podem veure, tant PUT com POST poden fer les mateixes funcions i quina ha d'exercir cada una d'elles encara genera cert debat. En general, s'empra POST exclusivament per a crear i PUT per a crear i per actualitzar recursos.

3.3.2 *Flask*

L'API per aquesta aplicació va ser desenvolupada amb *Flask*, un *microframework* de Python dissenyat per a desenvolupar aplicacions web. *Flask* es caracteritza per ser exageradament simple i flexible, oferint només la funcionalitat bàsica i permetent als usuaris afegir altres llibreries per implementar funcionalitats que no incorpora *Flask* nativament.

Per exemple, *Flask* no és un Object Relational Mapper (ORM), però gràcies al mòdul SQLAlchemy podem tenir aquesta opció per a gestionar els objectes de les bases de dades. *Django* seria l'altre *framework rival*, que ofereix una experiència *tot-inclòs* i que per tant no necessita *add-ons*, però és significativament més complex.

Gràcies a la seva simplicitat, crear una API que funcioni es pot fer amb menys de 10 línies de codi:

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def hello():
7      return "Hello, World!"
8
9  if __name__ == "__main__":
10     app.run()
```

Listing 3.2: Exemple d'una API amb Flask

Ara simplement consisteix en escriure els mètodes que siguin necessaris per a poder dur a terme totes les funcions que es desitja que l'aplicació web sigui capaç de fer. Aquests mètodes seran totalment diferents depenent de quin tipus d'aplicació s'està desenvolupant, tot i que en la majoria d'aplicacions web abunda la gestió de recursos de les bases de dades.

Per indicar quina funció s'ha d'executar amb una URL determinada es fa servir un decorador. Un decorador de Python és un canvi especific a la sintaxi del llenguatge que ens permet alterar la funcionalitat d'un mètode, funció o classe sense haver s'emprar una subclasse. En aquest cas ens serveix per definir quina URL s'espera per cridar cada funció.

Per exemple, en el nostre cas, quan es carrega la pàgina per primer cop es crida el mètode `scan_data()`, que retorna les metadades de cada taula segons els filtres que es passen per paràmetre, es crida amb la URL
`http://localhost:5000/scan?param1=foo&param2=bar...`
Per la banda del servidor, per tant, haurem de tenir un decorador que assigni aquesta URL —ignorant els paràmetres— al nostre mètode:
`@app.route('/scan', methods=['GET']).`
Aquest decorador indica que les URL què tenen la forma de `url_base + /scan` cridin la funció directament abaix del decorador i a més amb la restricció de que el mètode HTTP ha de ser forçosament GET.

3.4 Funcions de l'API

A continuació es llistaran i descriuran les funcions més importants que conformen l'API i que permeten el funcionament de l'aplicació web.

- `scan_data()`: Cridat amb la URL `/scan`, permet el mètode `GET`. Principalment s'encarrega de retornar un *array* de objectes JSON amb la informació de les taules. Com s'ha mencionat anteriorment, aquesta funció es pot cridar amb paràmetres o sense: si es crida sense paràmetres a la URL llavors la funció empra el *framework* `boto3` per llegir els primers 30 items de la taula. Normalment només es farà la cridada sense paràmetres quan es carrega la pàgina per primera vegada. Això es fa per que lla taula no estigui buida quan l'usuari la veu per primer cop. Aquesta informació es troba dins la taula `HBG.DWH.REDSHIFT.METADATA.DWH.TABLES`.

  D'altra banda, si s'incorporen paràmetres aniran per parelles: cada parella té un valor per el qual filtrar i un criteri per el qual filtrar. Per exemple, si filtram segons nom de taula "commercial" amb el criteri "contains" tendrem dos paràmetres, i s'aniran afegint depenent de quants filtres es vulguin aplicar fins a un màxim de 6, per tant 12 paràmetres.

  Per aplicar els filtres es fan servir les funcionalitats de `boto3` per a DynamoDB. Cal comentar que `boto3` ofereix 2 mètodes per recuperar ítems d'una taula: `scan()` i `query()`. Els dos aconsegueixen el mateix resultat però el primer llegeix tots els ítems de la taula i després aplica el filtre —més ràpid— mentre que el segon ho fa a mesura que llegeix. Per a construir la condició per filtrar s'ha creat un mètode que depenent de la condició de filtre —Contains, EQ (*equals*), NE (*not equals*), GT (*greater than*), LT (*less than*), GE (*greater or equals*) o LE (*less or equals*)— creï un objecte de la classe `Key` o `Attr` —depenent de quina condició s'empra una classe o una altra— i li apliqui la condició, que és un mètode que s'aplica sobre un objecte d'aquests tipus. Per exemple, si tenim una taula en la que volem obtenir els ítems que tenen l'atribut edat menor que 27, haurem d'escriure:
  `response = table.scan( FilterExpression=Attr('edat').lt(27) )`

  Si tenim múltiples filtres, amb el primer es crearà l'objecte i amb els següents s'aniran afegint amb l'operador `&=` fins que s'hagin afegit tots. D'aquesta manera podem crear condicions dinàmicament sense haver-nos de preocupar sobre quants de filtres tenim.

- `post_data(inf)`: Aquest mètode és molt simple i es crida amb la URL `/put/item`. Igual que per llegir, per a introduir nova informació a la taula es fa servir el mètode de `boto3` anomenat `put_item(...)`, que rep com a paràmetre la informació que ha de posar a la taula. La informació ve en format JSON i la creació de l'objecte es fa a la banda del client amb un procés molt simple, emprant els elements de l'`$scope`. L'objecte JSON s'envia com a `data` de la request.

```
inf = request.get_json(silent=True)
table.put_item(
    Item=inf
)
```

- **`update_cols(cols)`**: Respon a la URL `/updateColumns/cols` i s'encarrega de persistir els canvis que ha fet l'usuari emprant l'aplicació web a la base de dades. En aquest cas no es treballa sobre la taula de taules sinó sobre la taula de `HBG.DWH.REDSHIFT.METADATA.DWH.COLUMNS`. Un cop es fa clic al botó de guardar canvis s'envia un *array* de objectes JSON que contenen les dades que pot haver modificat l'usuari.

  Un cop les rep el servidor actualitza individualment els ítems amb un bucle amb la operació `update_item()`. Aquest mètode necessita que li especifiquin quina és la clau —per a determinar específicament quin ítem s'està actualitzant—, una *UpdateExpression*, que determina quins atributs s'actualitzen i finalment una *ExpressionAttributeValues*, que assigna els valors als identificadors emprats en la *UpdateExpression*.

- **`get_columns()`**: Aquest mètode serveix per obtenir les dades sobre totes les columnes d'una taula. Es crida amb la URL `/getcols`. En aquest cas, enlloc de emprar el mètode `scan()` es fa servir `query()`, ja que no necessitam tanta velocitat degut a que aquestes dades no són immediatament visibles —es troben a una altra pestanya—. La condició de cerca és que el nom de la taula sigui igual que el de la taula que s'està consultant —`table_name`— i que l'*slot* en el *data warehouse* també coincideixi —dwh_slot—. El nom de la taula i l'*slot* s'envien com a parametre de la URL.

- **`get_versions()` i `get_modif_versions()`**: Aquests mètodes s'encarreguen de retornar totes les versions i versions de modificació d'una taula en concret respectivament.

- **`get_pks`**: De manera idèntica a les columnes, aquest mètode obté les FKs de la taula `HBG.DWH.REDSHIFT.METADATA.DWH.FOREIGNKEYS`. Es crida amb la URL `/getfks`. La única diferència amb les columnes és que la pestanya d'informació de les FKs no permet la modificació de cap camp, per tant no existeix cap operació d'actualització.

CAPÍTOL 4

Data Entry

4.1 Descripció

Durant molts anys han existit llocs de feina que es centren en introduir i actualitzar dades de qualsevol tipus, depenent del context. Tot i que ara és molt menys comú, encara existeixen perfils així, que empren eines com *Microsoft Excel* per guardar dades de l'empresa. Fins i tot, tot i que no sigui la principal tasca d'un treballador, el fet d'haver d'introduir les dades manualment fa que hores de feina no es dediquin a la tasca important, fent així un mal ús dels recursos de l'empresa i perdent eficiència.

És per això que resulta interessant per totes les empreses que fan feina constantment amb moltes dades poder automatitzar tots aquests processos que consisteixen en validar i guardar dades en repositoris de dades de l'empresa. Especialment, dins *Hotelbeds* existeixen múltiples departaments que reben o confeccionen a diari grans quantitats de dades, ja sigui en format *.CSV* o *.xls* / *.xlsx* —Format que empra *MS Excel*— i que normalment acaben guardades dins bases de dades. La majoria dels treballadors que conformen aquests departaments no són usuaris tècnics cosa que implica que altres treballadors tècnics hagin de completar el procés de consolidació de les dades en les bases de dades, a més de tota la burocràcia que sol implicar el fet de crear una o múltiples taules dins un ambient corporatiu.

Per evitar tota aquesta problemàtica i optimitzar els recursos de l'empresa en les tasques realment importants es va idear el *Data Entry*, una eina que automatitza la pujada de fitxers en bases de dades. Aquesta eina consta d'unes definicions que descriuen certs paràmetres dels fitxers que s'esperen per pujar, com ara el nom, el nombre de columnes o la freqüència de pujada. La idea del *Data Entry* és que hi hagi un usuari autoritzat que creï les definicions de manera que després els altres usuaris simplement seleccionin una definició i puguin els seus fitxers.

Quan s'efectuen pujades de fitxers es deixa constància d'informació com qui ha fet l'intent de pujada, l'hora i altres factors en una taula d'auditoria. Seguint la mateixa línia que amb el diccionari de dades, tant les definicions de fitxers com els registres d'auditoria es troben dins taules de DynamoDB.

4.2 Funcionament

El funcionament d'aquesta eina és molt simple: en un principi es fa una petició per crear una o definició de fitxer ja que es preveu que s'hauran de guardar noves dades a la base de dades. Per crear la plantilla s'han d'especificar paràmetres com:

- **Nom de la plantilla**: Especifica el nom d'aquesta plantilla, no s'ha de confondre amb el nom del fitxer o fitxers que es volen persistir a la base de dades.
- **Patró del nom del fitxer**: indica amb una expressió regular quina forma ha de tenir el nom del fitxer en cas de tenir múltiples parts; per exemple, si cada dia obtenim un fitxer amb les reserves d'hotels es possible que obtinguem fitxers que tenguin de nom `bookings_DDMMYYY.csv`, de manera que ens convé tenir una manera d'indicar això.
- **Base de Dades**: A quina base de dades s'ha de guardar el document.
- **Nom de la Taula**: La taula a la qual s'ha de guardar el document.
- **Delimitador**: Tot i que normalment els fitxers CSV separen els valors emprant el caràcter coma —,— es poden emprar altres caràcters.
- **Periodicitat**: Com s'ha comentat abans, es possible que dades del mateix tipus vagin arribant periòdicament, per tant es podem esperar dades diàriament, mensualment o de manera aperiòdica, és a dir, quan no tenim una regularitat establerta.
- **És Seqüencial?**: Un fitxer és seqüencial si les dades d'un fitxer complementen les que ja existien dins la taula de la base de dades. Si no és seqüencial significa que podem eliminar tot el que ja existia abans de guardar els nous registres.
- **Permet Reprocessament?**: Un fitxer permet reprocessament si podem tornar a pujar un fitxer que ja s'ha pujat anteriorment.
- **Attribute Parameters**: És un JSON que conté l'especificació de les columnes del fitxer que es vol pujar. Per a cada columna tendrem una JSON amb informació com el tipus de dades, la longitud o si pot ser nul o no.

A més d'aquests paràmetres també n'existeixen de metadades com ara usuari o *timestamps* de creació. Un cop s'ha creat una definició completa es pot realitzar la pujada del fitxer. Per a fer-ho s'ha de seleccionar un fitxer que compleix els requisits d'alguna plantilla. Quan l'usuari ha seleccionat el fitxer pot fer clic al botó de validar. Això dispara una funció de l'API que s'encarrega de comprovar que el contingut del fitxer compleix totes les restriccions que s'han declarat en la plantilla.

La validació es separa en dues parts, la comprovació dels *headers* de les columnes —si en té— i la validació del contingut en si. A mesura que es van trobant errors durant la validació del fitxer es va construint un *String* que primer conté els errors de *headers* i després els errors de les dades. Quan s'ha completat la validació l'API retorna aquest *String* al client. En cas que l'API retorni un missatge d'error buit —no hi ha hagut errors— es mostra un missatge positiu indicant que la validació ha estat correcta i

s'activa el botó per fer la pujada dels registres del fitxer a la base de dades. En cas contrari es mostren detalladament tots els errors que s'han donat durant el procés de validació. No cal dir que en aquest cas no es podrà començar la pujada a la base de dades.

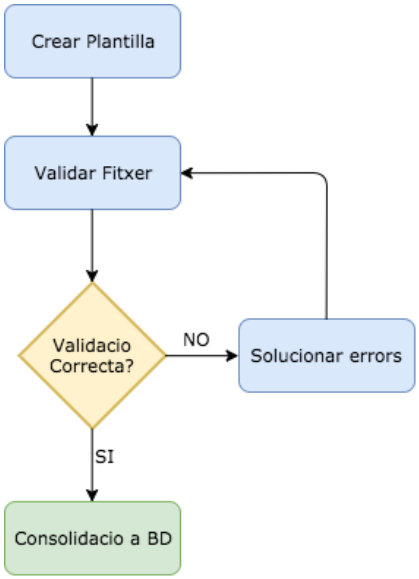


Figura 4.1: Diagrama de flux del funcionament del *Data Entry*

A més dels errors del contingut del fitxer —tipus incorrecte, longitud incorrecta, format incorrecte ...— hi poden haver altres errors. Per exemple, la data pot ser incorrecta, el format del fitxer no és el correcte o s'ha triat la opció *Append* —afegir les files noves enlloc d'eliminar les prèviament existents— i s'ha intentat copiar una fila amb una PK que ja existia dins la taula.

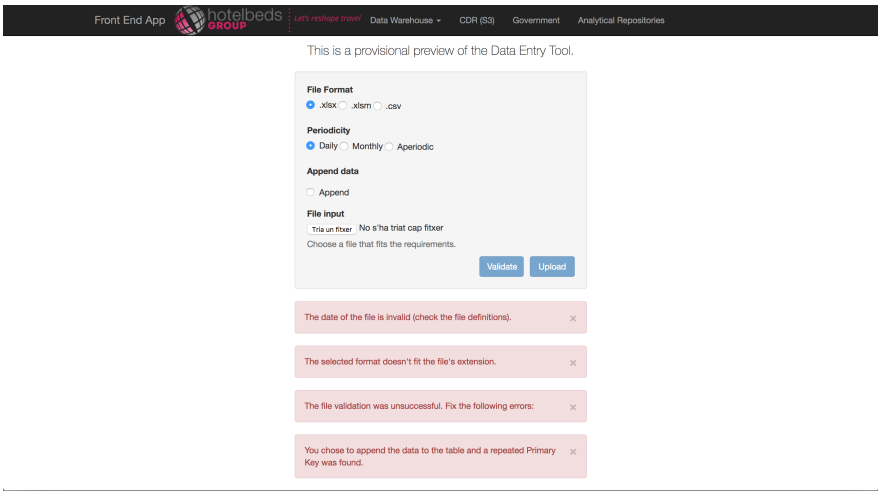


Figura 4.2: Altres possibles errors del *Data Entry*

Quan l'usuari fa clic al botó validar l'script JS crea un objecte `FileReader()` que llegeix tot el text del fitxeri el passa com a dades amb la cridada AJAX a l'API. Per tant, el tractament de les dades es fa en forma d'*String*.

El procés de pujada de les dades es pot fer de dues maneres:

1. Iterant sobre cada linia del fitxer, de manera que per a cada línia se separen els elements emprant el separador indicat a la plantilla i creant sobre la marxa una *query* SQL que faci un `INSERT` a la taula indicada de les dades que acabam de llegir. D'aquesta manera es fan tantes *queries* com línies tengui el fitxer, cosa probablement ineficient.

2. Fent servir la comanda SQL `COPY`. Aquesta comanda copia dades entre un fitxer i una taula de la base de dades [27]. A més, permet emprar diferents fonts de dades a més del sistema de fitxers local com per exemple S3. Com que aquesta comanda s'executa al servidor, el que es fa és crear un fitxer temporal i es puja emprant el *framework* `boto3` a S3, que s'utilitza com a passarel·la per fer la còpia. Per indicar on es troba el fitxer dins S3 es fa servir una URL que especifica el *bucket* i directori, concretament `s3://hbg-cdr-landing-dev/Data_Entry/file.csv`.

La opció que es va triar va ser la segona, ja que simplifica molt la tasca degut a que la gestió la fa el propi SGBD enlloc d'haver-se de fer manualment.

El *Data Entry* també ofereix la opció d'annexar les noves dades a les ja existents marcant un *checkbox*, de manera que si guardam les dades del dia actual no s'eliminen les dades prèviament existents. Si no es marca aquesta opció s'executarà la *query* `TRUNCATE table_name;` que elimina totes les files que tenia aquesta taula.

4.3 Estat Actual

Quan es va desenvolupar aquesta aplicació es va fer, en primera instància, com a experiment per a veure si era viable i útil. Quan es va veure que alguns usuaris realment necessitaven l'eina i que serviria per optimitzar els recursos de l'empresa va ser quan es va decidir crear una eina robusta, juntament amb el diccionari de dades i altres. És per això que es va refer l'eina emprant la següent versió d'*Angular*, *Angular 2*.

En general el funcionament de l'aplicació és molt semblant, tot i que té algunes diferències. Quan es va a pujar un fitxer s'ha de seleccionar la plantilla corresponent. A més, en aquesta versió relaxa la validació dels valors de les columnes de manera que només es tenen en compte els altres paràmetres. Per facilitar la tasca de creació de plantilles també es va implementar un editor que permet crear les definicions de fitxers que després se seleccionaran que es vulgui pujar un fitxer.

Una altra diferencia clau és que la validació no es duu a terme en el mateix moment que es produeix la pujada. En aquest cas es puja el fitxer a S3 i es marca com a pendent. El processament dels fitxers es fa mitjançant un *planner*, que s'encarrega llançar el procés, per tant, mentre estigui pendent de processar estarà marcat així. Sinó es marcaran com a *Succes* o *Failed* depenent de si han passat la validació o no respectivament.

Com a primera idea s'havia pensat que l'API funcionés amb els serveis de AWS Lambda i API Gateway, però aquesta opció va ser descartada ja que no es té la certesa que es pugui acabar l'execució d'una cridada amb 300 segons, que és el temps màxim

que pot estar en marxa un procés amb Lambda. Per tant es va optar per utilitzar una màquina EC2 com a servidor, que no consta d'aquestes limitacions.

4.4 Funcions de l'API

Igual que amb el diccionari, s'exposaran a continuació les funcions més rellevants que són necessàries per que aquesta eina funcioni.

- **`get_defs()`**: Funció que retorna les definicions de fitxers que existeixen dins la taula de DynamoDB. Es fa una cerca incondicional, ja que en la primera versió es prova de verificar amb totes les definicions fins que s'en troba una que compleix els requisits. Es crida emprant la URL `/get_definition`.

- **`get_uploads()`**: Fa el mateix que la funció anterior però consultant la taula d'auditoria. Essencialment és necessari per a saber si anteriorment s'ha pujat un fitxer amb el mateix nom. Si el *flag* de reprocessament no es troba a *true* llavors això provocarà un error. Té com a URL `/get_uploads`.

- **`put_upload(upl)`**: Crea un nou ítem dins la taula d'auditoria de DynamoDB cada vegada que s'intenta una pujada d'algun fitxer. En aquest ítem s'indiquen alguns paràmetres sobre el fitxer com ara la taula, la persona o entitat que l'ha pujat o un *timestamp*. Se li va posar `/put_upload` com a la seva URL.

- **`upload_file_redshift(file)`**: Funció que s'encarrega d'executar la comanda que puja el fitxer a una taula de *Redshift*. Com ja s'ha explicat abans, aquesta funció rep el contingut del fitxer en la *request* en format *String*, de manera que es crea un fitxer temporal en el que s'escriu aquest gran *String* i es puja a S3. Un cop s'ha pujat es construeix una *query* de la forma:

  ```
  COPY table FROM 's3://hbg-cdr-landing-dev/Data_Entry/file.csv'
  CREDENTIALS 'aws_access_key_id=xxxx;aws_secret_access_key=yyyy'
  IGNOREBLANKLINES
  DELIMITER 'delim'
  CSV;
  ```

  A continuació es crea un objecte connexió a la base de dades i un objecte cursor amb aquesta connexió, que es farà servir per executar la *query* anterior. Si el *flag* `Append` —annexar— que ve amb la *request* es troba a *true* s'executarà abans un `TRUNCATE` per buidar la taula.

- **`validate(file)`**: Aquesta funció analitza tot el contingut del fitxer i comprova que totes les dades i metadades són correctes. Això vol dir assegurar que les columnes del CSV tenen el tipus i longitud adequat —quan apliqui— i que el nom, i la periodicitat siguin les correctes.

CAPÍTOL 5

OLAP *Engine*

5.1 Descripció

L'OLAP *engine* és la darrera eina que vaig desenvolupar. Aquesta consisteix en un programa que permet calcular els KPIs de manera simple i implicant la menor burocràcia possible. Aquesta eina seria emprada en gran part pels usuaris de negoci, que prenen les decisions importants en funció de certs indicadors. En les grans empreses aquests KPIs són extremadament importants i el càlcul de la majoria involucra molta feina ja que se solen calcular amb programes dissenyats *ad hoc*.

L'objectiu d'aquesta eina és eliminar totes aquestes limitacions i complicacions i crear un sistema que permeti a qualsevol usuari de negoci o Key Account Manager (KAM) generar els seus propis KPIs.

5.2 Funcionament

Per a aconseguir això determinarem una taula d'on provenen les dades i especificarem unes dimensions i unes mètriques amb les que crearem un cub OLAP. Tot i que un cub és estrictament en tres dimensions, quan es parla en termes OLAP feim referència a un hipercub en un nombre arbitrari de dimensions. Aquestes dimensions, en aquest cas, són les columnes de la taula i poden ser conceptes com ara temps, lloc o producte.

Figura 5.1: Exemple d'un cub OLAP

Les mètriques, per altra banda, són els valors que desitjam calcular, és a dir, els KPIs. El que es fa, per tant, és agrupar totes les dimensions i definir les mètriques. Aquestes tenen la forma d'una sentència SQL de la forma: `AGREGACIO(FORMULA) AS NOM`, on AGREGACIO és `AVG`, `COUNT`, `SUM`, `MAX`, `MIN`, `FIRST` o `LAST`, FORMULA és una operació que representa el KPI i el NOM representa el nom que li hem posat a aquesta mètrica.

Aquestes operacions es fan fent servir el motor de processament de dades *Spark*, amb l'API de Python. Concretament amb les funcionalitats de *Spark SQL* podem tractar les taules com a elements en memòria fent servir els objectes *dataframe*.

Primer llegim la taula base d'on proven les dimensions i guardam els resultats dins un *dataframe*. Només inclourem les columnes que formin part de les dimensions o siguin necessàries per el calcul d'alguna mètrica. D'aquesta manera tendrem les dades que necessitam per a efectuar els càlculs dels KPIs en memòria accelerant el temps que trigarem en computar totes les mètriques. *Spark* SQL amb *Redshift* — `spark-redshift`— llegeix i escriu dades a S3 quan transfereix dades a/des de *Redshift*, de manera que resulta necessari especificar on hauria d'escriure aquestes dades. A més `spark-redshift` no pot eliminar aquestes dades temporals, de manera que és necessari establir una política de cicle de vida al bucket per assegurar que aquests fitxers temporals s'eliminen[28]. En l'apartat de tecnologies hem vist un exemple de com es llegeix 2.7.

Gràcies a *Spark* SQL podem executar *queries* SQL directament sobre un *dataframe* com si es tractés d'una taula normal. Combinant això amb el fet que aquest script s'executarà sobre un *cluster* amb EMR aquest procés que probablement impliqui Gigabytes de dades trigui una fracció del temps que trigaria amb una sola màquina.

Finalment, per crear el cub s'executa la següent *query* sobre el *dataframe*:
`SELECT dims + metriques FROM table GROUP BY dims ORDER BY dims`
Això retorna un *dataframe* amb les mètriques calculades, per tant el següent i darrer pas és inserir aquestes dades a la taula que li correspon.

```
1  df.write                                          \
2  .format('com.databricks.spark.redshift')          \
3  .option('url',  redshift_url)                     \
4  .option("user", REDSHIFT_USER)                    \
5  .option("password", REDSHIFT_PASS)                \
6  .option('temporary_aws_access_key_id', access_key)  \
7  .option('temporary_aws_secret_access_key', secret)  \
8  .option('temporary_aws_session_token', token)     \
9  .option('tempdir', tempdir)                       \
10 .option('dbtable', table_name)                    \
11 .mode('append')                                   \
12 .save()
```

Listing 5.1: Escriure a una taula emprant Spark SQL

Un cop s'ha executat podem trobar el resultat a la taula. Emprant un programa com *SQLWorkbench* veurem que efectivament els KPIs s'han calculat. En el següent exemple podem veure els resultats calculats amb les dimensions `cod_receptive`, `date_load_date`, `ind_booking_status` i `cod_currency`.

cod_receptive	date_load_date	cod_currency	ind_booking_status	num_hour	num_booking_advance	num_cancellation_in_advance	num_cancellation_post_creation
439	2016-10-31	EUR	A	36	42	34	8
459	2016-10-31	EUR	A	229	437	72	365
461	2016-10-31	EUR	A	408	0	0	0
489	2016-10-31	EUR	A	66	336	24	312
164	2016-10-31	GBP	A	4647	18700	5944	12756
197	2016-10-31	GBP	A	40	968	272	696
235	2016-10-31	GBP	A	84	2208	1254	954
318	2016-10-31	HKD	A	830	8681	6186	2495
277	2016-10-31	KWD	A	6	30	22	8
69	2016-10-31	MXN	A	3130	26433	3764	22669
235	2016-10-31	MXN	A	40	204	88	116
296	2016-10-31	MXN	A	92	60	4	56

Figura 5.2: Resultats després d'executar l'script

5.3 Estat Actual

Vista la utilitat que té aquesta eina es va decidir incorporar-la dins la plataforma *Data Governance*. Aquesta incorporació va implicar certs canvis importants en la forma de funcionar de l'aplicació: el canvi més important és l'estructuració de les dades. A partir d'ara les dades no provendran d'alguna taula de la base de dades sinó que s'estructuren en interfícies; una gran taula que conté tots els camps referents a una mateixa àrea temàtica. Aquestes interfícies són l'estàndard i totes les dades provinent d'empreses relacionades amb *Hotelbeds* es guardaran d'una manera normalitzada.

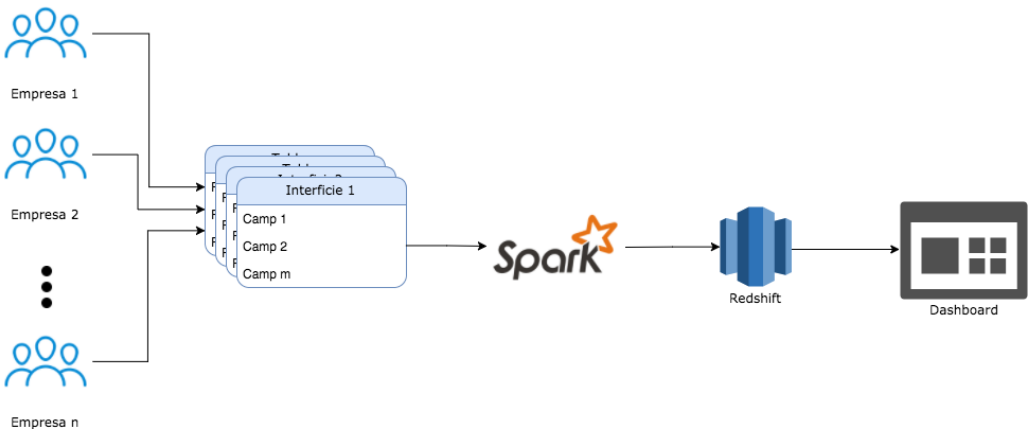


Figura 5.3: Diagrama de flux del funcionament de l'eina

Un cop les interfícies estan plenes de dades es poden calcular les KPIs amb la mateixa logística que es tenia abans. Els resultats es guarden dins taules per tal que després puguin ser explotats. La manera més fàcil i intuïtiva d'entendre un KPI i veure la seva evolució és mitjançant gràfiques. Després d'experimentar amb el servei d'AWS *QuickSight* i obtenir uns resultats no del tot convincents, es va optar per desenvolupar un *dashboard* propi fent servir *Superset*.

Les dimensions en aquesta versió s'agrupen en *Contextes Dimensionals* ja que és molt comú que els KPIs es calculin fent servir les mateixes dimensions. A més, s'ha creat un editor de KPIs per facilitar la tasca de la creació per a qualsevol usuari. Aquest editor permet consultar els KPIs ja existents i els seus detalls així com la creació de nous.

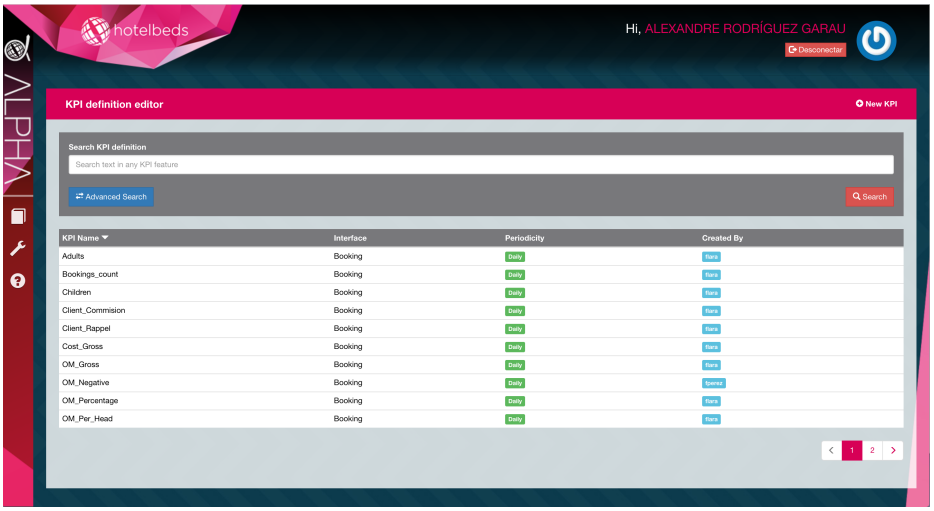


Figura 5.4: Vista principal de l'editor de KPIs

Amb aquest nou enfocament els KPIs fan referència a una interfície i tenen una periodicitat, és a dir es calcularan les noves dades cada cert temps. També es va afegir la opció d'afegir un filtre per només considerar certs resultats —el corresponent a un `WHERE` en SQL—. Tots aquests detalls es poden consultar fent clic sobre qualsevol registre.

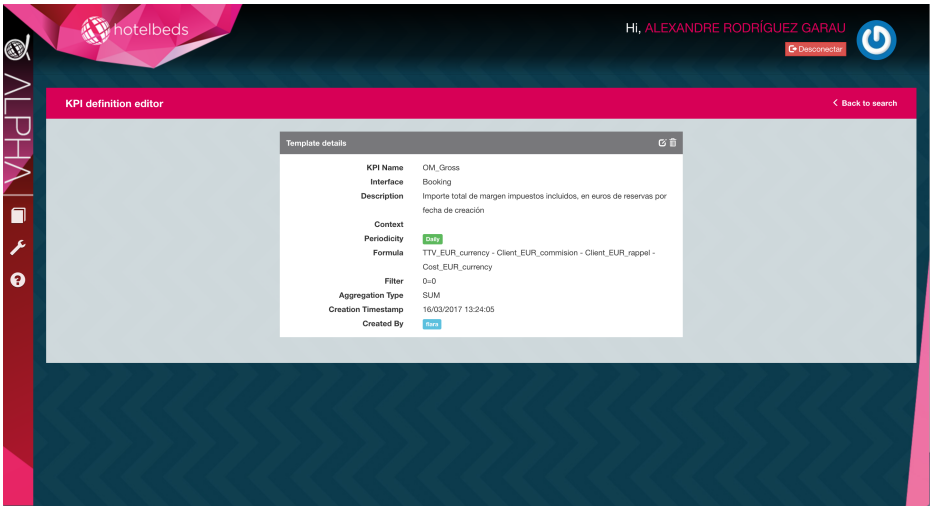


Figura 5.5: Vista detallada d'un KPI

En una de les primeres versions l'editor de KPIs requeria que l'usuari introduís

la formula del KPI directament en format SQL cosa que va totalment en contra de la finalitat que té aquesta eina, que és facilitar la tasca als usuaris de negoci no-tècnics. En un futur es va crear un editor de formules que no requereix cap tipus de coneixement de la sintaxi SQL i transforma una fórmula matemàtica en una SQL.

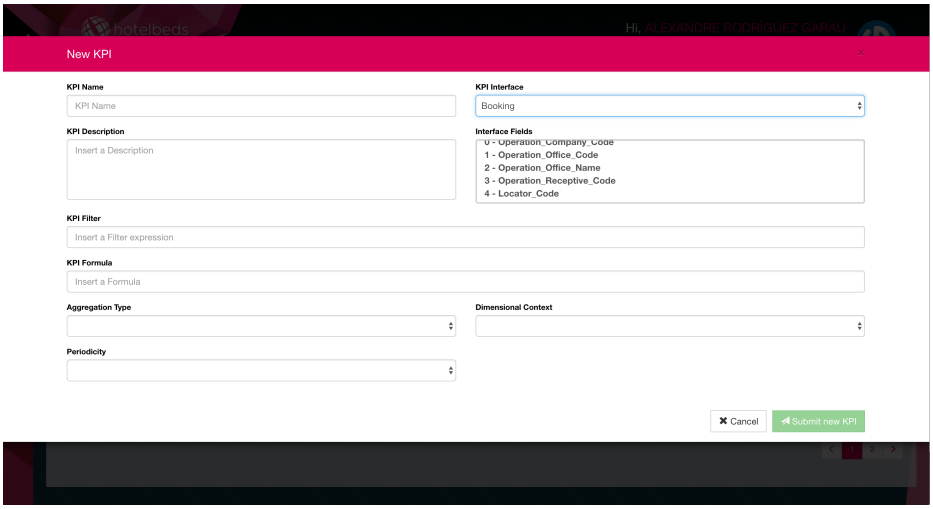


Figura 5.6: Creació d'un KPI

D'altra banda ens trobam amb un cas pràcticament idèntic amb les interfícies. Es va crear un editor d'interfícies que permet crear i afegir camps.

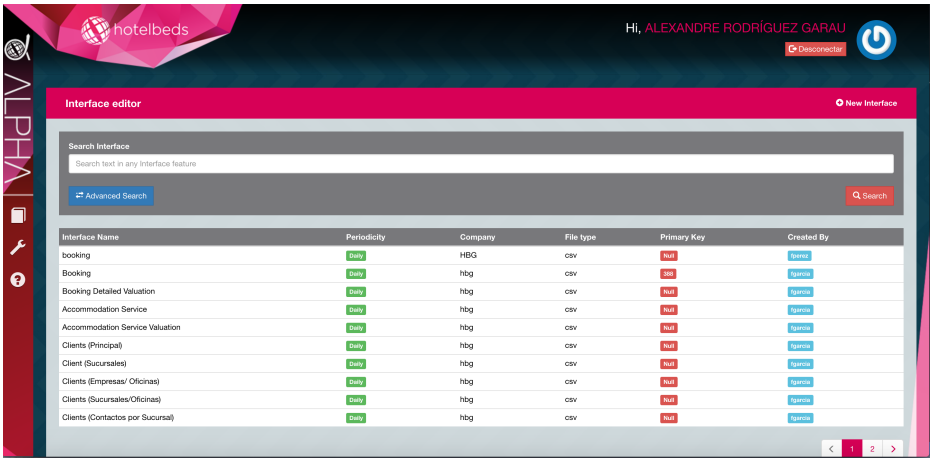


Figura 5.7: Vista principal de l'editor d'interfícies

Per a cada interfície es poden consultar tots els seus camps i alguns detalls més com ara descripcions o qui l'ha creada. Quan es fa clic sobre una interfície es mostra aquesta informació i un llistat de tots els seus camps, destacant el tipus del camp, si és PK o el seu nom.

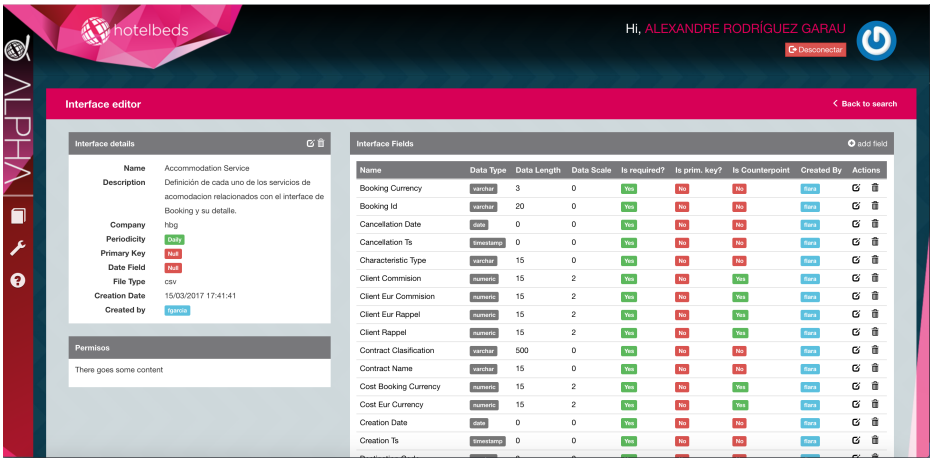


Figura 5.8: Camps d'una interfície determinada

Cada camp pot ser editat o eliminat. Per editar ens apareixerà un menú en el que podem canviar el valor de la majoria de paràmetres. També permet crear nous camps.

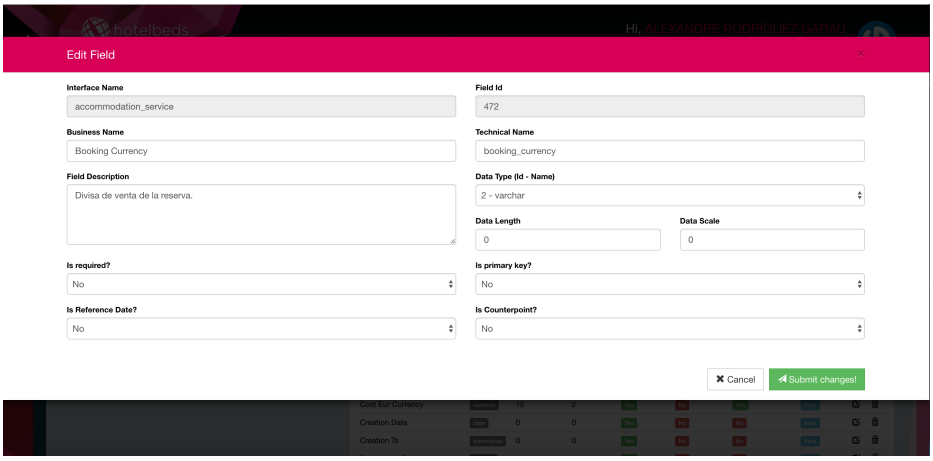


Figura 5.9: Edició d'un camp d'una interfície

Finalment hem de considerar el *dashboard*. Aquesta és la part més important ja que permet explorar l'evolució de les dades que hem calculat. Està distribuït de la següent manera: a la part superior trobam una barra que ens permet seleccionar quin KPI estam visualitzant. També ens permet canviar la data d'inici i de fi, per si volem veure la seva evolució en un període determinat. Sota aquesta barra trobam uns petits marcadors que indiquen els KPIs més importants de l'empresa.

Sota aquests, venen les gràfiques. Són tres: la primera ens mostra una previsió de com evolucionarà el KPI seleccionat en el període indicat per l'usuari. Les altres dues corresponen a les gràfiques que mostren l'evolució del KPI segons el client i el KPI en total —tots els clients—

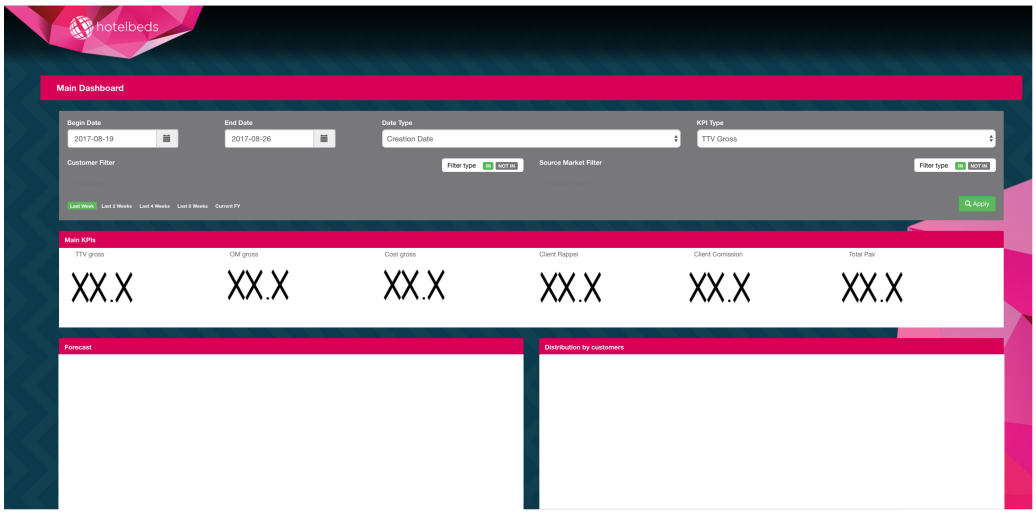


Figura 5.10: *Dashboard* de consulta de KPIs

CAPÍTOL 6

CONCLUSIONS

Amb el desenvolupament de la plataforma *Data Governance* no només he tengut l'oportunitat de realitzar un TFG diferent, sinó que he après a fer feina en un equip de desenvolupament real, interaccionar amb *Project Managers* i a adaptar-me a les situacions, ja que he hagut d'aprendre a fer servir tecnologies que no havia fet servir mai i programar en tres llenguatges que desconeixia.

Degut a que *Hotelbeds* és una empresa tan gran he pogut accedir a eines i recursos que en una empresa més petita probablement no hagués pogut. Això m'ha permès experimentar amb utilitats com gran part dels serveis d'Amazon —AWS—, que cada vegada són més populars, o amb eines de *Big Data* i programació distribuïda, utilitzant *Spark* i bases de dades tant relacionals com no-SQL.

En general, un de les coses que més importants consider haver après, a part de tota la part tècnica i de coneixements, és la importància de fer les coses de manera ordenada. Des de l'etapa de disseny, passant per la planificació de les tasques i l'execució d'aquestes, cada passa és igualment important, cosa de la que me n'he adonat experimentant-ho en primera persona. Tot i que és cert que implica més burocràcia i sol significar un inici del desenvolupament més lent, els resultats que s'obtenen després són, en la majoria de casos, millors.

El *Data Governance* aporta molt de valor dins de la pròpia empresa, estalviant moltes hores de feina que es podran invertir en activitats realment productives. És una plataforma que, un cop estigui acabada, serà emprada a diari i elimina les dependències d'eines de pagament com ara podria ser *Tableau*, que costa diners i obliga a dependre d'una plataforma de tercers, que no sempre és recomanable.

Tot i que en l'actualitat es troba acabat i en funcionament, el *Data Governance* probablement rebi actualitzacions i noves eines a mesura que vagi passant el temps i sorgeixin noves necessitats dins de l'ambient empresarial.

BIBLIOGRAFIA

[1] Google, "Visió conceptual d'angularjs," 2017. [Online]. Available: https://docs.angularjs.org/guide/concepts (document), 2.2

[2] Wikipedia, "Angular2," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Angular_(application_platform) (document), 2.1, 2.3, 3.5

[3] Amazon, "Lambda," 2017. [Online]. Available: https://aws.amazon.com/lambda/ (document), 2.4, 2.6

[4] w3schools, "What is new in html5," 2016. [Online]. Available: https://www.w3schools.com/html/html5_intro.asp 2.1

[5] Wikipedia, "Cascading style sheets, css," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Cascading_Style_Sheets 2.1

[6] ——, "Javascript," 2017. [Online]. Available: https://en.wikipedia.org/wiki/JavaScript 2.1

[7] ——, "Ajax," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Ajax_(programming) 2.1

[8] ——, "Jquery," 2017. [Online]. Available: https://en.wikipedia.org/wiki/JQuery 2.1

[9] ——, "Angularjs," 2017. [Online]. Available: https://en.wikipedia.org/wiki/AngularJS 2.1

[10] Microsoft, "Typescript, starts and ends with javascript," 2017. [Online]. Available: https://www.typescriptlang.org/ 2.1

[11] ——, "Typescript," 2017. [Online]. Available: https://en.wikipedia.org/wiki/TypeScript 2.1

[12] Wikipedia, "Postgresql," 2017. [Online]. Available: https://en.wikipedia.org/wiki/PostgreSQL 2.2

[13] ——, "Amazon redshift," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Amazon_Redshift 2.2

[14] Amazon, "Redshift i postgresql," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Amazon_Redshift 2.2

[15] ——, "Redshift i les seves mancances," 2017. [Online]. Available: http://docs.aws.amazon.com/redshift/latest/dg/c_unsupported-postgresql-features.html 2.2

[16] ——, "S3," 2017. [Online]. Available: https://aws.amazon.com/s3/ 2.2

[17] Wikipedia, "Object storage," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Object_storage 2.2

[18] Amazon, "Boto3," 2017. [Online]. Available: https://boto3.readthedocs.io/en/latest/ 2.3

[19] Wikipedia, "Ec2," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud 2.3

[20] Amazon, "Ec2 i els tipus de màquina," 2017. [Online]. Available: https://aws.amazon.com/ec2/instance-types/ 2.3

[21] ——, "Lambda i les seves limitacions," 2017. [Online]. Available: http://docs.aws.amazon.com/lambda/latest/dg/limits.html 2.3

[22] ——, "Lambda i les seves limitacions," 2017. [Online]. Available: https://aws.amazon.com/lambda/faqs/ 2.3

[23] Apache, "Superset," 2017. [Online]. Available: http://superset.incubator.apache.org/index.html 2.4

[24] Tableau, "Diferencies entre tableau online i server," 2017. [Online]. Available: http://static1.squarespace.com/static/553d5b0ce4b09e094f8e2464/t/555279ede4b0ccb06d6c589c/1431468525834/Tableau+Server+vs+Tableau+Online.pdf 2.4

[25] IBM, *IBM Dictionary of Computing*, 10th ed. McGraw-Hill, Inc., 1993. 3.1

[26] Wikipedia, "Representional state transfer," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer 3.3.1

[27] Postgre, "La comanda `COPY`," 2017. [Online]. Available: https://www.postgresql.org/docs/current/static/sql-copy.html 2

[28] Databricks, "Configuring a s3 bucket to hold temporary files," 2017. [Online]. Available: https://docs.databricks.com/_static/notebooks/redshift.html 5.2