FINAL DEGREE WORK

COMPUTER SCIENCE DEGREE

SMS

A proccesses acelerator for educational centers.

Author
Juan Antonio Fernández Sánchez

Mentor
Juan Julián Merelo Guervós

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Junio de 2017

SMS: A processes acelerator for educational centers

Juan Antonio Fernández Sánchez

Keywords: microservices, cloud, education.

Abstract

This work is about the building of a simple open source students management web-based app but with technologies and a architecture opposite to the common. This work is about the design a microservices based architecture that will be able to grow, easily scalable, technology agnostic and with the enough decoupled to be functionall separatelly. But above all, is an excuse to learn, research, about programming, workteam manage, project structuration and planification.

Me, **Juan Antonio Fernández Sánchez**, Computer Science Degree student at **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, ETSIIT, at **University of Granada** with DNI 45601217Z, allow the emplacement of a copy of this work in the library of the center to be able to consult by anyone that desire.

Fdo: Juan Antonio Fernández Sánchez

Granada, 21 June 2017

D. **Juan Julián Merelo Guervós**, University professor of department of Architecture and technology of computers (ATC) of University of Granada

Expose:

That present work, titled ***SMS: A processes acelerator for educational centers*** has been realized under his supervision by **Juan Antonio Fernández Sánchez**, and I authorize their defense front of the correspondent court.

And for the record, he dispatch and sign this report at Granada, 21 June 2017.

The mentor:

Juan Julián Merelo Guervós

Acknowledgements

This would not have been possible without all the support, patient and encourage of my family. Thank you so much for all that you have given me and that I will be never able to give back.

Thanks to Juan Jose Ramos, mentor at *Telefonica Talentum Program*, for all their tips and all help with business and technical approach, and of course to all fellows of the program, now friends also, with which I hope to match in future projects.

My sincere thanks of course to JJ Merelo, for their help, inspiration, for the opportunity to meet cool people and overall fot their patient with a student like me.

And especially to *Susana*. Thank you for being always there.

Contents

Chapter 1

Introduction

Welcome to the next revolution of digitalization of education.

Is time to focus on the students, we need to avoid that teachers spend more time managing student data, as a century ago. We need to give back their focus in the most important, the teaching, *leaving the rest to machines.*

1.1 What about it?

The idea behind this software development is not only to do a useful tool for educational centers, is also an excuse to research and learn a lot about a lot of technologies and patterns, but above all it is about how to do clean, simple and easily readable software.

This software has been thought to help a lot of teachers in their work, but which is exactly the problem? And how it will help?

Until a few years ago the digitalization of education was something impossible to think, the cost of equipment and the digital illiteracy did impossible to think in to do things of another way. The only computers that we could see were inside the *"computer room"*, where students learned to use a simple text processor, received simple notions of the Internet of websites, mail system or storage devices are not bad to start, but We don't talk about this. And what about teachers? Maybe a few of computers in some rooms or office allow them to send emails, maybe build a simple website or offers some resource to the students (in the better case). As we can imagine, the paper was the primary support of all about management of the center.

If we think about processes in a scholar center, we can't imagine the amount of paper that is necessary to registry all about students, teachers, subjects, relationships between them and the rest of work. And the worst of this is

that every three months and every year the most of this documents, papers, records, etc. will need to redo, because the majority of this relations will change and will be required record all again.

can you imagine how many sheets, time and effort is necessary to do this? We will talk more about this after.

Now all people have a smartphone, the hardware is very cheap (laptops, tablets, etc.) and open a website for an inexpert person is very easy, and almost free, in some clicks. There are a lot of system and tools about e-learning like Moodle[1], or Chamilo[2]. These allow us to manage a lot of things like classroom, activities, exams, all related academic workaday. All of us have heard about it, but they have a particular domain, and they are built to a specific way.

This area is more or less known for the majority of new teachers and school stuff. Many centers use some tools like this to do a lot of tasks, but in spite of it, there are another a lot of tools much very heavy that is following handmade.

1.2 Why to build a new tool?

But is really necessary build to zero a new tool to do this? Well, if we want, could reuse some preexisting tool, maybe adding some module to do all things that we detect that fault, but this present a lot of problems, that are closely related to the architecture and licenses of this software.

Because of this, is put a lot of effort in this project in the openness of code and architecture, because this is that make possible that the platform can change, evolve in many forks and feedback herself.

1.3 Domain of the problem

The idea to build SMS arises from the need of a management system for a specific educational center in Granada that mainly speed up all their internal management, avoid the spend of printed paper and save huge amounts of time.

[1] Moodle is a free and open-source software learning management system written in PHP and distributed under the GNU General Public License, know more at moodle.org .

[2] Chamilo is a free software under GNU/GPL licensing e-learning and content management system backed up by the Chamilo Association, know more at chamilo.org .

Besides this, they want a system that was helpful for making better management decisions, in marking paths for future improvements. To achieve that the system would need as core a subsystem of relationships management, teachers that impart class to students, students that are enrollments in subjects and groups or courses, and a long etcetera. This only as the base of the system, because with this would only be modeled the kind of relations that the center has.

A part of this and like the minimal requirements the center need that the system provides a simple way to do their most heavy and paper, time and money cost, the attendance control (we can start to see some requirements that the system would have). Apart of this but related they need a system to save marks related with students and disciplinary notes. Only this three things are done digitally may be a small internal revolution.

1.4 The cost of non-digitalization

Many times we do not perceive the cost of the processes in which we are involved because we do not pay directly this cost or because we are so used to it that is not evident.

1.4.1 Paper

How many papers are spent by a regular teacher in a high?

If we think that an ordinary teacher have eight groups of students (if it works in a full-time), this means twenty-five hours approximately at week, and each group have between twenty-five and thirty students of average. In all, there are two hundred students which a teacher have relation approximately every year.

Each educational center has an own control system but normally and like in this case, there are a lot of standard documents that each teacher manages to follow the evolution of their students (if it does their work fine

We are going to make the calculus of these through the summary of the mainly processes that they do.

Marks and annotations

To begin each teacher need to follow the evolution of each student in their subject quarterly, to do this they use an official document where they write all interesting annotations about the student, from marks of partials or final exams to annotation about homework or class issues.

If the common teacher (here "teacher") have two hundred students and in the Spanish system we have three-quarter this make up a total of six hundred sheets of paper, more another fifty by differents uses that a teacher spends commonly. Usually, each center give to teacher a book with all these pages officially formatted to this task, so the cost is upper that only the paper and ink. If we multiply this number by forty teachers approximately, we have two thousand four hundred.

Tutorizing

Each time that a parent want to check the evolution of his son the mentor of the student need to do a simple but time and money expensive process.

So, if in a medium center we have fifteen instructors (mentors) and each one mentoring about thirty students, and technically each of this students needs a meet with their parents each quarter, this mean 15 * 30 * 3 * 10 subjects that the student have and which need an inform, is equal to nine thousand pages. Including some pages necessary for the pedagogic control (parallel system of supervision in the center).

Delays

Delays record is another process which spends paper. It is a very inefficient whole of steps. To understand this better we are giving you an example, when a student is late (whatever the reason) a teacher need write in an officially formatted page when it was late and why (according to it).

After the student will move to another place of the center to give this paper to an admin staff person that will notify this to their parents (because the delay is not justified or they did not know). This process means another one hundred twenty-five sheets per week between all students on average.

Class attendances

All weeks all teachers that impart class to a group write in a paper all attendance controls of a class (independently of their records). This is done by all teachers. So, in on average a center have around one thousand of students and around twenty-five students by course, more or less forty courses and is necessary an attendance records paper by class, so is equal to forty documents per week and if the course has thirty-two weeks, the result is approximately one thousand two hundred eighty sheets of paper.

Another center communications

In addition to the above, there is another kind of process in which big amounts of paper is spent. When the direction of center wants to communicate something, in some special days when is necessary to do this or in some activities that simply require paper. In the most of the cases, the excessive cost is only because there are not another better way to do this, only for this reason. Sometimes some so easy as email or website or simply share document would save a lot of money but the culture and the dynamic of many places do difficult put into practice.

Summarize

With the follow constraint:

- A scholar course has 32 weeks at a year on average (32-36 with 160-180 days).
- The cost by printed page is about 0.03 cents of euros.
- We are going to consider that a common center has 1000 students and 40 teachers of average.
- More or less 25 group of students.
- Each center has 15 mentors.

So, with this and preceding conclusion, we can obtain this costs table:

Process	Per week	Per year	Cost
Marks and annotations	–	26000	780 €
Tutoring	–	9000	270 €
Delays	125	4000	120 €
Class attendances	40	1280	38,4 €
Others	–	1000	30 €
		Total	**1238,4 €**

Table 1.1: Paper consumption cost estimation per year.

As we can see, the cost is too much, independently of the environmental reasons. is not our goal to give an exhaustive explanation of the problem, but all understand what is. The are a lot of projects and papers that deal this issue, as for example V. Gallardo article, about a Spanish high.

1.4.2 Time and Money

Is not only the cost of the material resources but is also the time and their money equivalence what we are trying to save. If we count the hours of work that a common teacher dedicate to do this we can estimate the cost that all student management related work required.

Stuff	Amount	Hours/Month	At year	Price per hour	Total
Teacher	40	5	1800 h	15 €	27000 €
Admin	3	10	270 h	15 €	4050 €
Direction	4	10	360 h	20 €	7200 €
				Total	**38250** €

Table 1.2: Cost based on staff work hours.

This would be the cost if we think in the hours that all school staff spends in this kind of tasks and that have not been spent in projects of the own center.Can you imagine many projects that would be doing with this huge amount of hours of work by the staff of the center?

The problem is always the same, organization, control, and focused value. And the common pattern does not have control over all and as thinks like this, companies lost a lot of money and resources. The control of that happens in your business is the most important and obviously is the only way to can take decisions data based.

If we add the costs of the materials as paper joined to the teacher's work hours (strictly paid), we are talking about the incredible amount of approximately **39000 €** per year. A ridiculous number thinking in the cost of software implantation.

1.4.3 Value of business

Another thing that does go unnoticed is about the values of the data that big relational centers as school manage without any insight of their value. In a standard center with one thousand students is generated around ten thousand records per month, with an essential record schemas.

It is around 90.000 records per year and if we have a more complex system with a lot of data recorded the amount can be upper. So, what happens with all of this data. There is any value? Obviously yes, the exploitation of this data is not very clear at first because the most of people (centers directives included) not seen the center as a business, which actually is, with the difference that our product is the quality of the teaching that we are offering to our students.

So, as in any manufactured process, there are a lot of steps that can be improved, only if are controlled and measured. With students is the same (without any wrong charm), there are a lot of steps, processes, and stages that could be improved to offer a better education. If we detect that by whatever reason a student have a bad trend in their evolution is very difficult to detect if we don't have a strong system to recollect data and be able to do a fast actuation (proactive in opposite of late traditional reactive behavior) will be very difficult fix the problem.

And as there are a huge amount of data is difficult that anyone can process this manually, and for this reason, is not easy to detect what part of our processes fail or need to be improved. All this problem can be easily solved with a powerful system that can be recollected, processes and give human-readable way all the conclusion obtained.

And this the other big mission of the kind of software like this, offer business value of the data that help to manage.

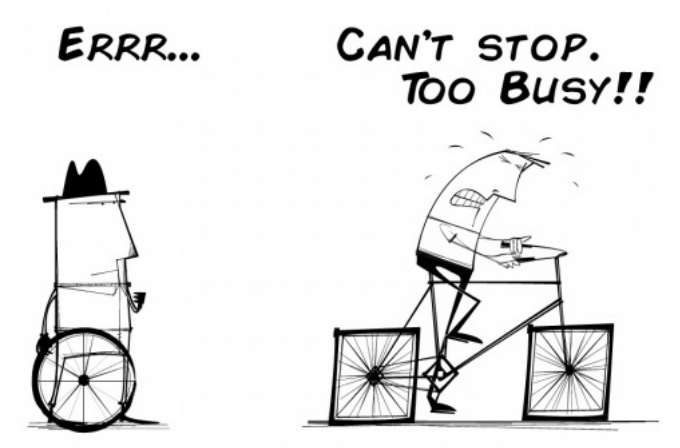


Figure 1.1: Too busy to improve. Author: Alan O'Rourke

Chapter 2

State of Art and first decisions

In this chapter, we are going to talk about the state of art in two domains. On one side we have the analysis of the similar projects or applications that already exists in the market, which would be the state of art of the problem that we want to solve and by other side, the technologies that we can choose to solve this, it would be the state of art to the implementation.

2.1 Problem domain

The developing of a new complete system have no sense if before have not been analyzed other existent solutions. As part of the research process has been analyzed some applications that (although are not a direct competence) represent the actual state of the art about the kind of software we are talking about.

First, we take a look at the main features that are important in our domain to see how many applications have covered them. These are a few requirements analysis but not very formal, we will discuss about this after.

In the following table, we are going to compare the rest of app with our idyllic software (with the features we hope it will cover).

Feature	classDojo	TeacherKit	A+	iDoceo	Additio	SMS
Attendance	✓	✓	✓	✓	✓	✓
Discipline	|	✓	✓	✓	✓	✓
Attitude	|	✓	✓	✓	✓	✓
Marks	✗	✓	✓	✓	✓	✓
Competencias	✗	✗	✗	✓	✓	✓
Rubrics	✗	✗	✗	✓	✓	✓
Message	✓	✓	✓	✗	✗	✓
patters	✗	✗	✗	✓	✓	✓
reports	✗	✓	✓	|	✓	✓
analysis	|	✓	|	|	|	✓
models and predictions	✗	✗	✗	✗	✗	✓
Centralizate and jerarqy	✗	|	✗	✗	✗	✓
Total	58.3%	80.5%	72.2%	72.2%	80.5%	99%

Table 2.1: Features

As we can see, there are some apps that are so focused on their specific domain that is very deficient in the rest. In spite of this are referents in their area, and their numbers of users are huge (only have been compared apps that have a good acceptance by users).

But is not only important the features of the software, because with time and effort any feature can be added. There are other things in the background that are really important too and that can be that to grow our app steadily or that crash when we have thousand of users. We are talking about something as data control, performance, scalability or centralización (we do not explain in detail each one because is out of his work but we think is also very important). And of course, all related to interface and user experience, that in most of cases can represent the difference between the successful and failure of our project.

So, based on it this is our comparative:

Feature	classDojo	TeacherKit	A+	iDoceo	Additio	SMS
Personalization	✗	✗	✗	✗	✗	✗
UI	|	✓	✓	✓	✓	✓
UX	|	✓	✓	✓	✓	✓
Price	✗	✓	✓	✓	✓	✓
Performance	✗	✗	✗	✓	✓	✓
Scalability	✗	✗	✗	✓	✓	✓
Data control	✓	✓	✓	✗	|	✓
Multiplatform	✗	✗	✗	✓	✓	✓
Centralization	✗	✓	✓	|	✓	✓
Total	44.4%	50.0%	16.6%	16.6%	44.4%	99%

Table 2.2: Architecture and design

Obviously, only in the hypothetical case of our app is fully developed will cover all the aspects considered, but before of the research is easy to understand that is not so difficult cover all with the correct design, architecture or design.

And we are going to try to give a simple and really little approximation of the way to achieve that.

So, **let's start!**

2.2 Implementation domain

Nowadays we can build almost any kind of software with a lot of differents language from which to choose, but as if that were not enough there are also a lot of architectures that are possible to follow, and beside of this it will be necessary choose where to deploy our software, or how doc it, or how to manage our work team, etc. A lot of options that it will be necessary to choose and that can make the difference between the success or failure of our project. So, what about of these decisions in this project? We are going to take a look at it, tools, patterns and *we will take some firsts decisions about it.*

2.2.1 Architecture

Almost any kind software or tool can be built of differents ways, independently of this behavior or the goals that it must achieve. Some ways are

more specific to achieve some specific behavior and other although useful are not the best option. Because of this, we need this phase of research, to avoid, as far as possible, make mistakes before to start.

Kind of software

We can think of a mobile app, but what happens with desktop users? And even if we do a mobile app, for what operating system? Too options for to develop a solution for each one. Based on informal requirements was thought that the best option could be a **web app** for a lot of reasons. First of all because is the simplest way to offer an app that can run on almost any device (with a *simple*[1] browser). Also because is the easiest way, with the best learning curve and to end because can be used a lot of different technologies to the same domain.

Why microservices?

When we think about an app, of any type, the most of the time we think of a software running on a single machine, with more or less hardware available, where all related to the software are inside of the same machine.

Now, we hearing all the time about microservices, which are in synthesis the opposite of the traditional monolithic classical approach in software architecture, and seem like if your design is not based in this mean you are outdated or your design is directly wrong. Well, this is not totally true but neither false. The goal of this section is not describe the advantages and drawbacks of this approach, but yes justify why is their choice.

Microservices are actually a variant of SOA[2],that simplify a lot the architecture of a software system spliting their different systems, doing that work together to offer the service which the software was design. Dr. Peter Rodgers introduced the term *"Micro-Web-Services"* during a presentation at textitCloud Computing Expo in 2005 and as we can see in the following graphic (extracted using *sotagtrends.com*), the interest of the community about this emerging technology has never stopped growing.

[1] Actually a browser is one of the more complex pieces of software, but here is labeled as simple because is a software that the majority of devices like smartphones, tablet, etc, have by default and for the most nontechnical people isn't a complex software, nothing could be further from the truth.

[2] Service-oriented architecture (SOA) was first described by Gartner in 1996 and is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.

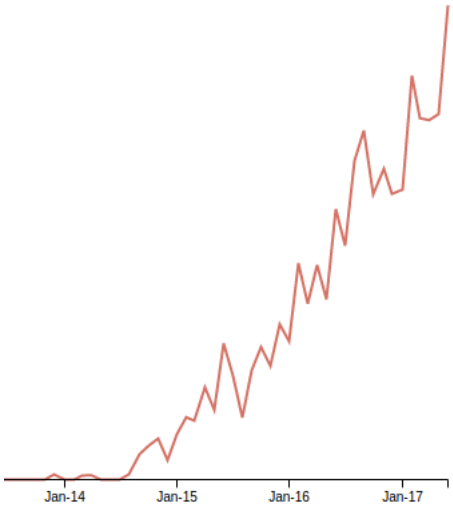


Figure 2.1: Search trend of "microservice" tag in Stack Overflow.

James Lewis[3] and Martin Fowler[4] are the major defensor of this arquitecture and these brings us to the perfect definition of microservices:

> *"In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralised management of these services, which may be written in different programming languages and use different data storage technologies."*
>
> (Lewis/Fowler)

To know more please see Martin Fowler [2014].

Language agnostics, scalables, adjustables size, independent. our system design splited in litle pieces with this boundaries make this architecture perfect for our goals, and this is why we take this choice.

[3]Principal Consultant at ThoughtWorks and member of the Technology Advisory Board.
[4]Software developer, author and international public speaker on software development, specializing in object-oriented analysis and design, UML, patterns, and agile software development methodologies, including extreme programming.

Why polyglot database?

With mircroservices-based approach the system will need differents databases to do some diferent things. In general we can see our backend like a black box when the data persists in a polyglot database. That means that the data is saved in differents ways, using different formats and diferent drivers to manage this. There are a group of data that would have need be saved with certain relations, due to its nature and a relational database seem perfect to do this, but this kind of databases (like MySQL) can be slowly or too heavy for other tasks of kind of processes, like data analysis.

2.2.2 Storage

With the storage occurs the same, there is a huge list of options to choose. The fast answer in the most of the cases is: Why do not use a relational database, as MySQL for example? If we are thinking in a little system it could be a good choice, always that our data model is adjusted to this kind of system. But every people knows the deficits and the complexity of the development of a system with this database. Mainly the performance when we are talking about million of arrays of objects stored. So, as we was talking before, our goal in this project is work with a lot of data, of which some are pure relational and another not, and can be processed by another way, using another engine and database.

If you have a logical data model like as this project have is not easy choice one of all database engine to model this. SQL systems is very powerful for some things but not for another (or not simply) and object oriented databases is really powerful and simple to develop with it when the data haven't a lot of relations (although can be modeled also). so, why choice one between them? Why do not both? This is the approach selected, build a system with a Polyglot Database, that means: much better select the better engine for any kind of data instead of trying to use the same for all.

And this approach joined with microservices architecture give us the first Conceptual design of our system, where each microservice have their domain work and their own data, using for their own database engine, and Until their own language is it will be required.

So, focused on this project, we are going to use mainly two systems, SQL relational database system in a service (we will talk more about it after), and a NoSQL service, in this case, Google Datastore, a fast and lightweight engine to mange an object oriented database.

At the moment to write this chapter had been evaluated MongoDB as the best choice, but the facilities of the platform selected (detailed after) was made that the G. Datastore was selected finally. Beside of this if the project are rebuild now have not doubt, Mongo will be selected after the experience (the reasons will be explained in more detail in the conclusion of the work).

2.2.3 Code strategy

Entire project will be stored in a single repository, using git as control versions system.

Version Control System

Git[5] is the perfect tool to achieve maintain a more o less easy workflow between developers minimizing the risks, and the huge list of plugins to different software and IDEs do this perfect to work, dismissing another as Subversion, Mercurial, Fossil, or Bazzar.

The reason to use a single repository is that the mechanism to maintain the consistency between projects is not easily enough to do the develop agile. In spite of, git offers us a lot of tools that can improve enormously our workflow, as the Git Submodules[6], or Git Hooks[7] between another.

Hub

For another hand we have to do another choice, select the remote hub of our repositories or Git repository hosting service. The most popular for some years has been GitHub, are a web-based Git or version control repository and Internet hosting service. It offers all of the distributed version control and source code management (SCM) the functionality of Git as well as adding its own features, launched at 2008 but there are other options as Bitbucket[8], SourceForge and especially GitLab[9] (with a big grow ultimately).

[5] Git is a free and open source, distributed version control system created by Linus Torvalds.
[6] Submodules allow you to keep a Git repository as a subdirectory of another Git repository. This lets you clone another repository into your project and keep your commits separate.
[7] Git has a way to fire off custom scripts when certain important actions occur. There are two groups of these hooks: client-side and server-side. Client-side hooks are triggered by operations such as committing and merging, while server-side hooks run on network operations such as receiving pushed commits.
[8] web-based hosting service that is owned by Atlassian, used for source code and development projects that use either Mercurial (since launch) or Git launched at 2008.
[9] launched at 2011, was written by Dmitriy Zaporozhets and Valery Sizov from Ukraine. The code is written in Ruby. Later, some parts have been rewritten in Go.

Develop workflow

Another of the main problem easy to find in the develop when are involved few people is the organization of the contribution of a repo, but if the project is composed of few subprojects, this can be worst. For this reason was choose the strategy to have an only repo with subfolders. So, took this decision remains to be defined how will work the repo in the different phases or the develop.

To do this, we will follow the standard, using branches to develop features and maintain a good state of the versioning of project.

So, as we can see in the picture, we are going to use *develop* branch to do the develop, where we are going to work in minimal changes and will do a fork when we want to do some representative change in the code, as new functionality or some big correction.

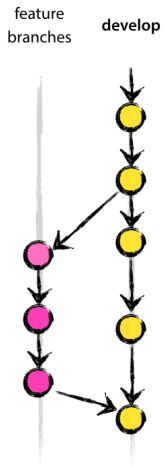


Figure 2.2: Workflow with features branches

For other hand when the all code to launch a final release we will fork the repo in a *release* branch to while the team continue with the develop in the mainly develop branch ot in a issue form of this another part of the team (or itself, is the same), will prepare the code to production, taking a look because there are some bug, fix it and when all are testend an correctly working put the project in exactly this verison in master (doing a merte) and this is when officially a new version of program will be launched.

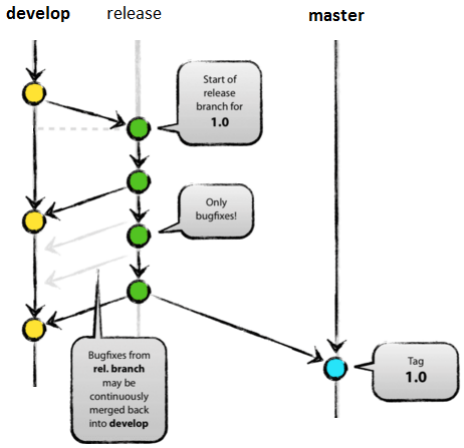


Figure 2.3: Workflow with releases branches.

There are some variations of this standard workflow but for this kind of project is perfect.

2.2.4 Languages and frameworks

As has been said before, almost any kind of software can be built at infinite ways, and this start with what language can do this. This means it could be built with Java, Python, C++, Go, Ruby, JavaScript, PHP, (only talking about backend) and with another list to the frontend, in this deep list of possibilities, we choose the most flexible and powerful of all of them, Python and JavaScript.

Python because is one of the most simplest and powerful languages nowadays, and JavaScript because is all a standard in the industry. Obviously, the choice is based also on the fact of both languages are really supported by the community, have a good learning curve and a lot of projects and systems are based on them.

JavaScript has been selected because Angular is written in it. AngularJS is the most powerful framework nowadays to build fast, clean and powerful web apps.

To exchange data between service we need to select an idiom which the services will talk, which they will exchange information. That's mean mainly select how will transform the objects and data structures that services will manage to be able sender across the net.

We have some data serialization solutions, some very popular and another most specific of very focused domains. So, the most commons are JSON,

YAML, BSON and ultimately MessagePack. Each one has their owns benefits and drawbacks but the selection has been easy, JSON.

XML

Extensible Markup Language, is a markup language, defined v1.0 by W3C[10] This is an example:

```
<exam>
  <result>5.8</result>
  <type>Partial</type>
  <subject>Science</subject>
  <date>17-06-2018</date>
</exam>
```

JSON

JavaScript Object Notation, is an open-standard file format that uses human-readable text to transmit data objects based of attribute–value pairs. Douglas Crockford originally specified the JSON format in the early 2000s two competing standards, RFC[11] 7159 and ECMA[12]-404, defined it in 2013. The ECMA standard describes only the allowed syntax, whereas the RFC covers some security and interoperability considerations.

In spite of exists a restricted version of JSON, known as I-JSON (short for "Internet JSON"), defined in RFC 7493, is not as popular as original. This is an example:

```
{
  "title": "The Picture of Dorian Gray",
  "author": "Oscar Wilde",
  "date": "July 1890"
}
```

YAML

Is other Markup Language, commonly used for configuration files but that can be used in transmision also. Actually is a superset of JSON, whitch latest release 1.2 was published in 2009 and was first proposed by Clark

[10] World Wide Web Consortium, founder by Tim Berners-Lee at 1994 at MIT (Massachusetts Institute of Technology) the consortium is made up of member organizations which maintain full-time staff for the purpose of working together in the development of standards for the World Wide Web.

[11] Request for Comments, is a type of publication from the Internet Engineering Task Force (IETF) and the Internet Society (ISOC), the principal technical development and standards-setting bodies for the Internet. were invented by Steve Crocker in 1969 to help record unofficial notes on the development of ARPANET.

[12] Ecma is a standards organization for information and communication systems founded in 1961 to standardize computer systems in Europe.

Evans in 2001. This is an example:

```
---
martin:
    name: Martin D'vloper
    job: Developer
    skill: Elite
```

BSON

Binary JSON, is a computer data interchange format used mainly as a data storage and network transfer format in the MongoDB database. MongoDB represents JSON documents in binary-encoded format called BSON behind the scenes. BSON extends the JSON model to provide additional data types, ordered fields, and to be efficient for encoding and decoding within different languages.

MessagePack

Byte array is an efficient binary serialization format, like JSON but faster and smaller and this is an example:

```
{"compact": true, "schema": 0}
27Bytes

82 A7 compact C3 A6 schema 00
18bytes
```

This has several properties really interesting to change in the future our architecture if will be precise by performance problems (almost all related to internal communications). See this, del Alba, interesting benchmark about it.

Protocol

Microservices must talk and can do this by differents ways. So, we are going to take a look over more interesting for us.

The first that we are talking about is **REST**. Representational state transfer or RESTful Web services are one way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations. Other forms of Web service exist, which expose their own arbitrary sets of operations such as WSDL and SOAP

On other hand we have **RPC**. In distributed computing, a remote procedure call is when a computer program causes a procedure (subroutine)

to execute in another address space (commonly on çanother computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

And finally **gRPC** . It is an open source remote procedure call (RPC) system initially developed at Google. It uses HTTP/2 for transport and provides features such as authentication, bidirectional streaming and flow control. It generates cross-platform client and server bindings for many languages. We will talk a bit more about this in the Develop chapter.

Testing

The tool that we will use for any testing involved in the backend of the project will be PyTest. Launched at 2004 by Holger Krenel is one of the most complete suites for testing over Python. Is really easy to use and allow do some things that is very difficult with another frameworks as built-in *unittest* python package, somethings as the use of fixtures of or the amount of plugint that it has.

Focused on the problem it will be used to make all unittest of the core of services, libraries and auxiliary programs and testing the entire functionality of the service when it has a role of black box.

Documentation

As is saided the tool Sphinx is used to build the doc of the service. That basically inspect the code files mixing this with all the files that we write (pure doc) to show this as a web based documetnation (easy to read and understand).

2.2.5 Platforms

Today, with the explosion of the cloud is necessary a lot of infrastructures that support All that all companies that want to migrate their businesses model to cloud need. This is possible thanks to the companies that offer solutions to this, and another that after Was introduced to the same businesses offering software and hybrid solutions. Actually, there are a lot of companies that offer some solutions to software (not necessary) Companies to work with the cloud to build cloud-based solutions and cloud computing easily.

There are a lot of companies and solutions but only a few are relevant in this domain. Although the classifications always can change there are three kinds of service where each solution can be classified.

The first is **IaaS** (Infrastructure as a Service), that means the pure infrastructure, only servers, virtual machines in cloud data centers with some management tools in the majority of the cases where all internal management of this machines is work of the customer. Examples of these are Amazon Web Service, Google Computer Engine, Rackspace Cloud or vCloud. The second is

PaaS (Platform as a Service), where above of infrastructure was implemented services that abstract the management of the *"metal"*. Normally services where the developers build the code and run this in some services. An example of more common is Google App Engine or Heroku.

The last one is **SaaS** (Software as a Service), in this case only is offered software solutions in line, which customers do not need worry about updates, installation or somethings like this. An example of this is famous Google Docs, Salesforce, Dropbox or the traditional email managers as Gmail. Almost any online service can be framed here. Let's take a look to more important nowadays.

Amazon Web Services

Launched at 2006 The most popular include Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3).

Microsoft Azure

Is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service and infrastructure as a service and supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems. Azure was announced in 2008 and released in 2010 as Windows Azure, before being renamed to Microsoft Azure in 2014.

IBM BlueMix

A PaaS developed by *IBM*. It supports a lot of languages and services. It is based on *Cloud Foundry* open technology and runs on *SoftLayer*[13] infrastructure.

Heroku

Another PaaS, founded by James Lindenbaum in 2007 at San Francisco, California,is used as a web application deployment model. It was acquired by Salesforce.com in 2010.

CloudFoundry

Is an open source PAAS governed by the Cloud Foundry Foundation that was originally developed by VMware.

Google Cloud Platform

Is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search and YouTube.Alongside a set of management tools, it provides, a series of modular cloud services including computing, data storage, data analytics and machine learning.

For all those services that offered and by some experience with it in some previous little project is because was selected.

2.2.6 Documentation

To do the documentation of the code as in the rest of sections, we have a few options to choose. So, as the backend is fully developed in Python we are going to use some python especially documentation framework and in this case Sphinx[14] has been selected.

For another hand, to the front, especially talking about Javascript with AngularJS we do not have the need to a doc because the docstrings have been sufficient, and neither have been found any really good tool as Sphinx for them.

[13] A dedicated server, managed hosting, and cloud computing provider, founded in 2005 and acquired by IBM in 2013.

[14] Sphinx is a simple and powerful Python-based documentation generator.

2.2.7 License

In this project, we are going to use GNU General Public License v3 (GPL-3) to license our code. There are a lot of licenses available, but it meets all our needs, this is a little summary of this:

Can	Cannot	Must
Commercial use	Sublicense	Include original
Modify	Hold liable	State changes
Distribute		Disclose source
Place warranty		Include License
Use patent claims		Include Copyright
		Include Install Instruction

Table 2.3: License characteristics summary.

Although this is one of the most famous, there are a lot of open source licenses. If we want to know how many projects are licensed with which licenses there are some research about it. As for example, the made by *Black Duck (blackducksoftware.com)* that looking at more of 9000 forges and repositories have calculated these use percentages: **MIT**[15] (32%), GNU 2.0 (18%), **Apache**[16] 2.0 (14%), GNU 3.0 (7%) and **BSD**[17] 2.0 (6%).

[15] A permissive free software license originating at the Massachusetts Institute of Technology (MIT).

[16] A permissive free software license written by the Apache Software Foundation (ASF).

[17] A family of permissive free software licenses by the Berkeley Software Distribution (BSD), a Unix-like operating system.

Chapter 3

Requirements

In this chapter, we are going to describe which are the requirement that our system has. It will be used User Stories[1] methodology which it will be talked a bit more in the next chapter. This tool will be useful to identify the requirements of the system by a human readable way., removing all possible layer between design and traditional requisites specification process.

User stories can be very minimal (unit) and very general. That mean that we can do a user story that is a simple requirement or a very general issue. If the criteria to follow is to do very minimal users stories, we could be hundred of them, so in this case, is better have an upper abstraction level to have fewer user stories but with more content, at least when is the first time that the team uses this methodology.

3.1 Minimal user requirements

The managers of the center can't have a precise idea about which must be the functions of the software. They have a clear idea about what kind of process in their daily work there are the heaviest and consume more paper, that is which they want to digitize, but they have not thought about navbars, icons, colors palettes, user interfaces design or something like this. They have abstract ideas that will need dimensioning and shaping but which in synthesis offer the minimum system requirements.

3.1.1 General needs

these are the general requisites that customers have, independently of the logic of the application. We are talking about software and it will need run in somewhere and must be cheap and easy to run. At first, there are not

[1] A user story is an informal, natural language description of some features of a software system. See bibliography to know more.

any pre-requirements. Maybe they prefer it can run in devices like smartphones, tablets, laptops, but they do not have a good technical knowledge and for they are indifferent that it will be a native app, a web app or some strange artifact.

On the other hand, exists an essential requirement, must be run on any device for the same person, independently of the machine where it be (sometimes in laptop and others in a smartphone, for example).
See user stories related: GN-1 to GN-3 on appendix A.

3.1.2 Academic Management System

As we can imagine, a center have a lot of differents kind of relationships, and as an obligatory part of the system, it must be offered a simple way to manage this. Teachers impart subjects to a group of students (called classes), students are enrolled in subjects, and so on.
See user stories related: AMS-1 to AMS-7 on appendix A.

Attendance Controls

This is one of most important process to digitize, the amount of data that is saved on paper make impossible to manage fluently and much less analyzing them. They need a simple way, a simple interface to do this that allow they to save time (doing it, analysis and management it), paper and effort.
See user stories related: AC-1 on appendix A.

3.1.3 Class Controls

A part of manage of classrooms they must offers a say to follow the students evolution in class, behavior, positives and negatives and another lot of things like paper save, time and effort as section above.
See user stories related: CC-1 on appendix A.

3.1.4 Marks

As another process that more paper consumes is the marks management. They want a system that simplify the process of insertion, analysis, and management. Without any special idea in mind are open to any good user interface that simplifies it.
See user stories related: MRKS-1 on appendix A.

3.1.5 Disciplinary Notes

It also must provide a management of this kind of notes, in which a bad behavior of a student is saved, managed and fully reported to the specific users inside of app, as tutors, pedagogue, etc, for example.
See user stories related: DN-1 to DN-2 on appendix A.

3.1.6 Simple and advance reports

Another part interesting for they is to improve the reports that they obtain from their data. The on paper support do this almost impossible to big scale, an important feature must be do this possible. Advanced reports about a lot of kind of items, like students, the state of subjects, marks, etc, in seconds with some clicks. This will improve the take of decisions and will do better meetings with better decisions based on good and contrastable data.
See user stories related: SAR-1 on appendix A.

3.1.7 Autonomic Official System Connection

The national system of education in which this center is framed have differents informatic regional systems. In this case, in Andalucia, where the center is, it called *SENECA* (other in other places of the country). All public and semipublic educational centers need save data in this systems necessarily, and this haven't a simple public API where connect us. They must be done dirty way, making a mixed mode way to simplify the download and download of data that minify the interaction with the official system.
See user stories related: SNC-1 on appendix A.

3.2 Minimal system requirements

The system, as if it was another user has their own requisites, and can be handled as normal user stories. So, with the architecture that we are talking about all parts of the system (the microservices) must be independent and low coupled also must be able to run in different places using always the same communication protocol (API Rest in this case). These are not a lot of requisites but are indispensable to the system has the properties that we expected.
See user stories related: MSR-1 and MSR-2 on appendix A.

Chapter 4

Planning

4.1 Methodology and planing

In this project, we are going to use an agile methodology. This kind of methodologies looking for alternatives to traditional project management. Agile approaches help teams respond to the unpredictability of changes. Not only the team has been ready to some change, it knows that they will happen.

Exists a lot of variants of the same philosophy, Lean Software, Kanban, Extreme Programing or Scrum, between others. All follow the same goals but they do with differents ways.

We are going to use Scrum (Sutherland [2014]) because some members of the team had some of experience in it and is the most used in the world of the agile approach. Is not the goal of this work explain in detail all characteristics of the system, but we are going to comment the most important to understand the planning.

We are going to split the six months approximately of time in two weeks slices. Each slice will be a sprint , a piece of time where will work a very bounded part of the project, based on very specified issues, with some special characteristics. All these issues are based on some user story (maybe not directly).

In SCRUM, almost everything is related with these sprints. So, we have the most important parts, as the Sprint Planning, Daily Scrum, Backlog and the Sprint Review Meeting. With these elements, we have enough to start to develop. The process to follow is very easy. In this case is a bit contradictory because SCRUM does not understand of deadlines, only of *Minimum Product Value* and here we follow a planning, but will work anyway.

So, to start we need split the high priority user stories in real issues (as a GitHub issues for example), little problems to solve some concrete problem, labeling this with the priority of the user history. After we do this, we need to estimate the complexity of this issues, using some tool as Scrum-poker-cards (see redbooth github repo, *github.com/redbooth/Scrum-poker-cards*) and mobile app. When we have this we need put in the sprint (2 weeks of work) all issues that we think that we can to complete, and this will be our sprint.

There are a lot of details that we are missing out but is not important to explain the essence of the system. So. the issues that for any reason will not put in the next sprint will be put in the backlog (a queue of tasks) where will be recovered from next. The role of the estimation is very important because do that the team know each other and knows how many lines of code can produce the whole team (independently where we are working at this time).

So, to give a general image of all project development we are going to split all issues (coarse-grained) to can to see where would put each issue in general calendar. Thus we can to know if we are at time in our long-term planning (independently or our sprint, the dev team and the sales team work with differents concepts of time) and will be (more or less) easy try to correct or analyze retrospectively.

This having this said this would be our Burndown chart, that beans, the normal evolution of remained work to obtain a release with the conditions and deadline putting ourself.

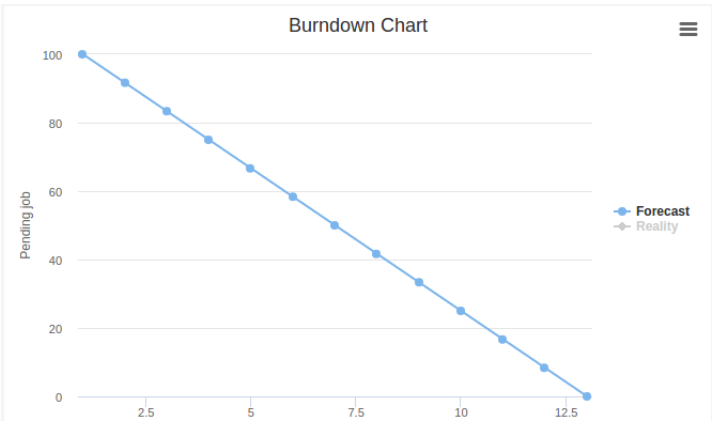


Figure 4.1: Burdown estimation chart.

Now is where play the mechanism Sprint Planning, Daily Scrum and the Sprint Review Meeting. For each sprint of this thirteen, we must do a sprint planning to know what issues are inside and which would put in the backlog and another meeting to review the evolution of sprint.Normally another meeting must be done diary, but in this case with the form of the team is not possible, so instead of is important, is not at the rest.

Chapter 5

Design

Once the requirements have been decided, is time to make the design of the system that will do all needed to cover them. First, we are going to take a look over the general architecture, the distribution of the domain of the services and how to delimit their behavior and communications with each other, by after, take a look to each one to know how to work in each domain and how have been solved all problems that this design present.

5.1 Responsibilities distribution

At the moment that we decided split the functionality of the system between services we found the problem that how to decide which is the domain

At the moment that we decided split the functionality of the system between services, we found the problem that how to choose which is the domain of each one. In the majority of the cases, we are beginning from a monolithic system that we want split, in this instance, is some like this but at first, it was thought to be splitted.

In a common app (whatever kind of it) we have three well-defined layers, user interface, logic and storage, something as common MVC[1] A part of user interface make the necessary to present the info and offer the way to interact with this to the user while the logic layer do this possible doing the logic steps to work with data structures saved in some kind of storage.

[1] Model–view–controller is a software architectural pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user, was introduced by *Trygve Reenskaug* into Smalltalk-76 at Palo Alto Research Center in 1970.

There are some frameworks as Django wherein all this layer can be developed together, is not strange their popularity, but if we use other as AngularJS this only help us in the user interface layer, not in the rest, but for other hands have another advantage.

So, that as it may, this would be the more simple schema to start to work in our domain, with the difference that we work with service instead of layers in a single program, but essentially, the concept is the same.

As we can see in the next picture, we will have a user interface, when will built all views and controllers that work with this to build a complete user interface, a logic section layer, here labeled as APIGmService (as a first approach) that will do the logic of the system, transform the data, restructuring and offering the interaction form UI to database. And at end, the database system, mainly an engine of the database and the drivers necessary to work with them.

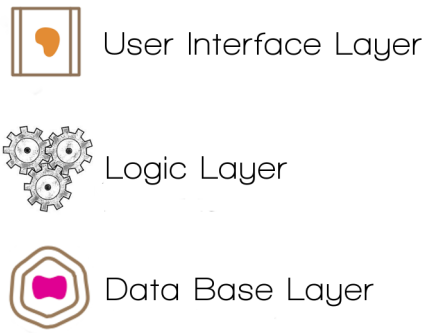


Figure 5.1: Basic layers in common app.

As we can imagine this work fine, is compact and stable when we are talking about the same code on the same machine but now we are going to split that in different services, maybe running at different places and maybe in different languages, so when we talk about divide the domain is about split the logic with the database access layer.

Next picture shows us perfectly what are talking about. This is an example of migration from monolithic to microservices from Nginx resources.

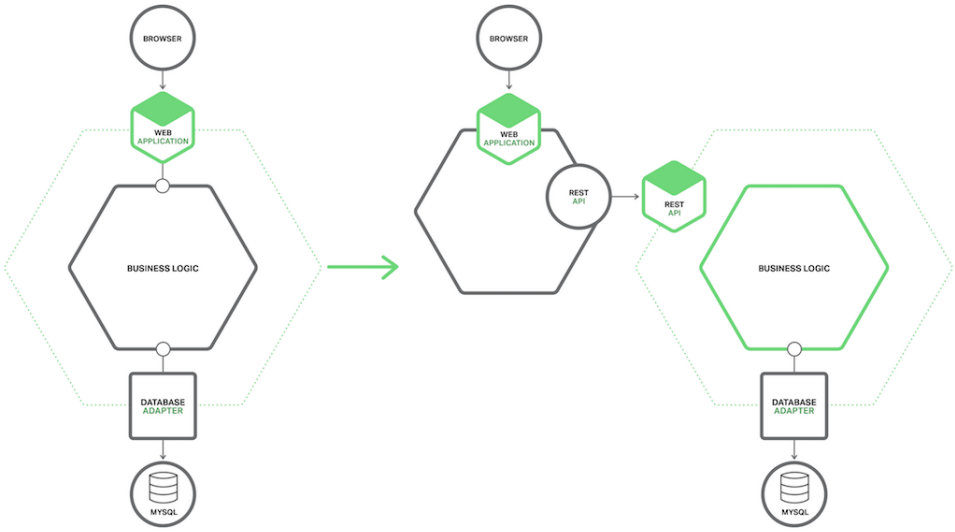


Figure 5.2: Monolithic to microservices, from nginx resources.

So, for us, each service will always be mainly the same components, communication protocol and logic layer while database layer will always be optional, a service can do some logic without need a database, imagine a simple service to do several calculations or consume the data from another service or act as third party services gateway, a social network, a payment service, whatever.

Coming back again to our problem we have decided to split all the back end in four services that will cover the domain of the problem, following a Domain Driven Development principles. Each one will be described with details after, for now, only some reasons that why is so. We need a logic part that controls any system call, some as the gateway that acts as a dispatcher of requests, that call to the service (maybe not only one) that can help it to resolve the requests and give back the info.

For another and, we need cover all requirements specified, and to achieve this we are going to design three services that split all logic issues, it will be **Teaching Database microService**, **Students Control** and **Analysis microService**. All together compose the backend of the app, together with other service that will enlarge the functionality of our project in the future.

So, as a big picture, the backend of our service would look like this, some services with approximately the same form (in the pictur only three, is the same).

Figure 5.3: Some as the first approach of the system.

Once that the architecture is clear and the services are well defined (not their behavior, only their domain) we must have clear how this services will be deployed in the platform or infrastructure chosen because the selection can modify the way to design the services.

So in our case, of all possibilities that we have already discussed before we only consider two as acceptable, Google and Amazon infrastructure, so as you know already, Google was the selected, so we need design our services to run inside of it and this mean that we need how configure our services.

As we always evaluate some options, we have done a simple design to see what will be the infrastructure and design required, first in Amazon Web Services.

In this case, as the code must run into the container we would need configure all our services as Docker Containers and to use the service Amazon EC2 Container Service to deploy them. And our gateway service could be deployed in another container or could be used a specific service called API Gateway Service with Lambda Service, both services was launched by Amazon when they saw that this pattern was really common in all projects, and they design this service to save efforts to developers (even though it have some drawbacks as for example that you do not try it in local, at least for now but is sure that will be solved).

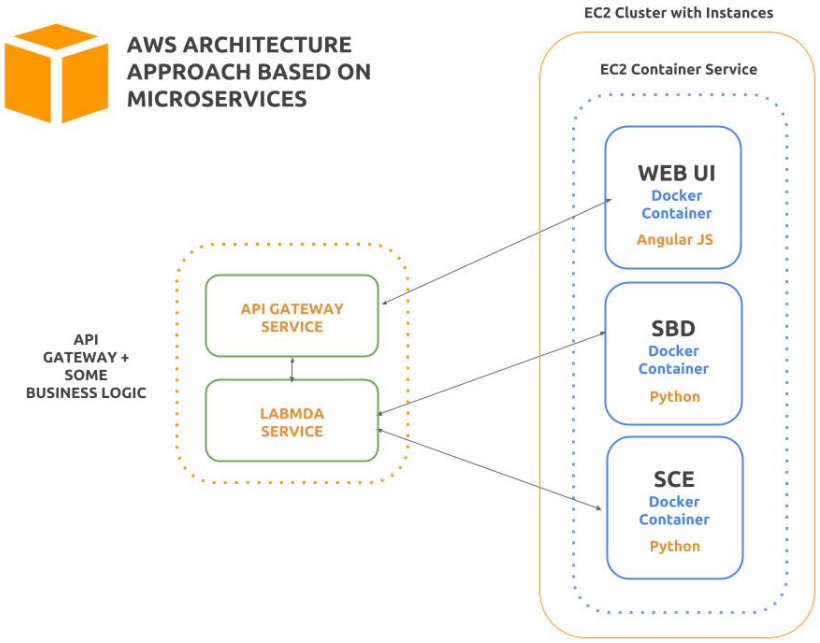


Figure 5.4: Approach in Amazon Web Service infrastructure.

The goal of this study was compare the facilities or drawbacks of each platform (doing a little bit deep study that only to take a look over the solutions each one offered), and the drawbacks of AWS for this kind of "amateur" project was too.

Come back to GAE, the deploy must be different because there are not docker container, we only have special sandbox to run our code with some special characteristics (as wee commented before). But, the architecture does not vary significantly between both architectures, as could not have been otherwise. If this not so, microservices and their platform independence would not have sense.

So, focused on GAE, the services must be deployed and developed (nothing else of some changes in the code) as modules (now called services by GAE) and this can (and must) put joined in GAE applications. So, as the platform are thought and to adjust this to our requirements we are going to launch two apps, one only with the user web interface service (front end) and another with the group of services that compound the backend, (here GAE app 2). You can see the distribution in the next picture.

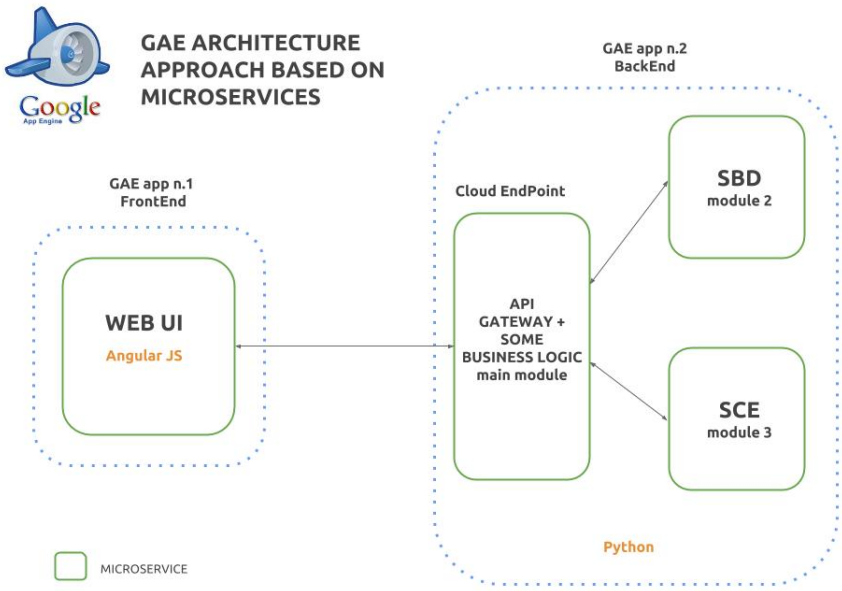


Figure 5.5: The same approach in Google App Engine infrastructure.

5.1.1 Be more realistic

As is easy to imagine, is needed a lot of services to complete all requirements of the project, and presented here is only the most important because define the core of app and their structure but of course are necessary a lot more. Only thinking about security and authentication is easy to see that is necessary other services to check this, but as is a standard service (not taking part in the design of the app) not will be developed now and neither explained here. Their goals are to ensure the authentication of the user and provide the user authorization of different parts of the app. So, this will be Authorization & Authentication microService.

On another hand, there are two more functionalities must be developed as soon as possible in the next iterations, a files storage, and a messaging service. Both are necessary for a more advanced stage of the deploy, not now, but essential in the final product. The next picture shows you all services, developed in this phase and the planned.

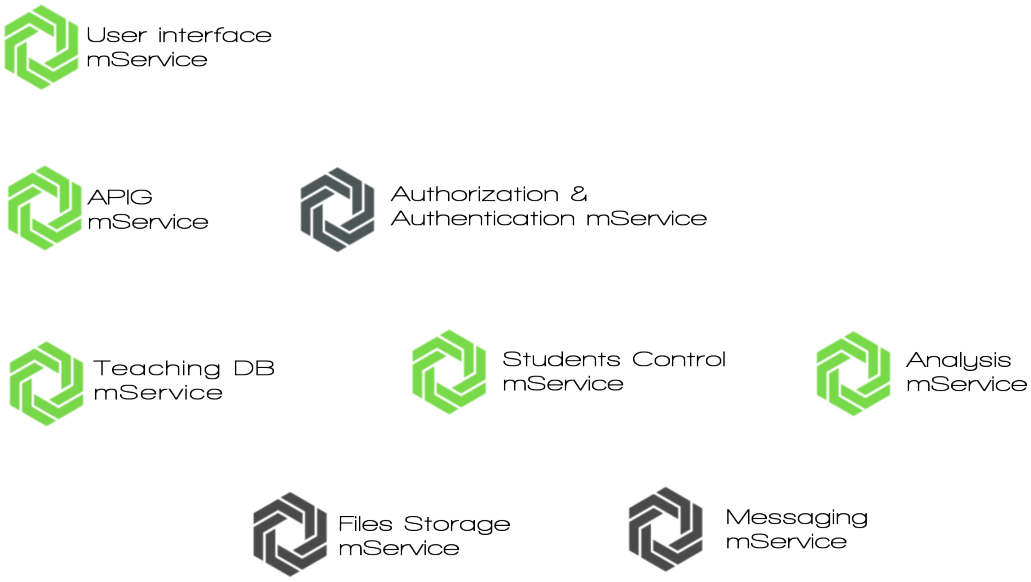


Figure 5.6: Group of microservices of SMS launched and planned.

Next picture shows us a more realistic snap of the deployment, with the services, technologies, tools and third party services consumed. We can see how SCmS need G Cloud Datastore, TDBmS Cloud SQL (for example) and all technologies related.

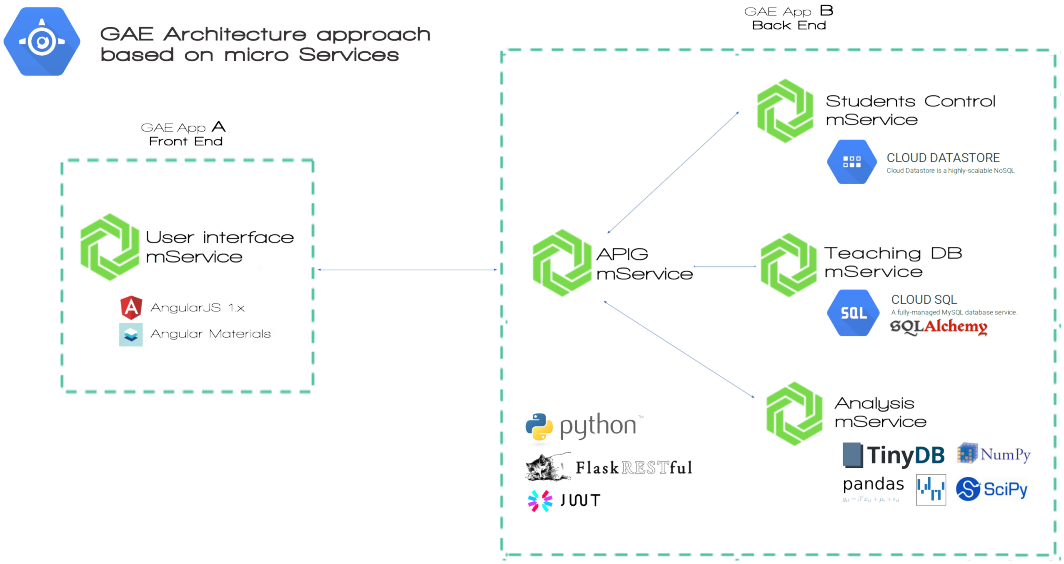


Figure 5.7: Final snap of the distribution of the services with their technologies.

And will be our road map to follow, without any service needed to obtain a pre-alfa product version that satisfy minimum requirements at this point.

5.2 API standar status codes

When we design an API over HTTP we are assuming that we are going to use standard status code of this protocol (HTTP/1.1 in particular) and is useful that from beginning all team knows what code will be used and the meaning. The next table shows us which is these, obviously, there are a lot of more, but this is most important for us.

Status Code	Meaning	Use/Behavior
200	Ok	Any request that returned content.
201	Created.	Not return content and the item has been saved.
204	Ok without body.	Call with success without content.
404	Not found.	Element not founded.
422	Unprocesable entity.	Bad body at request.
500	Server error.	Any unknown kind of server error.

Table 5.1: HTTP1.1 Status Code common used

And as is known, the numbers have a specific meaning and it will be useful to know just in case another will appear.

Class	Meaning
2xx	Successful
3xx	Redirection
4xx	Client Error
5xx	Server Error

Table 5.2: HTTP1.1 Status Code first digit meaning

5.3 Pesistence Strategy

When we are talking about persistence that meaning how the data that change their state in the database is manage. Obviously we are talking of the changes between created, deleted and the between states.

Is been identified three ways to do this:

- **Non persistence**

  When a item is updated or deleted any information of the last state remain.

- **Half persistence**

  When a data is logically erased, the data remain in the database but will never accessible by the user. In this case is useful if we want develop some mechanism to redo actions, in this case only deletion actions.

- **Complete persistence**

  When not only the elements erased can be restored, all items can be restored in any point of the history of the system. This is the more complex mechanism to persistence but also the most powerful and useful for the users, because as in a simple text editor allow do an infinite redo over actions realized.

Afte study all options have been decided use the second approach, and to achieve this is necessary get some kind of metadata plugged to our objects stored in databases to control their state in all moment. To do this has been designed this metadada pattern for all objects saved in any service or database:

```
1   Metadata parameters
2   createdBy       INT,
3   createdAt       DATETIME,
4   modifiedBy      INT,
5   modifiedAt      DATETIME,
6   deletedBy       INT,
7   deletedAt       DATETIME,
8   deleted         BOOL,
```

With it we get a simple way to save logically deleted items, that allow us develop a simple way to recovery this if user want, and not only this, also a system to know who do the actions, who save, delete or update an item and also when, a lot of data very useful in many ways.

5.4 API Gateway microService

This service has not owned API definition because at this point of development only takes the role of dispatcher (with a bit logic in some parts, but minimal) and their only goal is to copy the services API and clone it. Despite this, their task is very important in the system, talk with clients and with services and thanks to this nothing of them need to know about another.

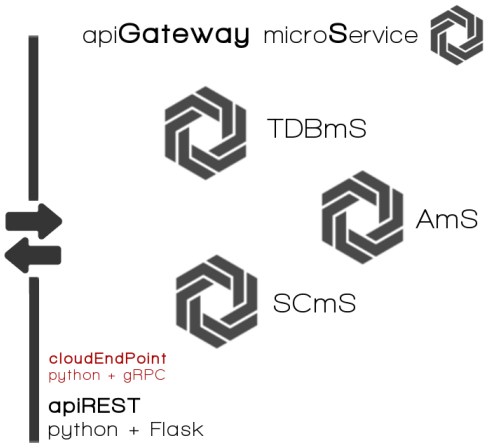


Figure 5.8: API Gateway service concept.

As it can be seen in the figure above, at first it was designed with Google Cloud Endpoints technology, using Python and gRPC but, as we will explain in the next chapter, was not the best way (at least with the system that we can build) and finally was decided to build as the rest of them, using Flak and building a simple and well know API Rest.

5.5 Teaching Data Base microService

5.5.1 Domain and design

This mService offers the managment of the teaching of the center. This means that persist in a relational database all relations between teachers, students, subjects, etc, and all resource availables to make this posible throuhg an API.

This like the rest offers his resource throuhg an API writed in Flask (follow the same architecture that all).

The engine to save all these relations is MySQL, for many reasons, mainly because is the best known engine and in which it has some experience and also because GCP offers as a cloud product Google Cloud SQL Databases. Until recently only offers MySQL but now (since March of 2017) they offer also PostgreSQL.

In the old version it was a little ORM that offers simple methods to access data transform this in SQL raw sentences. Now this library is only a wrapper of the SQLAlchemy to keep apart the APIrest of the service to the database access layer.

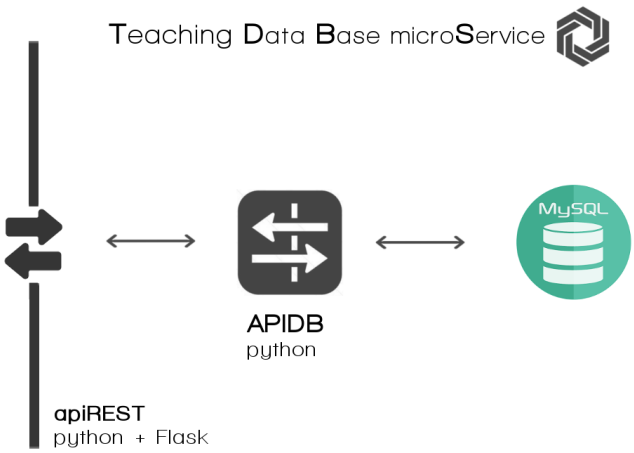


Figure 5.9: Teaching Data Base service parts.

5.5.2 API definition

To get API definitions will be used RAML[2]. In spite of to be more powerful, it only will be used to do a good definition of the APIs independently of the technology with after it will be developed, understandable for any developer.

To see definition please see Appendix B.

In this case, the API definition of this service is like this, and as we can see, is very easy to understand, and we only are going to use the most simple version of this definition standard, there a lot of other things that we can express, but is not necessary as first approach.

5.5.3 Database logical design

Based on user histories and the domain of the problem the design done based of this entity relation diagram:

The design follow some details that the domain presents, that is related (at least the more significantly) below:

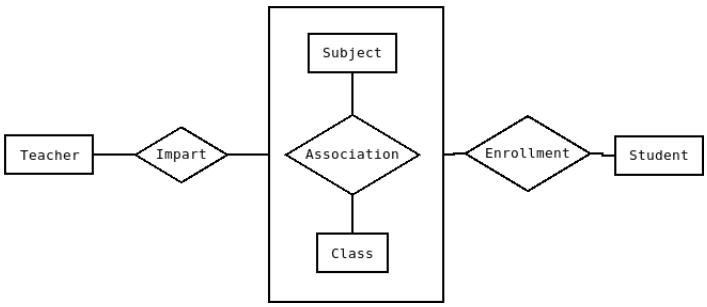


Figure 5.10: Entity-Relationship Diagram.

Data model

Based on requirements and on the previous diagram designed we need to define the model data of the database, the logical structures that will give shape to the objects that we will save with the restrictions and details required. This specifications is normally defined in SQL language. We can see this definition in the file *DBCreator.sql* file in the project folder or a summary in the Appendix C.

[2]RESTful API Modeling Language (RAML) was launched at 2013 and is YAML based language for describing RESTful APIs

5.5.4 Way to access to raw data

While at firs of develop the mainly strategy to follow was write all by zero, finally the point of view has been changed to follow the use of standard tools and avoid reinvent the wheel.

So, if in the first stages of the project the access to raw data through the engine was hand made, using an own simple library that worked like as a simple ORM[3], the evolution of it and especially the problems found and the unmaintainability of code have made that now the approach turned to use a good tool as ORM like SQLAlchemy[4].

The changes in the specifications of the API while the develop and the maintenance of the performance of the queries when is written hand made in raw SQL is not a good idea. After the develop of this mService it easy to understand that only in few projects is justified the use of raw SQL sentences and drivers without ORM (by easy that it was).

5.6 Students Control microService

5.6.1 Domain and design

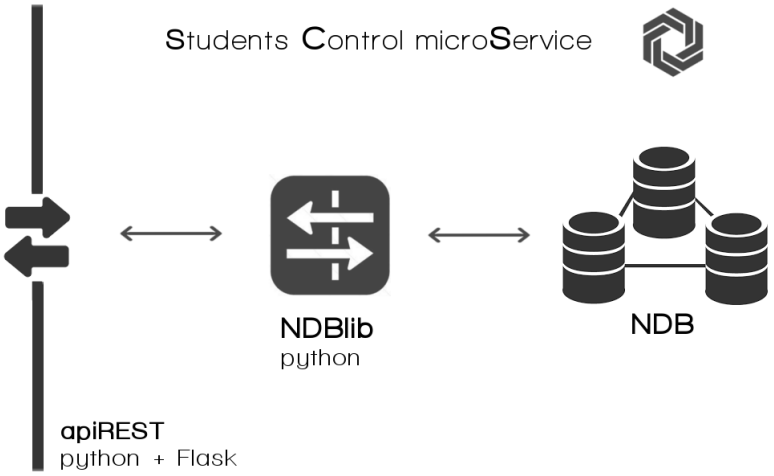


Figure 5.11: Students Control service parts.

[3]Object-relational mapping or **ORM** is a technique for converting data between incompatible type systems using oriented objects programmation to do this, as for example access to database systems.

[4]SQLAlchemy is most popular Python SQL toolkit and Object Relational Mapper. The first release was launched at 2006.

5.6.2 API definition

How was commented before, the design of the resources is based on getting an user interface with minimal requirements. So, there are a lot of ways to get a collection of data to present it on the interface, and we must decide who do the more complex part of this compose, and obviously, we want to save time to the user device. To better understand it, is best an example. If we want to get a list of attendance controls that have been done and if we only request the raw list we only will obtain a list with all data but without any reference to the data of teacher of data of the subject.

So, to can offer by the user interface all data that it need, is necessary to develop the fill of this data before to send. And all this complexity must be a backend issue, indecently of the request or the mechanism that this use. We can see an example of this in *ac resource* when is called with *GET* method.

To see definition please see Appendix B.

Attendance Controls API Subsection

The sub API which we can manage all related with attendance controls, that are records about the attendance of the students to class and something as if the students attends with delay, if this is justified, if gone with or without the uniform, etc. As with the previous, this is the behavior designed RAML based.

Disciplinary Notes API Subsection

With this API section, we can manage all related with disciplinary notes, that are records about the student's behavior in class. With this we can to register negative acts of the students to prevent big problems and to correct this as soon as possible. And this is the behavior designed RAML based.

Maks API Subsection

As the final part of the simple student control that we do we still have the marks section. With this we would record all exams marks that students achieve, that is another big part that was required. As with the previous, this is the behavior designed RAML based.

5.7 User Interface microService

5.7.1 Domain and design

As has been mentioned, we are going to use AngularJS to build the user interface. This means that the files of the app would be load from a service to user devices and from there it will make all requests to the server, so the single purpose of the service is to serve all app files. The design follows bellow schema.

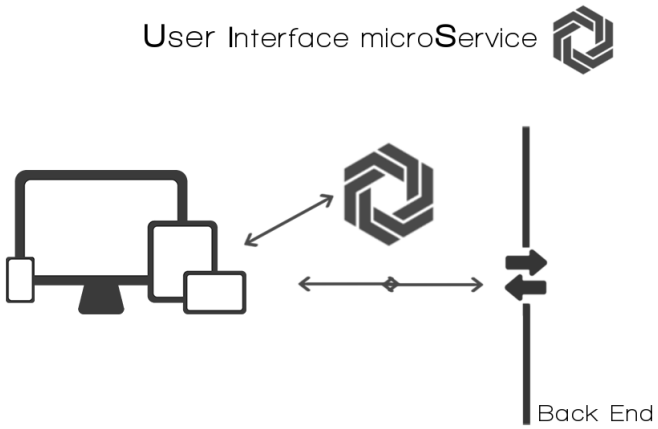


Figure 5.12: User interface service logic.

About design

All the design of the app is based of Java Script files. And the structure will be this, a section o Students Control where will placed all sections related, as Attendance Controls, Discipline, and Marks section and another section, the biggest, with all related with the teaching as can be all sections that the domain has, teachers, students, subjects, classes, and the rest of items related.

The most of the parts need the same javascript files, controllers to define the logic and services to interact with the server. Moreover as the idea is work as each logical part as independently modules they will be built as modules as well.

This is the wireframes that give an approach of the design that we want for our interface. There are a lot of parts, sections and detail that will filled hundred of pages, so this is some examples of the simple a common part as work with list, in this case of teachers.

About distribution

The design of the components of the interface will follow Google Material Design Philosofy[5], it is a design language developed by Google.

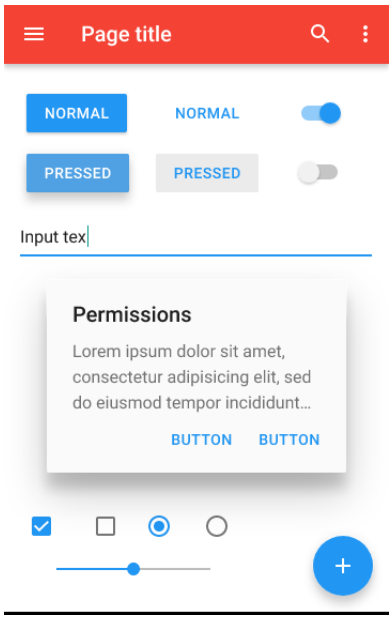


Figure 5.13: Some typical Material Design UI components.

To obtain this characteristic design there are a lot of options, but easiest is to use some CSS Framework existing instead of try to implement our own CSS components. There are some options availables, as *Materialize*, *MUI*, *Material UI*, *Polymer* (that is much more), we could talk about it hours, and *Angular Material*, that is which we are selected. Why? because is not only a CSS kit, is a group of Angular directives that allow us to use their CSS components as directives, some very powerful and simple, that increase the speed of develop when their behavior is well understood.

Indenpendly of the style guide that we use we need design the interface, their parts, the position of the elements and how interact this among them and amoung several screens of our app. To do this design most common way is use a system of wireframing, that is as a way to compound user interfaces using very abstract form, putting the focus in the position and general design instead of the details, colors, or somethings like this. There are some tools to do this, all do basically the same, and we are going to use Pencil (An open-source GUI prototyping tool that's available for all platforms).

[5]Google announced Material Design on June, 2014, at Google I/O conference

We would need a dozen of pages to show all sections of app. Bellow it can see some example of design with two samples of teachers section, that it will follow in the rest of the design, that possibly will suffer some change in the way.

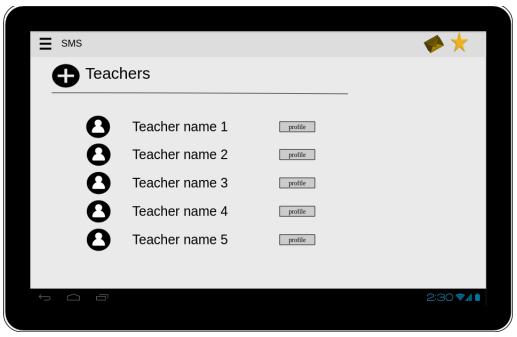


Figure 5.14: Teachers List view wireframed.

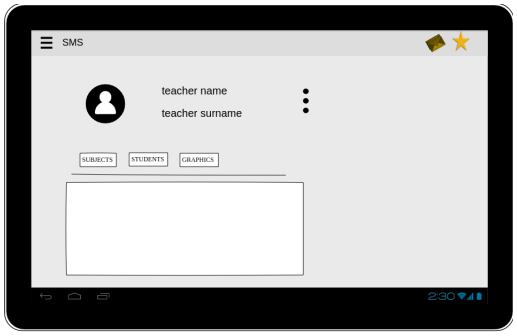


Figure 5.15: Teacher profile view wireframed.

The rest of them have been handmade (on paper) and it was believed unnecessary to include all digital wireframes here.

5.8 Analysis microService

The goal of this service is easy, we need a centered way to can retrieve reports of all of our relevant kind of items in the app, as courses, classes, teachers and most specially, of students. If we think only in simple reports maybe is not necessarily an independent service that offers that (only this is enough interesting the services to manage the relations keep their simplicity) but if we think in report that included statistics and data analysis as regression, trends and alert detection in students behavior the scenario changes completely.

To do this we need to use some specific tools, libraries, and techniques which have little to do with the domains of the rest of services. Moreover of this, to achieve some of the conclusion that this service will obtain will be necessary cross data between services, so externalize this in another actor in the system is more reasonable.

5.8.1 Domain and design

Logically, this service follow the same design of the rest. A library that wraps the complexity of the logic is accessed by a standard API (as the rest as well). The main difference is that the database used in this case is TinyDB[6], a lightweight document oriented database, written in pure Python (without external dependencies).

This was chosen for a simple reason, we need some storage system to save the data calculated to avoid do them again for each request, and as is a service that not store a big amount of data is a good option and really easy to implement. The picture below show the logic of the service.

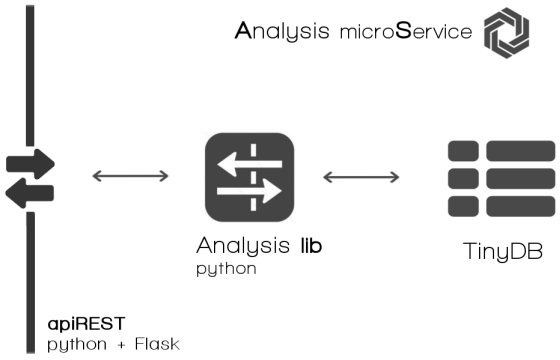


Figure 5.16: Teacher profile view wireframed.

Our goal with this service is can get obtain more complex data that the services can offer us to can offer to the user interface good data blocks to build powerful graphics. As in the domain that we are the students have been evaluated by competencies in addition to particular subject skills and knowledge the app must offer some way to can check these competencies at general scope (independently of the subject) and this kind of information is perfect to use (among others) *polar charts as these*:

[6]More info about the system here: https://pypi.python.org/pypi/tinydb

Figure 5.17: Subjects and competences polar charts.

In these, we can see as a specific student has their competencies covered and what it need to improve and the same with the subjects, not only in this course or semester, also comparing with previous of futures using simple regression equations. We can imagine the power of this kind of shots of data by the educators of the center, that would require a lot of work if they would do handmade.

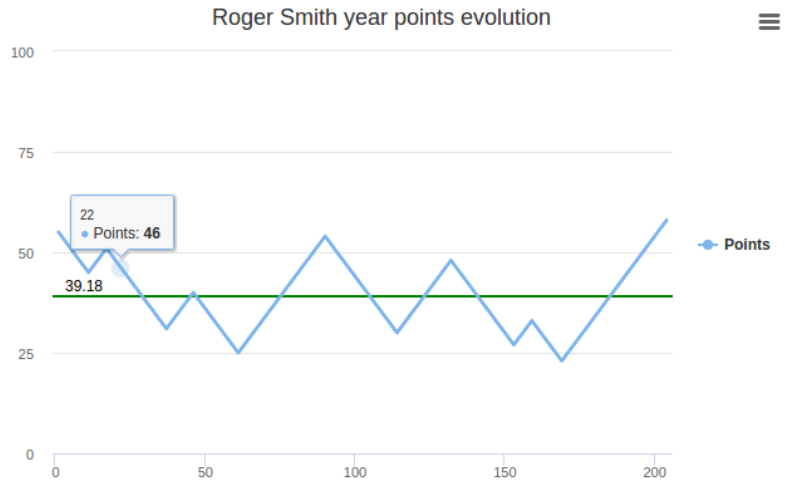


Figure 5.18: Example points evolution of a student.

Not is the goal review all possibilities of data flow that it can be exploited with all items and their relations, but the above picture shows another interesting bunch of data well presented to be analyzed by an educator. As we

save all things that a student do in the center, homework, marks, behavior, delays, etc. We can build a points system that in a range of 0 to 100, we can measure the efficiency of the student per week. That means that if a student does all their homework, all their exams well, without any delay, etc. It will be 100 points in this week and the opposite if it does not do anything.

It is mighty because in a look we can know how the student is, how is their evolution and the best interesting (using simple regression) which is their trend, to be able to make decisions proactively, not when will be too late, improving drastically the way to understand the student tracking systems. **And it is only the begin**.

5.8.2 API Definition

In this case, the API is very small (not meaning nothing) because is not few relations between items inside, we only need a way to request reports to the system, nothing else.

5.9 Auxiliar tools

5.9.1 Provisioner

A provisioner is simply a tool used to provide a system to insert test data, to avoid to do this hand-made, in our services.

Figure 5.19: General idea of service.

Normally this work without knowledge of the internal parts of the back-end, only working with their API gateway (or API if we want only fill the database of service), understanding it as a black box. this is only an approach, we can design our provisioner to fill data directly in the databases, which will require established a connection with them without the interaction of the services or their connection libraries. This is faster but on some occasions, we do not want to do this, because another of their goal is to check at the same time the correctly of all parts of APIs involved in the

data insertion.

So, the execution this kind of program required that the system has fully launched. This will be the most used tool when we want to try some feature that required a lot of data in the system.

How does it work? Easy, only need simple parameters as entry and based on some rules it will do all calls to the API gateway to insert all data required, besides to save all operations in a register or log file because is necessary to check this in a debug operation.

5.9.2 Data base helpers and testing tools

To all those repetitive tasks as clean the database, create it, populate or something like this, reinstall the engine or purge of the system we will need some scripts to help us and will be necessary develop this.

About the testing tools, side server will be enough with the normal testing using pytests (we could talk a lot of it) and over user interface, we would be using Selenium[7], in words of their creators, textitit is for automating web applications for testing purposes but is certainly not limited to just that.

Thinking of this pair of details, we would have covered all design decision taken, ready to start do develop, crossing the fingers to get all in time.

[7]Know more at *www.seleniumhq.org.*

Chapter 6

Develop

As talk about all details that are been solved or managed would take hundred of pages in this section we only are going to revise some part of developing that are been interesting or have required some bit specific design or decision process and are interesting to explain.

All details of implementation and the own code, are available in Github, at **github.com/ButterFlyDevs/StudentsManagementSystem**, including the branch with minimal website app for advertising purposes.

6.1 Api Gateway microService

As was discussed above, the principal goal of this service is to offer the abstraction of the whole system, so it will not be very complex because all functionality is actually implemented when this service is built.

Their unique task is to receive the requests, to construe this and do the call to the service that must answer it and return the data, maybe modified. In common case and in the point of the develop are the service only reply the queries and return the responses without more logic interaction, but in the future, this is not only that this will do, because is the perfect place to implement authorization and authentication with ACLs[1] exploiting the fact that through this service all calls go.

Another situation which this service would transform the response of a service or compound a response with the response of few of them is when must be offered a resource that is the composition of data arising from several services. For example, when a user profile info is required, the call will be redirected to Teaching Data Base mService that will return a simple data

[1] ACL of Access Control List specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects

block. So, this data block does not have the image of the user, because this is stored in another different service. Now the role of gateway take sense because will be it who will retrieve the image from other service, will insert it in the data block (by some way) and will return the data block complete.

It also provides some possibilities to have a dynamic and not locking develop process. How? Easy, because at first the images can be extracted in execution time from some library that put example images (or something like this) without locking the APIGmS and also UImS and so, this service of data storage could be developed after, in another phase of priorities. So, one more time, the architecture help us to develop team domain based without any blocking.

6.1.1 Cloud Endpoint spike

As we are working with GAE and GCE the first approach is use their technology, and if in any resource that you can read about this they are talking about Google Cloud Enpoints must be some good reason. This technology is based of an own version of RPC protocol developed by Google and used in their own intern architecture, called gRPC.

Basically is another implementation of a Remote Procedure Call system, but according to Google, really fast and nice, and is true, but has some drawbacks as for example that can you select other common technologies, as in our case.

This is an example of the spike related, it can see in the first phases of the project.

```
1  ...
2
3  import endpoints
4  from protorpc import messages
5
6  ...
7
8  class Alumno(messages.Message):
9    nombre = messages.StringField(1)
10   apellidos = messages.StringField(2)
11   id = messages.StringField(3)
12
13 class ListaAlumnos(messages.Message):
14   alumnos = messages.MessageField(Alumno, 1, repeated=True)
15
16 ...
17
18 @endpoints.api(name='gateway', version='v1')
19 class GatewayApi(remote.Service):
20   """Gateway API v1."""
21
```

```
22   @endpoints.method(message_types.VoidMessage, ListaAlumnos,
23                     #path=nombre del recurso a llamar
24                     path='alumnos/getAlumnos', http_method='GET',
25                     #Puede que sea la forma en la que se llama desde
                           la api:
26                     #response = service.alumnos().listGreeting().
                           execute()
27                     name='alumnos.getAlumnos')
28
29   def getAlumnos(self, unused_request):
30     ...
31     students_list = []
32     ...
33     # Logic to get the students list from another service with maybe
           more logic.
34     ...
35     alumnosItems.append(Alumno( id=idAlumno, nombre=nombreAlumno.
           decode('utf-8'), apellidos=apellidosAlumno.decode('utf-8') ) )
36     return ListaAlumnos(alumnos=alumnosItems)
37 ...
```

This is only an example, but the complete file has about 2000 lines in the first approximation of the service. Now take a look to the *Flask* version, an example bellow, having exactly the same functionality of previous code segment.

```
1 @api_gateway.route('/students', methods=['GET'])
2 def get_students():
3     ...
4     students_list = []
5     ...
6     # Logic to get the students list from another service with maybe
          more logic.
7     ...
8     return students_list
```

The reasons to select Flask instead of it? Is not necessary define the schema of the objects, this is sufficient reason to dismiss their use, but sound enough nice to consider to another applications in the future (as almost any technology researched).

6.1.2 As a simple dispatcher

This is an example of the behavior dispatching a simple request and answering exactly the same response from the service.

```
1   @tdbms_segment_api.route('/entities/<string:kind>', methods=['POST'
        ])
2   def post_entity(kind):
3       response = requests.post(url='http://' + str(modules.
            get_hostname(module='tdbms')) + '/entities/' + str(kind),
4                                json=request.get_json())
5       response.headers['Access-Control-Allow-Origin'] = "*"
6       return make_response(response.content, response.status_code)
```

```
1
```

```
2   @tdbms_segment_api.route('/entities/<string:kind>', methods=['GET'])
3   def get_entity(entity_id):
4       ...
5       response = requests.get(url='...', json=request.get_json())
6       ...
7       response['profile_image'] = requests.get(url='filesServicesUrl
            ...')
8       ...
9       return make_response(response, 200)
```

And the next natural step will be built a library that works as a general customer of any service in the system, to be used in any place where be needed make a call to any service. This will to work loading dynamically at execution time the API of each service to know what resources are available and turned this as Python methods.

```
1
2   @tdbms_segment_api.route('/entities/<string:entity_id>', methods=['
        GET'])
3   def get_entity(entity_id):
4       ...
5       response = service['teachingDBmS'].getEntity(entity_id)
6       ...
7       response['profile_image'] = services['filesStorage'].get(file_id
            )
8       ...
9       return make_response(response, 200)
```

With this new tool, the code of the whole project will be reduce 10% at least.

6.2 Teaching Data Base microService

The deploy of this service is very easy, only has been wrapped the SQL language in a kind of own ORM, that would be changed by SQLAlchemy as soon as possible in the next iterations of the project but that is enough for us now. So, instead of explaining how it is done (it can see in the code of the project) we are going to explain the more sensible parts and problem detected, to avoid extend too much this work.

6.2.1 Optional subjects, the *"class"* table

In the domain of the problem can be exists optional subjects and is needed search a way to implement this because has a specific details that aren't like the rest.

The studies plan forces in certain courses to select one or several optional subjects. For example, if a student has enrollment in 2ºESO (independently of the group, A, B...) the law and consequently the studies plan force to the student to choose between some optional subjects. So, maybe this subjects exists only in this optional case as *"rare subjects"* but in other cases this are

only normal subjects but that in this course are offer like optional.

A simple example of this is French subject, it in some courses like 3ESO and 4ESO is obligatory but it in *"Bachillerato"* (Spanish high upper level, something as Bachelor) is optional because the students can be select if they want make the final exam with this second language or select another like English or Greek or Latin p.e.

To obtain this we decide develop a simple solution without change the original database logic schema. So, how we have an entity that save the classes and it have three attributes, course, word and level mainly we going to add three more to this special cases, optional, groupNumber and subgroupNumber. Like this special cases have not word parameter when they have value word do not have and when the item have word (A, B, C...) then don't have this special attributes.

Maybe this don't be the best solution, but is a simple in the point of develop. Obviously like we can't have two autoincrement values in the same table definition in MySQL we will need control this programmatically, but is something that we can assume to get our goal easily.But we found a problem.

The same advantage that offer *UNIQUE* to deleted cases now is problematic here. While there works fine because this clause does not include to items that have fields to null and allow to exists without conflict in this case if we saved a optional group obviously would to be *"WORD"* field to null, and if we do this we can have two groups exactly equals, for example:

```
1 1 <null> ESO 1 1 1 0
2 1 <null> ESO 1 1 1 0
```

This could happens without conflict, and this should be impossible. For this reason we decide to use the same field *WORD*, with a special naming, because this never will be used by general groups, that help us to specify that the group at issue is an optional and also specify the group and subgroup.

This is something like this: OPT_n_m Where n will be the group number (must be increased handmade) and m the number of the subgroup (also increased handmade). Obviously this is only a simple approach to a first solution, will be improved in next iteration of the service, and we always follow the same philosophy, we prefer explicit way to do something to implicit, more understandable by any new developer.

This way to solve this problem with MySQL engine presents some problems. As all is managed by MySQL but there are not way to do this automatically with the own mechanism of MySQL is necessary to create some dispatcher

to the insertion and the deletion to maintain the consistency. When a new optional group is introduced must be checked that exist the previous (group and subgroup, can not exists group 4 if 3 does not exists), and the same way should not be possible delete the group 4 if the number 3 exists yet.

Note that is important understand that a lot of detail of consistency control can be relegated to UI, because although there is not allowed by the logic of interaction from the API must not be allowed neither, to prevent inconsistencies malicious induced.

Because of this not only the security, also the consistency must be ensured in any action that user can do.

6.3 Students Control microService

In general terms, this service follows the same philosophy of the rest, with the difference that they use the Cloud Datastore. In general terms, this service follows the same philosophy of the rest, with the difference that they use the Cloud Datastore. That only means that the library of access to raw data is different and instead of wrap the SQL language as in the last, here we wrap the access to a library designed by the Cloud Datastore service called **ndb**, designed by the same *Guido Van Rossum*[2] (in the time when worked at Google).

Most characteristic here is that we need to define the form of the data that we want to save (this is a heavy reason to do not use again this system, we will comment this a bit more after). So, to can store the disciplinary note item type, is necessary to define a class which ndb can manage it.

```
1  class DisciplinaryNote(ndb.Model):
2      # Related academic info.
3      studentId = ndb.IntegerProperty()
4      teacherId = ndb.IntegerProperty()
5      classId = ndb.IntegerProperty()
6      subjectId = ndb.IntegerProperty()
7
8      # Disciplinary Note
9      kind = ndb.IntegerProperty()
10     gravity = ndb.IntegerProperty()
11     description = ndb.StringProperty()
12     dateTime = ndb.DateTimeProperty()
13
14     # Item Metadata
15     createdBy = ndb.IntegerProperty()
16     createdAt = ndb.DateTimeProperty()
17     modifiedBy = ndb.IntegerProperty(default=None)
18     modifiedAt = ndb.DateTimeProperty(default=None)
```

[2]Dutch programmer, author of the Python programming language.

```
19        deletedBy = ndb.IntegerProperty(default=None)
20        deletedAt = ndb.DateTimeProperty(default=None)
21        deleted = ndb.BooleanProperty(default=False)
```

And using this definition is build a wrapper to interact, this is an example of this same section:

```
1   @classmethod
2     def post_dn(cls, disciplinary_note):
3         if cls.validate_dn(disciplinary_note):
4             dn_to_save = DisciplinaryNote(
5                 studentId=disciplinary_note.get('studentId'),
6                 teacherId=disciplinary_note.get('teacherId'),
7                 classId=disciplinary_note.get('classId'),
8                 subjectId=disciplinary_note.get('subjectId'),
9                 dateTime=datetime.datetime.strptime(disciplinary_note.
                      get('dateTime'),"%Y-%m-%d %H:%M"),
10                kind=disciplinary_note.get('kind'), gravity=
                      disciplinary_note.get('gravity'),
11                description=disciplinary_note.get('description'),
                      createdBy=1, createdAt=time_now())
12            key = dn_to_save.put()
13            return {'status': 200, 'data': {'disciplinaryNoteId': key.
                  id()}}
14        else:
15            return {'status': 400, 'data': None, 'log': None}
```

With their API resource in Flask:

```
1   @disciplinary_notes_api.route('/disciplinarynote', methods=['POST'])
2   def post_disciplinary_note():
3     return process_response(DisciplinaryNotesManager.post_dn(request.
          get_json()))
```

Actually is very simple, the rest of behavior follow the same pattern, and the only benefits that have the time spent in learning to build the service with this technology is the amazing performance and cost that store data here has.

6.4 User Interface microService

6.4.1 Save process flows

The logical process of saving data is apparently easy but it hides some details when we talk about update existing data. These and how to solve the problem that it presents will define how the user will use our interface. And spite of this is a design phase issue is in the develop moment when this appeared and is why it is explained here.

Focused on the problem, we have an object load in the interface, as a complete profile info of a student, and we want to update their data (change or add some new data) and this is not trivial because will define the signature between the API of service and the user interface.So, basically, we have

found three ways to do this so we are going to analysis each one to choose the best, even though there are combinations. When the object is loaded in our interface we modify some field and push the save button if all the form requirements are satisfied then the complete object is sent without any else requirements.

1. **Common save button**

   The object is sent always, independently if it really has suffered changes. Save button is always available while the requirements of the form are satisfied.

   Disadvantages: The object is always sent, even if is not necessary because it has not suffered changes, hence a lot of bytes will be sent without sense.

   Advantages: Is the simplest and faster implementation for the user interface and the server do not need to check anything only override the object updating their metadata.

2. **Automatically saved**

   The object is sent each time as it suffered any change, save button not exists. Their behavior is similar to an online text editor where all changes are saved implicitly.

   Disadvantages: The calls needed are huge if the user to do a lot of changes. Is necessary another parallel mechanism to maintain the possibility to do undo because if all changes are saving at the moment in the server there are not any way to do a simply undo action. A number of calls needed are huge.

   Advantages: Is really futurist because the user does not to need to push any button after of update their data. All forms become more clear.

3. **Totally checked**

   Is a more efficient way that the rest. In this case, when the object is loaded in the interface a copy is done and saved also, always, independently of the interaction of the user. With this, we achieve to get a copy of the object without any modification when the user is updating some field.

   So, each way that user change some field of the form the logic check if there are any difference between both objects (original copy and the modified) and only when this difference exists and the requirements of the form are satisfied the save button will change to enable. This way if the user cancels the update of the object or after of thousand modifications leave the object exactly with the same data the send

button will not enable because there are not any to update in the server.

Disadvantages: The computing requirements in the user device it will high because it will be needed hundred of checks in a simple interaction.

Advantages: The server is not involved until the object is really updated. It supposes a user experience that gives to interface more intelligent aspect.

As is easy to imagine the option chosen is the third because the reason explained above. Down below is shown a piece of code which we have developed the feature of object copy. When we can to see how is used a variable to save the state of the object, after the object is saved and watches is enabled with it to detect any possible change that will modify the state of save button, all after of the student object has been returned.

studentProfile.js

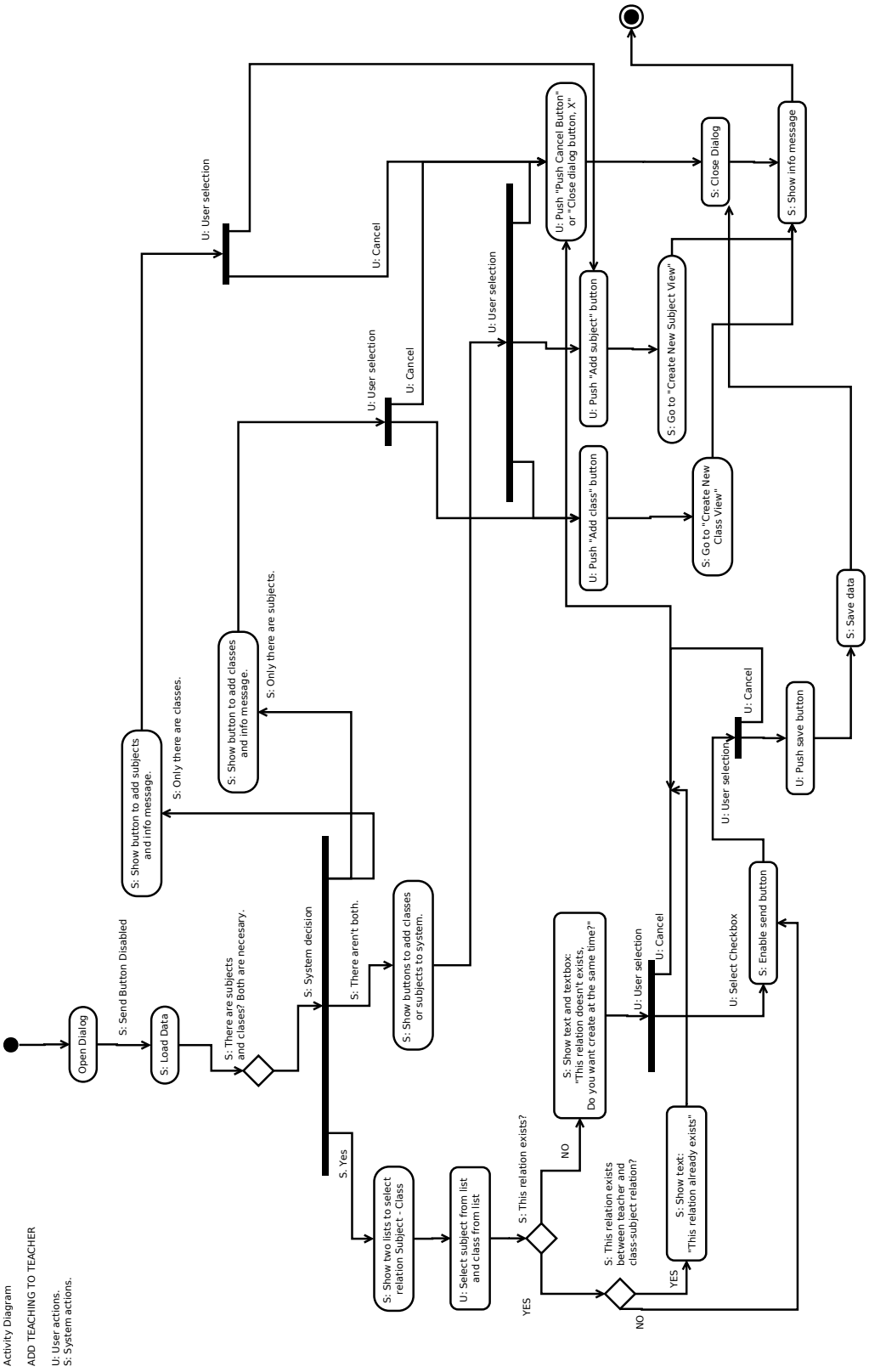
```
...
function loadData() {
    vm.student = StudentsService.get({id: vm.studentId}, function () {
        ...
        vm.studentOriginalCopy = angular.copy(vm.student);
        $scope.studentModelHasChanged = false;
        $scope.$watch('vm.student', function (newValue, oldValue) {
            $scope.studentModelHasChanged = !angular.equals(vm.student
                , vm.studentOriginalCopy);
            if ($scope.studentModelHasChanged)
                vm.updateButtonEnable = true;
            else
                vm.updateButtonEnable = false;
        }, true);
    }, function () {
        console.log('Student not found')
        ...
    });
    ...
}
...
```

So, this behavior is replied in all places where this kind of interaction appears.

Another important thing to design is the interaction between the differents parts of the interface. The menus, buttons and the flow among these can be the difference between a good and bad application because, at this time, there are few things that do not exists, but between the existing, a good UI/UX in one of two similar apps can be enough to triumph.

In our case, all interactions are well-defined, trying to build a simple and powerful user interface. This is an example of the interaction flow that

defines the processes to add some teaching to a teacher. This is only an example, we could fill dozens of pages with flow diagrams, but we think that it is enough representative.


Activity Diagram
ADD TEACHING TO TEACHER
U: User actions.
S: System actions.
Open Dialog
S: Send Button Disabled
S: Load Data
S: There are subjects and clases? Both are necesary.
S: Show button to add subjects and info message.
S: Only there are classes.
U: User selection
U: Cancel
U: Push "Push Cancel Button" or "Close dialog button, X"
S: Close Dialog
S: Show info message
S: Show button to add classes and info message.
S: Only there are subjects.
U: User selection
U: Cancel
U: User selection
U: Push "Add subject" button
S: Go to "Create New Subject View"
U: Push "Add class" button
S: Go to "Create New Class View"
S: System decision
S: There aren't both.
S: Show buttons to add classes or subjects to system.
S: Yes
S: Show two lists to select relation Subject - Class
U: Select subject from list and class from list.
S: This relation exists?
NO
S: Show text and textbox: "This relation doesn't exists, Do you want create at the same time?"
U: User selection
U: Cancel
U: Select Checkbox
S: Enable send button
S: This relation exists between teacher and class-subject relation?
YES
S: Show text: "This relation already exists"
NO
YES
U: User selection
U: Cancel
U: Push save button
S: Save data

One has been decided the design with wireframes and the interaction with flow/activity diagrams the rest is to implement this with CSS Framework selected and develop all logic so the system satisfies all requirements. This is some snaps of the interface already designed.

The teacher list, where we can see a simple dynamic list to manage teachers, where we can see basic info and can access to complete info. And can see the material design rules applied already.

Figure 6.1: Teachers list loading.

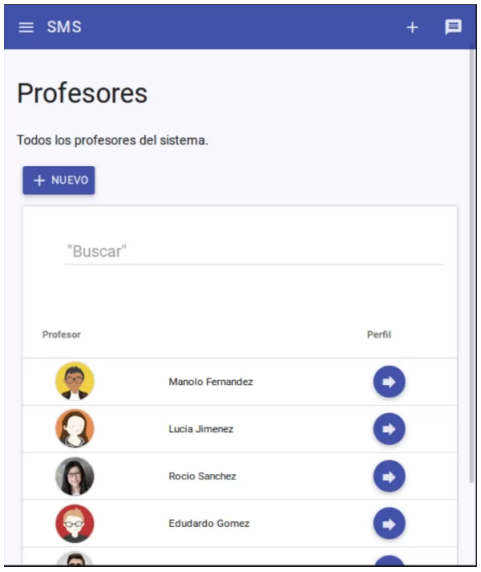


Figure 6.2: Teachers list loaded.

The pictures below show us teacher profile section, with the same philosophy.

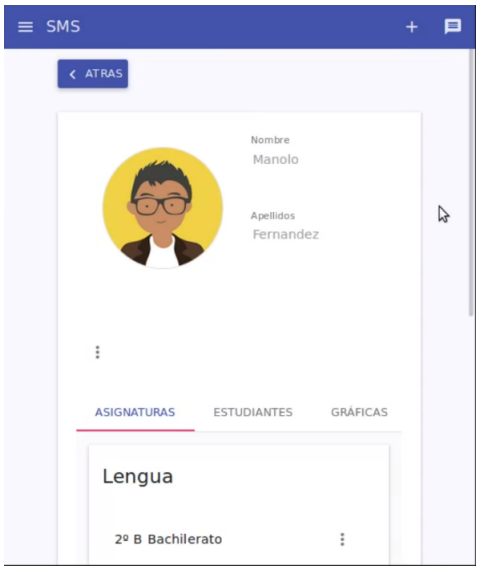


Figure 6.3: Teacher profile.

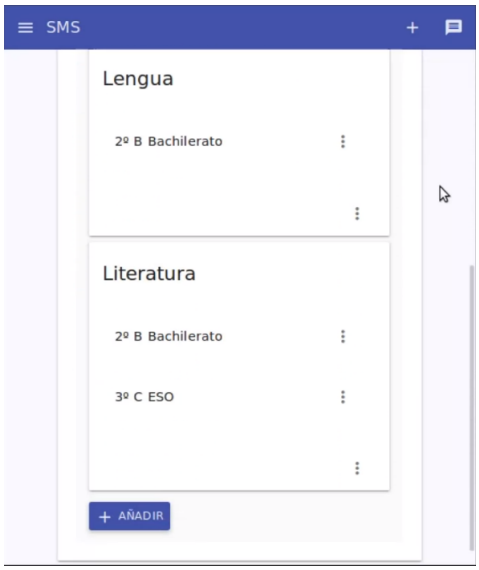


Figure 6.4: Teaching section.

Aso, for example, updating a teacher profile (note that this is the Spanish version).

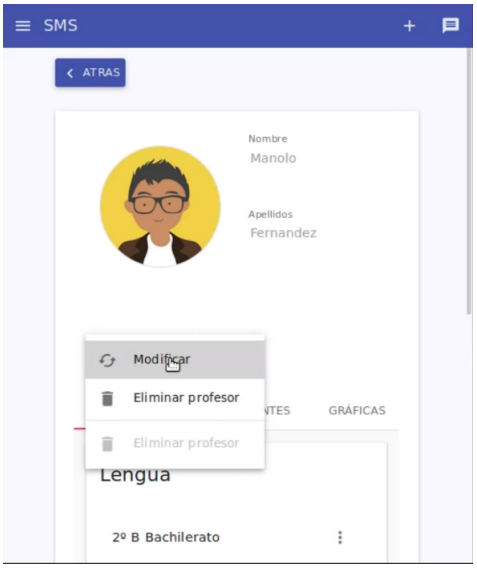


Figure 6.5: Updating a teacher.

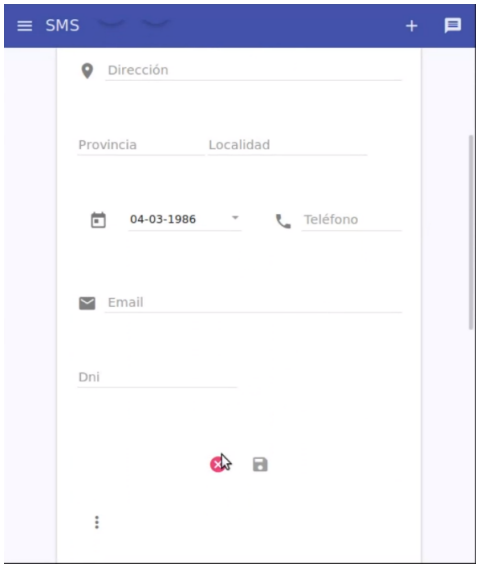


Figure 6.6: Fields hidden.

Now, students list view and some graphics.

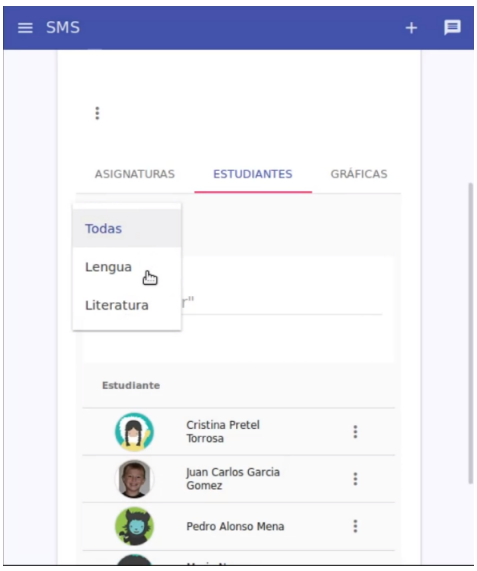


Figure 6.7: Students list.

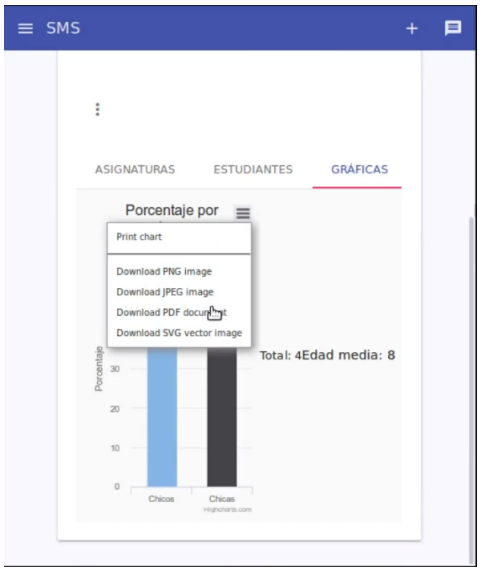


Figure 6.8: Students simple report.

And the rest of app follow the same look and feel, responsive and well adaptable for almost any device.

6.5 Analysis microService

In the case of this service, the developing not have gone too far and only have been implemented some simple linear regressions to can give a simple way to predict the future values of some data block, as student marks or their efficiency (as we have discussed already).

In conclusion, this service will offer mocks of all data expected and will be the main goal in futures sprints of the project.

Chapter 7

Conclusions

Only a few months might not sound much but is enough to obtain some conclusion about techonologíes, patterns and ways to develop.

This work can not finish without a reflexion about what have been the mainly drawbacks and locks in the develop, design and evolution of the entire project, trying to propose solutions and overall, learning a lot about all.

7.0.1 Scheduling

The scheduling of development is very easy to do, in the hypothetical case that all goes fine, but as is normal, there are a lot of factors hard to control that besides are unknown at the beginning.

If we join this with an inexpert scrum master and technologies absolutely new for the team (without a good spikes issues to select and try these) the result can be catastrophic.

If we remember the planning that we did in the chapter three, we had estimated the project in 6 months, that was equal to 12 sprints, and when them finished the project would be finished, as can see in the graphic.

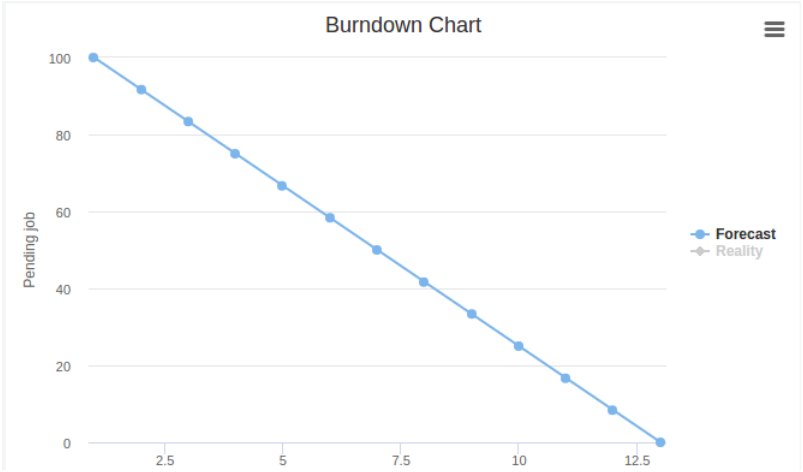


Figure 7.1: Project forecast.

The problem is that the develop has not passed as we would have liked, and, without going into too much detail, after of our auto evaluation and evaluation of the sprints and the work that has been finished we can say that only is finished approximately of 50% that we expected to be a stable first version that we can put in production. You can see the values in the next graphic.

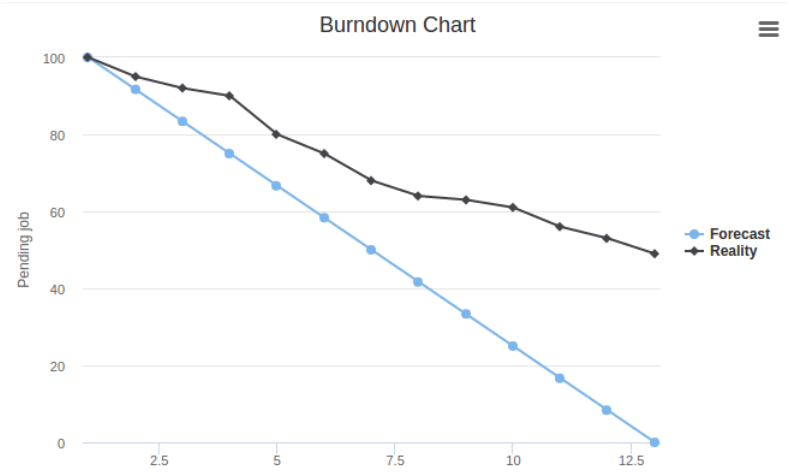


Figure 7.2: Project forecast and reality.

Obviously, if the project continues, it would need a lot of modifications and analysis to detect fine-grained which are the reasons of the 50% of delay, in addition to those already detected here, because doing an estimate of the time that at this rhythm the project would required without any modification in the process of production we have that we would need around

25 sprints, that means another 6 month of develop.

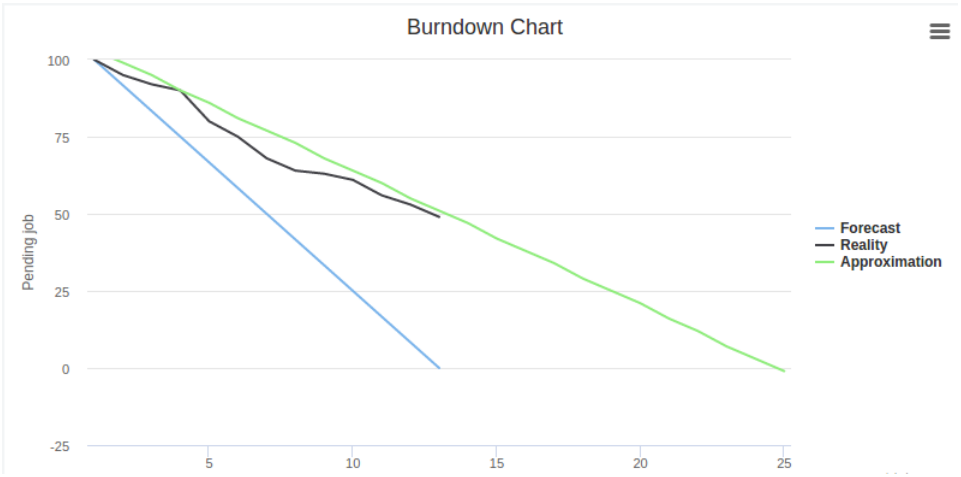


Figure 7.3: Burndown chart with planning, work done and prevision to deadline.

These are not good news, the double of time means the double of resources (time and money) and loose customer confidence, some that we never can not lose. Despite this, for our feeling, and talking about an pseudo academic work, the result is not as wrong as could be, and with all predictable fails, it has not been a bad finish. Knowing that this kind of things only can happens one time, and should be use to learn and to avoid to make the same mistakes.

And if we should be do the same project again, our provision would be other very different, planning exactly the same issues, we calculate that could be that in less of a half of time, as this forecast shows.

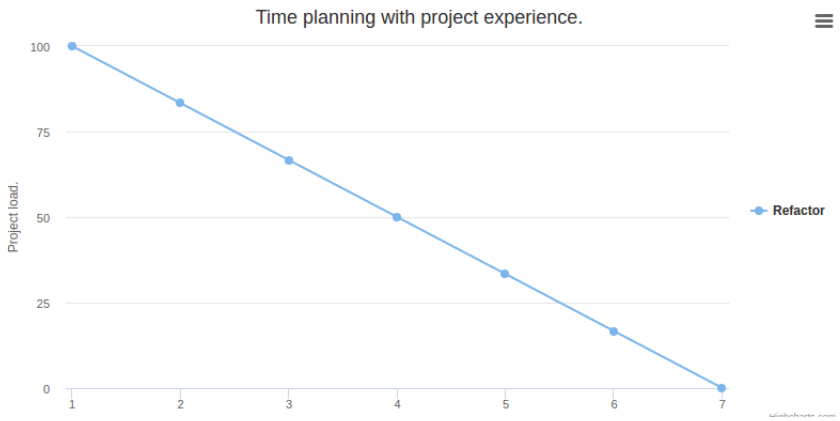


Figure 7.4: Replaning experience based.

7.0.2 Technologies and frameworks

Of one side, we have the difficulties with the technology choice. In this case can be said that use AngularJS and Python has been literally perfect, beyond that the typical novice errors with the languages and their learning curves. So, in general, without a doubt, they will be chosen as technologies again.

About the platforms or technologies the point of view changes. If you are an expert developer using platforms like Google App Engine can be really interesting, because you are evaluated the rest of the options, but when you don't have any practice, in my view, is not a good option. Moreover when the learning curve is so soft.

As in many new technologies is easy to do the first steps, but develop some bigger is another thing, especially when we are not talking about frameworks and languages standard as C++, Java or PHP. So, before to select one is really justify the spike of some of these.

About the API frameworks, Flask was a good selection, because their ease of use and lightness does it perfectly. However, when we analyze the performance, there are other solutions also in python faster. An example of this is Falcon[1]. As can be seen in the next picture, extracted from py-frameworks-bench[2] project, we can see how Falcon has better performance than Flask. In this graphic is measure the response in ms of encode an object to JSON and return the response.

[1]Defined as: *"A very fast, very minimal Python web framework for building microservices, app backends, and higher-level frameworks"*. More info at: *falconframework.org*.
[2]klen.github.io/py-frameworks-bench.

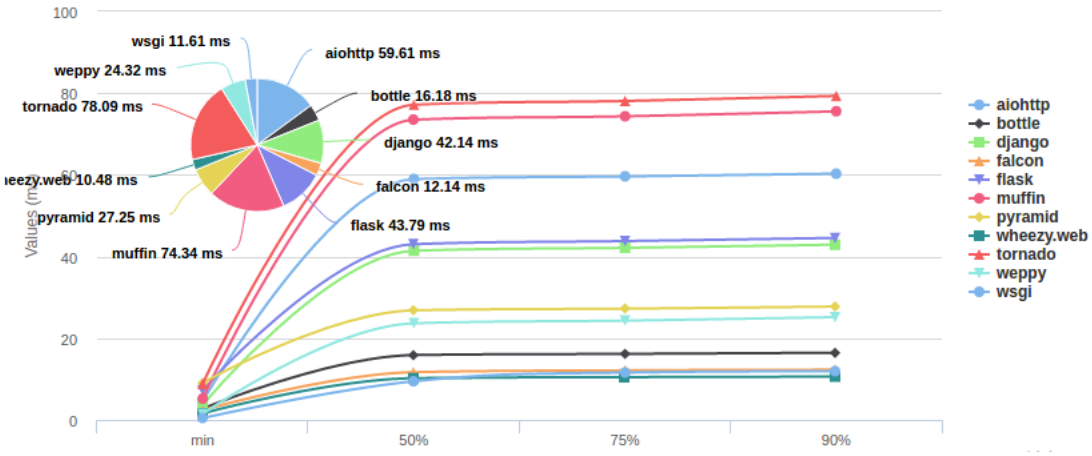


Figure 7.5: Frameworks performance comparison.

And Falcon would be our selection today (even more having some practice in company).

About Sphinx, is a good choice, but it has something that could be improved and is the requirements of that the project must have. To use Sphinx your project needs to be *executable*, understanding this as all imports must work. Many people do not know why Sphinx have this restriction, so if you only want doc, independently of if you structure is correct or not, well, this is impossible in the actual version, we hope that change this at any moment.

About the platform

In spite of Google App Engine is a good tool it has some drawbacks that some are very obvious at first and others that can go unnoticed until you are working with it. the sandbox restrictions as use python only in their version 2.7 at first is not a problem but when you discover that the most of the libraries that you need works better in 3.x, or some are deprecated in 2.7, out of maintenance is not a good signal, especially when you have some block and the help of community is focused on upper versions. It happened in the standard version of App Engine sandbox, to solve this the team of it launched App Engine Flexible Environment, where you can run any code but the configuration is not trivial and it although is configurable by docker file is a strange mix between the auto maintained and scalable isolated sandbox and the standard containers developer managed.

So, taking advantage of all services of the infrastructure of GCE, another architecture that we will develop in the future will run only into docker containers, using the services required (SQL, DataStore or another and running over compute layer, not with Google App Engine.

About the database

About database, without doubt, MongoDB is nowadays the better solution that can be used in a project with low relational requirements, because their learning curve is so good that is easy to have a good prototype of database layer soon, and the resources of the community greatly assist.

7.0.3 Design

About the user interface

There are any that has been critical in the develop and is that the assumption as a good idea that all interface always is loaded when a user enters to the app, but after some tests have been seen that is not the best way. For this reason, many teams that work with Angular choose an alternative, use a library specially designed to load the javascript files of the app needed in each part of the interface.

So, if a user is in the teaching section don't need all files of the reports section, and this files only will be loaded when the user insert in this section. To do this, the teams use the Require.js[3] library as the standard dynamically loader and their use will be the next step to do the load of the app faster.

About the compression of APIs

Is easy to notice that the API of Students Control microService is very semantic but very big also (become more complex to maintain and change) while of the teaching service API is very compact and little but less semantic, because the meaning of their resources is not clear and therefore their behavior neither. It was made on purpose it to check in the development which approach was more problematic or simple to explain or update.

And the conclusion is that to maintain the coherency between the domain drive development of the service and the expressiveness of the API the customers do not need to know the internal work of the service, but need understand the logic behind of the items, so, in the future, it will be moved

[3]Literally from their website ,requirejs.org: *"RequireJS is a JavaScript file and module loader. It will improve the speed and quality of your code".*

to an approach more human readable (in spite of all disadvantages, talking about code and maintenance).

7.0.4 The develop process

Can be really interesting if you use any methodology as SCRUM and the developer's team work together. But when the team is a single developer and the project only have sporadic contributions, all is more difficult to follow. Other techniques can be used but SCRUM, eXPrograming or something like this not works fine to only one person.

So, independently, is a good point to start to practice. On other hand, is noticed that the sporadic contributions as in a hackathon[4] or in a simple day are difficult too because, as is normal, the people require a time to understand the project, the technology and all related with it.

As we comment before if we does not have a good overview of the project to long-term and we not foresee the growth of it, we can get a point where the code is too big or messy that to require a big refactoring. This would make us lose a lot of time of development (that was not planned) and even though is better detect this early, it will be a problem. An example of this was the point of the development where was necessary to do a refactoring. See chart bellow.

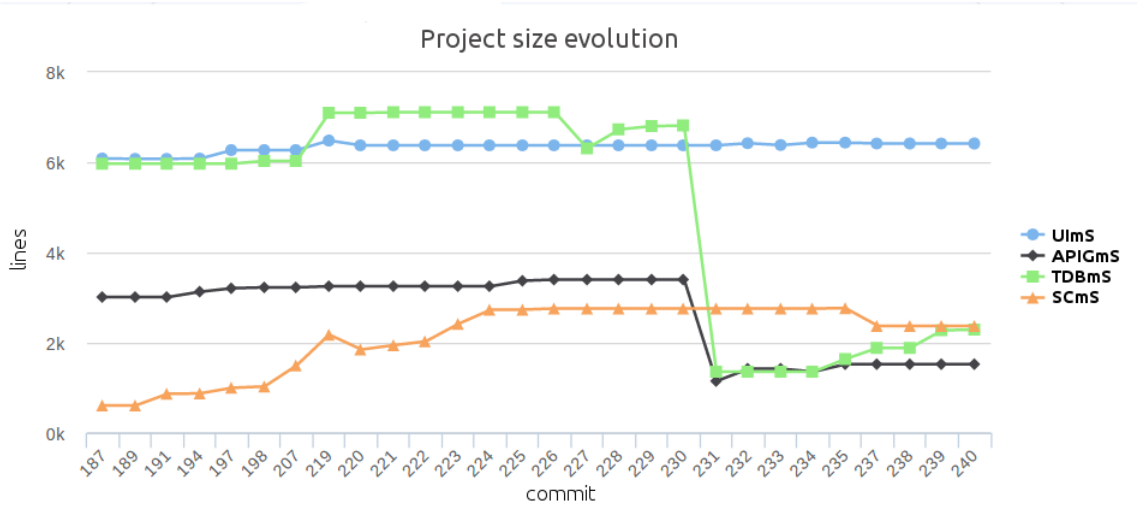


Figure 7.6: Project size in lines and refactoring effect.

As we can see, we had services that not stop to grow despite the logic was

[4]Is an event based on a sprint in which computer programmers and others are involved in the development of some software.

not very complex. It was because it was used some technologies that after was discarded. And after, the weight of the services was reduced drastically (the rest of services was a normal grow), to something more normal, having less code to maintain, cleaning and documenting, doing a lot more of things that before the refactoring. We would like to remind a great book about refactoring, Fowler [1999]).

7.0.5 Opportunities

Take part of some software contest is the better decision that any software student can take. Visibility, networking, new friendship are only some benefits that can achieve.

Thanks to enrolling this project of the contests that the Open Source Department of the University of Granada with *JJ Merelo* as principal organizes was offer a job in a related software company with the technologies and patterns used. So, if any people think that the participation, the contribution, and involvement is not useful, is absolutely wrong. In all cases, this attitude front the students only return benefits.

7.0.6 Future of the project

At the beginning of the develop, the idea behind of this was put in production the result in a few months in beta mode in a school center of Granada, but now, the jobs opportunities referred above have been done that the project go to the another plane, less important, because the ideas and the philosophy are developed just now with another really good engineers in a company, building a privative software, architecture, and new related tools.

So, independently the license of the code has not changed, and the develop can go ahead with any developer or group of them that want. For another hand the continuous evolution of the technologies do this issue a bit difficult, and actually, it is another learned lesson about the innovate software using three party technologies, we will never have the safety that the technology never will change. If you are working with C++ or even Java with you own infrastructure the changes are minimal through the months, with third party technologies and support you need be at day with all changes and update almost all your software each year. So, in this cases is difficult that the continuity of the project will be ensured, at least without the original designer inside the new developer's teams. But, anyway, is only a point of view, with the software nothing can be assumed.

Independently, the code is open, to learn, to review, and maybe to help someone, so for this part, we are happy.

Data processing

Finally the project has not grow as much as we would have liked, and this part is not enough powerful to their possibilities. The huge amount of data joined with their clear relations does this especially interesting to apply techniques as data mining or machine learning. The simple relations between data is easy to see but can be exists dozen of interesting relations that can go undetected and their work will be the mainly task in futures iterations.

7.0.7 Open Source

Other conclusions obtained in the solution development are related to open source and the viability to survive on this. Many time in the college is easy to hear that the open source is a good way to start and is true, but not if you want to work of this. Work in open source projects is really interesting, for the community, for the workflow, for build some useful to the community and by a huge list of advantages. But this is possible only when for one hand you are working in a company and some of the projects of this are decided to be open, independently of the reason, community, better visibility, etc. or when you are a student and have the opportunity to free amounts of code, as this case. But for another hand, thinking to build a company, more o less big based on an open source solution is very very rare and complex, mainly to younger and inexpert software developers.

Obviously in the most cases, always there are some exceptions that are wonderful examples that project with an amazing grow, and a really amount of code that any developer must have would be open, always open, because there are any developer that can learn alone, without the community (in any of their forms) and be in the obligation to contribute, to give back the favor.

7.0.8 Closing

This project has been a great opportunity to learn a lot about amount of things, but especially about myself, has been another opportunity to know how to deal with new challenges, how to work a first really subtle approach to project management and the most important, to know which my bigger faults. It have helped me to understand that this is only the begin, the beginning of the way that only can be covered if you do not stop to learn never, absolutely never.

So, **let's start!**

Appendix A

User Stories

This is the list of user stories related with the chapter 2, Requirements.

As user, I want to use the app in any device.

Aceptance criteria:

- Must be accesible from:
  - Smartphones
  - Tablets
  - Laptop
  - Standar Computers
- The item must be inserted, readed, updated and deleted.

GN #1 - High priority

As user, I want to load in the system data from another databases and formats.

Aceptance criteria:

- Must be compatible with formats:
  - XML
  - CSV
  - JSON

GN #2 - High priority

As user, I want to get the database of the app to migrate if was necesary.

Aceptance criteria:

- Must be compatible with formats:
  - XML
  - CSV
  - JSON

GN #3 - High priority

As manager, I want to save students to can to register them.
Aceptance criteria:

- The item must be this fields.
  - Name
  - Surname
  - DNI
  - Email
  - Address
  - Locality
  - Province
  - Birthdate
  - Phone
  - Image
  - Gender
- The item must be inserted, readed, updated and deleted.

AMS #1 - High priority

As manager, I want to save teachers to can to register them. Aceptance criteria:

- The item must be this fields.
  - Name
  - Surname
  - DNI
  - Email
  - Address
  - Locality
  - Province
  - Birthdate
  - Phone
  - Image
  - Gender
- The item must be inserted, readed, updated and deleted.

AMS #2 - High priority

As manager, I want to save subjects to can to register them.

Aceptance criteria:

- The item must be this fields.
  - Name
  - Description
- The item must be inserted, readed, updated and deleted.

AMS #3 - High priority

As manager, I want to save classes to can to can to register them.

Aceptance criteria:

- The item must be this fields.
  - Course
  - Word
  - Level
  - Description
- The item must be inserted, readed, updated and deleted.

AMS #4 - High priority

As manager, I want to save relation between subjects and classes.

Aceptance criteria:

- The item must be inserted, readed, updated and deleted.

AMS #5 - High priority

As manager, I want to save relation between teachers and the relation between subjects and classes.

Aceptance criteria:

- The item must be inserted, readed, updated and deleted.

AMS #6 - High priority

As manager, I want to save relation between students and the relation between subjects and classes.

Aceptance criteria:

- The item must be inserted, readed, updated and deleted.

AMS #7 - High priority

As teacher, I want to save attendance of students to can to register them.

Aceptance criteria:

- The item must be this fields.
  - Attendance
  - Delay
  - Justification
- The item must be inserted, readed, updated and deleted, in a list of all students of the class.

AC #1 - High priority

As teacher, I want to save attendance of students to can to register them.

Aceptance criteria:

- The item must be this fields.
  - Attendance
  - Delay
  - Justification
- The item must be inserted, readed, updated and deleted, in a list of all students of the class.

CC #1 - High priority

As teacher or educator, I want to record bad behaviour of a student.

Aceptance criteria:

- The item must be this fields.
  - Course
  - Word
  - Level
  - Description
- The item must be inserted, readed, updated and deleted.

MRKS #1 - High priority

As teacher or educator, I want to know most common disciplinary notes and data related.

Aceptance criteria:

- The item must be this fields.
  - Course
  - Word
  - Level
  - Description
- bla bla bla.

DN #1 - High priority

As teacher or educator, I want to record bad behaviour of a student.

Aceptance criteria:

- The item must be this fields.
  - Course
  - Word
  - Level
  - Description
- The item must be inserted, readed, updated and deleted.

DN #2 - High priority

As any person of center I want to have simple and advanced reports about the state of the student.

Aceptance criteria:

- The reports must be dynamics and useful in our domain.
- Should be able to export in some formats, as pdf or jpg.

SAR #1 - High priority

As a teacher, I would like not interact manually with the official system.

Aceptance criteria:

- The data must be uploaded automatically.
- The previous data must be downloaded and put in the app automatically.

SNC #1 - High priority

As a system, I need have all my services low coupled and be able to work independently.

Aceptance criteria:

- Some do not depend on others to run.
- If one fail the rest not, although they not working correctly.

MSR #1 - High priority

As a system, I need have all my services uses the same communication protocol.

Aceptance criteria:

- They must talk REST.
- They must implement a resource that retrieve their api specification.

MSR #2 - High priority

Appendix B

APIs definitions

These are the RAML 0.8 API specifications files for the project services. These do not implemented in any file, is only the formal way to give the behavior of the APIs between developers or customers. Each code block define an entire API or a segment of one.

B.1 Teaching Data Base microService API

```
#%RAML 0.8
title: Teaching Data Base API
version: 1.0
baseUri: ---

/test:
  description: For API testing purposes.
  get:
    description: Test the connection with the API.
    responses:
      200:
        description: Ok. The API is running.
      405:
        description: Method not allowed. When it is not running.

/test_mysql:
  description: For database testing purposes.
  get:
    description: Test the connection with the database through the
        API.
    responses:
      200:
        description: Ok. Database engine in running.

/entities/{kind}:
  description: Collection of kind type management resource.
  uriParameters:
     kind:
       description: Type of entity required.
       type: string
  get:
    description: Return all items by kind passed.
```

```
32      responses:
33        200:
34          description: Ok. A list of item will be returned.
35        404:
36          description: Not found. The kind of item does not exists.
37    post:
38      description: To save item in the database.
39      responses:
40        201:
41          description: Created. The item has been saved, no body will
                be returned.
42        400:
43          description: Bad request. Request have a syntax error.
44        404:
45          description: Not found. The kind of item does not exists.
46
47  /entities/{kind}/{entity_id}:
48    description: Item of specific kind management resource.
49    uriParameters:
50      kind:
51        description: Type of entity required.
52        type: string
53      entity_id:
54        description: Identifies of the item.
55        type: string
56    get:
57      description: Return an item.
58      responses:
59        200:
60          description: Ok. An item will be returned.
61        404:
62          description: Not found. The kind or item does not exists.
63    put:
64      description: To update item in the database.
65      responses:
66        204:
67          description: No content. Updated success without return.
68        400:
69          description: Bad requeest. Request have a syntax error.
70        404:
71          description: Not found. The kind or item does not exists.
72    delete:
73      description: To delete an item.
74      queryParameters:
75        action:
76          description: Use to specify some special action.
77                       Options:
78                        dd: delete all dependencies of the item
                              deleted.
79                              If you delete a student will be deleted
                                  all
80                              relations with the rest of item in the
                                  database
81                              of them.
82          type: string
83      respones:
84        204:
85          description: No content. Item has been deleted.
86        404:
87          description: Not found. The kind or item does not exists.
88
89
```

```
90    /entities/{kind}/{entity_id}/{optional_nested_kind}/{onk_entity_id}:
91      description: To delete relations between items. Only available for
            now
92                  between kind as class of subject and
                        optional_nested_kind as student.
93      uriParameters:
94        kind:
95          description: The kind of item to manage.
96          type: string
97        entity_id:
98          description: The id of the kind.
99          type: string
100       optional_nested_kind:
101         description: The nested kind (one related with first).
102         type: string
103       onk_entity_id:
104         description: Identifier of the nested kind.
105         type: string
106     delete:
107       description: Delete the relation between items.
108       respones:
109         204:
110           description: No content. Item has been deleted.
111         404:
112           description: Not found. The kind or item does not exists.
113
114   /entities/{kind}/{entity_id}/{related_kind}:
115     description: Manage subcollections of items of a kind.
116     uriParameters:
117       kind:
118         description: The kind of item to manage.
119         type: string
120       entity_id:
121         description: The id of the kind.
122         type: string
123       related_kind:
124         description: The related kind to get all items.
125         type: string
126     get:
127       description: Return a list of items of a related collection of
              another.
128       responses:
129         200:
130           description: Ok. An item will be returned.
131         404:
132           description: Not found. Some of ids do not exists.
133
134     /{rk_entity_id}/{subrelated_kind}:
135       description: To manage the relation nested with related kinds.
136       uriParameters:
137         rk_entity_id:
138           description: Related kind id
139           type: string
140         subrelated_kind: Subrelated kind id
141           description:
142           type: string
143       get:
144         description: To get all related items in nested related kind.
145         responses:
146           200:
147             description: Ok. An item will be returned.
148           404:
```

```
149             description: Not found. Some of ids do not exists.
150
151   /entities/{kind}/{entity_id}/report:
152     description: Retrieve the reports about items that have it.
153     uriParameters:
154       kind:
155         description: The kind of item to manage.
156         type: string
157       entity_id:
158         description: The id of the kind.
159         type: string
160     get:
161       description: To get a report from item.
162       responses:
163         200:
164           description: Ok. An report will be returned.
165         404:
166           description: Not found. Item does not exists.
```

B.2 Students Control microService API

B.2.1 Attendance Controls sub-API

```
1
2   #%RAML 0.8
3   title: Attendance Controls API
4   version: 1.0
5   baseUri: ---
6   /ac:
7     description: The resource to work with attendance controls
          database saved.
8     get:
9       description: Get a list of all attendance controls
10      responses:
11        200:
12          description: Ok. A list of items will be returned.
13          body:
14            application/json:
15              example: |
16              [
17                {
18                  "acId": 4785074604081152,
19                  "association": {
20                      "associationId": 13,
21                      "class": {
22                          "classId": 1,
23                          "course": 1,
24                          "level": "ESO",
25                          "word": "A"
26                      },
27                      "subject": {
28                          "name": "Science",
29                          "subjectId": 1
30                      }
31                  },
32                  "createdAt": "2017-03-12T13:20:23.906080",
33                  "createdBy": 1,
34                  "students": 2,
35                  "teacher": {
36                      "name": "Jhoan",
```

```
                        "profileImageUrl": "/imageservice/29372929.jpg",
                        "surname": "Mathew",
                        "teacherId": 1
                    }
                 }
              ]

  post:
    description: To save an item of "ac" kind into service.
    responses:
      201:
        description: Created without response. The item was added to database will not
                     returned body.
      body:
         application/json:
           example: |
           {
             "students": [
               {
                 "control": {
                   "delay": 0,
                   "assistance": true,
                   "uniform": true,
                   "justifiedDelay": true
                 },
                 "studentId": 113
               },
               {
                 "control": {
                   "delay": 0,
                   "assistance": true,
                   "uniform": true,
                   "justifiedDelay": true
                 },
                 "studentId": 213
               }
             ],
             "teacherId": 23,
             "association": {
               "associationId": 13,
               "classId": 24,
               "subjectId": 17
             }
           }

      400:
        description: Bad request. Request have a syntax error.
      404:
        description: Not found. The kind of item does not exists.

  /{ac_id}:
    uriParameters:
     ac_id:
       description: Attendance control identifier
       type: string
    get:
      description: Return an item of the attendance controls collection.
      responses:
        200:
          description: Ok. An item will be returned.
```

```
97            404:
98              description: Not found. The item required does not exists.
99        put:
100         description: Save a new attendance control in the database.
101         responses:
102           204:
103             description: No content. The server has fulfilled the
                    request but does not need to return an entity-body.
104           404:
105             description: Not found. The item required does not exists.
106
107       delete:
108         description: Resource to delete items of this type.
109         responses:
110           204: No content. The server has fulfilled the request but
                  does not need to return an entity-body.
111           404:
112             description: Not found. The item required does not exists.
113
114     /schema:
115       description: To know the schema of this kind of object.
116
117       get:
118         description: To get the schema of the type.
119         responses:
120           200:
121             description: Ok. Successful requests. An item will be
                    returned.
122           404:
123             description: Not found. The item required does not exists.
```

B.2.2 Disciplinary Notes sub-API

```
1
2   #%RAML 0.8
3   title: Disciplinary  Notes API
4   version: 1.0
5   baseUri: ---
6   /dn:
7     description: The resource to work with disciplinary notes database
           saved-
8     get:
9       description: Get a list of all disciplinary notes.
10      responses:
11        200:
12          description: Ok. Successful requests. An item list will be
                returned.
13          body:
14              application/json:
15                example: |
16                [
17                      {
18                          "classId": 3,
19                          "createdAt": "2017-03-13T11:12:23.846126",
20                          "createdBy": 1,
21                          "dateTime": "2000-12-03T10:30:00",
22                          "description": "A little problem with new
                                boy.",
23                          "disciplinaryNoteId": 5224879255191552,
24                          "gravity": 5,
25                          "kind": 3,
```

```
                            "student": {
                                "name": "Jhon",
                                "studentId": 1,
                                "surname": "adsf"
                            },
                            "subjectId": 21,
                            "teacher": {
                                "name": "Peter",
                                "profileImageUrl": "/imageservice
                                    /10072919.jpg",
                                "surname": "Smith",
                                "teacherId": 1
                            }
                        }
                ]

    post:
      description:
      responses:
        201:
          description: Created without response. The item was added to
               database will not
                        returned body.
          body:
            application/json:
              example: |
              {
                  "studentId": 5,
                  "teacherId": 42,
                  "classId": 3,
                  "subjectId": 21,
                  "dateTime": "2000-12-03 10:30",
                  "kind": 1,
                  "gravity": 5,
                  "description": "A little problem with new boy."
              }
        422:
          description: Unprocessable Entity. Probably because of the
              payload
                        sendes has not correct format.

    /{dn_id}:
      uriParameters:
       dn_id:
         displayName: Disciplinay Note ID
         type: integer
      get:
        description: Return an item of the disciplinary notes
            collection.
        responses:
          200:
            description: Ok. Successful requests. An item will be
                returned.
          404:
            description: Not found. The item required does not exists.
      put:
        description: Save a new disciplinary note in the database.
        responses:
          204: No content. The server has fulfilled the request but
              does not need to return an entity-body.
          404:
            description: Not found. The item required does not exists.
```

```
82
83      delete:
84        description:
85        responses:
86          204: No content. The server has fulfilled the request but
                does not need to return an entity-body.
87          404:
88            description: Not found. The item required does not exists.
89
90    /schema:
91      get:
92        description:
93        responses:
94          200:
95            description: Ok. The item schema will be returned.
96            body:
97                application/json:
98                  example: |
99                  {
100                     "gravities": [
101                         {"code": 1,"meaning": "mild"},
102                         {"code": 2,"meaning": "low"},
103                         {"code": 3,"meaning": "medium"},
104                         {"code": 4,"meaning": "high"}
105                     ],
106                     "kinds": [
107                         {"code": 1, "meaning": "Bullying"},
108                         {"code": 2, "meaning": "Disrespect"},
109                         {"code": 2,"meaning": "Gender violence"}
110                     ]
111                 }
112         404:
113           description: Not found. The item required does not exists.
```

B.2.3 Marks Subsection sub-API

```
1
2   #%RAML 0.8
3   title: Marks sub-API
4   version: 1.0
5   baseUri: ---
6   /mark:
7     description: The resource to work with marks database saved-
8     get:
9       description: Get a list of all marks
10      responses:
11        200:
12          description: Ok. A list of items will be returned.
13          body:
14              application/json:
15                example: |
16                  [
17                      {
18                          "createdAt": "2017-03-12T23:12:53.603021",
19                          "createdBy": 1,
20                          "enrollment": {
21                              "classId": 4,
22                              "enrollmentId": 42,
23                              "subjectId": 5,
24                              "teacherId": 8
25                          },
```

```
                        "markId": 6016527627190272,
                        "marks": {
                            "final": null,
                            "firstEv": 5,
                            "preFirstEv": 3,
                            "preSecondEv": 8,
                            "secondEv": 9,
                            "thirdEv": 10
                        },
                        "studentId": 5
                    }
                ]

  post:
    description:
    responses:
      201:
        description: Created without response. The item was added to database will not
                     returned body.
        body:
          application/json:
            example: |
              {
               "studentId": 5,
               "enrollment": {
                 "enrollmentId": 42,
                 "classId":4,
                 "subjectId": 5,
                 "teacherId": 8
               },
               "marks":{
                 "preFirstEv": 3,
                 "firstEv": 5,
                 "preSecondEv": 8,
                 "secondEv": 9,
                 "thirdEv": 10,
                 "final": 9
               }
              }
      422:
        description: Unprocessable Entity. Probably because of the payload
                     sendes has not correct format.

  /{mark_id}:
    uriParameters:
     mark_id:
       displayName: Mark ID
       type: integer
    get:
      description: Return an item of the marks collection.
      responses:
        200:
          description: Ok. Successful requests. An item will be returned.
        404:
          description: Not found. The item required does not exists.
    put:
      description: Save a new mark in the database.
      responses:
        204: No content. The server has fulfilled the request but
```

```
              does not need to return an entity-body.
          404:
            description: Not found. The item required does not exists.

    delete:
      description:
      responses:
        204: No content. The server has fulfilled the request but
            does not need to return an entity-body.
        404:
          description: Not found. The item required does not exists.

  /schema:
    get:
      description:
      responses:
        200:
          description: Ok. Successful requests. An item will be
              returned.
        404:
          description: Not found. The item required does not exists.
```

B.3 Analysis microService API

```
#%RAML 0.8
title: Analysis microService API
version: 1.0
baseUri: ---
ams/{kind}/{entity_id}/trend:
  description: Simple resource to get general reports.
  uriParameters:
    kind:
      description: Item type related with the reports.
      type: string
    entity_id:
      description: Item identifier.
      type: string
  get:
    description: Return the data block with the report of the item.
    responses:
      200:
        description: Ok. A report will be returned.
      404:
        description: Not found. The item or type required does not
            exists.

ams/{kind}/{entity_id}/{report_type}:
  description: To get other type of reports of more specific one.
  uriParameters:
    kind:
      description: Item type related with the reports.
      type: string
    entity_id:
      description: Item identifier.
      type: string
    report_type:
      description: The kind of report required.
      type: string
  get:
    description: Return the data block with the report of the item.
```

```
36        responses:
37          200:
38            description: Ok. A report will be returned.
39          404:
40            description: Not found. The item or type required does not
                  exists.
```

Appendix C

Data base model

Definition of the datamodel of Teaching Data Base micro Service SQL database. This definition is an extract of *DBCreator.sql* project file.

```
CREATE TABLE student (
  studentId       INT NOT NULL AUTO_INCREMENT,
  name            CHAR(100) NOT NULL,
  surname         CHAR(100) NOT NULL,
  dni             INT,
  email           CHAR(120),
  address         CHAR(100),
  locality        CHAR(50),
  province        CHAR(50),
  birthdate       DATE,
  phone           CHAR(50),
  profileImageUrl CHAR(200),
  gender          CHAR(1),

  #Metadata parameters
  createdBy       INT,
  createdAt       DATETIME,
  modifiedBy      INT,
  modifiedAt      DATETIME,
  deletedBy       INT,
  deletedAt       DATETIME,
  deleted         BOOL,

  PRIMARY KEY (studentId),
  UNIQUE (dni, deleted)
);
```

```
CREATE TABLE teacher (
  teacherId       INT NOT NULL AUTO_INCREMENT,
  name            CHAR(100) NOT NULL,
  surname         CHAR(100) NOT NULL,
  dni             INT,
  email           CHAR(120),
  address         CHAR(100),
  locality        CHAR(50),
  province        CHAR(50),
  birthdate       DATE,
  phone           CHAR(50),
```

```
    profileImageUrl CHAR(200),
    gender          CHAR(1)

    #Metadata parameters
    createdBy       INT,
    createdAt       DATETIME,
    modifiedBy      INT,
    modifiedAt      DATETIME,
    deletedBy       INT,
    deletedAt       DATETIME,
    deleted         BOOL,

    PRIMARY KEY (teacherId),
    UNIQUE (dni, deleted)
  );
```

```
  CREATE TABLE subject (
    subjectId   INT NOT NULL AUTO_INCREMENT,
    name        CHAR(100) NOT NULL,
    description CHAR(255),

    #Metadata parameters
    createdBy   INT,
    createdAt   DATETIME,
    modifiedBy  INT,
    modifiedAt  DATETIME,
    deletedBy   INT,
    deletedAt   DATETIME,
    deleted     BOOL,

    PRIMARY KEY (subjectId),
    UNIQUE (name, deleted)
  );
```

```
  CREATE TABLE class (
    classId    INT NOT NULL AUTO_INCREMENT,
    course     INT(1) NOT NULL,
    word       CHAR(10) NOT NULL,
    level      CHAR(20) NOT NULL,
    description CHAR(255),

    #Metadata attributes
    createdBy  INT,
    createdAt  DATETIME,
    modifiedBy INT,
    modifiedAt DATETIME,
    deletedBy  INT,
    deletedAt  DATETIME,
    deleted    BOOL,

    PRIMARY KEY (classId),
    UNIQUE (course, word, level, deleted)
  );
```

```
  CREATE TABLE association (
    associationId INT NOT NULL AUTO_INCREMENT,
    classId       INT,
    subjectId     INT,

```

```
    #Metadata parameters
    createdBy     INT,
    createdAt     DATETIME,
    modifiedBy    INT,
    modifiedAt    DATETIME,
    deletedBy     INT,
    deletedAt     DATETIME,
    deleted       BOOL,

    FOREIGN KEY (subjectId) REFERENCES subject (subjectId),
    FOREIGN KEY (classId) REFERENCES class (classId),
    PRIMARY KEY (associationId),
    UNIQUE (subjectId, classId, deleted)
  );
```

```
CREATE TABLE impart (
  impartId      INT NOT NULL AUTO_INCREMENT,
  associationId INT,
  teacherId     INT,

  #Metadata parameters
  createdBy     INT,
  createdAt     DATETIME,
  modifiedBy    INT,
  modifiedAt    DATETIME,
  deletedBy     INT,
  deletedAt     DATETIME,
  deleted       BOOL,

  FOREIGN KEY (associationId) REFERENCES association (associationId),
  FOREIGN KEY (teacherId) REFERENCES teacher (teacherId),
  PRIMARY KEY (impartId),
  UNIQUE (associationId, teacherId, deleted)
);
```

```
CREATE TABLE enrollment (
  enrollmentId  INT NOT NULL AUTO_INCREMENT,
  studentId     INT,
  associationId INT,

  #Metadata parameters
  createdBy     INT,
  createdAt     DATETIME,
  modifiedBy    INT,
  modifiedAt    DATETIME,
  deletedBy     INT,
  deletedAt     DATETIME,
  deleted       BOOL,

  FOREIGN KEY (associationId) REFERENCES association (associationId),
  FOREIGN KEY (studentId) REFERENCES student (studentId),
  PRIMARY KEY (enrollmentId),
  UNIQUE (studentId, associationId, deleted)
);
```

Bibliography

E. Rey V. Gallardo, A. Gaztelumendi. Paper consumption in la aunciata ikastetxea school, domiciliary rubis and advertising waste. `http://www.laanunciataikerketa.com/trabajos/odisea/meridies.pdf` [Accessed: 20 June 17].

James Lewis Martin Fowler. *Microservices - A definition of this new architectural term*. March 2014. `https://martinfowler.com/articles/microservices.html` [Accessed: 20 June 17].

Lucero del Alba. Data serialization comparison: Json, yaml, bson, messagepack. `https://www.sitepoint.com/data-serialization-comparison-json-yaml-bson-messagepack` [Accessed: 20 June 17].

Jeff Sutherland. *Scrum: The Art of Doing Twice the Work in Half the Time*. Crown Business, 2014.

Martin Fowler. *Refactoring - Improving the Design of Existing Code*. Addison-Wesley Professional., 1999.

Mike Cohn. *User Stories Applied*. Addison-Wesley Professional, 2004.

Sam Newman. *Building Microservices*. O'Reilly Media, 2015.

Sam Ruby Leonard Richardson, Mike Amundsen. *RESTful Web APIs*. O'Reilly Media, 2013.

Miguel Grinberg. *Flask Web Development*. O'Reilly Media, 2014.

Agile alliance. `https://www.agilealliance.org/` [Accessed: 20 June 17].

Microservices.io. `http//microservices.io` [Accessed: 20 June 17].

Chris Richardson. *Introduction to Microservices*, May 2015. `https://www.nginx.com/blog/introduction-to-microservices/` [Accessed: 20 June 17].

Butterfly Devs Team. *SMS Repository*, a. `https://github.com/ButterFlyDevs/StudentsManagementSystem`.

Butterfly Devs Team. *SMS Web*, b. `http://butterflydevs.github.io/StudentsManagementSystem/`.