CS 353 Database Management Systems Project

Implementation Report

Food Ordering and Delivery System

Group 19

Batuhan Özçömlekçi, 21703297

Yusuf Ziya Özgül, 21703158

Musa Ege Ünalan, 21803617

Mustafa Göktan Güdükbay, 21801740

A Brief Description of Food Ordering and Delivery System

Requirements Specification:

In this project, we implemented a food ordering and delivery system. The food ordering and delivery system is an online application for customers to order from registered restaurants. There are three types of users: *customers*, *restaurant owners*, and *delivery personnel*. All users in the database have a unique username and login to the system with their username and password. Users will have additional information such as their name, email, and birthdate. Customers can make orders, review them, and mark restaurants as their favorite restaurants. Customers can review both the restaurant and the delivery personnel. Customers can add more than one address and phone number information to the database and choose a delivery address. Restaurant owners can register their restaurants to the system, add food and beverages for their restaurants, and create menus from them for customers to order. Restaurant owners can specify the category of the food they add. Restaurants can also view reviews for their restaurant and give responses to them. Restaurants have average ratings for the food they serve to create an impression on the customers. Additionally, each restaurant has one address. The system organizes the delivery by randomly assigning the order to an available driver. Delivery personnel can deliver orders for various restaurants, if they accept the delivery, or they may reject the delivery, if they are not available. Customers can take out or add ingredients from the items they choose.

System Description:

YemekKutusu is a Food Delivery System. It provides a way of communication between restaurants and customers. Using their credits, customers are able to purchase any available orderable food bundles which are provided by restaurants. By browsing the restaurants, customers can search for the food they desire and choose the amount of orderable food bundles (i.e., food menus) they would like to purchase.

Users of the application system are separated into four categories which are *customers*, *restaurant owners*, *support staff*, and *delivery staff*. Beyond purchasing food, customers can also send a support ticket to the support staff about their food or other account related issues. Support staff are responsible for answering these questions and issuing a warning to the restaurant is under their initiative. Delivery staff is responsible for delivering the foods to the customers.

The process of food ordering starts by customers ordering food. After this, the restaurant has to approve the order and assign this order to a free delivery person. After the assignment, the delivery person needs to accept or reject the delivery order and they should also need to report the delivery once it is completed. This kind of staged delivery process together with support tickets make the application system more reliable.

We also try to address nonfunctional requirements of (i) *efficiency of operations*, such as retrieving, inserting and deleting information, (ii) *robustness*, by providing appropriate responses for every request, (iii) *usability*, by providing a simple but easy-to-use graphical user interface, (iv) *modifiability*, easily extending the system when the new requirements arise, (v) *scalability*, to meet the large-scale data storage and manipulation, (vi) *maintainability*, by handling the errors appropriately, and (vii) *cost efficiency*, in terms of monetary, labor, and maintenance costs.

ER Diagram

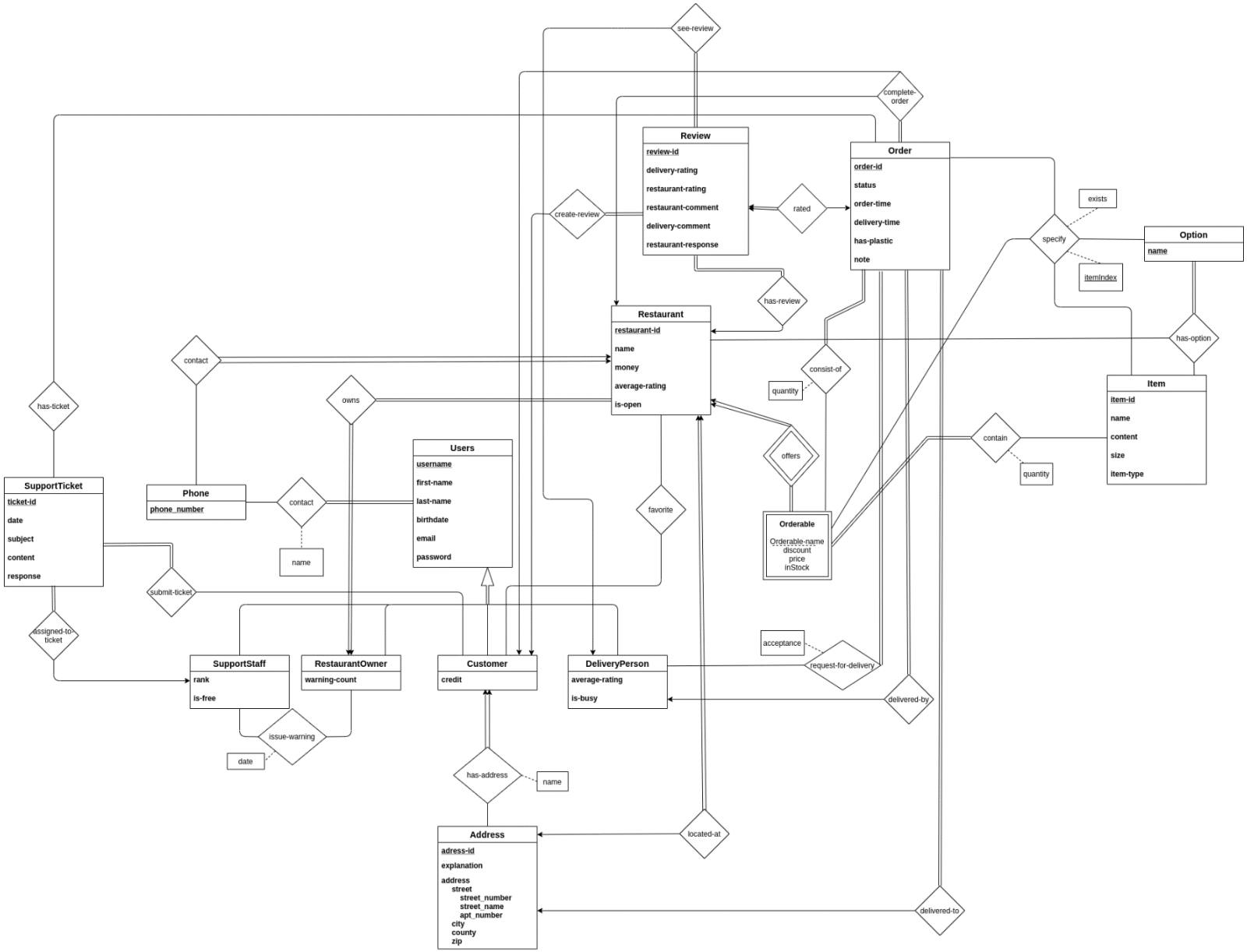
see-review
complete-order
Review
review-id
delivery-rating
restaurant-rating
restaurant-comment
delivery-comment
restaurant-response
Order
order-id
status
order-time
delivery-time
has-plastic
note
create-review
rated
exists
specify
Option
name
itemIndex
has-review
Restaurant
restaurant-id
name
money
average-rating
is-open
has-option
contact
consist-of
quantity
has-ticket
owns
Item
item-id
name
content
size
item-type
contain
quantity
offers
Users
username
first-name
last-name
birthdate
email
password
SupportTicket
ticket-id
date
subject
content
response
Phone
phone_number
contact
name
favorite
Orderable
Orderable-name
discount
price
inStock
submit-ticket
assigned-to-ticket
acceptance
request-for-delivery
SupportStaff
rank
is-free
RestaurantOwner
warning-count
Customer
credit
DeliveryPerson
average-rating
is-busy
delivered-by
issue-warning
date
has-address
name
located-at
Address
adress-id
explanation
address
street
street_number
street_name
apt_number
city
county
zip
delivered-to

Table Schemas

Item(item-id, name content, size, itemtype)

Option(name)

Order(order-id, status, order-time, delivery-time, has-plastic, note)

Review(review-id, delivery-rating, restaurant-rating, restaurant-comment, delivery-comment, restaurant-response, order-id)

- order-id: Foreign Key references Order(order-id)
- order-id is unique (Candidate Key)

Restaurant(restaurant-id, name, money, average-rating, is-open, address-id)

- address-id: Foreign Key references Address(address-id)
- address-id is unique (Candidate Key)

Orderable(restaurant-id, orderable-name, discount, price, instock)

- restaurant-id: Foreign Key references Restaurant(restaurant-id)

Address(address-id, explanation, street, street_number, street_name, apt_number, city, county, zip)

Users(username, first-name, last-name, birthdate, email, password)

DeliveryPerson(username, average-rating, is-busy)

- username: Foreign Key references Users(username)

Customer(username, credit)

- username: Foreign Key references Users(username)

RestaurantOwner(username, warning-count)

- username: Foreign key references Users(username)

SupportStaff(username, rank, is-free)

- username: Foreign Key references Users(username)

SupportTicket(ticket-id, date, subject, content, response)

Phone(phone-number)

Contain(restaurant-id, orderable-name, item-id, quantity)

- (restaurant-id, orderable-name): Foreign Key references Orderable(restaurant-id, orderable-name)
- item-id: Foreign Key references Item(item-id)

HasOption(restaurant-id, option-name, item-id)

- restaurant-id: Foreign Key references Restaurant(restaurant-id)
- option-name: Foreign Key references Option(name)
- item-id: Foreign Key references Item(item-id)

Specify(item-id, option-name, order-id, restaurant-id, orderable-name, item-index, exists)

- item-id: Foreign Key references Item(item-id)
- option-name: Foreign Key references Option(name)
- order-id: Foreign Key references Order(order-id)
- (restaurant-id, orderable-name): Foreign Key references Orderable(restaurant-id, orderable-name)

ConsistOf(order-id, restaurant-id, orderable-name, quantity)

- order-id: Foreign Key references Order(order-id)
- (restaurant-id , orderable-name): Foreign Key references Orderable(restaurant-id, orderable-name)

HasReview(restaurant-id, review-id)

- restaurant-id: Foreign Key references Restaurant(restaurant-id)
- review-id: Foreign Key references Review(review-id)

CompleteOrder(order-id, username, restaurant-id)

- username: Foreign Key references Customer(username)
- restaurant-id: Foreign Key references Restaurant(restaurant-id)
- order-id: Foreign Key references Order(order-id)

SeeReview(review-id, username)

- username: Foreign Key references DeliveryPerson(username)
- review-id: Foreign Key references Review(review-id)

HasAddress(address-id, username, name)

- username: Foreign Key references Customer(username)
- address-id: Foreign Key references Address(address-id)

IssueWarning(support-staff-username, restaurant-owner-username, date)

- support-staff-username: Foreign Key references SupportStaff(username)
- restaurant-owner-username: Foreign Key references RestaurantOwner(username)

AssignedToTicket(ticket-id, username)

- username: Foreign Key references SupportStaff(username)
- ticket-id: Foreign Key references SupportTicket(ticket-id)

SubmitTicket(ticket-id, username)

- username: Foreign Key references Customer(username)
- ticket-id: Foreign Key references SupportTicket(ticket-id)

HasTicket(ticket-id, order-id)

- order-id: Foreign Key references Order(order-id)
- ticket-id: Foreign Key references SupportTicket(ticket-id)

Favorite(username, restaurant-id)

- username: Foreign Key references Customer(username)
- restaurant-id: Foreign Key references Restaurant(restaurant-id)

Contact(username, phone-number, name)

- username: Foreign Key references Customer(username)
- phone-number: Foreign Key references Phone(phone-number)

RestaurantContact(restaurant-id, phone-number)

- restaurant-id: Foreign Key references Restaurant(restaurant-id)
- phone-number: Foreign Key references Phone(phone-number)

Owns(restaurant-id, username)

- username: Foreign Key references RestaurantOwner(username)
- restaurant-id: Foreign Key references Restaurant(restaurant-id)

CreateReview(review-id, username)

- username: Foreign Key references Customer(username)
- review-id: Foreign Key references Review(review-id)

RequestForDelivery(username, order-id, acceptance)

- username: Foreign Key references DeliveryPerson(username)
- order-id: Foreign Key references Order(order-id)

DeliveredBy(order-id, username)

- username: Foreign Key references DeliveryPerson(username)
- order-id: Foreign Key references Order(order-id)

DeliveredTo(order-id, address-id)

- address-id: Foreign Key references Address(address-id)
- order-id: Foreign Key references Order(order-id)

Implementation Details

Implementation part of the project consists of three main parts. These are *database* (SQL), *front-end* (React.js) and *back-end* (Node.js) parts.

We have used PostgreSQL13 as our database management system. We decided it is convenient for us since it is a free SQL based database management system. We initialized 34 distinct tables with our predefined statements. We added a Javascript code to reset and reinitialize the database tables and components when it is necessary.

Secondly, we structure the back-end part of the project with Node.js and Express.js routers. These routers enabled us to write Javascript functions which embody the SQL queries. By calling these functions via routers, we executed SQL queries and returned necessary rows from the database system (get, set, post, use on express).

Finally, we designed our user interface utilizing the *material-ui* package of React Javascript package. This package provided us with some of the prebuilt UI components like, layouts (Grid, Box), Forms, Papers, Buttons, TextField and Typography while creating an interactive environment. We combined these material-ui components with CSS, HTML to finalize UI components.

Throughout the project, we encountered several problems on back-end and front-end sides. Since our preliminary knowledge was only sufficient for creating the database management system via SQL queries, we had to learn React and Node.js technologies from scratch. Therefore, learning and implementing these technologies in our application took most of the time spent for the project. Corollary, the UI components and the connection of UI components to backend functions were lacking some functionalities and UI components weren't providing full support for all back-end functionalities. However, our SQL statements and functions calling them are mostly complete due to our preliminary knowledge. Briefly, the most challenging part was connecting the UI with back-end functionalities.

Individual Contribution

Batuhan Özçömlekçi: Batuhan contributed to the back-end and front-end part of the support staff, queries related to support staff. He also contributed to the design of UI with React in multiple pages.

Yusuf Ziya Özgül: Yusuf Ziya contributed to the implementation of the back-end using Node.js and Express.js. He also contributed to the design of UI with React in multiple pages.

Musa Ege Ünalan: Contributed to the back-end and front-end of the customer, delivery person, and restaurant owner. Primarily responsible from the user interface functionality.

Mustafa Göktan Güdükbay: Contributed to the back-end and front-end of the customer, delivery person, and restaurant owner. Primarily responsible for the database implementation and population. Implemented advanced features such as views, triggers, and serial constraints to ensure total participation constriants.

All members of the team contributed to connecting the components of the application in regularly hosted team meetings. Besides, all members contributed to the part of the reports related to their responsibilities.

Advanced DB Features

We create a view for delivery personnel so that they can see the details of the orders that they are assigned.

```
await client.query('CREATE VIEW delivery_person_order_view as

with Delivery_Person_Orders AS

(SELECT username, Orders.order_id, Rest_Address.address_id as
rest_address_id, Rest_Address.explanation as rest_explanation,
Rest_Address.street as rest_street, Rest_Address.street_number as
rest_street_no, Rest_Address.street_name as rest_street_name,
Rest_Address.apt_number rest_apt_no, Rest_Address.city as rest_city_no,
Rest_Address.county rest_county, Rest_Address.zip rest_zip,
Cust_Addresss.address_id, Cust_Addresss.explanation,
Cust_Addresss.street, Cust_Addresss.street_number,
Cust_Addresss.street_name, Cust_Addresss.apt_number, Cust_Addresss.city,
Cust_Addresss.county, Cust_Addresss.zip

FROM Orders NATURAL JOIN Restaurant NATURAL JOIN DeliveredBy, Address
Rest_Address, Address Cust_Addresss, DeliveredTo delTo

WHERE Rest_Address.address_id = Restaurant.address_id and
Cust_Addresss.address_id = delTo.address_id),

Delivery_Person_Orders_Prices AS

SELECT order_id as oid, sum (price*quantity) as totalPrice

FROM Orders NATURAL JOIN ConsistOf NATURAL JOIN Orderable group
by(order_id) ) select * from Delivery_Person_Orders_Prices d1,
Delivery_Person_Orders d2

where d1.oid = d2.order_id; ');
```

We also implemented the triggers. One trigger assigns a support ticket, which is inserted into the SupportTicket table, to a free support staff, if availability is true, and set the support staff's availability to false. The other trigger waits for a support staff to be available, and when a support staff's availability is updated to true, the trigger assigns the support ticket to thast support staff and set the support staff's availability to false.

```
await client.query(`CREATE OR REPLACE FUNCTION
assigned_ticket_insertion()
        RETURNS TRIGGER
        AS
        $$
            DECLARE support_username VARCHAR;
        BEGIN
        IF EXISTS (SELECT * FROM SupportStaff WHERE is_free = true) THEN
           SELECT username into support_username
           FROM SupportStaff
           WHERE is_free=true LIMIT 1;

           UPDATE SupportStaff
           SET is_free = false, current_ticket_id = NEW.ticket_id
           WHERE username = support_username;

           INSERT INTO AssignedToTicket
                VALUES(NEW.ticket_id, support_username);
        END IF;
        RETURN NEW;
        END;
        $$
        LANGUAGE 'plpgsql';


        CREATE TRIGGER assign_new_ticket
          AFTER INSERT ON
          SupportTicket
          FOR EACH ROW
          EXECUTE PROCEDURE assigned_ticket_insertion();
```

```
CREATE OR REPLACE FUNCTION assign_update()
   RETURNS TRIGGER
   AS
   $$
       DECLARE ticket_id_var INTEGER;

   BEGIN
   IF EXISTS (SELECT * FROM SupportTicket S WHERE S.ticket_id
      NOT IN (select ASA.ticket_id from AssignedToTicket ASA)) THEN
          SELECT S.ticket_id into ticket_id_var FROM SupportTicket S
          WHERE S.ticket_id NOT IN
           (select ASA.ticket_id from AssignedToTicket ASA) LIMIT 1;


       UPDATE SupportStaff
          SET is_free = false, current_ticket_id = ticket_id_var
          WHERE username = NEW.username;
       INSERT INTO AssignedToTicket
          VALUES(ticket_id_var, NEW.username);
   END IF;
   RETURN NEW;
   END;
   $$
   LANGUAGE 'plpgsql';


   CREATE TRIGGER assign_existing_ticket
          AFTER UPDATE OF is_free ON SupportStaff
   FOR EACH ROW
   WHEN (pg_trigger_depth() = 0)
   EXECUTE PROCEDURE assign_update();`);
```

We achieve total participation for the Rate relationship between Review and Order entity sets and Located_at relationship between Restaurant and Address using the serial datatypes implemented using automatically increasing integer datatypes.

```
await client.query(`CREATE TABLE Address (
      address_id SERIAL PRIMARY KEY, explanation VARCHAR, street VARCHAR,
     street_number INTEGER, street_name VARCHAR, apt_number INTEGER,
     city VARCHAR, county VARCHAR, zip VARCHAR);`);


await client.query(`CREATE TABLE Restaurant (
      restaurant_id SERIAL PRIMARY KEY, name VARCHAR, money FLOAT,
       average_rating FLOAT, is_open BOOLEAN, address_id INTEGER UNIQUE,
       FOREIGN KEY(address_id) references Address(address_id));`);

await client.query(`CREATE TABLE Orders ( order_id SERIAL PRIMARY KEY,
       status VARCHAR, order_time TIMESTAMP, time_to_deliver TIMESTAMP,
       delivery_time TIMESTAMP, has_plastic BOOLEAN, note VARCHAR);`);

await client.query(`CREATE TABLE Review ( review_id SERIAL PRIMARY KEY,
       delivery_rating FLOAT, restaurant_rating FLOAT,
       restaurant_comment VARCHAR, delivery_comment VARCHAR,
       restaurant_response VARCHAR, order_id INTEGER UNIQUE,
       FOREIGN KEY (order_id) REFERENCES Orders(order_id));`);
```

The User's Manual

Our system has a graphical user interface (GUI) to perform various actions by different types of users. The user interface has a login page (see Figure 1) where the users can login with their registered username and password stored in the database. They can also sign up if they are not registered to the system previously (see Figure 2).

Once the user successfully logins, the information about the user, such as personal data, including name and surname, birth date, address, phone number, and credit is displayed (see Figure 3). Signin process requires an SQL query to be executed for checking the username and password for the user. Signup process requires an insert statement to be executed for inserting the user to the appropriate table, depending on whether they are a restaurant owner, customer, or a delivery personnel.

Customers can use the search functionality to search for food items that they want to order. The system displays the results with the restaurants that carry these items. The users can also list the restaurants to select a specific restaurant (Figure 4). They can also search a restaurant by name and address and view the menus and food items of the restaurant that they select. They can select a specific menu and see the food items and beverages in that menu, create an order by adding the menu item to their box and see the content of their box. They can complete the order by specifying a delivery address, confirming the order, and making the payment. The users can update their addresses by listing them, inserting a new address or deleting an existing address (see Figure 5). The users can list their orders, check details of their orders, write comments for an order, and see the comments written by other customers. The users can also list the reviews for a restaurant (see Figure 6).

The customers can also issue a support ticket, which is assigned to a support staff for handling. When handled, a response to the support ticket is generated and can be displayed in the Graphical User Interface by the customer (see Figure 7). Various other operations related to support tickets and support staff are appropriately handled by the system.

Figure 8 displays the Restaurant Management (Owner) User Interface. A restaurant manager can insert, delete, or update orderables, new food items, and menus using that interface.

Login Page

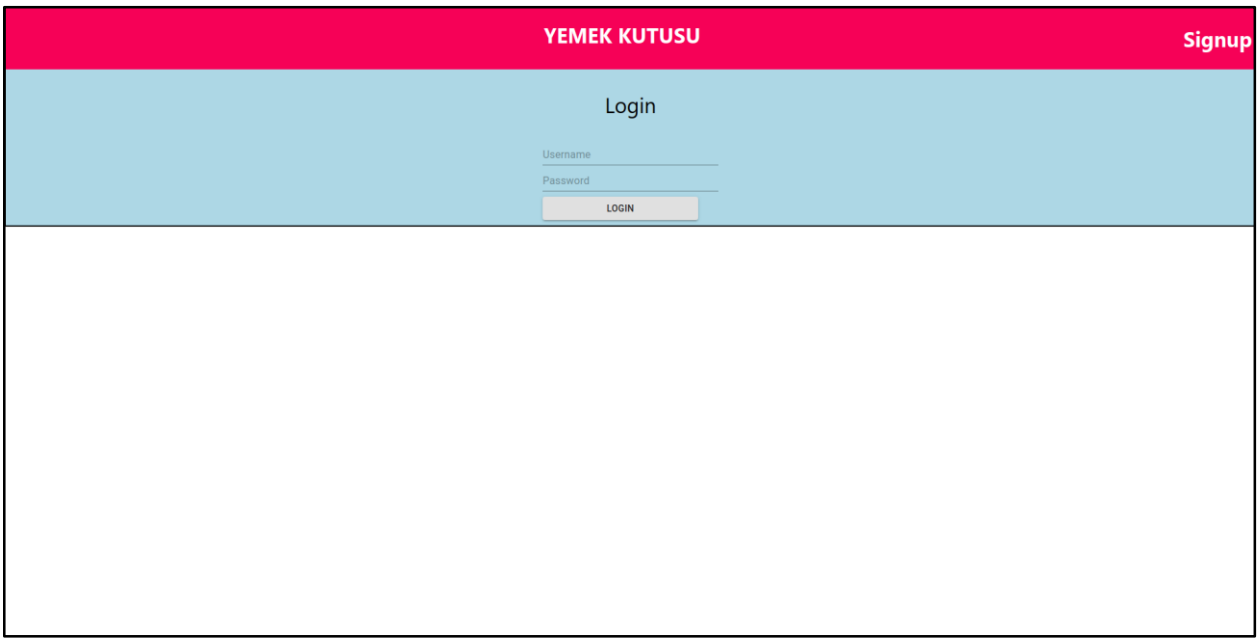

Figure 1. Login Page

Signup Page

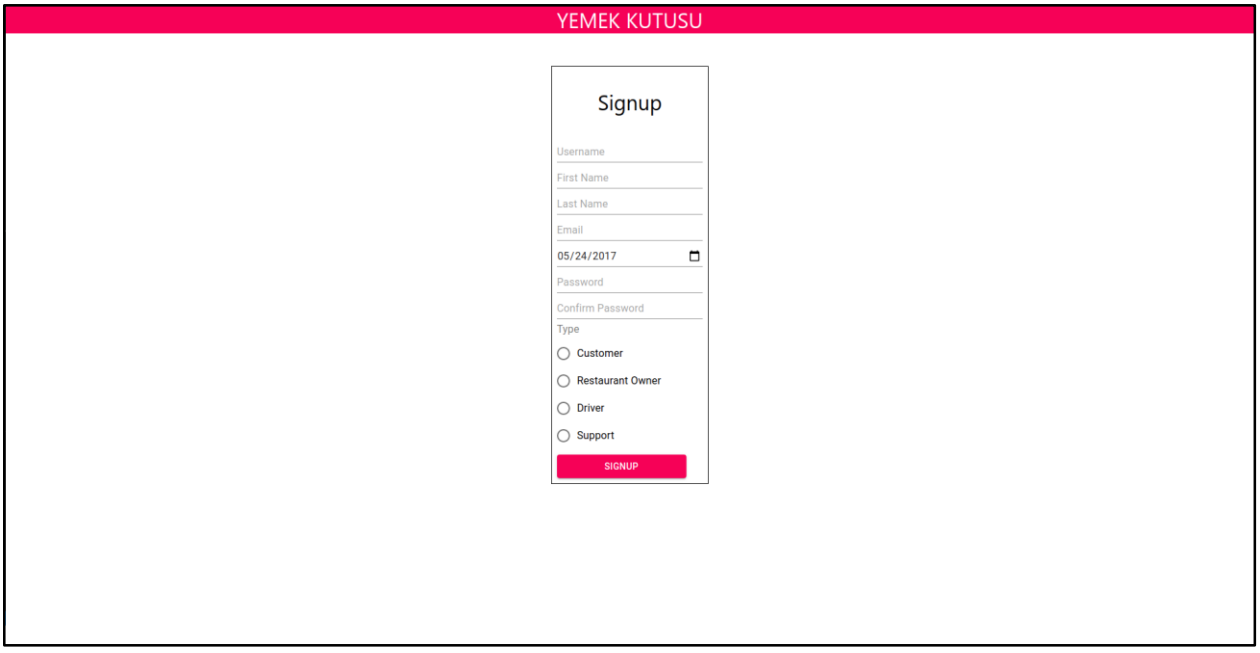

Figure 2. Signup Page

Customer Homepage

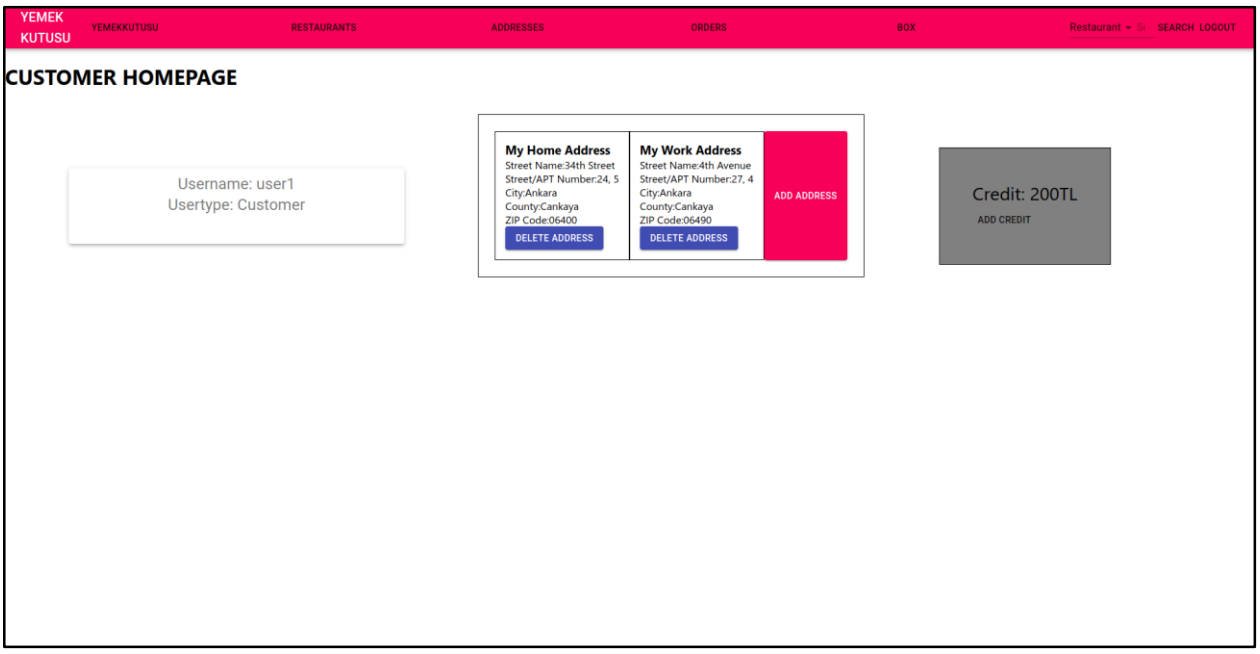

Figure 3. Displaying Customer Information

Restaurants

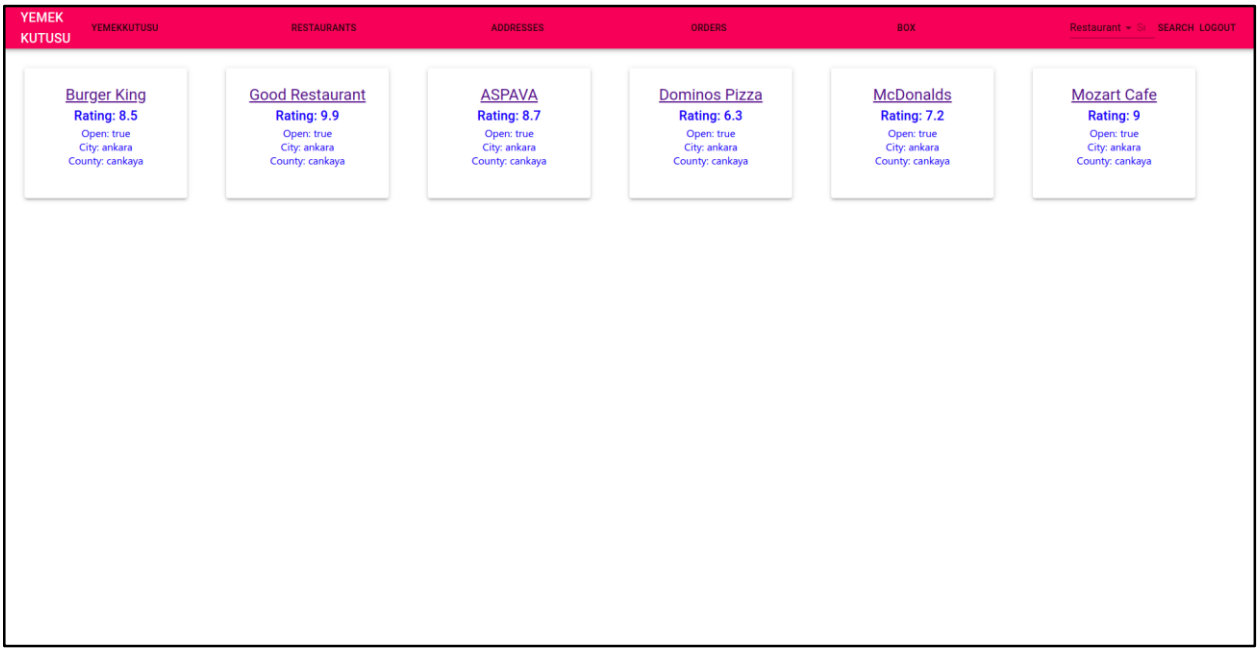

Figure 4. Restaurant Listing

Addresses

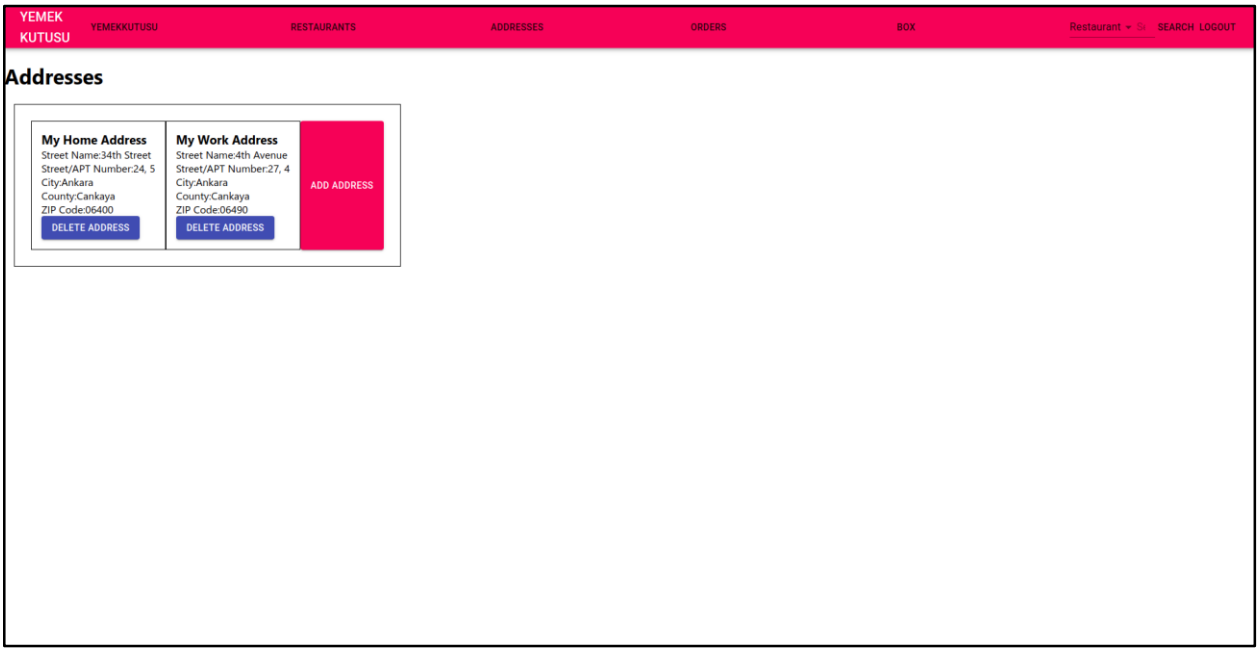

Figure 5. Address Listing for a customer

Orders Screen

Figure 6. Orders Listing for a Customer

Submit Support Ticket for an Order

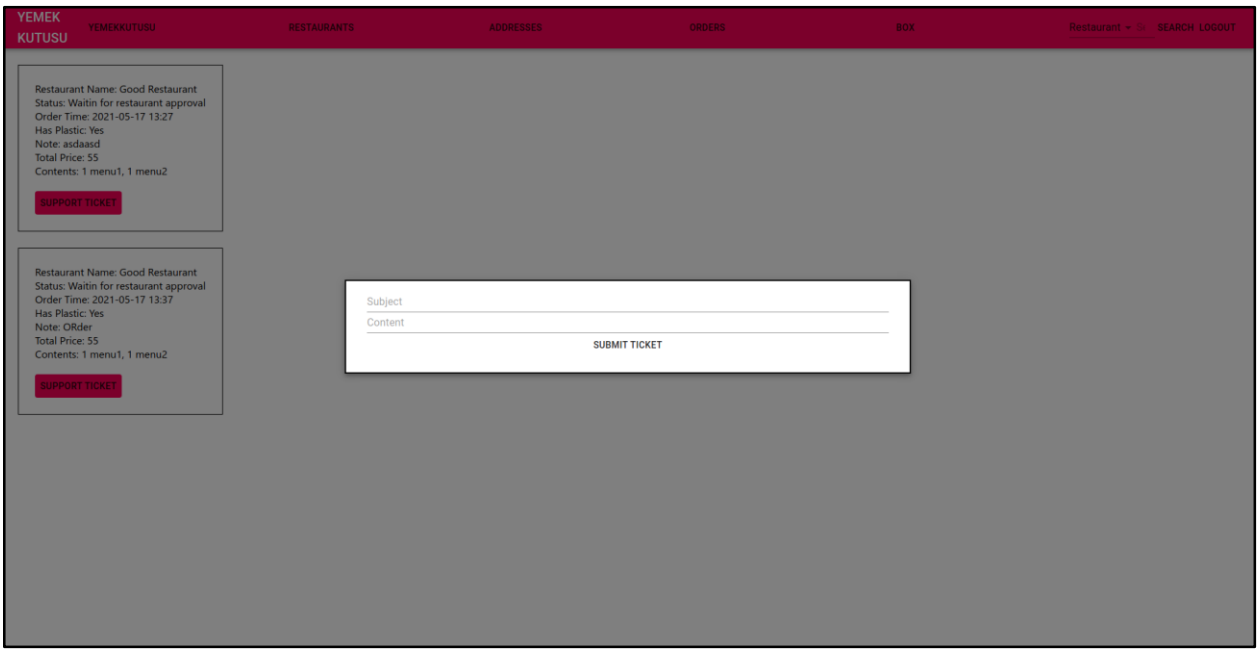

Figure 7. Screen for Submitting a Support Ticket

Restaurant Management (Owner) Screen

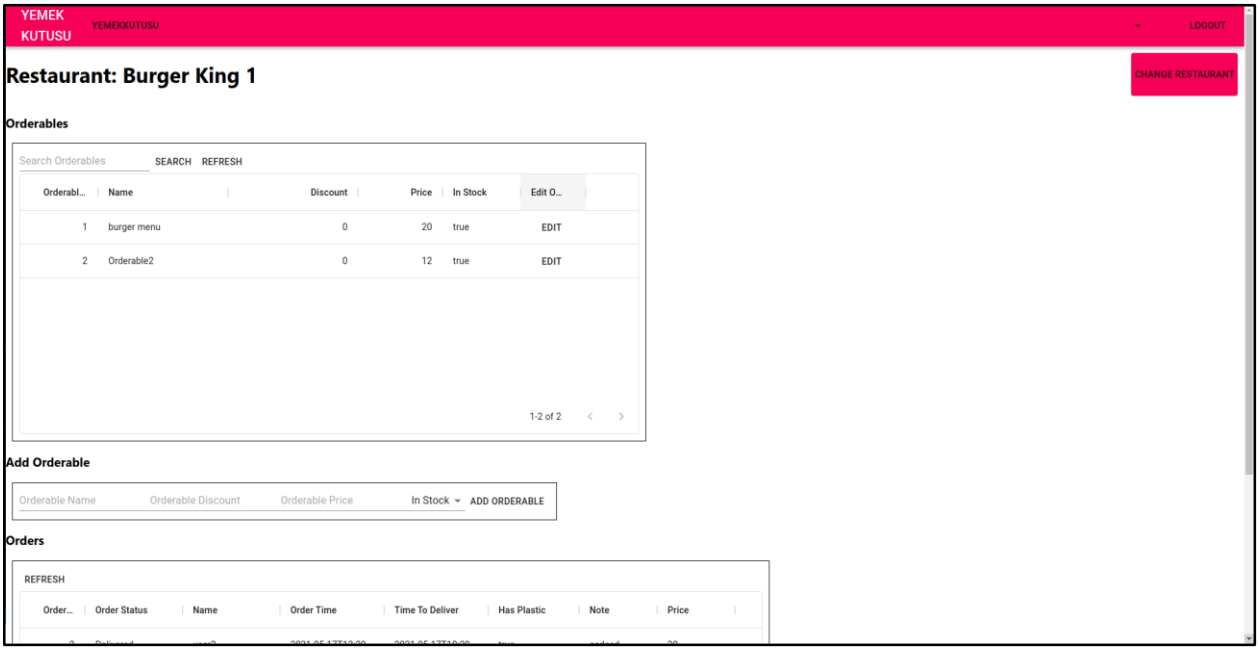

Figure 8. Restaurant Management (Owner) Screen