COOKEASY: AN APPLICATION TO TEACH HOW TO COOK

MSc Project and Dissertation
Arnaud Bertrand

School of Mathematical and Computer Sciences

AUGUST 20, 2015
HERIOT-WATT UNIVERSITY
Supervisor: Peter J.B. King
Declaration

I, Arnaud Bertrand,

Confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Arnaud Bertrand

Date: 20 August 2015
Abstract
Desktop computer monopoly is over. Nowadays the part taken by other devices such as mobiles, tablets or smartwatch has grown and cannot be ignored anymore. This project aims to bring a web application from market analysis to production in order to have an overview of all the sub-processes. The application will be developed as a web and mobile application; in this project both will be considered as having the same importance and the same rights.

Two individuals were involved in this project; this report will not discuss about the overall project but only about the application architecture, the web application and the back-end services. The mobile application will be considered but its implementation details won’t be the main concern of this report.
Table of contents

Declaration ........................................................................................................................................... 1
Abstract ................................................................................................................................................ 2
1. Introduction ..................................................................................................................................... 6
2. Project objectives ............................................................................................................................. 7
   2.1. Group objectives ....................................................................................................................... 7
   2.2. Personal objectives .................................................................................................................... 7
3. Literature review ............................................................................................................................... 8
   3.1. Application architecture .......................................................................................................... 8
       3.1.1. Architecture overview ....................................................................................................... 8
       3.1.2. Web services ..................................................................................................................... 8
       3.1.3. SOAP vs REST ............................................................................................................... 9
       3.1.4. REST overview ................................................................................................................ 9
   3.2. Web application .......................................................................................................................... 10
       3.2.1. Design patterns ............................................................................................................... 10
       3.2.2. Choosing a language ....................................................................................................... 12
       3.2.3. Choosing a Framework .................................................................................................... 12
       3.2.4. Application design .......................................................................................................... 15
       3.2.5. Testing ............................................................................................................................ 16
   3.3. Mobile applications .................................................................................................................... 17
       3.3.1. Language choice .............................................................................................................. 17
       3.3.2. Platform .......................................................................................................................... 17
   3.4. Back-end Server ......................................................................................................................... 18
       3.4.1. Introduction ...................................................................................................................... 18
       3.4.2. Possible solutions ........................................................................................................... 18
       3.4.3. Short comparisons ......................................................................................................... 19
       3.4.4. Solution chosen .............................................................................................................. 19
   3.5. Hosting solutions ....................................................................................................................... 20
       3.5.1. Different solutions ........................................................................................................... 20
       3.5.2. Cloud platforms .............................................................................................................. 21
       3.5.3. Hosting services comparisons ......................................................................................... 22
4. Project management ......................................................................................................................... 24
   4.1. Respect the objectives .............................................................................................................. 24
   4.2. Documentation ......................................................................................................................... 25
4.3. Version control ............................................................................................................. 25

5. Technical environment ........................................................................................................... 26
  5.1 Overview ............................................................................................................................. 26
  5.2 Quick reminder on views, controller, services ................................................................. 26
  5.3. Resources .......................................................................................................................... 26
  5.4. Scopes .................................................................................................................................. 26
  5.5. Modules and directives ....................................................................................................... 27
  5.6. CSS best practices and LESS .......................................................................................... 27
  5.7. Ng-boilerplate and grunt ................................................................................................. 27
  5.8. Bower and Npm ............................................................................................................... 28
  5.9. Mongo DB and Mongoose ............................................................................................... 28

6. Design and implementation ....................................................................................................... 29
  6.1 Authentication ...................................................................................................................... 29
    6.1.1. Login ............................................................................................................................. 29
    6.1.2. Register ........................................................................................................................ 30
    6.1.3. Authentication implementation .................................................................................... 31
  6.2. Games ..................................................................................................................................... 32
    6.2.1. Overview ...................................................................................................................... 32
    6.2.2. Display a quiz ............................................................................................................... 32
    6.2.3. Search for a game .......................................................................................................... 36
    6.2.4. Create a game ............................................................................................................... 37
    6.2.5. Games server implementation ...................................................................................... 41
  6.3. Layout ..................................................................................................................................... 43
    6.3.1. Header ........................................................................................................................... 43
    6.3.2. Menu .................................................................................................................................. 45
  6.4. Profile ...................................................................................................................................... 46
    6.4.1. Overview ........................................................................................................................ 46
    6.4.2. View a profile ................................................................................................................ 46
    6.4.3. Edit a profile .................................................................................................................. 47
    6.4.4. Profile server implementation ....................................................................................... 49
  6.5. Recipes ................................................................................................................................... 51
    6.5.1. Display a recipe .............................................................................................................. 51
    6.5.2. Recipe search ................................................................................................................ 55
    6.5.3. Create a recipe .............................................................................................................. 56
6.5.4. Recipes server implementation

7. Testing
7.1. Overview
7.2. Unit test for the web application
7.3. Unit test for the back-end server

8. Results and Evaluation
8.1. Evaluation
8.1.1. Introduction
8.1.2. Evaluation feedbacks
8.1.3. Evaluation analysis
8.2. Results

9. Problems encountered
9.1. Code maintainability
9.1.1. Web application
9.1.2. Back-end server
9.2. Back-end server framework
9.3. Features implied by other
9.4. Group members unavailability

10. Future work
10.1. Enhancements
10.2. Features ideas
10.3. Overall project

11. Conclusion

References

Appendix 1: Screens
1. Authentication
2. Game
3. Layout
4. Profile
5. Recipe
1. Introduction
At Heriot-Watt University, computer science master students should dedicate their last semester to an MSc Project allowing the student to apply skills they learned on a specific project. This report is the fruit of this project. It retraces everything that has been accomplished along with a preliminary research of information and the future work.

The application that has been developed is a cooking application but rather than concentrating on the recipe content like every other application does, it has been decided to focus more on the user needs. There are already many cooking applications and website but none of them actually took care of teaching users how to cook.

Many people enjoy having good food, but an important amount of them are as well afraid by the complexity it implies. Even a person who could be qualified as a good cook can sometimes be discouraged by the difficulty of some recipes. If you ever looked at some recipes you probably can understand the following:

- Did you ever see recipes using ingredients you do not know and just skip it because you didn’t feel confident about it?
- Did you ever see recipes with difficult vocabulary and give up because you do not understand half of it?
- Did you ever see some recipes that you just skipped because they were not well written and is not clearly explained step by step what you had to do?

Cooking applications in general have still many improvements to make in order to target all users and not only experts. This project consisted in making this content available to anybody, from someone who never touched a knife till a star-studded chef. One of the main goals was also to ‘gamify’ the learning process so people can learn how to cook by having fun.

The first part of the report will quickly introduce the objectives and the context of the project. Following it, a literature review is presented as the result of pre-researches. After this there will be a short chapter talking about how the project was managed. The heart of the report will be the design and implementation but just before a short chapter on the technical environment will be present to be able to understand clearly the implementation part. The design and implementation chapter will explain how the functionalities of the application were designed and how they were implemented. A short chapter showing examples on how testing was undertook in the application will follow. The report will continue on results and evaluation of the application, in this part the goal is to put things in perspective and see what has been accomplished. Next chapter will talk about the problems encountered in this project, how they were handled and how they could be avoided the next time. The report will finish by a chapter on the future work and a conclusion.
2. **Project objectives**

2.1. **Group objectives**

The purpose of this project was to create an application that will teach people how to cook in a user-friendly way. The scope of this project was not limited to Dissertation Project but to the creation of a real application possibly leading to a more serious project.

Therefore the long term objective is to have a functional and marketable application on several platforms within one year. The application should be easy to use by the end users and implement some processes related to ‘gamification’ to make users more implied in their learning path.

The group short term objective was to have one functional application with the most important features on both website and mobile application such as Android or IOS within 3 months.

2.2. **Personal objectives**

In order to reach the group’s objectives, personals objectives were split into 2: design, implement and test the mobile application and create the overall architecture of the whole application + design, implement, test and evaluate the web application + implementation of the back end. The part that concerns us it the latter one. Furthermore providing a working environment that will allow other people to join the project at any time will be highly recommended.

Therefore, the short term objective is to implement this working environment, develop the web application along with the back-end server and deliver it into production within 3 months. Back-end services should be independent from the platform in order to be reusable.
3. Literature review

3.1. Application architecture

3.1.1. Architecture overview

In order to share data between a mobile application and a web application, a back-server has to be implemented. Sharing data is essential in our application in order to have user accounts, store their score or store recipes for example. Therefore an external database storing data was mandatory. It has been decided that the back-end server should be independent of the platform so each devices has access to the same methods and same services.

![Diagram of application architecture](image)

It could have been possible to use the same server for the web application and the web services but then it would have given more importance to the web application rather than the mobile one. The goal was really to be able to access the database with same functionalities from the web application and the mobile application.

3.1.2. Web services

Definition: A web service is a computational entity accessible over the Internet. (Preist, 2004)

A Web server implementing web services allow people or programs to require some services from it (Rouse, 2007). They are becoming widely used as nowadays those services can be made available to anybody. The main point is that any tier-application can make call and use web services of a server if they are granted the rights. Two main implementation of web services are currently leading the web: REST¹, explained later on and SOAP² a message protocol using HTTP and XML to communicate.

---

¹ REST: Representational State Transfer
² SOAP: Simple Object Access Protocol
3.1.3. SOAP vs REST

If you do not know REST and SOAP you can be little confused about what is going to be said next, but here the goal is not to go through details on SOAP and REST and how they work. Instead this part just explain why REST will be choose rather than SOAP.

The main reason is performance. In order to compare SOAP and REST based Web services Belqasmi, F. (2012) made a case study by implementing a multimedia conferencing and found that REST delays were 3 to 4 times less than SOAP and that it was due to SOAP messages’ processing. This is an unacceptable considerable difference.

The second reason is the simplicity to implement. SOAP requires much more effort to be implemented than REST. Currently REST is simply using HTTP methods and has less overhead. Also REST uses JSON\(^3\) format and SOAP uses XML\(^4\), JSON is preferable as it is widely used in JavaScript it requires less effort to transform the data.

A second study made by Pedro A. Castillo (2011) also approves the previous points by mentioning that SOAP is slower than REST, the fact that the learning curve is smaller on REST and that it is much simpler to implement.

3.1.4. REST overview

REpresentational State Transfer is an architecture running over HTTP. It allows network communication between two different end points. It is often used in mobile applications and social networking websites. Usually a device will send an HTTP request toward the server, which then processes the requests and sends back a response accordingly.

In REST you will often send data to an URL and get data back. A famous REST API is for example Twitter API which is one of the most famous Rest API on the web. \(\text{https://dev.twitter.com/rest/public}\) can be explored to see a deeper example. By using the methods they provide you, you will be able for example to post a message on Twitter without being on their actual website.

Concretely most of REST API implements CRUD methods, CRUD standing for Create, Read, Update and Delete. Those methods are usually corresponding to HTTP GET, POST, PUT and DELETE methods. In other words those methods allow modifying the server’s data through web services.

---

\(^3\) JSON: JavaScript Object Notation, a language to describe data

\(^4\) XML: EXtensible Markup Language, a language to describe data
3.2. Web application
3.2.1. Design patterns
3.2.1.1. Definition

A design pattern is a course of action taken in order to design an application. When coding for an application programmer is totally free to write code the way he like without following any standard.

However, it is very easy to understand that for the code maintainability it is best practice to follow standards for coding. As an example a person working in a group needs to transfer an application to a new developer group; any developer must be able to understand what that person did. This is the purpose of architectural design patterns. The second advantage of design pattern is that sooner or later when a developer is coding on his own he will face problems that other developer already faced in the past and solved by using design patterns.

3.2.1.2. MV* design patterns

**Model View Controller – Figure 2**

![Model View Controller Diagram](attachment:figure2.png)

Figure 2 (Chauhan, 2014)

MVC pattern divides the application in 3 distinct layers:

- Model concerns everything about the data, the way it is represented or the way it is processed.
- View concerns everything in touch with the interface, how to display the data on the screen and how the user can interact with it.
- Controller concerns all controls done on the user actions and data. It uses the model in order to retrieve or store the data information and then choose which information to send to the view in order to be rendered.

MVC pattern is probably the most used pattern nowadays.
Model View Presenter – Figure 3

MVP and MVC are nearly the same patterns; only Presenter changes instead of Controller.

The difference between a Presenter and a Controller is that the controller doesn’t interfere in the business logic. A Presenter will handle all the business logic dealing with user events and will only communicate with the model to retrieve, amend or insert data. In other terms the view is not rendering the data of the model but the data delivered by the presenter.

This pattern is mostly used when having complex applications where there are needs to reuse the presentation logic.

Model View ViewModel – Figure 4

ModelView separate View from the Model. It uses methods to control the data to be rendered in the view and verify its integrity. It also provides the view some function to render the view properly and properly formatted.

The main point of MVVM in comparison of MVP and MVC is that MVVM handle two way data-binding between View and ViewModel. To explain two way data-binding in other terms, it is the possibility to have the data in the View exactly same as the one available in the ViewModel at every moment. It is of great benefit to perform real-time checking on the data.
3.2.1.3. **MVC vs MVP vs MVVM**

In MVC, the View is at the top of the stack, followed by the Controller and the Model underneath. Then the View knows the Controller and the Controller knows the Model. When rendering the data, the View has direct access to the Model. Therefore it might bring security and performance costs depending on the complexity of the application.

In MVP, we’ll have a controller instead of a Presenter. The Presenter listens to events from both View and Model and manages its actions subsequently. In MVP there are no data binding like in MVVM and therefore Views should implement interfaces allowing Presenters to interact with them.

MVVM uses ViewModel that takes specific information from the model and passes it to the View. It implements some logic information to manipulate the Model through the View. MVVM can use data binding which allow the ViewModel to represent exactly the same data as the view at each instant. However this can imply performance cost, if it is simply copying the data then everything is fine; but it won’t be the same if for example there are some verification on volatile data (Osmani, 2014)

3.2.2. **Choosing a language**

The only languages the group share in common are Java and JavaScript.

Those are the reason why JavaScript was chosen over Java for those reasons:

- JavaScript allow an instantaneous view of code modifications in the browser. In Java deploying the server is quite slow depending on the machine you have.
- JavaScript has many different frameworks for different purposes, therefore an application using those frameworks will be more modular than a Java application.
- Java’s way of handling MV* patterns is less transparent.
- JavaScript functions can run on client side, which means less computation on the server. This implies saving money on network traffic.

3.2.3. **Choosing a Framework**

In order to choose a framework here are the points that were considered:

1. Community
2. Documentation
3. Cost
4. Easy installation
5. Easy testing

It happens that the main JavaScript frameworks at that time are Angular, Ember and Backbone. Therefore the following study will only concentrate on those.
3.2.3.1. Community

The user community is a very important point in choosing a framework. With a strong community, the chances to find an answer at a question are greater and developers to help other developers when get into some trouble are more likely to exist.

<table>
<thead>
<tr>
<th></th>
<th>Angular</th>
<th>Backbone</th>
<th>Ember</th>
</tr>
</thead>
<tbody>
<tr>
<td>Github starring</td>
<td>36k</td>
<td>21k</td>
<td>13k</td>
</tr>
<tr>
<td>Stackoverflow questions</td>
<td>83k</td>
<td>18k</td>
<td>14k</td>
</tr>
<tr>
<td>Twitter followers</td>
<td>61k</td>
<td>3.8k</td>
<td>24.2k</td>
</tr>
<tr>
<td>Github contributors</td>
<td>1,205</td>
<td>252</td>
<td>464</td>
</tr>
<tr>
<td>Youtube channel followers</td>
<td>67k</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1, Comparison of frameworks popularity through different criterions

The table 1 shows that the Angular community is far more important than the two other which are quite similar.

Another interesting thing to look as well is the interest over time with Google trends.

The comparison was made between “Angular + AngularJS + Angular.js”, “Backbone + BackboneJS + Backbone.js” and “Ember + EmberJS + Ember.js”. The result is in figure 5.

![Comparison of angular, backbone and ember over time from google trends](image)

Figure 5, Comparison of angular, backbone and ember over time from google trends

Once again angular popularity is clearly ahead of other frameworks and has kept on rising up since 2 years.
3.2.3.2. **Documentations**

Angular documentation: [https://docs.angularjs.org/guide](https://docs.angularjs.org/guide)


Backbonejs documentation: [http://backbonejs.org/#Model-unset](http://backbonejs.org/#Model-unset)

Three of them are quite complete and easy to read. However Backbone doesn’t have an easy searching tool comparing to ember and angular. On the other hand Angular documentation has a tendency to use Angular vocabulary, which makes things get a little complicate to understand.

The result here is that on documentation Ember takes the lead followed by Angular and Backbone.

3.2.3.3. **Cost**

As we are in a project we’ll not talk about money cost but about time costs.

To compare this time cost the comparison will be about the learning curve and the reusability.

On those 2 Angular get the upper hand for two reasons. The first reason is that I am personally familiar with angular whereas I have never used the 2 other ones. The second reason is that by using Ionic framework to deal with the mobile application, Ionic being based on Angular, the difficulties to face are similar.

3.2.3.4. **Easy installation**

Each of these frameworks are quite easy to install.

The learning curve might be harder on Ember and Angular but the functionalities of Backbone are limited and it might be useful to have another framework on top of Backbone.

It is a draw for this criterion.

3.2.3.5. **Easy testing**

They all could be tested without problems and there is no framework harder than another actually. However Angular and Ember provide a better documentation on testing than Backbone so on this point they both take the lead.

3.2.3.6. **Final choice**

After analysing all the different important points the result was that Angular might fit better to the needs of this project than other frameworks.

Ember could have been another option and it was possible to take a deeper look into it. However the huge community behind Angular and the time cost earned on this project make the decision lean towards Angular.

Therefore the Web application will be written using AngularJS.
3.2.4. Application design

3.2.4.1. Overview

The framework to use for the project was found but what about the application design?

Actually, Angular is related to some design patterns that were previously stated in this report.

Choosing between MVC, MVVM and MVP can be difficult and developers can spend time to select one arguing pro and cons. Angular knows that and tries to make a change in order to see developers concentrating on developing apps well designed than wasting time on a discussion that isn’t so important. Therefore Angular decided to consider itself as a MVW framework, W standing for ‘Whatever’ in the meaning of ‘Whatever works for you’. (Minar, 2012)

Now that things are clearer about the chosen design pattern, let’s take a look at how to implement the web application with that design pattern.

3.2.4.2. Design

![Diagram of Angular main design principles for web applications](image)

Firstly, Views in Angular are mainly used through templates. Templates can be considered as fragments of View. To stay simple it is like including HTML code inside another HTML page dynamically. Nowadays those templates are sent to the client at once and it doesn’t need to make another request to the server to display a different template, this is called single page applications. It allows the server to dispatch some of the workload on the client and therefore process less. Actions available to user on a View are usually triggering Controller functions.

Controllers in Angular are the core of the code; they link the data to the view and process the user actions. Controllers tell which View to display and what elements in the View should be displayed. They also make calls to Factories to deal with data management when required. The controller shouldn’t impact the data directly; it has to go through Factories to do it. Controllers and Views handle two way data binding which makes them slightly different from MVC normal Controllers and make them closer to a ViewModel of MVVM.
The Model main role is data storage, update and processing. In this part we will mainly have Factories that deal with JavaScript objects. Those objects have attributes that are the representation of your data and methods that are used to get or process the data. The data is often represented using JSON data and Collections. In Angular, Controllers call the model through Services, which are based on Factory design pattern in JavaScript. Factories are in charge of making calls to the REST Api in order to update or retrieve data from database through HTTP calls.

3.2.5. Testing
3.2.5.1. Unit testing

Why unit testing?
Unit testing is important in an application as it allows the developer to know that a specific part of the code is working properly on a technical point of view. When another developer joins and amends the code, unit testing helps to verify that the code will not break and that everything works correctly.

Angular is a very simple framework for implementing unit testing due to the fact it uses dependency injection. They are plenty of different frameworks for unit testing JavaScript. Angular made some recommendations in their documentation that were considered. However choices were also based on the current trends in order to have high support and concrete implementation examples.

Test runner, Karma
Angular team has developed a JavaScript test runner named Karma. Karma is a command line tool that watch for changes in the code and runs tests every time a file has been modified. If the unit test fail you will be warned by a sound and the code error will be displayed in the console instantly. It is a very convenient tool to use it and can run over different unit testing frameworks.

Test framework, Jasmine vs Mocha
Karma is just a test runner; therefore we still need a unit-test framework. There are a lot of different frameworks available nowadays and it is difficult to find an exhaustive comparison of those. In order to start the project the decision was to pick up Jasmine, which is a well-known framework compatible with Karma and recommended by Angular. However after discovering that the online material for Mocha was more relevant on internet and having more professional acquaintances using it the final decision went for Mocha. They are quite similar therefore the switch wasn’t difficult.

3.2.5.2. End-to-End testing
End-to-end testing is important as well. When an application becomes big it is difficult to test every piece of the code and this is when end-to-end testing takes place. Instead of concentrating on the code we rather concentrate on the functionality that the user might be able to use.

Due to lack of time and the application not being big enough yet, end-to-end testing hasn’t been implemented. However when the application will be big enough the choice will probably lean toward Protractor which has been developed by angular and is highly recommended by the angular community.

---

5 JavaScript Object Notation is a way to store information in an organized, easy-to-access manner. In a nutshell, it gives us a human-readable collection of data that we can access in a really logical manner. [URL]
3.3. Mobile applications

3.3.1. Language choice

The mobile application was not the main priority of this part of the project therefore it will be short.

The solution picked for the mobile development is called Ionic. Ionic is a NodeJS framework sitting on top of Apache Cordova and Angular.

![Ionic logo]

Apache Cordova is a platform allowing developers to create application for mobile in HTML, CSS and JavaScript. When you develop on Apache Cordova you can still access the devices functionalities such as camera or accelerometer. Angular on its own like we saw previously is a MV* framework that makes it very easy to layout your application and that use two way data-binding to manipulate data.

Ionic exploits those two frameworks and adds to it a simple way of creating HTML pages by adding components and functionality. Therefore it becomes very easy to create an application that can be deployed on the main platforms (e.g. Android, iOS, Windows Phone).

The main reason of choosing Ionic is because it is a simple way of developing application for several platforms with the same source code. The second main reason why the choice to use ionic was made is because it uses Angular. By using Angular in the web application and the mobile application then the JavaScript code might be similar along with the problems encountered. The functionalities of the web applications and the mobile application will be same for most of it. The only part of the code that is going to be totally different are the templates, therefore services and controllers will probably have a very similar structure.

3.3.2. Platform

Platforms that takes about 90% of the current market are Android and iOS therefore those are the main priorities.

The only way to develop an iOS application is by having a MAC computer and in order to deploy the application to the store a developer license which cost 99$/year is required. On the other hand for Android it is possible to develop on every OS and the developer license is only 25$ without time length restriction. It was not planned to invest in a MAC computer and for a developer license therefore Android was the best solution to start with. This is the longer term plan:

- Develop Android mobile app for phone
- Adapt Android mobile app for tablet
- Develop iOS mobile app for phone
- Adapt for iPad
- Develop for WindowsPhone
- …. Long future ...

However 3 month is a really short time and it was not possible to produce a correct application within that time. In order to pass to the next steps there is a real need for more developers.
3.4. Back-end Server

3.4.1. Introduction

The choice of a server independent of platform through a REST API was discussed previously. There are plenty of different languages available to build it and plenty of frameworks as well. As we started with JavaScript for the Web Application, the choice to was to continue in this way for the API. The first reason is that it might be confusing to deal with 2 different languages. When a problem occurs it happens often to have to look to both back-end and front-end. Changing the language can be confusing and it mean as well you probably have to change your IDEs, this is really inconvenient. The second reason is that NodeJS, the main platform for JavaScript servers is low level and therefore is really efficient. The REST API will probably receive lots of requests and by processing them efficiently the cost will be lowered. Therefore NodeJS provides us both time and costs saves.

3.4.2. Possible solutions

**Express / Restify**

Express is probably the most popular framework for NodeJS, very handy and easy to use. It is well commented and has a lot of information at your disposal due to its big community. Restify is a framework that borrows a lot from Express. It is specialized for writing REST API. Express includes a lot of functionality used in normal web applications (e.g. functionality to render the views). In a REST API those functionalities are not required the reason why Restify does not include it.

**LoopBack**

LoopBack is another framework built on top of Express and which goal is to provide API as well. It looks quite simple to use and make common tasks very easy but there is a lack of feedbacks on the web about it which makes it unsure about its limit. However there’s a little apprehension as well because it makes a lot of abstraction and therefore could bring a lack of maintainability.

**Hapi**

Hapi’s concerns are to spend less time building infrastructure. This is the reason why it provides very clear methods to create a REST Api. The other great point of Hapi is that it is used in many big projects and has been implemented by Walmart Labs. In that way it provides more confidence on the future of the framework.

**ActionHero**

This framework looks like a nice tool to use and it’s easy. However the community is quite low at the moment and it might be risky to work with it. Also, the documentation is nearly non-existing. Those are the reasons why we will not consider working with this framework.

**Others**

There are so many other frameworks but they mainly do not correspond to our requirements.

---

6 IDE: Integrated Development Environment, it is currently where a programmer writes his code. It provides help for writing code leading to a better productivity.
3.4.3. Short comparisons

After taking a look at Express, Restify, Hapi and LoopBack. Those different criterions were evaluated:

- Community (Github / Stack overflow)
- Documentation
- Ease of use
- Stability / Maintainability

Table 2 shows the different results

<table>
<thead>
<tr>
<th></th>
<th>Github</th>
<th>StackOverflow</th>
<th>Documentation</th>
<th>Ease of use</th>
<th>Stability</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Express</td>
<td>18k</td>
<td>14k</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>8/10</td>
</tr>
<tr>
<td>Hapi</td>
<td>3.8k</td>
<td>60</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>7/10</td>
</tr>
<tr>
<td>LoopBack</td>
<td>3.5k</td>
<td>107</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>7/10</td>
</tr>
<tr>
<td>Restify</td>
<td>3.2k</td>
<td>255</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>6/10</td>
</tr>
</tbody>
</table>

Table 2, Comparison of back end frameworks depending on our criterions

Consider that those marks have to be placed in a global context. For example Restify and LoopBack main purpose are REST Api but Express can be used to do normal web applications as well, therefore it has a much wider audience.

3.4.4. Solution chosen

The starting choice was difficult to make. Express and Restify were the obvious solutions has they have the larger community behind it. However LoopBack seems doing more by less coding which can be really great if done carefully. On the other hand Hapi provide very clear coding and in same time they are supported by important groups such as Walmart Labs.

For the last reasons evocated the choice was made to start with Hapi but keeping in mind that it was possible to change Express at any time if we encounter too many problems since I was more familiar and already experienced with that framework. And that’s exactly what happened, the time spent on configuring Hapi was too important and within the 2 first weeks the server switch from Hapi to Express and everything went much simpler and much better after it. It was important to switch at an early stage to high amount of refactoring.
3.5. Hosting solutions
Several options for hosting your application on internet are possible:

3.5.1. Different solutions
3.5.1.1. Dedicated server
A dedicated server is a server that belongs to the owner during the period he rent it. In other words it allows a person to own all the configuration of the machine like having a real machine. It is usually expensive because it is equivalent to renting a real computer and it needs to use the bandwidth and services of the renter such as security or back-up services. This solution is not viable for as it is not part of the plan to pay that much money at the beginning of a project.

3.5.1.2. Virtual servers
Virtual servers are a cheaper way to have a server. Instead of owning a whole machine like a dedicated server only a virtual machine running on a server will be given to the owner. It is less configurable than a dedicated server as because you do not have access to the material and the component of the computer. Performances are less efficient as well due to the fact that the computer share its capacity with other. However price is approximately half with same performances and the only inconvenience will be a much lower storage allowance on the hard drive. Also, it is easier to upgrade performance of a virtual server than a dedicated one. Nowadays virtual servers starts to be deployed on Cloud as well.

3.5.1.3. Cloud
Cloud computing is quite trendy lately but let’s make a short intro still. Cloud computing is a new technology allowing to run applications or store data for example on a bunch of computers that will act same as a server. In other words it allows to centralize software, data or anything else on Internet and it can be accessible from anywhere. Cloud servers are not hosted on a single physical machine but on several servers.

The power of cloud computing is that it is highly scalable, you’ll probably find all price ranges and it also propose much more services than virtual and dedicated servers. Reason why implemented this solution rather than the other ones. Another important point provided by cloud computing is the default tolerance. The application not being hosted on a single server, if one of them goes down our application will still run transparently whereas a shutdown on a virtual or dedicated server will entail the application to not be available.
3.5.2. Cloud platforms

As mentioned previously, Cloud computing allows the user to access the services anytime and anywhere. (Antonopoulos, 2010) in his book states that there 3 main delivery models:

- **SaaS**\(^7\): In this model the user has access to an application located on the cloud. However they do not control the material configuration such as the hardware, the OS or the network infrastructure.
- **PaaS**\(^8\): In this model users host an environment for their applications. They control the application but they do not control the material configuration.
- **IaaS**\(^9\): In this last model user can access fundamental computing resources such as CPU, memory, middleware and storage. However the user cannot control the infrastructure where resources rely on.

Different models and their implementation are shown in Figure 7.

![Figure 7 (Buyya, 2013)](image)

SaaS solutions are not interesting since this layer doesn’t host application but provide them. Therefore the choice had to be between PaaS solutions and IaaS solutions. The point that will make the difference is the level of the configuration to deal with on the server. On this point IaaS is highly configurable but it is not interesting at an early stage like this project. This means that the most suitable solution to us was PaaS which can host the application with the least efforts possible.

---

\(^7\) SaaS: Software as a Service
\(^8\) PaaS: Platform as a Service
\(^9\) IaaS: Infrastructure as a Service
3.5.3. Hosting services comparisons

3.5.3.1. Overview

In order to compare the different hosting services the different points were considered:

- Price: as cheap as possible for starting, free would be the best option.
- Ease of use: avoid time lost on configuring the hosting server
- Scalable / Performing: easy to adapt to application needs

They are so many cloud solutions on the web, after looking over 30 different solutions this short study will only present the most famous one and that target single developer, not only companies.

3.5.3.2. Pricing

To be honest, pricing will be the first and main criteria. As students it is not possible to afford huge investments like a company could do. Therefore this were the rules to pass the pricing tests:

- Not free under 3 months = failed, free under timing/tokens conditions = warning, otherwise passed. (>1 year is considered as free)
- Pricing cost elevate = failed, pricing cost moderate = warning, otherwise passed.
- If warning on both conditions then fail.

Results are shown in table 3:

Legend: Red = fail, Orange = Warning, Yellow = Passed

<table>
<thead>
<tr>
<th></th>
<th>Free period</th>
<th>Price / month</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppFog</td>
<td>1 month</td>
<td>Complicated but not too expensive</td>
</tr>
<tr>
<td>AWS</td>
<td>1 year</td>
<td></td>
</tr>
<tr>
<td>Pivotal</td>
<td>2 month</td>
<td></td>
</tr>
<tr>
<td>Digital Ocean</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>Dotcloud</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>Engine yard</td>
<td>6 month or 500h</td>
<td>36$ / cpu and 1gb ram</td>
</tr>
<tr>
<td>Heroku</td>
<td>free</td>
<td>35$ / cpu and 512 ram</td>
</tr>
<tr>
<td>Linode</td>
<td>1 month</td>
<td></td>
</tr>
<tr>
<td>Modulus</td>
<td>1 month</td>
<td></td>
</tr>
<tr>
<td>OpenShift</td>
<td>free</td>
<td>15$ / cpu and 512 ram</td>
</tr>
<tr>
<td>Windows Azure</td>
<td>free - 125 pounds</td>
<td>35$ / cpu and 1.75gb ram</td>
</tr>
</tbody>
</table>

Table 3, Price comparison of cloud hosting solutions

The 4 solutions remaining are:

- Amazon Web Services (AWS)
- Heroku
- OpenShift
- WindowsAzure
3.5.3.3 Easy of use

Amazon Web Services
Amazon Web Services requires a lot of configurations and offer too many options. For users who are aware of all technologies available then it might be a good solution. However at a student level it would be very difficult to manage all those configurations and even for the time loss it implies for the project this solution is not suitable. AWS is out of the list.

Heroku
Heroku’s amazingly simple to use. It uses command line and allows the user to deploy an application in few seconds. It also uses git repository, which is a version control system planned to be used to manage versioning of the application.

Openshift
OpenShift is very nice to use as well and complete but it implies a little more difficulty than Heroku as it needs to do more configuration whereas Heroku just act as a simple Git repository. The simplicity to launch the application is an important point regarding the time costs linked to it so here OpenShift lost some points.

Windows Azure
Windows azure isn’t looking too complicated but the only website documentation is written for Windows Visual Studio. It tells that by choosing this solution there’s a high probably to depend on windows technology and that is risky; it is important to be free to code on Windows, Linux, Mac or any other platform. Therefore Windows Azure’s out of the competition.

3.5.3.4 Scalable / Performing
Only OpenShift and Heroku are remaining.
Concerning the scalability they are both very scalable, they allow to adapt the customer needs automatically in a single command line. They are also flexible in the way the customer want to spend their money by choosing their own threshold and therefore limit their bills.

Regarding performance, a study has been lead on OpenShift, Heroku and CloudFoundry (Ataş, 2014). The study conclude that the three platforms were stable in terms of CPU, memory operations and have to improve stability in database operations but there was no real difference between the three.

3.5.3.5 Conclusion
Considering short-term and mid-term objectives Heroku was the most suitable solution. It offers a very simple mean to push the application on the cloud for free or at least free for a small application. By keeping an eye on the statistics and admitting the number of users greatly increase it will be possible to consider a migration on either a cheaper solution such as OpenShift or a more complete solution such as AWS but it will definitely not be in the project scope.
4. Project management

4.1. Respect the objectives

When starting a project one of the most difficulty things is to ensure that the objectives are realistic and timely specifics. Once those objectives are specified it is important to review those quite often and ensure the project is going into the right direction. For that purpose the group implemented several processes.

The first important point was a weekly meeting with the supervisor. In that weekly meeting was commonly mentioned the work accomplished during the week and which topic will be worked on during the following week. Doing this task every week helped us to always keep in sight the progress of the project and inform the supervisor as well. In the same time it was always a good occasion to ask the supervisor for advices when required.

The second process was to allow the member of the project a time to discuss about what has been accomplished and what was still required in order to fulfil the objectives. In a way it was a bit like the weekly meeting but on a much larger scale. It was a way to redefined time allocations to the different part of the project and to see whether the project was progressing well on the overall picture.

The last thing used in order to respect the objectives was Trello (Picture 1), a project management collaborative tool. Trello allows the user to create tasks in categories, assign them to members, archive them, comment them, give a due date … In other words it makes very easy to creates tasks, categorize them and follow their progress. It keeps tracks of everything that has been done and everything that still need to be done. It is also used to keep track of defects and evolutions that needs to be solved.

![Picture 1]
4.2. Documentation

When developing an application documentation is important to keep track of the accomplished work in details and to allow other developers to understand how it was done. Several type of documentation can be encountered.

First one is the functional specifications of the applications, it explains the different functionalities of the app, telling how they should look like and what are the interactions between the user and the application. Those functional specifications are shown a bit later in the report. They were written at the beginning of the project in order to help the upcoming documentation: the technical specifications.

Technical specifications consist in writing how the functionality will be implemented within the application. In other terms the purpose of the technical specifications is to give details on how the functionality will be integrated within what has been done previously and what will be developed next. This step is important to help the developer writing maintainable and homogeneous code. This part will also be shown later on in the report.

Next documentation that we had to write was the API documentation. This document state all the different services of the API. It tells which URL is associate with which method (e.g. POST, GET...), what action will this perform on the server and what object it will return. This documentation can be provided to anybody who wants to use the REST Api and therefore doesn’t need to be private.

Last one is related to the database. This document groups all database ‘documents’ together, along with their variable, their sub-documents and the different types associates to the variables. This is very helpful when working on the back-end server. It makes very simple to see how a document should look like in the database, how they should be checked and to quickly see which attributes of the document is required by the different services of the API.

4.3. Version control

In order to keep a record of all pieces of codes that were written for the application it has been decided that a version control tool will be used, the choice turned out to be Git because we were more familiar with it than any other one.

Version control is important, it allows to always have a record of the code which means in case of material failure the code is still available and there’s no need for starting from scratch again. The other important point is that it keeps every version committed, which means if very bad code has been committed and breaks the application it is always possible to go back to a previous version.

---

10 A document in NoSQL is how an object is stored in a Collection (If it needs to be compared to SQL, a collection is table and a document is a row)
5. Technical environment

5.1 Overview

This chapter is essential to understand the technical context evoked later on in this report. Do not hesitate to come back and read it again if something in the following part of the report is not clear.

5.2 Quick reminder on views, controller, services

Since it has been already explained in the literature review previously this part is just a reminder.

A template in Angular is a piece of HTML code that can be included in other part of the HTML code. They access variables and functions of the controllers through the DOM element attributes usually prefixed by `ng-`, e.g. `ng-click="plop()"` will call the function plop of the controller when the DOM element is clicked.

A controller in Angular is in charge of controlling the data of the application, they are the entry point of the business logic in the application. They hold the methods and variables that can be called or rendered by the view and can act onto the services to persist or retrieve the data.

A service in Angular is a singleton\(^{11}\) that will be called by controllers or other services. Its only purpose is to fetch and manipulate the data, mostly from or toward the server.

5.3 Resources

Angular resources were created especially for REST Api. In order to make HTTP calls in Angular the normal way to do is to use the $http service provided by Angular. However it can be very repetitive to create services for REST Api, in order to implement CRUD\(^{12}\) operation for example you need 4 different $http calls and specify every time the arguments, the type of the request...

Resources wraps the $http service to provider a very simple way to make CRUD operation in a single line of code. It also allows to retrieve an object as a resource and act directly onto it. For example it is possible to GET a resource, change its name and SAVE it. Resources in angular make all this transparent; doing the same thing through a service with $http would have at least implemented 2 different functions.

5.4 Scopes

Previously it was mentioned that DOM elements bind data and functions from the controllers. In order to know which controllers is associated to which part of HTML document Angular uses scopes. A scope could be described as a given context to a part of the template.

It is possible to have scope including other scope. If scope A includes scope B then A is called parent scope of B. If we are in the B scope we can call all functions (or bind data) from B and A however B will have the priority. Otherwise if we are in the scope A but not B then functions and data of B will not be available but the ones of A will.

Scopes in Angular are very important and needs to be well understood. The root scope is the higher scope and is available within the whole application; all other scopes are children of the root scope.

\(^{11}\) A Singleton is an object that cannot be instantiated more than once

\(^{12}\) Create, Read, Update, Delete
5.5. Modules and directives

Angular is modular, it allows the creation and use of pluggable modules, which is very useful for reusability, maintainability and testing. Create modularity in the code is important to delimit each functionalities. It is easier to find where the code is located and it makes very easy to implement unit testing on it. Also it is possible to use modules that another person develop by plugging it into the application and use its methods transparently.

In Angular this modularity goes until the DOM elements. Some markers for those DOM elements allow to transform or attach a specified behaviour to it, e.g. ‘ng-click’ will give a function on click on an element; this is called Directives. Angular allows users to create their own directive. They are very useful when you have a piece of html code that is a bit modular but has the same structure or design. Directives can be linked to a template for the rendering and a controller to handle its behaviour.

5.6. CSS best practices and LESS

For a large application it is really important to avoid class duplications since this can lead to interferences and cause the code to not be maintainable. In this purpose CSS selectors should be specific enough to not interfere with each other but also not being too specific to avoid performance loss.

LESS is a CSS framework that has been included in the project in order to manage the CSS code. Firstly it allows to create hierarchical code within CSS, which means it make the code cleaner. Secondly it also create reusable functions to avoid repetition of the same functionality. For example in order to make a round border available in all navigators the correct way is to write:

```html
-webkit-border-radius: 15px;
-moz-border-radius: 15px;
-border-radius: 15px;
```

With LESS it is possible to create a function that will be called within a single line of code and that will handle it.

LESS is very useful to manage the CSS code however it is important to not abuse it. Creating hierarchical code can be dangerous if it is misused. Every time a class is added to the hierarchy the CSS selector becomes longer and decrease performances; it is important to not overuse it.

5.7. Ng-boilerplate and grunt

In order to kick-off and manage an Angular application there’s many possible solutions, boilerplates or generator are usually the common way. Boilerplates consist of being a structure of folder and features for an application that has been implemented and that you can use directly to kick-off your project. On the other hand generators provide more features, e.g. easy way to create new functionalities in command line.

The decision was made to use a boilerplate in order to have more control on the creation of functionalities, meaning a better code maintainability. Ng-boilerplate, the framework used for this project organize the folder structure in a very similar way as the current implementation, it also already implement LESS, the CSS framework discussed previously and grunt, a building system enhancing productivity.
Grunt is a JavaScript task runner, its purpose is to handle tasks automation, it can perform repetitive task in simple command line. For example it is possible to create a grunt task that will minify the code\(^{13}\), compile the code and test it by just writing “grunt build” or something likely in the command line.

5.8. Bower and Npm

Bower and Node Package Manager are both package/modules manager. Npm is commonly used with NodeJS application whereas Bower concentrate on the front-end. The difference is that npm will have nested dependency, which means it can duplicate some source and therefore size heavily. On the other hand Bower is flat and will let the user manage dependencies. The problem caused by the flat dependencies is that it can bring some conflicts between versions and it is likely that they have to be resolved by the user.

In this project Bower will be used for the angular application and npm for the back-end server.

5.9. Mongo DB and Mongoose

Mongo DB is a NoSQL\(^{14}\) database using documents as storing strategy. Mongo DB communicates with NodeJS through a driver allowing it to perform actions on the database. Since the database used for the project is not currently of a small size no shards\(^{15}\) nor replica sets\(^{16}\) are implemented.

In order to use Mongo DB on NodeJS the back-end server uses an ODM\(^{17}\) called Mongoose. The advantage to use Mongoose over relying on NodeJS drivers for MongoDB directly is that Mongoose will help to structure and validate the data. Mongoose uses Schemas which can be compared to a definition of a data object with its attributes, its functions etc…

Those schema are used to validate the data transparently when inserting, modifying or retrieving information from or to the database. It is also possible to add personalized validation methods in case the original ones are not enough. The last advantage of Mongoose are pre/post hooks functions, which will be executed before or after modifications to a document in a database.

---

\(^{13}\) Minify the code take the original code and rewrite it in the minimal size possible. It is used to have smaller files to exchange between client/server.

\(^{14}\) No Only SQL

\(^{15}\) Shards are used to scale horizontally a database. They allow to split the data between different replica sets.

\(^{16}\) Replica sets are used to replicate the data on different server, then if one goes down another server can handle request instead.

\(^{17}\) Object Data Manager, similar to ORM in a relational database.
6. Design and implementation

6.1. Authentication

6.1.1. Login

6.1.1.1. Functional specifications

Path: /login

**Description**

The login page will only have 2 inputs. First the user ID which could be the username or the e-mail address of the user, secondly the user password. At the bottom a button login will allow the user to submit his credentials. A link under the login will permit the user to sign up if he doesn’t have an account.

**Events:** On click on the login button the server will either respond with a success or an error. If the login is successful then the user will be logged in and redirected to the main page. In case of error the application will display an error message just over the login button in red and italic. On click on the ‘Need an account’ link the user will be redirected to the ‘Register’ page.

6.1.1.2. Implementation

The main controller for the Login page is called LoginCtrl, this controller will bind data and methods with the template linked to it. In order to know beforehand if the user is connected and store the logged user after authentication, the controller will inject the ceAuthentication service. The controller needs as well the $http service of Angular to make a login request to the REST Api.
6.1.2. Register

6.1.2.1. Functional specifications

Path: /register

Description

The register page will have 3 inputs. Firstly the username which is the ID that will be used within the application to refer to the user, it is required to be 2 characters minimum. Secondly, the e-mail of the user required to be formatted like an e-mail address. Lastly, a password containing at least 6 characters and that will be used for the user authentication. Before registering the user will have to accept terms and conditions by clicking a checkbox.

Events: On click on the register button the server will either respond with a success or an error. If the registration is successful then the user will be logged in and redirected to the main page. In case of error the application will display an error message just over the register button in red and italic. For each input a message will be displayed to inform the user when the data is not valid. The register button will be enabled only when all the data is valid.

6.1.2.2. Implementation

The main controller for the Register page is called RegisterCtrl, this controller will bind data and methods with the template linked to it. In order to know beforehand if the user is connected and store the logged user after the register process, the controller will inject the ceAuthentication service. The controller will also inject the User resources to be able to create a new User at the end of the process.
6.1.3. Authentication implementation

The implementation of the authentication is quite complicated since the back-end server is a REST API. REST services rely on HTTP messages which are stateless, they are independent from each other. For this reason the implementation of authentication by Token is necessary to avoid sending user’s credentials at each request.

On the Web application the login controller will make an http request with the credentials to the API. The API will check if the password and user ID are valid. If they are valid it will then generate a token, store it into a Redis Database, a fast database for caching. Once this token is saved the server will send it back through an HTTP response to the client. Finally the client store it into its own local storage.

In order to be authenticated an HTTP ‘Authorization’ header needs to be specified for each request along with its token. When the server receive the request it will read that header to find about the connected user, it will look in the Redis database the user linked with the token and will get back the user.

In Angular this token can be passed automatically on each request by implementing an HTTP Interceptor. Interceptors in Angular allows to intercept requests before they are send, modify them and send it afterwards. In that case we intercept any HTTP request, add an Authorization header with the token if we have one and send it to the server after it.
6.2. Games
6.2.1. Overview
Games in the application should be a fun way to learn and enhance the user’s knowledge on food culture. For this part 3 main screens will be available: Search for a game, Display a game and Create a game.

In the first version of the application only quizzes will be available. In a future version it is highly possible that other types of games will be implemented. Therefore the conception should consider that a quiz is a category of game.

6.2.2. Display a quiz
6.2.2.1. Functional specifications
Path: /game/display/quiz/:id, with id being the id of the quiz

In order to display quizzes 3 different screens are required. The two first screens concern the display of a question, whether the answers types are texts or images the rendering will be different. The last one is a quiz completion screen where the user will find information on how well he performed.

At the top of the 2 first screens will be displayed the title of the quiz, the author and the current score for the game.

Events: At each answer given whether correct or false the score will be updated as follow:
- Single answer question: True + 5 points, False -3 points
- Multi answer question: True + 10 points, False -3 points

6.2.2.1.1. Text answers
Description

Text answers start by a question which is followed by a list of answer (from 2 to 4). If the question has only one answer then the answer will be selected with a direct click on a radio button. Otherwise if the answer has multiple correct answer then the user will have to tick checkboxes and click on a submit button under the list.

Events: When submitting an answer the score will be calculated and refreshed on the page. If the answer is correct then go on to the next page, otherwise stay on the same page. If a single choice answer is incorrect the user should see that the response was incorrect by making the button darker.

6.2.2.1.2. Images answers

Description

Image answers start by a question which is followed by a list of answer (from 2 to 4). If the question has only one answer then the answer will be selected with a direct click on an image. Otherwise if the answer has multiple correct answers then the user will have select answer by clicking them and submit them with a button under the list. Images can have an optional text to describe the picture.

Events: When submitting an answer the score will be calculated and refreshed on the page. If the answer is correct then go on to the next page, otherwise stay on the same page. If a single choice answer is incorrect the user should see that the response was incorrect by changing the colour of the image.
Description

At the end of the quiz will be display a screen where the user can review his statistics. The page will start by a message saying that the quiz is finish and a possibility to like that quiz if the user enjoyed it. Below that statistics will be displayed as a list; statistics could be for example the final score on this quiz, the number of questions answered and the number of incorrect answers.

Under statistics will be:

- If a user is connect: the new total score of the user
- Otherwise: a message to tell the user to sign up to keep his score

A retry button to retake the quiz as well as a button to return to the quiz list will be available at the bottom of the page.

Events: On click on retry the application reload the quiz. On click on other quiz the user is redirected to the quiz list.
The main controller for the quiz display is called GameDisplayQuizCtrl, this controller will bind data and methods with the template linked to it. In order to display the quiz correctly the module have to preload the quiz and inject it through quizPre in the controller. The controller will also attach the $http service of angular in order to make call to the rest API when a user finish a quiz to send its score or to send a request when the user like the game.

The controller will implement different functions:

- Validation functions: check the answer, update the score, on click answer (different if multi).
- Answers types: check if the question is multi answer, check the type of the questions.
- Navigation functions: change page, retry quiz and search another game.
6.2.3. Search for a game

6.2.3.1. Functional specifications

Path: /game/search

**Description**

This purpose of this page is to easily find games; therefore at the top of the page the user will find a search input along with filters and sorting tools. Just under will be displayed games with their images if they have one, their title, their number of like and their author. Games will be displayed like a table with 2 columns of same width.

**Events:** When typing in the input bar the game list will automatically be updated according to the input. The same effect is applied when the filter is changed or when using the sorting tool. On mouse over games will scale a bit bigger in order to help the user to see clearly which one he is choosing. When clicking on the game element the user will be redirected on the game page of the game that he clicked.

6.2.3.2. Implementation

The main controller for the quiz research is called GameSearchCtrl, this controller will bind data and methods with the template linked to it. In order to display some pre-loaded games when the user is not searching anything the controller needs to inject gamesPre containing the recent trends that were retrieved from the module beforehand. The controller will also inject the games resources in order to makes request to the server for researching other games. The controller will only implement 2 functions; one to search games and another to display a game that has been selected.
6.2.4. Create a game

6.2.4.1. Functional specifications

Path: /game/create/quiz

In order to create a quiz 3 different screens are required. The first screen concerns the quiz information and the others regards the creation of questions, whether the answers types are texts or images then the display will be different.

6.2.4.1.1. Information page

![Diagram of create a quiz form]

**Description**

On the first page the user will have to set a title for the quiz in an input located at the top of the screen. This title is mandatory and should be at least 10 characters longs. Just below the user will have the opportunity to set a main picture for the quiz but this is not mandatory. Once the title is set correctly the user will be able to pass on the next with a button ‘Create questions’.

**Events:** Clicking on create questions button will redirect to the screens for creating questions.

6.2.4.1.2. Questions overview

When the user will create questions a navigation line will be displayed at the top of the screen where the user will be able to go through the questions. At the top of both screens the user will find an input field in order to type the question title. Just under this question title is located the answer type, depending on what is selected between text or images the screen will be rendered differently.

At the bottom of each screen is located 2 buttons ‘Next question’ and ‘Finish’ that allows the user to create more questions or finish the quiz.
6.2.4.1.3. Create text question

Description

The basic type of questions are text questions. The user will insert new answers to the question through an input field located under the answer type. Once the user typed the answer he has to submit it through Correct or Incorrect buttons in order to tell whether the answer is correct or not.

Every time an answer will be submitted the answer will appear in a list of answer just over the add answer input. In this list the user can see the answer text and if the answer is correct or no, if a mistake is made then the user can remove the answer with a removing button.

Conditions of validations are minimum 2 answers, maximum 4 answers and minimum 1 correct answer.

Events: On correct of the ‘correct’ button the current answer in the input will be added to the list of answer as a correct answer. Likewise for the incorrect button. On click of the ‘remove’ button on an answer in the list it removes the answer from the list. When changing answer type to images, answers are reset and go to create images question screen. On click on the ‘next question’ button, if the question is not valid show an error message just over it otherwise create a new question. On click of the ‘finish’ button, if the question is not valid show an error message just over it otherwise finish the quiz and redirect the user to the quiz.
6.2.4.1.4. Create image question

Description

The second type of questions are images questions. The user will insert pictures through a click/drop area located under the answer type. The user could also type a text to accompany the picture. The answer can be submitted through Correct or Incorrect buttons in order to tell whether the answer is correct or not.

Every time an answer will be submitted the answer will appear in a list of answer just over the add answer input as a 4x4 table. In this table the user can see the every submitted answers, if a mistake is made then the user can remove the answer with a removing button.

Conditions of validations are minimum 2 answers, maximum 4 answers and minimum 1 correct answer.
**Events:** On click of the ‘correct’ button the current answer will be added into the table of answers as a correct answer. Likewise for the ‘incorrect’ button. On click of the ‘remove’ button on an answer in the list it removes the answer from the list. When changing answer type to text, answers are reset and the application will go to create text question screen. On click on ‘next question’ button if the question is not valid show an error message just over it otherwise it creates a new question. On click on the ‘finish’ button, if the question is not valid show an error message just over it otherwise finish the quiz and redirect the user to the quiz.

6.2.4.2. **Implementation**

The main controller for the quiz creation is called GameCreateQuizCtrl, this controller will bind data and methods with the template linked to it. In order to create the quiz at the end, the controller needs to inject the resources ceGamesQuizzes therefore when the quiz needs to be saved it will go through this resource.

The template will implement 2 different directives, the first one being the step-line directive which is used to navigate through the questions and the second is the picture-upload directive used several times to add pictures for answers or the game principal image.

The controller will implement different functions:

- Navigation functions: go to question X, next question, finish quiz.
- Questions functions: add/delete answer, add/delete question and reset answer.
- Functions to handle picture upload: main picture of the game, picture for answers...
6.2.5. Games server implementation

6.2.5.1. Mongoose Schemas

In order to implement several kind of games all games will be built around an abstract object “Game”:

<table>
<thead>
<tr>
<th>Game</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>User Schema*</td>
<td>Author of the quiz</td>
</tr>
<tr>
<td>createdOn</td>
<td>Date</td>
<td>Date of creation</td>
</tr>
<tr>
<td>picture</td>
<td>Picture Schema*</td>
<td>Main picture</td>
</tr>
<tr>
<td>title</td>
<td>String</td>
<td>Date of birth</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>Type of game</td>
</tr>
<tr>
<td>updatedOn</td>
<td>Date</td>
<td>Date of update</td>
</tr>
</tbody>
</table>

For the moment only quiz are implemented, Quiz are extending the Game object by adding one attribute to the document:

<table>
<thead>
<tr>
<th>Quiz</th>
</tr>
</thead>
<tbody>
<tr>
<td>questions</td>
</tr>
</tbody>
</table>

Questions sub-schema:

<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>answers</td>
</tr>
<tr>
<td>picture</td>
</tr>
<tr>
<td>title</td>
</tr>
<tr>
<td>type</td>
</tr>
</tbody>
</table>

Answer sub-schema:

<table>
<thead>
<tr>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
</tr>
<tr>
<td>picture</td>
</tr>
<tr>
<td>correct</td>
</tr>
</tbody>
</table>

A Quiz document in the database will be a document that will combined Quiz (with its sub-schemas) and Game documents. This give more flexibility to implement other kind of games. It is possible to do it that way since Mongo DB is a document based database and it does not requires all documents to have the same structure. However Mongoose validation can help to validate each kind of games by creating personalized validations according to the attribute type of the game.
6.2.5.2. API Methods

6.2.5.2.1. External services

**Search for games** – GET /games

Sending a GET request without any attributes at ‘/games’ will return the list of the most common games lately. It is possible to add a query parameters “match” along with a text in order to filter games matching the different words in the text. Other query parameters to filter games can be added in the future.

**Get a game** – GET /games/:id

Sending a GET request at /games/:id with :id being the id of the game wanted will return the entire game object of the database.

**Create a game** – POST /games/quizzes, need authentication

Sending a POST request at /games/quizzes along with data matching the Mongoose Schemas will result in the creation of a game in the database. The author of the game will have its Recent Activities updated with a message tell that he has created a game.

**Like a game** – PUT /games/:id/like, need authentication

Sending a PUT request at /games/:id/like with :id being the id of the game will result in increasing the number of like of that game and also add a message on the Recent Activities of the user.

6.2.5.2.2. Internal functions

In order to match the external services of the API with the Mongoose Schemas some internal functions need to be implemented:

- **Get**: Search and return a game from its _id attribute
- **Like**: Add the username of User to the likes set\(^{18}\) and set a new Activity for the User.
- **List**: Search for recent games sorted by number of likes and filter match on name attribute
- **Remove**: Remove a game from its _id attribute
- **QuizHandler – Create**: Create a quiz if Schema is conform and set a new Activity for the User.

\(^{18}\) Set is a list were elements cannot be repeated, in that case it means a user cannot like a game twice.
6.3. Layout
6.3.1. Header
6.3.1.1. Functional specifications
6.3.1.1.1. Overview

Header will have the logo with the title of the application on the left of the screen. On the right will be situated the fast menu access. This menu will change whether a user logged it or not. However in any case this menu will have at least a short access to games and recipes. Those short access are buttons that will be a bit darker than the background; the active button will be even darker. For example if you are either in game search, game display or game create then the button for Games will be darker since it is ‘active’.

Events: On click of the logo on the left return to the home page. On click of the game button go to the game search page. On click of the recipe button go to recipe search page. Every time a page change check for changing the active button.

6.3.1.2. Header without user authenticated

If the user is not logged in then 2 more short access will be added one for login and another for sign up.

Events: On click on login will redirect to login page. On click on sign up will redirect to sign up page.
6.3.1.1.3. Header with authenticated user

Description

If the user is logged in then a round circle with his profile picture will appear. An arrow directed to the bottom will be just next to the picture and will toggle a little menu, when the menu is open the arrow will switch and turn up instead. In the small menu two icons will appear: one to go to the user profile and another one to logout.

Events: On click of the picture or the arrow next to it toggle the little menu and turn the arrow down. On click again hide the menu and turn the arrow up. On click of the profile icon go to the profile of the user. On click of the logout icon disconnect the user from the application.

6.3.1.2. Implementation

The index page will include a header directive. This directive will include a template and a controller ceHeaderCtrl. In order to watch for state changes the controller will need to inject the rootScope Angular service. The only function in the controller will be to listen for state changes in the rootScope, check the destination page (e.g. games or recipes...) and set the active button accordingly.
6.3.2. Menu

6.3.2.1. Functional specifications

Description

The menu can be shown and hidden by a little arrow on the left. When the menu is hidden the arrow will point to the right and when it is open it will point toward the left. When opening the menu the whole page will slide to the right in order to give some space to the menu. The menu will contain a link to each screen available in the application.

Events: On click on the arrow when menu is closed the menu will slide pushing the screen to the right, the arrow will spin and turn left. Clicking on the arrow again will be likewise but in the opposite way. On click on a menu item the application will go to the page selected. If the page requires authentication then it will check for authentication beforehand; if no users are connected it will redirect to the login page.

6.3.2.2. Implementation

The index page will include a menu left directive. This directive will include a template and a controller MenuLeftCtrl. In order to move the whole page the controller needs to inject the rootScope. The controller needs also the ceAuthentication services in order to check if the user is connected or not before changing to a new page. The controller functions only check if authentications are required and send the user to the page requested or redirect to the login page.
6.4. Profile

6.4.1. Overview
At this stage only 2 mains functionality will be built around the profile. One screen in order to see the profile of someone and a second one to allow the user to edit his own profile information.

6.4.2. View a profile
6.4.2.1. Functional specifications
Path: /profile/view/:username, with :username the username of the user

**Description**

The purpose of this page is to have general information concerning a user of the application. At the left aside of the page will be displayed the picture of the user. If the profile page belongs to the connected user then an ‘edit profile’ button will appear under the picture. On the central part of the page will be displayed the information of the user. At the top should appear distinctly the username. Below it his description, the age and location of the user. At the bottom of the screen the recent activities of the user will be displayed.

**Events:** On click of the user picture open it in the modal gallery. On click of the ‘edit profile’ button go to edit profile view. Recent activities will have on click events to redirect to the subject they relate.
6.4.2.2. Implementation

The main controller for the profile view is called ViewProfileCtrl, this controller will bind data and methods with the template linked to it. In order to display the user information the module will preload it and inject it into the controller through userPre. The controller will also inject ceAuthentication which is the service that handle the user connexion, it will be used to know if the profile belongs to the connected user. The last injection will be the modal gallery service in order to communicate easily between the directive and the controller when it needs to render an image.

The controller will only implement 2 functions; one to check if the user connected is the same as the profile visited and another one to show the picture in the modal gallery on click.

6.4.3. Edit a profile
6.4.3.1. Functional specifications
Path: /profile/edit/:username, with :username the username of the user
Description

The purpose of this page is to modify the profile information of the current authenticated user. At the left side of the page will be displayed the picture of the user along with a ‘save’ and a ‘cancel’ button. On the central part of the page will be displayed the information of the user in amendable inputs. At the top should appear distinctly the username. Below it the user can change his birth date with an input formatted dd/mm/yyyy and his location. At the bottom a large textarea is available to put his description.

Events: On click of the edit icon on the picture it opens a file explorer in order to import a picture. On click of the ‘save’ button it will send an edit request to the server and will go back to view profile. On click of the ‘cancel’ button it just go back to the view profile without saving the information the user typed.

6.4.3.2. Implementation

The main controller for the profile edit is called EditProfileCtrl, this controller will bind data and methods with the template linked to it. In order to display the user information the module will pre-load it and inject it into the controller through userPre. In order to save the profile of the user at the end of the edit then the controller needs to inject the resource for user called ceUsers. The controller will implement a single function in order to save the amendments made to the profile.
6.4.4. Profile server implementation

6.4.4.1. Mongoose Schemas

The users’ profile are handled via the User Schema as follow:

<table>
<thead>
<tr>
<th>Users</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Full name of the user</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
<td>Username</td>
</tr>
<tr>
<td>picture</td>
<td>Picture</td>
<td>Profile picture</td>
</tr>
<tr>
<td>location</td>
<td>String</td>
<td>User’s location</td>
</tr>
<tr>
<td>dob</td>
<td>Date</td>
<td>Date of birth</td>
</tr>
<tr>
<td>email</td>
<td>String</td>
<td>E-mail of the user</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>User’s password</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>User’s description</td>
</tr>
<tr>
<td>activities</td>
<td>[Activity]</td>
<td>User’s activities on the app</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>[Content]</td>
<td>Content</td>
</tr>
<tr>
<td>date</td>
<td>Date</td>
<td>Date of creation</td>
</tr>
<tr>
<td>picture</td>
<td>Picture Schema*</td>
<td>Image related</td>
</tr>
<tr>
<td>title</td>
<td>String</td>
<td>Main message of activity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Content</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>picture</td>
<td>Picture Schema*</td>
<td>Picture of content</td>
</tr>
<tr>
<td>text</td>
<td>String</td>
<td>Text content</td>
</tr>
<tr>
<td>title</td>
<td>String</td>
<td>Content title</td>
</tr>
</tbody>
</table>

User’s password is encrypted in Bcrypt and therefore are not directly visible in the database.
6.4.4.2.  **API Methods**

6.4.4.2.1.  **External services**

**Create a user** – POST /users

Sending a POST request at ‘/users’ along with data matching the Mongoose schema for User will result in the creation of a new user in the database. The password will be encrypted before being saved for the safety of the user.

**Get a user from username** – GET /users/:username

Sending a GET request at /users/:username with :username being the username of the user wanted will return the entire user object of the database, excluding password and e-mail for obvious reasons.

**Edit a user from username** – PUT /users/:username, need authentication

Sending a PUT request at /users/:username with :username being the username of the user wanted along with data matching the Mongoose schemas for User will result in the edition of the user information in the database if the username match the user authenticated.

**Add new score** – POST /users/:username/scores, need authentication

Sending a POST request at /users/:username/scores with :username being the username of the user along with a score for a quiz will result in: adding the score if there is no previous score for that quiz or keeping the higher score if a previous score exist. The response returned will contain the highest score performed for that quiz and the total score of the user.

6.4.4.2.2.  **Internal functions**

In order to match the external services of the API with the Mongoose Schemas some internal functions need to be implemented:

- **AddScore**: Add an entry quiz/score to the scores dictionary\(^{19}\).
- **GetFromUsername**: Get a user from its username.
- **Login**: Authenticate a user from username or email and password. Send back a token.
- **Signup**: Create a user in the application.
- **Update**: Update a user from its username.

\(^{19}\) A dictionary is a 1 to 1 storage. One key contain one value. In that case each we access scores through quiz Id.
6.5. Recipes
6.5.1. Display a recipe
6.5.1.1. Functional specifications
Path: /recipe/display/:id, with id being the id of the recipe

The purpose of this page is to display all information relative to the recipe.

The page will be constructed around 2 main axes:

- Getting general information on the recipe.
- Decline the recipe into step to accompany the user during the recipe.

6.5.1.1.1. Info page

![Recipe Display Diagram]

**A**

**B**

**C**
**Description**

- (A) At the top will be the name of the recipe along with a picture. The page will also display short information such as the difficulty, the number of persons expected to eat, the date of creation and the author username. **Events:** When clicking on the recipe picture, you will be able to see it in bigger size. When clicking on the author username, you will be transferred to his profile page.

- (B) After this short summary, will be displayed the ingredients of the recipe as a list. Following the ingredients, the page will then display all the steps you need to achieve in order to complete the recipe.

- (C) The page ends with the list of the useful utensils for the recipe, a button to start the recipe and a comment section to see users’ pictures of the recipe, read their comments, or add your own. **Events:** When clicking on the button ‘Start recipe’ you will be redirected to the second part of the recipe display. When clicking on the users’ picture of the recipe, you will have a modal gallery displaying all pictures in bigger size. When adding a comment, it will be possible to add a mark by clicking on the stars. When clicking on a username or a picture in the comment section, you will be redirected to the user profile.

**6.5.1.1.2 Steps pages**

**Description**

In this screen, each step will be displayed independently and should stay as simple as possible. For this reason, on this page, will be shown only: The picture of the step, the action of the step, a timer if required, and navigation buttons. On top of this screen, a line with all steps will allow the user to navigate quickly between steps.

**Events:** When clicking on the right or left arrow, it goes to previous or next step. Clicking on the start/stop/pause on the timer will activate/stop/pause the timer. On click of one of the step in the step line, the application will navigate to that step.
Description

Once a user has finished all steps the application will display an ending screen. This screen will be implemented in order to allow the user to interact with the recipe. Therefore the user will be able to add like the recipe, to leave a picture of its own masterpiece and leave a comment with a mark. For more information about the comment section read the ‘Info page’ which has been written previously. A step line will be implemented on top of the page, the same as described in ‘Steps pages’.

Events: When user will like the recipe it will add it to its recent activity list. When the user leaves a picture it will add it to the list of users’ picture of the recipe. The user will be able to drop the picture directly in the box or click on the box and find it through the file explorer. When the picture is upload it will be displayed instead of the add/drop box.
6.5.1.2. Implementation

The main controller for the recipe display is called RecipeDisplayCtrl. This controller will bind data and methods with the template linked to it. In order to display the recipe correctly the module needs to preload the recipe beforehand and inject it through recipePre in the controller. The controller will also attach the $http service of angular in order to make calls to the REST API when the user like the recipe. The controller will also need to inject the resource for comments ceRecipeComments in order to add, edit or delete comments. The last injection is the modal gallery services in order to share data between the controller and the directive in a simpler way.

The controller will implement different functions:

- Navigation functions: Go to X, previous/next.
- User interaction: Like a recipe, add/edit/delete comment, give a mark
- Timer (pause/start/stop) for steps
- Open image gallery for main picture, users pictures and steps pictures.
6.5.2. Recipe search
6.5.2.1. Functional specifications

Path: / (index page)

Description

This page purpose is to find recipes easily; therefore at the top of the page the user will find a search input along with filters and sorting tools. Just under will be displayed recipes with their images and their title. Recipes will be displayed in 3 columns of same width.

Events: When typing in the input bar the recipe list will automatically be updated according to the input. The same effect is applied when the filter is changed or when using the sorting tool. On mouse over recipe they will scale a bit bigger in order to help the user to see clearly which one he is choosing. On click on the recipe image or title the user will be redirected on the recipe page of the recipe he clicked.
6.5.2.2. *Implementation*

The main controller for the recipe research is called RecipeSearchCtrl, this controller will bind data and methods with the template linked to it. In order to display some pre-loaded recipes when the user is not searching anything the controller needs to inject recipePre containing the recent trends that were retrieved beforehand from the module. The controller will also inject the recipe resources in order to makes request to the server for researching other recipes. The controller will only implement two functions; one to search recipes and another to display a recipe that has been selected.

6.5.3. *Create a recipe*

6.5.3.1. *Functional specifications*

Path: /recipe/create

In order to create a recipe 3 different screens are required. The first screen concern the recipe information, the second is used to makes steps one by one and the last is to import a final picture for the recipe and confirm the creation.

6.5.3.1.1. *Info page*
Description

The first page presented to the user in order to create a recipe will ask information for the recipe to the user. At the top of the screen is located an input to type the name. Just below it there should be a sentence to help the user enter the number of persons and the duration of the recipe such as: “This is a dish for X persons and take X mins to cook.” Under this sentence will be located an input for number to enter the difficulty of the recipe (between 1 and 5).

The second part of the first page are concerns ingredients and utensils, the user should be able to create a list of them very easily through an input with a Add button. When a item is adding it will be displayed in the list with a remove button in case the user want to remove it. Utensils are simple strings but ingredients are separate in Names, Quantity and Unit.

At the bottom is located a button Next to continue the process.

Events: On click of add buttons for ingredients or utensils will add them to the list, beforehand it should check that it is not empty otherwise it will display an error. On click of the remove button in the list of ingredients/utensils it should remove the object from the list. On click of the Next button, the information will be entirely check, if any errors are on the page then it will display it otherwise it will pass to the step page.

6.5.3.2. Steps creation
Description

The second screen is the step creation. Each time the user create a new step or navigate to any step this page will be rendered. At the top of the page is displayed a navigation line to easily go from one step to any other one. Just under will be displayed the step number and a little icon in order to add a timer for the step.

The main part of the screen will be a textarea to enter the task the user has to do: an add/drop picture box below it and 2 large arrow on the right and left to navigate.

At the bottom of the page is located 2 buttons: Delete and Terminate, to delete a step or to finish the steps creation.

Events: On click of any navigation step, go to that step. On click of add/remove timer button toggle the timer display. On click of the arrow go to next or previous step. On click of the add/drop picture box a file explorer will open to search a picture. On drop of picture upload the picture. When the picture is uploaded replace the add/drop box by the picture. On click for delete then delete the current step. On click of Terminate go to the end of the recipe creation. Before each navigation event check for errors in the step, display an error message if required and do not process to the navigation.

6.5.3.1.3.   End of recipe

Description

The ultimate screen is just used to upload the main picture, therefore a step-line to go back to previous steps will be displayed at the top of the screen along with a add/drop picture box below it. At the bottom a button ‘Submit your recipe’ will end the recipe creation.

Events: On click of any navigation step, go to that step. On click of the add/drop picture box a file explorer will open to search a picture. On drop of picture upload the picture. When the picture is uploaded replace the add/drop box by the picture and active the final button. On click of ‘Submit your recipe’ send the recipe to the REST Api and redirect on the page of the create recipe.
6.5.3.2. Implementation

The main controller for the creation of recipes is called `RecipeCreateCtrl`, this controller will bind data and methods with the template linked to it. In order to create the recipe at the end, the controller needs to inject the resources `ceRecipes` therefore when the recipe needs to be saved it will go through this resource.

The template will implement 2 different directives, the first one being the step-line directive which is used to navigate through the steps and the second is the picture-upload directive used several times to add pictures for steps or the principal image.

The controller will implement different functions:

- Navigation functions: go to X, next/previous, last, finish recipe.
- Recipe information functions: add/delete utensils, add/delete ingredients.
- Steps functions: Delete step, toggle timer
- Picture upload functions: picture for steps, principal image.
### 6.5.4. Recipes server implementation

#### 6.5.4.1. Mongoose Schemas

<table>
<thead>
<tr>
<th><strong>Recipe</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the recipe</td>
</tr>
<tr>
<td>author</td>
<td>User Schema*</td>
<td>Author of the recipe</td>
</tr>
<tr>
<td>course</td>
<td>Number</td>
<td>Start / Main / Dessert</td>
</tr>
<tr>
<td>createdOn</td>
<td>Date</td>
<td>Creation date</td>
</tr>
<tr>
<td>difficulty</td>
<td>Number</td>
<td>Difficulty of the recipe (1-5)</td>
</tr>
<tr>
<td>comments</td>
<td>Comment</td>
<td>User comments on the recipe</td>
</tr>
<tr>
<td>ingredients</td>
<td>[Ingredient]</td>
<td>Ingredients of the recipe</td>
</tr>
<tr>
<td>nbPerson</td>
<td>Number</td>
<td>Number of person for the recipe</td>
</tr>
<tr>
<td>picture</td>
<td>Picture Schema*</td>
<td>Main picture of the recipe</td>
</tr>
<tr>
<td>pictures</td>
<td>[Picture Schema*]</td>
<td>Pictures imported by users</td>
</tr>
<tr>
<td>steps</td>
<td>[Step]</td>
<td>Each step of recipe</td>
</tr>
<tr>
<td>time</td>
<td>Date</td>
<td>Total time of the recipe</td>
</tr>
<tr>
<td>utensils</td>
<td>[String]</td>
<td>List of utensils</td>
</tr>
<tr>
<td>updatedOn</td>
<td>Date</td>
<td>Date of creation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Ingredient</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the ingredient</td>
</tr>
<tr>
<td>qte</td>
<td>Number</td>
<td>Quantity of the ingredient</td>
</tr>
<tr>
<td>unit</td>
<td>String</td>
<td>Unit for the ingredient</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Step</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Number</td>
<td>Step number</td>
</tr>
<tr>
<td>action</td>
<td>String</td>
<td>Definition of what to do</td>
</tr>
<tr>
<td>picture</td>
<td>Picture Schema*</td>
<td>Picture of the action</td>
</tr>
<tr>
<td>time</td>
<td>Date</td>
<td>Timer for the step</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Comment</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>User Schema*</td>
<td>Username of the message creator</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
<td>Content of the comment</td>
</tr>
<tr>
<td>createdOn</td>
<td>Date</td>
<td>Date of the message creation</td>
</tr>
<tr>
<td>mark</td>
<td>Number</td>
<td>Mark the user gave to the recipe</td>
</tr>
<tr>
<td>updatedOn</td>
<td>Number</td>
<td>Date of update</td>
</tr>
</tbody>
</table>
6.5.4.2. API Methods

6.5.4.2.1. External services

**Search for recipes** – GET /recipes
Sending a GET request without any attributes at “/recipes” will return the list of the most common recipes lately. It is possible to add a query parameters “match” along with a text in order to filter recipes matching the different words in the text. Other query parameters to filter recipes can be added in the future.

**Create a recipe** – POST /recipes, need authentication
Sending a POST request at “/recipes” along with data matching the Mongoose schema for recipe will result in the creation of a recipe in the database. The author of the recipe will have its Recent Activities updated with a message telling that he has created a recipe.

**Get a recipe** – GET /recipes/:id
Sending a GET request at “/recipes/:id” with :id being the id of the recipe wanted will return the entire recipe object of the database.

**Delete a recipe** – DELETE /recipes/:id, need authentication
Sending a DELETE request at “/recipes/:id” with :id being the id of the recipe will delete the recipe if the author is the authenticated user.

**Add comments to a recipe** – POST /recipes/:id/comments, need authentication
Sending a POST request at “/recipes/:id/comments” with :id being the id of the recipe along with data matching the Mongoose schema for comments will result in adding the comment to the list of comments for that recipe. It will also add a message on the Recent Activities of the user.

**Delete comments of a recipe** – DELETE /recipes/:recipeId/comments/:id, need authentication
Sending a DELETE request at “/recipes/:recipeId/comments/:id” with :id being the id of the comment and :recipeId the id of the recipe will result in deleting that comment if the authenticated user is the author of the comment.

**Like a recipe** - PUT /recipes/:id/like
Sending a PUT request at /recipes/:id/like with :id being the id of the recipe will result in increasing the number of like of that recipe and also add a message on the Recent Activities of the user.
6.5.4.2.2. Internal functions

In order to match the external services of the API with the Mongoose Schemas some internal functions need to be implemented:

- AddComment: Add a comment in the comments list and set a new Activity for the User.
- Create: Create a recipe document if Schema is conform and set a new Activity for the User.
- Get: Search and return a recipe from its _id attribute.
- Like: Add the username of User to the likes set\(^20\) and set a new Activity for the User.
- List: Search for recent recipes sorted by number of likes and filter match on name attribute.
- Remove: Remove a recipe from its _id attribute.
- RemoveComment: Remove a comment from its _id attribute and the _id of the recipe.
- UploadPicture: Add a picture to the pictures list and set a new Activity for the User.

\(^20\) Set is a list were elements cannot be repeated, in that case it means a user cannot like a recipe twice.
7. Testing

7.1. Overview

In order to fully test the application unit testing and end to end testing on the web application and unit testing on the back-end should have been implemented. With only 3 months it was planned to only make the unit tests for the web applications. However the redaction of this report took more time than expected and consequently even the unit tests could not be written entirely.

7.2. Unit test for the web application

Unit test for the web application were written under Mocha, Chai and SinonJS. In order to properly unit test an angular application you have to test: each controller, each services and each directives.

Here is an example of a test bench that was written for the controller display game:

<table>
<thead>
<tr>
<th>Steps navigation</th>
<th>test</th>
<th>result expected</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>start with info</td>
<td>should work</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>next from info</td>
<td>go to 1st step</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>go to any step</td>
<td>go to the step</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>next from last step</td>
<td>ok</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>previous on 1st step</td>
<td>go to info</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>previous on 2nd step</td>
<td>go to 1st</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>increase temp mark</td>
<td>increase</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>decrease temp mark</td>
<td>decrease</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>reset temp mark</td>
<td>reset</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>save mark</td>
<td>save</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>with mark</td>
<td>should work</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>without mark</td>
<td>should work</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>less than 10 char</td>
<td>raise error</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>call services success</td>
<td>add comment</td>
<td>ok</td>
<td></td>
</tr>
<tr>
<td>upload success</td>
<td>should add picture</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Show picture

<table>
<thead>
<tr>
<th>test</th>
<th>result expected</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>show first</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>show random</td>
<td>ok</td>
<td>ok</td>
</tr>
</tbody>
</table>

Modal galley

<table>
<thead>
<tr>
<th>test</th>
<th>result expected</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>open first picture</td>
<td>open first picture</td>
<td>ok</td>
</tr>
<tr>
<td>open random picture</td>
<td>open picture</td>
<td>ok</td>
</tr>
</tbody>
</table>

Like

<table>
<thead>
<tr>
<th>test</th>
<th>result expected</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>like recipe</td>
<td>like recipe</td>
<td>ok</td>
</tr>
</tbody>
</table>
7.3. **Unit test for the back-end server**

Unit test for the back-end server have not been written yet however some test cases have been written already and have been tested differently. Postman is a plugin for Google Chrome that can handle sending and receiving HTTP messages; this plugin was used in order to quickly test if the behaviour of the API was correct. However those tests are not enough for long term testing and this is the reason why unit tests will be written in the future anyway.

Below is an example of a test case written for creating recipe:

<table>
<thead>
<tr>
<th>test</th>
<th>Error</th>
<th>working</th>
</tr>
</thead>
<tbody>
<tr>
<td>course</td>
<td>undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong value</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>good value</td>
<td>-</td>
</tr>
<tr>
<td>difficulty</td>
<td>undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong value</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>good value</td>
<td>-</td>
</tr>
<tr>
<td>ingredients</td>
<td>undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong ingredient name</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>ingredient name undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong ingredient unit type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>ingredient unit undefined</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>wrong ingredient qte type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>ingredient qte undefined</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ingredient qte &lt; 0</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>good value for everything</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>length&lt;5</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>good value</td>
<td>-</td>
</tr>
<tr>
<td>nbPerson</td>
<td>undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>nb&lt;0</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>good value</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>steps</td>
<td>undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>empty array</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong action type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>action undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>action &lt; 10 char</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>stepnb undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>stepnb wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>step nb wrong order</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>time undefined</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>time wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>time &lt; 0</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>picture undefined</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>picture wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>everything good</td>
<td>-</td>
</tr>
<tr>
<td>picture</td>
<td>undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>good picture</td>
<td>-</td>
</tr>
<tr>
<td>utensils</td>
<td>undefined</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>empty array</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong array element</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>good</td>
<td>-</td>
</tr>
<tr>
<td>time</td>
<td>undefined</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>wrong type</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>&lt;0</td>
<td>err</td>
</tr>
<tr>
<td></td>
<td>good value</td>
<td>-</td>
</tr>
</tbody>
</table>
8. Results and Evaluation

8.1. Evaluation

8.1.1. Introduction

In order to evaluate the application 6 persons were interviewed and had to follow different tasks. At the end of each task interviewees were asked to give a mark on the difficulty to accomplish the task and to give feedback on what was difficult in their opinion. The tasks were:

- Find a recipe in order to cook carbonara pasta. Follow the recipe. Once you are done show that you like the recipe and leave a comment or a picture of your masterpiece.
- Your knowledge on fishes is quite poor, find a quiz to train yourself. If your score is too low then you can try again.
- Create an account and edit your profile information
- Create a recipe of your choice
- Create a game of your choice

8.1.2. Evaluation feedbacks

Display recipe - 4.6/6
User feedback
  - Difficult to know that we can see the pictures x1
  - Another version to all steps on a single page would be better when you are cooking x1
Test feedback
  - Users cannot see well the “Like this recipe” button.

Display quiz - 4.3/6
User feedback
  - Was not clear when you find the good answer (should say correct/incorrect) x4
  - Size of the question not big enough to focus on it x1
Test feedback – none

Create account / edit profile – 5.3/6
User feedback
  - Need help to understand password length x1
Test feedback
  - User should be help on inputs entry by messages as the same time he’s typing

Create game - 4.8/6
User feedback
  - Add green colour to correct answers in order to catch focus x1
Test feedback
  - Users don’t know the standards for answers (between 2 and 4 answers + at least one correct but multiple are possible)
Create recipe - 4.5/6

User feedback

- Add timer for step is confusing x1
- 10 char min is annoying x2
- Tell that unit is not required for ingredients x1
- Change terminate by another clearer word such as Finish x1
- Have a guide to help user understanding how to creating recipe x1

Test feedback

- Users have difficulty to find the button to finish the recipe
- Most of users will fill in utensils list
- Users is annoyed when he wants to step back and he is on a new step.

Other

- Menu not easy to find x1
- Not enough type of game x1
- Need a possibility to edit recipe x1

8.1.3. Evaluation analysis

At the first glance most of the screens have a correct grade which means the application is globally correct and does what it was required to do. With those grades it is possible to spot the main points to concentrate on.

The display quiz for example is one of the most important point of the application but has the lowest score. In the details of the feedback the real problem on the quiz display is how the users’ answers are handled, reason why this would be the priority point to fix by giving a message on the screen whether your answer is correct or not.

The second lowest score is the recipe creation, which has many small implementation to fix in order to be nice to the user. However, considering that display recipe will be much more used than create a recipe and the scores being nearly similar it is more advantageous to concentrate on the display recipe first.

On the other hand screens for create an account and editing profile are not the priority of the application but they are the higher marked. Therefore it will be definitely the last point to improve. Same for Create a game, which is not really the main point of the application and has a great score as well.

Once everything will be fixed, a second evaluation can be set up in order to have new ideas on the things to do and other problems that couldn’t be seen at this test.
8.2. Results

Screens that were designed and implemented are available in Appendix 1 at the end of the reports.

Table 1 shows the results according to the objectives fixed at the beginning and Table 2 shows the results according to what should be really implemented. The results compared to the objectives fixed in the beginning of the project are quite good however the second table clearly shows that the application is not ready for production.

The work produced during this project was clearly important and the only way this project could have gone into production in 3 months was to have another person working on the application. The overall process was very time consuming because it started from scratch. Many tasks were required before coding the application such as researches, designing the screens, etc... The coding part of the application took only 1/3 of the total time of the project.

In my own opinion I am very satisfied of the work produced, I think everything works as it was defined at the beginning of the project. However I am also conscious that the application cannot be launched at this instant since important features still need to be implemented beforehand.

<table>
<thead>
<tr>
<th>Part</th>
<th>% finish</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Games</td>
<td>80%</td>
<td>- Needs a better solution to display quizzes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Finishing a quiz need to create a recent activity for the user</td>
</tr>
<tr>
<td>Layout</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Profile</td>
<td>90%</td>
<td>- Not possible to edit profile picture</td>
</tr>
<tr>
<td>Recipes</td>
<td>80%</td>
<td>- Create a recipe should create a recent activity for the user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Filters for recipes needs to be implemented</td>
</tr>
</tbody>
</table>

Table 1

<table>
<thead>
<tr>
<th>Part</th>
<th>% finish</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>85%</td>
<td>- It works correctly but a solution for password forgotten should be implemented</td>
</tr>
<tr>
<td>Games</td>
<td>40%</td>
<td>- Needs a more kind of games</td>
</tr>
<tr>
<td>Layout</td>
<td>95%</td>
<td>- Header logo needs a better design</td>
</tr>
<tr>
<td>Profile</td>
<td>80%</td>
<td>- Needs a way to change password</td>
</tr>
<tr>
<td>Recipes</td>
<td>50%</td>
<td>- Needs a way to save/edit recipe and publish them independently</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td>- Need to implement a system to save favourite recipes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Need to implement a fridge / shopping cart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ...</td>
</tr>
</tbody>
</table>

Table 2
9. Problems encountered

9.1. Code maintainability

One of the most important problems to face was to weight the time loss due to an architecture not adapted to the growth of the project opposed to the time loss to refactor it in a better way. The web application was subject to 2 refactoring and the REST Api to 1.

9.1.1. Web application

Original architecture of the application

Second architecture
Final architecture

Folder organized into recursive modules, app use game.js itself using create, display and search

Tests are within the modules they test

Resources are shared within the whole application in order to avoid duplicates

From original to second architecture

The original architecture is the basic configuration for starting an Angular application, it allows to clearly separate the JavaScript code into their roles and is very efficient to kick off a project. However it happens that this is not scalable at all, more the functionality were implemented more it became complicated to get one’s bearings around the code.

After some researches it happened that many other developers had the same issue. (Papa, 2014-2015), one of the famous developer in the Angular community, wrote a style guide were he advices developer on how to properly manage an Angular application. Many applications have been developed around that guide and all maintainable applications follows similar guidelines.

The refactoring took one full day of work. However after this refactoring, due to the convenience it brought, approximately one hour of work per day was earned by being more efficient into writing code. Once it was sure that the refactoring worked properly and didn’t bring any defects the old application was replaced by the new one.

From second to final architecture

The second refactoring was brought for three reasons. The main reason was to take advantages of the modularity of Angular. There is a file app.js appearing in each file tree; this file concerns is the one called for controlling the overall application. In the two first version it contained all the information for each functionality and their details.

However as the project became bigger this file started to be hard to read and the list of repertories per functionality started to get longer as well. In that sense it was decided to use the modularity of Angular to create modules calling other modules. For example the Application module will use the Game module but doesn’t need to know any information about, all this information will be treated in
that Game module. The Game module itself will have 3 sub modules, one for Create a game another for Display a game and a last one to Search for games which will have their own information.

This brings a hierarchical structure leading to a greater readability of the code and a higher specification for each module.

The second point of this refactoring was to separate the common services such as the resources; the purpose was to reduce the code duplication to its minimum. Beforehand those resources were included inside the different controller or in the general app, now they have their own module and can be injected on their own.

The last point concerns unit testing. The refactoring was made when the unit tests were starting to be redacted and it became very annoying to have 2 different open trees to work on the same functionality. Therefore instead of having a different repertory for the unit tests those were directly included in the repertory of their functionalities and are differentiate from the normal code by a suffix `.spec.js`.

The refactoring was not as important as the previous one and therefore only took half a day. The effect of the refactoring were not as visible as the previous one on the code writing however it really helped for the testing part; firstly because they were now in the same repertory, secondly because Angular unit testing becomes easier when the code is more modular. The shifting process took place in similar conditions as the previous one.

9.1.2. Back-end server

The only refactoring which happened on the back-end was very quick, less than half a day.

The first reason was to pre-load the Mongoose Schemas before starting the server. It was not required at the beginning of the application because the Schema validation was not implemented yet. However all schemas should be loaded beforehand in order to be able to add sub-documents validation in Mongoose. Document validation is too important to be omitted therefore this refactoring had to happen.

The second reason concerned the repartition of the work process within files, previously when receiving an API call the server used to call a handler, the handler will pass its requests to a database object manager which will interact with the Mongoose Schema. The database object manager was not really required since all the functions could be moved to either the handler or the Schema’s methods. In that case having one more layer brings more complexity than it helps therefore it had to be removed. This reason was not a necessary reason to refactor at this stage, but since it was mandatory to refactor for the first reason it was just done in same time because it could have been burdensome on the long time.

9.2. Back-end server framework

At the kick-off of the project HapiJS was considered to be used as the back-end server framework. During the 2 first weeks it was very difficult to put in place all the configurations of the server and the authentication process for the REST API started to get very complicate.

It was not possible to lose more time just for configurations. For this purpose the decision to take one day off to rewrite a separate application in Express was made; I was already experienced with Express.
Within one day everything that was previously done was nearly covered therefore another day was given to see until where it could go.

In 2 days the server on Express implemented everything that was present in the HapiJS server, moreover there was enough time to implement a working authentication! The community being more important on Express it was very easy to find all necessary information in few seconds. After verifying that everything worked properly the definitive switch from HapiJS to Express happened.

9.3. Features implied by other
Every essential features that were required at the beginning of the project were correctly implemented. However meanwhile the application becomes more mature many unpredicted features seems to be essential before a launch in production. Those features takes time to develop and it is not possible to implement them without lengthen the deadline.

An example that was mentioned previously thanks to the evaluation is the implementation of a guide to create a recipe. With all the difficulty encountered by users on the recipe creation it seems that the application cannot be launched without a solution for it. Finding the right solution, design it, implement it and testing it will take a significant amount of time that was not planned at the beginning of the project.

Therefore the original objectives are half achieve. The application is working properly and can be used right now, however those missing features make it incomplete and the user experience could really be improved. The problem related to this is that there’s only one chance to make a first impression; therefore the solution to postpone the launch in production deadline is more appropriate than launching an incomplete product. Since there’s no customer needs for the project it is a better choice to wait until the product becomes more user-friendly and more mature.

9.4. Group members unavailability
During the project it happened several time that one of the member could not be present at the working place for some reasons. While developing the application the group had to exchange a lot concerning the features implementation, the different services provided by the REST Api, how to keep an integrity between the mobile app and the web app...

When a person was not available it became harder to exchange, the solution was to set up skype meetings frequently and to exchange via a messenger for short questions or notices. However even with those conditions it happened sometimes that schedules were not matching and when a response was required quickly then it could block the other person in his task. There was no proper solution to this problems and in my opinion the only solution would have been to keep the same schedule for everyone even if they are distant in order to be always available for each other’s.
10. Future work

10.1. Enhancements
As stated in the result the applications does not feel completed yet. In order to overcome this feeling some enhancement of the current application must occur.

The first important part of the application to improve is the recipe creation. It really needs a way to save a recipe before publishing it, store it and be able to edit it before and after publishing it. The current version lacks of flexibility and can produce a bad experience for the user.

Second important point to improve is the Quiz display. Many people complained during the evaluation that it was not clear enough whether their answer are correct or not. The solution to it would be to redesign the display along with more user interactions. After this a second user evaluation should confirm whether the new solution is suitable.

Another point to improve is the game and recipe research. Those research on the server side currently match every words even incompletes ones. This is not suitable for researches, it uses a lot of computation and instead it would be preferable to use the MongoDB default research. The computation gained on this could be used by implementing an auto-completion meanwhile the user is typing.

Currently people knowing a bit about how an API works can find a solution to cheat on their score as there is no validation for it on the server side. In order to avoid malicious users who would like to increase their score without efforts the application will need a validation on those scores submissions.

Other enhancement less important needs to be done in the future, the exhaustive list is on the Trello project.

10.2. Features ideas
For the moment only the basic and essential features has been implemented. In order to give a better user experience and differentiate from other application many ideas of new functionality or complementary features were thought.

A feature that was thought was to give the user access to a “fridge”. The user could list all his food in a page of the application. The application then can use those elements in order to match recipes to the food the user has. In order to push this idea to the maximum the creation of a shopping list that can directly transfer the elements you buy into the fridge was thought. This shopping list could be filled either by typing the items or by clicking on ‘add ingredients to my shopping list’ on a recipe page. The goal is to make it convenient to the user and effortless.

Another point that is considered is to create new types of games. Quizzes are not holding the user back enough and it will be essential to have games than can produce a higher addiction. To accomplish this games with higher user interactivity will be implemented. Those games should be diversified in order to not be repetitive and tedious to users.

21 Trello is a managing tool, refer to the Project management part to know more about it.
10.3. **Overall project**

The overall application goal is still the same, launch it into production as soon as it is ready. Once the project is working properly a second evaluation will be set up. If the result is satisfying the application can be finally launched.

Before the end of the project it is important to consider a mobile version of the web application. Currently the application is only designed for a normal desktop but when the screen size get smaller the web application design is not perfect. Therefore the consideration of a new mobile version or using responsive design should be done.

Once the application will be launched it is important to have a digital marketing strategy established beforehand. A good website with good contents but without users is useless. Before launching the project it is important to target the user and find a way to attract them to use the application. This means considering a social media strategy, using google ad words, establish a strategy to keep users ...

After the project is launched it is important to keep an eye on the defects and what functionalities the users want to be implemented. If everything is alright then the next step would be to adapt the mobile application to support the maximum number of OS, e.g. Android, iOS, Windows...
11. Conclusion
The dissertation project is coming to an end however the project in itself is not finished and there are still many things to learn from it. If the project keeps growing it might be possible to have a real launch in production and starting to invest money into it.

Working in group improved the experience of the project. It was great to be able to exchange ideas and have an external point of view on the personal work. Since the project was the same, just the work was different, it became very simple to exchange quick advices without having to explain the overall project.

The project went well however it would have been better with a longer deadline or with one more person working on it. The redaction of the report took a lot of time and greatly shorten the actual time to spend on the project. It is always complicate in IT to make the right choice between the amount of time to spend in writing documentation and the time to spend on the implementation. Since the report has a higher importance on the grade it was not possible to give as much time on the implementation as it would have been possible for a real project.

The project main purpose was to have an overview of the conception of an IT project. This includes finding the user requirements, the different choice for technologies, the designing of the functionalities, the implementation, the testing and the production launch. Except the last bit everything went well and in that sense this project was very informative.

This project was a great pleasure to carry out and very helpful for my future. The freedom of the project gave me the choice to learn what I was interested in and it paid out. I learned a lot on various field and get a much wider knowledge on the set up of a working environment for an application. From now on I will be able to easily take a global view in an IT project.
References


Appendix 1: Screens

1. Authentication

1.1. Login

1.2. Register
2. Game
2.1. Create quiz
2.1.1. Information page

![Image of quiz creation interface]

Set a title for your quiz:

*Amazing quiz*

Main picture of your quiz:

*Add or Drop picture*

Create questions

2.1.2. Text questions

![Image of text question interface]

Your question:

*Flop*

Answer type:

- [ ] Text
- [X] Images

one

deux

tres

Add answers

Answer

![Images of answer correctness options]

Next question  Finish quiz
2.1.3. Images questions

Your question:

plcp

Answer type:

- Text
- Images

.Text on picture (optional)

Add or Drop picture

Uncorrect Correct

Next question Finish quiz

2.2. Search game

Search for a game

le monde

by polinux on Jul 18, 2015

BRA BRA

by jazdeido on Aug 5, 2015

This is an amazing quiz

by Yummary on Jul 19, 2015

le capitale de

by polinux on Jul 20, 2015

sur le monde

by polinux on Jul 20, 2015

Carrot lovers

by Yummary on Jul 14, 2015

Find the fish

by Yummary on Jul 29, 2015

How to make curry

by Yummary on Jul 30, 2015
2.3. Display game

2.3.1. Questions

2.3.2. End
3. Layout

3.1. Header

3.1.1. No user logged

3.1.2. User logged

3.2. Menu
3.3. Modal gallery

Rice salad

- 3 Eggs
- 2 Tomatoes
- 1 Cucumber

4. Profile
4.1. View

Yummy

I love to eat bananas they are so good ! yumm comme ça c'est sur deux lignes pour Pauline J'adore écrire sur une ligne de plus, ou deux.More info fro, ne

31 Ans Edinburgh, Scotland

Recent activity

Yummy likes Find the fish
Jul 30, 2015
4.2. Edit

Yummy

Full name: Arnaud Bertrand bb
Date of birth: 03/06/1984
Location: Edinburgh, Scotland
Description: I love to eat bananas they are so good! yumm comme ca c'est sur deux lignes pour Pauline. J'adore écrire sur une ligne de plus, ou deux More info fro, ne
5. **Recipe**

5.1. **Display**

5.1.1. **Information page**

---

**Rice salad**

*Difficulty: ★★★☆☆  ☀  50 min  3 person(s)*

---

**Ingredients**

- 200 g of Rice
- 2 Eggs
- 2 Tomatoes
- 1 Cucumber
- 150 g of Tuna
- 50 g of Anchovies
- 60 g of Sweetcorn
- Mayonnaise
- Salt
- Pepper

**Steps**

1. Cook the rice and put it in a salad bowl
2. Cut the tomatoes in moon shape
3. Use half of the cucumber to make dices
4. Boil 2 eggs for 10m
5. Cut the anchovies in small pieces
6. When eggs are cooked dice them
7. Put all ingredients together. Add mayonnaise, salt and pepper. Mix everything
8. Leave it in the fridge for 2 hours. Then enjoy : )

---

*Created on Jul 3, 2013 by Yummy, .

Start Recipe
5.1.2. Steps

5.1.3. End of steps
5.2. Search

5.3. Create

5.3.1. Information page

Create a recipe

Recipe name:

Enter the name of your recipe

This is a dish for Nb persons and takes Nb mins to cook.

Course: Pick a course

Difficulty: (Between 1 and 5)

Ingredients

Name Quantity Unit

Utensils required

Name

Next
5.3.2. Steps

Step 3

☐ Add timer

To do...

Add or Drop picture

Delete  Terminate

5.3.3. End

Upload the main picture of your recipe

Add or Drop picture

Submit your recipe!