جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⴰⵢⵏ

AL AKHAWAYN UNIVERSITY

SCHOOL OF SCIENCE AND ENGINEERING

FOODCOURT AN ONLINE FOOD ORDERING PLATFORM

Submitted in

Spring 2019

By

Hamza Bentahar

Supervised By

Pr. Iraqi Houssaini Omar

FOODCOURT AN ONLINE ORDERING PLATFORM

Capstone Report

Student Statement:

I attest that I have applied ethics to the design process and in the selection of the final proposed design. And that, I have held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable.

Hamza Bentahar

Approved by the Supervisor

Pr. Iraqi Houssaini, Omar

Acknowledgement

I would like to acknowledge my supervisor, Pr. Iraqi Houssaini, Omar who me helped me during the design phase by providing great advice, as well as his assistance for understanding some of the technologies used in this project. I would also like to thank all the professors who made me learn all the required knowledge for this capstone. Finally, I am very grateful to my family who supported me during my stay at the university, and without whom I could not achieve this project.

Table of Contents

Abstract

The purpose of this capstone is to design and implement a web application that lets user order from restaurants online. This project will help users find restaurants that match their needs, other functionalities are added as well, such as the possibility to post a review, and the possibility of checking the whole menu for a given restaurant. Moreover, this application gives the ability to restaurant managers to see current orders, and to update the menu.

This report will show the whole process of creating the application, starting by the design phase, and then showing the final result, by explaining the different technologies used.

I. Introduction

Foodcourt is a web application that helps people choosing and ordering food from nearby restaurants, this should be achieved by implementing a search functionality along with options to sort and filter results. The user selects a restaurant of his or her choice and browse through the menu before proceeding to the order. This will help people discover new restaurants, have a larger choice of menus, and save time by ordering online.

The application should also give restaurant managers a platform for processing incoming orders, as well as, a way of communicating with their customers. Every new order should appear in the manager's dashboard. Managers should be able to modify their menus, publish a description for their restaurant, and upload pictures.

This application should be accessible through the most popular web browsers in computers, tablets or mobile phones.

II. STEEPLE ANALYSIS

A. Social Impact

The aim of this project is to help people choose their meals more accurately, by having access to a larger choice of restaurants and menus. In order to make the process of selection easier, a review system is needed to let users give feedbacks about restaurants. These features may be beneficial to tourists who do not know the best restaurants in the area. The online ordering feature may help workers and students order their lunch online.
This application would also help managers to better promote their restaurants.

B. Technological Impact

To build this application, new technologies and tools would be used. These technologies are open-source, and will be used to complete the project in the most efficient way.

C. Environmental Impact

This application has very little environmental impact.

D. Economic Impact

With this application, restaurants may attract more customers, which will increase their revenues. Since the main feature of the application are free for both the users and the managers, there is no potential loss for any of them. Some additional features might be added afterwards, to let managers have access to statistics about their restaurants, with the purpose of helping restaurants boost their revenues.

E. Political Impact

This application is not intended to have any political impact.

F. Legal Impact

Since this application uses only free opensource frameworks and libraries, it will have no legal impact.

G. Ethical Impact

Ethically, it is extremely important to secure the application to avoid any data leak. Any new feature should be tested to avoid vulnerabilities, passwords should be encrypted and stored securely in a database.

III. Feasibility study

The feasibility study is an important phase during the development of any project, its goal is to determine whether the project is doable or not. My capstone project is about building a web application, which purpose is to help people in choosing their favourite food from nearby restaurants, and giving the ability to restaurant managers to better communicate with their customers.

First, the technical feasibility is to understand if it is possible to complete the project with the current technologies. This application is going to use many programming languages and frameworks in order to ensure a good user experience for the end user, as well as adopting good coding practices for the developer. The structure of the application will consist of a backend and a frontend. The backend will be implemented using NodeJs, its purpose is to handle database queries, authentication, and to serve an API (application programming interface). The frontend should be completed using HTML, CSS, and javascript. On top of javascript, I will use vuejs framework, which is going to make the pages more reactive and help in building a single page application (SPA).

Second, the temporal feasibility is crucial to make sure that it is possible to complete the project on time. There would be a development phase in which I would have to code the application with the chosen tools. Also, there will be a learning phase, during which, I will have to read about the technologies that I would be using for this project. Since I am already familiar with some of these technologies, thanks to the courses I have completed and to my previous internship experience, I will only focus on the ones I do not know.

Third, the economic feasibility is essential to know the budget needed for the completion of the application, and how much income it would be able to generate once released. The necessary budget for this project is low, the technologies needed for this project are free to use. Regarding the IDE, I will be using WebStorm, which is excellent for javascript development, it is a paid software but free for students. This application can be monetized using different plans. The first one consists of displaying advertisements to the web application by using services such as google adsense. The

second technique is to get a commission for every order, which means that every time the user orders something from a restaurant, a small fee should be paid for the maintenance of the application. The third technique would be to adopt the freemium model, the application would have a free tier, which the users and the managers would be able to use without cost, and in addition, a paid tier that is going give more functionalities to the restaurant manager.

IV. Requirement specifications

A. Functional requirements

1. Users

- A user shall be able to sign up using his/her email address or Facebook account
- A user shall be able to sign in using his/her email address or Facebook account
- A user shall be able to search for restaurants by category, city, and name
- A user shall be able to sort restaurants by nearest, most popular, and top rated
- A user shall be able to filter results by delivery type, category, and neighborhood
- A user shall be able to get more information about a specific restaurant such as description, opening hours, address, and pictures
- A user shall be able to grade and post a review about a restaurant
- A user shall be able to view the average grade of a restaurant and reviews from other users
- A user shall be able to view the restaurant's menu
- A user shall be able to select items from the restaurant's menu
- A logged user shall be able to make an order with the selected items from the restaurant's menu
- A user shall be able to view his/her orders

2. Managers

- A manager shall be able to sign up using his/her email
- A manager shall be able to sign in using his/her email
- A manager shall be able to add a restaurant
- A manager shall be able to add a menu to his/her restaurant
- A manager shall be able to process the received orders
- A manager shall be able to view his/her customers
- A manager shall be able to change his/her restaurants information

B. Non-functional requirements

1. Performance

- Initial load time should not exceed one second
- API requests shall not exceed 500ms

2. Scalability

- The increasing number of users should not affect the performance of the application

3. Extensibility

- New features shall be easy to implement with separation of concern

4. Security

a) Confidentiality

- Traffic confidentiality shall be protected, all operations performed by users must be preserved

b) Integrity

- The integrity of all operations performed by users must be preserved

c) Availability

- No single point of failure shall be tolerated

C. Use case diagram

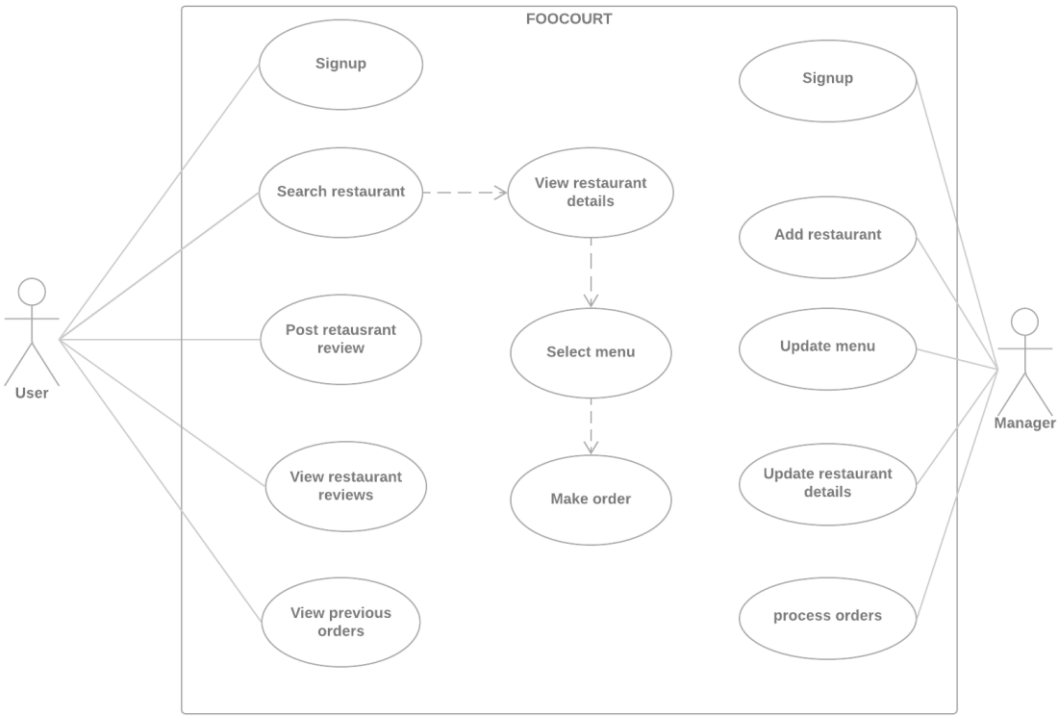

Figure 1 Use case diagram for foodcourt

V. Technological enablers

A. Languages and frameworks

For this project, I decided to use javascript, as well as HTML and CSS. Hypertext markup language (HTML) is a markup language necessary for building a web application, it is used to describe the structure of a web page [7]. A cascading style sheet (CSS) should also be added for a better design [8]. While HTML is used for structuring the web page, CSS is used for styling. It enables to have complete control over the colors, fonts, and other important aspects of web design. In addition to those two languages, I used javascript, which makes the page more dynamic and interactive.

At first, when javascript was released in 1995 [9], it was used in the client side and interpreted by web browsers for the manipulation of the document object model (DOM) [10]. The language has improved a lot over the years, it can now be used outside of the browser with NodeJS, which is a runtime environment for javascript [11]. Released in 2009 [12], NodeJS lets developers write javascript code for the backend. In order to make the implementation easier by following the best practices and to have a better user experience, I decided to use two frameworks: VueJs for the frontend, and Express for the backend.

B. VueJs

VueJs is an open-source framework for javascript, first released in 2014 by Evan You. It is used for building single page applications (SPA), which is a way of interacting with the webpage by dynamically changing some parts of the page instead of reloading the entirety of it. This approach makes a better user experience by avoiding the constant interruptions during navigation. Even though it is possible to build a single page application using only javascript, it is better to use a framework, vuejs in this case, to achieve a better result with fewer lines of code.

The way VueJs works is by building small components, which are then assembled together for building views. Components are composed of three parts: a template that contains the HTML, a script that contains the javascript code, and a style which holds the CSS. The specificity of these components is their reactivity, the data is reflected on the template, making it re ender when the data is updated.

Sometimes, components need to communicate with one another, there are different approach of achieving this. One approach would be to pass data from a parent component, to a child component by using props, similar to html attributes. Another approach is to use an event bus, which is a way of communicating between unrelated components, this works by making a component subscribe to an event, so every time that event is fired, the component is notified about it. Even though the event bus way is a good way of sending and receiving data from a component to another, it can quickly become messy and difficult to maintain. The third approach is to use a state manage library, in my case I used VueX, which is maintained by the core team of VueJS. This library is used as a centralized store for all components in the application, the data is stored in it as well as the logic, any component can have access to the data and manipulate it while ensuring that the change is consistent.

VueX is one of the many libraries of VueJs, another library that I used is Vue-router, which is also an official VueJs library used for routing. This works by mapping components to paths, so that when a user accesses a specific URL, it displays the correct view.

C. NodeJs

Nodejs is a runtime environment for javascript, it was first release in 2009 to run javascript code on the server side. The particularity of NodeJs is its single threaded nature, which can be a drawback, but thanks to the non-blocking I/O, it can support thousands of concurrent connections. On top of NodeJs, I also used expressjs, which is a micro framework that helps providing the necessary features for building APIs.

For this project, I decided to use GraphQL, which is an open-source data query and manipulation language for APIs, it was used by Facebook internally before making it publicly available in 2015. As said in the official Graphql website: “GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.” [13]. Contrarily from a Rest API, GraphQl has a single endpoint, and returns only the requested fields, no more, no less. This way, we can get many resources from a single request, which helps in achieving better performance while saving data. Figure 5 shows an example of a request using GraphQL.

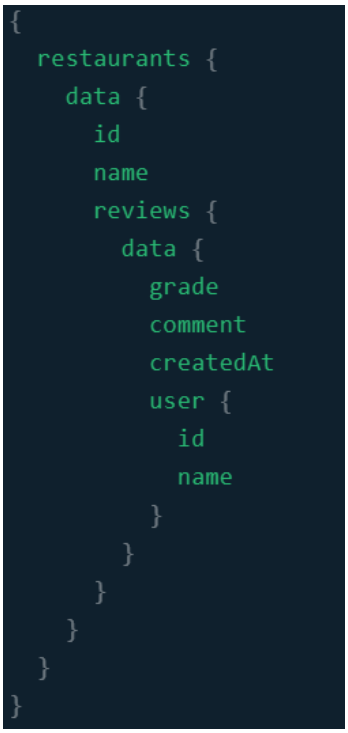

Figure 2: Example of a graphql request

In figure 5, GraphQL is requesting a list of restaurants, their reviews, and the users who posted these reviews. Figure 6 shows a possible response of the request.

Figure 3: Possible response of a graphql request

This project follows a model-view-controller (MVC) architecture, with the view being generated by vuejs, and the model/controller by NodeJs. However, because GraphQL needs only one endpoint, we will need only one controller which handles all requests.
The controller calls a GraphQL schema, which contains several type definitions and resolvers. Type definition represents the structure of the application by defining object types and their attributes. Resolvers deals with the internal logic of the application to retrieve and modify data. There are two types of resolvers:

- Queries: Used for data fetching.
- Mutations: Used for data manipulation.

D. Database

As a database, I decided to use MongoDB, which is a non-relational database management system (DBMS). Because MongoDB is a NoSQL (not only SQL) database, there is more freedom for storing and retrieving data by accepting numerous data types. Not only we can store strings, numbers, and dates like other relation databases such as MySQL, but also Arrays and Objects. Thanks to this, retrieving data becomes easier and avoids the use of complex queries.
MongoDB consists of collections [14], which is the equivalent of a table in a relational database. Every collection contains documents, which are of types BSON [15], a binary representation of JSON (javascript object notation). These documents hold the data, similarly to a row in a relational database. However, these collections do not enforce a schema, which gives a great flexibility of storing data, but with the drawback of dealing with persistent data storage, which may lead to future bugs and errors. This is one of the reasons I chose mongoose as an ORM, it enforces the use of a schema, and makes the communication with the database easier.
One of the reasons I decided to use MongoDB is its great support for common operations in web applications. In the case of Foodcourt, there is a search bar to search for restaurants, the result should be sorted by relevance. MongoDB gives the possibility for search queries [16] by indexing the attributes to search. While performing the search, MongoDB gives a relevance score for each document, depending on how well it matches the query, then the result can be sorted from the highest score to the lowest.

VI. Software Architecture

The software architecture describes the different components of the application, and the relation between them. Foodcourt follows a model – view – controller (MVC) architecture [4]. The view is generated in the browser using Vuejs, which communicates with the backend. Since I am using GraphQL for this project, there is only one controller. That controller has access to GraphQL schema [5], which in turn, calls the right mutation or query [6]. The model takes care of accessing and retrieving the data from MongoDB.

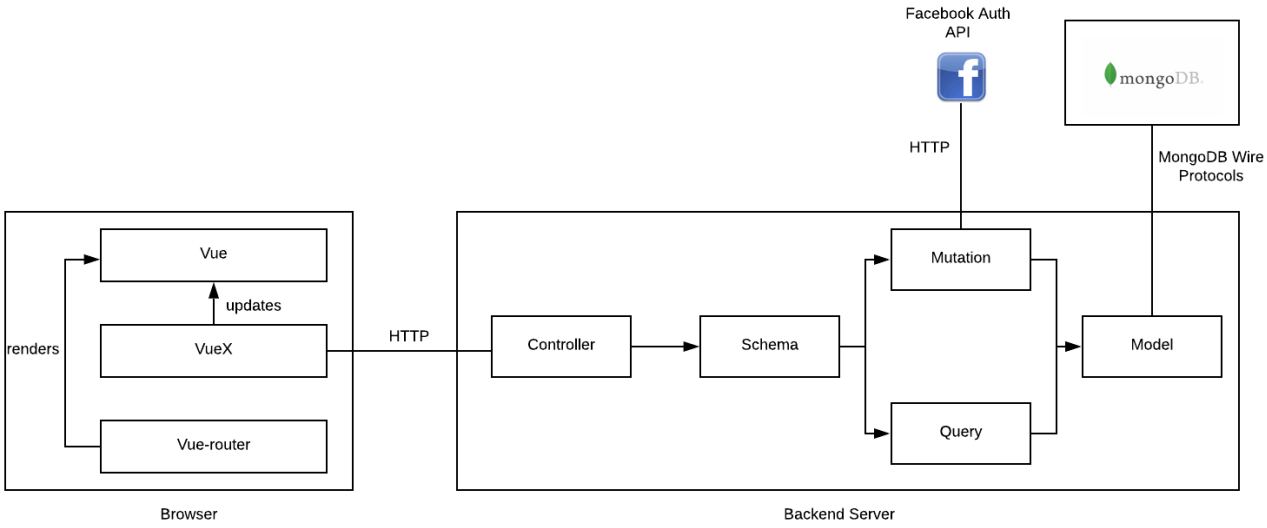

Figure 4: Software architecture for FoodCourt

VII. Software Design

A. Class Diagram

The class diagram is used to describe the models needed to build the application. There are ten classes, each of them has a relation with one or more other class. There are two types of relations:

- Bi-directional association [1]: This is used to show that the class is aware of the other class by adding a reference to the other class. The representation of this link is similar to the relationship between Order and Product (Figure 2).
- Composition aggregation: This means that the class is part of the other class, the child cannot exist without the parent. The representation of this link is similar to the relationship between Restaurant and Opening (Figure 2). The arrow part points to the child, the other part points to the parent. In this case, Opening is part of Restaurant.

While building this class diagram, I got into this issue about when to use the bi-directional association, and when to use the composition aggregation. For classes that needs to be accessed from multiple other classes, and for classes that may grow very large, it is better to use the bi-direction associations. For small classes that have a link with only one other class, the best solution is to use the composition aggregation [2].

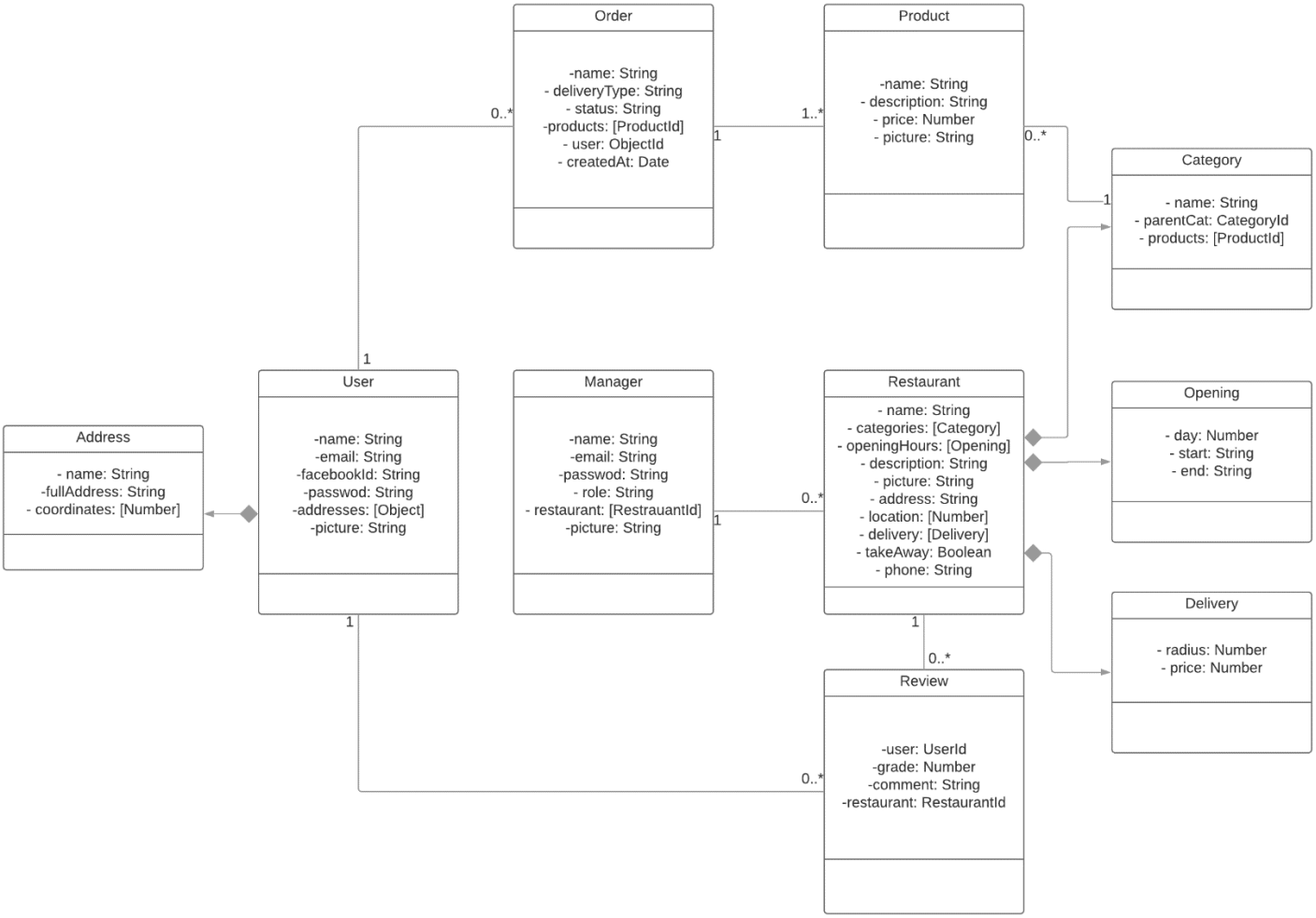

Figure 5 Class Diagram for Foodcourt

B. Sequence Diagram

The sequence diagram [3] shows how the user can interact with the application, by specifying the order of action, along with the possible failures and the expected response. In figure 3, the sequence diagram shows the necessary actions that a user needs to make to complete an order. The user has to search for a restaurant using keywords, then the server returns a list of restaurants that matches those keywords, the user selects one of those restaurants and submit his order, which is then processed in the server. Afterwards, there are two possibilities, either the request is submitted successfully, or the request fails.

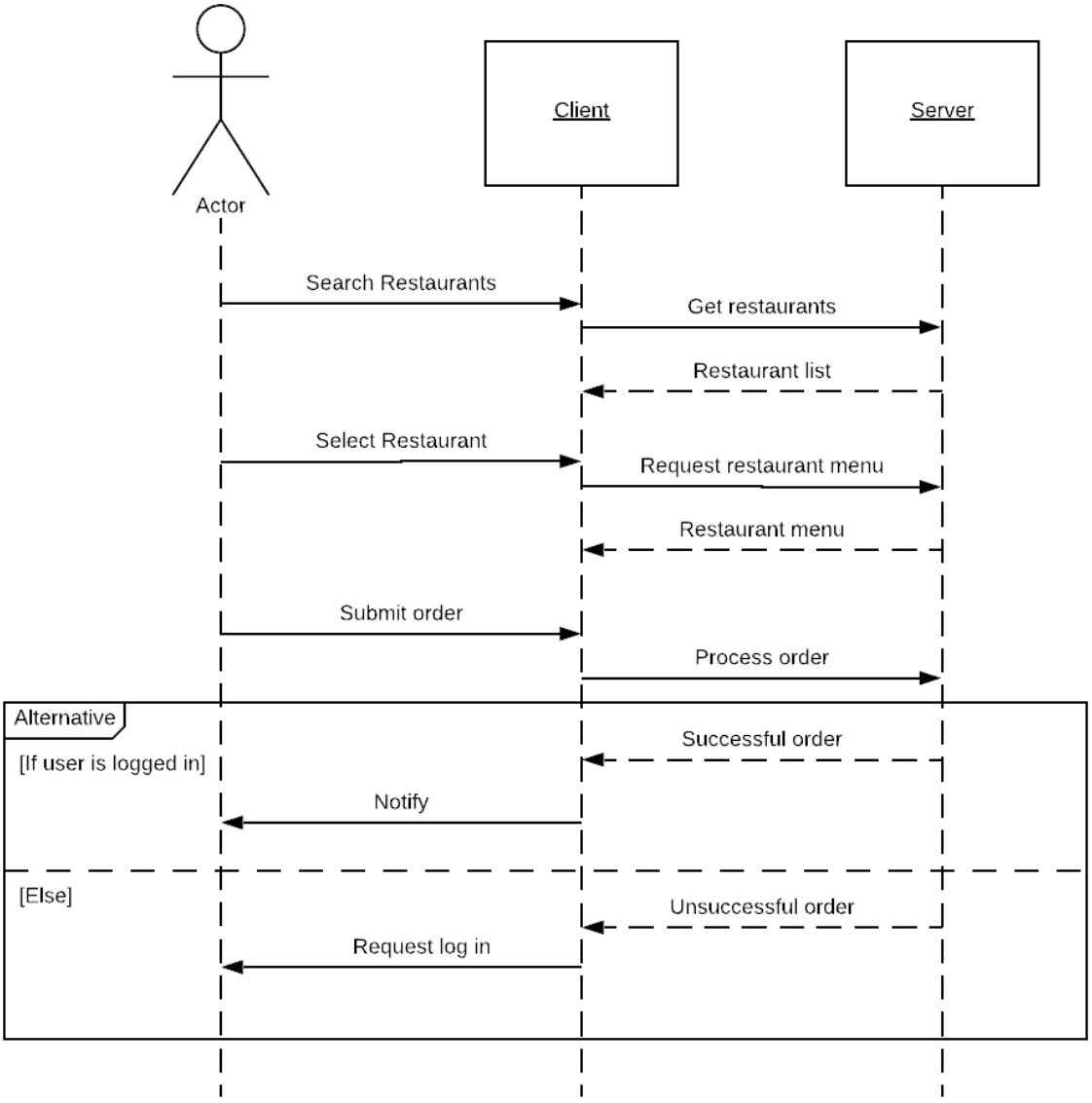

Figure 6 Sequence Diagram for Foodcourt

VIII. Implemented features

A. User side

1. Authentication

Every user can search restaurants and navigate through them; however, to make an order, the user needs to be authenticated. In order to implement the authentication, I used json web tokens (JWT), which is a group of encoded json objects in base64URL assembled together [17]. Every object contains specific information needed to achieve the information by making it consistent and secure. A json web token contains a header, a payload and a signature, separated by a dot "." as follow "header.payload.signature". Following is a description of each part:

- Header: Holds the necessary information to describe the token. It consists of two parts, the first one is the type of token, in this case "JWT", the second part is the algorithm used, such as SHA256 or RSA.
- Payload: Contains the actual information, in the case of foodcourt, the payload contains the information regarding the user, such as the name and the email.
- Signature: Generated by combining the encoded header, the encoded payload, and a secret key, then sign it using the algorithm declared in the header. This way, we can verify that the data was not changed.

After the user logs in the application, a json web token is generated, then stored in the local storage. When the user makes an action that needs authentication, the token is sent to the backend with the header, which checks if the token is valid using the secret key.

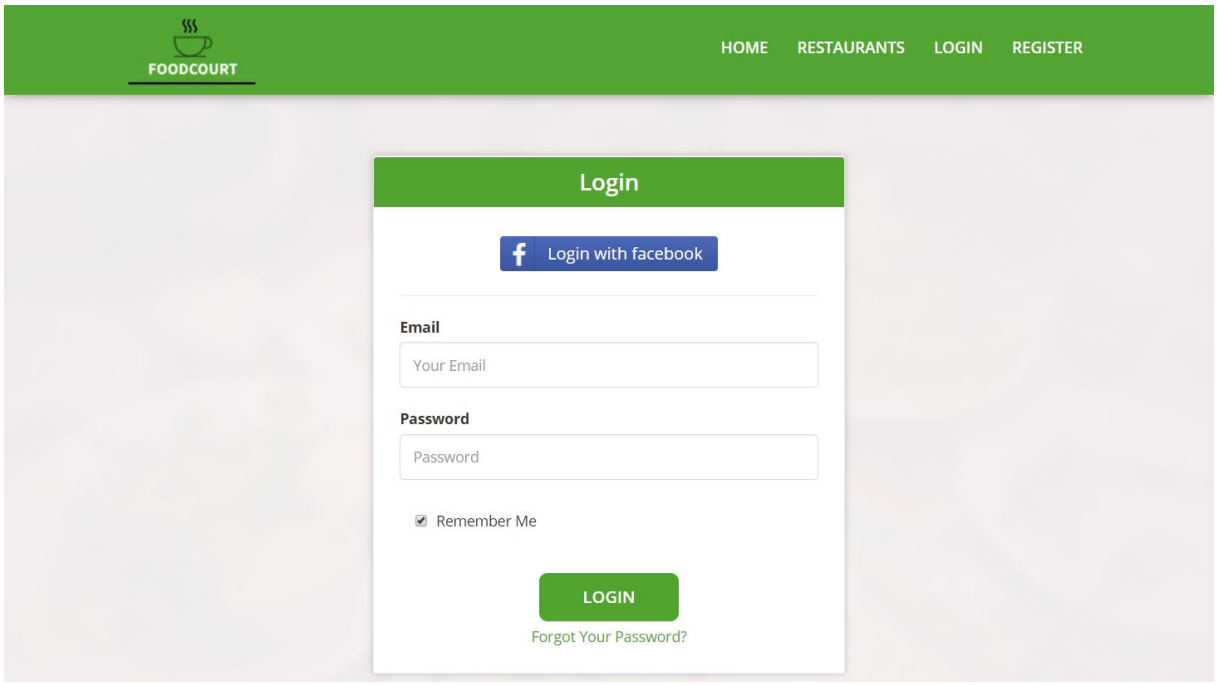

Figure 7: Foodcourt login page

For the registration, the user has the choice of signing in using his email address or his Facebook account, as show in figures 7 and 8.

- Email: If the user decides to sign up with his email address, he will be asked to fill up a form with his name, email, and password. For security, the password is hashed using bcrypt algorithm and stored in MongoDB.
- Facebook: If the user has a Facebook account and wants to login, he or she can authorize Facebook to give his email, name, and FacebookId, which are going to be store in MongoDB for future use. This way, the user does not have to choose a password, making the process of registration easier.

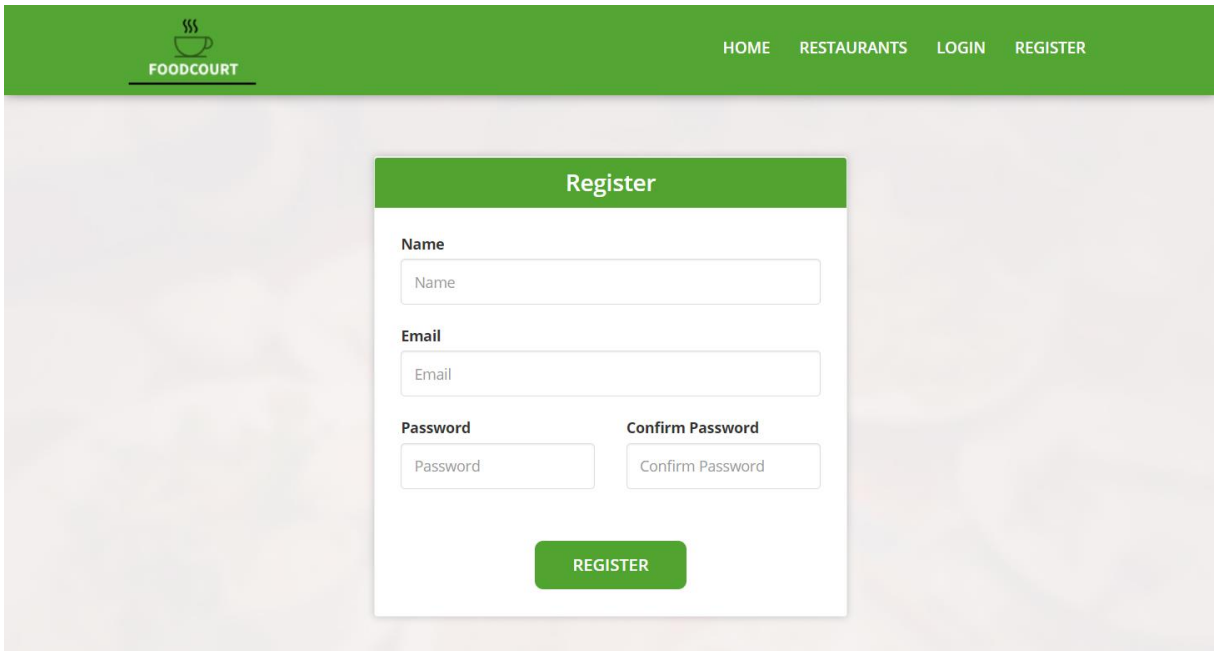

Figure 8: Foodcourt register page

2. Restaurant Search

Search is one of the main functions of the application, the user types a sentence and should find restaurants that best match that sentence. This is achieved by using the text search functionality of MongoDB. The indexed fields are the categories, restaurant name, and location, which means that the user can search for restaurants depending on what they need, and the result would try to best match the search query, and sort the result by relevance. The search bar is accessible from the home page of the application, and does not need the user to be authenticated.

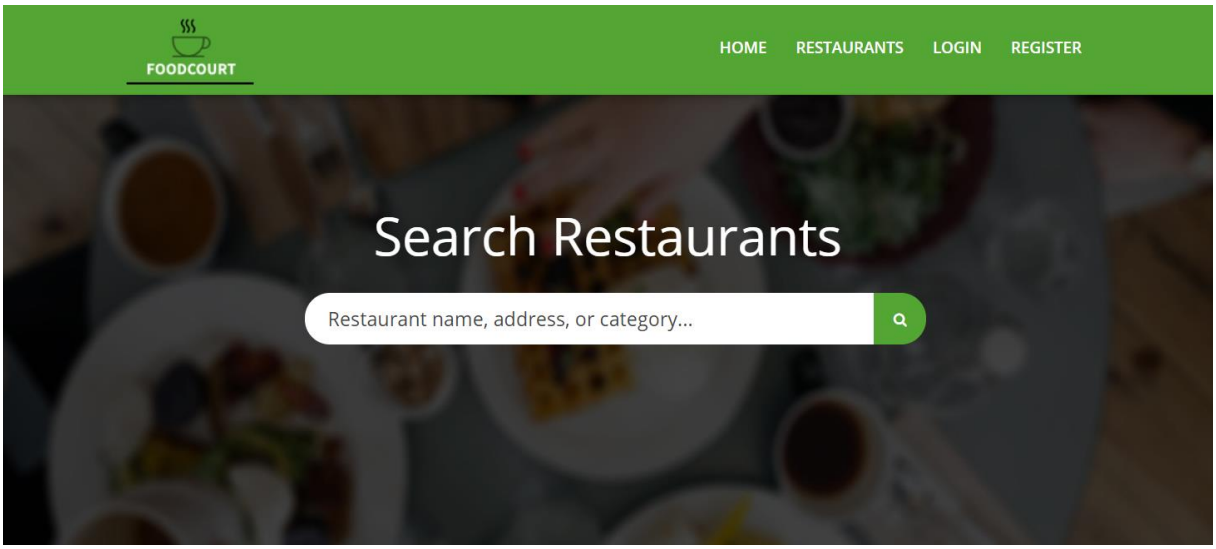

Figure 9: Foodcourt homepage

Once the results are retrieved, they are displayed by page and shown to the user by displaying small details about the restaurant, such as the name, the location, and the average rating. The user can then choose the restaurant that best suits his needs. Restaurants are displayed by page to avoid loading the page with too much information.

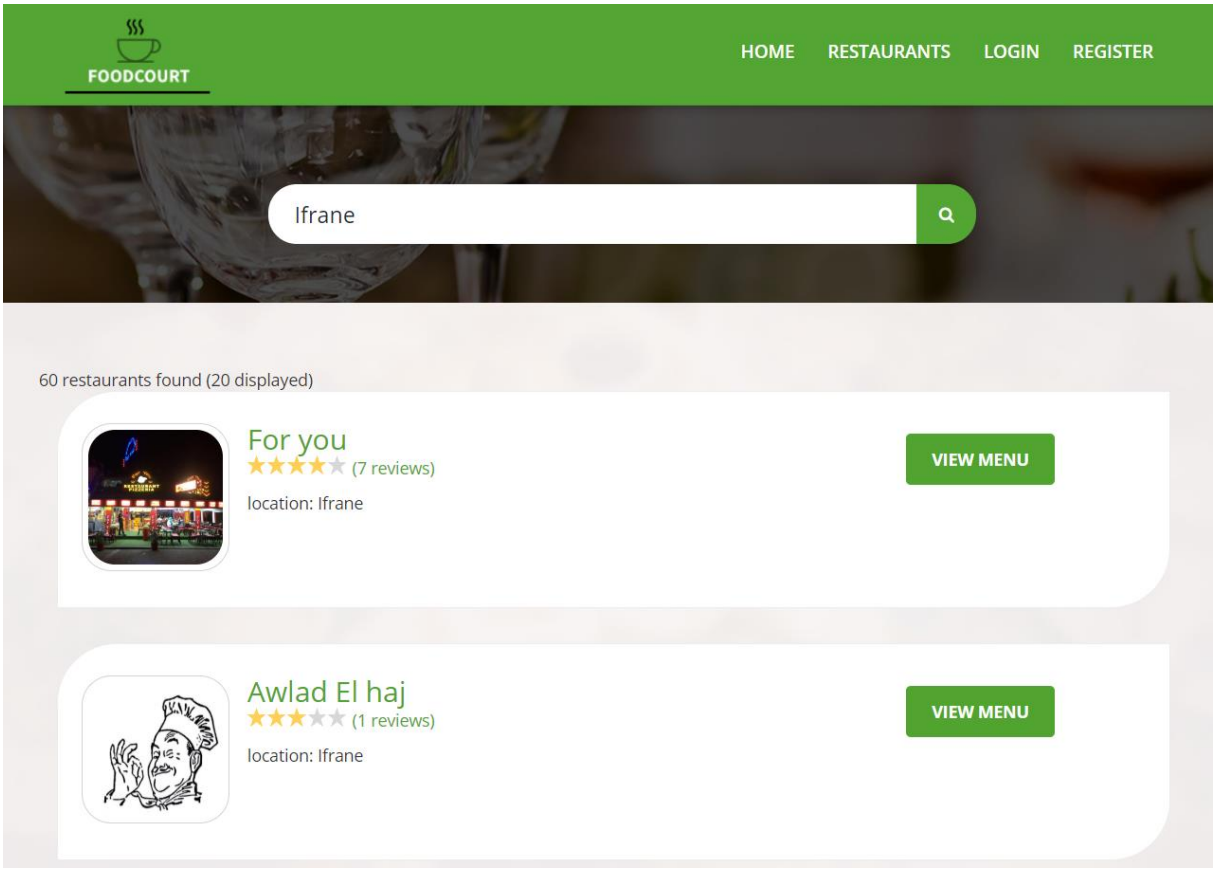

Figure 10: Foodcourt restaurant results page

3. Making an order

After the user decides from which restaurant to make an order, he or she can then choose items from the menu, which contains the necessary information to make the right choice. The price is computed dynamically, so that when the user makes modifications to the cart, the price is computed again.

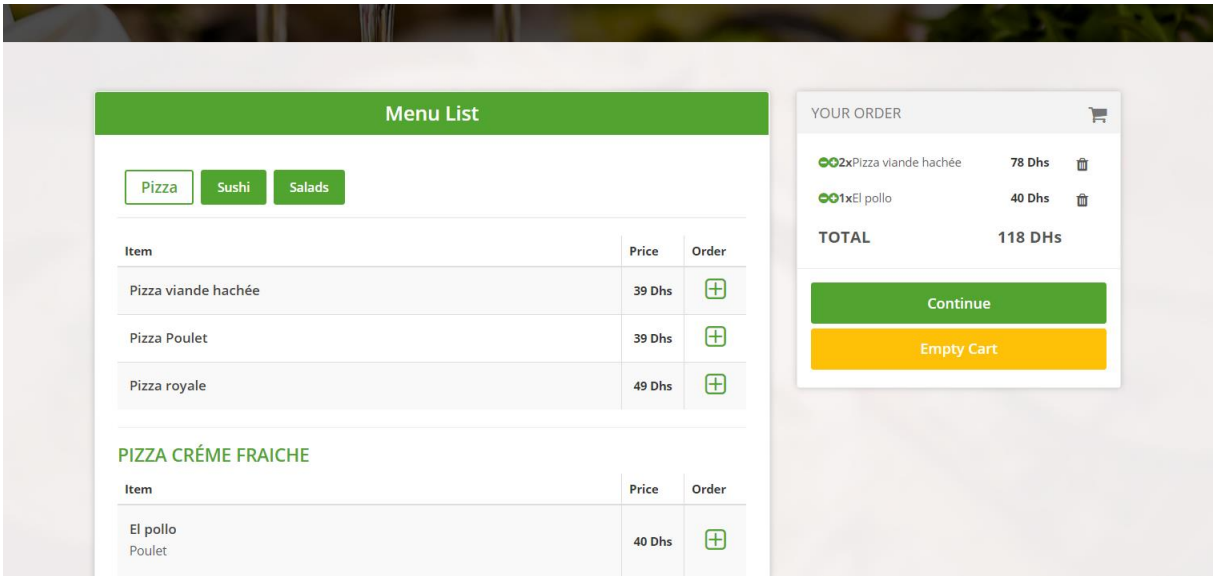

Figure 11: Foodcourt view menu page

When the user is satisfied with the selected items, he or she can then proceed to confirm the order. To complete the transaction, the user must be logged in, and should choose the delivery method, either pickup or delivery. For the latest, the user must provide an address or choose one of the saved addresses. After submission, the order is sent to the manager.

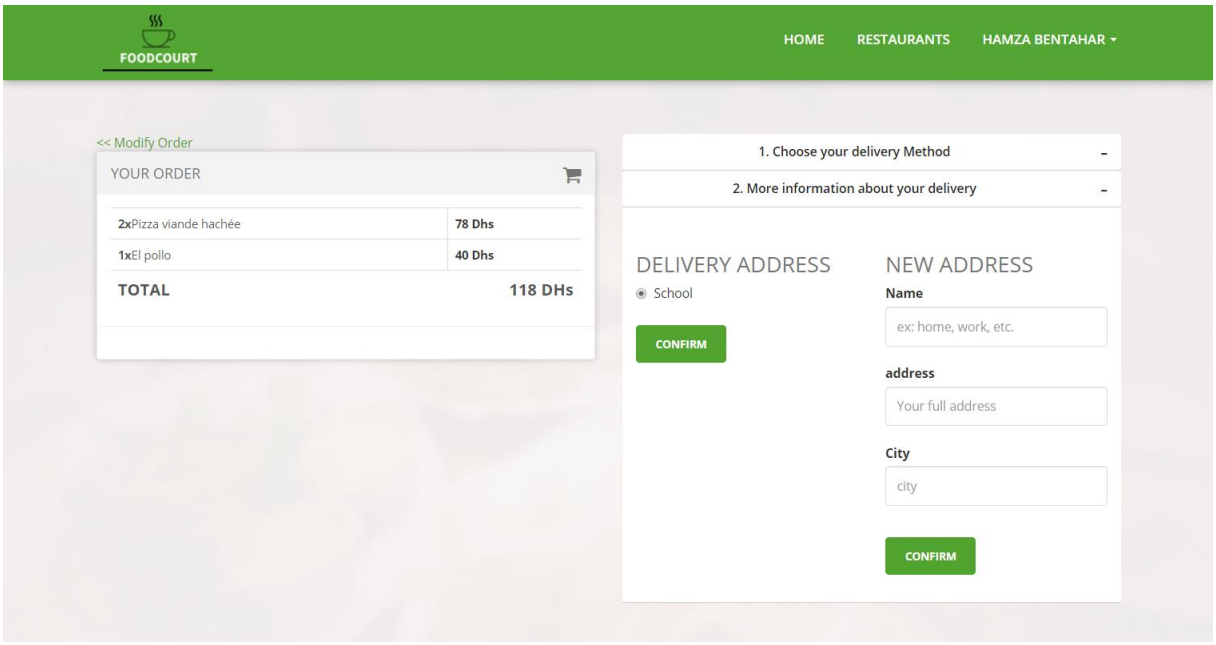

Figure 12: Foodcourt confirm order page

4. Reviews

The review section gives users the possibility to rate their experience for a given restaurant. Other users can then see the overall rating and comments of the restaurant. To post a review, the user must be logged in.

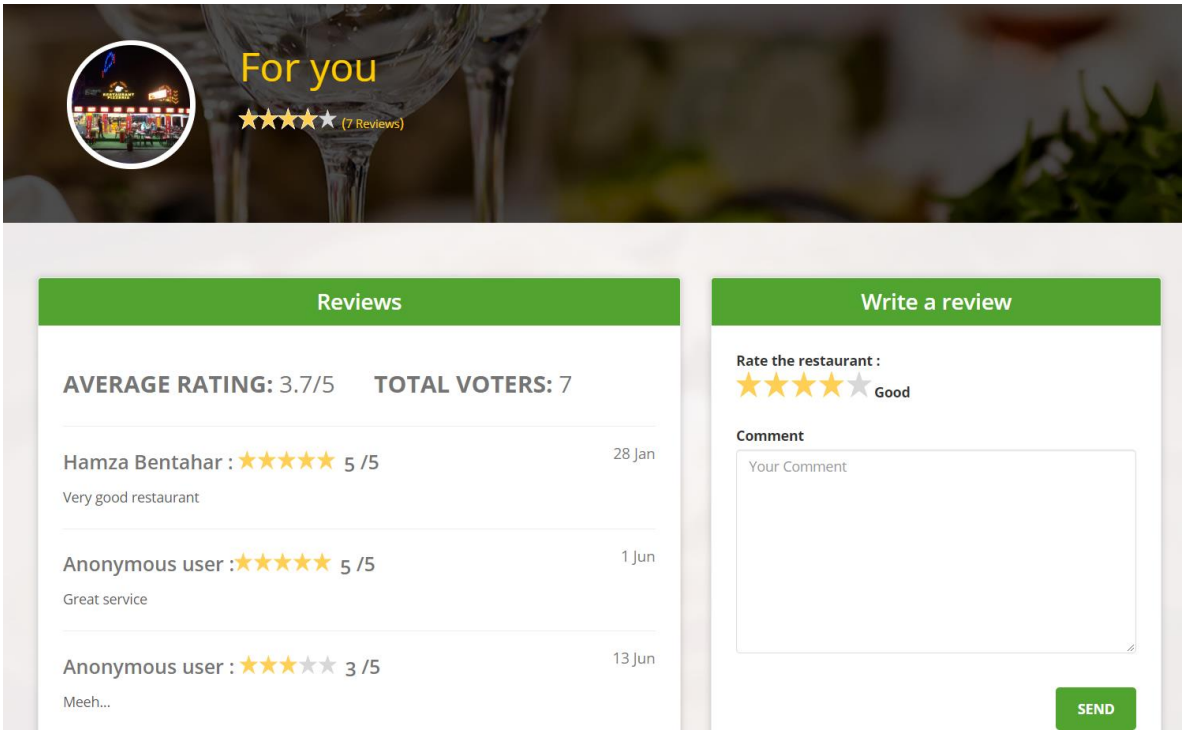

Figure 13: Foodcourt reviews page

B. Manager side

1. Authentication

Any restaurant can create an account and add his or her restaurant to Foodcourt. The process is simple, the manager creates an account, and then add details about the restaurant, as shown in Figure 16. The authentication is similar to the user side, by using json web tokens.

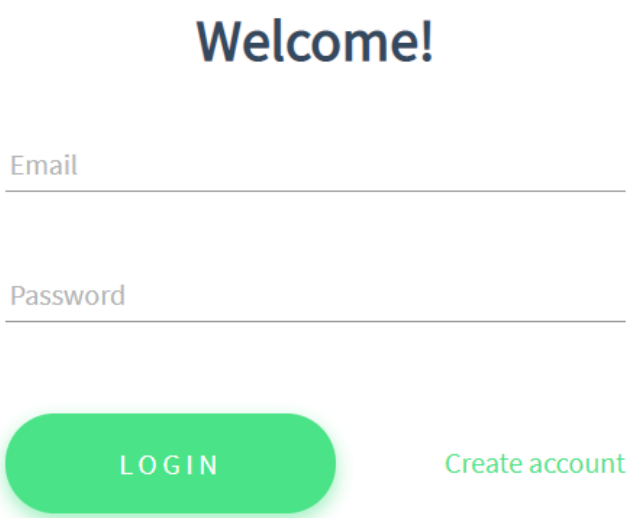

Figure 14: Foodcourt login manager form

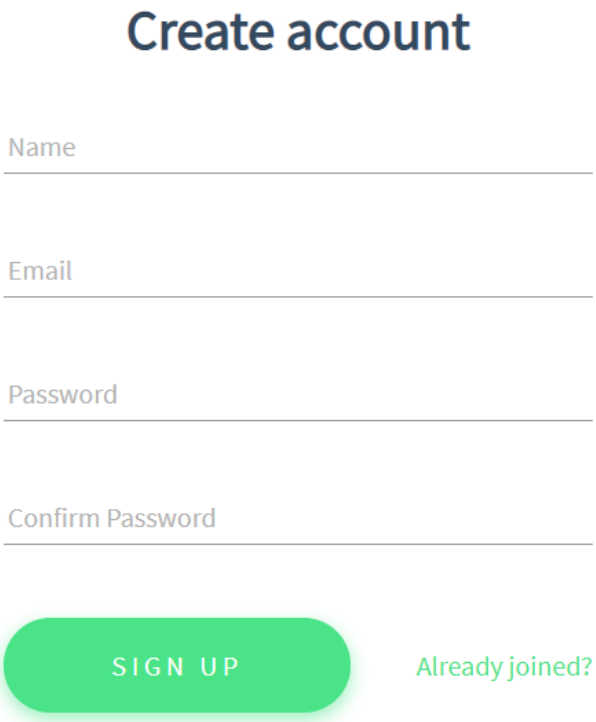

Figure 15: Foodcourt register manager form

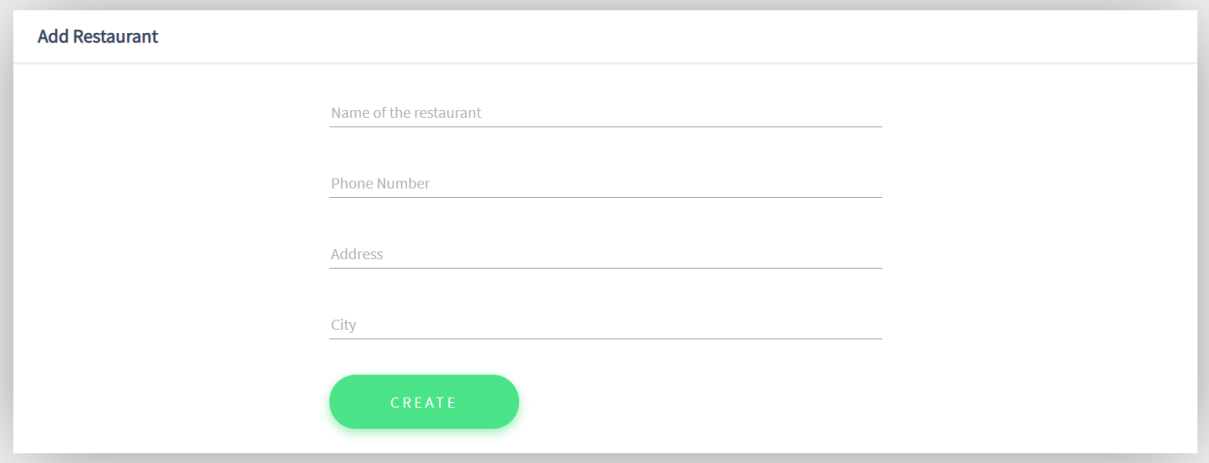

Figure 16: Foodcourt add restaurant form

2. Manage Orders

Once the manager logs in, the first page would display the list of current orders. Each order displays the basic information, such as the delivery type, the status and the delivery address. To get more details, the manager clicks on the order to display more details along with the possibility to update the status.

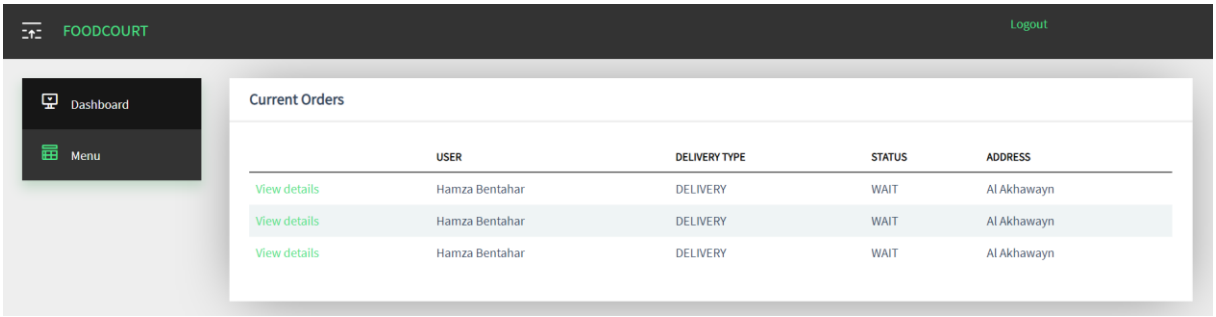

Figure 17: Foodcourt dashboard

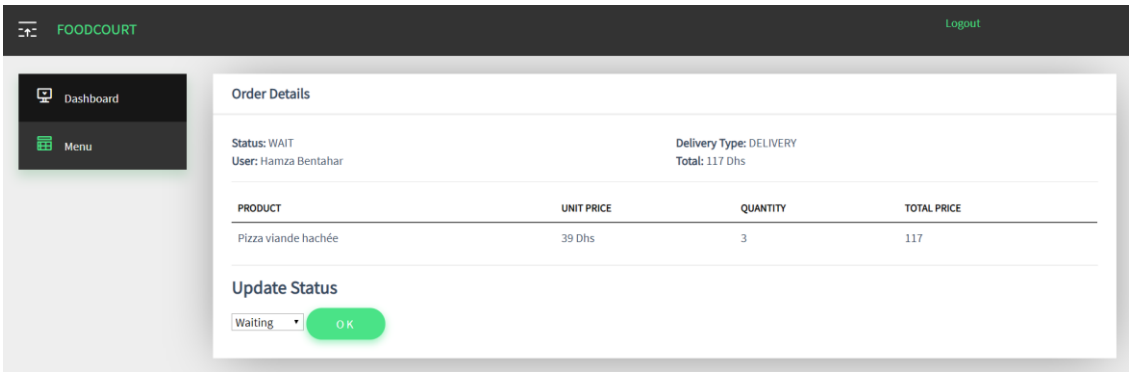

Figure 18: Foodcourt order details page

3. Update menu

This section allows the manager to update the menu by adding more categories and more products.

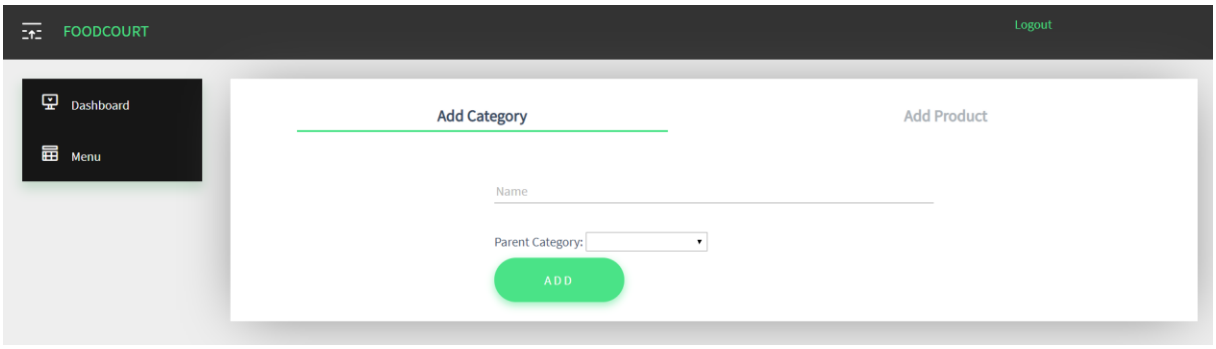

Figure 19: Foodcourt add category form

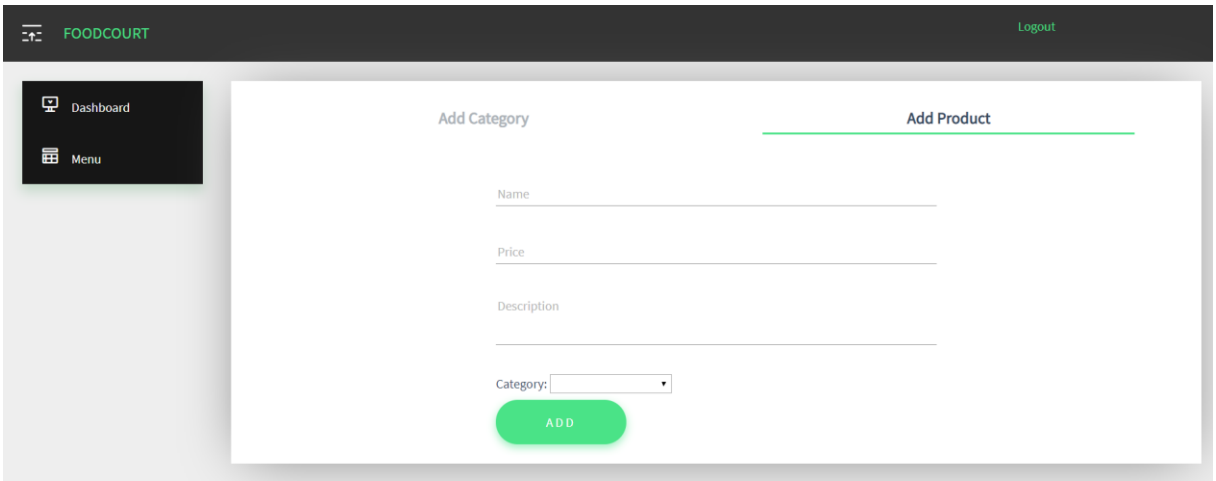

Figure 20: Foodcourt add product form

IX. Conclusion

Foodcourt was an interesting project to work on, I used many great tools to achieve this result. Working with javascript and NodeJs was very interesting, it requires a deep knowledge in programming, and a good understanding of internal features. Working on this project made me learn more about many of the recent technologies such as Nodejs, MongoDB and Graphql.

There are many possibilities to improve this project for future work. For instance, we can add more features on the manager side that will help in better understanding customers, by displaying accounting and financial results. Also, one improvement would be to suggest new restaurants to a user depending on previous choice, by using machine learning algorithms. For now, the customer can pay only by cash when the order is delivered; however, it would be better if the user could pay by credit or debit card.

X. References

[1] “The class diagram,” *UML basics*, 15-Sep-2004. [Online]. Available: https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html#N102A9. [Accessed: 17-Feb-2019].

[2] “Data Models¶,” *Data Models - MongoDB Manual*. [Online]. Available: https://docs.mongodb.com/manual/data-modeling/. [Accessed: 24-Feb-2019].

[3] “The sequence diagram,” *UML basics*, 16-Feb-2004. [Online]. Available: https://www.ibm.com/developerworks/rational/library/3101.html. [Accessed: 24-Feb-2019].

[4] *IBM Knowledge Center*. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSRTLW_9.5.0/com.ibm.etools.jsf.doc/topics/cmvc.html. [Accessed: 27-Feb-2019].

[5] “A query language for APIs.,” *GraphQL*. [Online]. Available: https://graphql.org/learn/schema/. [Accessed: 07-Mar-2019].

[6] “A query language for APIs.,” *GraphQL*. [Online]. Available: https://graphql.org/learn/queries/. [Accessed: 07-Mar-2019].

[7] “What is HTML?,” *What is HTML*. [Online]. Available: https://www.w3schools.com/whatis/whatis_html.asp. [Accessed: 10-Mar-2019].

[8] “What is CSS?,” *What is CSS*. [Online]. Available: https://www.w3schools.com/whatis/whatis_css.asp. [Accessed: 10-Mar-2019].

[9] *Press Release Announcing Javascript*, 04-Dec-1995. [Online]. Available: https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html. [Accessed: 10-Mar-2019].

[10] *What is the Document Object Model?*[Online]. Available: https://www.w3.org/TR/WD-DOM/introduction.html. [Accessed: 10-Mar-2019].

[11] N. Foundation, “About,” *Node.js*. [Online]. Available: https://nodejs.org/en/about. [Accessed: 17-Mar-2019].

[12] Nodejs, “nodejs/node-v0.x-archive,” *GitHub*. [Online]. Available: https://github.com/nodejs/node-v0.x-archive/tags?after=v0.0.4. [Accessed: 18-Mar-2019].

[13] “GraphQL: A query language for APIs.,” *A query language for your API*. [Online]. Available: https://graphql.org/. [Accessed: 21-Mar-2019].

[14] “Glossary¶,” *Glossary - MongoDB Manual*. [Online]. Available: https://docs.mongodb.com/manual/reference/glossary/#term-collection. [Accessed: 01-Apr-2019].

[15] “JSON and BSON,” *MongoDB*. [Online]. Available: https://www.mongodb.com/json-and-bson. [Accessed: 01-Apr-2019].

[16] “Text Search¶,” *Text Search - MongoDB Manual*. [Online]. Available: https://docs.mongodb.com/manual/text-search/. [Accessed: 05-Apr-2019].

[17] auth0.com, “JSON Web Tokens Introduction,” *JSON Web Token Introduction*. [Online]. Available: https://jwt.io/introduction. [Accessed: 10-Apr-2019].