BRAUDE
College of Engineering, karmiel

Capstone Project Phase B

SMART FOOD

22-2-D-25

Zeev Brazily

zbarzily@braude.ac.il

Shai Siso

shai.siso@e.braude.ac.il

Shahar Avital

Shahar.avital@e.braude.ac.il

Table of contents

1. Project review and process description

1.1 Description of our solution

In order to implement the solution, we designed a WEB system that will work as a complete system that performs all the required actions to provide user experience convenient and effective and help both the customers and the restaurant itself.

The customer will be able to set a table reservation and to get in a waiting list if needed. Then he will be notified with SMS and mail if a table turns out available.

The customer could also make a take-away/delivery order from the restaurant. He will be notified on each status change along the way.

There is also the possibility for the customer to scan a barcode which is on the table and make an order himself and save time.

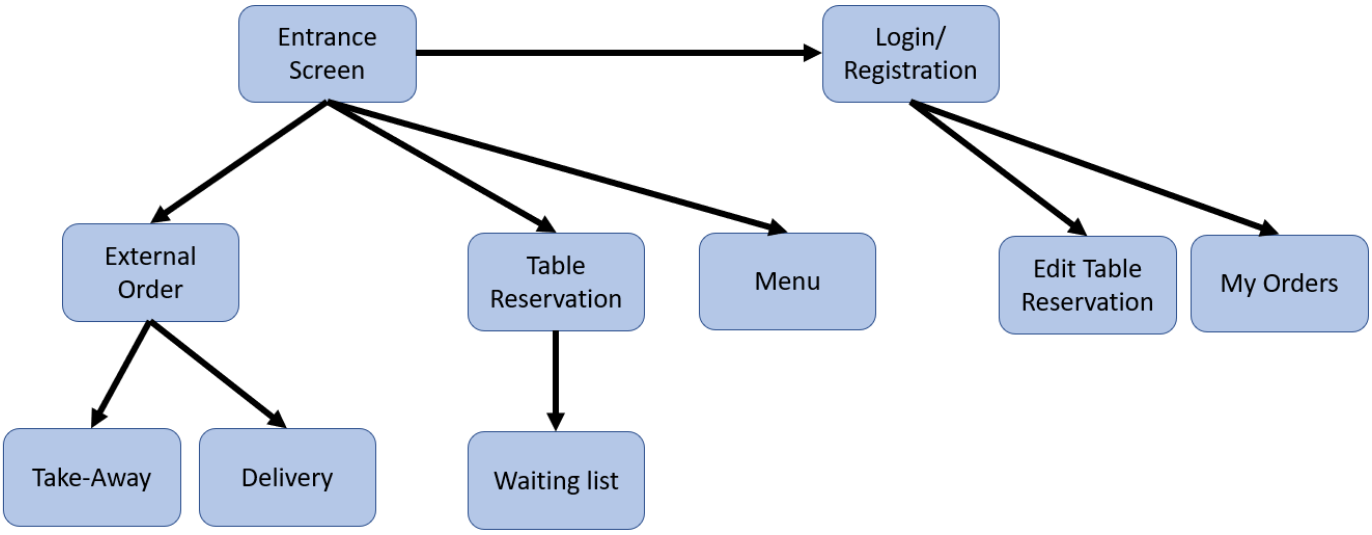
There is also the possibility for members to register. A registered user(member) will get more discounts, watch existing reservations, and watch past orders.

On the restaurant side, each employee needs to login with a phone number and password. He will be able to mark in a check box if he is starting a shift. After the login, each employee will be able to execute actions according to his role. The permissions are hierarchical, and each level includes the levels below. Examples of some of the actions:

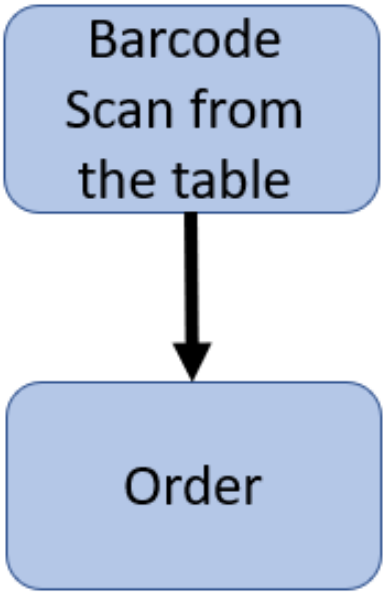
- Hostess: Reservations – add, update, delete
- Waiter: Table orders: add, update, delete, add items to an existing order
- Shift Manager: approve/disapprove shifts, cancel item requests, external orders – add, update, delete, change status, payment.
- Manager: Employee – add, update, delete, menu management, tables management, discounts management, generate barcodes for tables for specific time and generate reports.

1.1.1 The general flow of the system

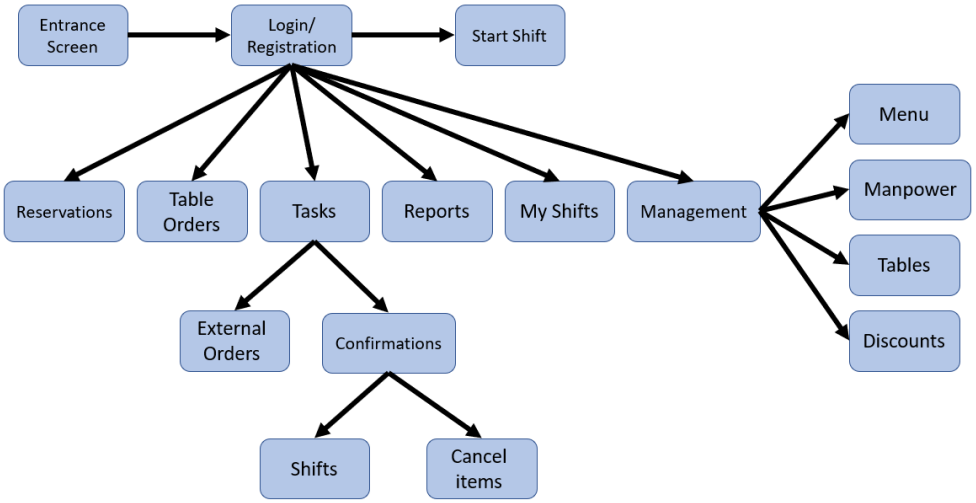
1.1.1.1 User flow – website



1.1.1.2 User flow - restaurant



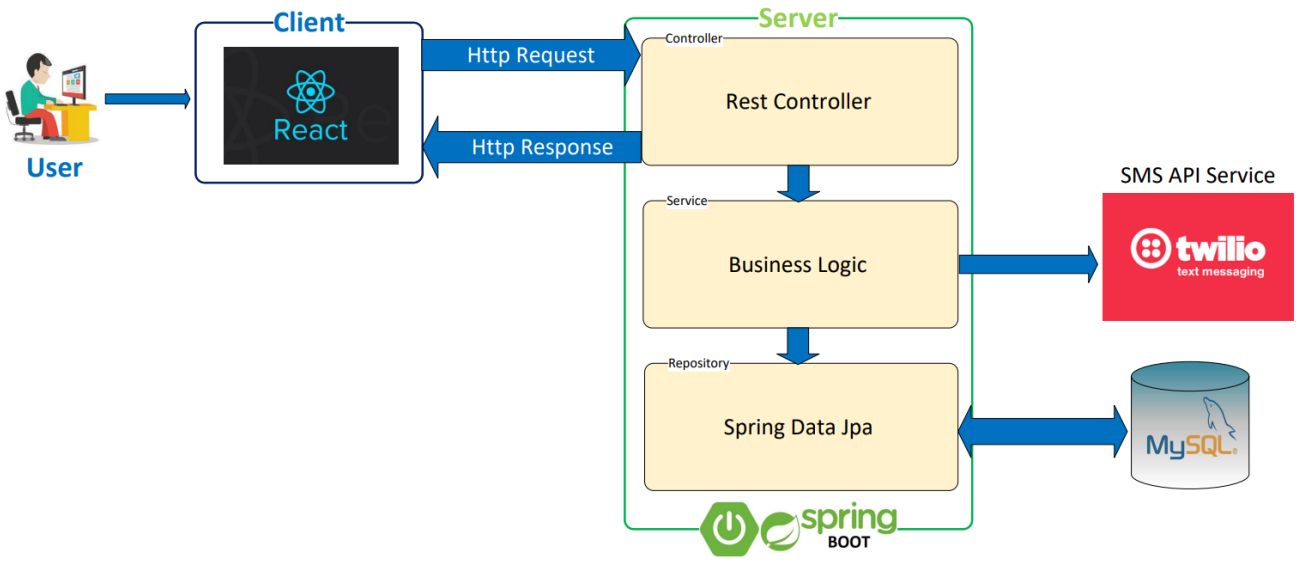
1.1.1.3 Employee flow



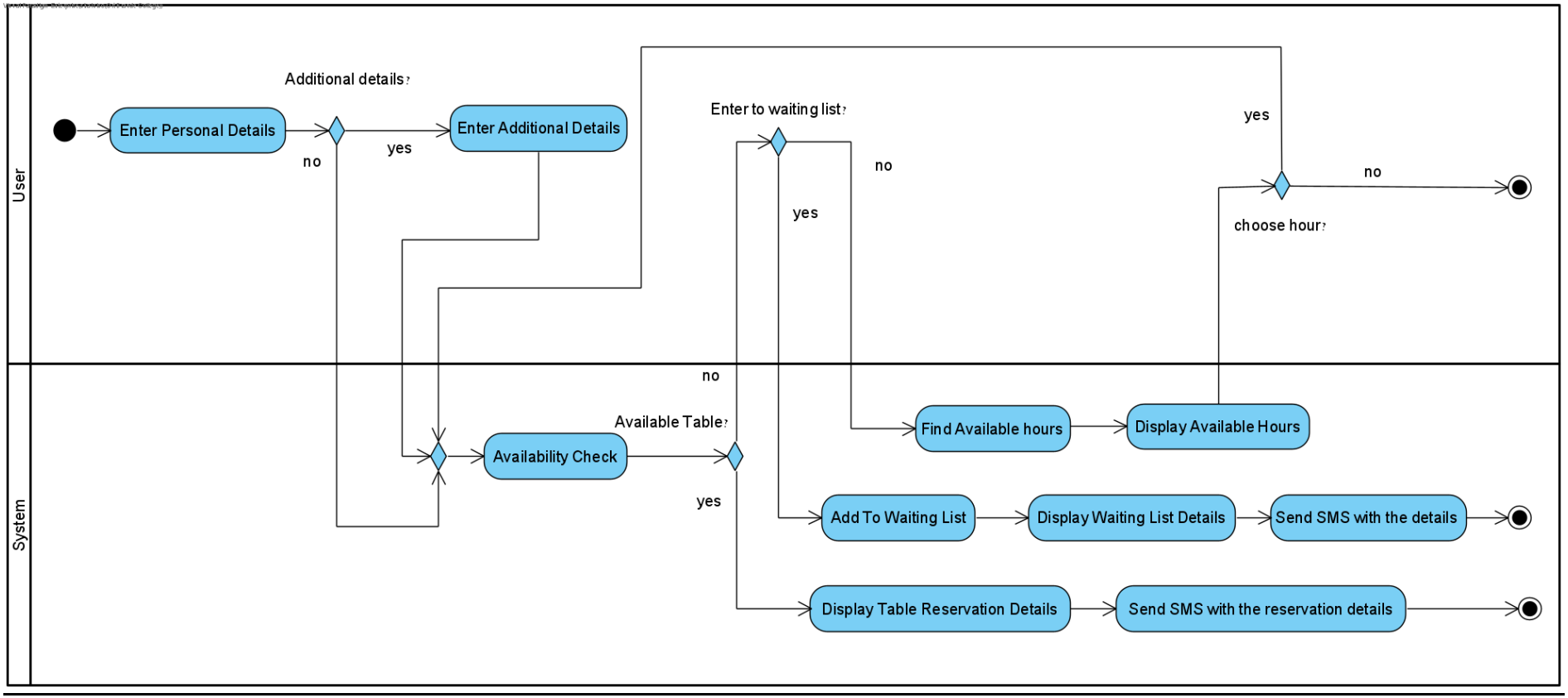
---

* Each action will be limited to the permission level.

1.1.2 Software Architecture diagram



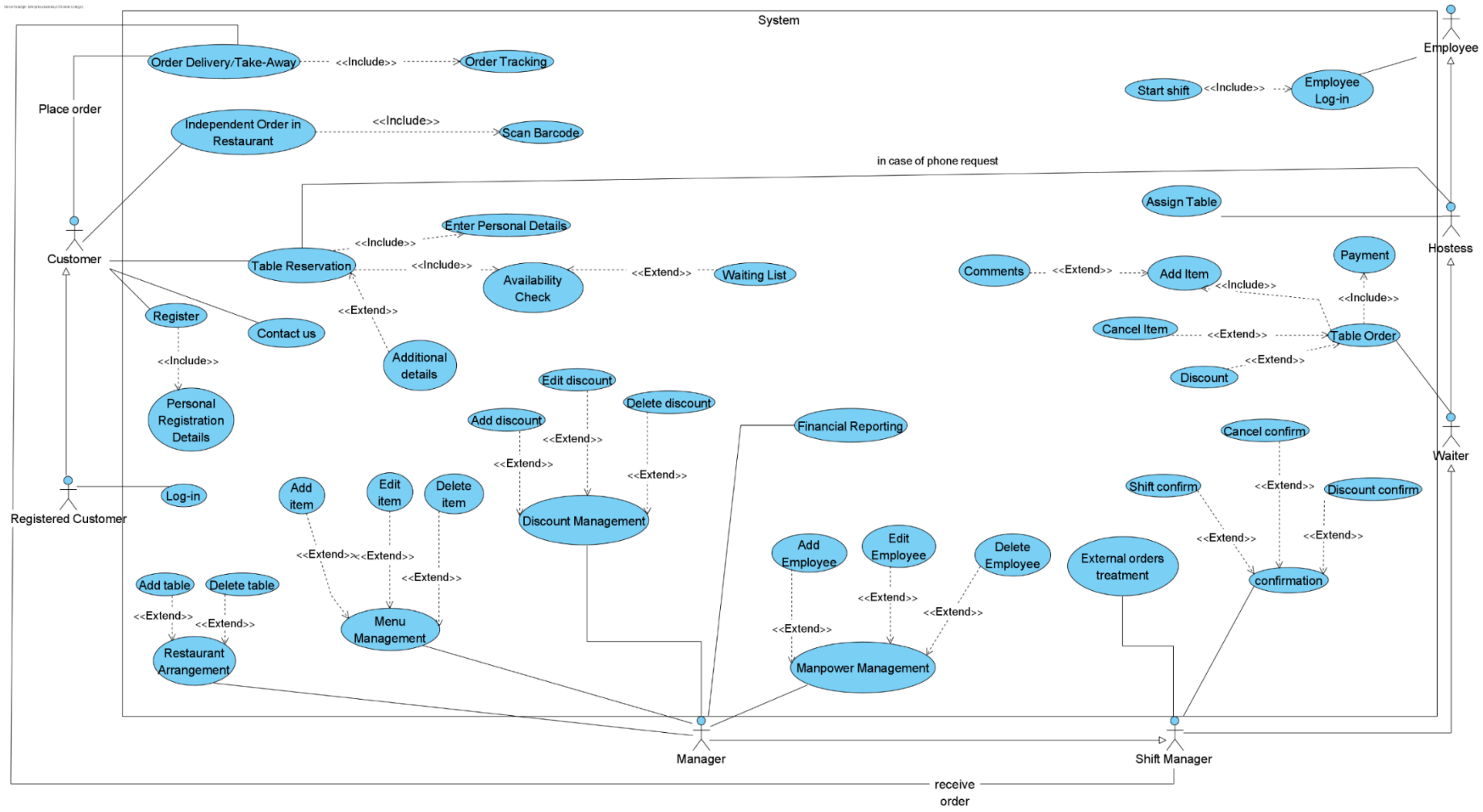
1.1.3 Activity diagram – table reservation



Activity diagram -description

1. The user enters personal details – name, phone number, date, time, number of diners.
   1.1 The user also can enter additional details if he wants – special occasion, location preference (inside, outside, bar), allergies, additional requests.
2. Availability check – the system checking if there is an available table that matches the request.
   2.1 if there is an available table – the system will display the reservation details.
      2.1.1 The system will send detailed SMS to the customer.
      2.1.2 End activity.
   2.2 if there is not an available table – the user will choose if he wants to enter the waiting list.
      2.2.1 YES – the system will add the user to the waiting list.
         2.2.1.1 The system will display the waiting list details.
         2.2.1.2 The system will send detailed messages to the customer.
         2.2.1.3 End activity.
      2.2.2 NO – the system will display other available hours in this date
         2.2.2.1 If the user chooses one of the hours – GOTO '2'
         2.2.2.2 Else – end activity.

1.1.4 Use case diagram



1.2 Description of development process

1.2.1 The system's development stages

First, we developed all the infrastructure to build our website. The development was carried out using technologies of web application such as JavaScript, HTML/CSS languages, React JS, Bootstrap, Spring boot. In order to start working we were required to learn in depth those technologies. We studied it by watching tutorials on YouTube and Pluralsight and reading documentation of the frameworks.

After learning and understanding the technologies, we started the stage of development.

In the beginning, we set the required database on MySQL and build the model according to the database.

Then, we started to work on the server side (back-end). for each functionality that we defined in the use case:

- Repository: for each entity we defined a repository that is in charge of saving the data in the database.
- Service: include the logic of the functionality.
- Controller: accept client requests from a specific route.
- Tested the functionality on Postman that was used as a client and allowed us to check the infrastructure of the communication between the server and the client.

We also defined for each functionality a permission level with Spring Security.

The next step was the client side (front-end). The UI development was carried out according to the screens we specified in part A. Like the server side, we preferred to do each functionality from start to end. We performed input integrity testing which verified the data that is sent to the server. After that we added some features like the barcode scan and the SMS and mail notifications, and real time notifications for the managers regarding the tasks (with use of WebSockets).

Through the development process we consulted with potential stakeholders such as restaurant owners and customers. We shared our UI and got feedback.

1.2.2 Description of the challenges and their solutions

The First challenge was learning the new technologies we needed in order to start developing. We had basic knowledge on the technologies, so our main tactic was to watch tutorials about those fields on YouTube and Pluralsight. Regarding the development itself, we had several challenges. We wanted the Waiting lists to be as efficient as possible without blocking the main thread of the server, so we decided to use an async method that will start to operate after a reservation has been canceled or that a new table was added to the restaurant. Another challenge was

the customer's notifications: SMS (on Twilio) and Email (Spring Framework JavaMail). We needed to understand how to use those APIs and to make sure the messages are sent correctly. About table reservations, we needed to think of a way to decide which tables to favor for each reservation. In addition, in case of canceling a reservation, our first instinct was to search if there is a customer from the waiting list that we can confirm his reservation. But later we understood that it is possible that more than one reservation can be confirmed after one cancellation. Also there was a necessity to save the reservation while we wait for the customer response to prevent other customers from making a new reservation and taking the place.

We wanted our API to be as secure as possible, so we had to learn Spring Security. Each registered user gets a token after the login. The token is sent to the server on each request in order to verify the permissions and block the request in case he doesn't have a permission.

We needed to learn how to generate a barcode for each table. The challenges were learning QRCode.React and limiting the time that each barcode is active to prevent malicious orders.

We wanted our website to be available for each user. We had several options to deploy our website. Eventually we decided to use AWS and Heroku. We had to understand how to use Services on those platforms.

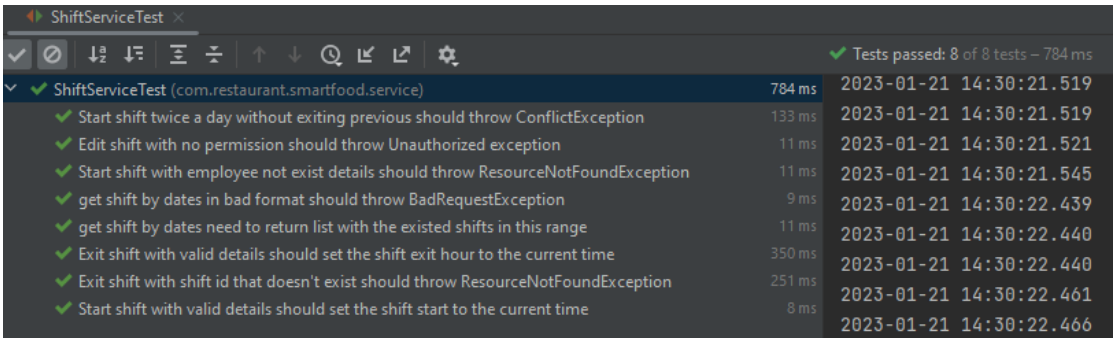
Also, we had an unpleasant event of hacking. When storing our code on GitHub, it actually revealed our credentials which included the authentication details to send Emails, and someone took advantage of it and sent emails from our account. We solved it by storing all the sensitive data in a specific file which is not exposed to GitHub (We added it separately in the submission).

1.3 Description of testing process

We can divide the testing process to 3 parts:

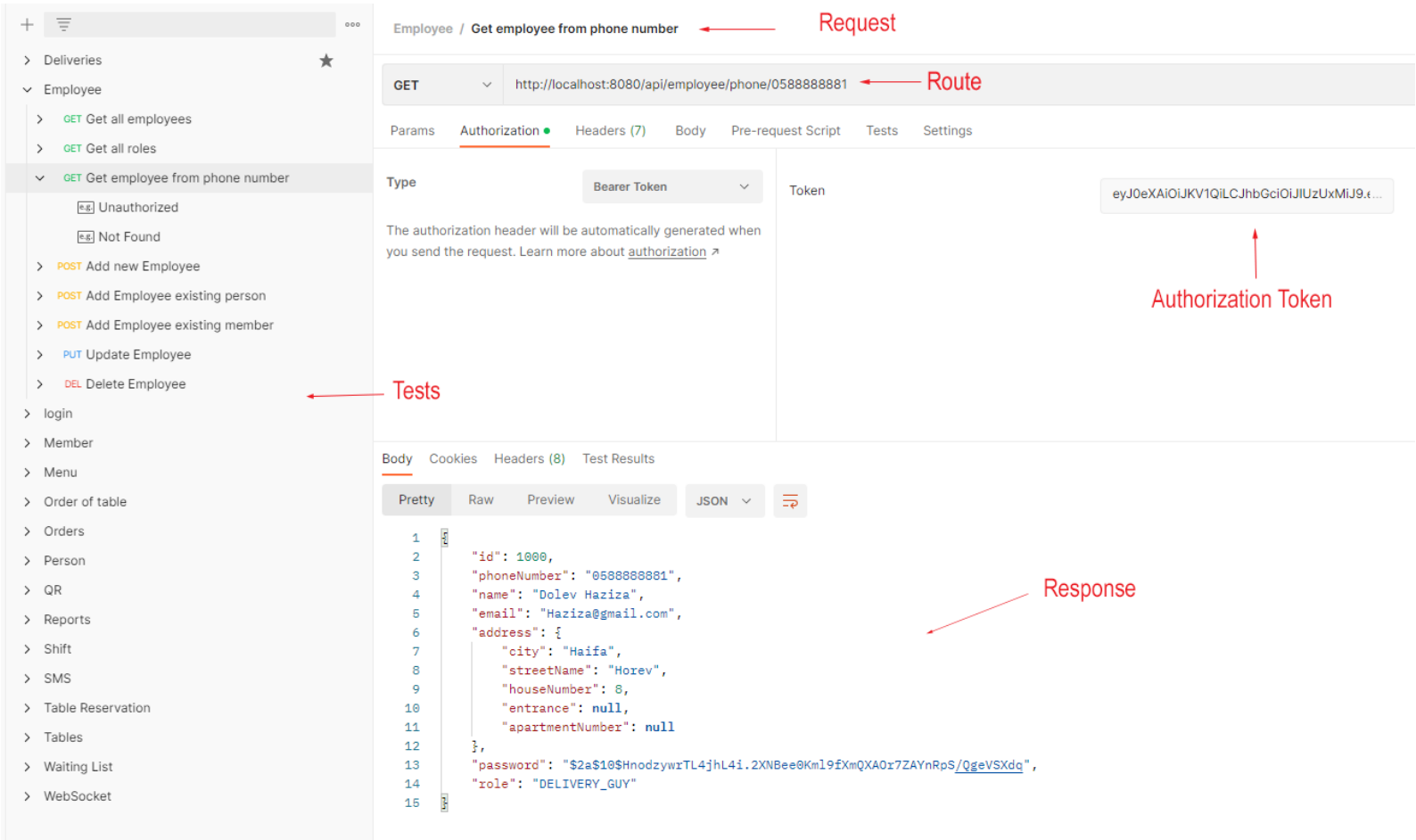
- "simple" checks: the purpose was to make sure that each input from the user will be get through validation according to the predetermined criteria. There are number of pages on our website that contains forms with input fields that needs to be validated, for example:
    - table reservation needs to be in future time.
    - Validation on text type (number/free text/mail).
    - Passwords need to be at minimum length.

- Unit tests: we tested our main logic by creating unit tests for specific methods in the service layer (back-end).
  Example:



- Functional tests:
  - Postman: for each functionality we tested the functionality on Postman that was used as a client and allowed us to check the infrastructure of the communication between the server and the client. Also, we verified that the data is saved in the database.
  - Functional tests of the system itself - we wanted to make sure that all buttons displayed to the user perform the operations for which they were created.

  Postman Example:



1.4 Results and conclusions

Our work on the project was very productive, we achieved all the goals we wanted to achieve and beyond that. The work was difficult and intensive but very satisfying and worthwhile, we learned a lot from it.
In phase A, we dealt with the definition of the problem and characterized our system as a solution to this problem. As part of the process, we used a lot of topics we purchased during the degree, and we also used our supervisor and learned how to perform a process of characterization and definition of the system.

In phase B, we learned new technologies, implemented our new knowledge in the project and developed the system. About the development process, at the start we worked together in order to implement the infrastructure of the system such as the entities and the packages layout and the hierarchy. Later, we worked simultaneously on different branches on different parts of the project. Approximately once a week we have met in order to go through each other's code and to merge the branches. Overall, we feel we worked well. One thing we would change is to start working sooner in order to take pressure off at the end.

2. User Documentation

2.1 User's Guide Operating Instructions

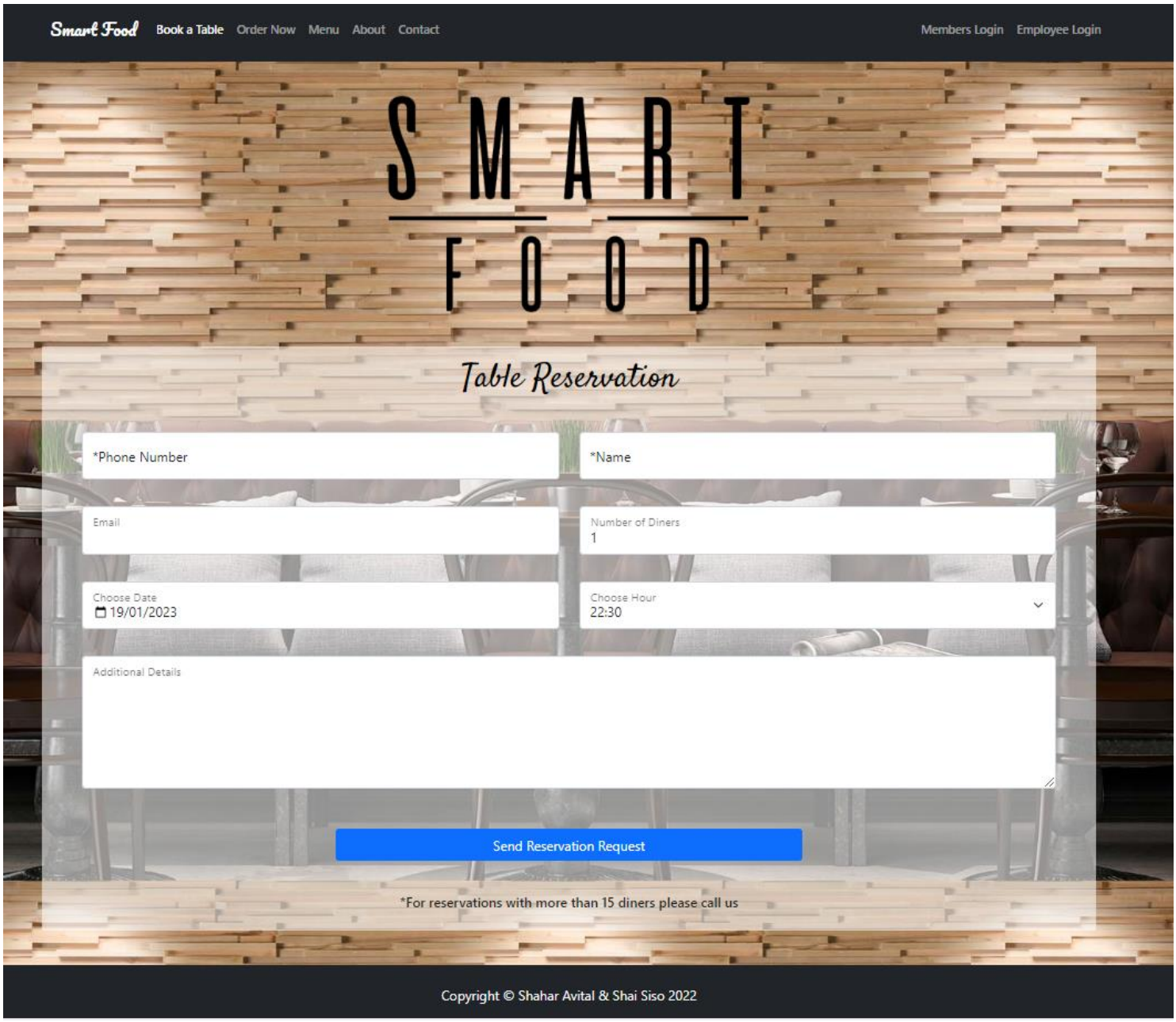
2.1.1 General description

The Smart-Food website will be used by restaurant customers and will cover all their needs regarding the restaurant such as: external orders – delivery, take-aways and order tracking, table reservations including automated waiting lists with notifications accordingly. It is also for the restaurant itself, from the employees that will use it for the day-to-day job until the manager who will have the management operations available like menu, discount, tables layout, employees and generate reports.
In addition, our system will have new features such as table reservation and entering a waiting list automatically if necessary and independent ordering at the restaurant through a barcode scan and without waiting to a waiter.

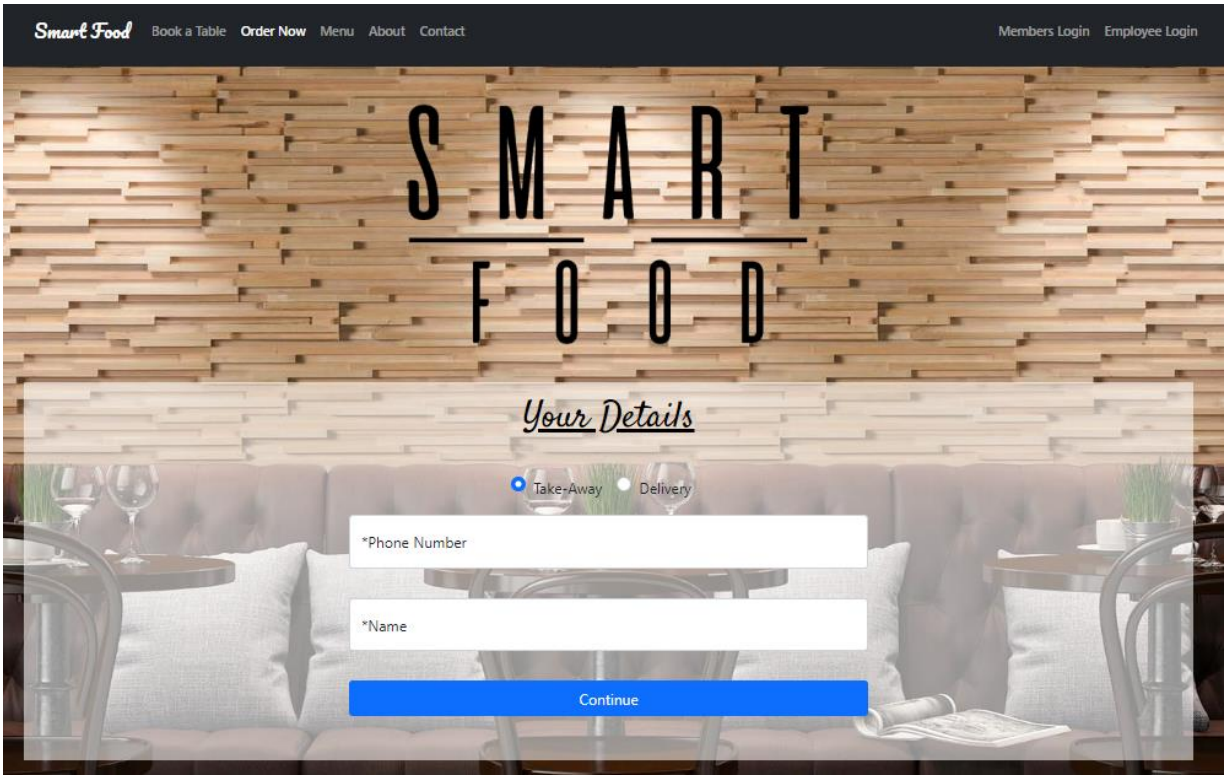
2.1.2 The operational process of the system

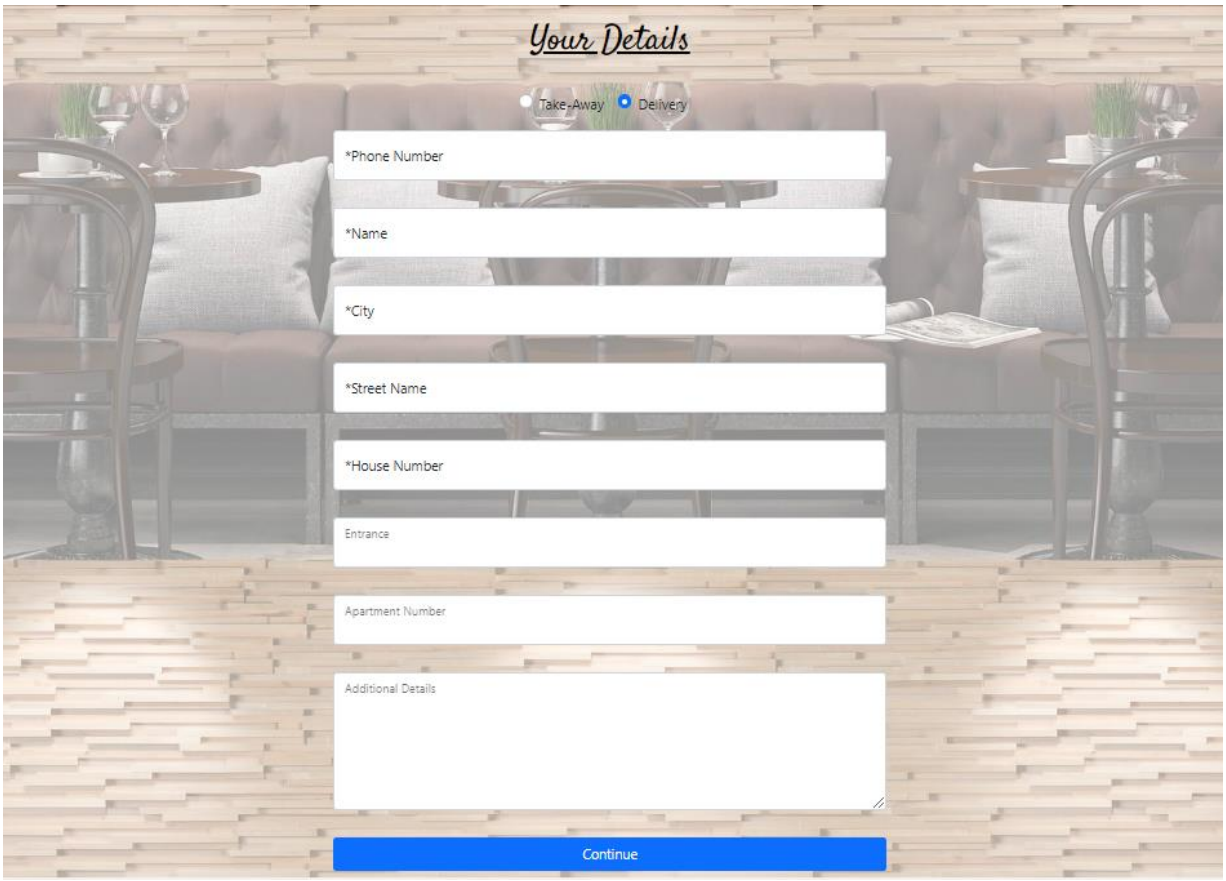
The main page will be for the customers' use. The activities that customers will be able to do are:

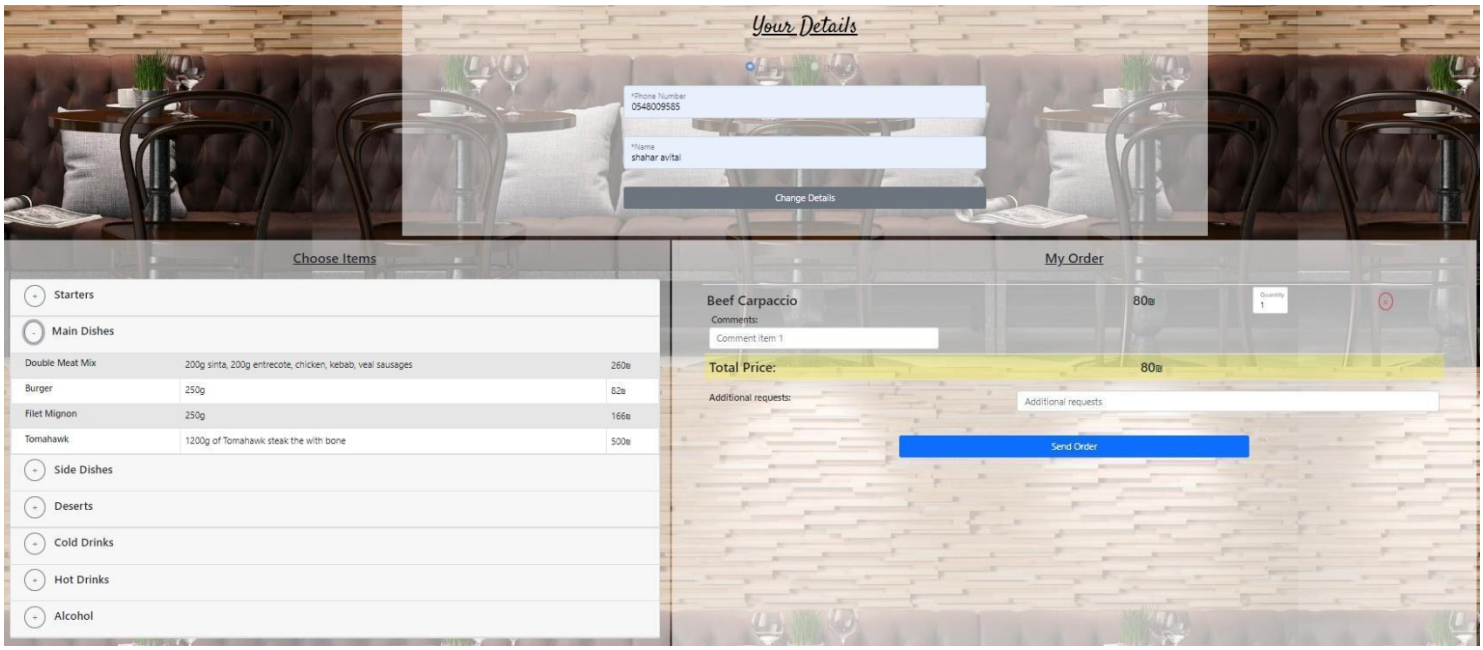
- Book a table: set a new table reservation and in case there is no option due to limited seats, enter a waiting list. If a table is available, the customer will receive mail and SMS and the table will be saved up to one hour unless he confirms the reservation.


Smart Food
Book a Table
Order Now
Menu
About
Contact
Members Login
Employee Login
SMART FOOD
Table Reservation
*Phone Number
*Name
Email
Number of Diners
1
Choose Date
19/01/2023
Choose Hour
22:30
Additional Details
Send Reservation Request
*For reservations with more than 15 diners please call us
Copyright © Shahar Avital & Shai Siso 2022

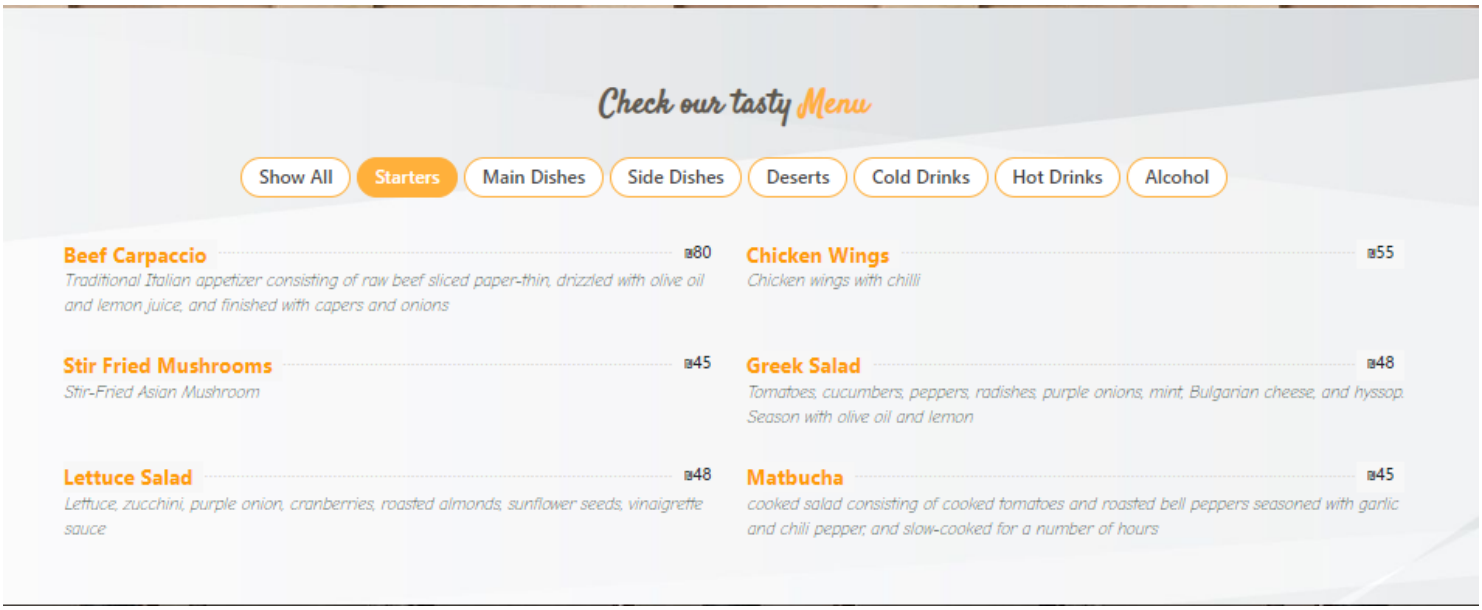
- Order now: the customer can choose between take-away and delivery and make an order. The customer will be getting an update on each progress of the order for example: in preparation, on its way.





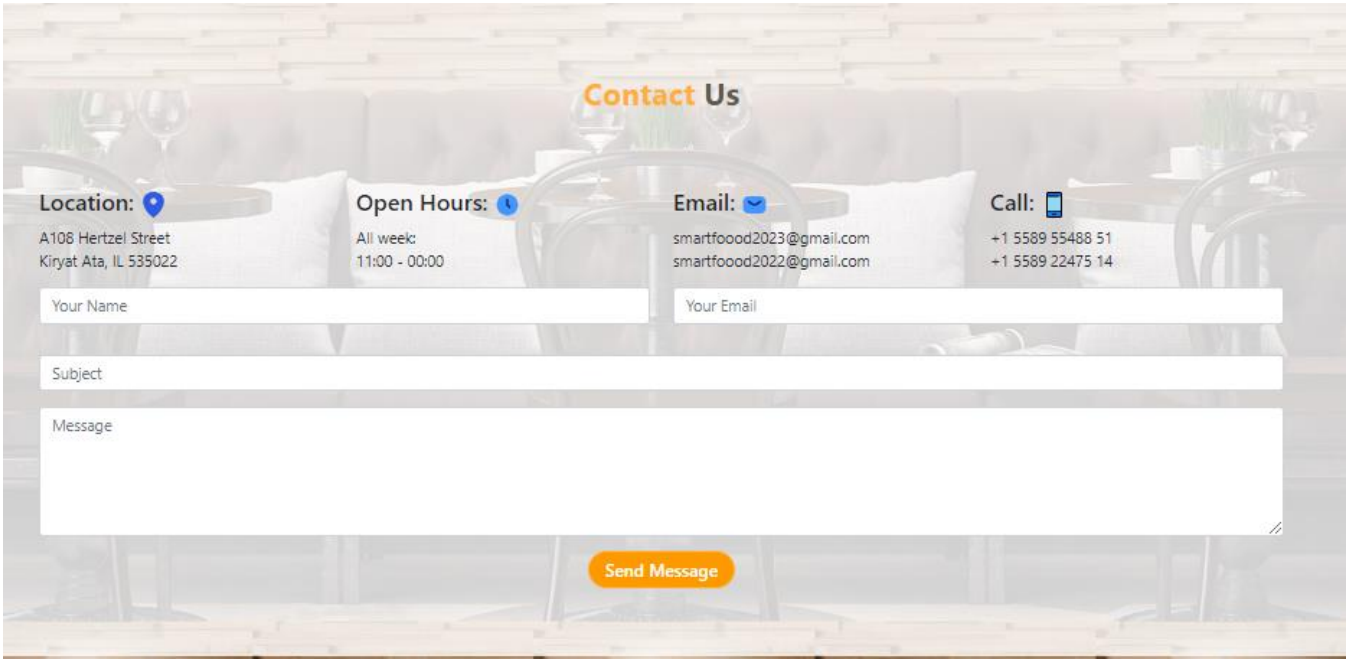


- Menu: the customer will be able to watch the menu of the restaurant.

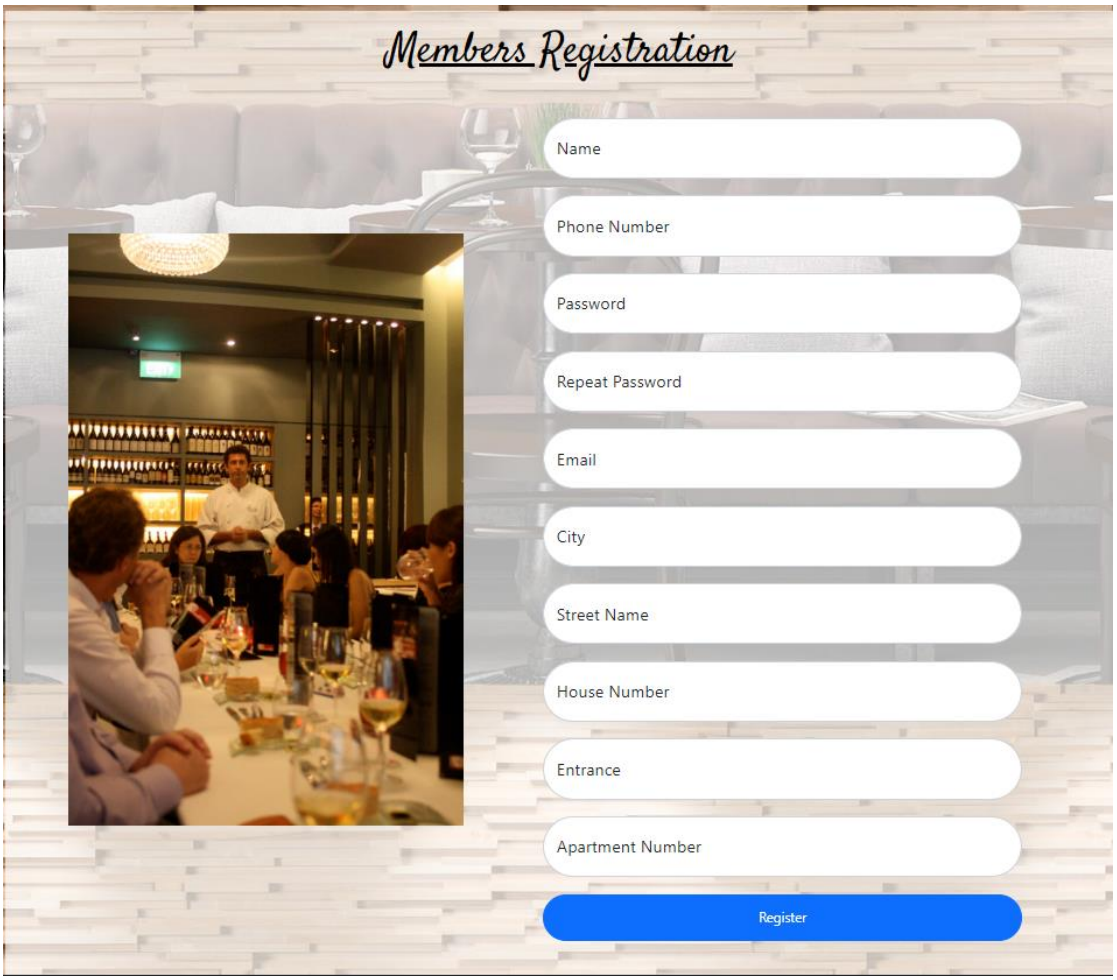


- About: the customer will be able to read about the restaurant.

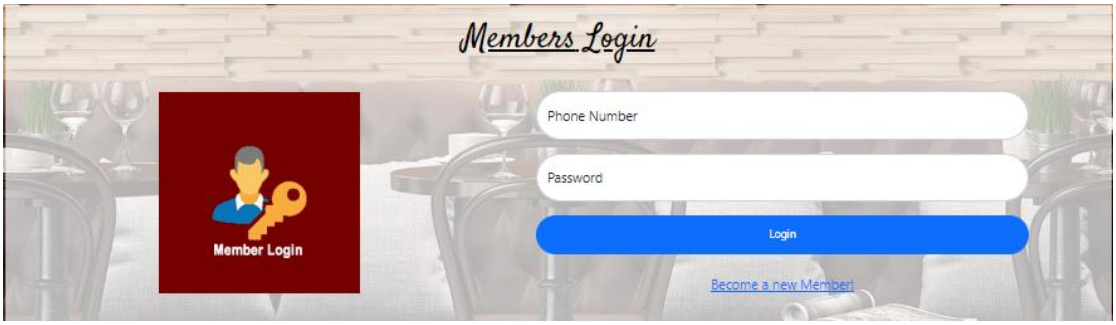
- Contact: the customer will be able to contact the restaurant management.



- Become a new member: a customer will be able to register to the website and become a member of the restaurant.



- Members Login: as a member, the customer will have special discounts. after a customer login-in as a member, he will have additional activities:

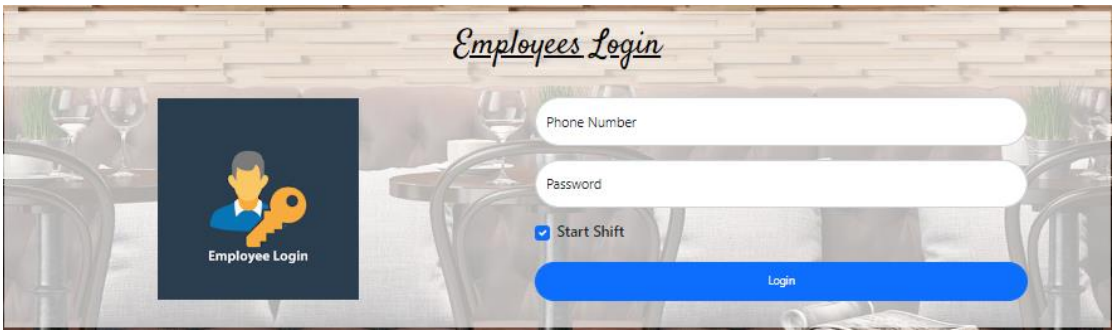


  - My reservations: the customer will be able to watch past and future reservations. He will also be able to edit future reservations.

  - My orders: the customer will be able to watch past and future external orders. He will also be able to see the order status.

For the employees use, there is the option “Employee Login” that requires a phone number and password. After he fills the fields, and if the details are valid, he will be able to do all the activities that required to do for the day-to-day work, depends on the role:

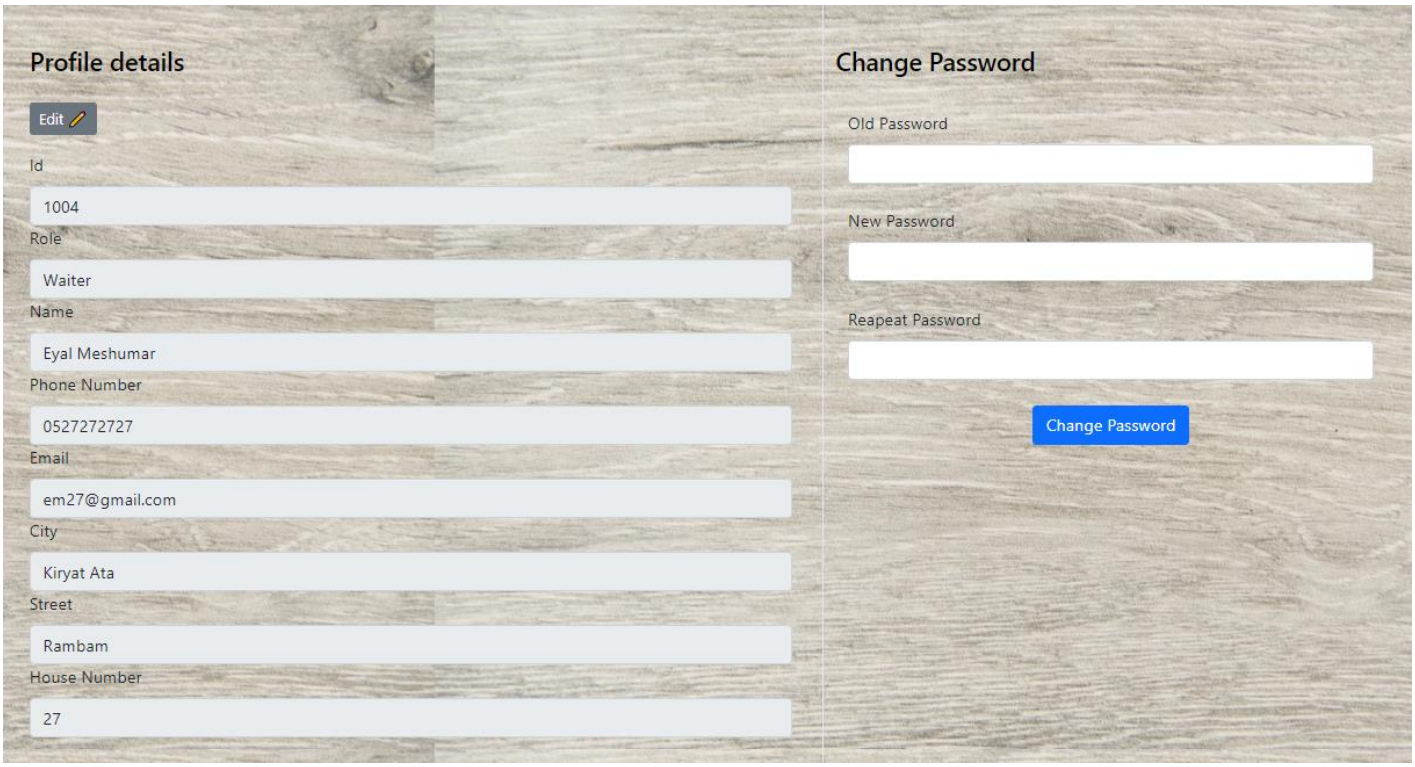
- Check “start shift” if the employee is starting a shift.



- When the employee clicks his name, a drop-down will appear:

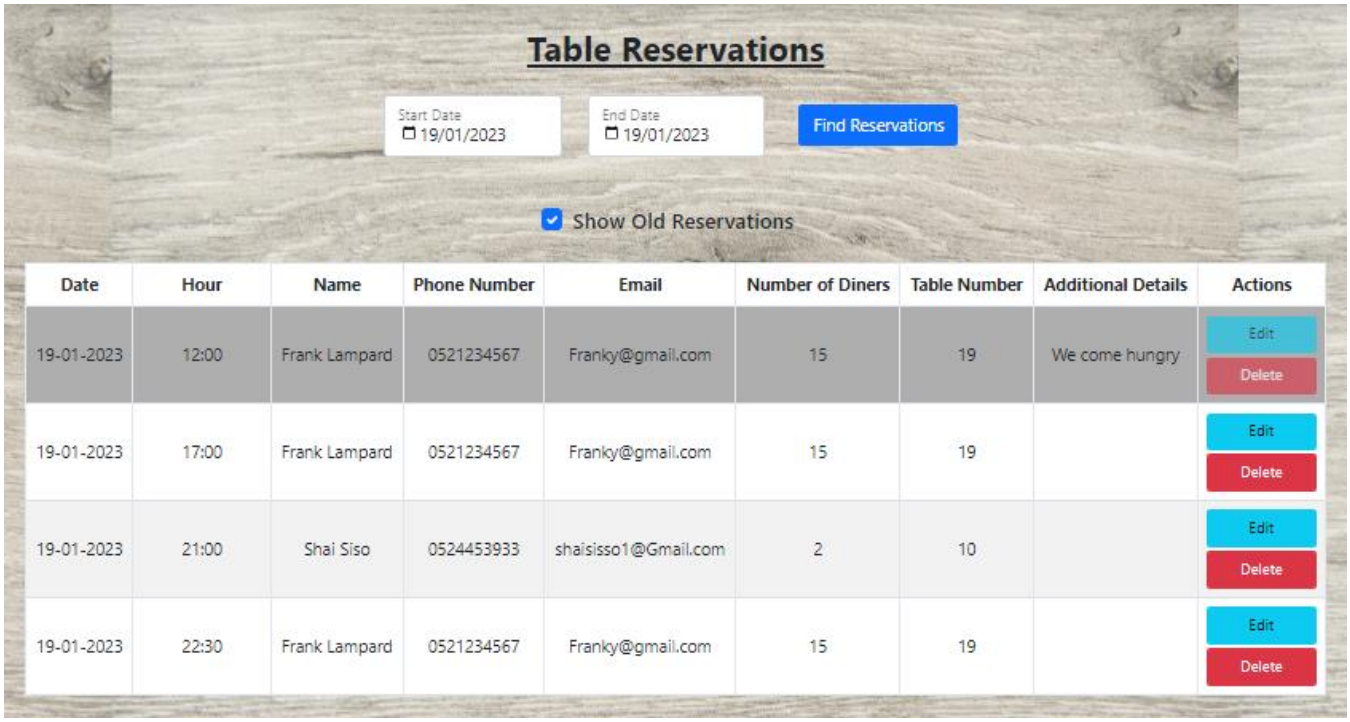
  - Exit shift: the employee will press “exit shift” at the end of his shift.
  - My shifts: the employee will be able to watch his previous shifts and to see if there are shifts that weren't approved yet.

- Profile: the employee will be able to edit his personal details, and log-out from the system.

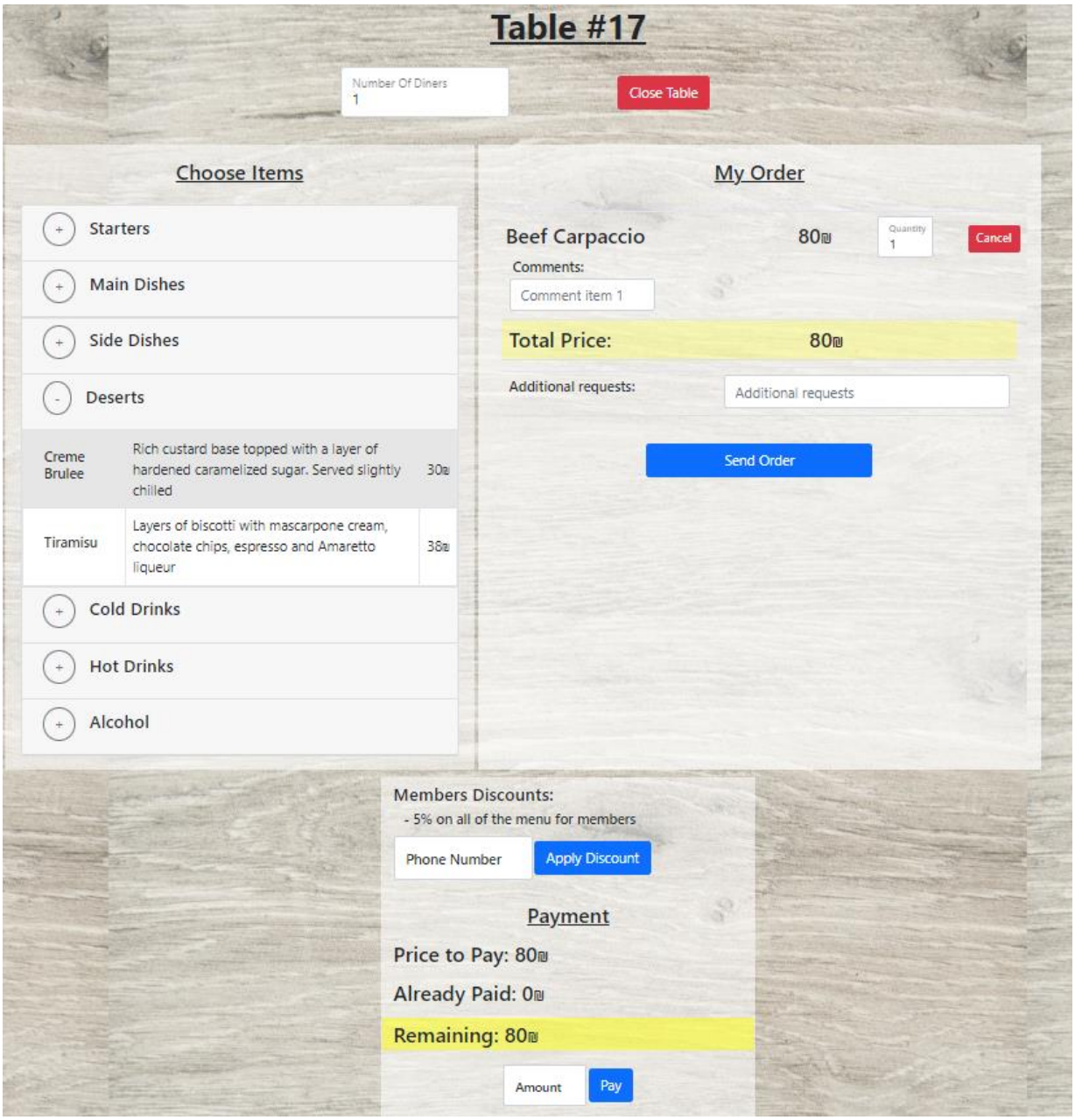


The activities that can be done depend on the permissions that are derived from the role. The permissions are hierarchical, and each level includes the levels below:

- Hostess: Reservations – add, update, delete



- Waiter, Bar: Table orders- choose table, open table, add new order, add item to order, cancel item request that will be sent for approval to the shift manager, payment, apply member discount.



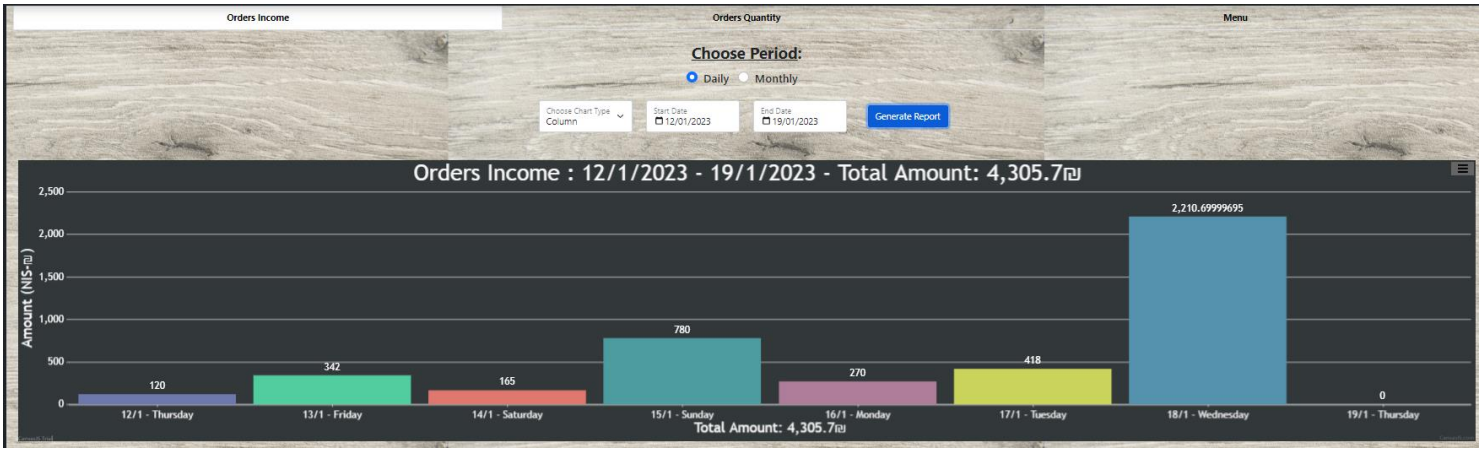
- Shift manager, Bar manager, Delivery manager, Kitchen manager:
  - Tasks-
    - Approve/disapprove: shifts, cancel item requests

- External orders – add, update, delete, change status, assign delivery guy from the ones that are currently on shift.

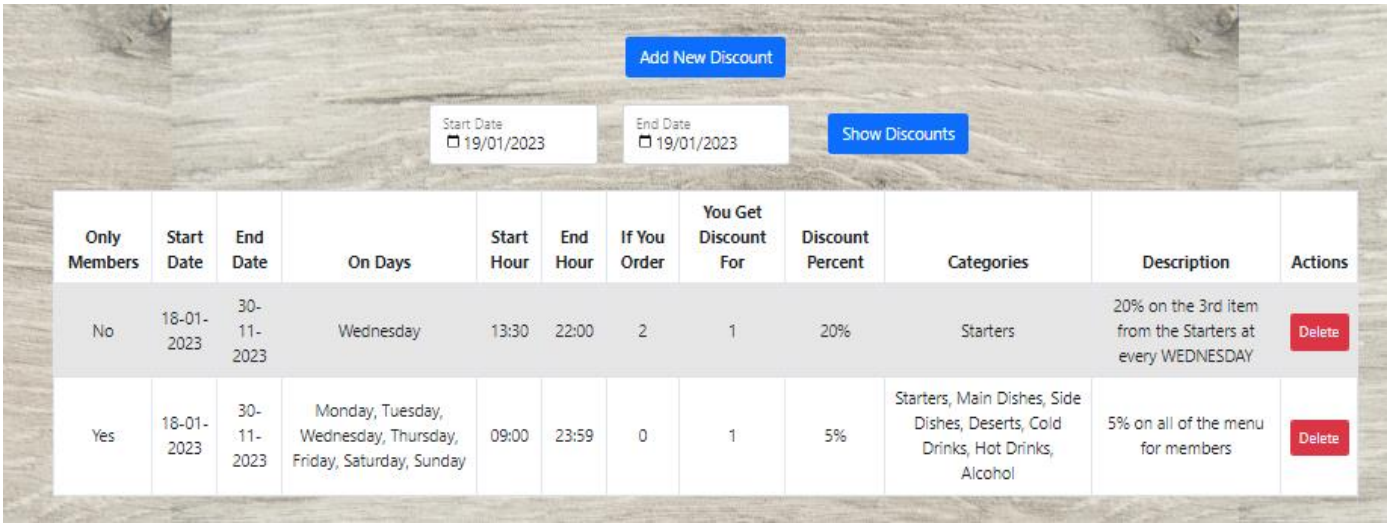
*When one of the manager is starting a shift or trying to cancel item from an order, it is done automatically without need for approvement

- Manager:
  - Reports – by choosing dates range
    - Orders income - daily / monthly
    - Orders quantity – daily / monthly
    - Menu – canceled items requests / ordered items

    The manager can choose the type of the chart and can export the report to JPEG/PNG or print the report.

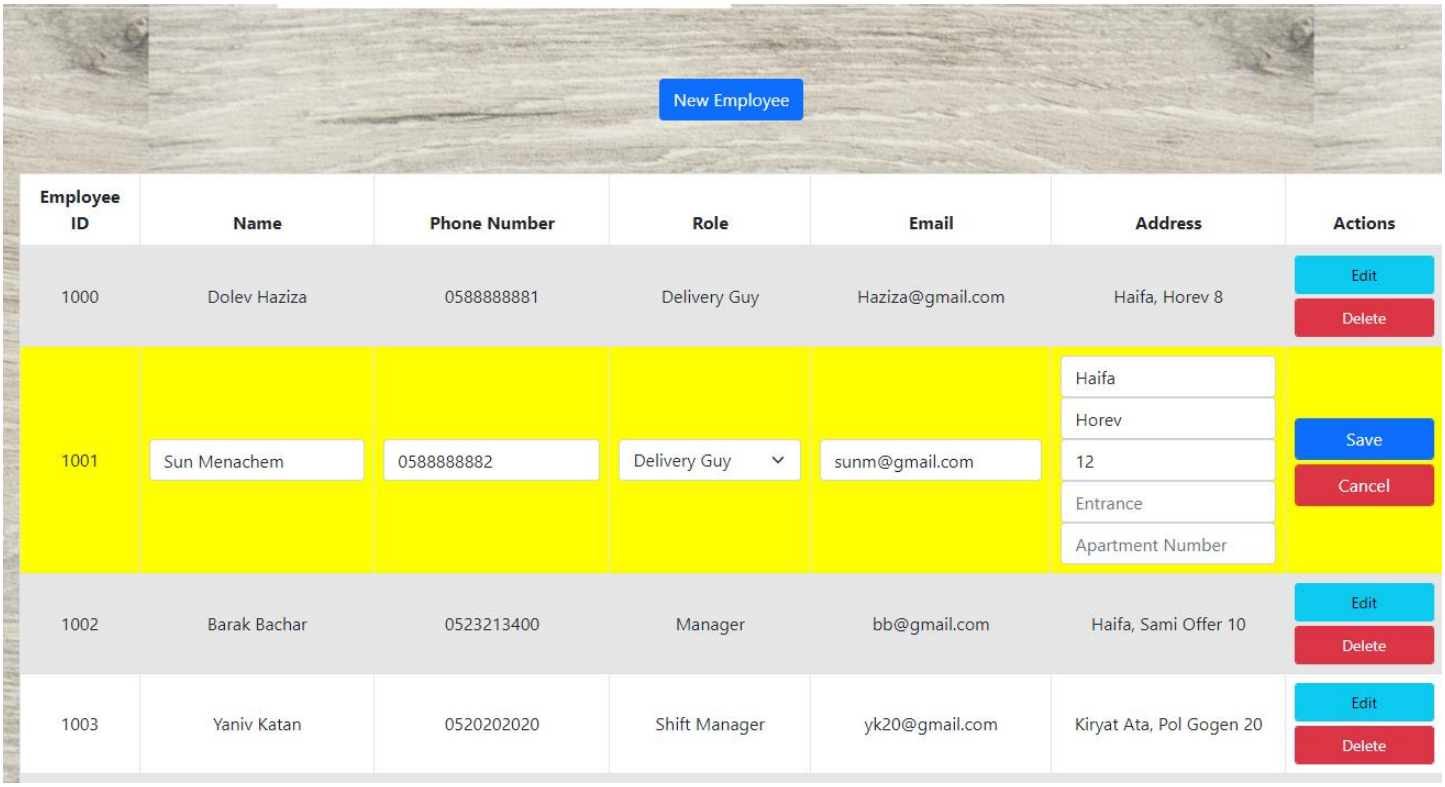


- Management –
  - Discounts – add, delete



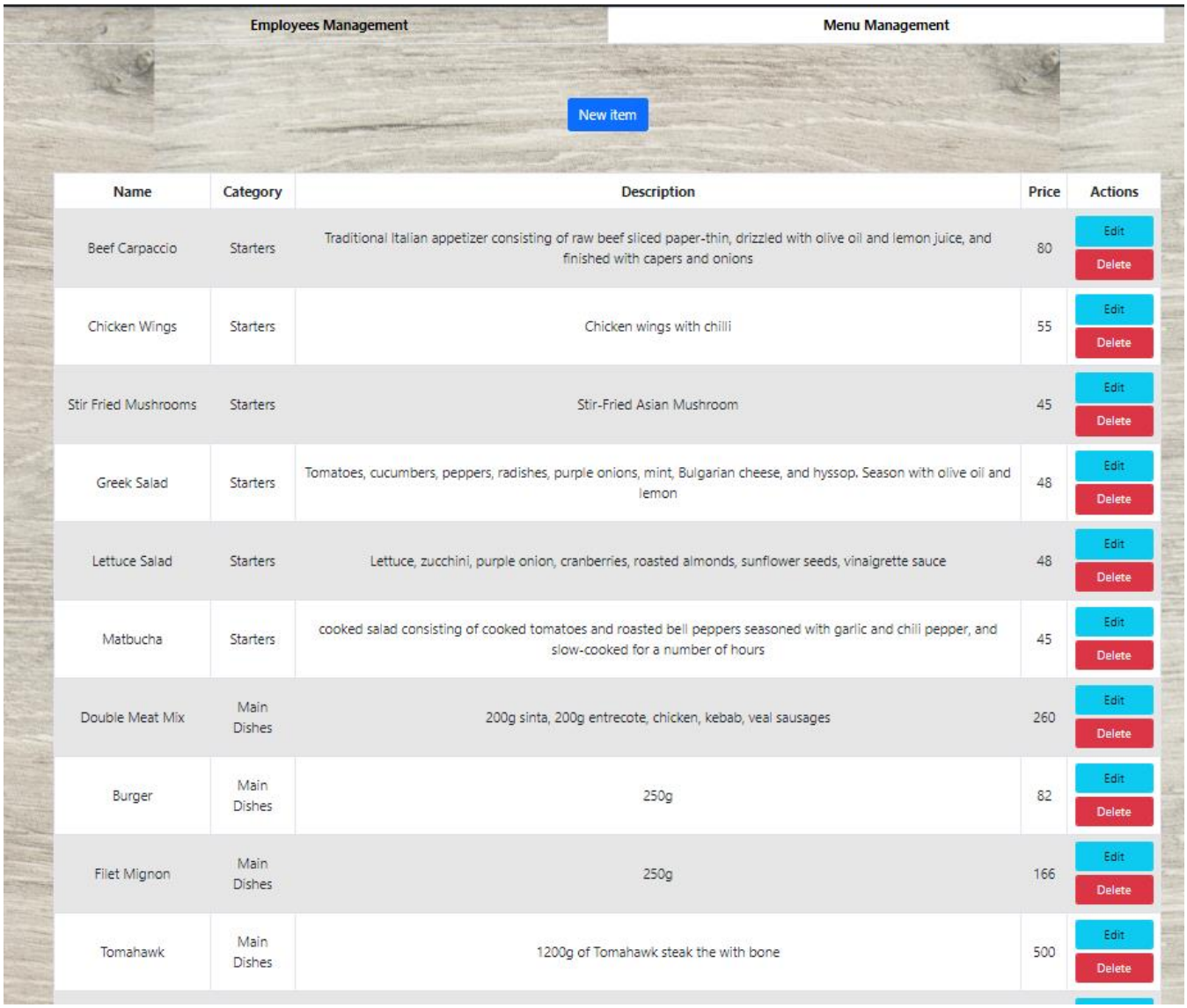
Only Members	Start Date	End Date	On Days	Start Hour	End Hour	If You Order	You Get Discount For	Discount Percent	Categories	Description	Actions
No	18-01-2023	30-11-2023	Wednesday	13:30	22:00	2	1	20%	Starters	20% on the 3rd item from the Starters at every WEDNESDAY	Delete
Yes	18-01-2023	30-11-2023	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday	09:00	23:59	0	1	5%	Starters, Main Dishes, Side Dishes, Deserts, Cold Drinks, Hot Drinks, Alcohol	5% on all of the menu for members	Delete

  - Employee – add, update, delete

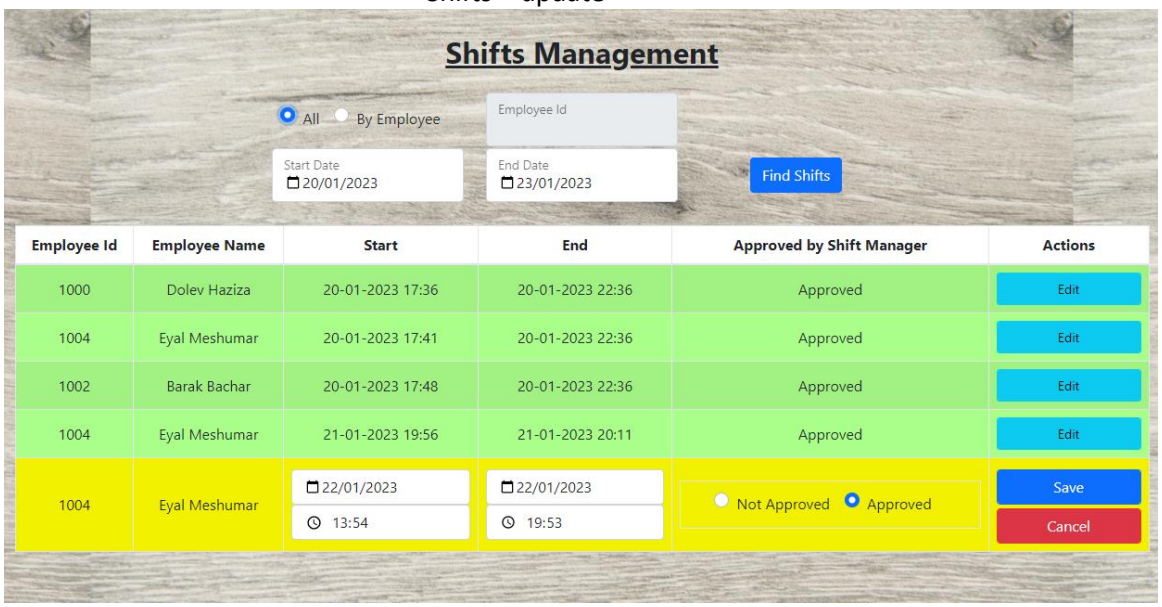


Employee ID	Name	Phone Number	Role	Email	Address	Actions
1000	Dolev Haziza	0588888881	Delivery Guy	Haziza@gmail.com	Haifa, Horev 8	Edit Delete
1001	Sun Menachem	0588888882	Delivery Guy	sunm@gmail.com	Haifa, Horev, 12, Entrance, Apartment Number	Save Cancel
1002	Barak Bachar	0523213400	Manager	bb@gmail.com	Haifa, Sami Offer 10	Edit Delete
1003	Yaniv Katan	0520202020	Shift Manager	yk20@gmail.com	Kiryat Ata, Pol Gogen 20	Edit Delete

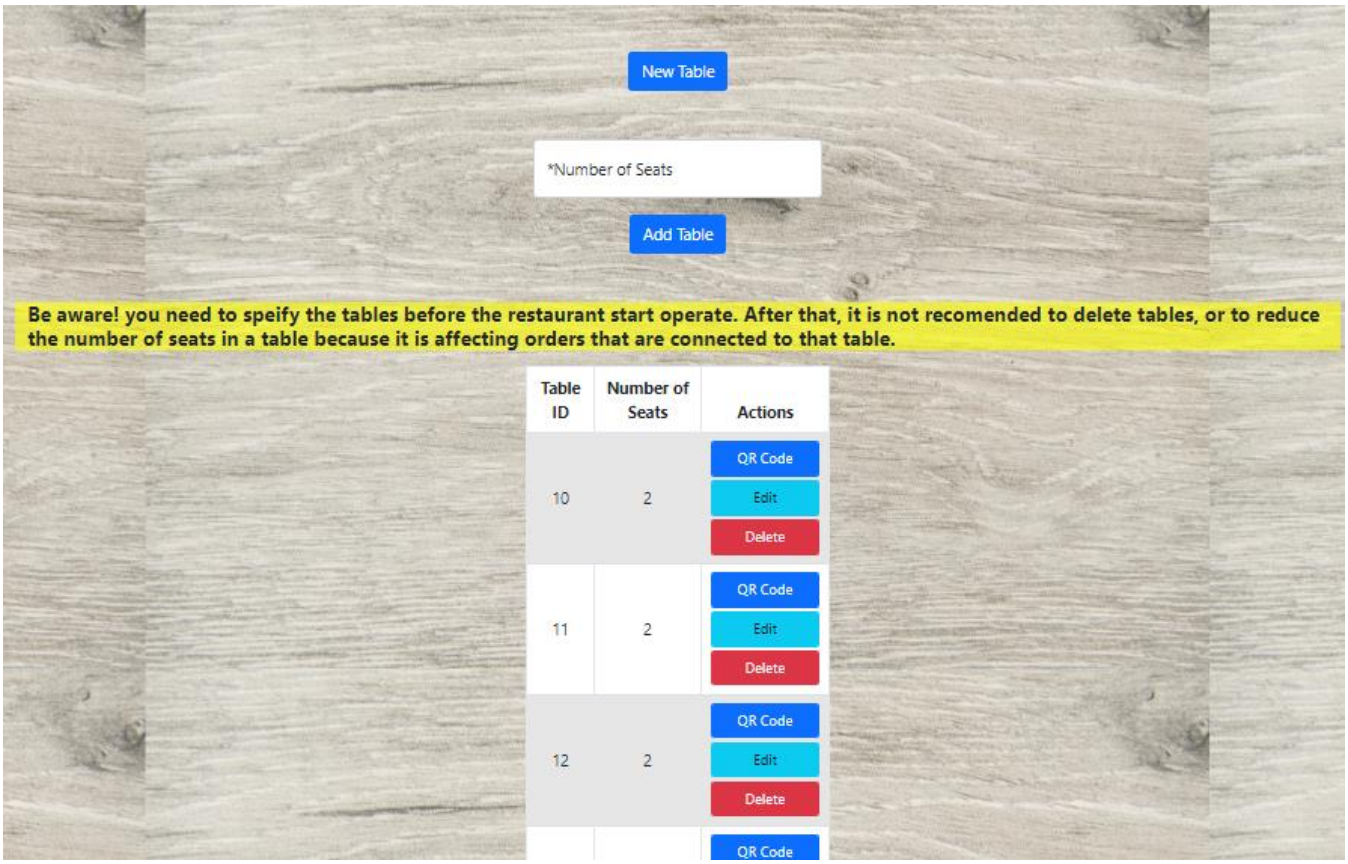
- Menu – add, update, delete



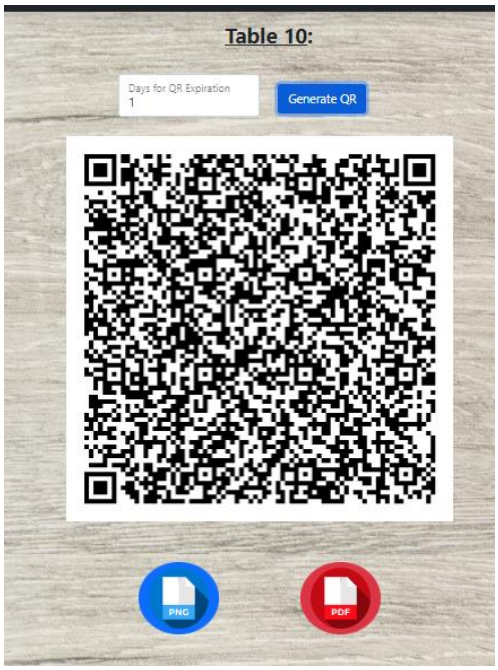
- Shifts – update



- Restaurant arrangement
  - ✓ Tables – add, update, delete



  - ✓ QR-Code – generate new barcodes and define for how long it will be valid. There is the option to export the barcode to a PNG/PDF file.



** In order to send SMS for each registered user using Twilio, we had to purchase a premium account. We have a limited account (free), so there can be only one phone number that can receive SMS (The phone that registered to Twilio account).

** Emails are sent to each valid mail that will be typed.

Website link: www.smartfood-project.link

GitHub link:

- Smart-Food server: https://github.com/shaisiso/SmartFood-Server
- Smart-Food Client: https://github.com/shaisiso/SmartFood-Client

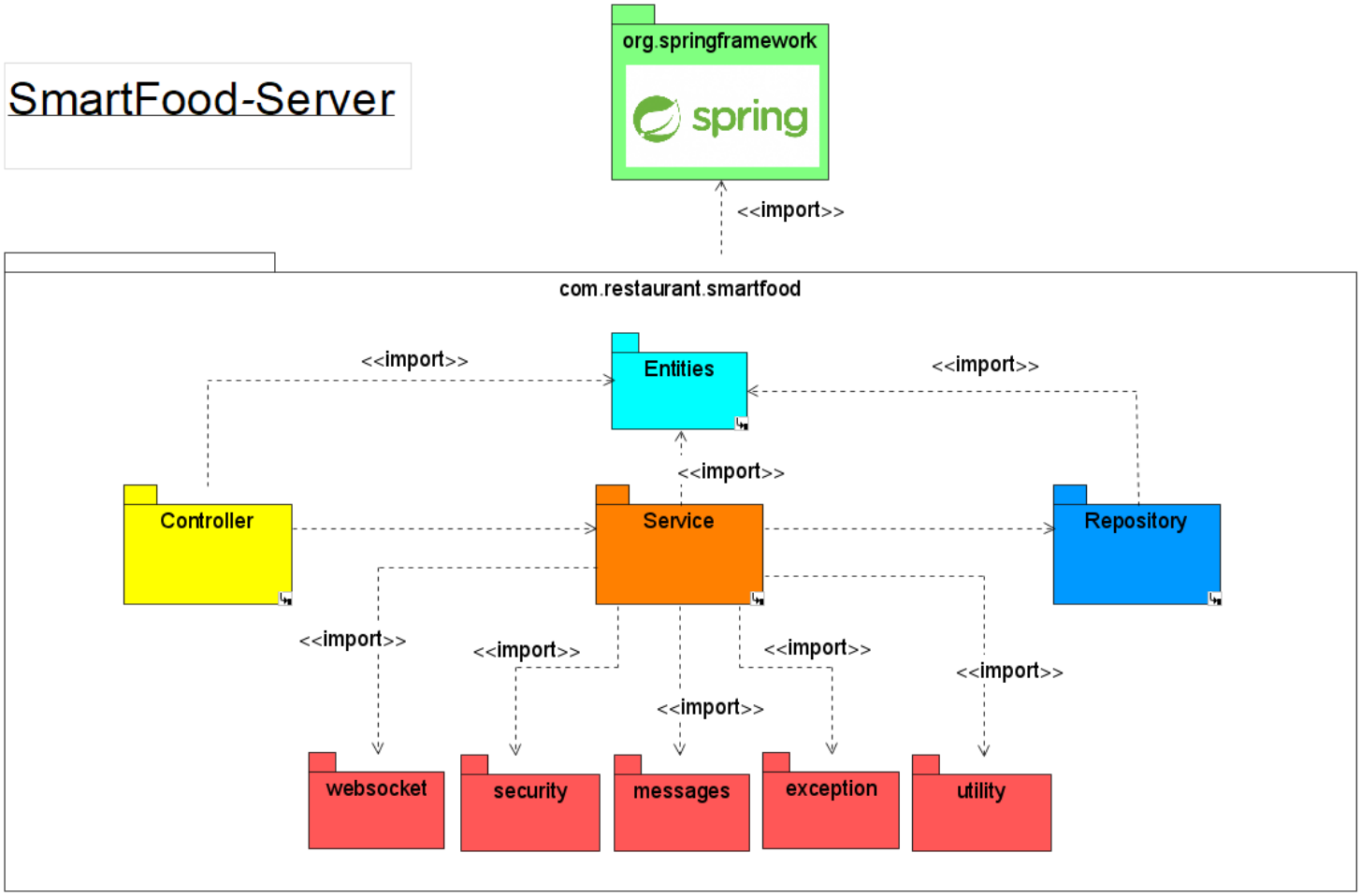
Existing type of users for website experience:

- Member:
  - Phone number: 0521234567
  - Password: 123456
- Hostess:
  - Phone number: 0500000001
  - Password: 123456
- Waiter:
  - Phone number: 0527272727
  - Password: 123456
- Delivery guy:
  - Phone number: 0588888881
  - Password: 123456
- Shift manager:
  - Phone number: 0520202020
  - Password: 123456
- Manager:
  - Phone number: 0523213400
  - Password: Aa123456

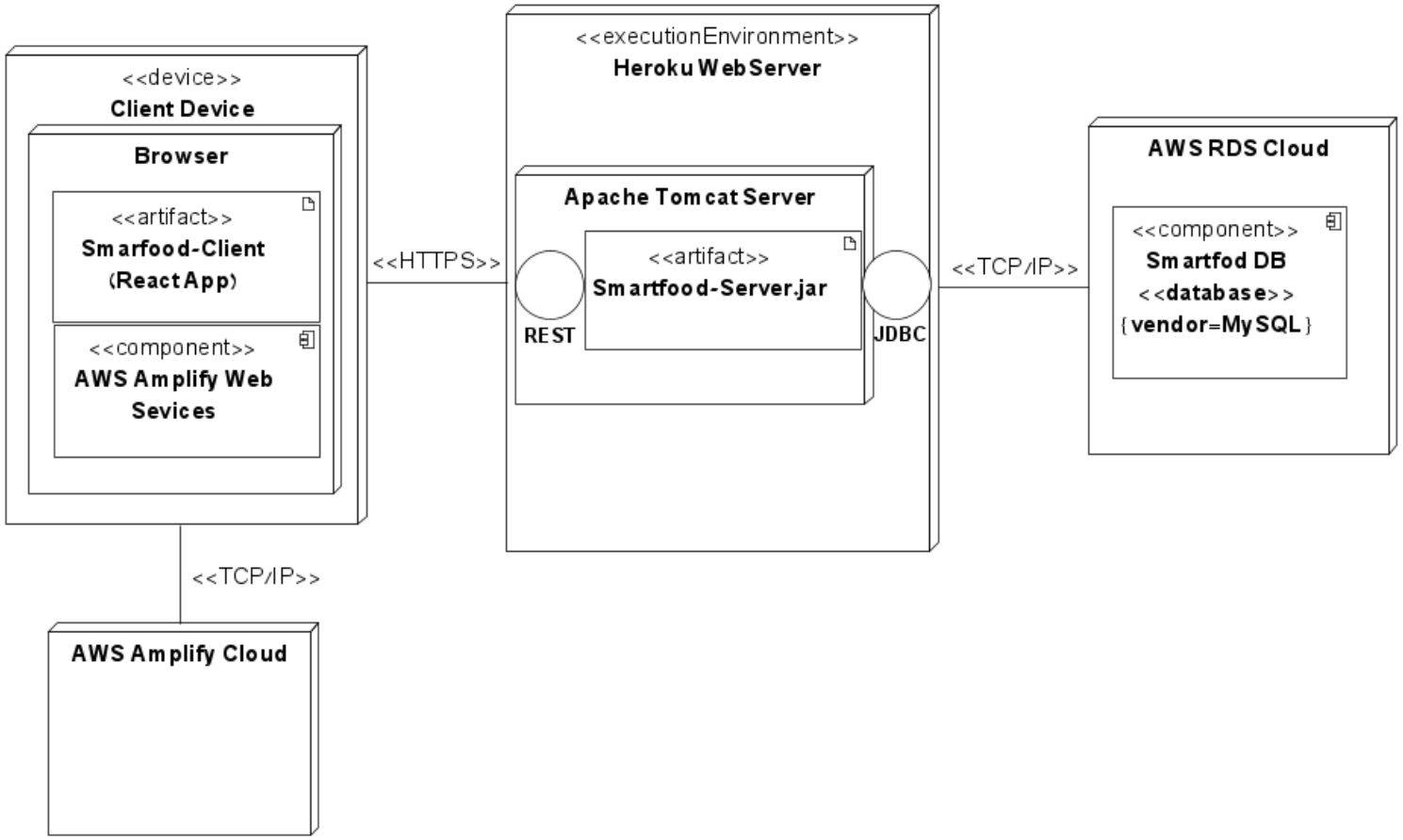
2.2 Maintenance Guide

In order to enable future use of the project's products to implement updates or changes, a maintenance guide is attached. One change for example could be real interaction with payment services such as credit cards companies or PayPal for example.

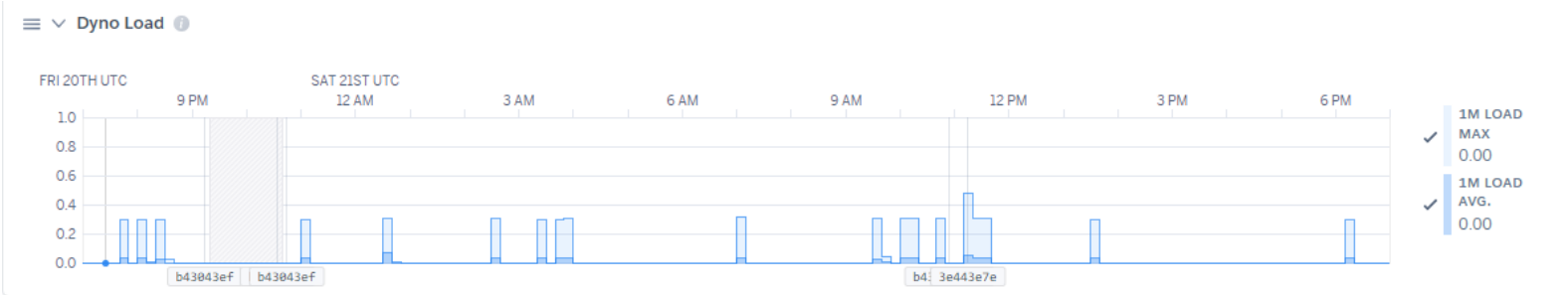
2.2.1 Package Diagram

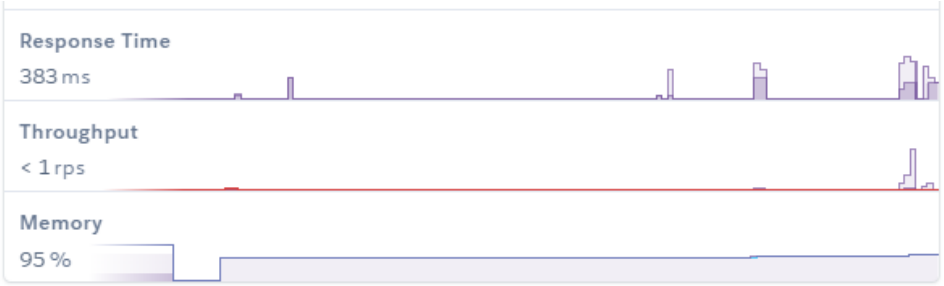


2.2.2 Deployment Diagram



Heroku supplies some information about the server environment, and we can use it to maintain our system and to get extra data like number of requests, busy hours and more.





2.3 Database and system

2.3.1 Database schemes

- Cancel_item_request: id bigint PK, date datetime(6), is_approved bit(1), reason varchar(255), item_in_order_id bigint, menu_item_item_id bigint, order_of_table_id bigint
- Deliveries: id bigint PK, delivery_guy_id bigint
- Discounts_categories: discount_discount_id bigint, categories varchar (255)
- Discount_days: discount_discount_id bigint PK, days int PK
- Discounts: discount_id bigint PK, discount_description varchar(255), end_date date, end_hour time, for_members_only bit(1), if_you_order int, percent int, start_date date, start_hour time, you_get_discount_for int
- Employees: password varchar(255), role varchar(255), id bigint PK
- Items_in_orders: id bigint PK, item_comment varchar(255), price float, item_item_id bigint, order_id bigint
- Members: password varchar(255), id bigint PK
- Menu-items: item_id bigint PK, category varchar(20), description varchar(255), name varchar(30), price float
- Orders: id bigint PK, already_paid float, date date, hour time, order_comment varchar(255), original_total_price float, status varchar(255), total_price_to_pay float, person_id bigint
- Orders_of_table: number_of_diners int, id bigint PK, table_table_id int
- Person: id bigint PK, apartment_number int, city varchar(85), entrance char(1), house_number int, street_name varchar(85), email varchar(255), name varchar(20), phone_number varchar(10)
- Restaurant_tables: table_id int PK, is_busy bit(1), number_of_seats int
- Shifts: shift_id bigint PK, is_approved bit(1), shift_entrance datetime(6), shift_exit datetime(6), employee_id bigint
- Table_reservations: reservation_id bigint PK, additional_details varchar(255), date date, hour time, number_of_diners int, person_id bigint, table_table_id int
- Take_away: id bigint PK
- Waiting_list: id bigint PK, date date, hour time, number_of_diners int, was_notified bit(1), person_id bigint

2.3.2 System environment

2.3.2.1 Deployment system environment

- Server: running on Heroku machine.
  URL: https://smartfood-server.herokuapp.com/
  (URL for API is /api/{some specific route})
- Client: deploy to AWS Amplify
  URL: https://www.smartfood-project.link/
- Database: deploy to AWS RDS
  URL for connection: jdbc:mysql://smartfood-db.cfveqwj8ivr1.eu-central-1.rds.amazonaws.com:3306/sys

2.3.2.2 Installation instruction (local environment)

- Server:
  - Java 11
  - Maven
  - MySQL
  - *Intellij IDEA – (not required but recommended)
  - Installation instructions:
    - Open terminal from Intellij IDEA (or another terminal).
    - Type:
      "Git clone https://github.com/shaisiso/SmartFood-Server.git"
    - Build (build with Maven after dependencies were fetched).
    - Run server (SmartfoodServerApplication.java)

    * There is a file in src\main\resource\application.yml where you can choose the environment of this app. You can choose if you use a local database or the database on AWS cloud.

    * When choosing "dev" profile there is also a class in the service package which is called "DBInit" and there are methods for initializing the DB in your local environment. The method public void run(String... args) do the initialize so put down the comment for the methods there.

* We attached a file which is called “env.properties” and stored in it all the sensitive data. You need to add it to the root folder of the project (“SmartFood-Server”).

After this, you can run the app and then the server is listening to requests and the URL http://localhost:{PORT} where PORT is defined in the .yml file. (The default port is 8080 if no port was configured)

- Client:
  - Internet Browser (Chrome is recommended for full support).
  - NPM (which is included in NodeJS).
  - Installation instructions:
    - Open terminal from VS code (or another terminal).
    - Type:
      “git clone https://github.com/shaisiso/SmartFood-Client.git”
    - Type: “cd SmartFood-Client”
    - Type: “npm install”
    - Type: “npm start”
      And then the browser should be open with our website at the url: http://localhost:3000 (if no other port was configured).

      * there is a file src/utility/Utils.js where you can change the urls for the environment (from deployment to local)

* When starting the server you need to wait until you see in the terminal “Tomcat started on port(s):{PORT}”

* The server needs to run first in order for the client to run successfully.

3. References

Tabit: www.tabitisrael.co.il
Walt: www.walt.com
Mishloha: www.mishloha.co.il
Pluralsight: www.pluralsight.com
Twilio SMS API: www.twilio.com
Spring: www.spring.io/projects/spring-boot