Global Distributed Software Development
Master Team Project WS 2022
“ReservEat”
Milestone 5
Team 1:

Utsav Shrestha
Team Lead, Frontend Lead
utsav.shrestha@informatik.hs-fulda.de

Muhammad Bilal
Frontend Developer
muhammad.bilal@informatik.hs-fulda.de

Sanjay George
Backend Lead, DevOps
sanjay.varkey-george@informatik.hs-fulda.de

Omer Zogubi
Backend Developer, GitHub Admin
omer.zogubi@informatik.hs-fulda.de

Parthiv Jani
Frontend Developer
parthiv-yogesh-kumar.jani@informatik.hs-fulda.de

Kristijan Lazeski
Backend Developer, Database admin
kristijan.lazeski@informatik.hs-fulda.de

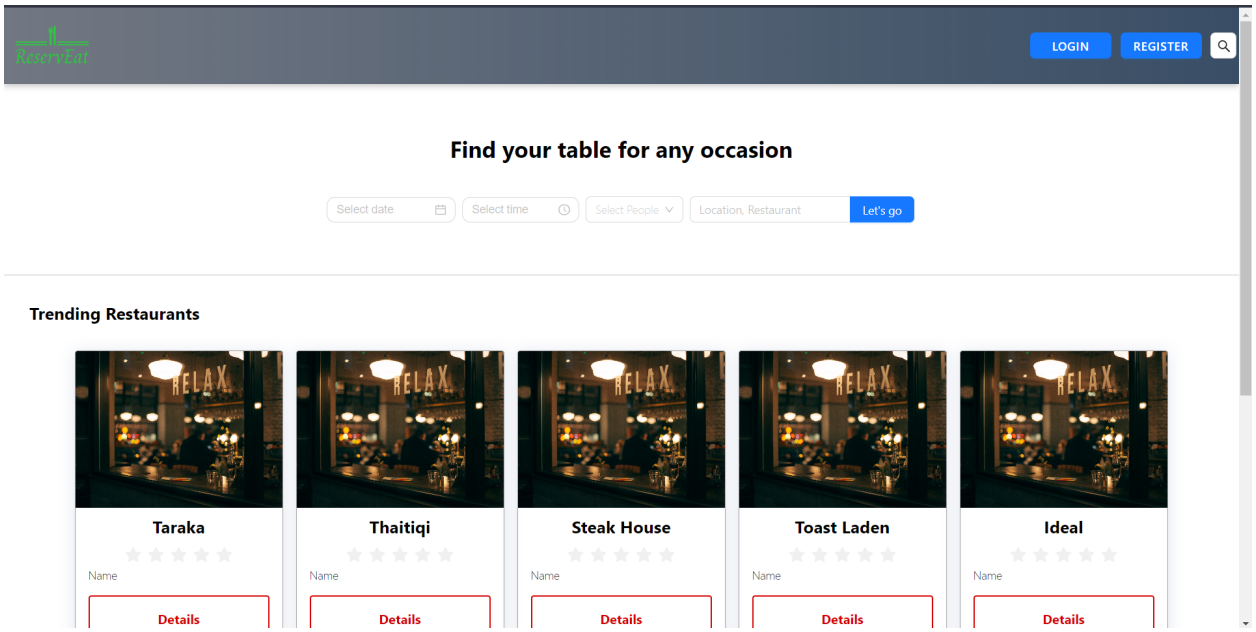
Date Submitted	Date Revised	Revision Summary

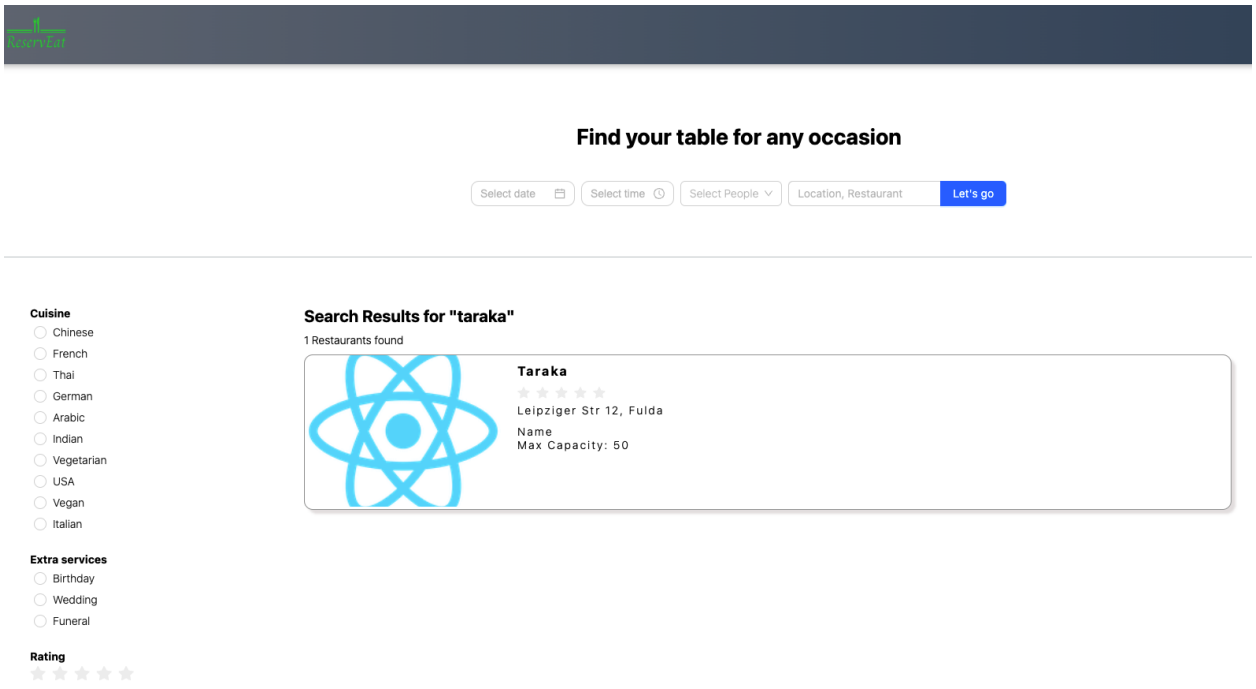
1. Webpage

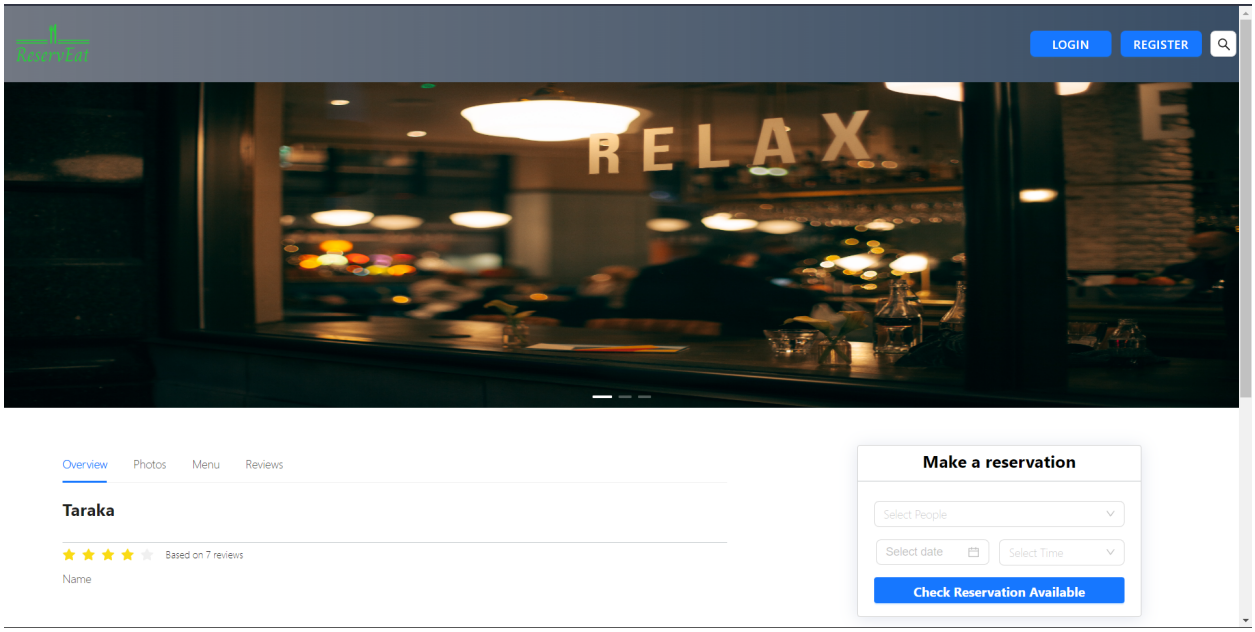
The Webpage is to be found under:

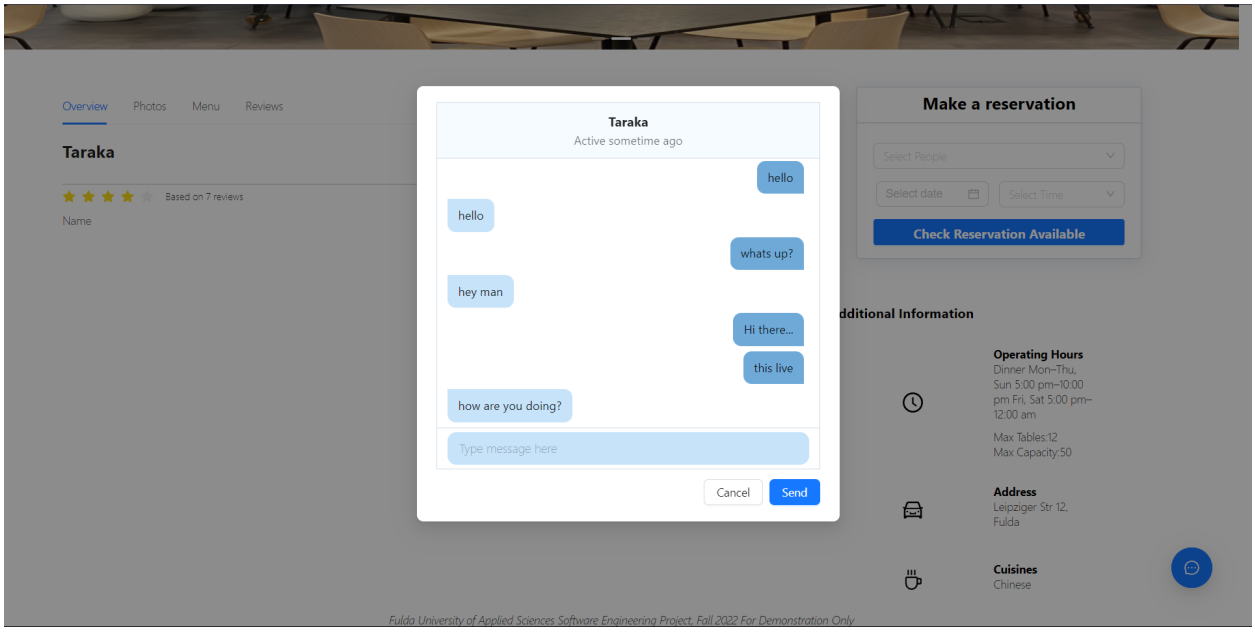
→ https://gdsd1restaurant.live/

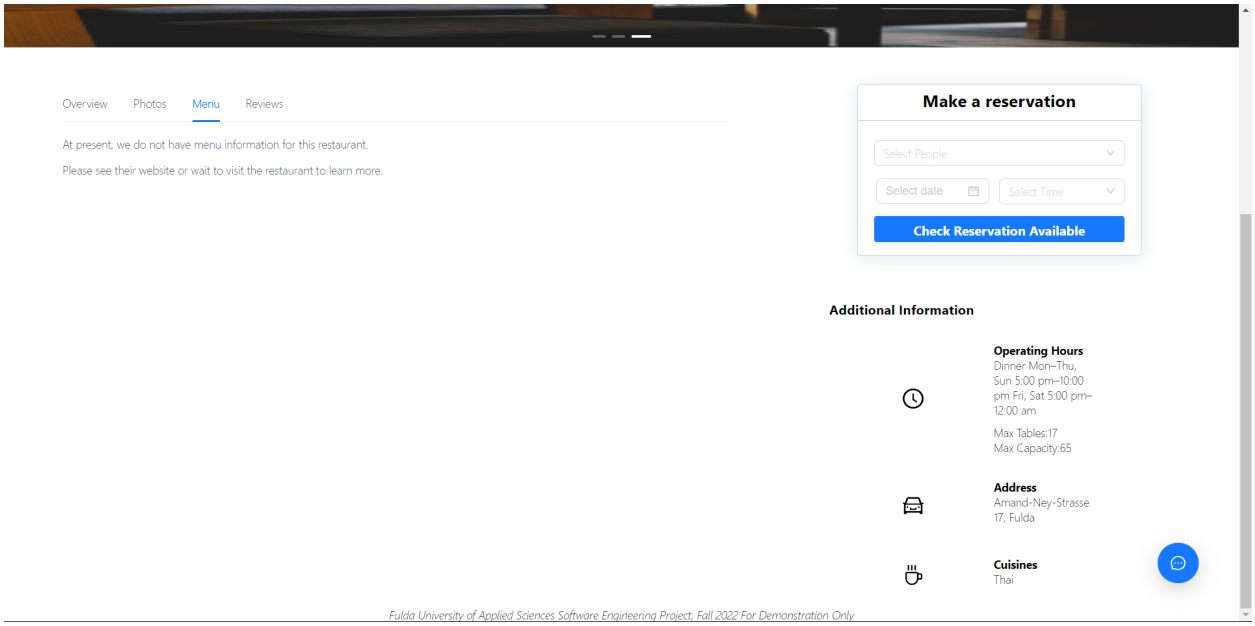
2. Screenshots

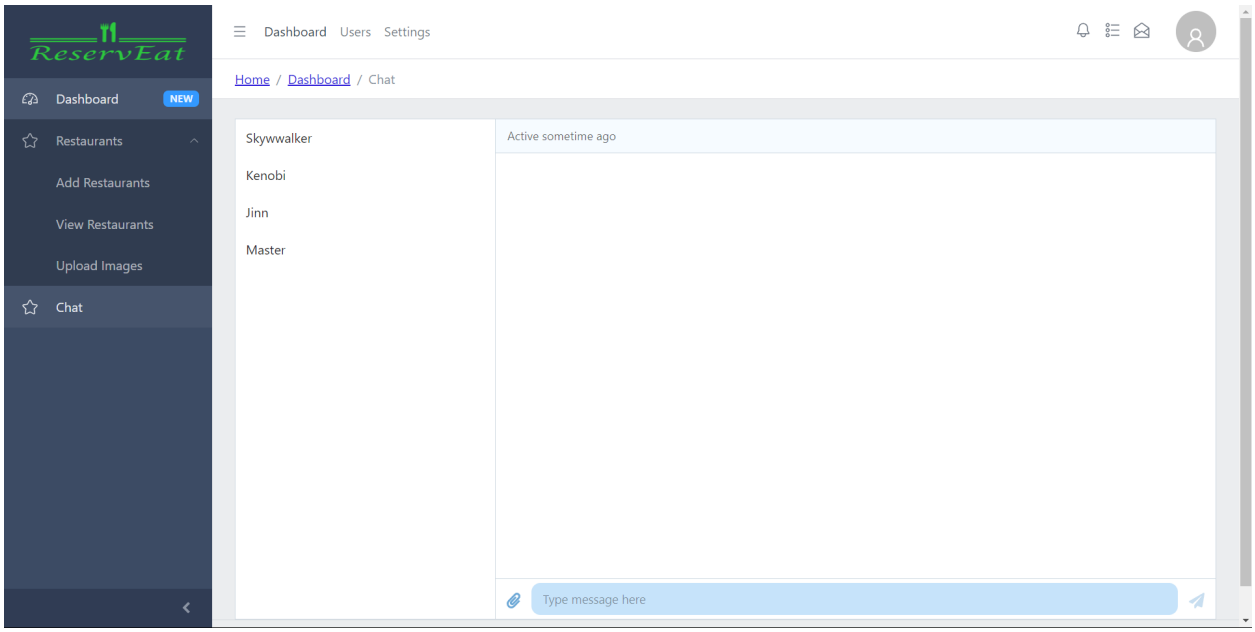





LOGIN
REGISTER
RELAX
Overview
Photos
Menu
Reviews
Taraka
Based on 7 reviews
Name
Make a reservation
Select People
Select date
Select Time
Check Reservation Available


Overview
Photos
Menu
Reviews
Taraka
Based on 7 reviews
Name
Taraka
Active sometime ago
hello
hello
whats up?
hey man
Hi there...
this live
how are you doing?
Type message here
Cancel
Send
Make a reservation
Select People
Select date
Select Time
Check Reservation Available
dditional Information
Operating Hours
Dinner Mon–Thu, Sun 5:00 pm–10:00 pm Fri, Sat 5:00 pm–12:00 am
Max Tables:12
Max Capacity:50
Address
Leipziger Str 12, Fulda
Cuisines
Chinese
Fulda University of Applied Sciences Software Engineering Project, Fall 2022 For Demonstration Only





3. Summary of Requirements

ID	Functional Requirement	Priority

1	The Application is a Vendor for Multiple Restaurants	1
2	User can Search for Restaurants by name, cuisine, food, and rating	1
2.1	Search results can be filtered additionally and sorted	1
3	User can reserve a table and specify the number of seats in a given Restaurant	1
3.1	User reserves one or Multiple Tables at the Restaurant	1
3.2	User specifies the number of seats	1
4	Restaurants can be added, removed and updated by Restaurant owners	1
4.1	Restaurants can be Added by the Restaurant Owners	1
4.2	Restaurants can be Removed by the Owners of given Restaurant	1
4.3	Restaurants can be Updated by the owners of said Restaurant	1
5	There is a Grace Period to Cancel or update the booking of the table	1
5.1	Every User can Cancel the reservation up to 2h before said reservation	1
6	The number of booked seats are specified per table	1
7	Users can view Menus of the Restaurants	1
8	Users search for restaurants, make/change/cancel reservations and also post reviews.	1
8.1	User can cancel Reservation	1
8.2	User can change Reservation	1
8.3	User can Post Reviews	1
9	User can View the Restaurant Reviews	1
10	Every booking is associated with an account	1
11	One Account can not be associated with multiple users	1
12	User can chat with the Restaurant Host	1
13	User accumulates Reward Points for every reservations with the application	1

ID	Functional Requirement	Priority

1	Restaurants can add Extra Services with themes through Application	2
1.1	Restaurants can add Themes as a Service	2
1.2	Restaurants can add Special Foods or Beverages as a Service to be clicked/booked beforehand	2
2	Restaurants can change the availability status of any table at any time.	2
2.1	Restaurant owner updates the number of free seats for given table	2
3	Restaurant Owners can deny or approve of a booking	2
3.1	Restaurants owner can disapprove a booking made for given number of Seats at a given Time	2
3.2	Restaurants owner can approve a booking made for given number of Seats at a given Time	2
4	User can choose Event Themes	2
5	User can only book two tables daily (to avoid spam reservations)	2
6	Site administrator approves restaurant info for posting. Also deals with typical admin duties like managing user registrations etc.	2
6.1	Site Admin approves Restaurant info for Posting	2
6.2	Site Admin Identifies and addresses usability Problems	2
6.3	Site Admin manages user Registration	2
6.4	Site Admin tracks and analyses the Traffic that is being made on our Website	2
6.5	Site Admin Develops Training Manual for use for Restaurants	2
6.6	Site Admin Flags user if there is a valid reasoning	2
7	User can Check his Reservations	2

ID	Functional Requirement	Priority
1	Users can Provide Reviews for the Restaurant which they have visited through our Application	3
1.1	User provides Text of the Review	3

1.2	User gives a Number Rating between 1 and 5	3
2	Restaurant owners can Flag users for their lack of attendance on booking	3
2.1	Restaurant owners submit a report for time and place and reasoning of why said user should be flagged	3
3	Admin can Ban Restaurants	3
4	Host/hostess reviews daily calendar and checks and greets incoming guests This List might be Edited as the Project goes on.	3

4. Executive Summary

4.1 Motivation

Your kid has Birthday soon and wants to organise something in a restaurant? Do you need a night out with friends? Wanna enjoy yourself with a nice dinner? Or just wanna go out and grab a drink? No problem, our Website **ReservEat** has got you covered. With **ReservEat** you can book a table in any restaurant in the area. See which ones are free, and choose the one that fits you the most. There is no more need to have a phone list of all the restaurants. One does not always update it and checks the new restaurants. Don't go and seek out restaurants, let the restaurants seek you out. Best fit for the best night.

4.2 Functions and Services

Our Websites offer a very Sophisticated and easy to use User Interface (UI) that also gives the User the feeling that they have everything in grasp. Hereby is the User Experience (UX) on a really high level. As a User one can Look and Search for the Restaurants around. After choosing one the user can book a number of seats for a given date and time. Everything presented in an easy to use Graphical User Interface (GUI). After booking the User shows up at the restaurant on the chosen day at the chosen Time.

Our website has an admin panel where a restaurant owner can manage their restaurants and update their details. The restaurant management can chat with users if they have some enquiries and flag them for attendance.

4.3 Why should you use our Website

ReservEat is an application that gives you the best experience to reserve a table in your favourite restaurant in a very smooth and simple way. We have made it easier for the users to

communicate with us by introducing the chat functionality. The users can always message us about the enquiries they have. Every restaurant has their own chat service where the users can communicate with. The users can leave their reviews about the restaurants and one can have an idea of how a restaurant is.

4.4 About us

We are a Team of Motivated Master Students that have a goal to bring this problem not to be a problem anymore. We Utsav, Sanjay, Bilal, O¨ mer, Parthiv and Kristijan will continue to support and bring the Website to the masses and the life of everyone. We are developing this website for the netizens of Fulda to help them search for restaurants and make it easy for them to book a restaurant.

5. Personae and main Use Cases

5.1 Personae

5.1.1 Administrator

Administrator is the person who moderates all content on the application. The responsibility of the administrator includes approving restaurant listings, images, reviews as well as rejecting content that does not meet the required standards. An Administrator should be familiar with the flow of the software.

5.1.2 Users

Users are the end consumers of the application who search for restaurants and make reservations. Users can use the functionalities of the application to find a restaurant based on their needs and reserve seats at the place for their desired date and time. Additionally, they can connect with the restaurant host to clarify queries and concerns about the restaurant, cuisine or timings. Users can then choose to leave a review about their dining experience in the application, to help other users with their decision.

5.1.3 Restaurant Administrator

A restaurant administrator is a person who manages all data of a restaurant. It is the responsibility of the restaurant administrator to create an account for a restaurant and also authorise the restaurant host to access the reservations. Restaurant administrator adds all details about the restaurant including images and menu. They should be able to manage their hosts and synchronise with them.

5.1.4 Restaurant Host

A restaurant host has the access to view all reservations for a particular date and make necessary arrangements. The host will also receive messages from users and can clarify their queries through the chat. The host can also flag users in our application who did not show up without cancelling their reservation, which inadvertently causes loss of revenue for the restaurant. The restaurant host should have organising and communication skills to manage the customers.

5.2 Main Use Cases

5.2.1 Account Registration

All users of the application, as listed in the previous section, need to register an account to use the platform. Due to the nature and context in which the application is being built, only email ids affiliated with Hochschule Fulda and ending with ‘hs-fulda.de‘ will be allowed registration.

5.2.2 Managing Restaurant Details

A restaurant administrator can upload all details regarding his restaurant to be displayed to the users. These details will cover all basic data a user needs to make an informed decision about reserving a restaurant, such as the name of the restaurant, its location, items on the menu with price, images, etc.

5.2.3 Search Restaurants

Users of the application can search for restaurants in a location and find all details about the restaurants available in the area.

5.2.4 Reserve Restaurants

Users can reserve a restaurant for a desired date and time and also choose the required number of seats. Additionally, users can also modify or cancel their reservation. The Restaurant Host will be able to view all reservations of a day to plan and manage their customers.

5.2.5 Reviews

Users who have booked reservations through the application can leave reviews of their dining experience. The reviews can include text and images and will be useful for other users with their

decision in the future. The reviews also serve as a feedback for the restaurant and the Restaurant Host.

5.2.5 Chat

Users can chat with the Restaurant Host to clarify any queries that could range from doubts about a particular kind of cuisine to questions about organising events in the restaurant. Such a chat feature helps avoid miscommunication or misunderstandings about the services provided by the particular restaurant, thus contributing to a satisfactory experience for the user.

6. List of main data items and entities

The main data items and entities used in this project are:

6.1 Registration

A user needs to be registered and logged in to be able to reserve a restaurant. Similarly a restaurant admin should register themselves before they are able to list their restaurants in the system. The restaurant admin then can register their host/hostess in the system.

6.2 Restaurant

Restaurants are the main entity of our system. All the data about the restaurants are added and shown in the website so that a user can choose the restaurant according to their liking and reserve them.

6.3 Roles

The system has 4 roles that an actor can be assigned to. Each role has its own responsibility. The roles are listed below:

- **Users:** Searches for a restaurant and books it.
- **Restaurant Administrator:** Upload the information about the restaurant and manage the data.
- **Site Administrator:** Approves or rejects the changes made by the restaurant admins and Restaurant hosts.
- **Restaurant Host:** Reviews the reservations made by the users and manages the user activities.

6.4 Reviews

A user can post a review about a restaurant in the restaurant details page. They can rate by giving a number of stars and adding a review of how they felt about the restaurant. The review needs to be approved by system admin. The restaurant can flag the user reviews if they deem it inappropriate.

6.5 Chat

A user can chat about the more enquiries they have regarding the restaurants with the restaurant host. The chat will be available in each restaurant's details page.

6.6 Flag

A restaurant admin or host/hostess can flag a user or their reviews. The flagged user’s reservation needs to be then approved by the restaurant management to finalise it.

7. Initial list of functional requirements

The list of the Functional requirements is as follows:

1. The Applications is a Vendor for Multiple Restaurants
2. User can Search for Restaurants
3. User can reserve a table and specify the number of seats in a given Restaurant
4. Users can Provide Reviews for the Restaurant which they have visited through our Application
5. Restaurants can be added, removed and updated by Restaurant owners
6. Restaurants can add Extra Services with themes through Application
7. There is a Grace Period to Cancel or update the booking of the table
8. Restaurants can change the availability status of any table at any time.
9. The number of booked seats are specified per table
10. Restaurant owners can Flag users for their lack of attendance on booking
11. Restaurant Owners can deny or approve of a booking
12. Admin can Ban Restaurants

13. User can choose Event Themes

14. User can only book two tables daily (to avoid spam reservations)

15. Users can view Menus of the Restaurants

16. User can View the Restaurant Reviews

17. Site administrator approves restaurant info for posting. Also deals with typical admin duties like managing user registrations etc.

18. Users search for restaurants, make/change/cancel reservations and also post reviews.

19. Host/hostess reviews daily calendar and checks and greets incoming guests This List might be Edited as the Project goes on.

8. List of non-functional requirements

Hereby are a High-Level Version of the non-functional specifications (how the app is delivered and other constraints) presented:

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in Milestone 0. Application delivery shall be from chosen cloud server
2. Application shall be optimised for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers
3. All or selected application functions must render well on mobile devices
4. Data shall be stored in the database on the team's deployment cloud server
5. Full resolution free media shall be downloadable directly, and full resolution media for selling shall be obtained after contacting the seller/owner
6. No more than 50 concurrent users shall be accessing the application at any time
7. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users.
8. The language used shall be English (no localization needed)
9. Application shall be very easy to use and intuitive
10. Application should follow established architecture patterns
11. Application code and its repository shall be easy to inspect and maintain
12. Google analytics shall be used (optional for Fulda teams)
13. No email clients shall be allowed.

14. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.
15. Site security: basic best practices shall be applied (as covered in the class) for main data items
16. Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today
17. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development
18. For code development and management, as well as documentation like formal milestones required in the class, each team shall use their own GitHub to be set-up by class instructors and started by each team during Milestone 0
19. The application UI (WWW and mobile) shall prominently display the following exact text on all pages ”Fulda University of Applied Sciences Software Engineering Project, Fall 2022 For Demonstration Only” at the top of the WWW page. (Important to not confuse this with a real application).

9. Competitive analysis

Competitors / Feature Comparison	ReservEat	DinnerBooking	Chope	Table Agent	Dish Cult
Extra Services Selection	Yes	No	No	Yes	No
Chat with Restaurant	Yes	Yes	No	No	No
Flag User	Yes	No	No	No	No
Restaurant Menu	Yes	No	No	No	Yes
User Review	Yes	No	Yes	No	Yes

ReservEat reservation system is aimed to meet the needs of today’s mobile generation. This system will allow users to make online table reservations at their favourite restaurants in a hassle free manner. Users can view the relevant restaurant menu, can select extra services, edit their reservation details and even chat with the restaurant host for detailed information. They can also share their experience with a restaurant by giving review and feedback about quality, service,

pricing and environment. On the other hand, the restaurant host has authority to flag users according to their reservation history and behaviour.

10. High-level system architecture and technologies used

• **Hosting:** Microsoft Azure
• **Back-end Framework:** Express JS
• **Back-end Language:** JavaScript (Node JS)
• **Front-end Framework:** React JS
• **Front-end Language:** JavaScript
• **Code Editor:** Visual Studio Code
• **Target Web Browser:** Google Chrome, Firefox, Safari
• **Database:** MySQL
• **SSL Certificate:** Certbot

11. Team and roles

Name	Role	Email
Utsav Shrestha	Team Lead, Frontend Lead	utsav.shrestha@informatik.hs-fulda.de
Muhammad Bilal	Frontend Developer	muhammad.bilal@informatik.hs-fulda.de
Sanjay George	Backend Lead, DevOps	sanjay.varkey-george@informatik.hs-fulda.de
Omer Zogubi	Backend Developer, GitHub Admin	omer.zogubi@informatik.hs-fulda.de

Parthiv Jani	Frontend Developer, Document Master	parthiv-yogesh-kumar.jani@informatik.hs-fulda.de
Kristijan Lazeski	Backend Developer, Database admin	kristijan.lazeski@informatik.hs-fulda.de

12. Checklist

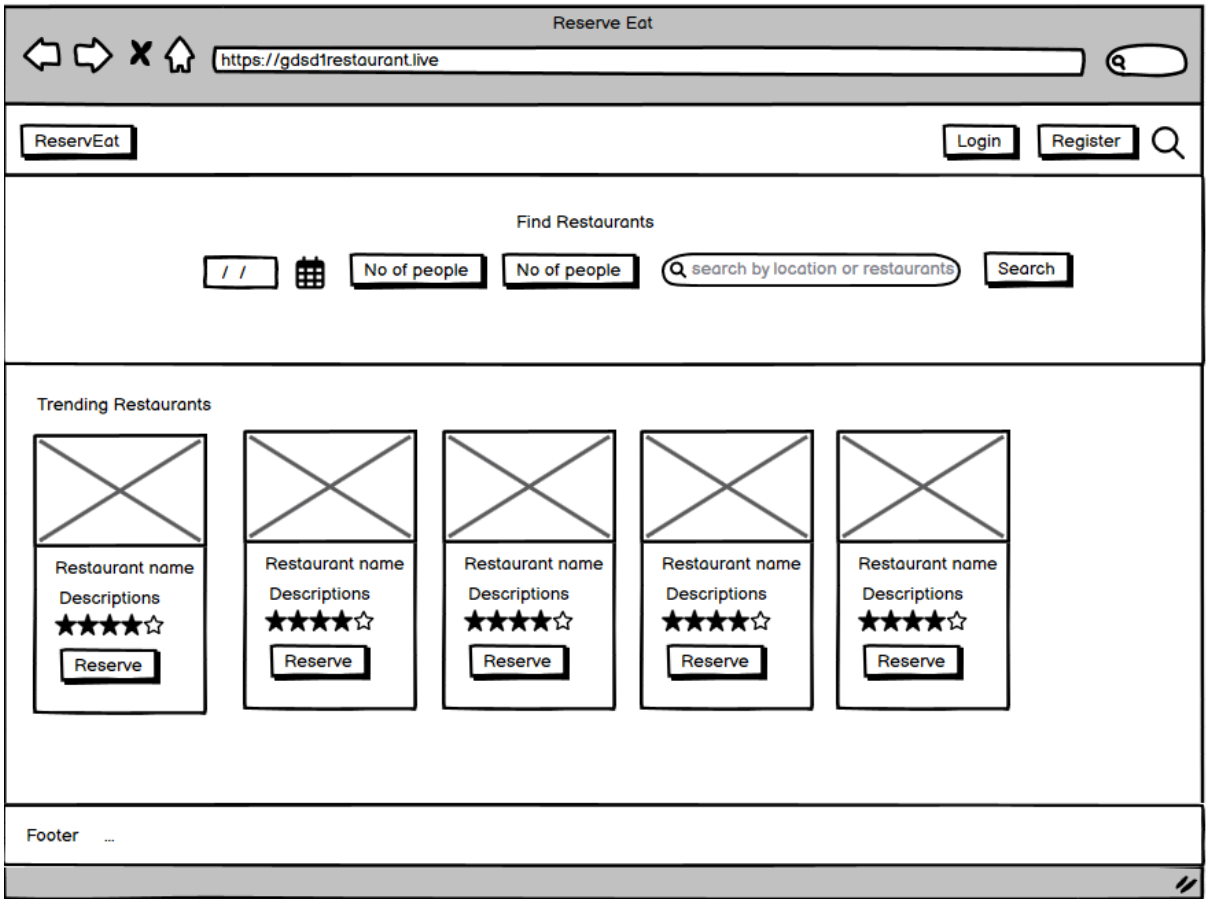
No.	Task	Status
1	Team found a time slot to meet (online) outside of the class	Done
2	GitHub master chosen	Done
3	Team decided and agreed together on using the listed SW tools and deployment server	Done
4	Team ready and able to use the chosen back and front end frameworks and those who need to learn are working on learning and practising	Done
5	Team lead ensured that all team members read the final M1 and agree/understand it before submission	Done
6	GitHub organised as discussed in class (e.g. master branch, development branch, folder for milestone documents etc)	Done

13. UI and functionality feedback (P1 functions only)

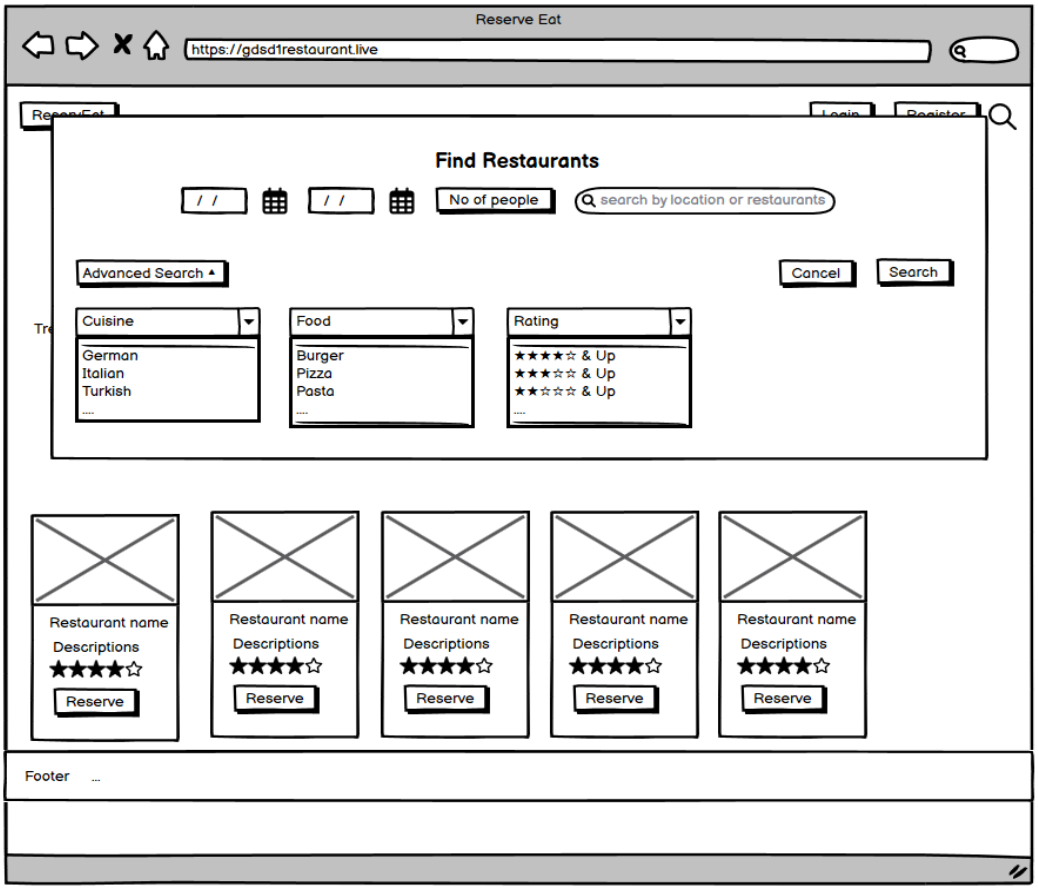
UI-Mockups

As discussed in the meeting and gone over the UseCases upon the current working version of our Website we came to the conclusion that from the UI department we will stick to the current mockups without any changes to them. Thereforth here following the mockups:

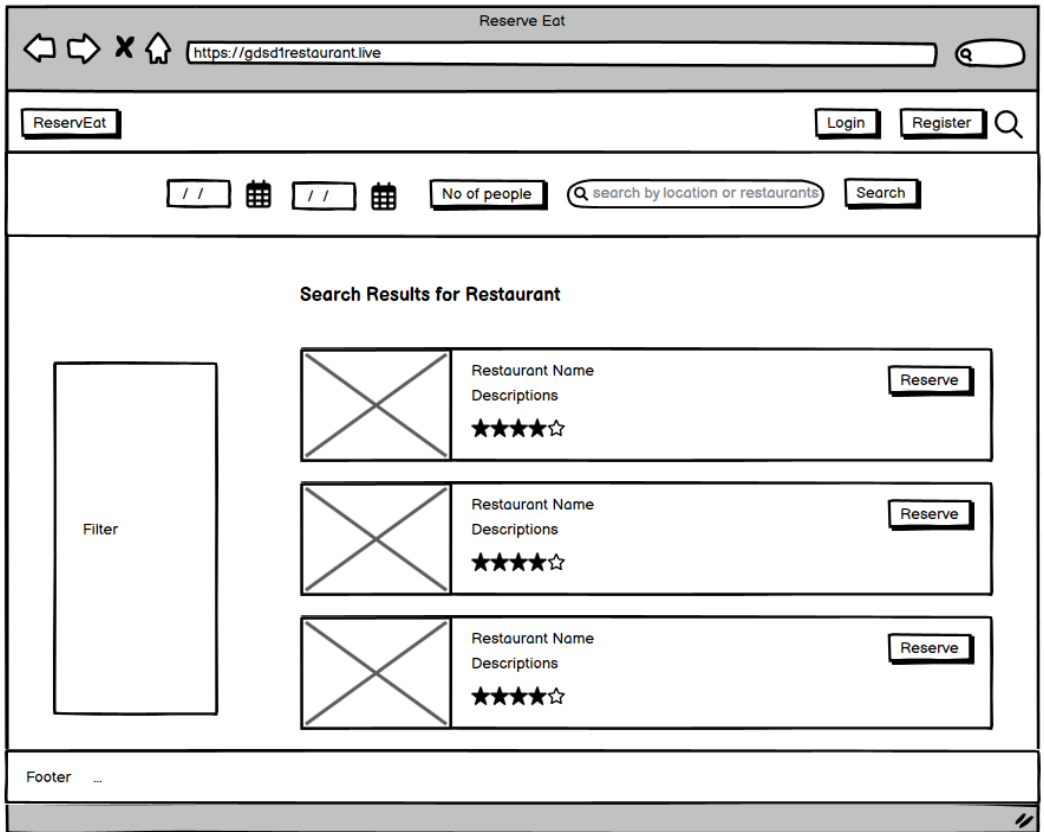
1- Home Page



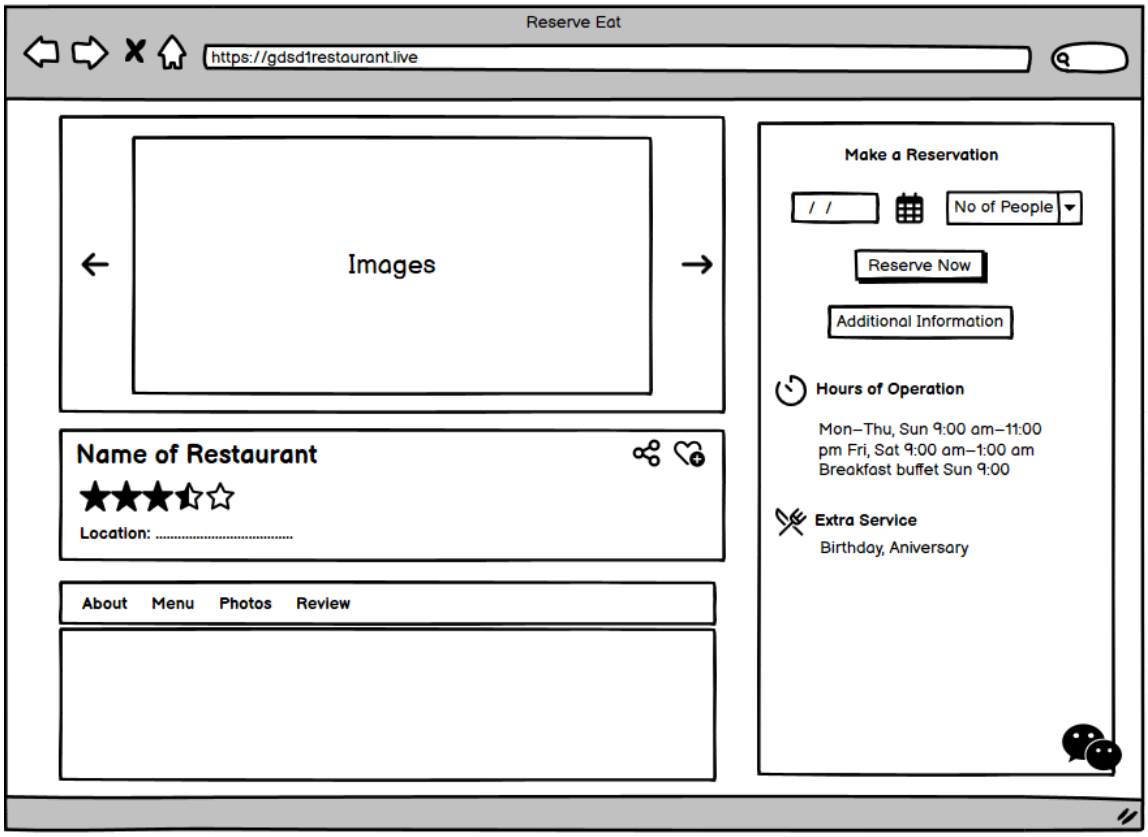
2-Search Restaurant Page



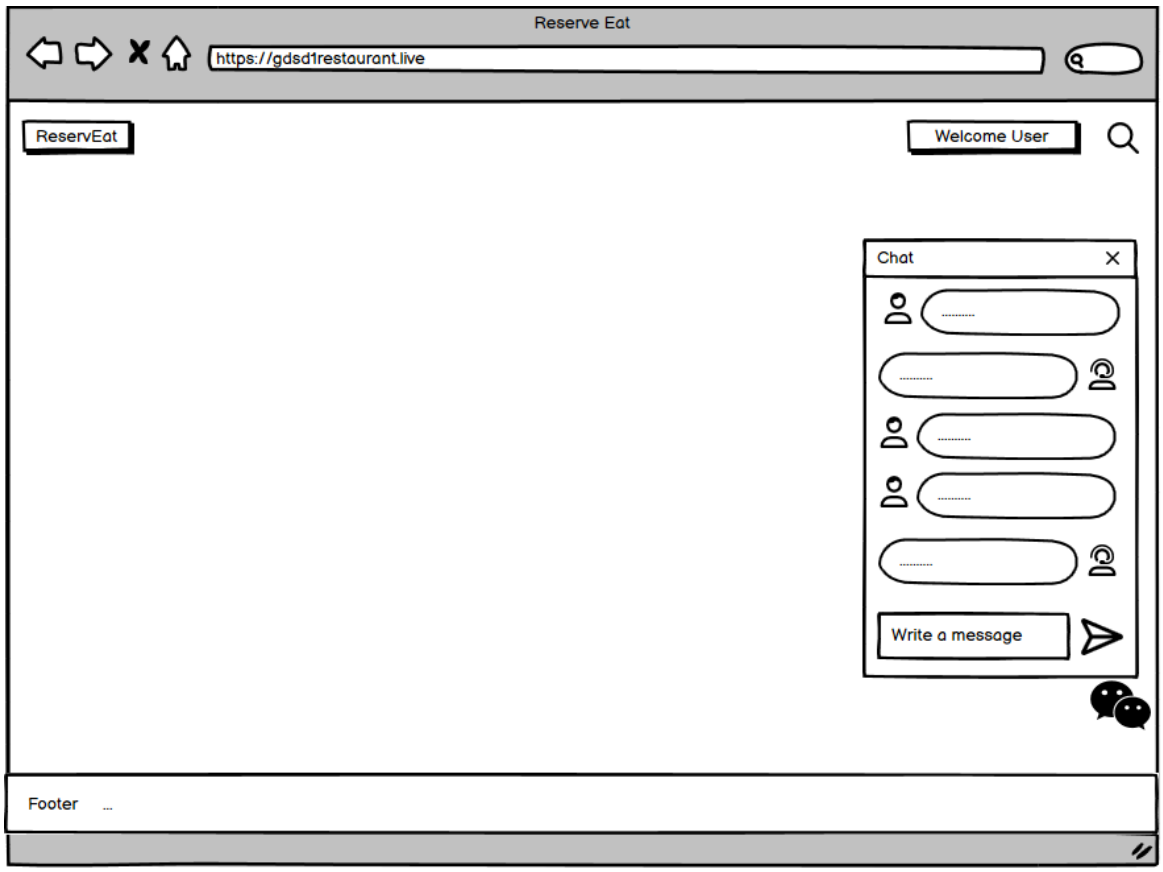
3-Restaurants List Page



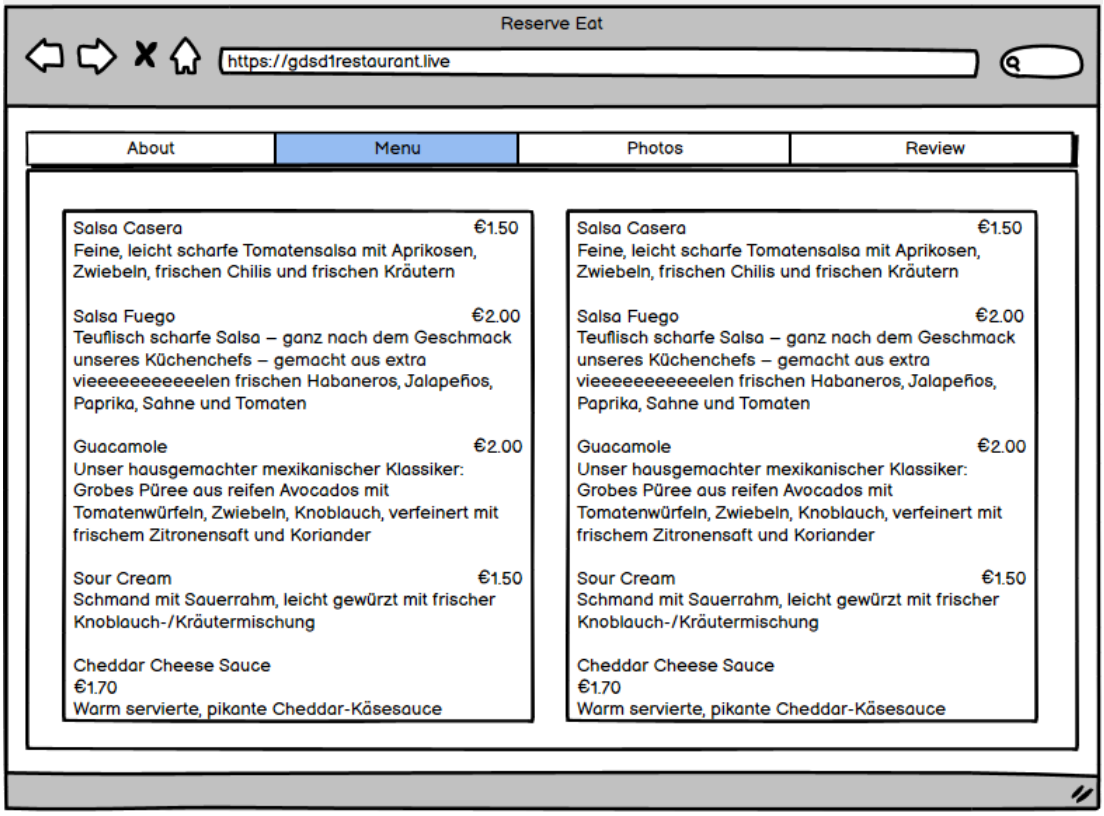
4-Restaurant Details Page



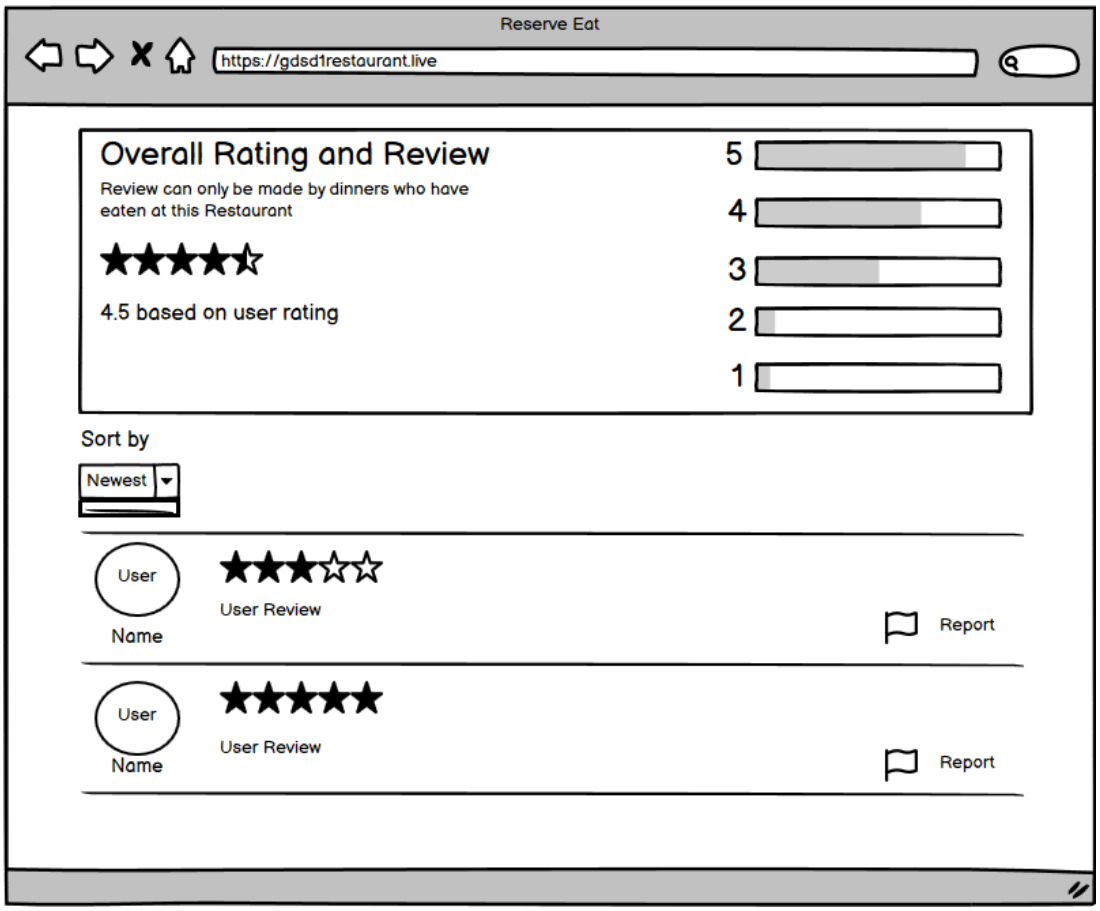
5- Chatting Page



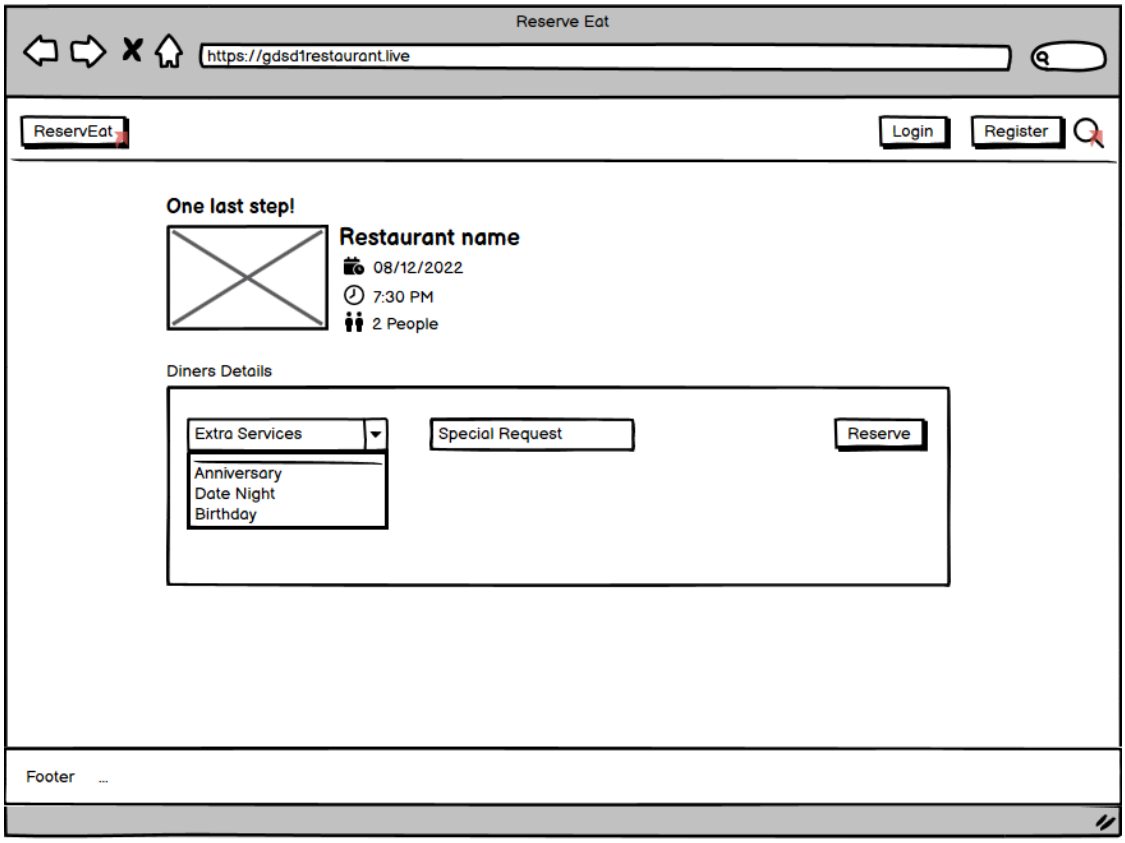
6-Menu Tab


Reserve Eat
https://gdsd1restaurant.live
About
Menu
Photos
Review
Salsa Casera €1.50
Feine, leicht scharfe Tomatensalsa mit Aprikosen, Zwiebeln, frischen Chilis und frischen Kräutern
Salsa Fuego €2.00
Teuflisch scharfe Salsa – ganz nach dem Geschmack unseres Küchenchefs – gemacht aus extra vieeeeeeeeeeelen frischen Habaneros, Jalapeños, Paprika, Sahne und Tomaten
Guacamole €2.00
Unser hausgemachter mexikanischer Klassiker: Grobes Püree aus reifen Avocados mit Tomatenwürfeln, Zwiebeln, Knoblauch, verfeinert mit frischem Zitronensaft und Koriander
Sour Cream €1.50
Schmand mit Sauerrahm, leicht gewürzt mit frischer Knoblauch-/Kräutermischung
Cheddar Cheese Sauce
€1.70
Warm servierte, pikante Cheddar-Käsesauce
Salsa Casera €1.50
Feine, leicht scharfe Tomatensalsa mit Aprikosen, Zwiebeln, frischen Chilis und frischen Kräutern
Salsa Fuego €2.00
Teuflisch scharfe Salsa – ganz nach dem Geschmack unseres Küchenchefs – gemacht aus extra vieeeeeeeeeeelen frischen Habaneros, Jalapeños, Paprika, Sahne und Tomaten
Guacamole €2.00
Unser hausgemachter mexikanischer Klassiker: Grobes Püree aus reifen Avocados mit Tomatenwürfeln, Zwiebeln, Knoblauch, verfeinert mit frischem Zitronensaft und Koriander
Sour Cream €1.50
Schmand mit Sauerrahm, leicht gewürzt mit frischer Knoblauch-/Kräutermischung
Cheddar Cheese Sauce
€1.70
Warm servierte, pikante Cheddar-Käsesauce

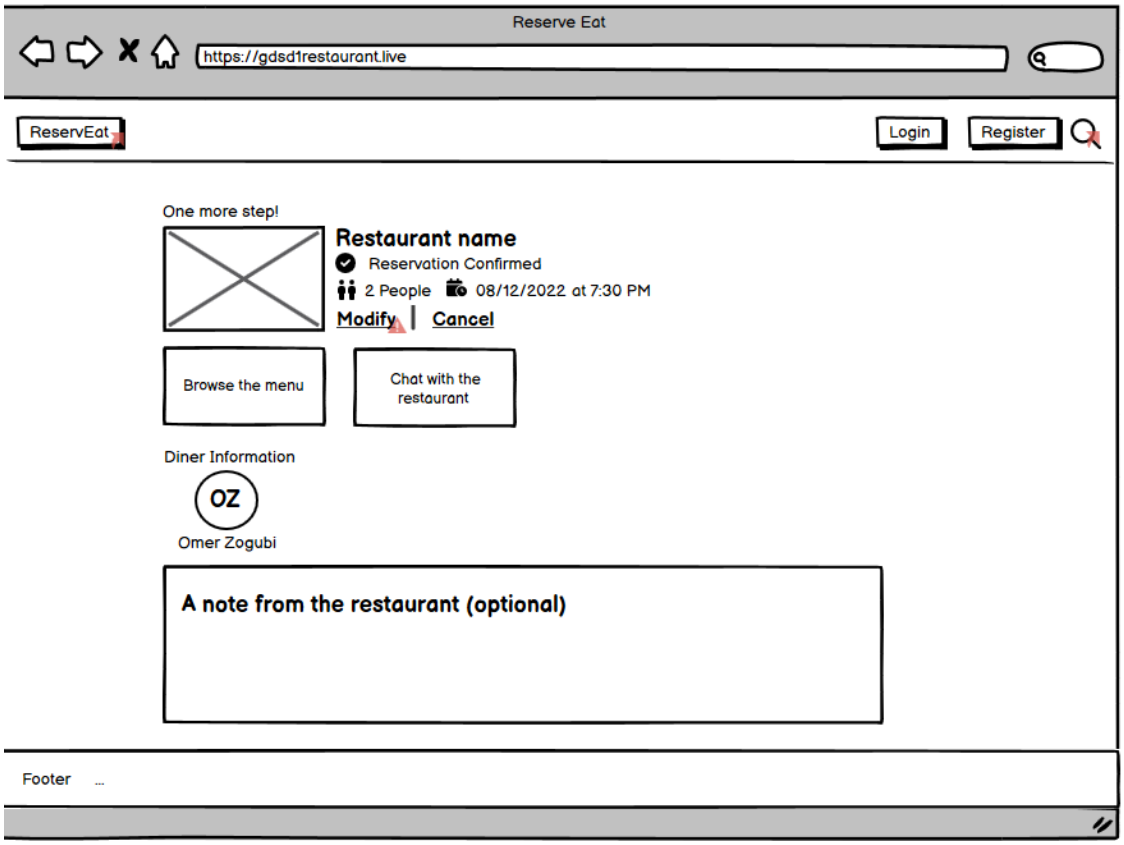
7-Review Tab


Reserve Eat
https://gdsd1restaurant.live
Overall Rating and Review
Review can only be made by dinners who have eaten at this Restaurant
4.5 based on user rating
5
4
3
2
1
Sort by
Newest
User
Name
User Review
Report
User
Name
User Review
Report

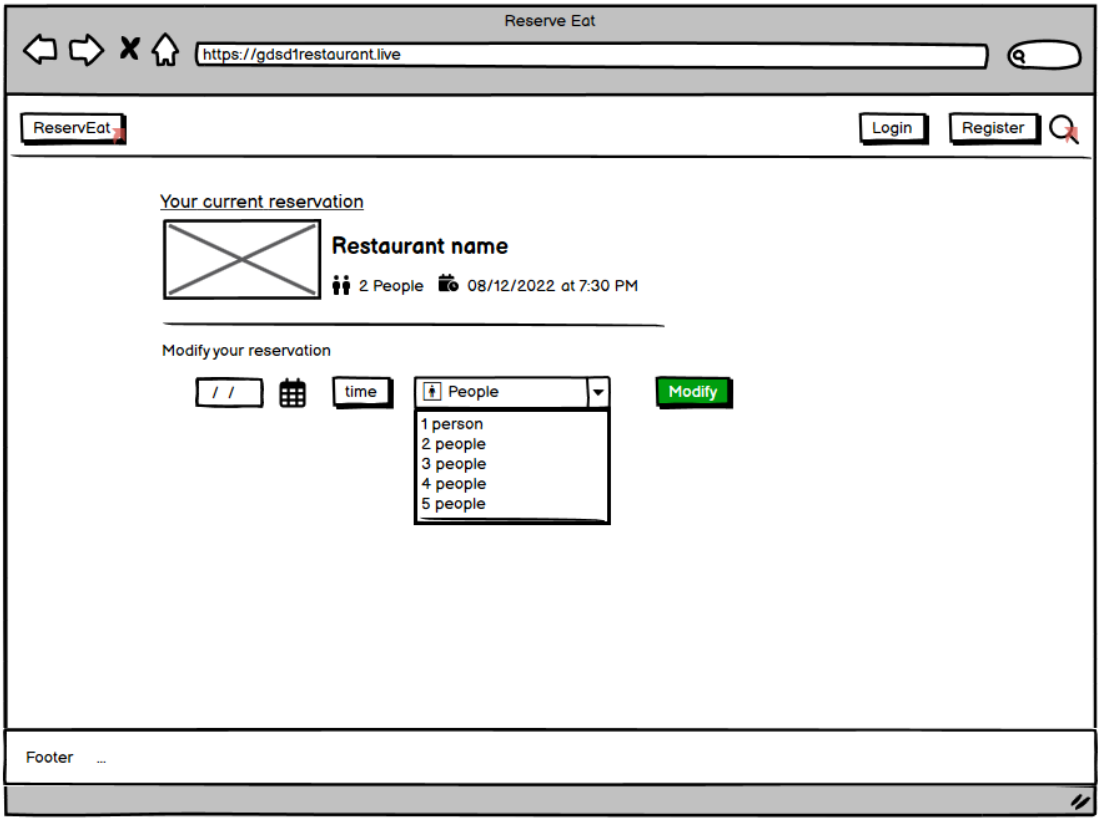
8-Restaurant Reservation Page



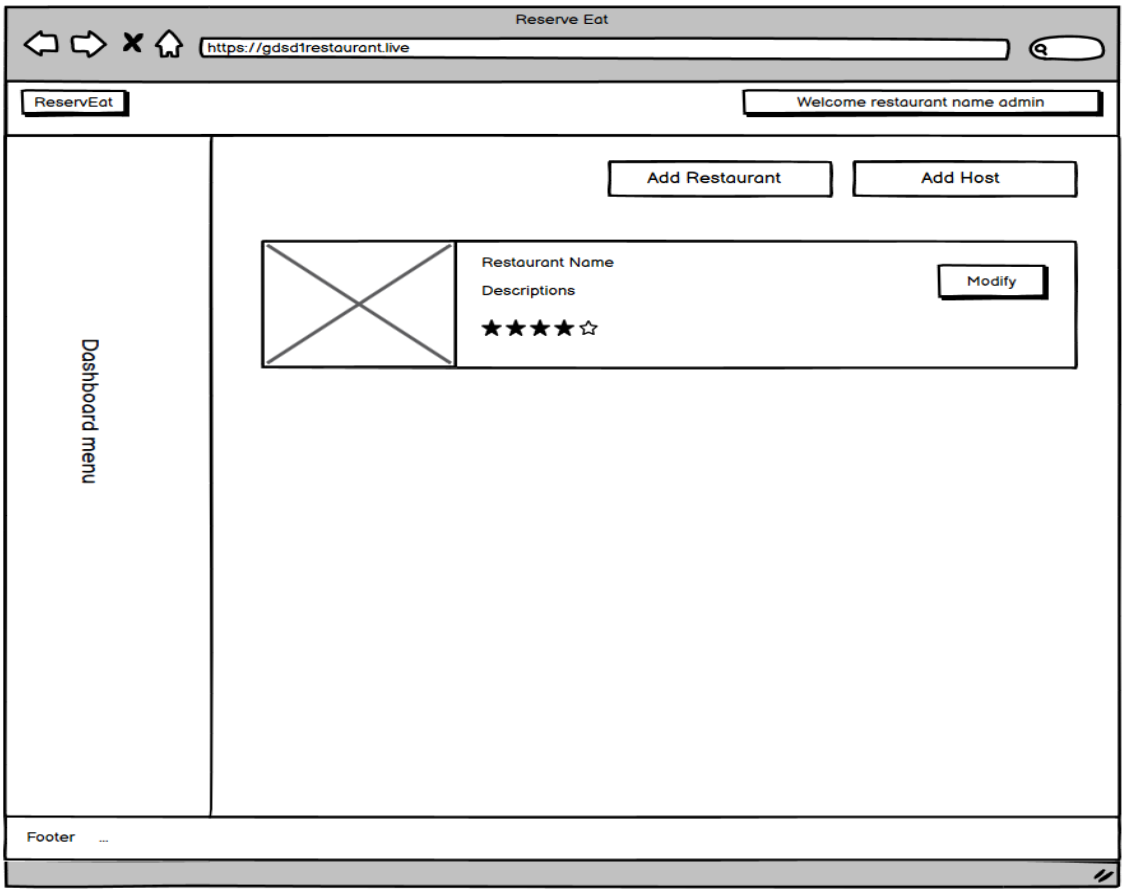
9- Reservation Confirmed/Details Page



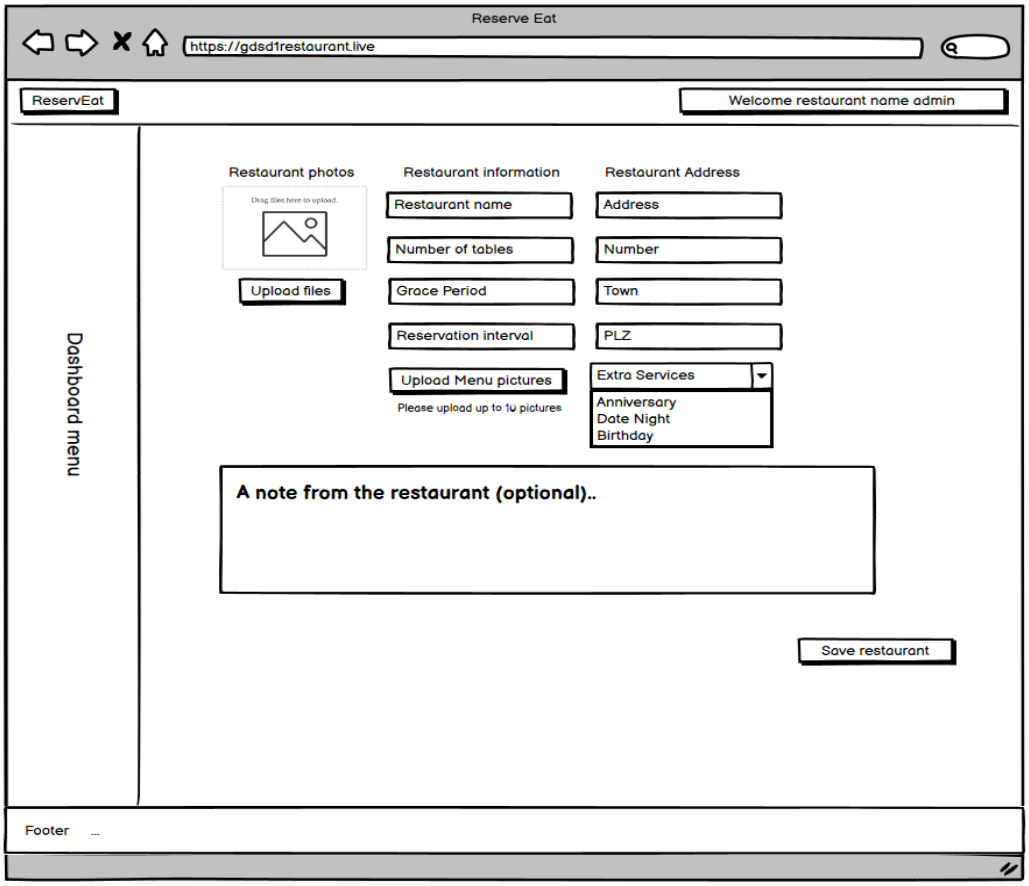
10- Modify Reservation Page



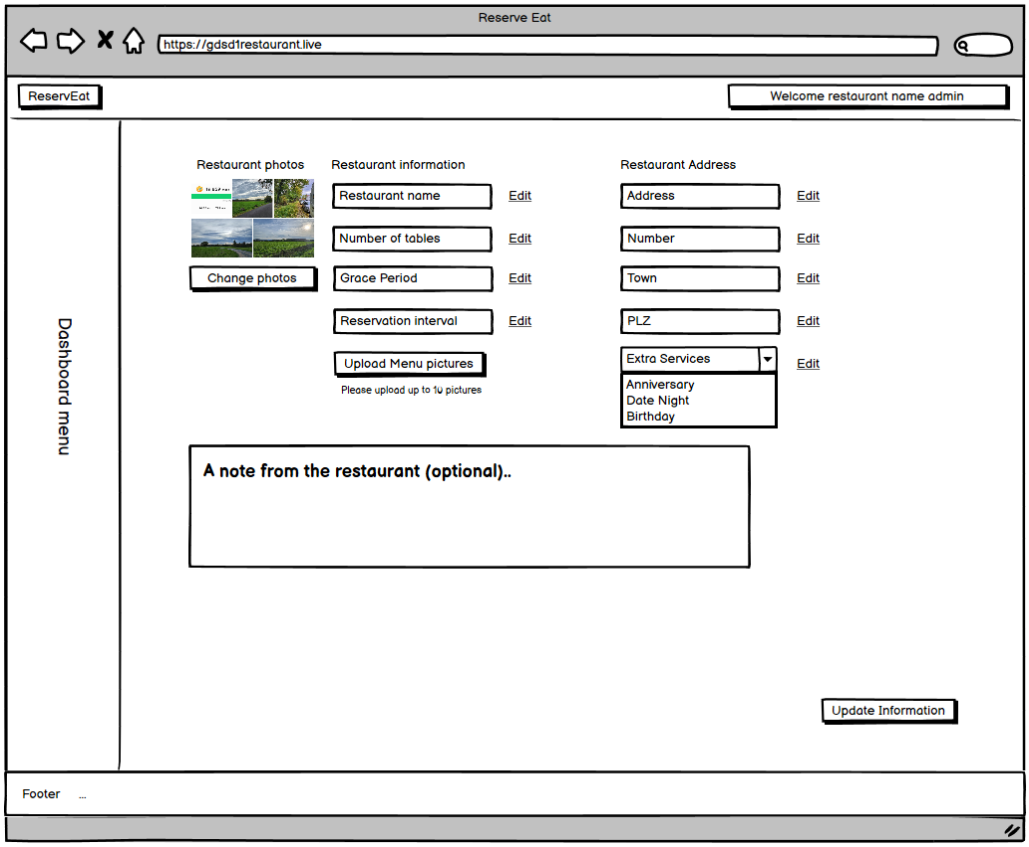
11- Restaurant Host Dashboard Page



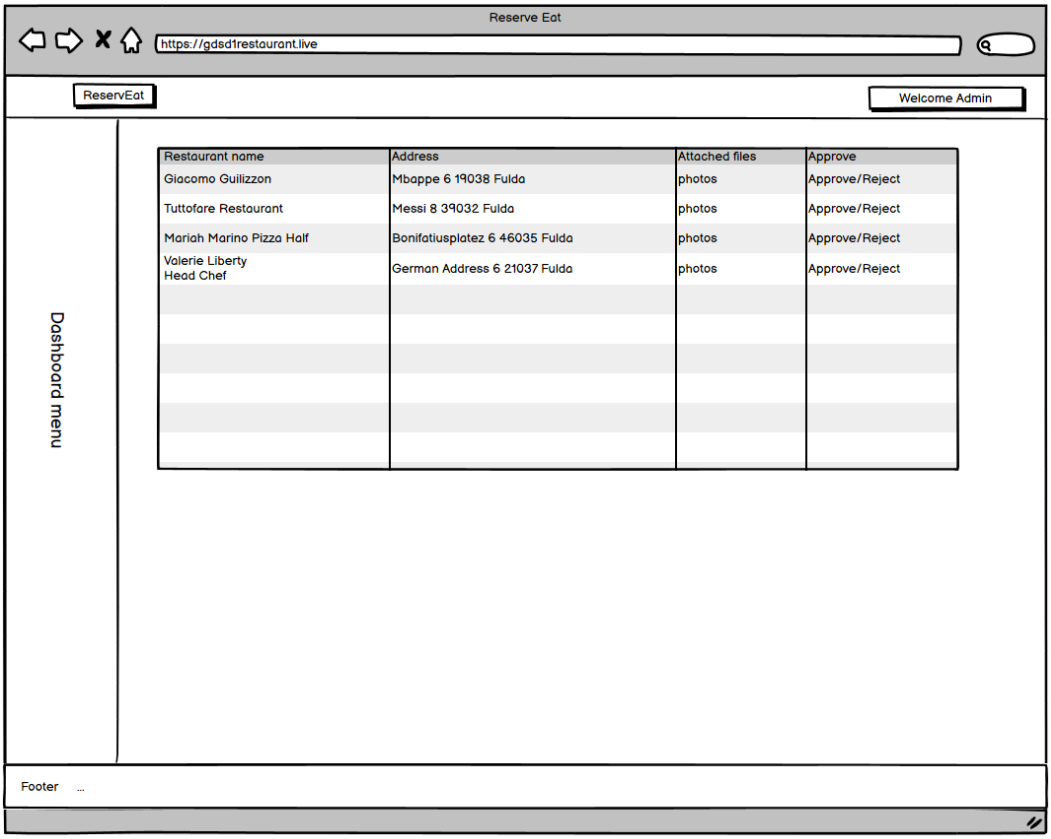
12- Restaurant host add new restaurant page



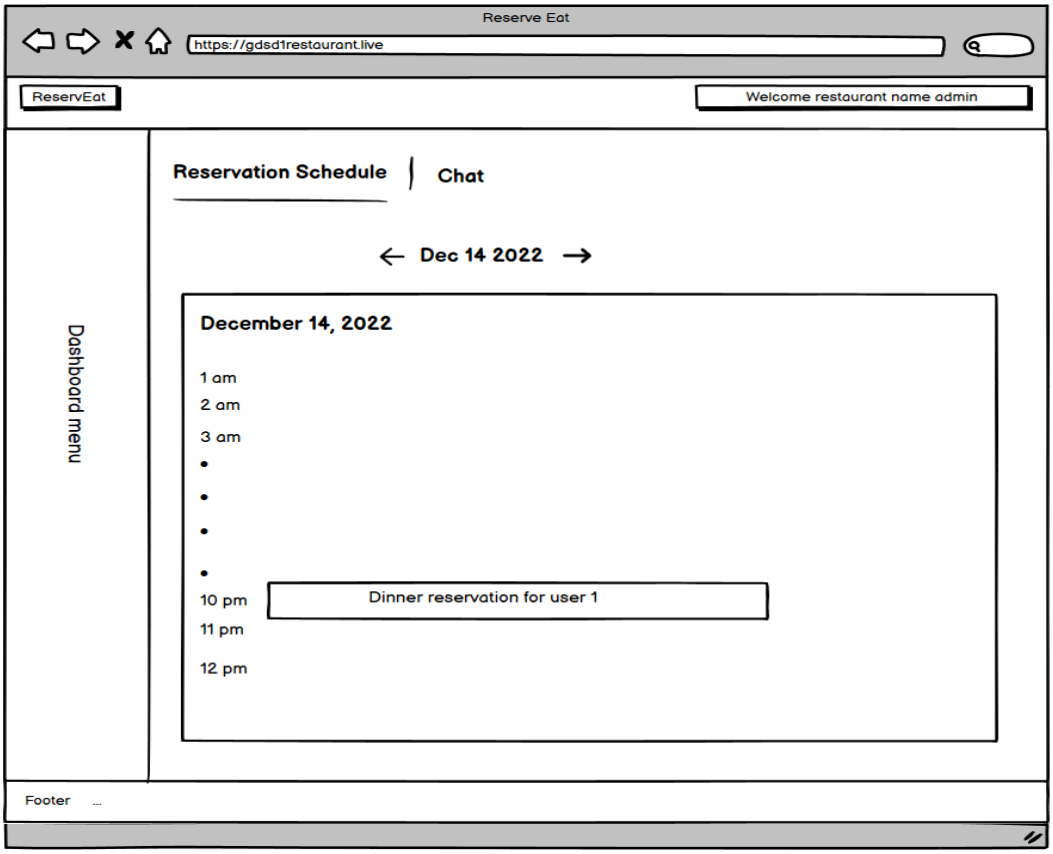
12- Restaurant host modify existing restaurant page



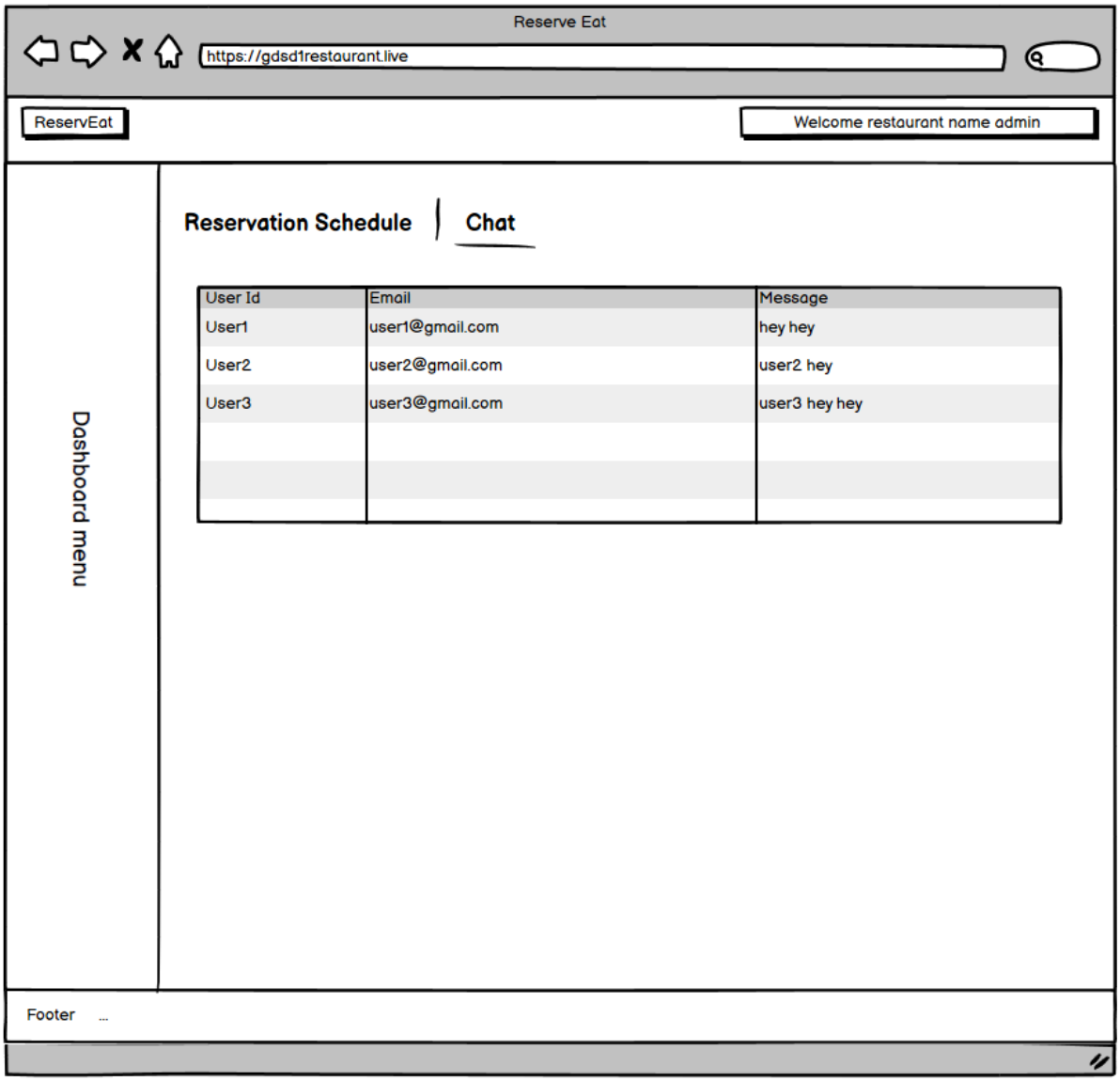
13- Site Administrator Restaurants approval page



14-Restaurant Host Page



15- Restaurant host messages/chat page



Instructor will check functionality and pages/use cases as below:

- Home page
  → Working as intended
- Search (including search field validation)
  → Working as intended
- Search results
  → Working as intended
- Filtering
  → To be Implemented to the Frontend
- Search Details and maps (if applicable)
  → Working as intended
- Messaging/contact agent/user (if applicable)
  → Work in progress
- Data Upload
  → Working as intended
- dashboards (user, admin)
  → Working as intended
- UI responsiveness (resize the browser)
  → Work in Progress
- Performance (e.g. display of results list)
  → Working as intended

14. Brief review of coding, GitHub, database etc.

Key DB Tables

As before in Milestone 2 given Tables, here is a list of all the Tables updated. The Tables are from this point onwards set in stone and will not get any further Updates or changes.

Table: User	
Column	**Data Type**
Id	int
FirstName	varchar
LastName	varchar
Email	varchar
Role	varchar
Password	varchar

PhoneNumber	varchar

Table: Restaurants	
Column	**Data Type**
Id	int
Name	varchar
PostalCode	tinyint
Address	varchar
City	varchar
MaxCapacity	int
MaxTables	int
RestaurantNote	varchar
GracePeriod	timestamp
TimeInterval	timestamp

Table: Reservation	
Column	**Data Type**
Id	int
UserId	varchar
RestaurantId	int
NumberOfSeats	int
Status	enum(“Reserved”, “Modified”, “Cancelled”)
Date	date
Time	timestamp
ExtraServiceId	int

LastUpdated	timestamp

Table: ExtraServices	
Column	**Data Type**
Id	int
Name	varchar
Description	varchar

Table: Chats	
Column	**Data Type**
Id	int
User1	int
User2	int

Table: Messages	
Column	**Data Type**
ChatId	int
Message	varchar
Timestamp	datetime

Table: Flags	
Column	**Data Type**
Id	int
UserId	int

ReservationId	int
Text	varchar

Table: Rewards	
Column	**Data Type**
Id	int
UserId	int
Type	enum(“ForReservation”, “ForUserReview”)
Points	int

Table: Reviews	
Column	**Data Type**
Id	int
UserId	int
RestaurantId	int
Rating	int
Text	varchar

Table: Images	
Column	**Data Type**
Id	int
RestaurantId	int
Status	enum(“Uploaded”, “Approved”, “Rejected”)
Path	varchar

IsMenuImage	boolean

Search

2	User can Search for Restaurants by name, cuisine, food, and rating	1
2.1	Search results can be filtered additionally and sorted	1

As per the Priority 1 Functional Requirement 2 and 2.1 we hereby point out that the Search functionality of the website is completely working and functional as it is supposed to be.

15. Project status

a) Teamwork
   i) So far the Team has been working efficiently and productively. Any Arguments or misunderstandings were talked through and a solution to them was directly found. Hence the Team is working perfectly onwards.
b) Risks
   i) Health: As it goes for everyone, we are all Human Beings and it might be that some of us are Sick during the Project. Therefore will the others do more and compensate with the other missing members when they are back and ready to continue on with work
   ii) Vacation: As some of us have already planned some Vacation leave, we have already discussed this and so therefore also already compensated
   iii) Schedule and Technical skills: As far as the Project has gone onwards we are sure that our Team is capable enough to finish the rest of the Priority 1 Functions in to the due date
c) Coding Practices
   i) The project uses the Google Java Coding Style
d) Usage of proper SE code management practices:
   i) Each developer creates his own Branch and then does his work on it. After finishing up the work he then proceeds to push it to the Git Repository and creates a Pull Request. This Pull Request has to be reviewed by another Developer before it is allowed to be pushed upon the Master.
e) How did you address the Site Security and safe coding practices
   i) The use of standardized Security mechanisms was used in the Project
f) Digital Content
   i) Royalty Free Images were used. The other Text Data was generated by the Team members

16. Summary Feedback and Tasks to do

Upon the Presentation the following Priority 1 Requirements are done and we are aiming to finish implementing the rest of the Priority 1 Requirements up until the due date (Bright Green = Done):

ID	Functional Requirement	Priority
1	The Application is a Vendor for Multiple Restaurants	1
2	User can Search for Restaurants by name, cuisine, food, and rating	1
2.1	Search results can be filtered additionally and sorted	1
3	User can reserve a table and specify the number of seats in a given Restaurant	1
3.1	User reserves one or Multiple Tables at the Restaurant	1
3.2	User specifies the number of seats	1
4	Restaurants can be added, removed and updated by Restaurant owners	1
4.1	Restaurants can be Added by the Restaurant Owners	1
4.2	Restaurants can be Removed by the Owners of given Restaurant	1
4.3	Restaurants can be Updated by the owners of said Restaurant	1
5	There is a Grace Period to Cancel or update the booking of the table	1
5.1	Every User can Cancel the reservation up to 2h before said reservation	1
6	The number of booked seats are specified per table	1
7	Users can view Menus of the Restaurants	1
8	Users search for restaurants, make/change/cancel reservations and also post reviews.	1
8.1	User can cancel Reservation	1
8.2	User can change Reservation	1
8.3	User can Post Reviews	1
9	User can View the Restaurant Reviews	1
10	Every booking is associated with an account	1
11	One Account can not be associated with multiple users	1

12	User can chat with the Restaurant Host	1
13	User accumulates Reward Points for every reservations with the application	1

ID	Functional Requirement	Priority
1	Restaurants can add Extra Services with themes through Application	2
1.1	Restaurants can add Themes as a Service	2
1.2	Restaurants can add Special Foods or Beverages as a Service to be clicked/booked beforehand	2
2	Restaurants can change the availability status of any table at any time.	2
2.1	Restaurant owner updates the number of free seats for given table	2
3	Restaurant Owners can deny or approve of a booking	2
3.1	Restaurants owner can disapprove a booking made for given number of Seats at a given Time	2
3.2	Restaurants owner can approve a booking made for given number of Seats at a given Time	2
4	User can choose Event Themes	2
5	User can only book two tables daily (to avoid spam reservations)	2
6	Site administrator approves restaurant info for posting. Also deals with typical admin duties like managing user registrations etc.	2
6.1	Site Admin approves Restaurant info for Posting	2
6.2	Site Admin Identifies and addresses usability Problems	2
6.3	Site Admin manages user Registration	2
6.4	Site Admin tracks and analyses the Traffic that is being made on our Website	2
6.5	Site Admin Develops Training Manual for use for Restaurants	2
6.6	Site Admin Flags user if there is a valid reasoning	2
7	User can Check his Reservations	2

ID	Functional Requirement	Priority
1	Users can Provide Reviews for the Restaurant which they have visited through our Application	3
1.1	User provides Text of the Review	3
1.2	User gives a Number Rating between 1 and 5	3
2	Restaurant owners can Flag users for their lack of attendance on booking	3
2.1	Restaurant owners submit a report for time and place and reasoning of why said user should be flagged	3
3	Admin can Ban Restaurants	3
4	Host/hostess reviews daily calendar and checks and greets incoming guests This List might be Edited as the Project goes on.	3

17. Product Summary

Introducing ReservEat, the ultimate restaurant booking website that will change the way you dine out! ReservEat is a vendor for multiple restaurants, giving you access to a wide range of dining options. With ReservEat, you can easily search for restaurants by name, cuisine, food, and rating. Our search results can be filtered and sorted, making it easy to find the perfect restaurant for your taste.

ReservEat allows you to reserve a table and specify the number of seats at any given restaurant. You can reserve one or multiple tables, and the number of booked seats is specified per table. Additionally, every booking is associated with an account, so you can keep track of all your reservations in one place.

Our platform allows restaurant owners to add, remove, and update their restaurant information. You can view menus of the restaurants and post reviews of your dining experience. You can also chat with the restaurant host, making it easy to communicate any special requests or preferences.

With ReservEat, you can cancel or change your reservation up to 2 hours before the reservation time. This grace period ensures that you have the flexibility you need in case your plans change. Additionally, for every reservation you make with ReservEat, you accumulate reward points that can be redeemed for discounts at participating restaurants.

We understand the importance of safety and security when it comes to your personal information. That's why one account can only be associated with one user, ensuring that your data is kept safe and secure.

Experience the convenience of ReservEat and discover new dining experiences today. Start searching for your next dining destination with ReservEat!

List of Functional Requirements:

ID	Functional Requirement	Status (Pass/Fail)
1	The Application is a Vendor for Multiple Restaurants	
2	User can Search for Restaurants by name, cuisine, food, and rating	
2.1	Search results can be filtered additionally and sorted	
3	User can reserve a table and specify the number of seats in a given Restaurant	
3.1	User reserves one or Multiple Tables at the Restaurant	
3.2	User specifies the number of seats	
4	Restaurants can be added, removed and updated by Restaurant owners	
4.1	Restaurants can be Added by the Restaurant Owners	
4.2	Restaurants can be Removed by the Owners of given Restaurant	
4.3	Restaurants can be Updated by the owners of said Restaurant	
5	There is a Grace Period to Cancel or update the booking of the table	
5.1	Every User can Cancel the reservation up to 2h before said reservation	
6	The number of booked seats are specified per table	
7	Users can view Menus of the Restaurants	
8	Users search for restaurants, make/change/cancel reservations and also post reviews.	
8.1	User can cancel Reservation	
8.2	User can change Reservation	
8.3	User can Post Reviews	
9	User can View the Restaurant Reviews	

10	Every booking is associated with an account	
11	One Account can not be associated with multiple users	
12	User can chat with the Restaurant Host	
13	User accumulates Reward Points for every reservations with the application	

18. Usability Test Plan

Objective: The objective of this usability test is to evaluate the ease of use, functionality, and user experience of the ReservEat website.

Participants: The test participants will consist of 6 individuals who frequently dine out and are comfortable using technology. Participants will be chosen based on their accessibility and willingness to take part on the test with restaurant booking websites.

Tasks:

1. Search for a restaurant by name, cuisine, food, and rating.
2. Filter and sort the search results to find a suitable restaurant.
3. Reserve a table and specify the number of seats at a given restaurant.
4. Cancel or change a reservation up to 2 hours before the reservation time.
5. Post a review of a restaurant and view other restaurant reviews.
6. Chat with a restaurant host to communicate any special requests or preferences.
7. Accumulate reward points for every reservation with the application.

Metrics:

1. Success rate: The percentage of participants who are able to complete each task successfully.
2. Time on task: The time taken by participants to complete each task.
3. Error rate: The number of errors made by participants while completing each task.
4. Satisfaction: The overall satisfaction of participants with the ReservEat website.

Procedure:

1. Brief the participants about the purpose of the test and provide them with a consent form.
2. Ask the participants to complete each task while thinking aloud, and record their responses.
3. Collect metrics such as success rate, time on task, error rate, and satisfaction score for each task.
4. After the completion of all tasks, ask participants to provide feedback on the website's usability, functionality, and user experience.
5. Debrief participants and thank them for their time.

Analysis: Analyze the data collected from the test and identify any areas of improvement for the website. Use the feedback provided by participants to make necessary changes and modifications to enhance the website's usability, functionality, and user experience.

19. QA test plan

Test Case ID	Test Scenario	Test Steps	Expected Result	Actual Result	Status (Pass/Fail)
1	Search for a restaurant by name	1. Enter restaurant name in search bar	Search results should display the restaurant		
		2. Click on search button			
		3. Verify that the correct restaurant is displayed in results			
2	Filter search results by cuisine type	1. Select a cuisine type from the filter dropdown	Search results should display restaurants of that cuisine		

		2. Verify that only the selected cuisine type is displayed			
3	Sort search results by rating	1. Select "rating" from the sort dropdown	Search results should be sorted by rating		
		2. Verify that the highest rated restaurants are displayed			
4	Reserve a table at a restaurant	1. Select a restaurant from search results	The reservation page should be displayed		
		2. Enter the number of seats needed	The reservation should be confirmed		
		3. Click "reserve table" button			
		4. Verify that reservation is confirmed			
5	Cancel or change a reservation	1. Click on "My Reservations" tab	The user's reservation history should be displayed		
		2. Select the reservation to be cancelled or changed	The reservation details should be displayed		
		3. Click on "cancel" or "change" button	The reservation should be cancelled or changed		

6	Post a review of a restaurant and view other restaurant reviews	1. Select a restaurant from search results	The restaurant page should be displayed		
		2. Click on "write a review" button	The review form should be displayed		
		3. Enter the review details and click on "submit" button	The review should be posted and displayed on the page		
		4. Verify that the posted review is displayed on the page			
7	Chat with a restaurant host	1. Select a restaurant from search results	The restaurant page should be displayed		
		2. Click on "chat with host" button	The chat window should be displayed		
		3. Enter the chat message and click on "send" button	The chat message should be sent and displayed on the page		
8	Accumulate reward points for every reservation	1. Make a reservation at a restaurant	Reward points should		

20. Code Review

1. Kristijan
    a. Add Hochschule email validation and encrypt password Done

```
43      -           const accessToken = jwt.sign(data[0], "" + process.env.ACCESS_TOKEN_SECRET);
44      -           res.send({ msg: 'Login successful', data: data[0], accessToken: accessToken });
        51  +       const validPass = await bcrypt.compare(password, data[0].password);
```

kikolazeski Pending

this should be done in the login function not here.

Reply...

```
        52  +       if (validPass) {
        53  +           const accessToken = jwt.sign(data[0], "" + process.env.ACCESS_TOKEN_SECRET);
        54  +           delete data[0].password;
```

kikolazeski Pending

why is password getting returned from the login function data

Reply...

```
        55  +           res.send({ msg: 'Login successful', data: data[0], accessToken: accessToken });
        56  +       } else {
        57  +           res.status(500).send({ msg: 'Wrong password' },);
        58  +       }
45      59      } else {
46      60          res.status(500).send({ msg: 'email or password is in incorrect' },);
47      61      }
```

```
31      32  };
32      33  const login = (email, password) => {
33          -   const query = `SELECT us.id, us.firstName, us.lastName, us.phonenumber, us.email, r.name as role FROM users us INNER J
                = '${password}'`;
        34  +   const query = `SELECT us.id, us.firstName, us.lastName, us.phonenumber, us.email, us.password, r.name as role FROM us
                ='${email}'`;
```

kikolazeski Pending

check the password decryption here. and check if password and email matches and then query this

Reply...

```
34      35      return new Promise((resolve, reject) => {
35      36          pool.query(query, function (error, results) {
36      37              if (error) reject(error);
```

11 ■■■■■ package-lock.json

2. Utsav Shrestha
    a. Implement restaurant time (Pull request)

omerzogubi requested a review from **Utsav170** last week

Utsav170 reviewed last week

View changes

libs/database/restaurants.js

```
158 +  const query = `
159 +          SELECT *
160 +          FROM restaurantTime
161 +          WHERE restaurantID=${restaurantID}
```

Utsav170 last week

and you need to add `availability=true`

Reply...

Resolve conversation

libs/database/booking.js Outdated

```
21  -
21  +  var tableBookedSeats = `SELECT seatsBooked FROM restaurantTime WHERE restaurantId =${restaur
22  +  var totalSeats = tableBookedSeats + numberOfSeats;
23  +  const updateRestaurantTime = `UPDATE restaurantTime SET seatsBooked = '${totalSeats}' WHERE
```

Utsav170 last week

here first you need to get the number of seats available in a restaurant and then add the condition if the totalNumberOfSeats >= totalSeats only after that you should update here

omerzogubi last week Author

But you should not allow to have that in the FE, there is an endpoint to get seats for each time for the restaurant you can check and allow/disallow user from selecting it, we can have a safety check yes, but I guess user shouldn't progress if there is no place for that hour

Reply...

Resolve conversation

libs/database/booking.js Outdated

```
22  24      return new Promise((resolve, reject) => {
23      -     pool.query(sql, (error, result, fields) => {
        25  +     pool.query(sql, updateRestaurantTime, (error, result, fields) => {
```

Utsav170 last week

it cannot take two querys remove updateRestaurantTime from here

Utsav170 last week

create a new pool.query and execute that updateRestaurantTime

Reply...

Resolve conversation

libs/database/booking.js Outdated

```
42  +
43  +   var reservationNumberOfSeats = `SELECE numberOfSeats FROM reservations WHERE userId = '${use
44  +   var reservationRestaurantID = `SELECE restaurantId FROM reservations WHERE userId = '${userI
45  +   var reservationTime = `SELECE time FROM reservations WHERE userId = '${userId}' AND id = ${r
```

Utsav170 last week

first thing select spelling is wrong. second you can get all of the above in the same query
`SELECT time, numberOfSeats, restaurantId  FROM reservations WHERE userId = '${userId}' AND id = ${reservationId}'`

Reply...

Resolve conversation

libs/database/booking.js Outdated

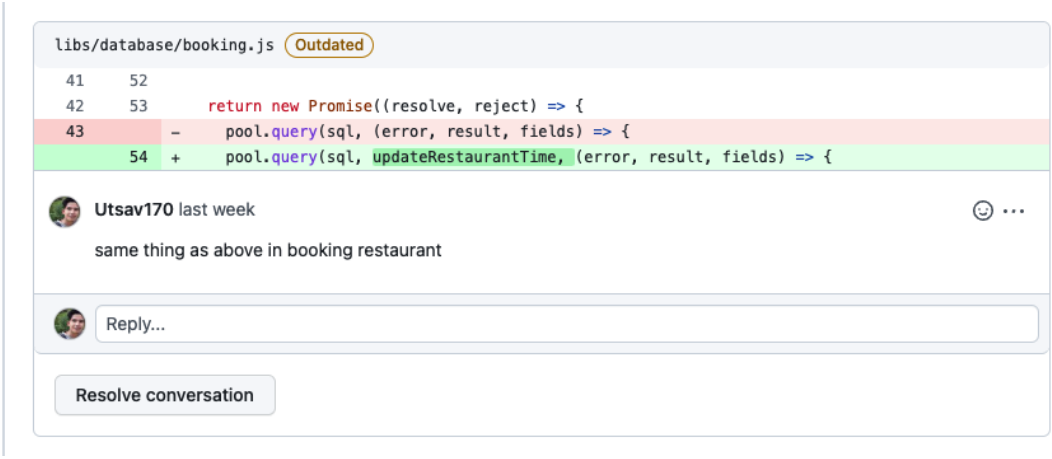
```
44  +   var reservationRestaurantID = `SELECE restaurantId FROM reservations WHERE userId = '${userI
45  +   var reservationTime = `SELECE time FROM reservations WHERE userId = '${userId}' AND id = ${r
46  +   var tableBookedSeats = `SELECT seatsBooked FROM restaurantTime WHERE restaurantId =${reserva
47  +   var totalSeats = tableBookedSeats - reservationNumberOfSeats;
```

Utsav170 last week

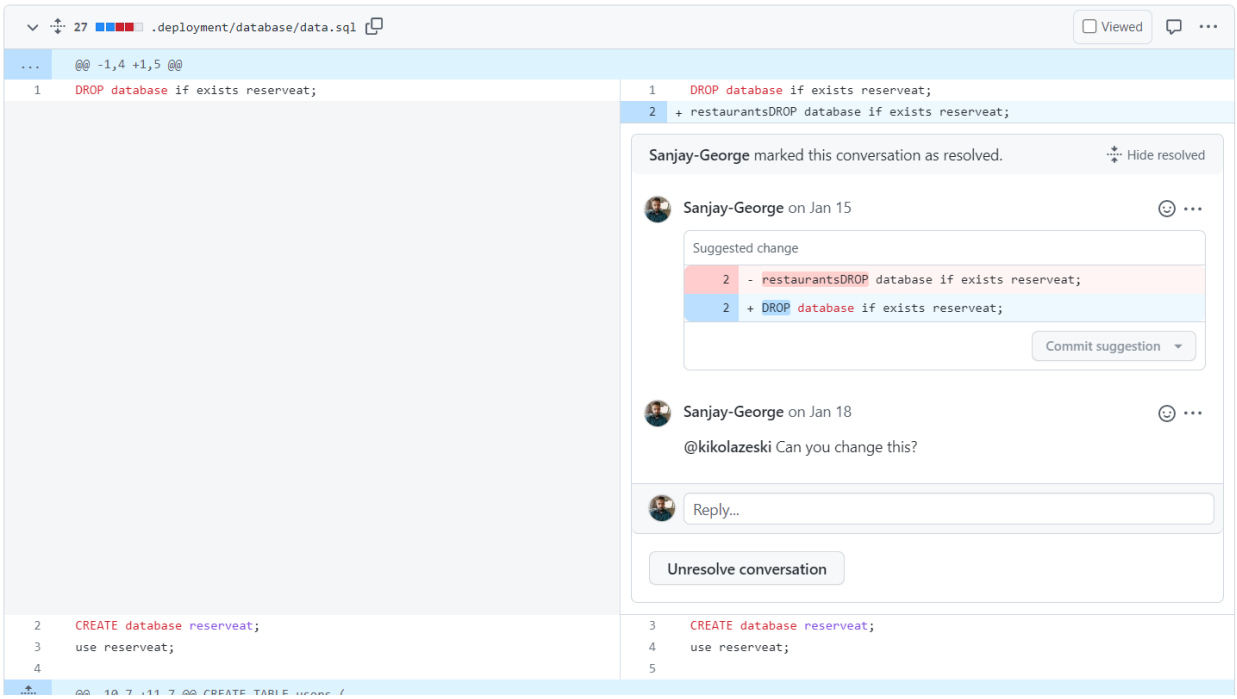
here `reservationNumberOfSeats` this query is not yet executed yet. execute the query first and then from the results you can get it.

Reply...

Resolve conversation



3. Sanjay George
    a. Review on Pull Request containing backend changes to add new restaurants



libs/controllers/store.js Outdated — Hide resolved

```
7  +    const { id, name, plz, street, town, cuisine, seatAmount, seatsFree, text, timeSlot, time } = req.b
8  +    const sql = `INSERT INTO restaurants (name, location, cuisine) VALUES ('${id}', '${name}', '${plz}'
9  +
10 +    connection.query(sql, (error, res) => {
11 +        if (error) throw error;
12 +        res.send({ message: 'Restaurant added successfully.' });
13 +    });
```

Sanjay-George on Jan 18

move this into restaurants DB. Files in controller should only handle requests and return responses.

Reply...

libs/controllers/store.js Outdated — Hide resolved

```
13 -         console.error(ex);
14 -                 res.sendStatus(500);
15 -     }
6  + router.post('/add-restaurant', (req, res) => {
```

Sanjay-George on Jan 18

Keep the path : `POST /api/restaurants/` which means that we are POSTing to restuarants entity (basically creating a new restaurant).

Sanjay-George on Jan 18

Also, I think we already have a controller and DB file for restaurants, we can use that itself. All CRUD operations on restaurants can be in thoese files

dbms.txt Outdated — Hide resolved

```
10 -    Role varchar(255),
11 -    Password varchar(255),
12 -    Phonenumber varchar(255)
1  + restaurantsDROP database if exists reserveat;
```

Sanjay-George on Jan 15

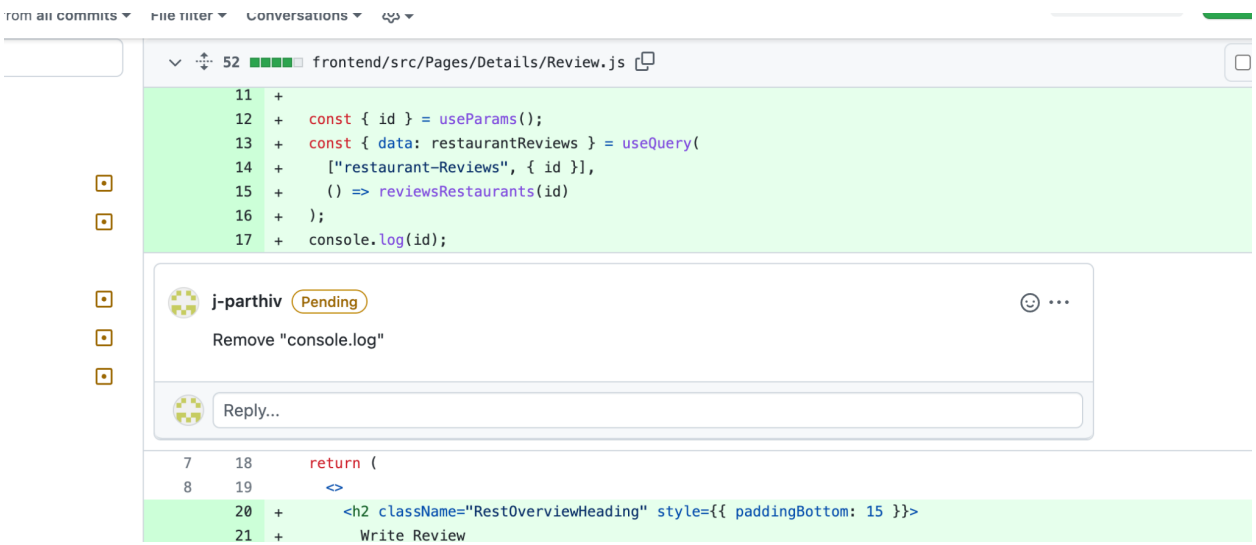
We can get rid of this file, then you don't have to update both files.

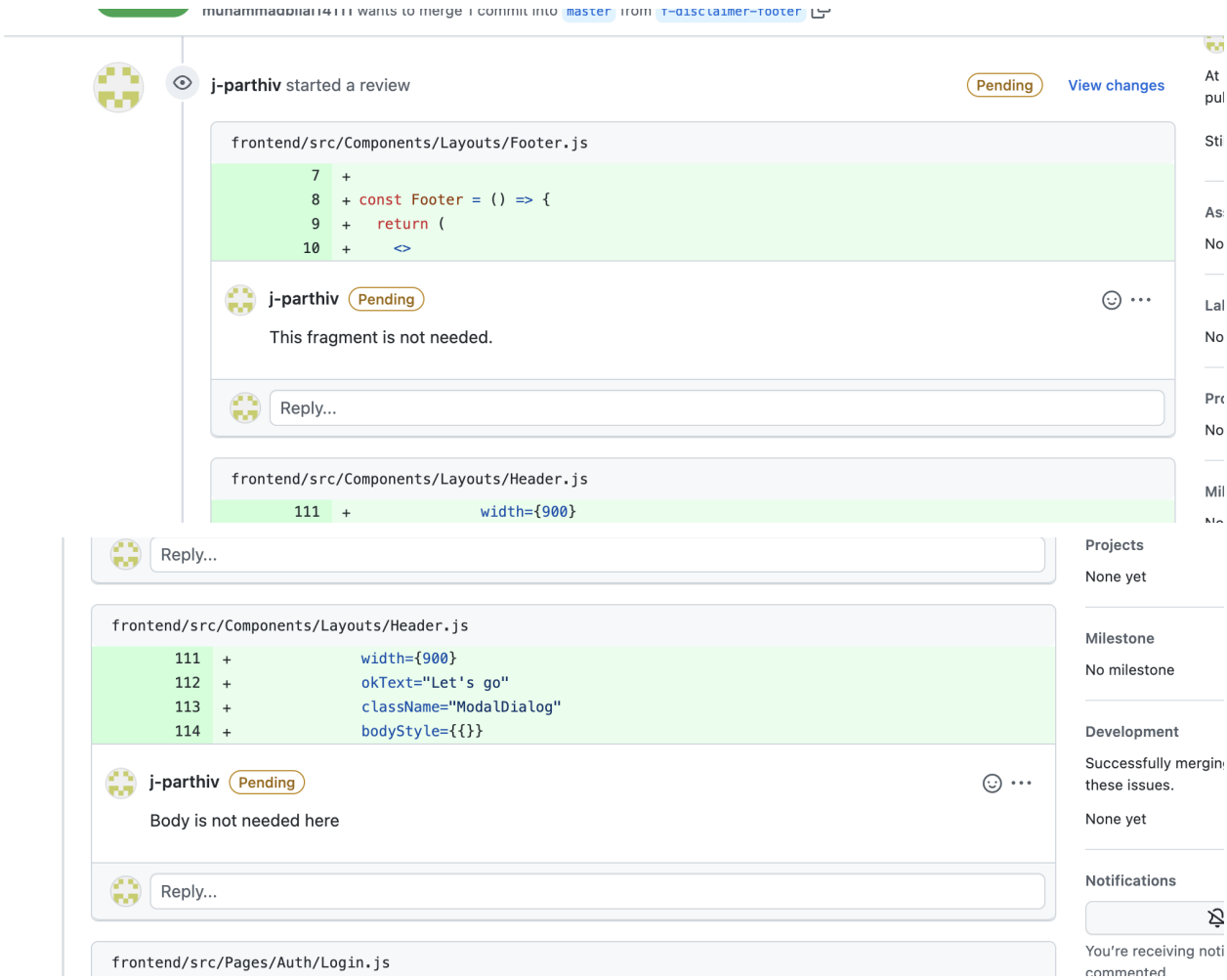
Sanjay-George on Jan 18

@kikolazeski Let's delete this file. It's redundant

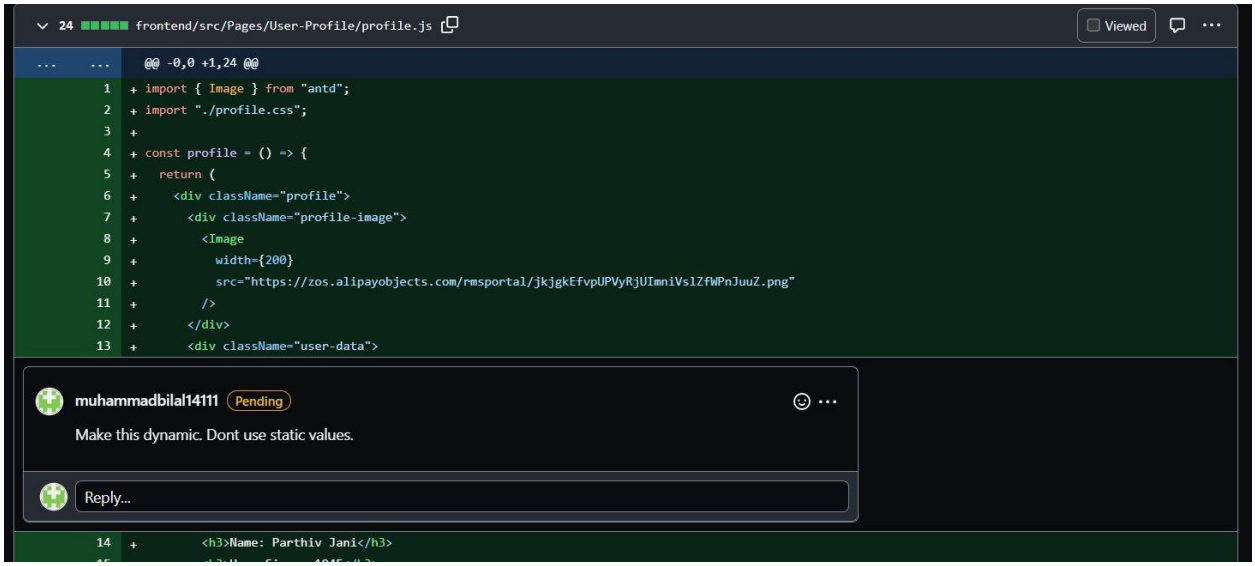
Reply...

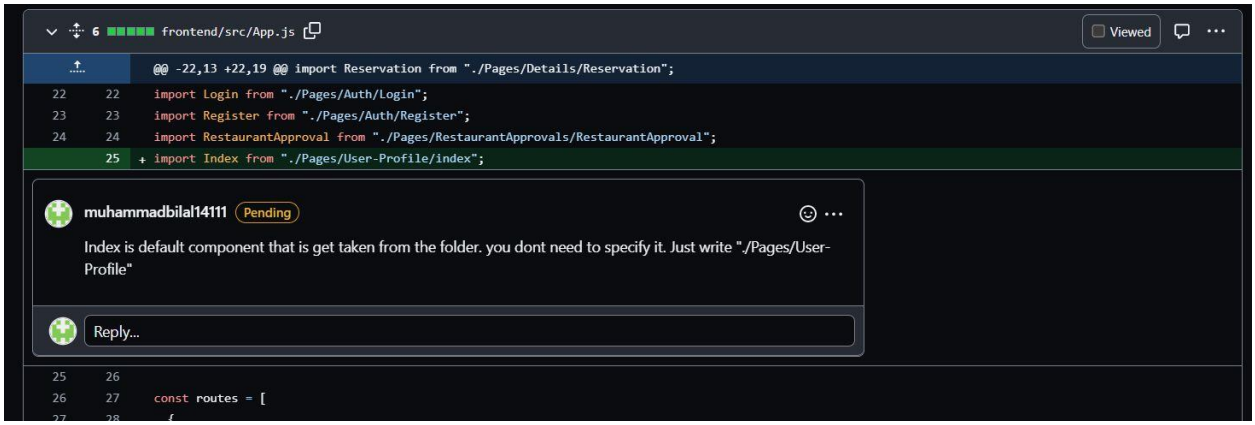
4. Parthiv Yogesh Kumar Jani
    a. Add Disclaimer Message and Responsiveness



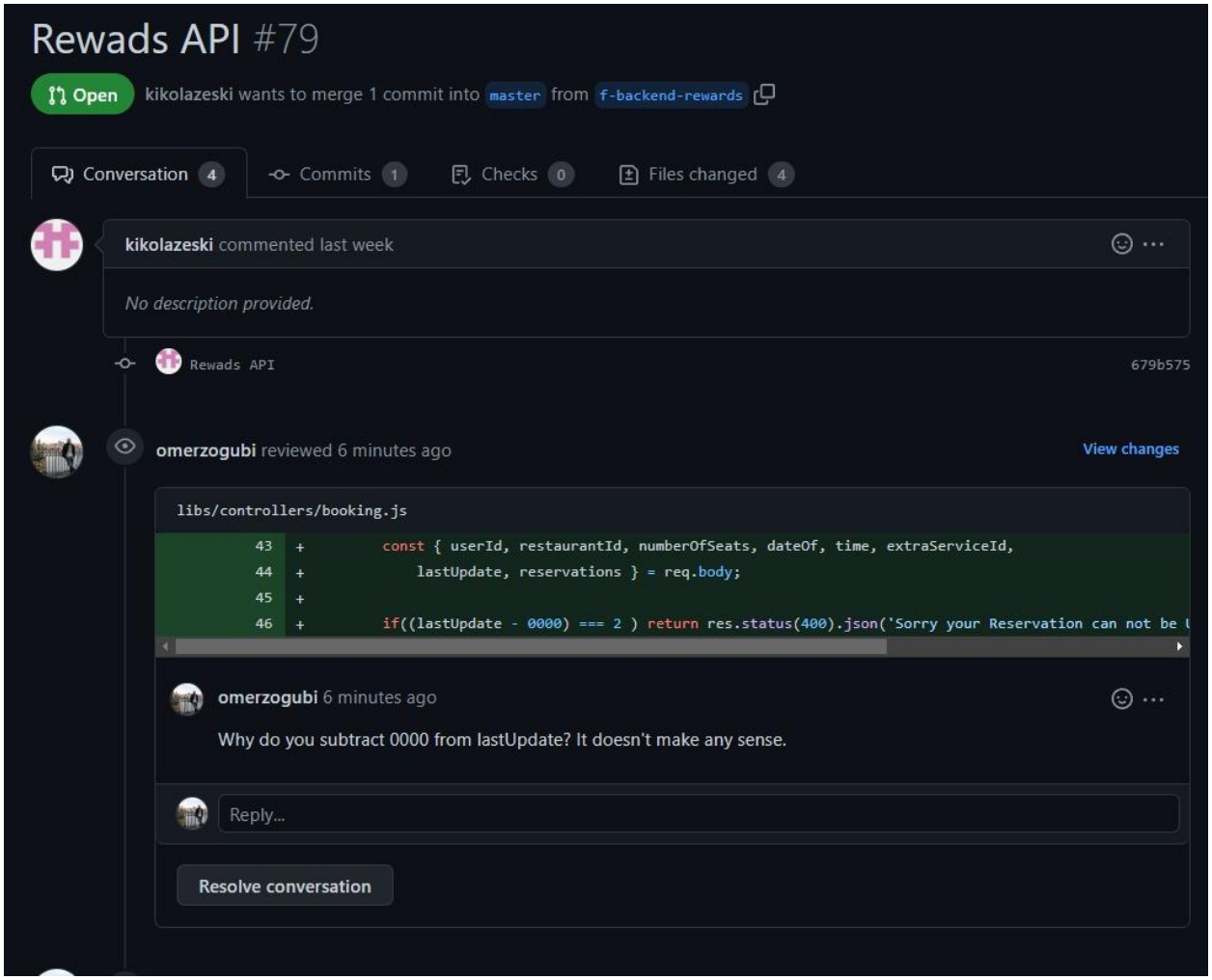


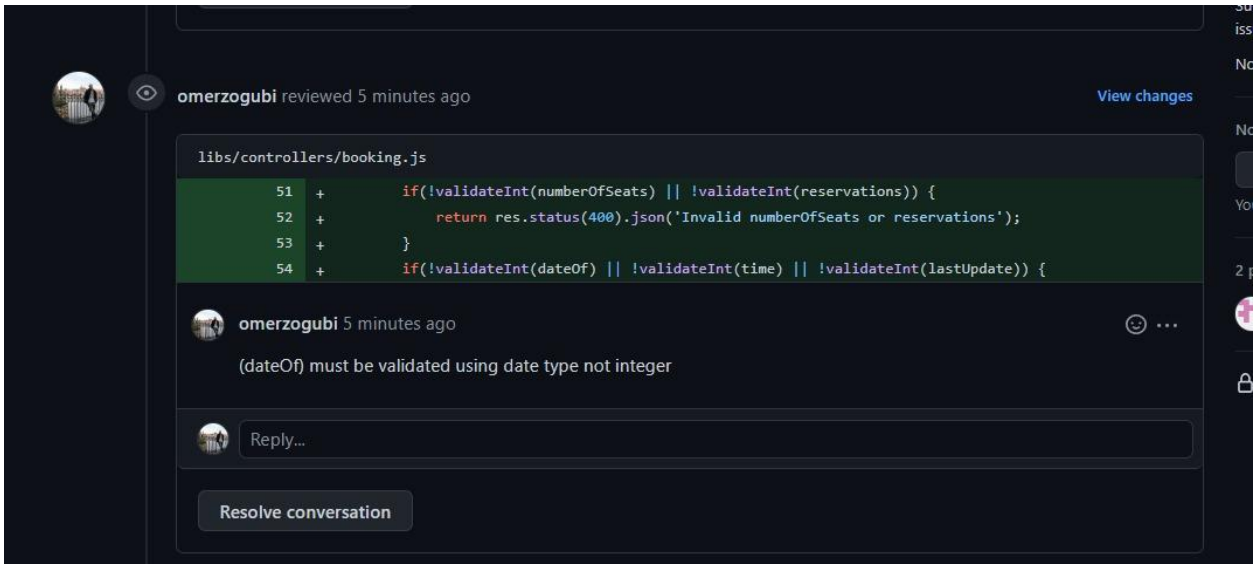
5. Muhammad Bilal
    b. Make User Profile

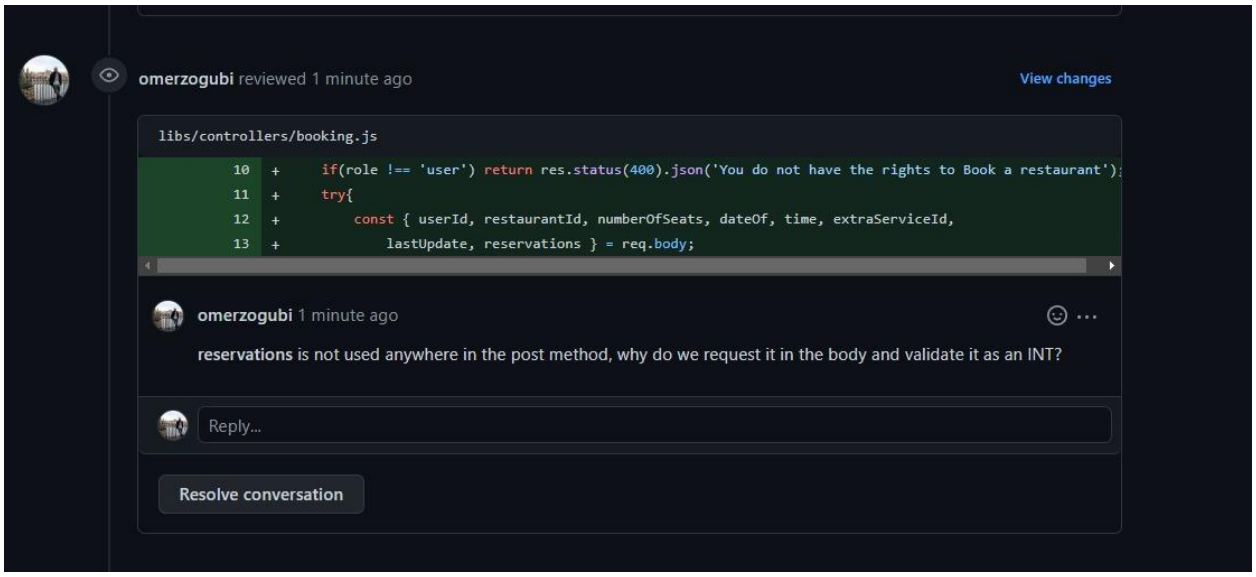


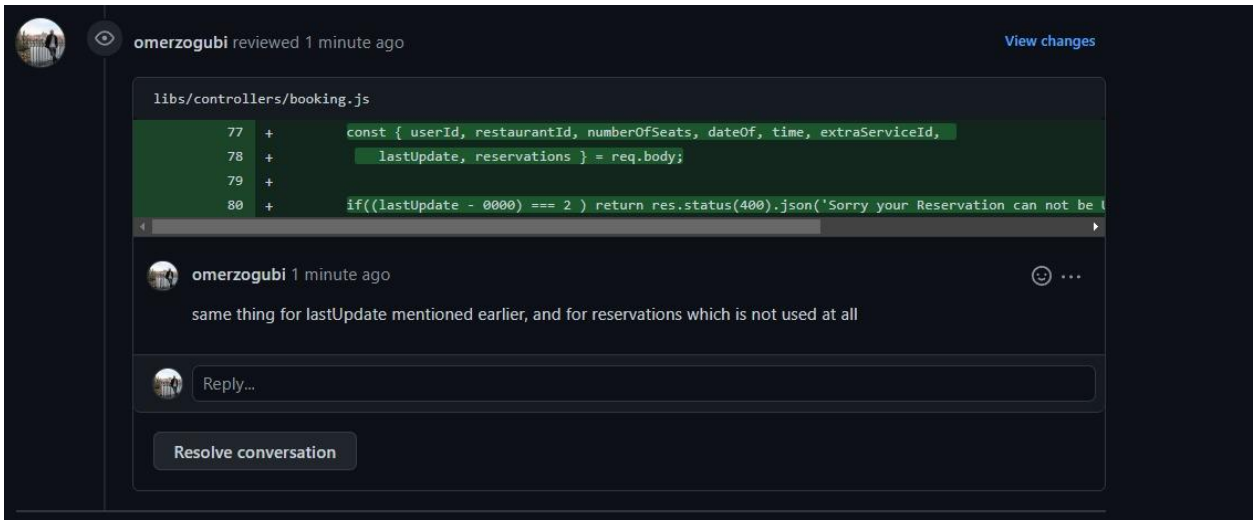


6. ÖMER ZOĞUBİ
    a. Rewards API


Rewads API #79
Open
kikolazeski wants to merge 1 commit into master from f-backend-rewards
Conversation 4
Commits 1
Checks 0
Files changed 4
kikolazeski commented last week
No description provided.
Rewads API
679b575
omerzogubi reviewed 6 minutes ago
View changes
libs/controllers/booking.js
43 + const { userId, restaurantId, numberOfSeats, dateOf, time, extraServiceId,
44 + lastUpdate, reservations } = req.body;
45 +
46 + if((lastUpdate - 0000) === 2) return res.status(400).json('Sorry your Reservation can not be
omerzogubi 6 minutes ago
Why do you subtract 0000 from lastUpdate? It doesn't make any sense.
Reply...
Resolve conversation


omerzogubi reviewed 5 minutes ago
View changes
libs/controllers/booking.js
51 + if(!validateInt(numberOfSeats) || !validateInt(reservations)) {
52 + return res.status(400).json('Invalid numberOfSeats or reservations');
53 + }
54 + if(!validateInt(dateOf) || !validateInt(time) || !validateInt(lastUpdate)) {
omerzogubi 5 minutes ago
(dateOf) must be validated using date type not integer
Reply...
Resolve conversation





21. Self-check on best practices for security

Assets:

1. Customer data (e.g. names, contact information)
2. Restaurant data (e.g. menus, location, availability)
3. Website functionality (e.g. booking forms)

Threats:

1. Data breaches or hacks targeting customer information
2. Malicious attacks on the website or server infrastructure
3. Payment fraud or theft

Protection measures:

1. Using strong encryption to protect customer data in transit and at rest, and implementing robust access controls to limit access to sensitive information
2. Regularly testing and updating website security measures, such as firewalls and intrusion detection systems, to protect against attacks
3. Implementing secure payment processing systems, such as tokenization or two-factor authentication, to reduce the risk of payment fraud.

Confirmations:

1. Yes, it is important to encrypt passwords in the database to protect against unauthorized access in the event of a data breach.
2. For the search bar input validation, the code might include checks for the length of the input string, as well as any special characters or other characters that may be invalid or could indicate an attempted attack (e.g. SQL injection).

As Security is the most important part of each Website that is being hosted on the Web, we as a Project should stay realistic and implement as much Security mechanism as we can after we finish all of our First Priority Functional Requirements.

22. Self-check: Adherence to original Non-functional specs – performed by team leads

ID	Functional Requirement	Status
1	The Application is a Vendor for Multiple Restaurants	DONE
2	User can Search for Restaurants by name, cuisine, food, and rating	DONE
2.1	Search results can be filtered additionally and sorted	DONE
3	User can reserve a table and specify the number of seats in a given Restaurant	DONE
3.1	User reserves one or Multiple Tables at the Restaurant	DONE
3.2	User specifies the number of seats	DONE
4	Restaurants can be added, removed and updated by Restaurant owners	DONE

4.1	Restaurants can be Added by the Restaurant Owners	DONE
4.2	Restaurants can be Removed by the Owners of given Restaurant	DONE
4.3	Restaurants can be Updated by the owners of said Restaurant	DONE
5	There is a Grace Period to Cancel or update the booking of the table	DONE
5.1	Every User can Cancel the reservation up to 2h before said reservation	DONE
6	The number of booked seats are specified per table	DONE
7	Users can view Menus of the Restaurants	DONE
8	Users search for restaurants, make/change/cancel reservations and also post reviews.	DONE
8.1	User can cancel Reservation	DONE
8.2	User can change Reservation	DONE
8.3	User can Post Reviews	DONE
9	User can View the Restaurant Reviews	DONE
10	Every booking is associated with an account	DONE
11	One Account can not be associated with multiple users	DONE
12	User can chat with the Restaurant Host	DONE
13	User accumulates Reward Points for every reservations with the application	DONE

23. Code that was Used to show and tell

23.1. Utsav Shrestha

23.1.1 Booking-backend.js

```
const express = require("express");
const router = express.Router();
const bookingDB = require("../database/booking");
const {
 validateInt,
 isStringUndefinedOrEmpty,
} = require("../utils/inputValidator");
```

```
const auth = require("../utils/auth");

router.post("/", auth, async (req, res) => {
 const role = req.role;
 const userId = req.userId;

 if (role !== "user")
   return res
     .status(400)
     .json("You do not have the rights to Book a restaurant");
 try {
   const {
     restaurantId,
     numberOfSeats,
     date,
     time,
     extraServiceId = 0,
     specialRequest = "",
   } = req.body;

   if (!validateInt(restaurantId)) {
     return res
       .status(400)
       .json("Invalid user, restaurant or Extra Service ID");
   }
   if (!validateInt(numberOfSeats)) {
     return res.status(400).json("Invalid numberOfSeats or reservations");
   }

   const resp = await bookingDB.addReservation({
     userId,
     restaurantId,
     numberOfSeats,
     date,
     time,
     extraServiceId,
     specialRequest,
   });
   return res.send({
     msg: "Booking succesful",
     url: `api/booking`,
     reservationId: resp?.insertId,
```

```
    });
  } catch (ex) {
    console.log("ex", ex);
    if (ex.statusCode === 403) {
      return res.status(403).send({ msg: ex.msg });
    }
    return res.sendStatus(500);
  }
});

router.delete("/cancel/:id", auth, async (req, res) => {
  const role = req.role;
  const userId = req.userId;
  const { id: reservationId } = req.params;

  if (role != "user")
    return res
      .status(400)
      .json("You do not have the rights to Update a booking");
  try {
    await bookingDB.cancelReservation({
      userId,
      reservationId,
    });
    return res.send({ msg: "Reservation Cancelled", url: `api/booking` });
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.put("/update", auth, async (req, res) => {
  const role = req.role;
  const userId = req.userId;

  if (role !== "user")
    return res
      .status(400)
      .json("You do not have the rights to Update a booking");
  try {
    const {
      restaurantId,
```

```
    numberOfSeats,
    date,
    time,
    extraServiceId,
    reservationId,
    specialRequest,
  } = req.body;

  if (
    !validateInt(userId) ||
    !validateInt(restaurantId) ||
    !validateInt(extraServiceId)
  ) {
    return res
      .status(400)
      .json("Invalid user, restaurant or Extra Service ID");
  }
  if (!validateInt(numberOfSeats)) {
    return res.status(400).json("Invalid numberOfSeats");
  }
  if (!validateInt(date) || !validateInt(time)) {
    return res.status(400).json("Invalid date or time");
  }

  await bookingDB.updateReservation({
    userId,
    restaurantId,
    numberOfSeats,
    date,
    time,
    extraServiceId,
    reservationId,
    specialRequest,
  });
  return res.send({ msg: "Reservation Updated", url: `api/booking` });
 } catch (ex) {
  // console.error(ex);
  return res.sendStatus(500);
 }
});

router.post("/checkReservationAvailability/:restaurantId", async (req, res) => {
```

```
 const { restaurantId } = req.params;
 const { time, date, numberOfSeats } = req.body;

 if (!validateInt(restaurantId)) {
   return res.sendStatus(400);
 }

 try {
   await bookingDB.checkReservationAvailability(
     restaurantId,
     time,
     date,
     numberOfSeats
   );

   return res.send({ msg: "Reservation available", url: `api/booking` });
 } catch (ex) {
   if (ex.status == "403") {
     return res.status(403).send({ msg: ex.msg });
   }
   return res.sendStatus(500);
 }
});

router.get("/user-reservations", auth, async (req, res) => {
 const role = req.role;
 const userId = req.userId;

 if (role !== "restaurantOwner")
   return res.status(400).json("You do not have the rights to view these");
 try {
   const data = await bookingDB.getUserReservations(userId);

   return res.send(data);
 } catch (ex) {
   console.log(ex);
   return res.sendStatus(500);
 }
});

module.exports = router;
```

```
// libs/database/booking.js
const { pool } = require("./config");
const dayjs = require("dayjs");

const addReservation = ({
 userId,
 restaurantId,
 numberOfSeats,
 date,
 time,
 extraServiceId,
 specialRequest,
}) => {
 const convertedDate = dayjs(date).format("YYYY-MM-DD");
 const lastUpdated = dayjs().format("YYYY-MM-DD");

 const insertReservationSql = `INSERT INTO reservations
           (userId, restaurantId, numberOfSeats, status, extraServiceId,
           lastUpdate , times, date, specialRequest)
           VALUES (${userId}, ${restaurantId} ,${numberOfSeats}, 'Reserved',
           ${extraServiceId}, '${lastUpdated}', ${time}, '${date}',
'${specialRequest}')`;

 return new Promise((resolve, reject) => {
   pool.query(
     insertReservationSql,
     (insertReservationError, insertReservationResult) => {
       if (insertReservationError) {
         return reject(insertReservationError);
       }
       const seatsBookedSql = `SELECT * FROM restaurantTime WHERE
restaurantID=${restaurantId} AND times=${time} AND date='${convertedDate}'`;

       pool.query(
         seatsBookedSql,
         (seatsBookedSqlError, seatsBookedSqlResult) => {
           if (seatsBookedSqlError) {
             return reject(seatsBookedSqlError);
           }

           var seatsBooked = seatsBookedSqlResult[0]?.seatsBooked || 0;
           var totalSeats = numberOfSeats + seatsBooked;
```

```
          let restaurantTimeQuery = `UPDATE restaurantTime SET seatsBooked =
${totalSeats} WHERE restaurantID = ${restaurantId} AND times = ${time} AND date =
'${convertedDate}'`;

          if (seatsBookedSqlResult.length === 0) {
            restaurantTimeQuery = `
              INSERT INTO restaurantTime (times, availability, restaurantID,
seatsBooked, date)
                VALUES (${time}, true, ${restaurantId} ,${totalSeats},
'${convertedDate}')
            `;
          }

          let getRewardPointsSql = `SELECT * from rewards where userID=${userId}`;
          pool.query(
            restaurantTimeQuery,
            (updateRestaurantError, updateRestaurantResult) => {
              if (updateRestaurantError) {
                return reject(updateRestaurantError);
              }
              pool.query(
                getRewardPointsSql,
                (rewardPointsSqlError, rewardPointsSqlResult) => {
                  if (rewardPointsSqlError) {
                    return reject(rewardPointsSqlError);
                  }

                  let prevRewardPoint = rewardPointsSqlResult[0]?.points || 0;
                  let newRewardPoint = prevRewardPoint + 25;

                  let updateRewardpointsSql = `UPDATE rewards SET
points=${newRewardPoint} where userID=${userId}`;

                  if (rewardPointsSqlResult.length === 0) {
                    updateRewardpointsSql = `INSERT INTO rewards (userId, typeOf,
points)
                    VALUES ('${userId}', 'ForReservation', '25')`;
                  }
                  pool.query(
                    updateRewardpointsSql,
                    (pointsError, pointsResult) => {
```

```
                        if (pointsError) {
                          return reject(pointsError);
                        }
                        resolve(insertReservationResult);
                      }
                    );
                  }
                );
              }
            );
          }
        );
      }
    );
  });
};

const cancelReservation = ({ userId, reservationId }) => {
 const resTimeSql = `SELECT * FROM reservations WHERE userId = ${userId} AND id =
${reservationId}`;
 return new Promise((resolve, reject) => {
   pool.query(resTimeSql, (error, resTimeResult) => {
     if (error) reject(error);
     const reservationRestaurantId = resTimeResult[0].restaurantId;
     const reservationTime = resTimeResult[0].times;
     const reservationBookedSeats = resTimeResult[0].numberOfSeats;
     const tableBookedSeatsSql = `SELECT seatsBooked FROM restaurantTime WHERE
restaurantId =${reservationRestaurantId} AND times =${reservationTime}`;
     pool.query(tableBookedSeatsSql, (error2, tableBookedSeatsResult) => {
       if (error2) reject(error2);
       var tableBookedSeats = tableBookedSeatsResult[0].seatsBooked;
       var totalSeats = tableBookedSeats - reservationBookedSeats;
       const cancelSql = `UPDATE reservations SET status = 'Cancelled' WHERE userId =
${userId} AND id = ${reservationId}`;
       const updateRestaurantTimeSql = `UPDATE restaurantTime SET seatsBooked =
${totalSeats} WHERE restaurantId = ${reservationRestaurantId} AND times =
${reservationTime}`;
       if (resTimeResult) {
         pool.query(cancelSql, (error3, cancelResult) => {
           if (error3) reject(error3);
           pool.query(
             updateRestaurantTimeSql,
```

```
            (error4, updateRestaurantTimeResult) => {
              if (error4) reject(error4);
              let getRewardPointsSql = `SELECT * from rewards where
userID=${userId}`;

              pool.query(
                getRewardPointsSql,
                (rewardPointsSqlError, rewardPointsSqlResult) => {
                  if (rewardPointsSqlError) {
                    return reject(rewardPointsSqlError);
                  }

                  let prevRewardPoint = rewardPointsSqlResult[0]?.points || 0;
                  let newRewardPoint = prevRewardPoint - 25;

                  let updateRewardpointsSql = `UPDATE rewards SET
points=${newRewardPoint} where userID=${userId}`;

                  pool.query(
                    updateRewardpointsSql,
                    (pointsError, pointsResult) => {
                      if (pointsError) {
                        return reject(pointsError);
                      }
                      resolve(pointsResult);
                    }
                  );
                }
              );
            }
          );
        });
      }
    });
  });
});
};

const updateReservation = ({
 userId,
 restaurantId,
 numberOfSeats,
```

```
 date,
 time,
 extraServiceId,
 reservationId,
 specialRequest,
}) => {
 const convertedDate = dayjs(date).format("YYYY-MM-DD");
 const lastUpdated = dayjs().format("YYYY-MM-DD");

 const sql = `UPDATE reservations SET userId = ${userId}, restaurantId =
${restaurantId},
   numberOfSeats = ${numberOfSeats}, date = '${convertedDate}', times = ${time},
   extraServiceId = ${extraServiceId}, lastUpdate = '${lastUpdated}',
specialRequest='${specialRequest}'
   WHERE userId = ${userId} AND id = ${reservationId}`;

 return new Promise((resolve, reject) => {
   pool.query(sql, (error, result, fields) => {
     if (error) reject(error);
     resolve(result);
   });
 });
};

const checkReservationAvailability = (
 restaurantId,
 time,
 date,
 numberOfSeats
) => {
 const convertedDate = dayjs(date).format("YYYY-MM-DD");
 let currentDate = dayjs().format("YYYY-MM-DD");

 if (convertedDate >= currentDate) {
   const query = `SELECT * FROM restaurantTime WHERE restaurantID=${restaurantId} AND
times=${time} AND date='${convertedDate}'`;
   return new Promise((resolve, reject) => {
     pool.query(query, (error, results) => {
       if (error) {
         reject(error);
       } else {
```

```
        const restaurantSql = `SELECT maxCapacity FROM restaurants where
id=${restaurantId}`;
        pool.query(
          restaurantSql,
          (restaurantSqlError, restaurantSqlResults) => {
            if (restaurantSqlError) {
              reject(restaurantSqlError);
            } else {
              const maxCapacity = restaurantSqlResults[0]?.maxCapacity || 0;

              const seatsBooked = results[0]?.seatsBooked || 0;

              const totalSeats = numberOfSeats + seatsBooked;

              if (totalSeats <= maxCapacity) {
                resolve(results);
              } else {
                const customError = new Error("Hello");
                customError.status = "403";
                customError.msg = `Maximum seats booked for this time. Seats
available for this time is ${
                  maxCapacity - seatsBooked
                }`;
                reject(customError);
              }
            }
          }
        );
      }
    });
  });
 } else {
  const customError = new Error("Hello");
  customError.status = "403";
  customError.msg = `Booking date must be greated than current date`;
  throw customError;
 }
};

const getReservationByUser = (userId, restaurantId) => {
 const query = `select * from reservations where userid = ? and restaurantId = ? and
status != "cancelled"`;
```

```
 return new Promise((resolve, reject) => {
   pool.query(
     query,
     [userId, restaurantId],
     function (error, results, fields) {
       if (error) reject(error);
       resolve(results);
     }
   );
 });
};

const getUserReservations = (userId) => {
 const query = `SELECT reserv.*, res.name as restaurantName, res.address as
restaurantAddress, res.city as restaurantCity, u.lastName, u.firstName, u.email as
userEmail, u.phoneNumber as userPhone from reservations as reserv LEFT JOIN
restaurants as res ON reserv.restaurantId=res.id LEFT JOIN users as u ON
reserv.userId=u.id WHERE res.userId=${userId} GROUP BY reserv.id`;

 return new Promise((resolve, reject) => {
   pool.query(query, function (error, results, fields) {
     if (error) reject(error);
     resolve(results);
   });
 });
};

module.exports = {
 addReservation,
 cancelReservation,
 updateReservation,
 checkReservationAvailability,
 getReservationByUser,
 getUserReservations,
};
```

23.1.2 Booking-frontend.js

```
import { useMutation } from "react-query";
import {
```

```
 Button,
 Row,
 Col,
 Form,
 Divider,
 Select,
 DatePicker,
 message,
} from "antd";
import { useNavigate } from "react-router-dom";

import "./ReservationBox.css";
import { Persons, Times } from "../../utils/data";
import { checkReservationAvailability } from "../../query/restaurant";
import dayjs from "dayjs";

const ReservationBox = ({ resId }) => {
 const navigate = useNavigate();

 const checkReserveAvailableMutation = useMutation((body) =>
   checkReservationAvailability(resId, body)
 );

 const onFinish = (values) => {
   const body = {
     ...values,
     date: dayjs(values?.date).format("YYYY-MM-DD"),
   };

   checkReserveAvailableMutation.mutate(body, {
     onSuccess: () => {
       message.success("Reservations available, redirecting...");
       setTimeout(() => {
         {
           const queryParams = new URLSearchParams({
             ...body,
             resId,
           });
           navigate(`/reservation?${queryParams.toString()}`);
         }
       }, 1000);
     },
```

```
    onError: (data) => {
      const errorMessage = data?.response?.data?.message;
      message.error(errorMessage);
    },
  });
};

const disabledDate = (current) => {
  // Can not select days before today and today
  return current && current < dayjs().endOf("day");
};

return (
  <>
    <div className="ReserveBox">
      <div className="card_body">
        <h2 className="ReserveBox_title">Make a reservation</h2>
        <Divider
          style={{
            borderWidth: 1,
            borderColor: "#C3c9cb",
            marginTop: 10,
          }}
        />
        <Form
          name="basic"
          style={{ maxWidth: 600 }}
          initialValues={{ remember: true }}
          onFinish={onFinish}
          autoComplete="off"
        >
          <Row>
            <Col xs={24}>
              <Form.Item
                name="numberOfSeats"
                rules={[{ required: true, message: "Please select people!" }]}
              >
                <Select
                  placeholder="Select People"
                  className="ddstyle"
                  style={{ width: "100%" }}
                >
```

```
                {Persons.map((person, index) => {
                  return (
                    <Select.Option key={index} value={index + 1}>
                      {person}
                    </Select.Option>
                  );
                })}
              </Select>
            </Form.Item>
          </Col>
          <Col xs={12}>
            <Form.Item
              name="date"
              rules={[{ required: true, message: "Please set the date!" }]}
            >
              <DatePicker disabledDate={disabledDate} />
            </Form.Item>
          </Col>
          <Col xs={12}>
            <Form.Item
              name="time"
              rules={[{ required: true, message: "Please select time!" }]}
            >
              <Select placeholder="Select Time" className="ddstyle">
                {Times.map((time, index) => {
                  return (
                    <Select.Option key={index} value={time?.value}>
                      {time?.label}
                    </Select.Option>
                  );
                })}
              </Select>
            </Form.Item>
          </Col>
        </Row>
        <Button htmlType="submit" type="primary" className="reserveBox-btn">
          Check Reservation Available
        </Button>
      </Form>
    </div>
  </div>
</>
```

```
 );
};

export default ReservationBox;

// reservation page
import React, { useState, useEffect } from "react";

import "./Reservation.css";
import { Image, Form, Row, Col, Select, Input, Button, message } from "antd";
import {
 FieldTimeOutlined,
 CalendarOutlined,
 UserOutlined,
} from "@ant-design/icons";
import { useQuery, useMutation } from "react-query";

import {
 restaurantDetailsById,
 reserveRestaurant,
 updateReserveRestaurant,
 cancelReserveRestaurant,
} from "../../query/restaurant";
import { getAllExtraServices } from "../../query/searchFilters";
import { Link, useNavigate, useSearchParams } from "react-router-dom";

const { TextArea } = Input;

export default function Reservation() {
 const [form] = Form.useForm();
 const navigate = useNavigate();
 const [searchParams] = useSearchParams();
 const resId = searchParams.get("resId");
 const numberOfSeats = searchParams.get("numberOfSeats");
 const date = searchParams.get("date");
 const time = searchParams.get("time");
 const [reserveStatus, setReserveStatus] = useState("Pending");
 const [reservationId, setReservationId] = useState(null);

 const userToken = localStorage.getItem("token");
 const userName = localStorage.getItem("userName");
```

```
const { data: restaurantDetails } = useQuery(
  ["restaurant-listings", { resId }],
  () => restaurantDetailsById(resId)
);

const { data: allExtraServices } = useQuery(
  "all-extra-service",
  getAllExtraServices
);

const reserveRestaurantMutation = useMutation((body) =>
  reserveRestaurant(body)
);

const updateReserveRestaurantMutation = useMutation((body) =>
  updateReserveRestaurant(body)
);

const cancelReserveRestaurantMutation = useMutation((id) =>
  cancelReserveRestaurant(id)
);
const restaurantData = restaurantDetails?.data[0];

useEffect(() => {
  if (reserveRestaurantMutation?.data?.data?.reservationId) {
    setReservationId(reserveRestaurantMutation?.data?.data?.reservationId);
  }
}, [reserveRestaurantMutation]);

const onFinish = (values) => {
  const body = {
    ...values,
    date,
    time: parseInt(time),
    numberOfSeats: parseInt(numberOfSeats),
    restaurantId: parseInt(resId),
    ...(reserveStatus === "Modify" && { reservationId }),
  };

  reserveStatus === "Modify"
    ? updateReserveRestaurantMutation.mutate(body, {
        onSuccess: () => {
```

```
          message.success("Restaurant Reserve modified successfully");
          setReserveStatus("Reserved");
        },
        onError: (data) => {
          const errorMessage = data?.response?.data;
          message.error(errorMessage);
        },
      })
    : reserveRestaurantMutation.mutate(body, {
        onSuccess: () => {
          message.success("Restaurant reserved successfully");
          setReserveStatus("Reserved");
        },
        onError: (data) => {
          const errorMessage = data?.response?.data;
          message.error(errorMessage);
        },
      });
};

const cancelReservation = () => {
  cancelReserveRestaurantMutation.mutate(reservationId, {
    onSuccess: () => {
      message.success("Restaurant cancelled successfully");
      setTimeout(() => {
        navigate(`/details/${resId}`);
      }, [300]);
    },
    onError: (data) => {
      const errorMessage = data?.response?.data;
      message.error(errorMessage);
    },
  });
};

return (
  <>
    <div className="main-class" style={{ maxWidth: 800, margin: "0 auto" }}>
      <Row gutter={30}>
        <Col sm={6}>
          <h2>One last step!</h2>
          <Image
```

```
            width={200}
            style={{ borderRadius: "50%", marginTop: 20 }}

src="https://zos.alipayobjects.com/rmsportal/jkjgkEfvpUPVyRjUImniVslZfWPnJuuZ.png?x-oss-process=image/blur,r_50,s_50/quality,q_1/resize,m_mfit,h_200,w_200"
            preview={false}
          />
        </Col>
        <Col sm={10} className="res-detail">
          <div style={{ textAlign: "left" }}>
            {" "}
            <h2>{restaurantData?.name}</h2>{" "}
          </div>
          <div>
            <div className="res-detail-child">
              <CalendarOutlined />
              <div>{date}</div>
            </div>
            <div className="res-detail-child">
              <FieldTimeOutlined />
              <div>{time}:00</div>
            </div>
            <div className="res-detail-child">
              <UserOutlined />
              <div>{numberOfSeats} people</div>
            </div>
            {reserveStatus !== "Modify" && (
              <div
                className="reserve-actions"
                style={{ textAlign: "left", marginTop: 10 }}
              >
                <div onClick={() => setReserveStatus("Modify")}>
                  <span className="EditFields">Modify</span>
                </div>
                {reserveStatus !== "Pending" && (
                  <div onClick={cancelReservation}>
                    <span className="EditFields" style={{ color: "#625f5f" }}>
                      Cancel
                    </span>
                  </div>
                )}
              </div>
```

```
          )}
        </div>
      </Col>
    </Row>
    <div style={{ textAlign: "left", marginTop: 10 }}>
      {!userToken && (
        <div>
          <Link to="/login">
            <b>Login</b>
          </Link>{" "}
          to collect points for this reservation
        </div>
      )}
    </div>
    {reserveStatus === "Reserved" && (
      <div className="show-other-info">
        <Link to={`/details/${resId}`}>
          <div className="back-to-res">Back to the restaurant</div>
        </Link>
        <div className="booked-by">
          <b>Booked By: </b>
          {userName}
        </div>
        <div className="special-request">
          <h3>Your Special Request</h3>
          <div>{form.getFieldValue("specialRequest")}</div>
        </div>
      </div>
    )}
    {reserveStatus !== "Reserved" && (
      <Form
        name="basic"
        style={{ marginTop: 20 }}
        initialValues={{ remember: true }}
        onFinish={onFinish}
        autoComplete="off"
        form={form}
      >
        <Row gutter={30} style={{ width: "60%" }}>
          <Col xs={24}>
            <Form.Item label="" name="extraServiceId">
              <Select
```

```
                placeholder="Select extra service"
                className="ddstyle"
                style={{ width: "100%" }}
                disabled={reserveStatus === "Reserved"}
              >
                {allExtraServices?.data.map((service, index) => {
                  return (
                    <Select.Option key={index} value={service.id}>
                      {service.value}
                    </Select.Option>
                  );
                })}
              </Select>
            </Form.Item>
          </Col>
          <Col xs={24}>
            <Form.Item name="specialRequest">
              <TextArea
                rows={3}
                placeholder="Special Request"
                disabled={reserveStatus === "Reserved"}
              />
            </Form.Item>
          </Col>
          <Col xs={24}>
            <Button
              htmlType="submit"
              type="primary"
              className="reserveBox-btn"
              disabled={reserveStatus === "Reserved"}
            >
              {reserveStatus === "Reserved" ? "Modify Reserve" : "Reserve"}
            </Button>
          </Col>
        </Row>
      </Form>
    )}
  </div>
 </>
);
}
```

```
// query/reestaurant
import http from "../utils/http";

export function restaurantDetailsById(id) {
 return http({
   url: `/restaurants/${id}`,
   method: "get",
 });
}

export function reserveRestaurant(data) {
 return http({
   url: `/booking`,
   method: "post",
   data,
 });
}



export function updateReserveRestaurant(data) {
 return http({
   url: `/booking/update`,
   method: "put",
   data,
 });
}

export function cancelReserveRestaurant(id) {
 return http({
   url: `/booking/cancel/${id}`,
   method: "delete",
 });
}

export function checkReservationAvailability(restaurantId, data) {
 return http({
   url: `/booking/checkReservationAvailability/${restaurantId}`,
   method: "post",
   data,
 });
}
```

23.1.3 ViewEditRestaurant-frontend.js

```
import React, { useState } from 'react'
import { useQuery } from 'react-query'

import { getMyRestaurants } from '../../query/restaurants'
import VerticalCard from '../../components/Cards/VerticalCard'
import { Button } from 'antd'
import { Link } from 'react-router-dom'
import { CToast, CToastBody, CToastClose } from '@coreui/react'

const ViewRestaurants = () => {
 const { data: myRestaurantData } = useQuery('my-restaurants', () =>
getMyRestaurants(20))
 const [showToast, setShowToast] = useState({
   visible: false,
   message: '',
 })
 return (
   <div>
     <div className="view-res-wrapper">
       <Link to="/dashboard/add-restaurant">
         <Button>Add Restaurant</Button>
       </Link>
     </div>
     <CToast autohide={false} visible={showToast?.visible}
className="align-items-right">
       <div className="d-flex">
         <CToastBody>{showToast?.message}</CToastBody>
         <CToastClose className="me-2 m-auto" />
       </div>
     </CToast>
     {myRestaurantData?.data?.length > 0
       ? myRestaurantData?.data?.map((data) => (
           <VerticalCard
             name={data?.name}
             address={data?.address}
             city={data?.city}
             restaurantNote={data?.restaurantNote}
             maxCapacity={data?.maxCapacity}
             status={data?.status}
             id={data?.id}
             key={data?.id}
```

```
            setShowToast={setShowToast}
            images={data?.images}
          />
        ))
      : 'No Restaurants available'}
  </div>
 )
}

export default ViewRestaurants

// veritcal card

import React, { useState } from 'react'
import './Card.css'
import defaultImage from '../../assets/images/RestImages/default-restaurant.jpeg'
import { Button, Tag, Popconfirm } from 'antd'
import { useMutation, useQueryClient } from 'react-query'
import { deleteRestaurant } from '../../query/restaurants'
import EditRestaurant from '../../views/restaurants/editRestaurants'

const VerticalCard = (props) => {
 const queryClient = useQueryClient()
 const [isModalOpen, setIsModalOpen] = useState(false)
 const [resId, setResId] = useState(null)

 const showModal = () => {
   setIsModalOpen(true)
 }

 const handleOk = () => {
   setIsModalOpen(false)
 }

 const handleCancel = () => {
   setIsModalOpen(false)
 }

 const deleteRestaurantMutation = useMutation((id) => deleteRestaurant(id))

 const confirmDeleteRestaurant = (e) => {
   deleteRestaurantMutation.mutate(props?.id, {
```

```
    onSuccess: () => {
      queryClient.invalidateQueries(['my-restaurants'])
      props?.setShowToast({
        visible: true,
        message: 'Restaurant deleted successfully',
      })
    },
    onError: (data) => {
      const errorMessage = data?.response?.data?.msg
      props?.setShowToast({
        visible: true,
        message: errorMessage,
      })
    },
  })
}

const allImages = props?.images ? props?.images?.filter((img) => !img.menuImage) : []

return (
  <div className="VerCard">
    <div className="VerCardImgContainer">
      <img
        src={`${process.env.REACT_APP_IMAGE_STORE_URL}/${allImages[0]?.path}`}
        className="VerCard_Img"
        alt=""
        onError={(e) => {
          e.target.src = defaultImage
        }}
      />
    </div>
    <div class="VerCard_Content">
      <div className="ver-card-wrapper">
        <h3 className="VerCard_title">
          <span style={{ marginRight: 10 }}>{props.name}</span>
          <Tag
            color={
              props?.status === 'approved'
                ? 'green'
                : props?.status === 'rejected'
                ? 'red'
                : 'blue'
```

```
            }
          >
            {props?.status}
          </Tag>
        </h3>
        <div>
          {/* <Link to={`/dashboard/edit-restaurant/${props?.id}`}> */}
          <Button
            onClick={() => {
              showModal()
              setResId(props?.id)
            }}
          >
            Edit
          </Button>
          {/* </Link> */}
          <Popconfirm
            title="Delete restaurant"
            description="Are you sure to delete this restaurant?"
            onConfirm={confirmDeleteRestaurant}
            okText="Yes"
            cancelText="No"
          >
            <Button style={{ backgroundColor: 'red', color: '#fff', marginLeft: 10
}}>
              Delete
            </Button>
          </Popconfirm>
        </div>
      </div>
      {/* <span className="VerCardReviews">Based on 21 reviews</span> */}
      <p className="VerCard_Description">
        {props?.address}, {props?.city}
      </p>
      <div>{props?.restaurantNote}</div>
      <div>Max Capacity: {props.maxCapacity}</div>
    </div>
    <EditRestaurant
      isModalOpen={isModalOpen}
      handleOk={handleOk}
      handleCancel={handleCancel}
      resId={resId}
```

```
        setShowToast={props?.setShowToast}
      />
    </div>
  )
}

export default VerticalCard

// edit restaurant
import React, { useEffect, useState } from 'react'
import { Button, Select, Form, Input, Modal, message, Upload } from 'antd'
// import { InboxOutlined, PlusOutlined } from '@ant-design/icons'
import { useQuery, useQueryClient } from 'react-query'
import { useMutation } from 'react-query'
import { getAllCuisines, getAllExtraServices } from '../../query/searchFilters'
import { updateRestaurant, restaurantDetailsById } from '../../query/restaurants'
import { CToast, CToastBody, CToastClose } from '@coreui/react'
import { useParams, Link } from 'react-router-dom'

// const { Dragger } = Upload
const EditRestaurant = ({ handleCancel, handleOk, isModalOpen, resId, setShowToast })
=> {
 const queryClient = useQueryClient()
 const [form] = Form.useForm()
 const updateRestaurantMutation = useMutation((data) => updateRestaurant(resId, data))

 const { data: allCuisines } = useQuery('all-cuisines', getAllCuisines)
 const { data: resDetail } = useQuery(
   ['restaurant-details', resId],
   () => restaurantDetailsById(resId),
   { enabled: !!resId },
 )
 // const { data: allExtraServices } = useQuery('all-extra-service',
getAllExtraServices)

 useEffect(() => {
   const resData = resDetail?.data?.[0]
   if (resData) {
     form.setFieldsValue({
       ...resData,
     })
   }
```

```
}, [resDetail, form])

const onFinish = (values) => {
  updateRestaurantMutation.mutate(values, {
    onSuccess: () => {
      queryClient.invalidateQueries(['my-restaurants'])
      setShowToast({
        visible: true,
        message: 'Restaurant updated successfully',
      })
      handleOk()
      setTimeout(() => {
        setShowToast({
          visible: false,
          message: '',
        })
      }, 2000)
    },
    onError: (data) => {
      const errorMessage = data?.response?.data?.msg
      setShowToast({
        visible: true,
        message: errorMessage,
      })
    },
  })
}
const onFinishFailed = (errorInfo) => {
  console.log('Failed:', errorInfo)
}

const cuisineOptions =
  allCuisines?.data?.values?.length > 0
    ? allCuisines?.data?.values.map((data) => ({
        label: data,
        value: data,
      }))
    : []

// const extraServicesOptions =
//   allExtraServices?.data?.values?.length > 0
//     ? allExtraServices?.data?.values.map((data) => ({
```

```
//         label: data,
//         value: data,
//       }))
//     : []

// const props = {
//   name: 'file',
//   multiple: true,
//   action: 'https://www.mocky.io/v2/5cc8019d300000980a055e76',
//   onChange(info) {
//     const { status } = info.file
//     if (status !== 'uploading') {
//       console.log(info.file, info.fileList)
//     }
//     if (status === 'done') {
//       message.success(`${info.file.name} file uploaded successfully.`)
//     } else if (status === 'error') {
//       message.error(`${info.file.name} file upload failed.`)
//     }
//   },
//   onDrop(e) {
//     console.log('Dropped files', e.dataTransfer.files)
//   },
// }
return (
  <div>
    <Modal
      title="Edit restaurants"
      footer={false}
      open={isModalOpen}
      onOk={handleOk}
      onCancel={handleCancel}
    >
      <Form
        name="basic"
        labelCol={{ span: 8 }}
        wrapperCol={{ span: 16 }}
        style={{ maxWidth: 600 }}
        initialValues={{ remember: true }}
        onFinish={onFinish}
        onFinishFailed={onFinishFailed}
        autoComplete="off"
```

```
          form={form}
        >
          {/* <Form.Item label="Images" valuePropName="fileList">
          <Dragger {...props}>
            <p className="ant-upload-drag-icon">
              <InboxOutlined />
            </p>
            <p className="ant-upload-text">Click or drag file to this area to
upload</p>
            <p className="ant-upload-hint">Upload Restaurant Files</p>
          </Dragger>
        </Form.Item> */}
          <Form.Item
            label="Name"
            name="name"
            rules={[{ required: true, message: 'Please enter your restaurant name!' }]}
          >
            <Input />
          </Form.Item>
          <Form.Item
            label="Max Capacity"
            name="maxCapacity"
            rules={[{ required: true, message: 'Please enter restaurant table count!'
}]}
          >
            <Input type="number" />
          </Form.Item>
          <Form.Item
            label="Number of tables"
            name="maxTables"
            rules={[{ required: true, message: 'Please enter restaurant table count!'
}]}
          >
            <Input type="number" />
          </Form.Item>
          <Form.Item
            label="Grace period"
            name="gracePeriod"
            rules={[{ required: true, message: 'Please enter your restaurant grace
period!' }]}
          >
            <Input type="number" />
```

```
        </Form.Item>
        <Form.Item
          label="Cuisines"
          name="cuisine"
          rules={[{ required: true, message: 'Please select the cuisines!' }]}
        >
          <Select
            // mode="multiple"
            allowClear
            style={{ width: '100%' }}
            placeholder="Please select the cuisines"
            options={cuisineOptions}
          />
        </Form.Item>
        {/* <Form.Item
        label="Extra services"
        name="extraService"
        rules={[{ required: true, message: 'Please select the cuisines!' }]}
      >
        <Select
          mode="multiple"
          allowClear
          style={{ width: '100%' }}
          placeholder="Please select the cuisines"
          options={extraServicesOptions}
        />
      </Form.Item> */}
        <Form.Item
          label="Reservation Interval"
          name="timeInterval"
          rules={[{ required: true, message: 'Please enter your restaurant booking
interval!' }]}
        >
          <Input type="number" />
        </Form.Item>
        {/* <Form.Item label="Upload" valuePropName="fileList">
        <Upload action="/upload.do" listType="picture-card">
          <div>
            <PlusOutlined />
            <div style={{ marginTop: 8 }}>Menu Picture</div>
          </div>
        </Upload>
```

```
      </Form.Item> */}

        <Form.Item
          label="Address"
          name="address"
          rules={[{ required: true, message: 'Please enter your restaurant address!'
}]}
        >
          <Input />
        </Form.Item>

        {/* <Form.Item
        label="Mobile Number"
        name="number"
        rules={[{ required: true, message: 'Please enter your restaurant contact
number!' }]}
      >
        <Input type="number" />
      </Form.Item> */}

        <Form.Item
          label="City"
          name="city"
          rules={[{ required: true, message: 'Please enter your restaurant city!' }]}
        >
          <Input />
        </Form.Item>

        <Form.Item
          label="Plz"
          name="postalCode"
          rules={[{ required: true, message: 'Please enter your restaurant PLZ!' }]}
        >
          <Input type="number" />
        </Form.Item>

        <Form.Item
          label="Restaurant Note"
          name="restaurantNote"
          rules={[{ required: true, message: 'Please enter your restaurant note!' }]}
        >
          <Input />
```

```
        </Form.Item>

        <Form.Item wrapperCol={{ offset: 8, span: 16 }}>
          <Button type="primary" htmlType="submit">
            Submit
          </Button>
          <Button type="primary" style={{ marginLeft: 10 }} onClick={handleCancel}>
            Cancel
          </Button>
        </Form.Item>
      </Form>
    </Modal>
  </div>
 )
}

export default EditRestaurant
```

23.2 Muhammad Bilal

23.2.1 rewards.js-backend

```
const express = require("express");
const router = express.Router();
const rewardsDB = require("../database/rewards");

router.get("/", async (req, res) => {
  try {
    const { userId } = req.query;
    const data = await rewardsDB.getRewards(userId);
    res.send(data);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

module.exports = router;
```

23.2.1 Review.js- Frontend

```
import React, { useState } from "react";
import "./details.css";
import { Rate, Divider, Button, Form, Input, message } from "antd";
import ReviewComment from "./ReviewComment";
import { useParams } from "react-router-dom";
import { useQuery, useMutation, useQueryClient } from "react-query";
import {
  getRestaurantReviews,
  postReview,
  editReview,
  deleteReview,
} from "../../query/review";
import { isEmpty } from "lodash";

const { TextArea } = Input;
export default function Review({ restaurantDetails }) {
  const [form] = Form.useForm();
  const queryClient = useQueryClient();
  const token = localStorage.getItem("token");
  const userId = localStorage.getItem("id");

  const [selectedReview, setSelectedReview] = useState(null);

  const { id } = useParams();
  const { data: restaurantReviews } = useQuery(
    ["restaurant-Reviews", { id }],
    () => getRestaurantReviews(id)
  );

  const postReviewMutation = useMutation((body) => postReview(id, body));
  const editReviewMutation = useMutation((body) => editReview(id, body));
  const deleteReviewMutation = useMutation((reviewId) =>
    deleteReview(id, reviewId)
  );

  const hasUserRating =
    restaurantReviews?.data?.filter(
      (review) => review.userID === parseInt(userId)
    )?.length > 0
```

```
    ? true
    : false;

const onFinish = (values) => {
  const body = {
    ...values,
    ...(selectedReview?.id && { reviewId: selectedReview?.id }),
  };
  !isEmpty(selectedReview)
    ? editReviewMutation.mutate(body, {
        onSuccess: () => {
          queryClient.invalidateQueries(["restaurant-Reviews", { id }]);
          queryClient.invalidateQueries(["restaurant-listings", { id }]);
          message.success("Restaurant edited successfully");
          form.resetFields();
          setSelectedReview(null);
        },
        onError: (data) => {
          const errorMessage = data?.response?.data?.msg;
          message.error(errorMessage);
        },
      })
    : postReviewMutation.mutate(body, {
        onSuccess: () => {
          queryClient.invalidateQueries(["restaurant-Reviews", { id }]);
          queryClient.invalidateQueries(["restaurant-listings", { id }]);
          message.success("Restaurant reviewed successfully");
          form.resetFields();
        },
        onError: (data) => {
          const errorMessage = data?.response?.data?.msg;
          message.error(errorMessage);
        },
      });
};

const onEditReview = (review) => {
  setSelectedReview(review);
  form.setFieldsValue({
```

```
    rating: review?.rating,
    description: review?.descriptionReview,
  });
};

const onDeleteReview = (review) => {
  setSelectedReview(review);
  deleteReviewMutation.mutate(review?.id, {
    onSuccess: () => {
      queryClient.invalidateQueries(["restaurant-Reviews", { id }]);
      message.success("Restaurant deleted successfully");
      setSelectedReview(null);
    },
    onError: (data) => {
      const errorMessage = data?.response?.data?.msg;
      message.error(errorMessage);
    },
  });
};

return (
  <>
    {token && (!hasUserRating || !isEmpty(selectedReview)) && (
      <>
        <h2 className="RestOverviewHeading" style={{ paddingBottom: 15 }}>
          Write Review
        </h2>
        <Form
          name="basic"
          layout="vertical"
          labelCol={{ span: 4 }}
          wrapperCol={{ span: 18 }}
          style={{ maxWidth: 800 }}
          autoComplete="off"
          onFinish={onFinish}
          form={form}
        >
```

```
        <Form.Item name="rating" label="Rating" wrapperCol={{ span: 5 }}>
          <Rate defaultValue={0} className="RestOverviewRate" />
        </Form.Item>
        <Form.Item label="Review" name="description">
          <TextArea
            rows={4}
            placeholder="Share your experience and help others make better choices!"
          />
        </Form.Item>
        <Form.Item wrapperCol={{ span: 1 }}>
          <Button type="primary" htmlType="submit">
            Submit
          </Button>
        </Form.Item>
      </Form>
      <Divider
        style={{
          borderWidth: 1,
          borderColor: "#C3c9cb",
        }}
      />
    </>
  )}
  {!!restaurantReviews?.data?.length && (
    <>
      <h2 className="RestOverviewHeading">
        What {restaurantReviews?.data?.length} people are saying
      </h2>
      <Divider
        style={{
          borderWidth: 1,
          borderColor: "#C3c9cb",
        }}
      />
    </>
  )}
  <h3 className="RestOverviewHeading">Overall rating and reviews</h3>
  <div className="ReviewTabContainer">
    <div className="ReviewTabContainerLeft">
```

```
        <div className="ReviewTabContainerLeft">
          <p style={{ paddingBottom: 2, fontSize: 16 }}>
            Reviews can only be made by diners who have eaten at this restaurant
          </p>
          <Rate
            allowHalf
            value={parseFloat(restaurantDetails?.rating)}
            className="RestOverviewRate"
            disabled
          />
          <span style={{ paddingLeft: 15, fontSize: 12 }}>
            Based on {restaurantReviews?.data?.length} reviews
          </span>
        </div>
        <div className="ReviewTabContainerRight"></div>
      </div>
      <Divider
        style={{
          borderWidth: 1,
          borderColor: "#C3c9cb",
          marginBottom: 0,
        }}
      />
      {restaurantReviews?.data?.map((review, i) => {
        return (
          <ReviewComment
            review={review}
            onEditReview={onEditReview}
            onDeleteReview={onDeleteReview}
            key={i}
          />
        );
      })}
    </>
  );
}
```

23.2.1 HorzCard.js- Frontend

```
import { Rate } from "antd";
import "./Card.css";

const HorzCard = (props) => {
  const allImages = props?.images?.filter((img) => !img.menuImage) || [];
  return (
    <div className="card">
      <div className="card_body">
        <img
          src={`${process.env.REACT_APP_IMAGE_STORE_URL}/${allImages[0]?.path}`}
          className="card_image"
          onError={(e) => {
            e.target.src = "/images/RestImages/default-restaurant.jpeg";
          }}
        />
        <h2 className="card_title">{props.title}</h2>
        <Rate
          allowHalf
          disabled
          value={parseFloat(props.ratings) || 0}
          className="StarIcon"
        />
        <p className="card_description">{props.description}</p>
      </div>
      <button className="card_btn">Details</button>
    </div>
  );
};

export default HorzCard;
```

23.3 Sanjay George

23.3.1 Chat Frontend

```
/*
** File: /src/Pages/Chat/Chat.js
*/

import React, { useState, useEffect } from "react";
import { io } from "socket.io-client";
```

```
import { Input, Modal, Row, Col, Avatar, Button, List, Skeleton } from "antd";
import Cookies from "universal-cookie";

import styles from "@chatscope/chat-ui-kit-styles/dist/default/styles.min.css";
import {
  MainContainer,
  ChatContainer,
  Sidebar,
  Search,
  Conversation,
  ConversationList,
  ConversationHeader,
  TypingIndicator,
  MessageSeparator,
  MessageList,
  Message,
  MessageInput,
} from "@chatscope/chat-ui-kit-react";
import { useQuery  } from "react-query";
import {
  getListOfConnectedRestaurants,
  getChatMessages,
} from "../../query/chat";

const socketUrl = process.env.REACT_APP_SOCKET_URL || "127.0.0.1:5001";
const socket = io(socketUrl, {
  auth: {
    token: localStorage.getItem("token"),
  },
});

export default function ChatWrapper({ showModal, setShowModal, chatDetails }) {
  const { chatId, userId, restaurantOwnerId, restaurantId, recipientLastName } =
    chatDetails;

  const [chatMessages, setChatMessages] = useState([]);
  const [message, setMessage] = useState("");

  const { data: chatData, isLoading } = useQuery(
    ["chats", { chatId }],
    () => getChatMessages(chatId),
```

```
  {
    enabled: !!chatId,
  }
);

useEffect(() => {
  socket.on("chat-s2c", msg => {
    console.log(msg);

    if (restaurantOwnerId === msg.senderId) {
      setChatMessages(chatMessages => [...chatMessages, msg]);
    }
  });

  return () => {
    socket.removeAllListeners("chat-s2c");
  }
}, [restaurantOwnerId]);

useEffect(() => {
  console.log(chatData?.data);
  if (!chatData || !chatData.data.length) {
    return;
  }
  setChatMessages(chatData.data);
}, [chatData]);

const handleSendMessageClick = () => {
  if (!message.length || restaurantOwnerId <= 0) return;

  const messageObj = {
    // token: userToken,
    message: message,
    chatId: chatId,
    senderId: userId,
    destinationId: restaurantOwnerId,
    timestamp: Date.now(),
  };

  setChatMessages((chatMessages) => [...chatMessages, messageObj]);
```

```
  console.log(messageObj);
  socket.emit("chat-c2s", messageObj);

  setMessage("");
};

return (
  <>
    <Modal
      open={showModal}
      closable={false}
      onOk={handleSendMessageClick}
      onCancel={() => setShowModal(false)}
      okText="Send"
    >
      <div
        style={{
          height: 450
        }}
      >
        <MainContainer responsive>

          <ChatContainer>
            <ConversationHeader>
              <ConversationHeader.Back />
              <ConversationHeader.Content
                userName={recipientLastName}
              />
            </ConversationHeader>
            {chatMessages && chatMessages.length && (
              <MessageList>

                {chatMessages.map((item, index) => {
                  const direction =
                    item.senderId === userId ? "outgoing" : "incoming";
                  const sender =
                    item.senderId === userId ? "You" : recipientLastName;
                  return (
                    <Message
                      key={index}
                      model={{
```

```
                        message: item.message,
                        // sentTime: "15 mins ago",
                        sender: sender,
                        direction: direction,
                        position: "single",
                      }}
                    ></Message>
                  );
                })}
              </MessageList>
            )}

            <MessageInput
              placeholder="Type message here"
              value={message}
              onChange={(val) => setMessage(val)}
              onSend={handleSendMessageClick}
              style={{ textAlign: "left" }}
              attachButton={false}
              sendButton={false}
            />
          </ChatContainer>
        </MainContainer>
      </div>
    </Modal>
  </>
 );
}


/*
** File: /src/query/
*/

import http from "../utils/http";

export function getListOfConnectedRestaurants(userId) {
  return http({
    url: `/chats/connected-users/`,
    method: "get",
  });
```

```
}

export function getChatMessages(chatId) {
    return http({
      url: `/chats/${chatId}/`,
      method: "get",
    });
  }
```

23.3.2 Chat Backend

```
/*
 ** File: ./server.js
 */
const { Server } = require("socket.io");

const io = new Server(5001, {
  cors: {
    origin: true,
  },
});

io.use(handleSocketAuthorization);
io.on("connection", (socket) => {
  socket.on("chat-c2s", async (msg) => {
    try {
      await chatDB.addMessage(msg);
      const recpientSocketId = getSocketId(msg.destinationId);
      console.log({
        ...msg,
        senderSocketId: socket.id,
        recpientSocketId: recpientSocketId,
      });
      io.to(recpientSocketId).emit("chat-s2c", msg);
    } catch (ex) {
      console.error(ex);
    }
  });

  socket.on("disconnect", () => {
```

```
    removeSocketMapping(socket.id);
    console.log(`\nuser disconnected, socketId: ${socket.id}`);
  });
});

/*
 ** File: ./libs/utils/socket.js
 */
const jwt = require("jsonwebtoken");
const connectedUsers = {};
const socketUserMapping = {};

handleSocketAuthorization = (socket, next) => {
  const token = socket.handshake.auth.token;
  const socketId = socket.id;

  if (!token) {
    console.error(
      `SOCKET - Connection with id ${socketId} does not have a valid web token. Not
allowed to connect!`
    );
    return;
  }

  const decodedToken = jwt.verify(token, "" + process.env.ACCESS_TOKEN_SECRET);
  const userId = decodedToken.id;

  if (!userId) {
    console.error("SOCKET - Undefined or invalid user Id");
    return;
  }

  console.log(`a user connected. socketId: ${socketId}, userId: ${userId}`);

  socketUserMapping[socketId] = userId;
  if (!connectedUsers[userId]) {
    connectedUsers[userId] = [socketId];
  } else {
    connectedUsers[userId].push(socketId);
  }
```

```
  next();
};

const getSocketId = (userId) => {
  return connectedUsers[userId];
};

const removeSocketMapping = (socketId) => {
  const userId = socketUserMapping[socketId];
  connectedUsers[userId] = connectedUsers[userId].filter(
    (item) => item !== socketId
  );
};

/*
 ** File: ./libs/controllers/chats.js
 */
const express = require("express");
const auth = require("../utils/auth");
const { validateInt } = require("../utils/inputValidator");
const router = express.Router();
const chatDB = require("../database/chat");
const restaurantDB = require("../database/restaurants");

// GET /api/chats/connected-users/  - Get list of rest owners connected with
router.get("/connected-users/", auth, async (req, res) => {
  try {
    const { userId, role } = req;
    // TODO: check role and call appropriate DB method
    const data = await chatDB.getListOfConnectedRestaurants(userId);
    res.send(data);
  } catch (ex) {
    console.error(ex);
    res.sendStatus(500);
  }
});

// GET /api/chats/v2/connected-users  - Duplicate created temporarily for restaurant
owner chats.
// TODO remove when role is set properly
router.get("/v2/connected-users/", auth, async (req, res) => {
```

```
  try {
    const { userId, role } = req;
    // TODO: check role and call appropriate DB method
    const data = await chatDB.getListOfConnectedUsers(userId);
    res.send(data);
  } catch (ex) {
    console.error(ex);
    res.sendStatus(500);
  }
});

// GET /api/chats/{id}  - Get all chats by chatId
router.get("/:id", async (req, res) => {
  try {
    const { id: chatId } = req.params;
    const data = await chatDB.getMessages(chatId);
    res.send(data);
  } catch (ex) {
    console.error(ex);
    res.sendStatus(500);
  }
});

// GET /api/chats/chat-id/{restaurantId}
router.get("/chat-id/:restaurantId", auth, async (req, res) => {
  try {
    if (!req.role || req.role !== "user" || !req.userId) {
      return res.status(401).send({
        msg: "You are not authorized! Only registered users can chat",
      });
    }

    const { restaurantId } = req.params;
    let data = await chatDB.getChatIdByRestaurantAndUser(
      req.userId,
      restaurantId
    );

    if (data && data.length) {
      return res.send(data);
    }
```

```
    const restaurantDetails = (
      await restaurantDB.getRestaurantDetails(restaurantId)
    )[0];
    await chatDB.createNewChat(req.userId, restaurantDetails.userId);
    data = await chatDB.getChatIdByRestaurantAndUser(req.userId, restaurantId);

    res.send(data);
  } catch (ex) {
    console.error(ex);
    res.sendStatus(500);
  }
});
module.exports = router;

/*
 ** File: ./libs/database/chat.js
 */
const {
  validateInt,
  isStringUndefinedOrEmpty,
} = require("../utils/inputValidator");
const { pool } = require("./config");

const getListOfConnectedRestaurants = (userId) => {
  const query = `select
        c.id as chatId,
        c.user as userId,
        c.restaurantOwner as restaurantOwnerId,
        u.firstName,
        u.lastName
        from chats c
        inner join users u
        on u.id = c.restaurantOwner
        where c.user = ${userId}
        `;

  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
```

```
    });
  });
};

const getListOfConnectedUsers = (ownerId) => {
  const query = `
    select
        c.id as chatId,
        c.user as userId,
        c.restaurantOwner as restaurantOwnerId,
        u.firstName,
        u.lastName,
        count(m.id) as messageCount
        from chats c
        inner join messages m on m.chatid = c.id
        inner join users u on u.id = c.user
        where c.restaurantOwner = ?
        group by userId
        order by chatId;
        `;

  return new Promise((resolve, reject) => {
    pool.query(query, [ownerId], function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const getChatIdByRestaurantAndUser = (userId, restaurantId) => {
  const query = `
        select
        c.id as chatId,
        c.user as userId,
        c.restaurantowner as restaurantOwnerId,
        r.id as restaurantId,
        u.lastName as recipientLastName,
        u.firstName as recipientFirstName
        from chats c
        inner join restaurants r on r.userid = c.restaurantowner
        inner join users u on u.id = c.restaurantowner
```

```
        where r.id = ?
        and c.user = ?;
    `;

  return new Promise((resolve, reject) => {
    pool.query(
      query,
      [restaurantId, userId],
      function (error, results, fields) {
        if (error) reject(error);
        resolve(results);
      }
    );
  });
};

const createNewChat = (userId, restaurantOwnerId) => {
  const sql = `
        INSERT INTO chats
        (user, restaurantowner)
        VALUES (?, ?)`;

  return new Promise((resolve, reject) => {
    pool.query(sql, [userId, restaurantOwnerId], (error, result, fields) => {
      if (error) reject(error);
      resolve(result.insertId);
    });
  });
};

const getMessages = (chatId) => {
  const query = `SELECT
        chatid, senderId, message
        FROM messages
        where chatid = ?
        order by id;
        `;

  return new Promise((resolve, reject) => {
    pool.query(query, [chatId], function (error, results, fields) {
      if (error) reject(error);
```

```
      resolve(results);
    });
  });
};

const addMessage = ({
  chatId,
  senderId,
  destinationId,
  message,
  timestamp,
}) => {
  if (
    !validateInt(chatId) ||
    !validateInt(senderId) ||
    isStringUndefinedOrEmpty(message)
  ) {
    return;
  }

  const sql = `INSERT INTO messages (chatId, senderId, message) VALUES (?, ?, ?)`;

  return new Promise((resolve, reject) => {
    pool.query(sql, [chatId, senderId, message], (error, result, fields) => {
      if (error) reject(error);
      resolve(result?.insertId);
    });
  });
};
module.exports = {
  getListOfConnectedRestaurants,
  getListOfConnectedUsers,
  getChatIdByRestaurantAndUser,
  getMessages,
  addMessage,
  createNewChat,
};
```

23.3.3 Image upload backend

```
/*
 ** File: server.js
 */
app.use("/api/restaurants/:id/images", passOnRestaurantId, imagesRouter);

/*
 ** File: ./libs/controllers/images.js
 */

const express = require("./express");
const router = express.Router();
const fs = require("fs");
const path = require("path");

const imagesDB = require("../database/images");
const auth = require("../utils/auth");
const { IMAGE_FOLDER, IMAGE_PATH, upload } = require("../utils/imageStore");

/*
    GET /restaurants/:id/images             - get all images
    GET /restaurants/:id/images/menu        - get menu images
    POST /restaurants/:id/images            - upload image
    DELETE /restaurants/:id/images/:id      - delete image by id
    PUT /restaurants/:id/images/:id/status  - moderate image status
*/

router.get("/", async (req, res) => {
  try {
    const { restaurantId } = req;
    const { status, count, offset } = req.query;
    const data = (
      await imagesDB.getImages(restaurantId, status, count, offset)
    ).map((item) => {
      return {
        ...item,
        isMenuImage: item.isMenuImage == 1,
      };
    });
    res.send(data);
```

```
  } catch (ex) {
    console.error(ex);
    res.sendStatus(500);
  }
});

router.get("/menu", async (req, res) => {
  try {
    const { restaurantId } = req;
    const { count, offset } = req.query;
    const data = await imagesDB.getMenuImages(restaurantId, count, offset);
    res.send(data);
  } catch (ex) {
    console.error(ex);
    res.sendStatus(500);
  }
});

router.post("/", [auth, upload.single("image")], async (req, res) => {
  try {
    if (!req.role || req.role !== "restaurantOwner" || !req.userId) {
      return res.status(401).send({
        msg: "You are not authorized! Only restaurant owners can add images",
      });
    }

    const { restaurantId, fileName } = req;
    const { isMenuImage } = req.body;

    const result = await imagesDB.addImage(restaurantId, fileName, isMenuImage);

    if (result?.affectedRows > 0) {
      const imageId = result.insertId;

      res.send({
        msg: "Image uploaded",
        imageId: imageId,
        url: `${IMAGE_PATH}/${fileName}`,
      });
    }
  } catch (ex) {
```

```
    console.error(ex);
    res.sendStatus(500);
  }
});

router.delete("/:id", auth, async (req, res) => {
  try {
    if (!req.role || req.role !== "restaurantOwner" || !req.userId) {
      return res.status(401).send({
        msg: "You are not authorized! Only restaurant owners can delete images",
      });
    }
    const { id: imageId } = req.params;

    const data = await imagesDB.getImage(imageId);
    if (!data || !data.length) {
      return res.sendStatus(404);
    }

    const imageDetails = data[0];
    const imagePath = path.join(IMAGE_FOLDER, imageDetails.path);

    fs.unlink(imagePath, async (err) => {
      if (err) {
        console.log(err);
        return res.status(500).send("Error deleting image!");
      } else {
        await imagesDB.deleteImage(imageId);
        return res.send({
          msg: "Image deleted",
        });
      }
    });
  } catch (ex) {
    console.error(ex);
    res.sendStatus(500);
  }
});

router.put("/:id/status", auth, async (req, res) => {
  try {
```

```
    if (!req.role || req.role !== "admin" || !req.userId) {
      return res.status(401).send({
        msg: "You are not authorized! Only admins can moderate images",
      });
    }

    const { id: imageId } = req.params;
    const { status } = req.body;

    await imagesDB.updateImageStatus(imageId, status);

    return res.send({
      msg: "Image moderation updated",
    });
  } catch (ex) {
    console.error(ex);
    res.sendStatus(500);
  }
});

module.exports = router;

/*
 ** File: ./libs/utils/imageStore.js
 */

const multer = require("multer");
const fs = require("fs");
const path = require("path");

const IMAGE_FOLDER = path.resolve(__dirname, "../../uploads/");
const IMAGE_PATH = "/uploads/images";

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, IMAGE_FOLDER);
  },
  filename: function (req, file, cb) {
    const ext = path.extname(file.originalname);
    const fileName = file.fieldname + "-" + Date.now() + ext;
    req.fileName = fileName;
```

```
    cb(null, fileName);
  },
});

const upload = multer({
  storage: storage,
  fileFilter: function (req, file, cb) {
    // Only accept images
    if (!file.originalname.toLowerCase().match(/\.(jpg|jpeg|png|gif)$/)) {
      return cb(new Error("Only image files are allowed!"));
    }
    cb(null, true);
  },
});

module.exports = {
  upload,
  IMAGE_FOLDER,
  IMAGE_PATH,
};

/*
 ** File: ./libs/database/images.js
 */

const { pool } = require("./config");

/* File author: Sanjay George */

const getImages = (
  restaurantId,
  status = "approved",
  count = 20,
  offset = 0
) => {
  const statusCheck = status !== "all" ? `and i.status = "${status}"` : "";

  const query = `
      select i.*, r.name as restaurantName from images i
      inner join restaurants r on i.restaurantid = r.id
      where i.restaurantid = ${restaurantId}
```

```
      ${statusCheck}
      limit ${count}
      offset ${offset};
      `;

  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const getMenuImages = (restaurantId, count = 20, offset = 0) => {
  const query = `
      select i.*, r.name as restaurantName from images i
      inner join restaurants r on i.restaurantid = r.id
      where i.restaurantid = ?
      and i.status = "approved"
      and i.ismenuimage = true
      limit ?
      offset ?;
      `;

  return new Promise((resolve, reject) => {
    pool.query(
      query,
      [restaurantId, count, offset],
      function (error, results, fields) {
        if (error) reject(error);
        resolve(results);
      }
    );
  });
};

const getImage = (imageId) => {
  const query = `
      select * from images
      where id = ?
      `;
```

```
  return new Promise((resolve, reject) => {
    pool.query(query, [imageId], function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const deleteImage = (imageId) => {
  const sql = `
        delete from images where id = ?;
    `;

  return new Promise((resolve, reject) => {
    pool.query(sql, [imageId], (error, result, fields) => {
      if (error) reject(error);
      resolve(result);
    });
  });
};

const addImage = (restaurantId, path, isMenuImage) => {
  const sql = `insert into images (restaurantid, status, path, ismenuimage)
    values (?, "uploaded", ?, ${isMenuImage});`;

  return new Promise((resolve, reject) => {
    pool.query(sql, [restaurantId, path], (error, result, fields) => {
      if (error) reject(error);
      resolve(result);
    });
  });
};

const updateImageStatus = (imageId, status) => {
  if (!status) return;

  const sql = `
      update images
      set status = ?
      where id = ?`;
```

```
  return new Promise((resolve, reject) => {
    pool.query(sql, [status, imageId], (error, result, fields) => {
      if (error) reject(error);
      resolve(result);
    });
  });
};

module.exports = {
  getImages,
  getMenuImages,
  getImage,
  deleteImage,
  addImage,
  updateImageStatus,
};
```

23.4 Omer Zogubi

23.4.1 (ImageUploader)

```
import React, { useState } from 'react'
import axios from 'axios'
import { useQuery, useMutation, useQueryClient } from 'react-query'
import { getImages, getMyRestaurants, deleteImage } from
'../../query/restaurants'
import {
  Typography,
  Form,
  Button,
  Checkbox,
  Space,
  Table,
  Tag,
  Popconfirm,
  Select,
  Input,
```

```
  Divider,
} from 'antd'
import { CToast, CToastBody, CToastClose } from '@coreui/react'
const { Title } = Typography

const ImageUploader = () => {
  const [imageUploadForm] = Form.useForm()

  const [selectedFiles, setSelectedFiles] = useState([])
  const [isMenuImage, setIsMenuImage] = useState(false)
  const queryClient = useQueryClient()

  const [showToast, setShowToast] = useState({
    visible: false,
    message: '',
  })

  const [restaurantId, setRestaurantId] = useState(0)

  const { data: myRestaurantData } = useQuery('my-restaurants', () =>
getMyRestaurants(100))

  // Query to fetch existing images
  const { data: existingImages = [], refetch: refetchExistingImages } = useQuery(
    ['existingImages', restaurantId],
    () => getImages(restaurantId),
  )

  // Mutation to delete an image
  const deleteImageMutation = useMutation(
    ({ imageId, restaurantId }) => deleteImage(imageId, restaurantId),
    {
      onSuccess: () => {
        queryClient.invalidateQueries('existingImages')
        setShowToast({
          visible: true,
          message: 'Image deleted!',
        })
        setTimeout(() => {
```

```
          setShowToast({
            visible: false,
            message: '',
          })
        }, 2000)
      },
    },
  )

  // Mutation to add new images
  const addImagesMutation = useMutation(
    (formData) =>
      axios.post(`/restaurants/${restaurantId}/images`, formData, {
        baseURL: process.env.REACT_APP_BASE_URL,
        headers: {
          'Content-Type': 'multipart/form-data;
boundary=---011000010111000001101001',
          ...(localStorage.getItem('token') && {
            Authorization: 'Bearer ' + localStorage.getItem('token'),
          }),
        },
      }),
    {
      onSuccess: () => {
        queryClient.invalidateQueries('existingImages')
        setSelectedFiles([])
        setIsMenuImage(false)
        imageUploadForm.resetFields()
      },
    },
  )

  const handleFileChange = (event) => {
    setSelectedFiles(event.target.files)
  }

  const handleUploadClick = (event) => {
    // event.preventDefault()
    const formData = new FormData()
```

```
    formData.append('image', selectedFiles[0])
    formData.append('isMenuImage', isMenuImage)
    // for (let i = 0; i < selectedFiles.length; i++) {
    //   formData.append('images', selectedFiles[i])
    // }
    addImagesMutation.mutate(formData)
  }

  const handleDeleteClick = (id) => {
    deleteImageMutation.mutate({ imageId: id, restaurantId: restaurantId })
  }

  const columns = [
    {
      title: 'Restaurant Name',
      dataIndex: 'restaurantName',
      key: 'restaurantName',
      render: (text) => <p>{text}</p>,
    },
    {
      title: 'Image',
      dataIndex: 'image',
      key: 'image',
      render: (image) => (
        <img src={`${process.env.REACT_APP_IMAGE_STORE_URL}/${image}`}
alt={image} height="100" />
      ),
    },
    {
      title: 'IsMenuImage',
      dataIndex: 'isMenuImage',
      key: 'isMenuImage',
      render: (isMenuImage) => <p>{isMenuImage.toString()}</p>,
    },
    {
      title: 'Status',
      key: 'status',
      dataIndex: 'status',
      render: (text) => (
```

```
      <Tag
        color={
          text.toLowerCase() === 'approved'
            ? 'green'
            : text.toLowerCase() === 'rejected'
              ? 'red'
              : 'blue'
        }
      >
        {text}
      </Tag>
    ),
  },
  {
    title: 'Action',
    key: 'action',
    render: (_, record) => (
      <Space size="middle">
        <Popconfirm
          title="Delete image"
          description="Are you sure you want to delete this image?"
          onConfirm={() => handleDeleteClick(record?.id)}
          okText="Yes"
          cancelText="No"
        >
          <Button>Delete</Button>
        </Popconfirm>
      </Space>
    ),
  },
]

const data =
  existingImages?.data?.length > 0
    ? existingImages?.data?.map((item) => ({
      key: item?.id,
      id: item?.id,
      restaurantId: item?.restaurantID,
      restaurantName: item?.restaurantName,
```

```
      isMenuImage: item?.isMenuImage,
      image: item?.path,
      name: 'Test' || item?.name,
      status: item?.status,
    }))
    : []

const restaurantOptions =
  myRestaurantData?.data?.length > 0
    ? myRestaurantData?.data?.map((item) => ({
      label: item.name,
      value: item.id,
    }))
    : []

const handleRestaurantSelection = (id) => {
  setRestaurantId(id)
}

return (
  <div>
    <Form
      name="restaurant-form"
      style={{ maxWidth: 600 }}
      initialValues={{ remember: true }}
      autoComplete="off"
    >
      <Form.Item
        label="Restaurants"
        name="restaurants"
        rules={[{ required: true, message: 'Please select a restaurant!' }]}
      >
        <Select
          // mode="multiple"
          allowClear
          style={{ width: '100%' }}
          placeholder="Select restaurant to upload image"
          options={restaurantOptions}
          onSelect={(value) => handleRestaurantSelection(value)}
```

```
          />
        </Form.Item>
      </Form>

      {restaurantId > 0 && (
        <>
          <Title level={5}>Upload Image</Title>
          <Divider />

          <Form
            name="image-upload-form"
            initialValues={{ remember: true }}
            autoComplete="off"
            onFinish={handleUploadClick}
            form={imageUploadForm}
          >
            <Space>
              <Form.Item label="" name="file" rules={[{ required: true, message:
'Choose file' }]}>
                <Input type="file" onChange={handleFileChange} />
              </Form.Item>

              <Form.Item label="" name="file">
                <Checkbox
                  onChange={(e) => {
                    setIsMenuImage(e.target.checked)
                  }}
                >
                  Mark as a menu image?{' '}
                </Checkbox>
              </Form.Item>

              <Form.Item label="" name="file">
                <Button type="primary" htmlType="submit">
                  Upload
                </Button>
              </Form.Item>
            </Space>
          </Form>
```

```
          <Title level={5}>Uploaded Images</Title>
          <Divider />

          <div>
            <CToast autohide={false} visible={showToast?.visible}
className="align-items-right">
              <div className="d-flex">
                <CToastBody>{showToast?.message}</CToastBody>
                <CToastClose className="me-2 m-auto" />
              </div>
            </CToast>
            <Table columns={columns} dataSource={data} />
          </div>
        </>
      )}
    </div>
  )
}

export default ImageUploader
```

23.4.2 (User Controller and database)

```
const express = require("express");
const router = express.Router();
const userDB = require("../database/users");
const jwt = require("jsonwebtoken");
const auth = require("../utils/auth");
const bcrypt = require("bcryptjs");
const { hochshuleEmailValidation } = require("../utils/inputValidator");

// Roles:
// admin => 1
// user => 2
// restaurantOwner => 3
```

```
router.post("/register", async (req, res) => {
  try {
    const email = req.body.email;
    const password = req.body.password;
    const firstName = req.body.firstName;
    const lastName = req.body.lastName;
    const phonenumber = req.body.phonenumber;
    const hashedPassword = await bcrypt.hash(password, 8);

    if (hochshuleEmailValidation(email)) {
      const data = await userDB.putUser(
        email,
        hashedPassword,
        firstName,
        lastName,
        phonenumber
      );
      if (data) {
        delete req.body.password;
        return res.send({ msg: "Registration Successful", data: req.body });
      } else {
        return res.send({ msg: "Can not register at the moment" });
      }
    } else {
      res.status(400).send({
        msg: "You only can register using Hochschule Fulda email address",
      });
    }
  } catch (ex) {
    if (ex) {
      return res.send({ msg: ex });
    }
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.post("/login", async (req, res) => {
  try {
```

```
    const email = req.body.email;
    const password = req.body.password;
    const data = await userDB.login(email);
    if (data.length > 0) {
      const validPass = await bcrypt.compare(password, data[0].password);
      if (validPass) {
        const accessToken = jwt.sign(
          data[0],
          "" + process.env.ACCESS_TOKEN_SECRET
        );
        delete data[0].password;
        return res.send({
          msg: "Login successful",
          data: data[0],
          accessToken: accessToken,
        });
      } else {
        res.status(500).send({ msg: "Wrong password" });
      }
    } else {
      res.status(500).send({ msg: "email or password is in incorrect" });
    }
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.post("/assign-role", auth, async (req, res) => {
  try {
    if (req.role === "admin") {
      const roleId = req.body.roleId;
      const email = req.body.email;
      const data = await userDB.assignRole(email, roleId);
      if (data) {
        return res.send({ msg: "User role updated successfully" });
      } else {
        return res.send({ msg: "Can not update users role" });
      }
```

```
    } else {
      res.status(401).send({ msg: "You are not authorized" });
    }
  } catch (ex) {
    if (ex) {
      return res.send({ msg: ex });
    }
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.get("/my-profile", auth, async (req, res) => {
  try {
    const { userId } = req;
    const data = await userDB.getMyProfile(userId);
    let userData = data[0];
    delete userData?.password;
    return res.send(userData);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

module.exports = router;
```

```
const { pool } = require("./config");

const putUser = (email, password, firstName, lastName, phonenumber) => {
  const emailQuery = `Select email FROM users WHERE email = '${email}'`;
  const query = `INSERT INTO users (email, password, firstName, lastName,
phonenumber, roleId) VALUES ('${email}',
'${password}','${firstName}','${lastName}','${phonenumber}' , 1)`;

  return new Promise((resolve, reject) => {
```

```
    pool.query(emailQuery, function (error, results) {
      if (error) reject(error);
      try {
        if (results.length > 0) {
          reject("User already exists");
        } else {
          pool.query(query, function (error, results) {
            if (error) reject(error);
            try {
              resolve(results);
            } catch (ex) {
              console.error(ex);
            }
          });
        }
      } catch (ex) {
        console.error(ex);
      }
    });
  });
};
const login = (email) => {
  const query = `SELECT us.id, us.firstName, us.lastName, us.phonenumber,
us.email, us.password, r.name as role FROM users us INNER JOIN roles as r ON
us.roleId = r.id WHERE email ='${email}'`;
  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results) {
      if (error) reject(error);
      try {
        resolve(results);
      } catch (ex) {
        console.error(ex);
      }
    });
  });
};

const assignRole = (email, roleId) => {
  const emailQuery = `Select email FROM users WHERE email = '${email}'`;
```

```
  const query = `UPDATE users SET roleId = '${roleId}' WHERE email ='${email}'`;
  return new Promise((resolve, reject) => {
    pool.query(emailQuery, function (error, results) {
      if (error) reject(error);
      try {
        if (results.length == 0) {
          reject("User does not exist");
        } else {
          pool.query(query, function (error, results) {
            if (error) reject(error);
            try {
              resolve(results);
            } catch (ex) {
              console.error(ex);
            }
          });
        }
      } catch (ex) {
        console.error(ex);
      }
    });
  });
};

const getMyProfile = (userId) => {
  const query = `SELECT us.*, r.points as rewardPoints, roles.name as roleName
FROM users us INNER JOIN rewards as r ON r.userId = us.id INNER JOIN roles ON
roles.id = us.roleId WHERE us.id=${userId}`;
  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results) {
      if (error) reject(error);
      try {
        resolve(results);
      } catch (ex) {
        console.error(ex);
      }
    });
  });
};
```

```
module.exports = {
  putUser,
  login,
  assignRole,
  getMyProfile,
};
```

23.4.3 (Restaurants Controller and database)

```
const express = require("express");
const router = express.Router();
const restaurantsDB = require("../database/restaurants");
const {
  validateInt,
  isStringUndefinedOrEmpty,
} = require("../utils/inputValidator");
const auth = require("../utils/auth");

router.get("/trending", async (req, res) => {
  try {
    const { limit } = req.query;
    const data = await restaurantsDB.getTrending(limit);
    return res.send(data);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.get("/all", auth, async (req, res) => {
  try {
    const role = req.role;

    if (role !== "admin") return res.status(400).json("You are not authorized");

    const { limit } = req.query;
```

```
    const data = await restaurantsDB.getAllRestaurants(limit);
    return res.send(data);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.patch("/update-status", auth, async (req, res) => {
  try {
    const role = req.role;
    const { status, restaurantId } = req.body;

    if (role !== "admin") return res.status(400).json("You are not authorized");

    await restaurantsDB.updateRestaurantStatus(status, restaurantId);
    return res.send({ msg: "Status successfully updated" });
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.post("/", auth, async (req, res) => {
  const role = req.role;
  const userId = req.userId;
  if (role !== "restaurantOwner")
    return res
      .status(400)
      .json("You do not have the rights to add a restaurant");

  try {
    const {
      name,
      postalCode,
      address,
      city,
      cuisine,
```

```
    maxCapacity,
    maxTables,
    restaurantNote,
    gracePeriod,
    timeInterval,
  } = req.body;

  if (
    isStringUndefinedOrEmpty(name) ||
    !validateInt(postalCode) ||
    isStringUndefinedOrEmpty(address) ||
    isStringUndefinedOrEmpty(city)
  ) {
    return res.status(400).json("Invalid name, address or postal code");
  }
  if (
    !validateInt(maxCapacity) ||
    !validateInt(maxTables) ||
    isStringUndefinedOrEmpty(cuisine)
  ) {
    return res.status(400).json("Invalid cuisine, maxTable or maxCapacity");
  }
  if (!validateInt(gracePeriod) || !validateInt(timeInterval)) {
    return res
      .status(400)
      .json("Invalid grace period or time interval between bookings");
  }

  const insertId = await restaurantsDB.addRestaurant({
    name,
    postalCode,
    address,
    city,
    cuisine,
    maxCapacity,
    maxTables,
    restaurantNote,
    gracePeriod,
```

```
      timeInterval,
      userId,
    });
    return res.send({ msg: "Added succesfully", url: `api/restaurants` });
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.get("/my-restaurants", auth, async (req, res) => {
  try {
    const role = req.role;
    const userId = req.userId;

    if (role !== "restaurantOwner")
      return res
        .status(400)
        .json("You do not have the rights to add a restaurant");

    const { limit } = req.query;
    const data = await restaurantsDB.getMyRestaurants(limit, userId);
    return res.send(data);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.put("/update/:id", auth, async (req, res) => {
  const role = req.role;
  const userId = req.userId;
  const { id: restaurantId } = req.params;

  if (role !== "restaurantOwner")
    return res.status(400).json("You are not authorized");
  try {
    const {
```

```
    name,
    postalCode,
    address,
    city,
    cuisine,
    maxCapacity,
    maxTables,
    restaurantNote,
    gracePeriod,
    timeInterval,
  } = req.body;

  if (
    isStringUndefinedOrEmpty(name) ||
    !validateInt(postalCode) ||
    isStringUndefinedOrEmpty(address) ||
    isStringUndefinedOrEmpty(city)
  ) {
    return res.status(400).json("Invalid name, address or postal code");
  }
  if (
    !validateInt(maxCapacity) ||
    !validateInt(maxTables) ||
    isStringUndefinedOrEmpty(cuisine)
  ) {
    return res.status(400).json("Invalid cuisine, maxTable or maxCapacity");
  }
  if (!validateInt(gracePeriod) || !validateInt(timeInterval)) {
    return res
      .status(400)
      .json("Invalid grace period or time interval between bookings");
  }

  const result = await restaurantsDB.updateRestaurant({
    name,
    postalCode,
    address,
    city,
```

```
      cuisine,
      maxCapacity,
      maxTables,
      restaurantNote,
      gracePeriod,
      timeInterval,
      restaurantId,
      userId,
    });

    if (result?.affectedRows > 0) {
      return res.send({ msg: "Updated succesfully", url: `api/restaurants` });
    } else {
      return res.status(401).send({
        msg: "You are not authorized to change",
        url: `api/restaurants`,
      });
    }
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.delete("/:id", auth, async (req, res) => {
  const role = req.role;
  const userId = req.userId;
  const { id: restaurantId } = req.params;

  if (role !== "admin" && role !== "restaurantOwner")
    return res
      .status(400)
      .json("You do not have the rights to delete a restaurant");
  try {
    const result = await restaurantsDB.deleteRestaurant(
      userId,
      restaurantId,
      role
```

```
    );

    if (result?.affectedRows > 0) {
      return res.send({ msg: "Delete succesfull", url: `api/restaurants` });
    } else {
      return res.status(401).send({
        msg: "You are not authorized to delete this restaurant",
        url: `api/restaurants`,
      });
    }
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

// GET /api/restaurants/{id}
router.get("/:id", async (req, res) => {
  const { id } = req.params;
  if (!validateInt(id)) {
    return res.sendStatus(400);
  }
  try {
    const data = await restaurantsDB.getRestaurantDetails(id);
    if (!data || !data.length) {
      return res.sendStatus(404);
    }
    return res.send(data);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.get("/all/public", async (req, res) => {
  try {
    const { limit } = req.query;
    const data = await restaurantsDB.getAllApprovedRestaurants(limit);
```

```
    return res.send(data);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

module.exports = router;
```

```
const { pool } = require("./config");
const dayjs = require("dayjs");

// TODO: Update table and column names once data is set on local
// TODO: remove select *
const getTrending = (limit = 5) => {
  const query = `
        SELECT res.*,
        (
          SELECT JSON_ARRAYAGG(
            JSON_OBJECT(
              'id', img.id,
              'path', img.path,
              'menuImage', img.isMenuImage
            )
          )
          FROM images AS img
          WHERE img.restaurantID = res.id and status='Approved'
        ) AS images,
        avg(re.rating) as rating
        FROM restaurants as res
        LEFT JOIN reviews re
        ON re.restaurantID = res.id
        WHERE res.status='approved'
        GROUP BY res.id
        HAVING avg(re.rating) > 3
        LIMIT ${limit}
        `;
```

```
  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const getRestaurantDetails = (id) => {
  const query = `
        SELECT res.*, avg(re.rating) as rating,
        (
          SELECT JSON_ARRAYAGG(
            JSON_OBJECT(
              'id', img.id,
              'path', img.path,
              'menuImage', img.isMenuImage
            )
          )
          FROM images AS img
          WHERE img.restaurantID = res.id and status='Approved'
        ) AS images
        FROM restaurants res
        LEFT JOIN reviews re
        ON re.restaurantID = res.id
        WHERE res.id = ${id}
        GROUP BY res.id
        `;
  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const getAllRestaurants = (limit = 5) => {
  const query = `
```

```
          SELECT r.*, u.firstName as ownerFirstName, u.lastName as ownerLastName
          FROM restaurants as r
          LEFT JOIN users as u
          ON r.userId = u.id
          LIMIT ${limit}
          `;

  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const updateRestaurantStatus = (status, restaurantId) => {
  const query = `
          UPDATE restaurants SET status='${status}' WHERE id=${restaurantId}
          `;

  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const addRestaurant = ({
  name,
  postalCode,
  address,
  city,
  cuisine,
  maxCapacity,
  maxTables,
  restaurantNote,
  gracePeriod = 0,
  timeInterval = 0,
```

```
  userId,
}) => {
  const sql = `INSERT INTO restaurants
    (name, postalCode, address, city, cuisine, maxCapacity, maxTables,
        restaurantNote, gracePeriod, timeInterval, status, userId)
    VALUES ('${name}', ${postalCode} ,'${address}', '${city}', '${cuisine}',
    ${maxCapacity}, ${maxTables}, '${
    restaurantNote || ""
  }', ${gracePeriod}, ${timeInterval}, 'pending', ${userId})`;

  return new Promise((resolve, reject) => {
    pool.query(sql, (error, result, fields) => {
      if (error) reject(error);
      resolve(result);
    });
  });
};

const getMyRestaurants = (limit = 5, userId) => {
  const query = `
          SELECT
          res.*,
          (
            SELECT JSON_ARRAYAGG(
              JSON_OBJECT(
                'id', img.id,
                'path', img.path,
                'menuImage', img.isMenuImage
              )
            )
            FROM images AS img
            WHERE img.restaurantID = res.id and status='Approved'
          ) AS images
        FROM restaurants AS res
        WHERE res.userId = ${userId}
        LIMIT ${limit}
          `;

  return new Promise((resolve, reject) => {
```

```
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const updateRestaurant = ({
  name,
  postalCode,
  address,
  city,
  cuisine,
  maxCapacity,
  maxTables,
  restaurantNote,
  gracePeriod = 0,
  timeInterval = 0,
  restaurantId,
  userId,
}) => {
  const sql = `UPDATE restaurants SET name = '${name}', postalCode =
${postalCode},
    address = '${address}', city = '${city}', cuisine = '${cuisine}',
    maxCapacity = ${maxCapacity}, maxTables = ${maxTables},
    restaurantNote = '${restaurantNote}', gracePeriod = ${gracePeriod},
timeInterval = ${timeInterval}, status='pending' WHERE id=${restaurantId} AND
userId=${userId}`;

  return new Promise((resolve, reject) => {
    pool.query(sql, (error, result, fields) => {
      if (error) reject(error);

      resolve(result);
    });
  });
};

const deleteRestaurant = (userId, restaurantId, role) => {
```

```
  let sql = `DELETE FROM restaurants
    WHERE id=${restaurantId}`;

  if (role === "restaurantOwner") {
    sql = sql + ` AND userId=${userId}`;
  }

  return new Promise((resolve, reject) => {
    pool.query(sql, (error, result, fields) => {
      if (error) reject(error);
      resolve(result);
    });
  });
};

const getAllApprovedRestaurants = (limit = 5) => {
  const query = `
          SELECT res.*,
          (
            SELECT JSON_ARRAYAGG(
              JSON_OBJECT(
                'id', img.id,
                'path', img.path,
                'menuImage', img.isMenuImage
              )
            )
            FROM images AS img
            WHERE img.restaurantID = res.id and status='Approved'
          ) AS images,
          avg(re.rating) as rating
          FROM restaurants as res
          LEFT JOIN reviews re
          ON re.restaurantID = res.id
          WHERE res.status='approved'
          GROUP BY res.id
          LIMIT ${limit}
          `;

  return new Promise((resolve, reject) => {
```

```
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

module.exports = {
  getTrending,
  addRestaurant,
  getRestaurantDetails,
  updateRestaurant,
  deleteRestaurant,
  getMyRestaurants,
  getAllRestaurants,
  updateRestaurantStatus,
  getAllApprovedRestaurants,
};
```

23.5 Parthiv Jani

23.5.1 File - 1(Frontend)

```
import "./Filter.css";
import { Rate, Radio } from "antd";
import SearchField from "../../Pages/Home/SearchField";
import VerticalCard from "../Cards/VerticalCard";
import { Divider } from "antd";
import { Link, useSearchParams } from "react-router-dom";
import { useQuery } from "react-query";


import { searchRestaurants } from "../../query/restaurant";
import { getAllCuisines, getAllExtraServices } from
"../../query/searchFilters";
```

```
import { isArray } from "lodash";
import { useState } from "react";

const Filter = () => {
 let [searchParams] = useSearchParams();
 const [filterParams, setFilterParams] = useState({
   cuisine: "",
   rating: "",
   extraService: "",
   term: searchParams.get("search"),
 });

 const { data: searchRestaurantsData } = useQuery(
   ["search-restaurants", filterParams],
   () => searchRestaurants(filterParams)
 );

 const { data: allCuisines } = useQuery("all-cuisines", getAllCuisines);
 const { data: allExtraServices } = useQuery(
   "all-extra-service",
   getAllExtraServices
 );

 return (
   <>
     <SearchField />
     <Divider
       style={{
         borderWidth: 1,
         borderColor: "#C3c9cb",
       }}
     />

 <div className="FilterAndSearchResultsContainer">
       <div className="FilterSection">
         <div className="sider">
```

```
          <div className="filter">
            <h3>Cuisine</h3>
            <Radio.Group
              onChange={(e) =>
                setFilterParams({ ...filterParams, cuisine:
e.target.value })
              }
            >
              {allCuisines?.data?.values?.map((service, i) => {
                return (
                  <div className="filteroption" key={i}>
                    <Radio value={service} name="cuisine">
                      {" "}
                      {service}{" "}
                    </Radio>{" "}
                  </div>
                );
              })}
            </Radio.Group>
          </div>
          <div className="filter">
            <h3>Extra services</h3>
            <Radio.Group
              onChange={(e) =>
                setFilterParams({ ...filterParams, cuisine:
e.target.value })
              }
            >
              {allExtraServices?.data?.map((service, i) => {
                return (
                  <div className="filteroption" key={i}>
                    <Radio value={service?.value} name="extraService">

                      {" "}
                      {service?.value}{" "}
                    </Radio>{" "}
```

```
                </div>
              );
            })}
          </Radio.Group>
        </div>
        <div className="filter">
          <h3>Rating</h3>
          <Rate
            allowHalf
            className="RateIcons"
            onChange={(value) =>
              setFilterParams({ ...filterParams, rating: value })
            }
          />
        </div>
      </div>
    </div>
    <div className="SearchResultsSection">
      <h2>Search Results for "{filterParams.term}"</h2>
      <p>{searchRestaurantsData?.data?.length} Restaurants found</p>
      {isArray(searchRestaurantsData?.data)
        ? searchRestaurantsData?.data?.map((data, i) => (
            <Link to={`/details/${data?.id}`} key={i}>
              <VerticalCard
                name={data?.name}
                address={data?.address}
                city={data?.city}
                restaurantNote={data?.restaurantNote}
                maxCapacity={data?.maxCapacity}
                images={data?.images}
                rating={data?.rating}
              />

          </Link>
          ))
        : "No Restaurants Found"}
```

```
      </div>
    </div>
  </>
 );
};

export default Filter;
```

23.5.2 File - 2(Frontend)

```
import React from "react";
import { useQuery } from "react-query";
import { Table, Tag } from "antd";
import { getUserReservations } from "../../query/restaurants";
import dayjs from "dayjs";

const UserReservations = () => {
 const { data: allUserReservations } = useQuery(
   "user-reservations",
   getUserReservations
 );
 const columns = [
   {
     title: "Restaurant Name",
     dataIndex: "restaurantName",
     key: "restaurantName",
   },
   {
     title: "Address",
     dataIndex: "address",

      key: "address",
   },
   {
```

```
    title: "Booked By",
    dataIndex: "bookedBy",
    key: "bookedBy",
  },
  {
    title: "User Email",
    dataIndex: "userEmail",
    key: "userEmail",
  },
  {
    title: "User Phone",
    dataIndex: "userPhone",
    key: "userPhone",
  },
  {
    title: "User Special Request",
    dataIndex: "specialRequest",
    key: "specialRequest",
  },
  {
    title: "Seats Reserved",
    dataIndex: "numberOfSeats",
    key: "numberOfSeats",
  },
  {
    title: "Reserved Date",
    dataIndex: "date",
    key: "date",
    render: (text) => <p>{dayjs(text).format("YYYY-MM-DD")}</p>,
  },
  {
    title: "Reserved Time",

     dataIndex: "time",
    key: "time",
    render: (text) => <p>{text}:00</p>,
```

```
    },
    {
      title: "Status",
      key: "status",
      dataIndex: "status",
      render: (text) => (
        <Tag color={text === "Reserved" ? "green" : "red"}>{text}</Tag>
      ),
    },
  ];
  const data =
    allUserReservations?.data?.length > 0
      ? allUserReservations?.data?.map((data) => ({
          key: data?.id,
          id: data?.id,
          restaurantName: data?.restaurantName,
          status: data?.status,
          address: `${data?.restaurantAddress}, ${data?.restaurantCity}`,
          bookedBy: `${data?.firstName} ${data?.lastName}`,
          specialRequest: data?.specialRequest,
          numberOfSeats: data?.numberOfSeats,
          time: data?.times,
          date: data?.date,
          userEmail: data?.userEmail,
          userPhone: data?.userPhone,
        }))
      : [];
  return (
    <div>
      <Table columns={columns} dataSource={data} />
    </div>
  );
};

export default UserReservations;
```

```
export function getUserReservations() {
 return http({
   url: '/booking/user-reservations',
   method: 'get',
 })
}
```

23.5.3 File - 3(Backend)

```
const express = require("express");
const router = express.Router();
const { pool } = require("./config");

const getReviews = (restaurantId, count = 10, offset = 0) => {
 const query = `
   SELECT re.*, u.lastName, u.firstName, u.email, u.phoneNumber
   FROM reviews re
   inner join users u on re.userid = u.id
   inner join restaurants r on re.restaurantid = r.id
   inner join roles ro on u.roleid = ro.id
   where ro.id = 1
   and r.id = ${restaurantId}
   limit ${count}
   offset ${offset};
   `;

 return new Promise((resolve, reject) => {
   pool.query(query, function (error, results, fields) {
     if (error) reject(error);
     resolve(results);
   });
 });
};
```

```
// GET /restaurants/:id/reviews?count=10&offset=0
router.get("/", async (req, res) => {
 try {
   const { restaurantId } = req;
   const { count, offset } = req.query;
   const data = await getReviews(restaurantId, count, offset);
   return res.send(data);
 } catch (ex) {
   console.error(ex);
   return res.sendStatus(500);
 }
});

module.exports = router;
```

23.6 Kristijan Lazeski

23.6.1 File - 1

```
DROP database if exists reserveat;

CREATE database reserveat;
use reserveat;

drop table if exists users;
CREATE TABLE users (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    lastName varchar(255),
    firstName varchar(255),
    email varchar(255),
    UNIQUE (email),
    roleId int,
    password varchar(255),
    phoneNumber varchar(255)
);
```

```
drop table if exists roles;
CREATE TABLE roles (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    name varchar(255)
);

drop table if exists restaurants;
CREATE TABLE restaurants (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    name varchar(255),
    postalCode int,
    address varchar(255),
    city varchar(255),
    cuisine enum ('Italian', 'Chinese', 'French', 'Thai', 'German',
'Arabic', 'Indian', 'Vegetarian', 'USA', 'Vegan'),
    maxCapacity int,
    maxTables int,
    restaurantNote varchar(255),
    gracePeriod int,
    timeInterval int,
    status enum ('pending', 'approved', 'rejected'),
    userId int
);

drop table if exists reservations;
CREATE TABLE reservations (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    userId int,
    restaurantId int,
    numberOfSeats int,
    status enum('Reserved', 'Modified', 'Cancelled'),
    extraServiceId int,
    lastUpdate date,
    times INT,
    specialRequest varchar(255),
    date date
```

```
);

drop table if exists extraServices;
CREATE TABLE extraServices (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    name varchar(255),
    description varchar(255)
);

drop table if exists chats;
CREATE TABLE chats (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    user int,
    restaurantOwner int
);

drop table if exists messages;
CREATE TABLE messages (
    id int NOT NULL AUTO_INCREMENT,
    chatId int,
    senderId int,
    message varchar(255),
    timeOfSending int,
    PRIMARY KEY (id)
);

alter table messages add column isSeen tinyint after timeofsending;


drop table if exists flags;
CREATE TABLE flags (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    userID int,
    reservationID int,
    descriptionOf varchar(255)
);
```

```
drop table if exists rewards;
CREATE TABLE rewards (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    userID int,
    typeOf enum("ForReservation", "ForUserReview"),
    points int
);

drop table if exists reviews;
CREATE TABLE reviews (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    userID int,
    restaurantID int,
    rating int,
    descriptionReview varchar(255)
);

drop table if exists images;
CREATE TABLE images (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    restaurantID int,
    status enum("Uploaded", "Approved", "Rejected"),
    path varchar(255),
    isMenuImage boolean
);

drop table if exists restaurantTime;
CREATE TABLE restaurantTime (
    id INT(11) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    times INT,
    availability boolean,
    restaurantID INT(11),
    seatsBooked int,
    date date
);
```

```
INSERT INTO roles VALUES(1, 'user');
INSERT INTO roles VALUES(2, 'admin');
INSERT INTO roles VALUES(3, 'restaurantOwner');
INSERT INTO roles VALUES(4, 'waiter');

INSERT INTO users (lastName,firstName,email,roleId,password,phoneNumber)
VALUES

('Skywwalker','Luke','root@hs-fulda.de',1,'$2a$08$LlUi/vmMosiRhs9ABa3D3.SO
..2Nh/.v88PqOyxND/EWVvOxMUQ2e','00425781567328'),

('Taraka','Owner','taraka@hs-fulda.de',3,'$2a$08$LlUi/vmMosiRhs9ABa3D3.SO.
.2Nh/.v88PqOyxND/EWVvOxMUQ2e','004254563267328'),

('Master','Yoda','yoda.eat@hs-fulda.de',1,'$2a$08$LlUi/vmMosiRhs9ABa3D3.SO
..2Nh/.v88PqOyxND/EWVvOxMUQ2e','00124781567328'),

('Fett','Boba','boooba@hs-fulda.de',3,'$2a$08$LlUi/vmMosiRhs9ABa3D3.SO..2N
h/.v88PqOyxND/EWVvOxMUQ2e','078913781567328'),

('Fett','Jango','jangooo@hs-fulda.de',3,'$2a$08$LlUi/vmMosiRhs9ABa3D3.SO..
2Nh/.v88PqOyxND/EWVvOxMUQ2e','004256857567328'),

('Funk','Boba','bobaarrrr@hs-fulda.de',1,'$2a$08$LlUi/vmMosiRhs9ABa3D3.SO.
.2Nh/.v88PqOyxND/EWVvOxMUQ2e','09345781567328'),

('Jinn','Qui-Gon','qupupupupu@hs-fulda.de',1,'$2a$08$LlUi/vmMosiRhs9ABa3D3
.SO..2Nh/.v88PqOyxND/EWVvOxMUQ2e','07351281567328'),

('Kenobi','Ben','ben.kenobi@hs-fulda.de',1,'$2a$08$LlUi/vmMosiRhs9ABa3D3.S
O..2Nh/.v88PqOyxND/EWVvOxMUQ2e','00425781567328'),

('Vader','Darth','red.lightsaber@hs-fulda.de',1,'$2a$08$LlUi/vmMosiRhs9ABa
3D3.SO..2Nh/.v88PqOyxND/EWVvOxMUQ2e','666'),

('Admin','Admin','admin@hs-fulda.de',2,'$2a$08$LlUi/vmMosiRhs9ABa3D3.SO..2
Nh/.v88PqOyxND/EWVvOxMUQ2e','07351281567328'),

('owner','rest','ro@hs-fulda.de',3,'$2a$08$LlUi/vmMosiRhs9ABa3D3.SO..2Nh/.
v88PqOyxND/EWVvOxMUQ2e','07351281567328');
```

```
INSERT INTO restaurants
(name,postalCode,address,city,cuisine,maxCapacity,maxTables,restaurantNote
,gracePeriod,timeInterval,status,userId) VALUES
     ('Taraka',36037,'Leipziger Str
12','Fulda','Chinese',50,12,'Name',2,1200,'approved',2),
     ('Thaitiqi',36037,'Magdeburger Str
45','Fulda','French',20,5,'Name',2,1200,'approved',2),
     ('Steak House',36037,'Amand-Ney-Strasse
17','Fulda','Thai',65,17,'Name',2,1200,'approved',6),
     ('China Town',36037,'Heinrich Str
28','Fulda','German',35,25,'Name',2,1200,'pending',12),
     ('Toast Laden',36037,'Buttlar Str
79','Fulda','Arabic',5,5,'Name',2,1200,'approved',5),
     ('Ideal',36037,'Stadtschloss Str
2','Fulda','Indian',40,8,'Name',2,1200,'approved',5),
     ('Cafe Thiele',36037,'Mittel Str
178','Fulda','Vegetarian',80,6,'Name',2,1200,'rejected',6),
     ('Hopfenglueck',36037,'Heinrich Str
28','Fulda','USA',35,20,'Name',2,1200,'pending',6),
     ('Eck',36037,'Bahnhoff Str
76','Fulda','Vegan',15,15,'Name',2,1200,'approved',12),
     ('Goeppel',36037,'Leipziger Str
158','Fulda','Italian',46,40,'Name',2,1200,'pending',12);


INSERT INTO extraServices VALUES('001', 'Birthday', 'We Make Cake!');
INSERT INTO extraServices VALUES('002', 'Wedding', 'We Make 2 Cakes!');
INSERT INTO extraServices VALUES('003', 'Funeral', 'You bring your own
Cake!!!');
```

23.6.2 File - 2

```
import SearchField from "./SearchField";
import HorzCard from "../../Components/Cards/HorzCard";
import "../../Components/Cards/Card.css";
import { Divider, Button } from "antd";
import { useQuery } from "react-query";
```

```
import { trendingRestaurants, allRestaurants } from
"../../query/restaurant";
import { isArray } from "lodash";
import { Link } from "react-router-dom";

const Home = () => {
  const { data: trendingRestaurantsData } = useQuery(
    "restaurant-listings",
    trendingRestaurants
  );
  const { data: allRestaurantsData } = useQuery(
    "all-restaurant-listings",
    allRestaurants
  );

  return (
    <div>
      <SearchField />
      <Divider
        style={{
          borderWidth: 1,
          borderColor: "#C3c9cb",
        }}
      />
      <div className="cardHeading">
        <h2>Trending Restaurants</h2>
      </div>
      <div className="cardContainer">
        {isArray(trendingRestaurantsData?.data) &&
          trendingRestaurantsData?.data?.map((restaurant, i) => (
            <Link key={i} to={`/details/${restaurant?.id}`}>
              <HorzCard
                alt="img"
                title={restaurant.name}
                description={restaurant.restaurantNote}

                ratings={restaurant.rating}
                images={restaurant?.images}
              />
            </Link>
```

```
          ))}
      </div>
      <Divider
        style={{
          borderWidth: 1,
          borderColor: "#C3c9cb",
        }}
      />
      <div className="cardHeading">
        <h2>All Restaurants</h2>
      </div>
      <div className="cardContainer">
        {isArray(allRestaurantsData?.data) &&
          allRestaurantsData?.data?.map((restaurant, i) => (
            <Link key={i} to={`/details/${restaurant?.id}`}>
              <HorzCard
                alt="img"
                title={restaurant.name}
                description={restaurant.restaurantNote}
                ratings={restaurant.rating}
                images={restaurant?.images}
              />
            </Link>
          ))}
      </div>
    </div>
  );
};

export default Home;
```

23.6.3 File - 3

```
const express = require("express");
const router = express.Router();
const searchDB = require("../database/search");


// GET /api/search/?term=something&limit=20&cuisine=chinese&city=&rating=5
router.get("/", async (req, res) => {
  try {
```

```
    const { term, limit } = req.query;
    const cuisine = req.query.cuisine;
    const rating = req.query.rating;
    const data = await searchDB.get(term, limit, cuisine, rating);
    return res.send(data);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.get("/filters/cuisines", async (req, res) => {
  try {
    const data = await searchDB.getCuisine();
    const filter = {
      query: "cuisine",
      values: data.map((item) => item.cuisine),
    };
    return res.send(filter);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

router.get("/filterCuisine", async (req, res) => {
  try {
    const data = await searchDB.filterCuisine(req.query.cuisine);
    return res.send(data);
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});


router.get("/filters/extra-services/", async (req, res) => {
  try {
    const data = await searchDB.getExtraServices();
    return res.send(data);
```

```
  } catch (ex) {
    console.error(ex);
    return res.sendStatus(500);
  }
});

module.exports = router;

const { pool } = require("./config");

const get = (term, limit = 5, cuisine, rating) => {
  if (!term || !term.trim().length) return [];
  if (cuisine) {
    cuisine = ` AND res.cuisine = '${cuisine}'`;
  } else cuisine = ``;
  if (rating) {
    rating = ` AND rating = ${rating}`;
  } else rating = ``;

  const query = `SELECT res.*, avg(reviews.rating) as rating,
    (
      SELECT JSON_ARRAYAGG(
        JSON_OBJECT(
          'id', img.id,
          'path', img.path,
          'menuImage', img.isMenuImage
        )
      )
      FROM images AS img
      WHERE img.restaurantID = res.id and status='Approved'
    ) AS images
      FROM restaurants as res
      LEFT JOIN reviews
      ON res.id = reviews.restaurantID


WHERE (res.name LIKE "%${term}%" OR res.address like "%${term}%" OR
res.city like "%${term}%") ${cuisine} ${rating}
      GROUP BY res.id
      ORDER BY avg(reviews.rating) DESC
```

```
      LIMIT ${limit}`;

  //query += ` limit "${limit}"`;

  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results, fields) {
      if (error) reject(error);
      try {
        resolve(results);
      } catch (ex) {
        console.error(ex);
      }
    });
  });
};

const getExtraServices = () => {
  const query = `
        select id, name as value from extraServices
        `;
  return new Promise((resolve, reject) => {
    pool.query(query, function (error, results) {
      if (error) reject(error);
      try {
        resolve(results);
      } catch (ex) {
        console.error(ex);
      }
    });
  });
};

const getCuisine = () => {
  const cuisine = `
        SELECT DISTINCT cuisine

         FROM restaurants;
        `;

  return new Promise((resolve, reject) => {
```

```
    pool.query(cuisine, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};

const filterCuisine = (cuisine) => {
  const filter = `
        SELECT *
        FROM restaurants
        WHERE cuisine = '${cuisine}';
        `;

  return new Promise((resolve, reject) => {
    pool.query(filter, function (error, results, fields) {
      if (error) reject(error);
      resolve(results);
    });
  });
};
module.exports = {
  get,
  getCuisine,
  filterCuisine,
  getExtraServices,
};
```