Design & Engineering- Process Document
Mauricio Hernandez

SOFTWARE REQUIRIMENTS SPECIFICATION (SRS)

SRS or “what the system needs to do”.

We started with an **Inception** phase where we got a basic understanding of the problem. This phase consisted of gathering multiple points of view from each team member and from the major stakeholder (Dion).

After having a general overview of the problem we moved to an **Elicitation** phase, where we kept brainstorming and having meetings with the stakeholder, in order to make sure that we have captured every input.

By the end of those phases we had a much clearer idea of the problem domain and we moved to an **Elaboration** phase, where we started doing initial design and management tasks, such as Use Case diagrams, an ER diagram, prototyping and scheduling. We also finished our initial SRS.

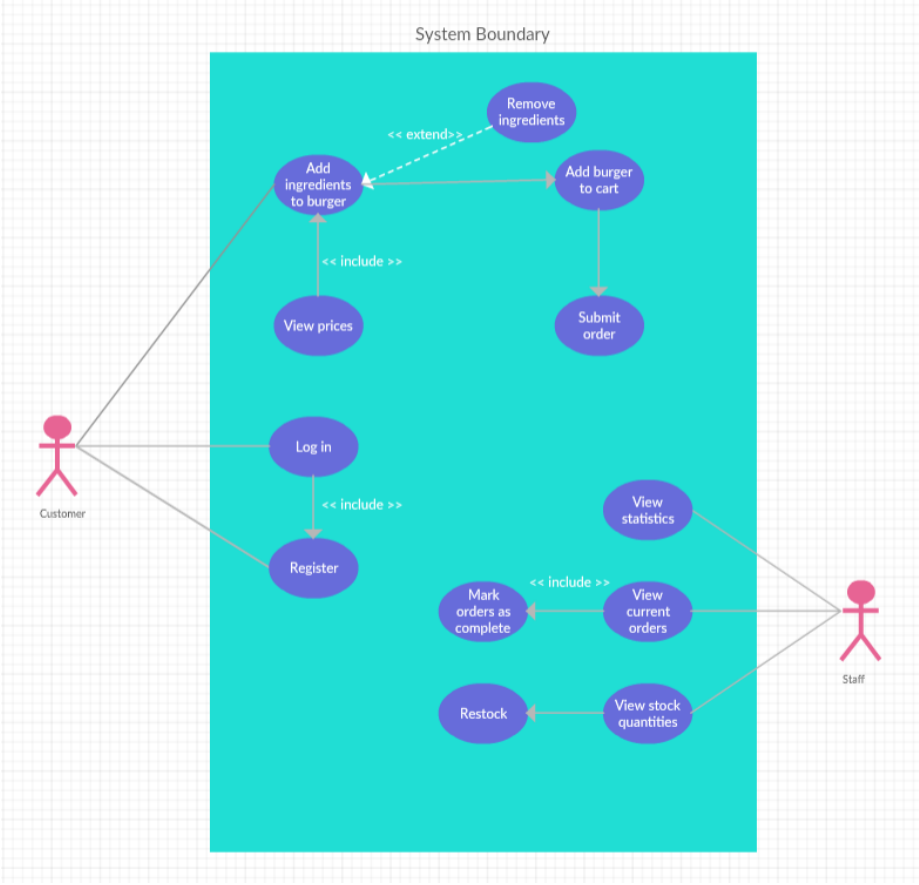
INITIAL FUNCTIONAL REQUIREMENTS	
Customer	Staff
C1. Customer shall be able to create a custom burger from select ingredients.	S1. An employee shall be informed about new orders.
C2. The system shall display a catalogue of premade burgers.	S2. An employee with manager clearance shall be able to review stock.
C3. The system shall inform that there is no current stock of a particular item.	S3. An employee with manager clearance shall be able to update stock.
C4 The system shall preview the ingredients of a custom burger before submitting an order (a cart view).	S4. An employee shall be able to modify the status of an order (therefore confirm it).
C5. The system shall allow to order more than one burger at a time	S5. An employee shall be able to login and be designated a clearance status depending on details.
C6. The system shall inform a customer the status of their order	S6. A manager shall have access to a statistical view of: >Most popular ingredients / Top selling premade option. > Busiest days of operation. >Monthly and annual turnover.
C7. The system shall register new users.	
C8. The system shall login existing users.	S7. An employee shall be able to assemble burgers with remaining ingredients.
C9 The system shall allow automatic payment.	S8. An employee shall have the option to donate to the local homeless community.
C10. The customer shall be able to modify the order (remove and add ingredients as desired).	
C11 The customer shall be able to save a preferred burger (favorite it).	
C12. The system shall filter ingredients according to dietary requirements.	
C13. The system shall be able to remember a particular customers’ order history.	

A functional requirement is something that the system must be able to do and that can be testable. Based on this we aimed to focus on single and very specific functionalities; not adding any technology choices.

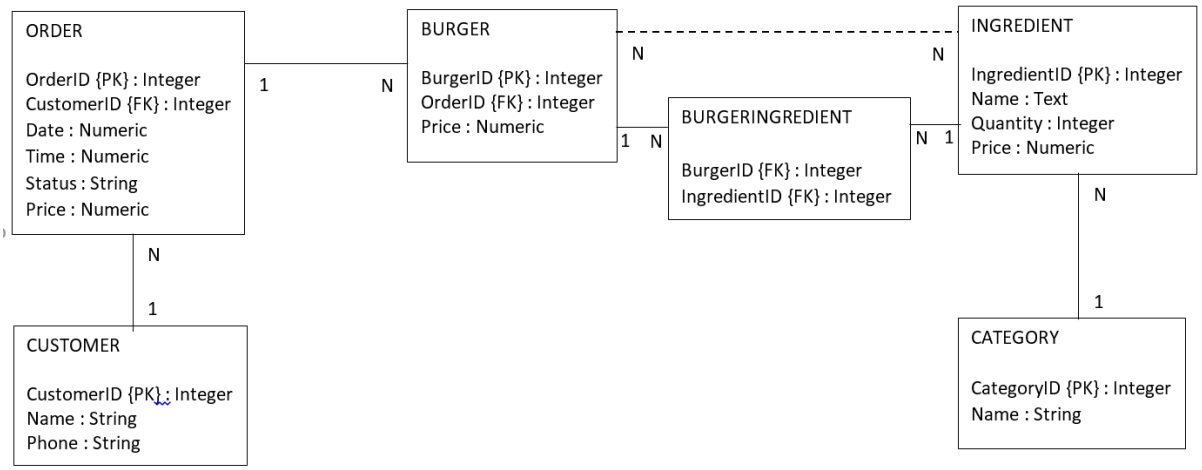
After defining our functional requirements we focused on the non-functional ones, which relate to the Quality of the Service (QoS).

INITIAL NON-FUNCTIONAL REQUIREMENTS
Q1. The system shall avoid SQL injections and
Q2. The system shall avoid unsafe password storage.
Q3. The system shall offer password encryption.
Q4. Web and Mobile applications shall be responsively designed to allow use by a variety of devices.
Q5. A backup of our database (usability consideration).
Q6. The general usability of both Mobile and Web applications shall be optimized.

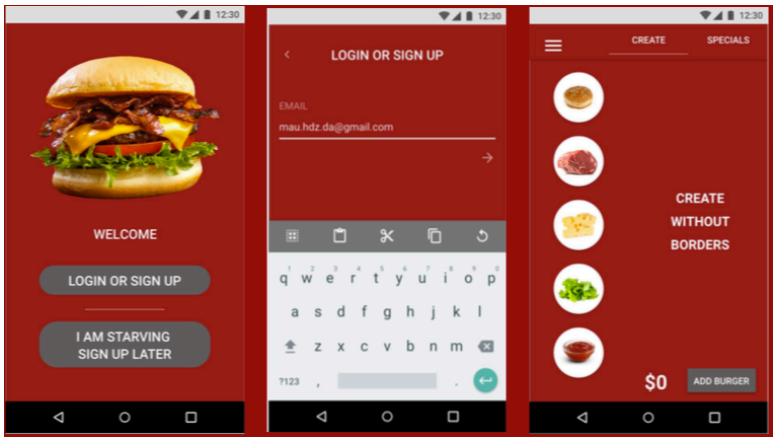
Regarding the use case diagrams, we focused on black box functionality; this means that we focused only on the externals of the system. In the diagram an Actor represents an external role that interacts with the system and trigger use cases. These Actors interactions helped us define our System Boundary.

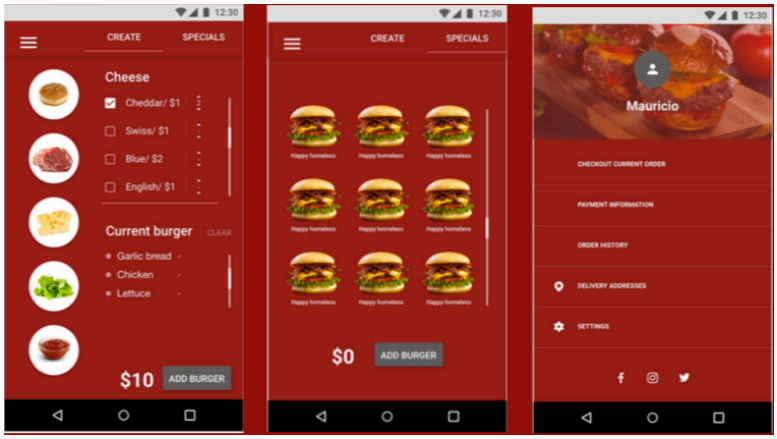


On very early meetings the decision of using a NoSQL database was taken, however the ER- diagram was always essential, as it defined our information structure, by clearly showing us how each system entity relates to each other.

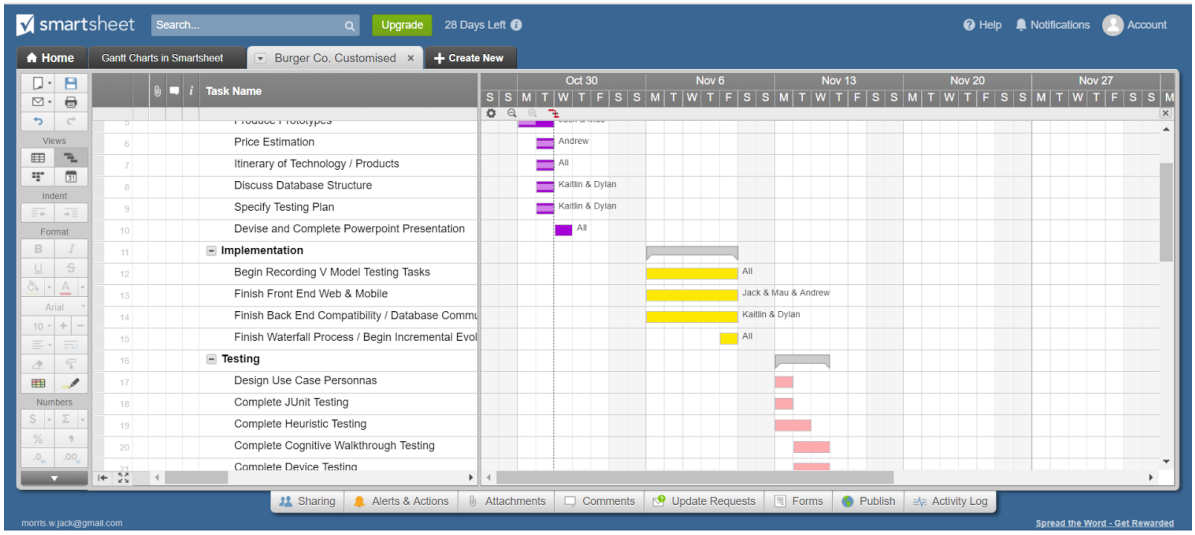


One of my responsibilities on the first week was to produce the mobile app prototype. To achieve this I used Adobe XD, a piece of software used to create great user experiences. This challenging process required multiple iterations over a design. I always kept in mind **Heuristics** principles such as “User control and freedom”, “Error prevention” and “Aesthetic and minimalist design”.





We also did a schedule in a Gantt chart, which is actually really ambiguous and it didn't represent the reality of the process we followed, as we will see later in this document.



Following the completion of these tasks we entered into a **Negotiation** phase that allowed us to **Validate the SRS**.

In the Negotiation phase we noticed that many requirements were not feasible due to our lack of experience with software's technologies and our tight deadlines. We also concluded that there were no unique requirements for either our mobile or web applications. The reason: we wanted the customer's' experience to be as identical as possible in both platforms.

REVISED FUNCTIONAL REQUIREMENTS	
Customer	Staff
C1. Customer shall create a custom burger via web or mobile app using ingredients that are in stock. C2. Customer shall be able to check out with a single burger or add multiple burgers to an order. C3. Customer shall be able to view the price of each individual ingredient before it is added, C4. Customer shall be able to view the current price of the burger with all ingredients added so far. C5. Customer shall be able to remove ingredients from the current burger. C6. Customer shall be able to view the current price of the order C7. Customer shall be able to view the burgers that have been added so far to the order C8. Customer shall be able to register an account and log in with registered account.	S1. Staff shall be able to view the stock of all ingredients. S2. . Staff shall be able to restock ingredients. S3. . Staff shall be able to view orders that are not yet completed with these details: - Customer name. - Ingredients used & quantities used. - Total price. S4. Staff shall be able to update status of orders as they are completed. S5 Staff shall be able to view statistics: - See how many burgers can be created with the ingredients in stock.

REVISED NON FUNCTIONAL REQUIREMENTS

Q1. Web and mobile apps shall each be responsively designed to work on a variety of devices.

Q2. There shall be stylistic and design consistency across produced apps.

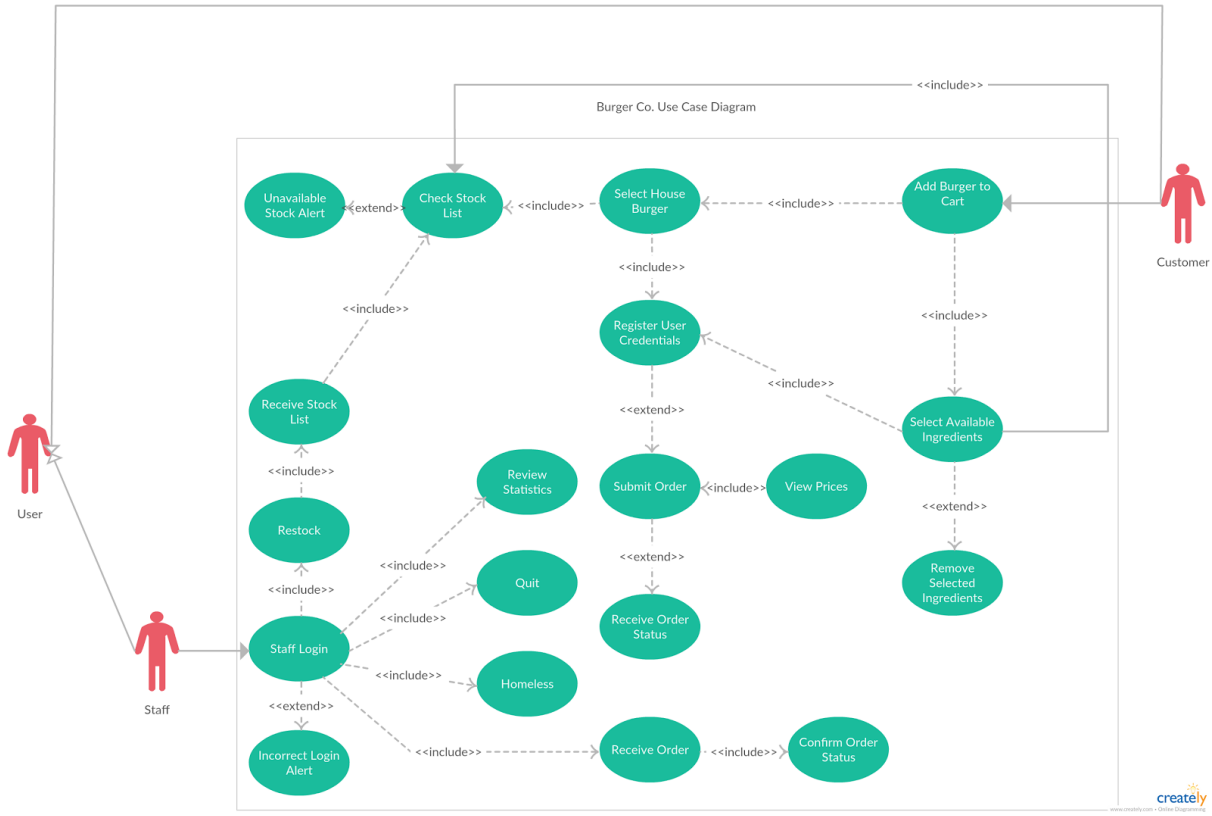
Q3. Interfaces shall be intuitive which enable customers and staff to understand how to use them without external instruction.

Q4. Interfaces shall respect and adhere to Nielsen's Heuristics.

Q5. Passwords shall be stored using up-to-date security such as hashing, salting and key stretching.

Q6. The database shall be backed up regularly in case of unforeseen failure.

After validating the requirements we were also able to do a refined Use Case diagram that shows "include" and "extend" relationships. Include relationships are used for common patterns that need to be factored out, and extend for variations of the system, that not necessarily occur every time.



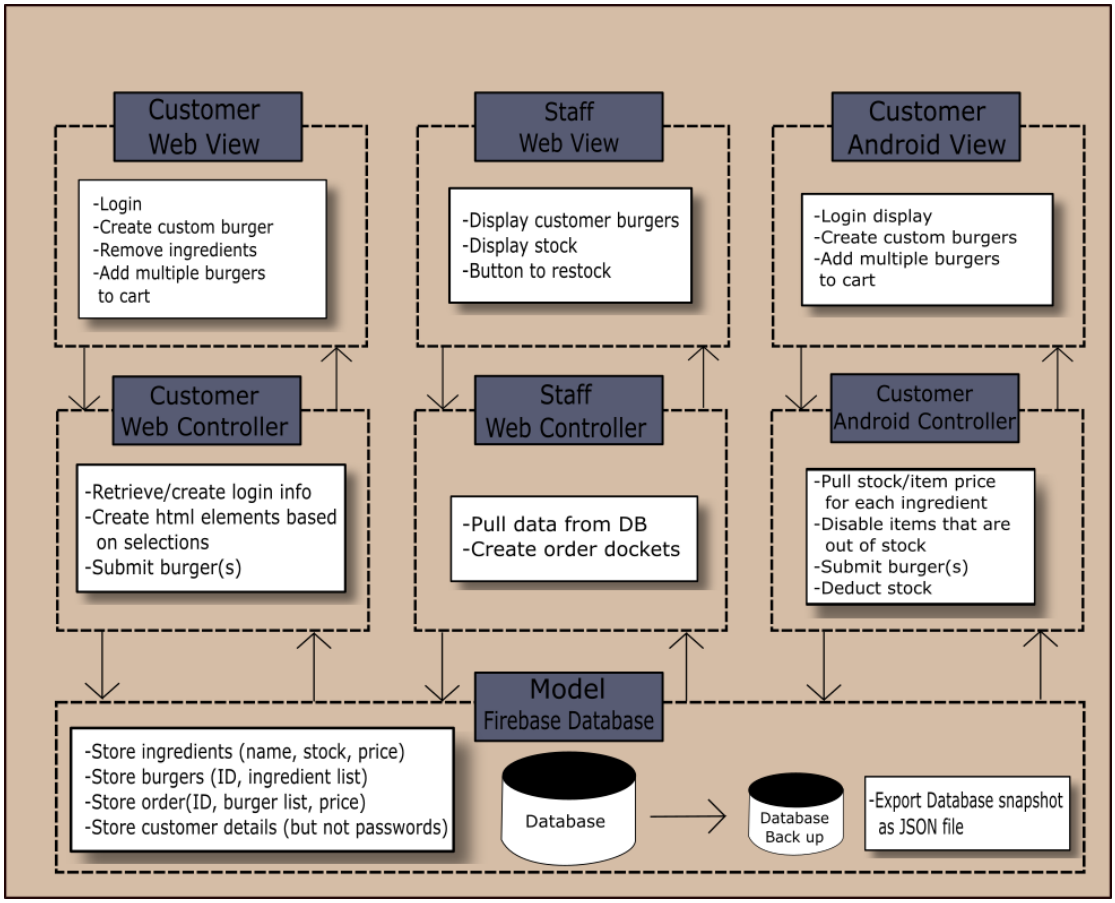
OVERALL SOFTWARE ARCHITECTURE

To achieve an architecture overview we decomposed our system into subsystems (**components**), and its connections (**interfaces**).

We aimed to maximize cohesion and minimize coupling. **Coupling** being the degree of dependency between the components, and **Cohesion** referring to how related are the things that we put in each component.

We also tried to keep only high-level components, making sure that they performed a major service.

The architecture design pattern we followed was **Model-View-Controller (MVC)**, as we found it representative for app development. The Model gets and manipulates data, connects to our database, and communicates with the controller. On the other hand the Controller receives the user input and gets data from the Model; meanwhile the View function is to display data.



TECHNOLOGICAL DECISIONS

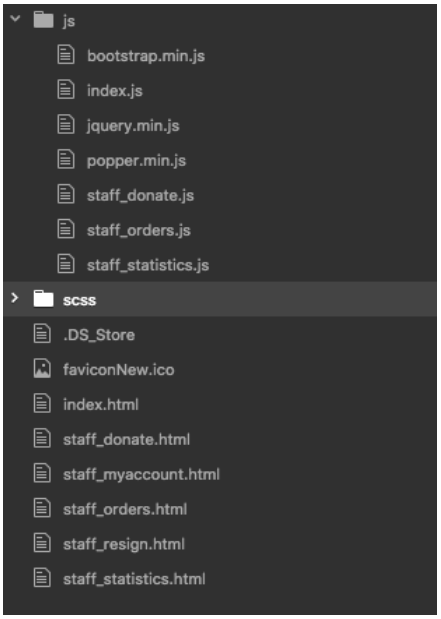
We used technologies that weren't taught in class such as Firebase for our server and NoSQL database. Also we did not use any version control tool, instead we shared our code using Slack.

Full list of technologies used
martsheet - management planning, including Gantt chart
Creately - UML diagram creation, including use case diagrams
Adobe Illustrator - high fidelity web page prototypes
Adobe Experience Design - high fidelity mobile app prototypes
Android Studio - mobile development
Slack - communication and file sharing
Google Docs - documentation of daily tasks, draft designs and schedules
Google Slides - slideshow presentations
Firebase (JSON and NoSQL) or XAMPP (MariaDB and PHP) - database and server communication
Atom - writing HTML, CSS, and Javascript code
JQuery - developing dynamic web pages
Node.js - Server side javascript
Sass - a scripting language that is interpreted or compiled into Cascading Style Sheets
BootStrap: A front-end web application framework important in establishing responsive design.
JQuery: A JavaScript library that provides accessible functions for a variety of purposes.

ABOUT MY PART

I was a front-end developer. In the first week I mainly developed an interactive mobile app prototype; and from the second week onwards I focused into the development and deployment of the Staff web application.

Developing the web application was a huge challenge. I had zero experience with any web development technology; furthermore this project was ambitious and it wouldn't have been possible to create using only the technologies learnt in class. So besides learning HTML, CSS and JavaScript, I also had to learn frameworks and technologies such as JQuery, NodeJS, Sass and Bootstrap.



From all the files, the one that deserves our attention is *staff_orders.js,* the reason: it is by far the most complex and also implements functions common to other files. This file has two main purposes:

1. Displaying the Stock, with an option to Restock

2. Showing the orders and being able to change their status.

The following image shows all methods signatures with comments.

```
1   // Waits for the HTML document to be ready and
2   //then calls the function getOrders() and listStock()
3 › $(document).ready(function(){…
15
16  //Retrieves the quantity of each ingredient
17 › function getQuantity(ingName){…
22
23  //Retrieves the list of all ingredients in a category
24 › function promiseListCat(selectCat) {…
37
38  //Retrieve a list of ingredients, with their quantities
39 › async function listStock() {…
84
85  //Restocks to a minimun of 50 each ingredient
86 › function restock(){…
110
111 //Gets and displays all not completed orders
112 › async function getOrders(){…
193
194 //Changes the status of an order to Complete
195 › function completeOrder(orderKey) {…
198
199 //Retrieves an order
200 › function orderPromise(){…
207
208 //Retrieves the status of an order
209 › function orderStatusPromise (thisOrdKey){…
214
215 //Retrieves the name of the Customer order
216 › function orderCustomerPromise(thisOrdKey){…
221
222 // Restrives a burger list from an order
223 › function getBurgerListPromise(thisOrdKey){…
228
229 //Retrieves a burger, this will let us access to the ingredients
230 //and quantities
231 › function getBurgSnapPromise(thisBurgKey){…
238
```

This file was done with peer programming with Kaitlin. I was in charge of the connection to the DOM and the proper rendering of information, while she dealt with the database connection. Personally the biggest challenge was the **async** nature of JavaScript. I was dealing with a new kind of programming and very soon I was stuck in a Callback Hell. Only after three days of researching we came to Async Heaven, using **promises** and **awaits**, functionalities from ES7 that allowed us to work with async code that actually looked synchronous.

It is worth noticing that the functions signatures that start with async are the ones that write to the DOM. They await for the promise of getting the information from the database, and only until that promise gets resolved they are send to the DOM.

getOrders() is the longest function, consisting of 80 lines of code(mainly because of html syntax). This function awaits for the results of: *orderPromise(),orderStatusPromise(thisOrdKey),orderCustomerPromise(thisOrdKey), getBurgerListPromise(thisOrdKey) and getBurgSnapPromise(thisBurgKey)* respectively.

listStock() uses *promiseListCat(categories[i])* and *getQuantity(key)* respectively;

As a side note, I also tried to achieve Data Visualization using D3 libraries, however I had a problem with JSON object parsing and I ran out of time.

To be honest, for me **testing** occurred simultaneously with development (I designed, implemented and tested). The team did place a particular emphasis on testing, understanding that the product was meaningless unless it was considered and assessed of consumer potential and evaluated as being compatible with the original brief. Sadly I didn't do any testing documentation, however you can look at the documentation done by Jack.

ENGINEERING PROCESS

At the start we intended to use Waterfall to fulfill a Minimum Viable Product (MVP). Once we had our MVC we planned to use an Incremental Evolutionary Model that build upon that core product.

Waterfall seemed appropriate, as it was goal oriented. However, once we established our MVP, waterfall became an obstacle particularly in our flexibility to review our product.

Regarding the incremental evolutionary model we intended to extend our applications with new requirements which we decided concurrent with a development phase.

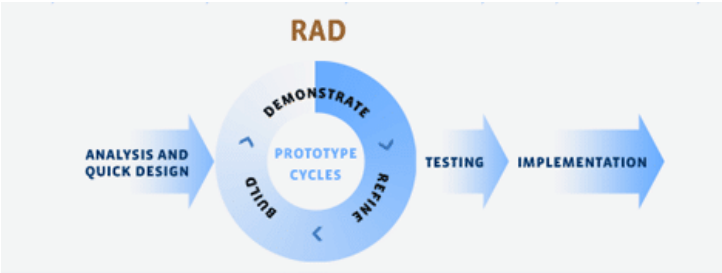
Sadly our reality was quite different and we did not fulfill the Waterfall requirement of providing an in depth SRS document until it was too late. Furthermore our management plan outlined in the Gantt chart failed. We didn't

have defined unique phases, and also did not prioritize to meet the MVP, instead we focused more on non-functional requirements like responsiveness and aesthetic design.

The actual process we followed was RAD; which is almost opposite to Waterfall.

RAD places more emphasis on the process itself and less emphasis on planning. It does not demand rigid specification of requirements and allows requirements to be added as they occur to the team. It also acknowledges and allows that requirements may need to be adjusted if they become unlikely or if new requirements emerge.

RAD, as a process is unique in that it is appropriate for projects involved in designing user interfaces: exactly what our purpose was.



REFLECTION

We would have liked to implement the Waterfall model in its entirety. However we were underprepared to tackle such model. We lacked an understanding of programming technologies and a deep understanding of the processes available for Software Development. We spent the bulk of our time during this project learning and not doing.

Personally I am happy with this project because it has made me learnt heaps of different things. It gave me plenty of practical skills in multiple technologies and also I am starting to grasp with more clarity what Software Engineering is all about. I am confident to apply my knowledge in upcoming projects.

REFERENCES

https://www.youtube.com/channel/UCXu_vmvAKCXgZxm2kN4Nj7A

Software Engineering 10th Edition by Ian Sommerville