VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Software Engineering (CO3001)

Semester 211

Restaurant POS 2.0

Advisor:	Assoc. Prof. Quản Thành Thơ	
	Assoc. Prof. Nguyễn Đức Anh	
Students:	Nguyễn Hải Đăng	1952652
	Trần Đăng Khoa	1952788
	Lưu Nguyễn Hoàng Minh	1952845
	Nguyễn Phú Nghĩa	1952355

HO CHI MINH CITY, SEPTEMBER 2021

Contents

Member list & Workload

No.	Full name	Student ID	Percentage of work
1	Nguyễn Hải Đăng	1952652	25%
2	Trần Đăng Khoa	1952788	25%
3	Lưu Nguyễn Hoàng Minh	1952845	25%
4	Nguyễn Phú Nghĩa	1952355	25%

1 Requirement elicitation

1.1 Identify the context of this project. Who are relevant stakeholders? What are expected to be done? What are the scope of the project?

1.1.1 Identify the context of this project.

- In the Covid 19 pandemic, customers want to limit contacts with clerks when ordering food at the restaurants so that they can ensure their health. Given that they do not want to talk with the clerks, they can access the website through the QR code and start to order their meals and pay for them.

1.1.2 Who are relevant stakeholders?

- Customers
- Clerks
- Kitchen
- Software developers
- Restaurant managers
- Contractor managers

1.1.3 What are expected to be done?

- The system should allow non-direct contact between Clerks and Customers.
- The system should be implemented using Web technology, so customers will not have to install apps. The restaurant should prepare the QR code for customers' convenient access to the system.
- The system should be usable from mobile devices, tablet devices or normal computers/laptops.
- The system should be extendable to use in multiple restaurants in the future.
- The current transactions rate is about 300 orders per day.

1.1.4 What are the scope of the project?

- The web app is compatible for mobile devices, tablet devices, computers/laptops.
- Extendable for multiple restaurants in the future.
- Time scope: 7 weeks.
- Budget: $620.

1.2 Describe all functional and non-functional requirements of the desired system. Draw a use-case diagram for the whole system

1.2.1 Describe all functional and non-functional requirements of the desired system.

- **Functional requirements**
  - Table reservation
  - Ordering food
  - Alerts
  - Billing
  - Credit card processing
  - Customer management
  - Take-away option
  - Web technology + QR
- **Non-functional Requirements**
  - **Scalability/Backward compatibility**
    Metric: Installation of a new version shall leave all database contents and all personal settings unchanged.
  - **Efficiency**
    Metric: The food ordering system should be able to handle 300 orders each day while maintaining the performance requirement of a 0.1 second response time.
  - **Responsive web-based**
    Metric: At most 1 screen related-problem received from 10000 users.
  - **Usability**
    Metric: Four out of five users shall be able to order food within 5 minutes after a 2-hour training.
  - **High-compatibility**
    Metric: Compatible with all OSes such as Windows, Linux, iOS, Android, etc.
  - **Non-direct contact between clerks and customers**
    Metric: Normal business process (tables reservation, food ordering, payment, etc.) is not affected despite not having direct contact between clerks and customers.

1.2.2 Draw a use-case diagram for the whole system.

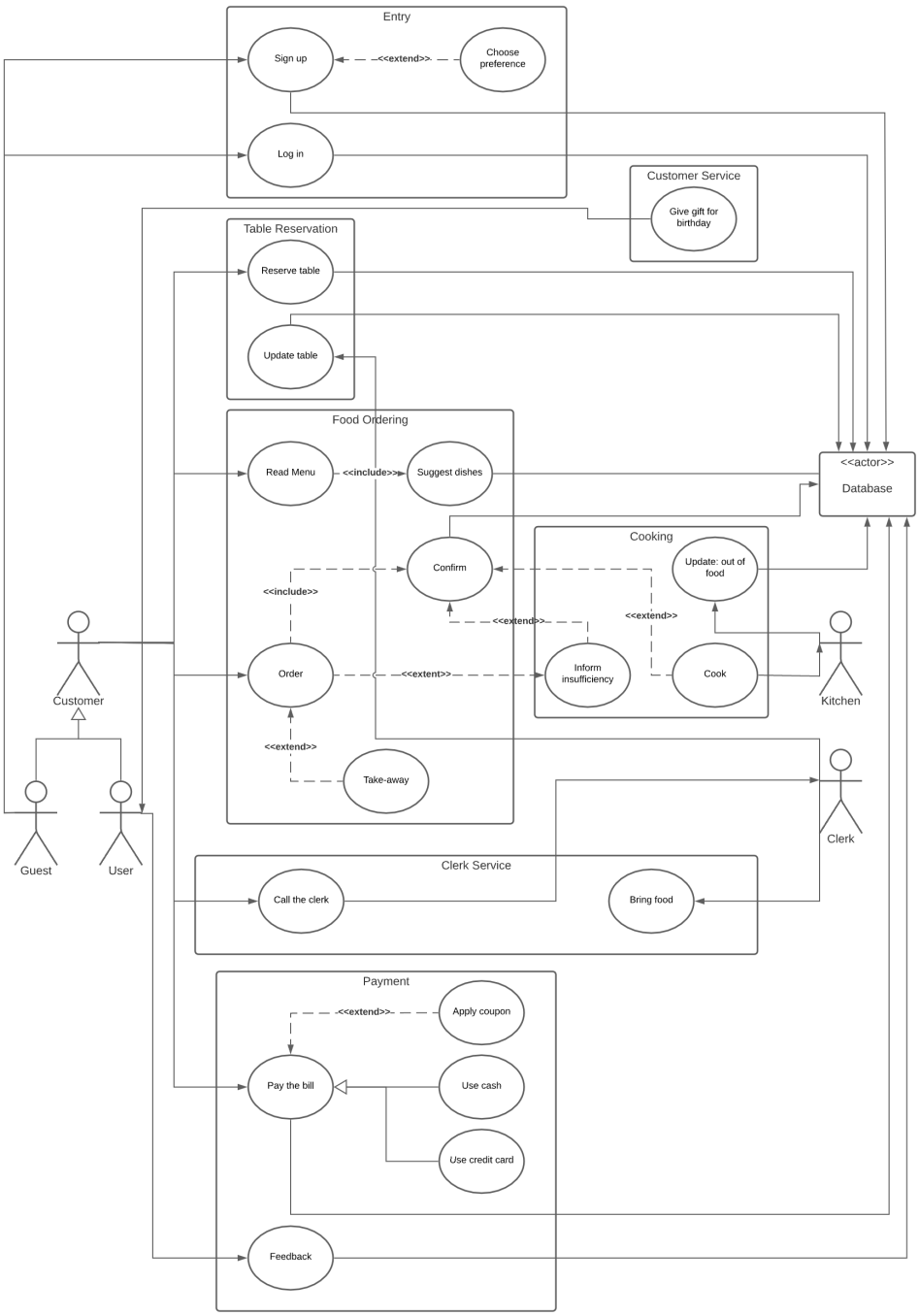

Figure 1: Use-case diagram for the system

1.3 Choose one specific feature, i.e. food ordering, table reservation, customer management. Draw its use-case diagram and describe the use-case using a table format

1.3.1 Use-case diagram for table reservation feature

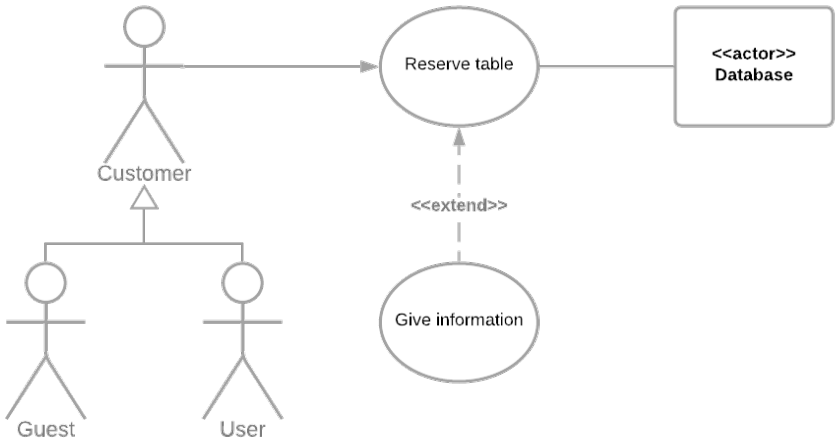

Figure 2: Use-case diagram of the table reservation

1.3.2 Tabular format for the table reservation use case

Use-case name	**Table Reservation**
Actor	Customer and database
Description	The customer is reserving table(s) in a branch of restaurant on a particular date
Trigger	Customer wants to have a meal at our restaurant
Preconditions	The customer is at the restaurant home page
Postconditions	The customer successfully reserves the table(s)
Normal Flow	1. Customer selects “Reservation” on the home page 2. System presents choice of restaurant branches (at Reservation page) 3. Customer selects a branch from among the choices 4. System presents choice of available dates for the current month (at Reservation page) 5. Customer selects a date from among the choices 6. System requests the quantity that customer wants to reserve (at Reservation page) 7. Customer enters the quantity he/she wants 8. System presents a Confirmation page to confirm the customer’s reservation 9. Customer confirms the selection in the Confirmation page

	10. System updates the reservation information
Exception	*Exception 1: at step 8* 8a. If the input is not a number, System request customer to re-enter 8b. If the selected branch does not meet the required quantity, System presents a page saying that there are no available tables for the selected quantity
Alternative Flows	*Alternative 1: at step 2* 2a. If customer is a guest, he/she need to give information first Continue step 2 in the normal flow *Alternative 2: at step 5* 5a. Customer selects a different month 5b. System presents choice of months (at Reservation page) Continue step 5 in the normal flow *Alternative 3: at step 10* 10a. Customer cancels confirmation. 10b. Confirmation time out. End the flow
Notes and Issues	Customer must have Internet connection

Table 1: Tabular description of the table reservation use case

2 System modelling

2.1 Draw an activity diagram to capture Major (not all) functional requirements of the desired system

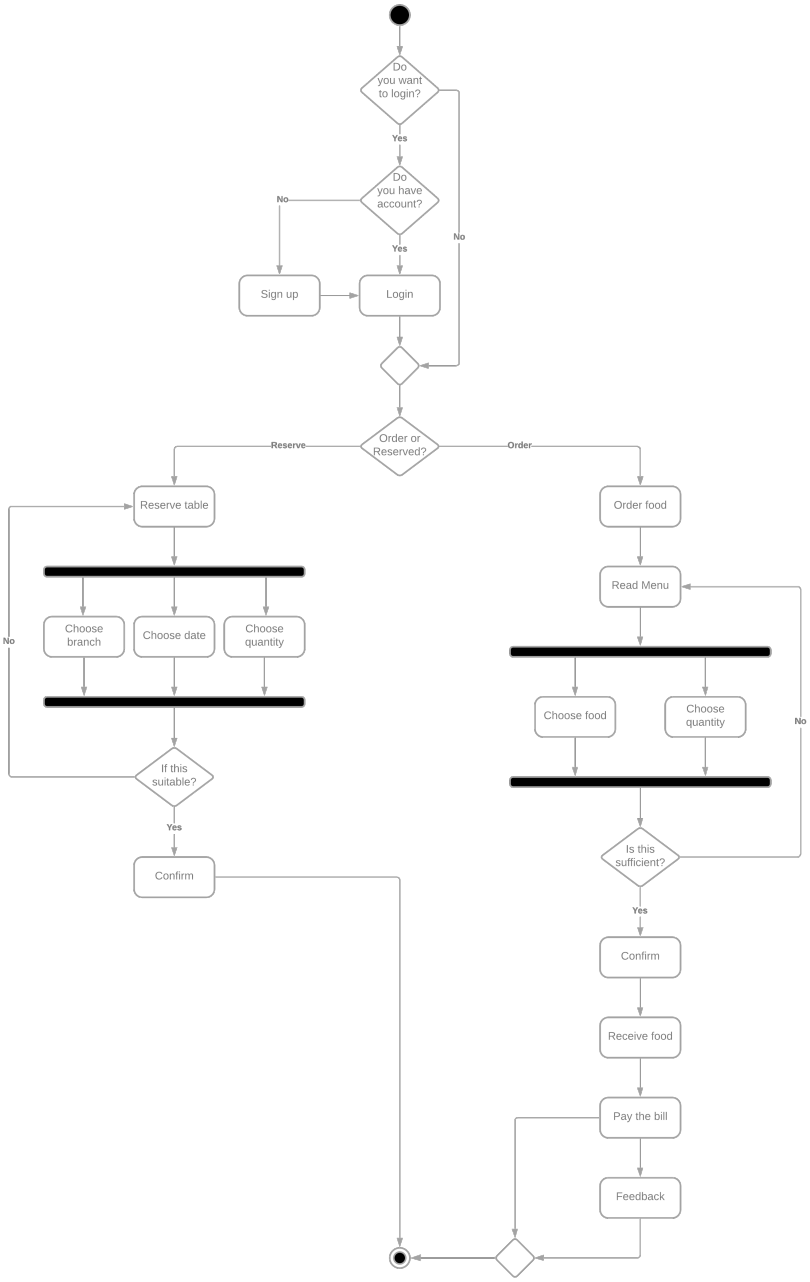

Figure 3: Activity diagram for the major functional requirements

2.2 Draw a sequence diagram for use-case in Task 1.3.

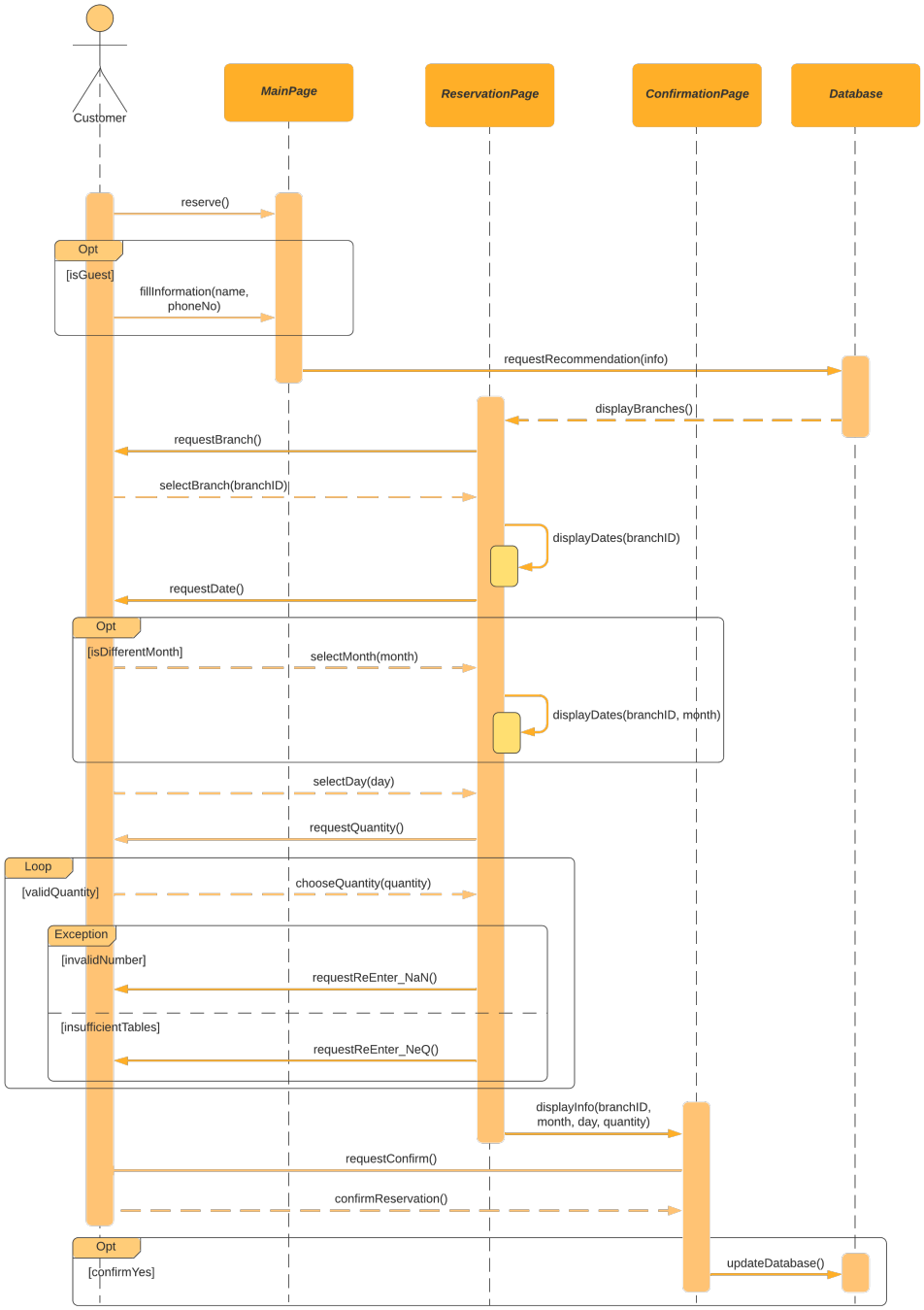

Figure 4: Sequence diagram for Table Reservation function

2.3 Draw a class diagram

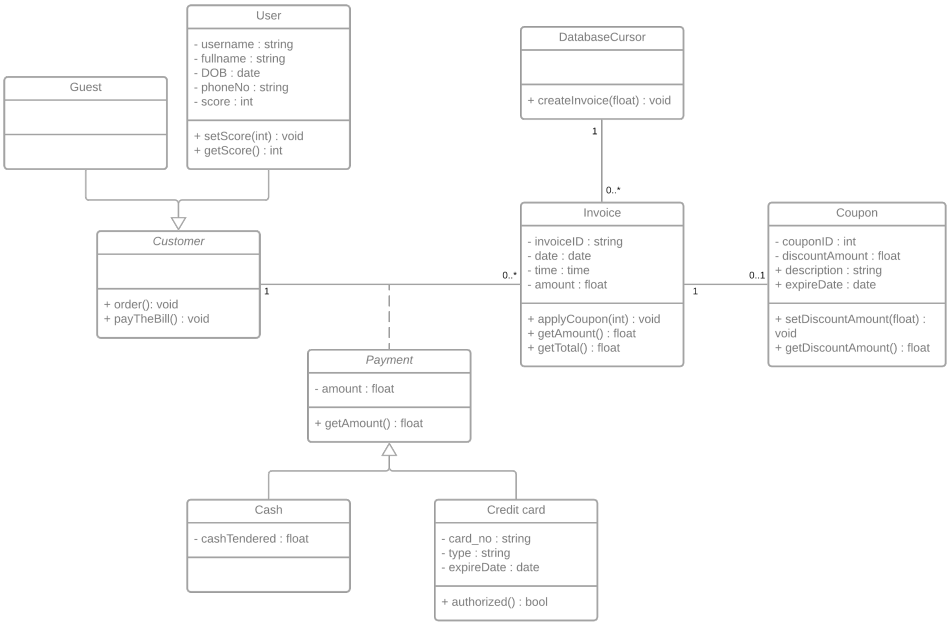

Figure 5: Class diagram for Payment function

3 Architecture Approach

3.1 Describe an architectural approach you will use to implement the desired system

Here we consider non-functional requirements of our restaurant POS system. Firstly, features are separated and they provide independent functions such as ordering food, reserving tables, etc. Moreover, with the purpose of maintainability, the system architecture should be designed using fine-grain, self-contained components for independent development and upgrade.

As a consequence, we choose the Model-View-Controller (MVC) pattern, in which:

- **Model** is responsible for manipulating the data based on instructions provided by the controller. This provides consistency throughout our application output and will give an enjoyable user experience. This is consider as the backend of the application, which handles the logic and queries to the database.
- **View** displays all features of the food ordering system such as food list, buttons, sliders, etc. Here, the view is consider as HTML/CSS part of the application.
- **Controller** handles all the logic corresponding to the User Interface (UI).

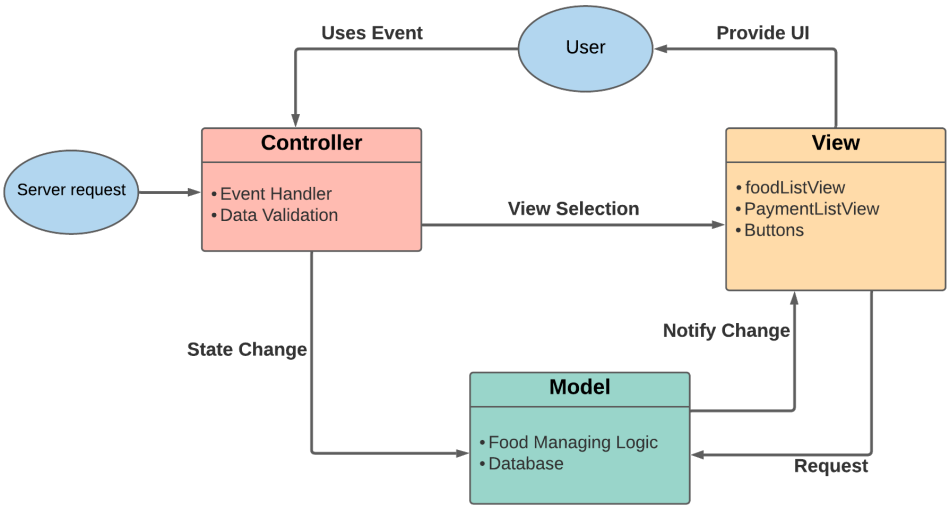

Figure 6: Restaurent POS application architecture using the MVC pattern

The advantages and disadvantages of using MVC pattern for the restaurant POS system:

- **Advantages**
  - MVC pattern will separate the UI from the business logic so that the development process can be faster.
  - Different components of the application in MVC can be independently deployed and maintained.

- Because the components are independent, the MVC help improving testing process of the website.

- **Disadvantages**
  - View and controller are often hard to separate in some cases.
  - The complexity of the model processing will slow down the data display and reduce the UI performance.

3.2 Draw an implementation diagram for Major (not all) functional requirements

3.2.1 Component diagram

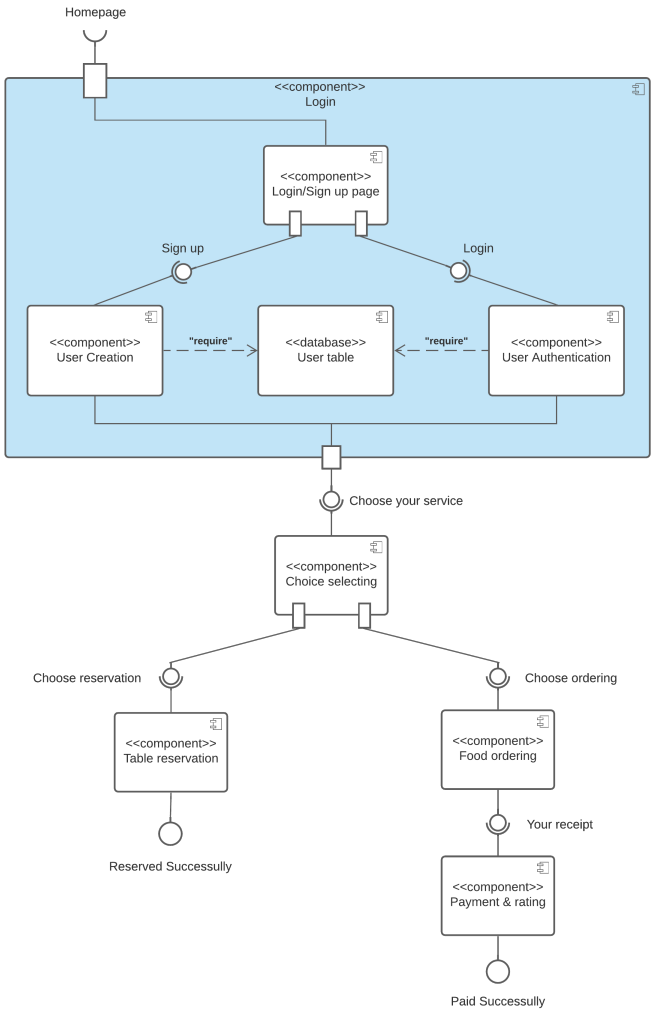

Figure 7: Component Diagram for the major functional requirements

3.2.2 Deployment diagram

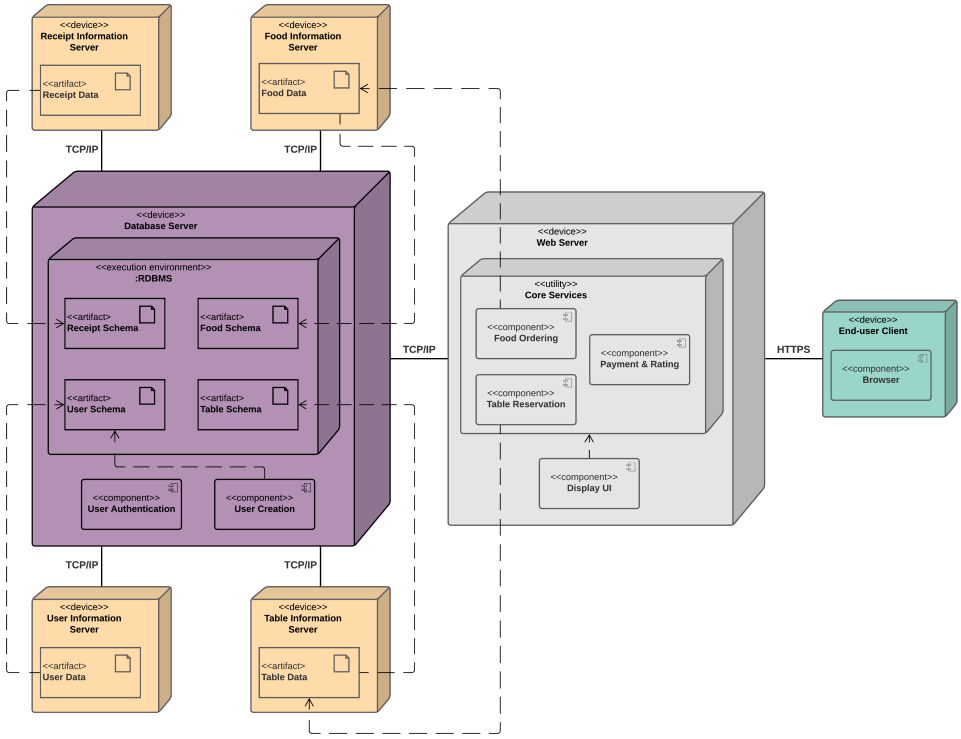

Figure 8: Deployment Diagram for the major functional requirements

4 Implementation – Sprint 1

4.1 GitHub

Here is the link to out current GitHub repository for the project.

We have the documents, materials and folders for Requirement, System modelling and Architectural design in the repository (please see the docs folder). The changes to these files/folders can be seen via the commits section. The documents were updated in the `docs` folder and the website was updated in the `website` folder.

4.2 The Minimum Viable Product

The source code of the Minimum Viable Product (MVP) is at our GitHub repository and its demonstration is presented in the video submitted with this report.

5 Implementation – Sprint 2

5.1 Firebase - Cloud Firestore

In this sprint, we are going to attach a database called Cloud Firestore to our aplpication.

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Realtime Database, it keeps our data in sync across client apps through realtime listeners and offers offline support for mobile and web so we can build responsive apps that work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions.

Cloud Firestore is a cloud-hosted, NoSQL database that the Apple, Android, and web apps can access directly via native SDKs. Cloud Firestore is also available in native Node.js, Java, Python, Unity, C++ and Go SDKs, in addition to REST and RPC APIs.

Cloud Firestore is categorized into NOSQL key-value stores. A key–value database, or key–value store, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, and a data structure more commonly known today as a dictionary or hash table. Dictionaries contain a collection of objects, or records, which in turn have many different fields within them, each containing data. These records are stored and retrieved using a key that uniquely identifies the record, and is used to find the data within the database.

Following Cloud Firestore's NoSQL data model, we store data in documents that contain fields mapping to values. These documents are stored in collections, which are containers for your documents that we can use to organize our data and build queries. Documents support many different data types, from simple strings and numbers, to complex, nested objects. We can also create subcollections within documents and build hierarchical data structures that scale as our database grows. The Cloud Firestore data model supports whatever data structure works best for our app.

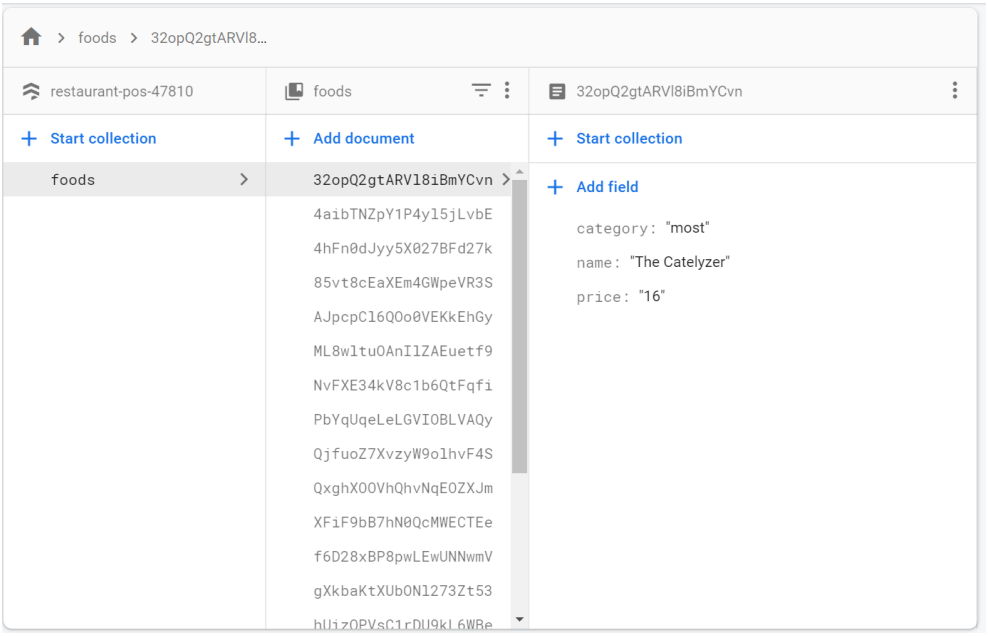

Figure 9: Cloud Firestore managing with the Firebase Console

Additionally, querying in Cloud Firestore is expressive, efficient, and flexible. Create shallow queries to retrieve data at the document level without needing to retrieve the entire collection, or any nested subcollections. Add sorting, filtering, and limits to our queries or cursors to paginate our results. To keep data in our apps current, without retrieving our entire database each time an update happens, add realtime listeners. Adding realtime listeners to our app notifies us with a data snapshot whenever the data our client apps are listening to changes, retrieving only the new changes.

Protect access to our data in Cloud Firestore with Firebase Authentication and Cloud Firestore Security Rules for Android, Apple platforms, and JavaScript, or Identity and Access Management (IAM) for server-side languages.

5.2 Testing

Here we have applied vảio methods for functionality Testing of a Website.

- **HTML and CSS syntax**
  When coding the MVP, we apply the vscode extension named problem in order to keep track of HTML and CSS syntax.
  ESLint is a built into most text editors so that we can run ESLint as part of your continuous integration pipeline. This will statically analyzes our code to quickly find problems.

- **Usability**
  We apply the WCAG 2.1 for testing our color schema and contrast in the website. We want to ensure that our color schema should be suitable with the WCAG rules so that users can read and interact with the website easily.

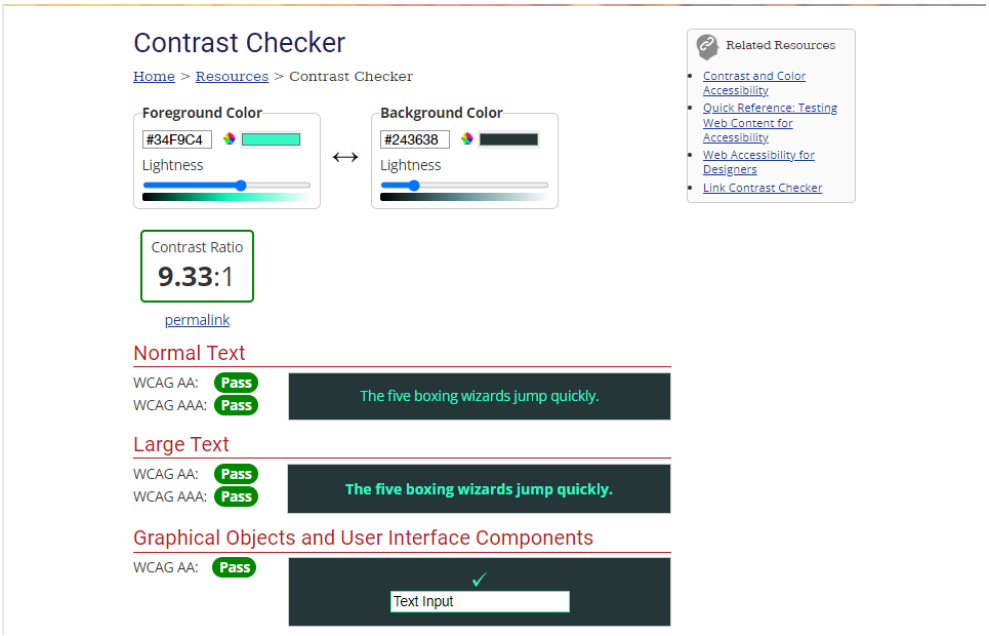

Figure 10: Color Schema Checking

- **Test syntax and logic**
  There is the confict when applying database into the code since the Firebase Firestore use the NOSQL that asynchronously run the getting the data. After researching for this problem, we came into the promises method, a modern alternative to callbacks for asynchronous code. A promise represents an operation and the future value it may return. It also lets us propagate errors similar to try/catch in synchronous code. By applying Promises function, we can synchronize the data from database with the function in the code.

5.3 Future work

Everything we do until this sprint 2 is just for a demonstration. Therefore, in the future, we want to tune the whole app to make it usable in practice. Some possible improvements might include:

- We have planned the testing section into 2 ways: surveys and interviews.
  - For the interviewing section, we planned to reach 10 users and let them use the MVP so that we can collect more insights and improve our functional and non-functional requirments.
  - For the surveys, we intend to find more about the users behavior, how they think and use the MVP.
- Adding conditions to check at form's inputs to make the application more robust.
- Use Firebase Authentication to authorize and clasify our customer roles.
- Implement a food suggestion algorithm using Machine Learning.

6 Conclusion

After finishing the project, we have gained a lot of knowledge about different aspects of the development process, from creating various diagrams to technical architectures. Additionally, we have learnt about the skills such as Firebase, Github, time management, etc. when developing the website. This is also a very interesting experience for us to code an application hands-on, even though it is just a Minimal Viable Product. In the end, this is an inspiration for us to practice more on developing and maintaining products in the future.

References

[1] Complete course on UML diagrams (Unified Modeling Language). https://www.youtube.com/watch?v=WnMQ8HlmeXc&t=1756s.

[2] Diagramming Tutorials. https://www.youtube.com/playlist?list=PLUoebdZqEHTwbYD8oo6Wr81Xb7uCAh_oz.

[3] Key–value database. https://en.wikipedia.org/wiki/Key%E2%80%93value_database.

[4] Getting started with Firebase for the web – Firebase Fundamentals. https://www.youtube.com/watch?v=rQvOAnNvcNQ&t=343s.

[5] Firebase Firestore Tutorial. https://www.youtube.com/playlist?list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB.