

# Study of the problems and solutions of information aggregators: UAB mobile app

Karim Kammach Montiel

**Resum**– L'època moderna es essencialment caracteritzada per les tecnologies de la informació. Malgrat l'alt nivell de disponibilitat de dades, existeix la necessitat de dissenyar sistemes capaços de proporcionar informació de manera ràpida i amb el mínim nivell d'esforç per part de l'usuari per tal de facilitar el procés de búsqueda. Aquest article presenta el procés de desenvolupament per crear un aplicació que actua com a agregador de contingut que soluciona les dificultats d'ubicació i sobrecarrega d'informació que existeixen en el web de l'universitat Autònoma de Barcelona. Això s'aconsegueix mitjançant l'implementació d'una funció al núvol sense servidor que col·lecciona dades del web de la universitat, analitza i modifica els resultats, i els guarda en una base de dades per tal de satisfer les peticions de l'aplicació.

**Paraules clau**– Agregador de contingut, Rascador web, Funció al núvol sense servidor, Planificador de funcions al núvol, Operacions asíncrones

**Abstract**– The modern era is essentially characterized by information technology. Notwithstanding data availability, there is the urge to design systems capable of providing information quickly and effortlessly for the user. This paper presents the development process of creating a content aggregator application that solves data overloading and location issues existing in the Autonomous University of Barcelona web services. This is achieved by implementing a serverless cloud function that scrapes the university web service, parses the results, and stores them to a database to satisfy the application requests.

**Keywords**– Content aggregator, Web scraper, Serverless cloud function, Cloud function scheduler, Asynchronous operations

## 1 INTRODUCTION

**N**OWADAYS, the use of the mobile phone as a source of information has become indispensable, replacing encyclopedias, newsletters, and even traditional computers. Mankind has the entire knowledge of human history in a device no larger than the size of its hands. The main concern at this stage is information overload, as well as having many different information sources, causing difficulties in understanding the data. Nevertheless, different software tools have been developed over the years to give a solution to these matters, although new techniques and improvements are needed: the content aggregators.

- Contact e-mail: karim.kammach@e-campus.uab.cat
- Branch: Information Technologies
- Tutor: Victor García Font( Departament d'Enginyeria de la Informació i de les Comunicacions Àrea de Ciències de la Computació i Intel·ligència Artificial)
- Curs 2018/19

This scenario is present in the Autonomous University of Barcelona. The information available on its web is very scattered in different services and sometimes duplicated, becoming a doubtful source of information. Students may search in different services for university-related data, such as professor office address and subject information in the web service, faculties and restaurants location in *Google maps*, and even the availability of books in the library digital directories.

The main goal of this paper is to propose and offer a solution for the information overload and location problems, focusing on the Autonomous University of Barcelona niche, designing, building, and testing an information system made of a mobile application and a serverless cloud infrastructure. Furthermore, this paper explains the technique used to obtain, parse and store the data obtained from the different university services, and it analyzes the communication method in which the serverless cloud infrastructure and the mobile application send data in order to perform

CRUD operations <sup>1</sup>.

The rest of this article is structured as follows. Section 2 states existing projects that share similar goals with the one presented in this paper.

## 2 RELATED WORK

Before reaching the first stages of the project, some research has been done in favor of the implementation correctness of every phase cycle and for taking as an example similar successful approaches.

As a result of the huge amount of variety of information encountered on the internet, users end up using a content aggregator to facilitate the information search process. Such is the impact of these tools that big companies implemented their own. For instance, *Google* developed *Google News*, a tool that gathers the most recent news based on a previous poll regarding the user's interest [1]. Another successful example of a content aggregator is *Feedly*, a software tool that collects news feeds from a variety of online sources for the user to customize and share with others [2].

There are a lot of content aggregators available and for lots of platforms. The main difference among them is the particular niche where they focus.

Commonly, these tools are designed by implementing a web scraper that operates periodically over time to detect new information. In order to achieve this purpose, a cloud structure is designed. This is a computed solution for developers to create single-purpose functions that respond to Cloud events without the need to manage a server or a runtime environment [3].

## 3 OBJECTIVES

The main objective of this project is to facilitate the process in which the students search for university-related information, making it easier, quicker and clearer. Therefore, to achieve this goal, a set of sub-objectives need to be satisfied:

- Analyze various content aggregators, study the classic difficulties of these implementations, and introduce efficient solutions.
- Collect the requirements of the project by interviewing several students.
- Design a system in which the students can access up-to-date information in a clearer and quicker way than the actual scheme.
- Implement a user-friendly application interface based on a paper prototype.
- Implement a back end architecture adapted to receive requests from the clients and to retrieve data from the university multiple services.
- Guarantee application data availability by programming the back end code in a time-based job scheduler.

## 4 METHODOLOGY

The procedure to accomplish the previously listed objectives is the waterfall methodology with iterative feedback, introduced by Dr. Winston W. Royce [4]. This is an iterative model that allows the possibility of moving back to prior steps if there is a need. The waterfall technique consists of five stages: analysis and requirements acquisition, design, implementation, testing, and delivery.

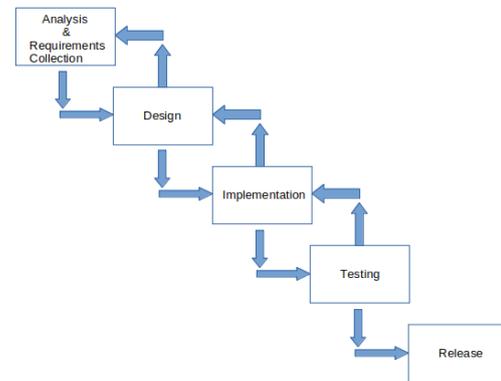


Fig. 1: Iterative waterfall methodology

The reason for choosing this method instead of other popular ones such as *Agile* is on account of the fact that the objectives are very clear from the start of the project and there is not much expectation concerning changes in the project.

The following pages describe how the previous steps have been effectuated.

## 5 REQUIREMENTS ACQUISITION

The first stage of the project is the requirements elicitation. This process collects what data has to be gathered from the university services by realizing various conversations among students and examine its viability.

To obtain the requirements, some students of the university have been interviewed to have a better understanding of the information that they request the most on the web services. Before this process, a study on how to obtain software requirements has been done to prepare a suitable interview so that all the information from the interviewed student can be extracted in benefit of the application to develop. [5].

The material used in this process is available in the Appendix.

## 6 DESIGN

Essentially, the purpose of this stage is to reflect the solution plan, focusing on the best option among the different alternatives, having into account economic and technical perspectives.

As soon as the data to be fetched is clear, it is precise to determine how to obtain, store, and serve the information to the users. In order to achieve this, several activities have taken place.

Firstly, bearing in mind the results given by the requirements collection process and the information that will be

<sup>1</sup>Create, Read, Update, and Delete

manipulated and displayed on the application, an analysis of the data model is structured.

Secondly, the application architecture is designed and discussed, focusing on the main characteristics of content aggregators based on previous research.

Finally, with the results of the previous tasks, a paper prototype and a navigation map are proposed for the content aggregator application.

### 6.1 Data modeling

In order to obtain information from the web services, there is the need to design a mechanism to accomplish this purpose. Similar to the stated projects in Section 2, the system to build needs to collect the data from the web by scraping it. This is a traditional approach when populating a content aggregator with data.

Moreover, since the data to be fetched consists of keys and values, the use of a NoSQL database for storing the information fits like a glove. These databases are non-relational, a method that does not follow the traditional classification of data into the form of tables, rows, and columns. Instead, data is stored in documents, which are organized into collections. Each document contains a set of key-value pairs. Documents can contain sub-collections and nested objects, both of which can include primitive fields like strings or complex objects like lists.

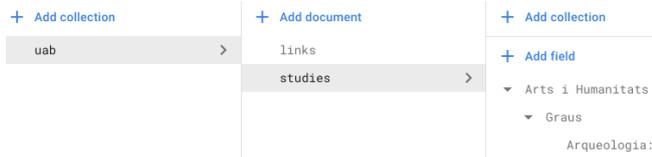


Fig. 2: Firestore database

This type of database is designed to overcome the limitations of relational databases when it comes to data that is growing and moving fast, being more scalable and flexible than their relational counterparts [6]. Additionally, NoSQL databases offer flexible schema design, making it much easier to update the database. On top of that, the system can be horizontally scaled by taking advantage of cheap servers. Besides, these databases are typically open source, so it is not necessary to pay any software licenses [7].

Since the application frontend is implemented by an object-oriented programming language, a class diagram has been designed so that the application structure can be better understood.

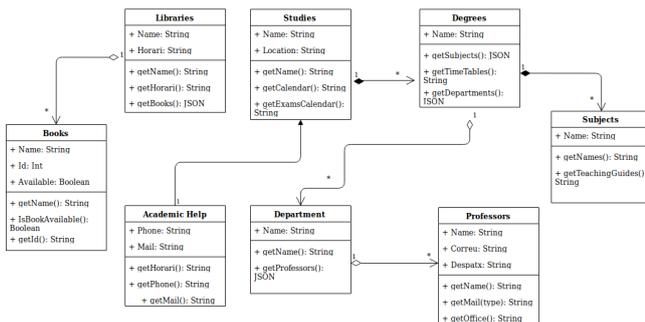


Fig. 3: Frontend application class diagram

### 6.2 Architecture

The solution presented here is to offer a technique that allows users to get data from a database which content is being periodically updated by a cloud function that scrapes the university web service. Two structures are being designed: the frontend user application that performs queries to the database and displays the data, and the backend implemented by a cloud function<sup>2</sup> that populates the database with the information retrieved from the website. The second one is of central interest since the project is more concerned about the information system behind, rather than the software design itself.

This scheme can be defined as a serverless computing based structure, in which there is no need to worry about the underlying infrastructure such as configuring, provisioning, load balancing, sharding, scaling, capacity, maintenance, and infrastructure management. Thus, all efforts are put to write efficient and reusable code.

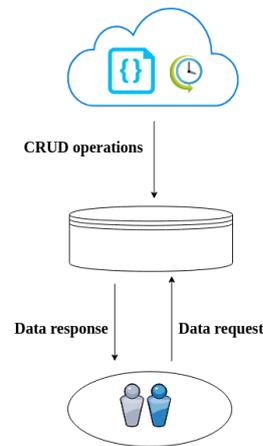


Fig. 4: System architecture

Notice the importance of the scraper function running on the cloud. This function is the core of this system since it is the mechanism that gathers web service data. Therefore, the implementation of this function is considered the bottleneck of the system, and an efficient implementation of it is critical in order to optimize the necessary time to fetch data from the web, parse it, and store it to the database through CRUD operations.

A problem arises when designing this software. As soon as the scraper function finishes collecting data, it must have sufficient autonomy to decide what database operations should be done with that information. On account of the fact that these operations strongly depend on the data existing on the database, the solution proposed here is to compare the data fetched from the web services with the existing entries in the database. Consequently, if these two data sets are not equal, the function must create new entries and/or update the existing ones. This comparison can be done examining the resulting hash functions of both data sets. Hence, the backend renders a high level of both availability and dynamism.

<sup>2</sup>Computed solution for developers to create single-purpose functions that respond to Cloud events without the need to manage a server or runtime environment.

### 6.3 Paper prototyping

Prototyping is one of the main techniques used in software projects since it helps to figure out what is going to be built. Instead of expensive Prototypes, a throwaway paper prototype method is suggested for requirements engineering. This technique is based on a visual representation of what the System will look like. It can be hand drawn or created by using a graphics program. Usually, a paper prototype is used as part of the usability testing, where the user gets a feel of the User Interface [8].

Previous to the software prototyping, a study and research about both Android and Apple design methods have been done to implement a suitable software design. This research has been focused on both operative systems design documentation [9] [10].

On this step, a student has been selected to realize a usability test with the previously developed paper prototype. As a result, the process has clarified some aspects and errors of the application in terms of design, such as the well-known design term "ocean of buttons" and badly located information screens.

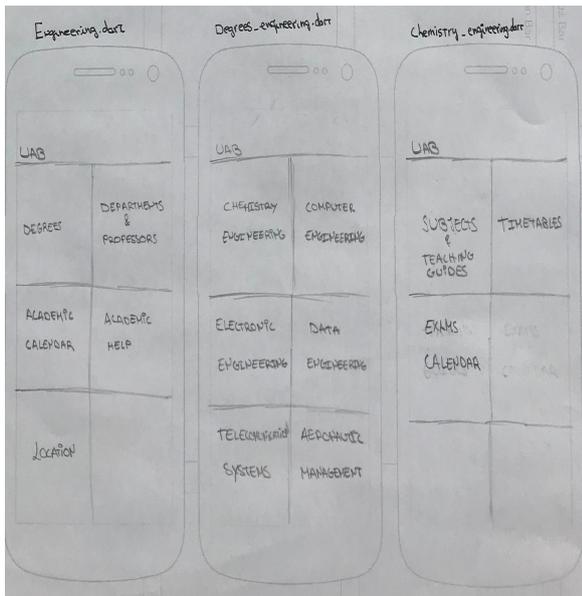


Fig. 5: Paper prototype of the application frontend

### 6.4 Navigation map

This technique is used to obtain a comprehensive understanding of the application magnitude, revealing how the application screens are related to each other and how to navigate through them. Figures 6 and 7 show the navigation map of the content aggregator.

## 7 IMPLEMENTATION

Coding takes place in this phase. The conceptual solution from the previous stage is translated into the creation of a functional product.

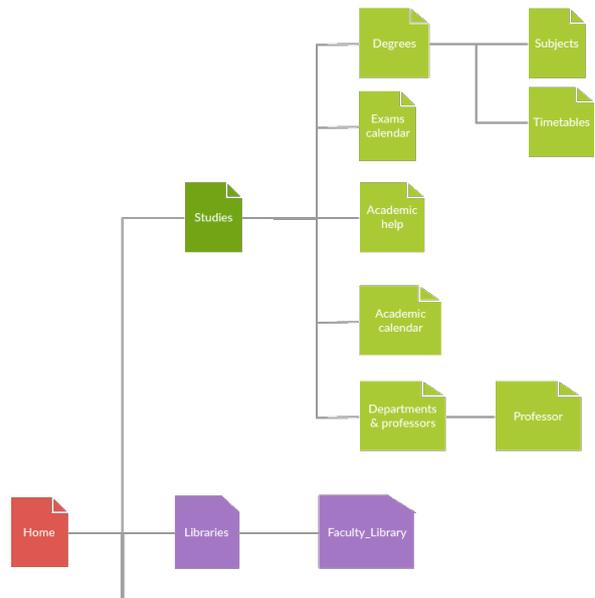


Fig. 6: Application navigation map

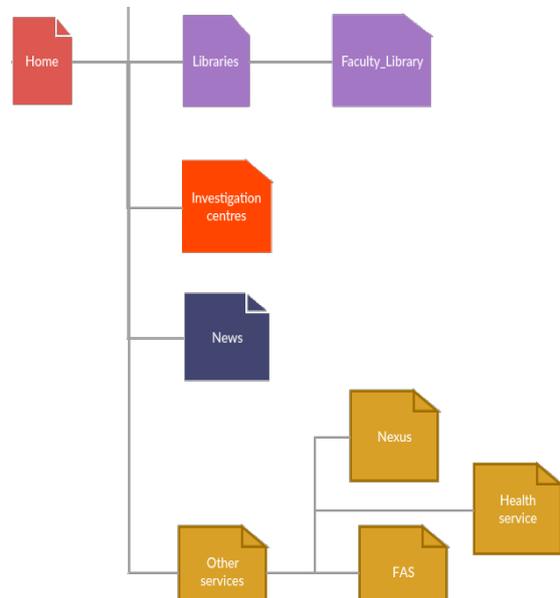


Fig. 7: Application navigation map

### 7.1 Previous considerations

Considering the designed system discussed in Section 3, the solution is based on scraping the web getting the data of interest, parse it, store it in the database, and serve it to the application frontend every time a user performs a petition. The main strength of this approach is that data is gathered directly from the web site view, therefore, if the web developers create new content, the application will be capable of collecting these new implementations. This guarantees a high level of dynamism.

Nonetheless, this implementation has its difficulties. The process of creating a considerable amount of HTTP requests in such a short time could lead to an IP blocking process and therefore a failure in the scraping method.

On top of that, since verified data is stored in the database, stability is guaranteed. Notice the contrast between the stability of the application and the dynamism of the web service, caused by changes in its view. Here the

time to process a request will be less, since the API has only the relevant data, instead of the full web service.

## 7.2 Web scraping using Nodejs

### 7.2.1 Introduction

Web scraping is defined as a technique of getting useful information from web pages, removing the hassle of having to browse pages manually and providing automation. The three main steps involved in this process are: get the HTML source code from the website, make sense of that content by parsing the data and extract the blocks of interest, and send that information to a storage system.

In this project, Nodejs<sup>3</sup> is the chosen technology to develop the backend structure. Nevertheless, this could have also been accomplished with other languages such as Python, but because of the Nodejs asynchronous nature, really helpful for the scraper developed in order to not to block the main thread of execution an obtain efficient results, for the overwhelming documentation and examples available on the internet and other sources, and the excellent libraries available for web scraping, Nodejs has been selected.

Following, a piece of code extracted directly from the actual content aggregator is exposed, in order to present the scraper main functionality.

```

51 var optionsGrausLink = {
52   url : jsonRoot.links.grausLink,
53   encoding: "latin1", // per treure els accents i evitar problemes
54   transform: function (html){
55     | return cheerio.load(html);
56   }
57 };
58
59 /**-----SECOND REQUEST-----*/
60 await rp(optionsLinksGraus)
61 .then(($=> {
62   console.log('inside second request, scraping ' + optionsLinksGraus.url);
63   //-----AGAFEM ELS STUDIES-----
64   var listStudies = [];
65   $('h3.section-title').each(function () {
66     | listStudies.push($(this).find('span').text());
67   })
68   jsonRoot.studies = {};
69   jsonRoot.links.studies = {};
70   listStudies.forEach(element => {
71     | jsonRoot.studies[element] = '';
72     | jsonRoot.links.studies[element] = '';
73   })
74 })
75 .catch((err) => {
76   | console.log('error in rp(options2).then()' + err);
77 })
78

```

Fig. 8: Scraper code

The prior function makes a request to a certain URL using the request-promise library<sup>4</sup>, captures its HTML content in the variable \$, parses it extracting the data of interest, and pushes it into a list (81-83). Finally, the list values are stored in a global JSON where all the information retrieved from the UAB web is stored for later comparison with the data from the database to determine if the data should be updated.

Notice that there are some special instructions on this code. It is using a special request library that returns a Promise. Nodejs nature gives priority to micro-tasks, so functions that involve asynchronous activity are queued to the next event loop. For this reason, this library is much

needed, since a function that returns a promise is required in order to take advantage of the await keyword, together with the async instruction in front of the function call. Await helps to control the execution flow and the event loop, telling the program to wait for the function to return the result before continuing the execution.

### 7.2.2 Libraries

Regarding web scraping in Nodejs, two main libraries are essential: request-promise<sup>5</sup> and cheerio<sup>6</sup>. The first module is a client Node package designed for making HTTP calls. It is a simplified version of the HTTP module. This library is used together with the cheerio package, an implementation of core JQuery designed for the server that helps developers to interpret and analyze web pages. It is used to traverse and manipulate the DOM. Thanks to it, it is easy to manipulate the HTML data obtained by the request module.

In the project, the request module is used to make the HTTP request to the root web service of the university: "https://www.uab.cat". Next, cheerio is used to manipulate and store the data of interest. Here, it is interesting to highlight the automation level of the implemented web scraper. By giving just the root link to the university web service, the scraper is capable to retrieve other links and access to deeper information from lower level web pages of the university. This is commonly known as web crawling.

## 7.3 An automotive technique to detect new fetched data

In order to check the new information retrieved from the scraper and decide if the database should be updated, the code first checks if the data retrieved at some key value, such as "professors" is empty or not. If it is empty it means that the scraper could not get the data from the web, probably because of a change in a class name in the HTML web code. After this first comparison, if the data fetched is not empty, the hash function (SHA256) of both data sets are compared. If both hash functions are equal, there is no need to update because the data is the same. A simpler explanation of the process is found in figure 9.

## 7.4 Serverless application backend

As seen in Section 6.2, the backend of the environment is composed of a database and a cloud structure. This section covers the implementation aspects of that structure part.

As described in Section 7.2, the backend consists of a function running in the cloud that scrapes the university web service and decides which database operations should be performed with the fetched data. This design has been implemented using *Google cloud serverless platform*, specifically the products *Google Cloud Functions*, the *Google Cloud Scheduler* and the *Google Cloud Firestore NoSQL database* to store and synchronize the data fetched from the university web services. Given its prices (payments depending on the use), user friendliness, support, documentation, and other features, this service has been preferred to develop the implementation of the design.

<sup>3</sup>Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.

<sup>4</sup>Simplified HTTP request library with Promise support

<sup>5</sup><https://www.npmjs.com/package/request-promise>

<sup>6</sup><https://cheerio.js.org/>

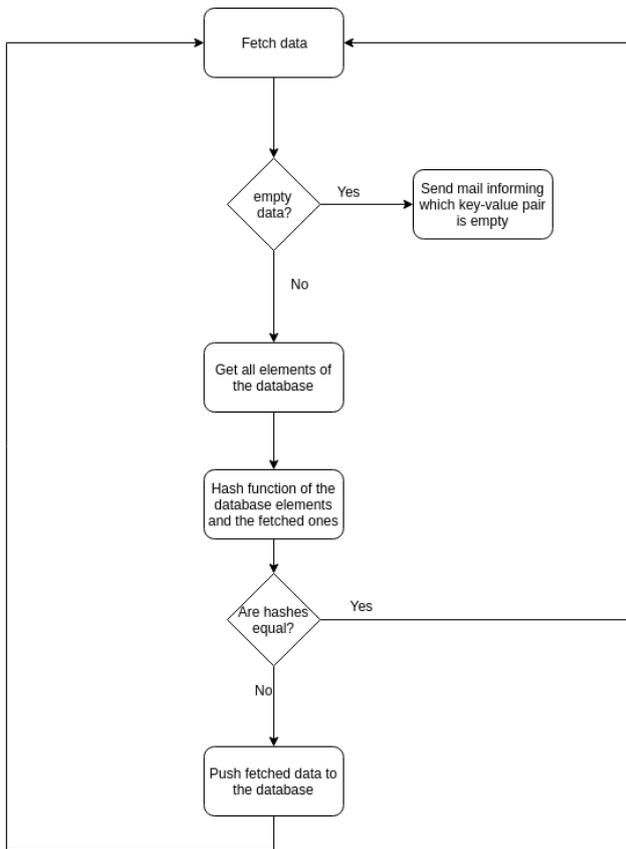


Fig. 9: Scraper update method flux diagram

## 7.4.1 Cloud Firestore database

### 7.4.1.1 Characteristics, features, and capabilities

Among all the different existing database products, *Cloud Firestore* is the chosen one in this project. This Software supports a data model that fits like a glove with the data manipulated by the scraper cloud function developed. The data fetched from the web services is stored in a JSON, which is mainly composed of keys and values. In the same way, Firestore data model supports hierarchical data structures and stores data in documents, organized into collections as explained in Section 6.1.

Furthermore, it uses data synchronization to update data on any connected device. As a result, users can see real-time updates. Besides, it is designed to scale, bringing a powerful infrastructure that guarantees consistency and good latency results.

On top of that, this database provides several tools that are helpful to check statistics and logs such as daily active users, failed queries and invocations, crashes and errors, tests and grow-predictions, among others.

### 7.4.1.2 CRUD operations

The main database operations are: create, read, update and delete. Essentially, these operations create or add new entries, read or retrieve existing data, update or edit entries, and delete data.

The procedures that involve creating and updating entries have been implemented in Nodejs, as the web scraper has been written in this language as detailed on earlier sections. The only operation that has not been used is Delete since the

scraper only creates data or updates it if there have been any changes in the university website, and the Read operation is performed in the frontend, to display the data.

For the sake of simplicity and to clarify further explanations, consider the following example:

```

var uab = {
  studies: {
    Enginyeries: {
      EnginyeriaInformatica: {
        Professors: {
          name: '',
          office: '',
        },
        Subjects: {
          name: '',
          course: '',
        }
      }
    }
  },
  services: {
    SAF: {
      description: '',
      contact: '',
    },
    HealthCentre: {
      description: '',
      contact: '',
    }
  }
}
  
```

Fig. 10: JSON variable example

It is important to highlight that, in the project code, the JSON is dynamic, the key values of it are other objects. For instance, the key “studies” is an object of keys containing every study category retrieved from the university website, such as Engineering, Sciences, Humanities, among others. In the previous example, the JSON has been hard-coded for simplicity.

This is how the scraper will behave when it has to put the Professors properties and subjects of *EnginyeriaInformatica* into the Firestore database:

```

admin.firestore().collection('uab')
.doc('studies/Enginyeries/EnginyeriaInformatica')
.set(uab.studies.Enginyeries.EnginyeriaInformatica);
  
```

Fig. 11: Create operation in the scraper function

Here, “admin” is a variable that stores the required library “firebase-admin”, so that the scraper function has the authorization to manage the database with root permissions (a further explanation about different roles and permissions of the database can be found in the very next section). After accessing the database as admin, CRUD operations can be implemented. The keyword “collection” is used to access the specified collection “uab”. Subsequently, one can access to its documents with the *doc* keyword and push data to that entry in the database with the *set* keyword, specifying the data to be pushed.

Nevertheless, if one intentions are to update an entry, the process consists on getting a certain document from a certain collection from the database, and then proceed to update it with the new value. Take a look at figure 12.

In the prior document, *studies* from the *uab* collection is fetched from the database (57-59). Once the data is retrieved, one can proceed to do further actions (60-84). Nevertheless, if there has been a problem while getting the data,

```

linkRef = admin.firestore().doc('uab/studies');
linkRef
.get()
.then(doc => {
  //comparar
  fetchedObject.forEach(element => {
    if (element in doc.data()){
      //ja esta a la bd, no cal actualitzar studies
      console.log('lelement ' + element + ' SI esta en db a uab/studies' );
    }else{
      //no esta a la bd, cal actualitzar uab/studies amb la nova entrada (un nou study)
      console.log('lelement ' + element + ' NO esta en db a uab/studies' );
      //fer el update
      linkRef
      .update({
        'rootLink' : fetchedObject.links.rootLink
      })
      .then(doc => {
        console.log('rootLink successfully updated!');
        return null
      })
      .catch((err)=>{
        console.log('error updating linkRef '+ err)
      });
    }
  });
  return null
})
.catch((err)=>{
  console.log('error getting uab/studies '+ err)
});

```

Fig. 12: Update operation in the scraper function

the catch block will print an error message (85-87). Notice how the data is manipulated in the *then* block. The code is iterating over the elements of a scraped JSON (*fetchedObject*) and compares every element with the entries fetched before (57-59). If the data is the same, there is no need to update. On the other hand, if the data has changed or there is a new entry element, for example, the scraper retrieved a new computer engineering professor from the university web, it will update the respective entry in the database (71-73).

Now that it is clear how the backend works with the Firestore database, let's proceed to understand how the data is synchronized with the frontend Dart counterpart, displaying it to the final application user in real-time regarding the changes in the database provided by the scraper triggers. As an example, the following figure is extracted from the frontend Dart development.

```

import 'package:cloud_firestore/cloud_firestore.dart';
class Studies extends StatelessWidget {
  String id = "studies";
  @override
  Widget build(BuildContext context) {
    print("estas al metode build de studies");

    Future<List<String>> getStudies() async {
      DocumentSnapshot querySnapshot = await Firestore.instance
        .collection('uab')
        .document(id)
        .get();
      if (querySnapshot.exists) {
        // Create a new List<String> from List<dynamic>
        return List<String>.from(querySnapshot.data);
      }else{
        print('error! no such data');
      }
    } //end getStudies()
    return Layout(id);
  } //end widget build
} //end class Studies

```

Fig. 13: Get operation in Dart

As a previous consideration, notice that there is no need to effectuate any kind of authentication since the database rules clearly specify that anyone can read its elements. Additional explanations about these implementations can be found in the next section.

With this in mind, pay attention how the code gets the document *studies* from the *uab* collection. Moreover, it implements an asynchronous operation with the keyword *Future* that produces a list of strings as a result. It is important to highlight the use of the *async* and *await* keywords, used to have major flow control, as explained previously in Section 7.2. Since Firestore APIs <sup>7</sup> are asynchronous, the code written in the frontend to get the data should be implemented considering parallel executions, otherwise the application will obtain unexpected results, such as empty responses from the database due to code execution ending even if database operations are still being processed. In fact, in this hypothetical situation, it is suggested to use the *then* and *catch* blocks, since the code would execute the second block with a proper informational log informing why the database response could not be processed).

### 7.4.1.3 Security rules

The purpose of this subsection is to provide the reader a brief description of how the project database is managed in terms of access control and privileges associated with the two different agents participating in the project structure: the admin and the application users. Moreover, it covers how the injection of secrets into the serverless cloud structure has been managed with environment variables for the admin to have the authorization to perform the *create* operation.

In this project, the rules are very simple: give read permission to anyone to read anything (the application users) in the database, but give write permission just to someone that has authorization (the admin). The following figure shows how this rules are specified in the Firestore database.

```

1 service cloud.firestore {
2   match /databases/{database}/documents {
3     match /{document=**} { //rules for any document in the entire database
4       allow read: if true;
5       allow write: if request.auth != null;
6     }
7   }
8 }

```

Fig. 14: Firestore database security rules

Notice here that it would have been impossible to write the data fetched from the web scraper into the database without specifying any authentication (line 5 instruction clearly specifies that write permission is only allowed to someone who can authenticate as admin). The scraper is the only element which creates entries to the database, hence it has to have access into the database as admin. The following figure shows exactly how that function does it:

```

var admin = require('firebase-admin');
const serviceAccount = require('./uapp-fc0b9-firebase-adminsdk-rlcv1-f4484b154d.json');

```

Fig. 15: Firestore authentication in Nodejs

The required *.json* file is a document where the encrypted login credentials are stored <sup>8</sup>. This refers to secrets management, a technique to use credentials without directly showing them in clear text.

<sup>7</sup>Tool that allows applications to communicate with one another

<sup>8</sup>The encryption algorithm used is AES128

If anyone could have admin privileges to the database, there would have been extremely dangerous security flaws, given that anyone could delete entries, create new registers, and so on. In Firestore this is not the case, so authentication can be done by simply requiring a local file with all the authentication fields. For obvious security reasons, the file content will not be exposed here, since it contains sensitive information concerning the project database, even though the contents are encrypted.

### 7.4.2 Cloud scheduler

This section explains the software tool used to execute the scraper code periodically. The code runs in response to triggered events such as HTTPS requests, without the need to manage and scale servers. While developing the function, it can be tested locally with the terminal instruction *"firebase serve"*. As soon as the function is ready to be deployed, one can do so by typing the instruction *"firebase deploy"* in the terminal.

The scraper function must detect and push changes to the database. Hence, there is the necessity of executing this function over time. This is when the cloud Scheduler plays its part. This tool is mainly a cron job scheduler, connected with the cloud functions service, that allows to schedule automate tasks given a certain frequency. In this project, this engine is used to periodically trigger the cloud function by making HTTP requests, resulting in the scraper execution.

Since the university web service does not change so often, the scraper will rarely have to update data. Nevertheless, when this occurs, users must not see old irrelevant information. Thus, the backend structure must be implemented in a way that the application users can observe the same content displayed on the web. For this reason, the scheduler has been given a frequency of 5 hours. Thus, the application users will rapidly see the changes.

Function	Trigger	Region	Runtime	Memory	Timeout
scrapUAB	Request https://us-central1-uapp-fc089.cloudfunctions.net/scrapUAB	us-central1	Node.js 8	256 MB	60s

Fig. 16: Scraper function registered in the cloud

Name	State	Description	Frequency	Target	Last run	Result	Logs
scrapUAB	Enabled	execute scrapUAB function periodically in order to fetch data from the web	0 * * * * *	URL: https://us-central1-uapp-fc089.cloudfunctions.net/scrapUAB	23 May 2019, 00:00:00	Success	View Runnow

Fig. 17: Cloud scheduler

### 7.4.3 Application frontend

This project main goal is not to focus on the frontend part, otherwise, it is to implement a solution for the design, which mainly is concerned about how the backend operates. Therefore, the application is remarkably simple, since it principally performs queries to the database to show its results to the end users.

The responsible language to develop the frontend application has been Dart together with Flutter <sup>9</sup>, an open

<sup>9</sup><https://flutter.dev/>

source mobile application development framework created by Google. The main difference between other languages is that it can be used to develop applications for both Android and iOS natively [11]. This is the turning point for choosing this language above any other.

## 8 TESTING

This section covers how the testing process has been implemented. In this cycle of the waterfall methodology, the main task is to ensure that the model developed acts as the user expects it to do, investigating the application functionalities to ensure the quality of the software product under the test.

Since this project focuses on the backend implemented rather than its frontend counterpart, tests have been solely set to that section. Furthermore, even though the numerous existing types of tests, each one with its purpose, manual tests have been set in practice in this project to help achieve further development.

Since the scraper function is executed in the cloud, this platform runs tests over it when executed, and log files are given with error messages or, otherwise, messages notifying of the correct execution. Additionally, try and catch blocks are implemented in the scraper code, therefore, a proper error message is displayed if the execution fails.

Finally, future lines of work as exposed in Section 12 should be performed, considering that this type of project demands to deepen on testing in an attempt to improve the project quality. Nevertheless, no problems have been detected with manual testing.

Time	Level	Function	Event message
16 Jun 2019			
7:00:03.031 pm	INFO	scrapUAB	inside first request, scraping https://www.uab.cat
7:00:06.347 pm	INFO	scrapUAB	inside second request, scraping https://www.uab.cat/web/estudiar/grau/oferta-de-graus/tots-els-graus-13...
7:00:06.454 pm	INFO	scrapUAB	Function execution took 3648 ms, finished with status code: 200
8:00:02.952 pm	INFO	scrapUAB	Function execution started
8:00:04.149 pm	INFO	scrapUAB	Function execution started
8:00:06.833 pm	INFO	scrapUAB	inside first request, scraping https://www.uab.cat
8:00:07.446 pm	INFO	scrapUAB	inside first request, scraping https://www.uab.cat
8:00:08.522 pm	INFO	scrapUAB	inside second request, scraping https://www.uab.cat/web/estudiar/grau/oferta-de-graus/tots-els-graus-13...
8:00:08.700 pm	INFO	scrapUAB	Function execution took 4748 ms, finished with status code: 200

Fig. 18: Logs when running the scraper function

## 9 PLANNING

A few changes in the organization tasks have been incorporated from the initial planning to guarantee the correct project flow and development.

Firstly, the research phase has been updated with an end date corresponding to the 3/10/19, since this is the delivery deadline of the first document. As a result of this change, all the other tasks start dates have been modified accordingly.

Another significant change that has been observed is the duration time of various tasks, to match the project development methodology alongside the document deliveries. Hence, the design phase has significant changes from the

initial planning. Since the requirement collection has been reduced one week due to the fast experience of the student interviews, more time has been provided on the Paper prototype and data modeling activities. Therefore, a well-defined design is guaranteed.

Finally, the remaining time is divided between Implementation, tests, and release. Since the waterfall methodology is used on this project, the implementation and tests phases have been done simultaneously.

**Research : 11/02/19 - 10/03/19**

**Requirements collection : 11/03/19 - 17/03/19**

**Design: 18/03/19 - 14/04/19**

**Implementation : 15/04/19 - 09/06/19**

**Test : 10/06/19 - 21/06/19**

## 10 RESULTS

The project structure proposed in the design phase has been solved. Both the frontend and the backend are operating correctly according to it, satisfying the discussed requirements and critical features.

As a key performance indicator, various executions through various months at different moments of the day (morning, noon, and night) have been performed, and the average resulting value is approximately 270.503 milliseconds ( $\approx 5$  minutes). This is the time that takes the scraper to fetch all the information of interest of the university web service, parse it, and store it to the database. Considering that one of the requirements of this project is to avoid different executions to collapse, this is a valid result, since the execution time is less than the time between executions.

Moreover, changes in the HTML code of the university web service have been observed during the project development, and the scraper has continued to work as expected.

One important issue is the delay time between requests. For instance, if the scraper is executed making a considering number of requests per second, the website administrator would know that is not a human activity, but a scraping tool or even it can be considered a “Denial of Service attack”, and could block the upcoming requests coming from that IP. In this project, there has not been any moment where the university web administrator has blocked the IP where the scraper cloud function is executed, but it could happen in the future. In order to propose a solution to this problem, the scraper function can be modified in order to ensure programmatic delays between requests, so that it will work without getting blocked. In addition, the problem also can be solved by using a pool of IP addresses and use them randomly, therefore it will give the impression that the requests petitions are being generated from different servers.

In addition, there is a major difficulty that the project would have trouble to resolve, the “Captchas”. If the university web administrator introduces Captchas, the scraper cloud function would not be intelligent enough to get past this barrier. Nevertheless, with further research and code implementation with existing libraries such as “cloudscraper” and other techniques, this difficulty can be overcome.

On top of that, the legality aspects of scraping data from websites using software have to be considered. This question regards on how the data extracted is going to be used.

As the data retrieved In this project is being used to re publish it to a new format (the Dart application), and because the data displayed in the university website is for public consumption, there are no legal difficulties [12].

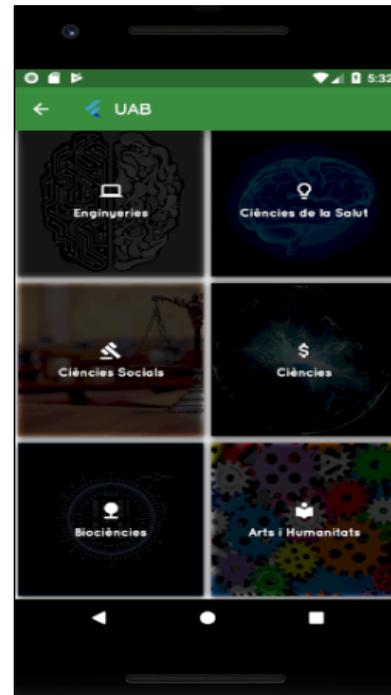


Fig. 19: Application frontend studies screen



Fig. 20: Application frontend studies screen

## 11 CONCLUSIONS

This document has properly explained a content aggregator project from scratch. First of all, previous research has been done to gain knowledge about the problem and the environment. Moreover, various techniques have been carried out

in the design phase to propose a solution plan for information overload and location issues. Furthermore, the design has been implemented and tested using a serverless architecture in the backend and using the Dart language in the frontend. Lastly, some results of the application are shown and discussed.

Some important facts learned in this project are that, with a growing awareness regarding web scraping, sites have become more aware of these activities. As a result, these sites have begun to take actions in order to prevent web scraping attempts and a number of key issues arise for the web scraper developers. Fortunately, the university web service being scraped does not offer any kind of web scraping prevention, therefore the discussed difficulties in the result section are hypothetical situations which could occur in the future if the university web administrators considered it pertinent to introduce. Nevertheless, for every possible scenario, a solution is proposed in order to keep the scraper cloud function operative.

Finally, the author would like to highlight that the university web service is simple, therefore the scraper implementation is also simple in terms of functionality and adaptation to modern pages having brand new techniques, such as IP blocking and Captchas. As seen in this section, if the web service improves, also the web scraper should improve. However, this is not likely to happen, as the university web is mainly a place where information about the university is shown, without the data manipulation being really complicated. Furthermore, while developing the scraper in the implementation stage, a lot of requests in a short period of time has been done in the university web, without it expressing any refusal to the web crawler, so it is not likely to happen that IP rotation is going to be needed, as IP banning is not likely to happen.

## 12 FUTURE WORK

One of the most striking issues here refers to the frequent structural changes of a website. These, on their mission to improve user experience and add new features, undergo a lot of structural modifications such as class names or id changes in the HTML code. Since the web crawler is implemented with respect to the code elements present on the HTML web page at the time of crawler setup, these structural changes would make the scraper less functional, with this returning empty responses as a result of not finding the information required. This is something that should be investigated in future work. Notice, however, that if new elements are introduced to the web, such as new subjects or professors, the web scraper function is capable of retrieving that new information and update it to the database, so structural changes do specifically mean changes in the HTML tag properties such as classes and ids.

Another improvement to contemplate is to add a login schema to the actual structure. With this, the application users would benefit from personalized information, such as notifications when something new happens in the virtual campus or when a book is available again, user address book to store professors addresses, and much more.

Lastly, it is imperative to implement unit testing frameworks such as Jasmine. As seen in Section 8, manual testing for ensuring data availability in the frontend application is

achieved, although it is crucial to reinforce these tests to improve the application quality.

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to several people. Firstly my family, who is continuously looking after me and constantly pushing me to become a better version of myself. Secondly, my sentimental partner, who helped me being the student interviewed and for the love and support that gives me everyday. Finally, I would like to thank Prof. Victor Garcia for his advice and support throughout the project work.

## REFERENCES

- [1] Das, A.S., Datar, M., Garg, A. and Rajaram. "Google news personalization: scalable online collaborative filtering," in Proceedings of the 16th international conference on World Wide Web, 2007, May (pp. 271-280).
- [2] Where readers become leaders. [online]. Available at:<https://feedly.com/>. [Access: 15 Feb. 2019].
- [3] Joan Calzada and Ricard Gil, "What Do News Aggregators Do? Evidence from Google News in Spain and Germany," Universitat de Barcelona and Hohns Hopkins Carey Business School, June 20, 2017.
- [4] Royce, W., "Managing the Development of Large Software Systems," IEEE WESCON, August 1970, pp. 1-9.
- [5] VIJAYAN, Jaya; RAJU, G. "Requirements elicitation using paper prototype," in: International Conference on Advanced Software Engineering and Its Applications. Springer, Berlin, Heidelberg, 2010. p. 30-37.
- [6] Stonebraker, M. "SQL databases v. NoSQL databases". Communications of the ACM, 2010. pp. 10-11.
- [7] Moniruzzaman, A.B.M., Hossain, S. A.. "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," 2013. arXiv preprint arXiv:1307.0191.
- [8] Slides and notes from Software Design subject. 2018-2019 course
- [9] Apple Developer Website. [online]. Available at: <https://developer.apple.com/design/human-interface-guidelines>. [Access:16 April 2019]
- [10] Android Developer Website. [online]. Available at: <https://developer.android.com/design>. [Access:6 April 2019]
- [11] G. Bracha. "The Dart Programming Language," Addison-Wesley, 2015.
- [12] O'Reilly, S. "Nominative fair use and Internet aggregators: Copyright and trademark challenges posed by bots, web crawlers and screen-scraping technologies," 2006. p.273.

## **APPENDIX**

### **A.1 Student interview**

- Question 1. How often do you search information about the university on the internet?
- Question 2. What do you often search on the web about the university?
- Question 3. Do you find it difficult to find specific information about the university?
- Question 4. Has ever happened to you that you didn't find the information that you were searching about?
- Question 5. Would you find it helpful to have an application containing all the university information that you search the most?
- Question 6. How would you like that application to be? How do you imagine it?
- Question 7. What kind of information would you like to check on the application?