

THE STATE UNIVERSITY OF ZANZIBAR

SCHOOL OF BUSINESS DEPARTMENT OF COMPUTER SCIENCE, IT AND MASS MEDIA

IT WITH ACCOUNTING

COURSE TITLE: WT822

COURSE NAME: ADVANCED WEB

LECTURE NAME: MR. MASSOUD MMANGA

SEMESTER: SECOND SEMESTER

YEARS: SECOND YEAR

STUDENT NAME: RUKIA ALLY KAIZA

REG.NO: BITA/6/22/068/TZ

ASSIGNMENT: INDIVIDUAL

PROJECT TITLE: ONLINE RESTAURANT SERVICES SYSTEM.

FRONTEND OVERVIEW OF ONLINE RESTAURANT SERVICES SYSTEM.

This part shows the User Interface of my project, which I build using React. where it interacts with the backend services developed in Spring Boot via RESTful APIs. Here is the detailed breakdown of each component and its functionalities:

COMPONENTS.

1. Registration Component.

This component handles new user registration, which uses Axios for HTTP requests to interact with the backend.

-I used features like: Input fields for username, email, and password.

-Validate and submit registration details.

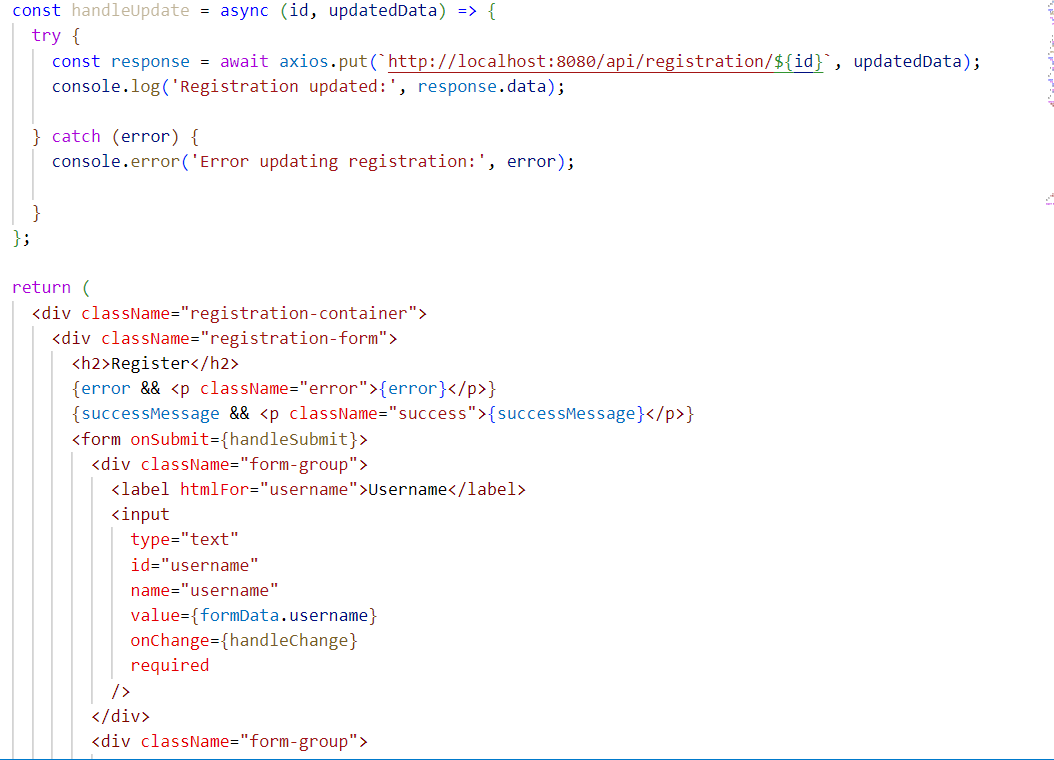
-Display success message and redirect to the Menu page upon successful registration.

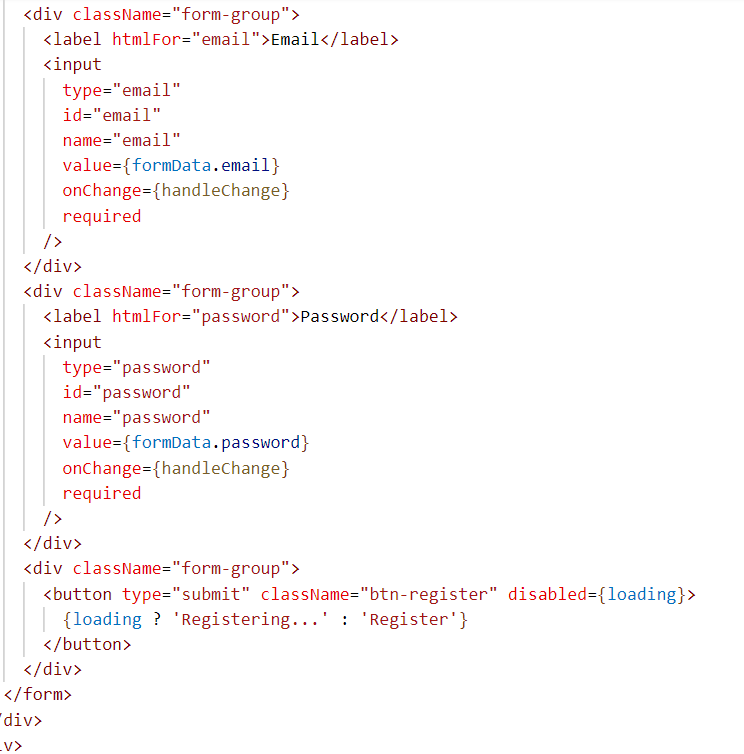
-There are APIs Endpoints of registration and management, where I have used POST, GET, PUT, DELETE Methods for registering new users, retrieving user details, updating user information, and deleting users.

HERE ARE THE CODES.

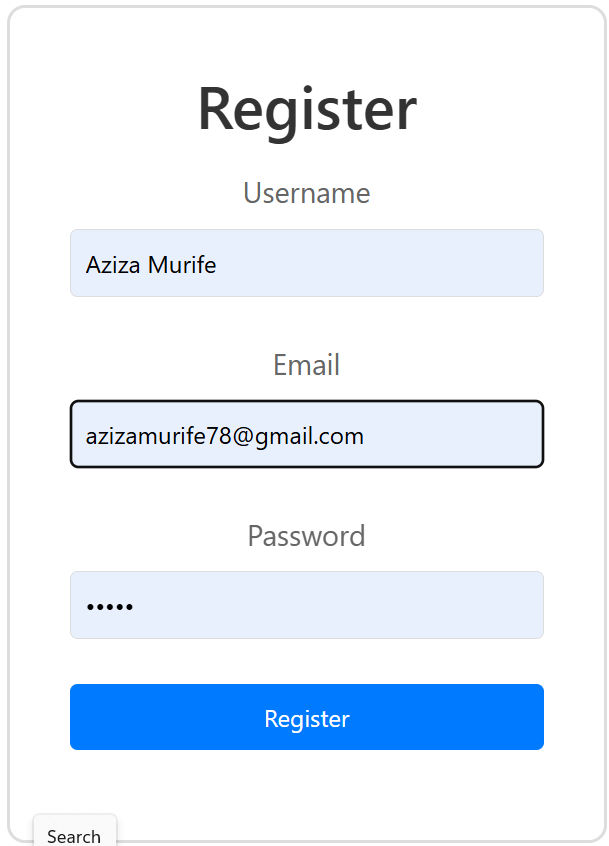








HERE IS THE VIEW.



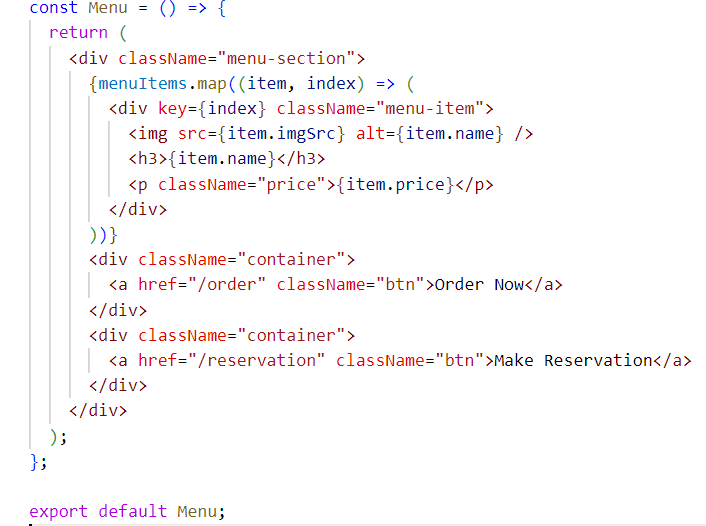
1. Menu component.

This component is serving as the interface through which users can view and select foods for ordering. Here’s a detailed breakdown of its structure, functionalities, and integration with the backend:

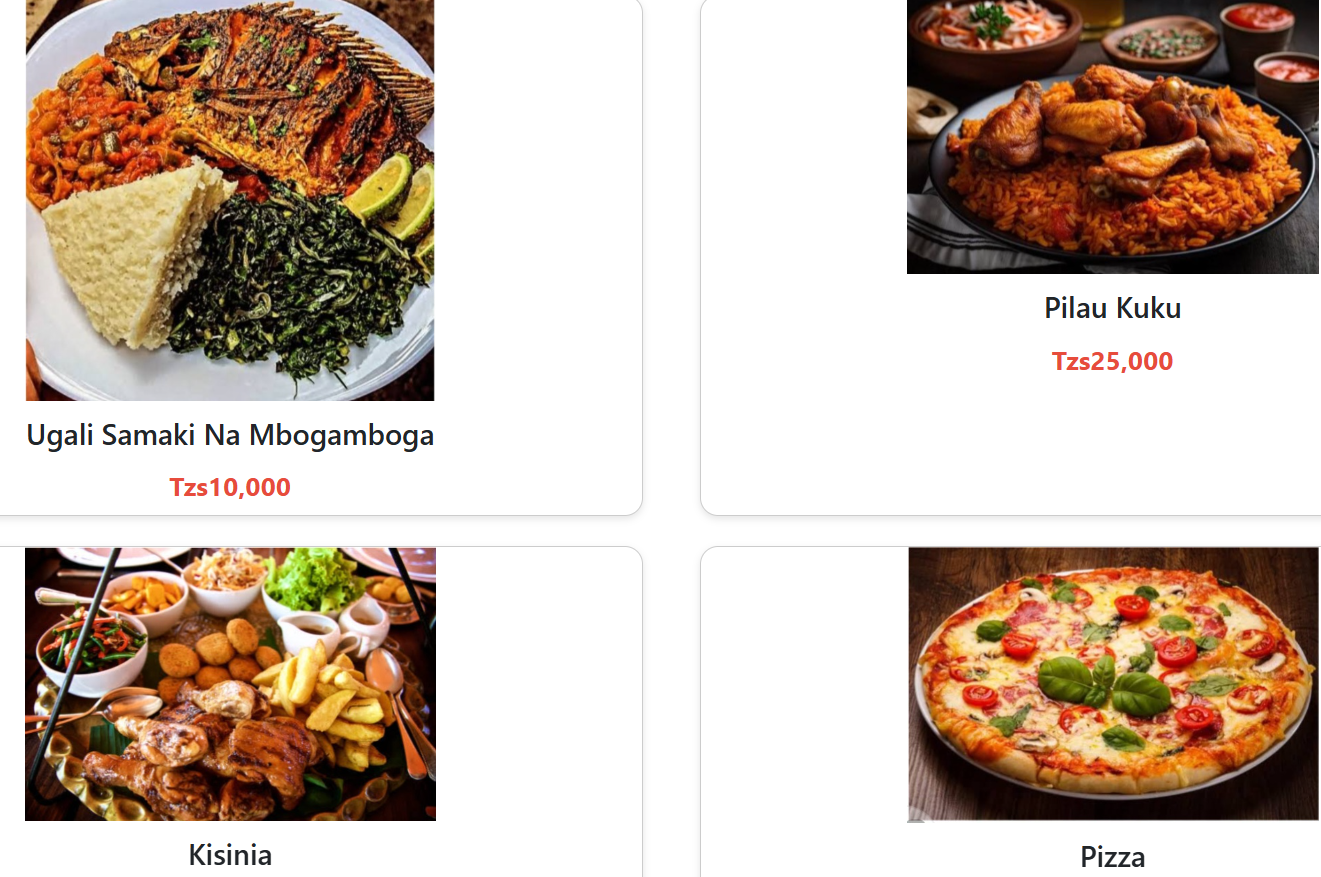
It displays a list of menu items available for ordering. Allow users to just select the food they want to order. And at the end user can navigate to order page.

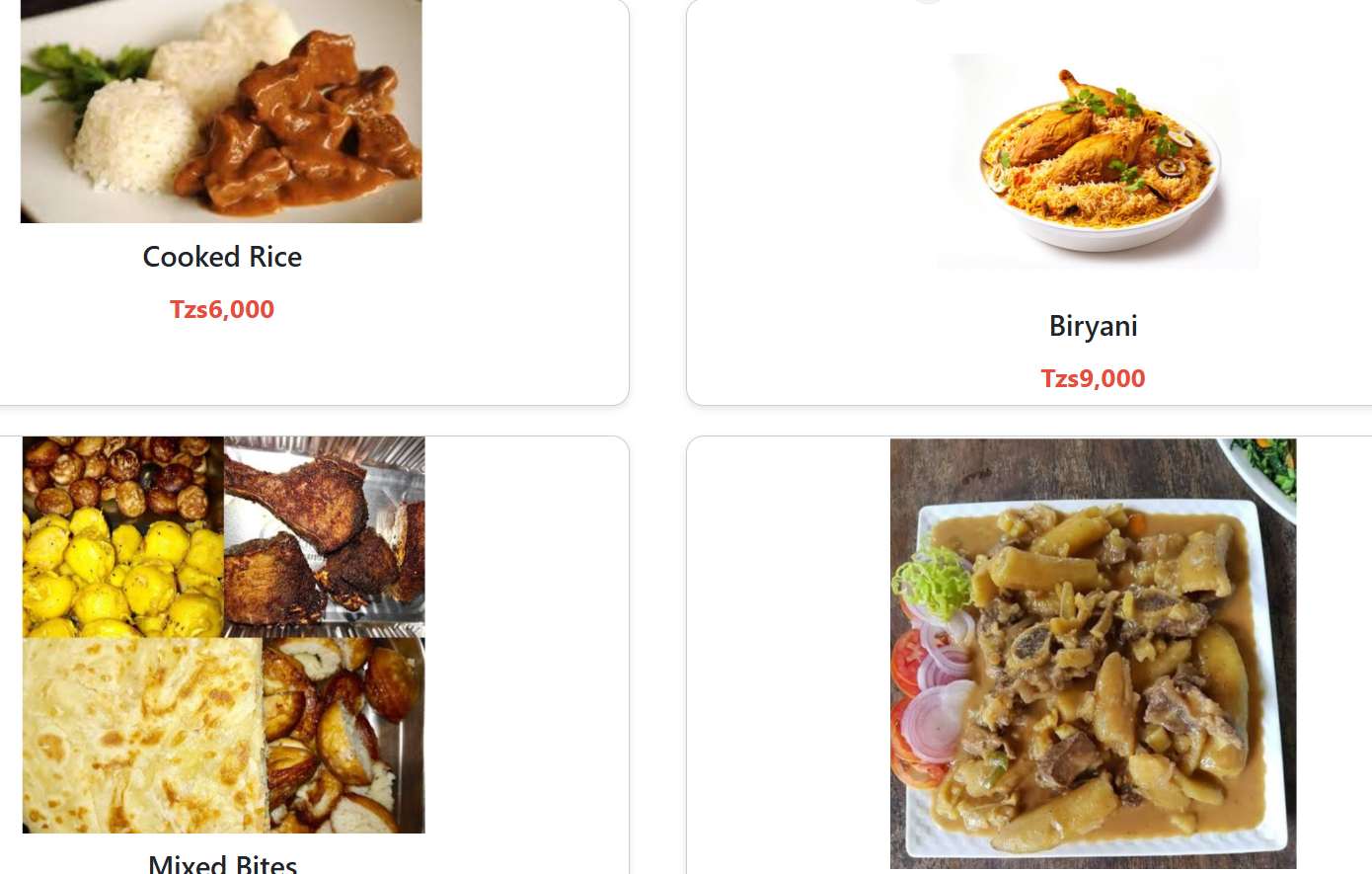
HERE ARE THE CODES.





HERE IS A VIEW.





3.Order Component.

It allows users to place orders. It uses Axios for HTTP requests to interact with the backend, and Send order details to the database. It also include order APIs Endpoints for placing, retrieving, updating, and deleting orders.

HERE ARE THE CODES.

import React, { useState } from 'react';

import axios from 'axios';

import { useNavigate } from 'react-router-dom';

import '../App.css';

const Order = () => {

  const initialFormData = {

    name: '',

    email: '',

    phoneNumber: '',

    address: '',

    orderDetails: '',

  };

  const [formData, setFormData] = useState(initialFormData);

  const [error, setError] = useState(null);

  const navigate = useNavigate();

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({

      ...formData,

      [name]: value,

    });

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    try {

      const response = await axios.post('http://localhost:8080/api/orders', formData);

      console.log('Order placed:', response.data);

      alert('Order placed successfully!');

      setFormData(initialFormData);

      setError(null);

      navigate('/');

    } catch (error) {

      console.error('Error placing order:', error);

      setError('Failed to place order. Please try again.');

    }

  };

  const handleDelete = async (id) => {

    try {

      const response = await axios.delete(`http://localhost:8080/api/orders/${id}`);

      console.log('Order deleted:', response.data);

    } catch (error) {

      console.error('Error deleting order :', error);

    }

  };

  const handleUpdate = async (id, updatedData) => {

    try {

      const response = await axios.put(`http://localhost:8080/api/orders/${id}`, updatedData);

      console.log('Order updated:', response.data);

    } catch (error) {

      console.error('Error updating order:', error);

    }

  };

  return (

    <div className="container">

      <h2>Order Page</h2>

      {error && <p className="error">{error}</p>}

      <form onSubmit={handleSubmit}>

        <div>

          <label htmlFor="name">Name:</label>

          <input

            type="text"

            id="name"

            name="name"

            value={formData.name}

            onChange={handleChange}

            required

          />

        </div>

        <div>

          <label htmlFor="email">Email:</label>

          <input

            type="email"

            id="email"

            name="email"

            value={formData.email}

            onChange={handleChange}

            required

          />

        </div>

        <div>

          <label htmlFor="phoneNumber">Phone Number:</label>

          <input

            type="tel"

            id="phoneNumber"

            name="phoneNumber"

            value={formData.phoneNumber}

            onChange={handleChange}

            required

          />

        </div>

        <div>

          <label htmlFor="address">Address:</label>

          <input

            type="text"

            id="address"

            name="address"

            value={formData.address}

            onChange={handleChange}

            required

          />

        </div>

        <div>

          <label htmlFor="orderDetails">Order Details:</label>

          <textarea

            id="orderDetails"

            name="orderDetails"

            value={formData.orderDetails}

            onChange={handleChange}

            required

          ></textarea>

        </div>

        <button type="submit">Place Order</button>

      </form>

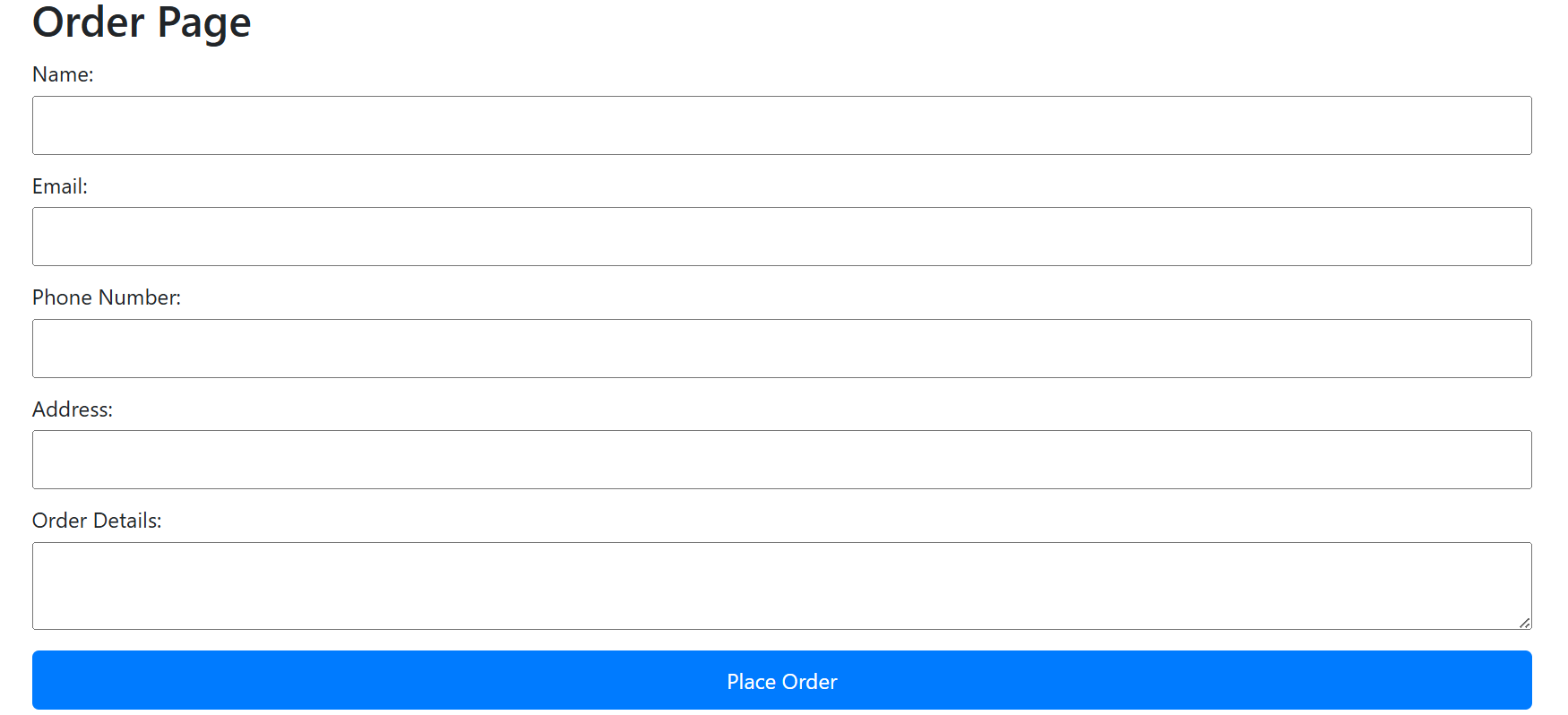
    </div>

  );

};

export default Order;

HERE IS A VIEW.



4.Reservation Component.

It facilitates making table reservations. It uses Axios for HTTP requests to interact with the backend. It has features for selecting date and time for the reservation. Input customer details e.g., name, contact information. Submit the reservation to the backend and display confirmation message. It has Endpoints for making, retrieving, updating, and canceling reservations.

HERE ARE THE CODES.

import React, { useState } from 'react';

import axios from 'axios';

import { useNavigate } from 'react-router-dom';

import '../App.css';

const Reservation = () => {

  const initialFormData = {

    fullName: '',

    email: '',

    phoneNumber: '',

    reservationDate: '',

    reservationTime: '',

    numberOfGuests: '1',

    specialRequests: ''

  };

  const [formData, setFormData] = useState(initialFormData);

  const [loading, setLoading] = useState(false);

  const [successMessage, setSuccessMessage] = useState('');

  const [error, setError] = useState('');

  const navigate = useNavigate();

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({

      ...formData,

      [name]: value,

    });

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    setLoading(true);

    setSuccessMessage('');

    setError('');

    try {

      const response = await axios.post('http://localhost:8080/api/reservations', formData);

      console.log('Reservation submitted:', response.data);

      alert('Reservation Creation successfully!');

      setFormData(initialFormData);

      setError(null);

      navigate('/');

    } catch (error) {

      console.error('Error creating reservation:', error);

      setError('Failed to create reservation. Please try again.');

    }

  };

  const handleDelete = async (id) => {

    try {

      const response = await axios.delete(`http://localhost:8080/api/reservations/${id}`);

      console.log('Reservation deleted:', response.data);

    } catch (error) {

      console.error('Error deleting reservation:', error);

    }

  };

  const handleUpdate = async (id, updatedData) => {

    try {

      const response = await axios.put(`http://localhost:8080/api/reservations/${id}`, updatedData);

      console.log('Reservation updated:', response.data);

    } catch (error) {

      console.error('Error updating reservation:', error);

    }

  };

  return (

    <div className="reservation">

      <h2>Make a Reservation</h2>

      {error && <p className="error">{error}</p>}

      <form onSubmit={handleSubmit}>

        <div className="form-group">

          <label htmlFor="fullName">Full Name</label>

          <input

            type="text"

            id="fullName"

            name="fullName"

            value={formData.fullName}

            onChange={handleChange}

            required

          />

        </div>

        <div className="form-group">

          <label htmlFor="email">Email</label>

          <input

            type="email"

            id="email"

            name="email"

            value={formData.email}

            onChange={handleChange}

            required

          />

        </div>

        <div className="form-group">

          <label htmlFor="phoneNumber">Phone Number</label>

          <input

            type="tel"

            id="phoneNumber"

            name="phoneNumber"

            value={formData.phoneNumber}

            onChange={handleChange}

            required

          />

        </div>

        <div className="form-group">

          <label htmlFor="reservationDate">Reservation Date</label>

          <input

            type="date"

            id="reservationDate"

            name="reservationDate"

            value={formData.reservationDate}

            onChange={handleChange}

            required

          />

        </div>

        <div className="form-group">

          <label htmlFor="reservationTime">Reservation Time</label>

          <input

            type="time"

            id="reservationTime"

            name="reservationTime"

            value={formData.reservationTime}

            onChange={handleChange}

            required

          />

        </div>

        <div className="form-group">

          <label htmlFor="numberOfGuests">Number of Guests</label>

          <select

            id="numberOfGuests"

            name="numberOfGuests"

            value={formData.numberOfGuests}

            onChange={handleChange}

            required

          >

            {[...Array(10).keys()].map((i) => (

              <option key={i + 1} value={i + 1}>

                {i + 1}

              </option>

            ))}

          </select>

        </div>

        <div className="form-group">

          <label htmlFor="specialRequests">Special Requests</label>

          <textarea

            id="specialRequests"

            name="specialRequests"

            rows="4"

            value={formData.specialRequests}

            onChange={handleChange}

          ></textarea>

        </div>

        <div className="form-group">

          <button type="submit">Reserve </button>

        </div>

      </form>

      <div className="container">

        <a href="/" className="btn">

          Back To Homepage

        </a>

      </div>

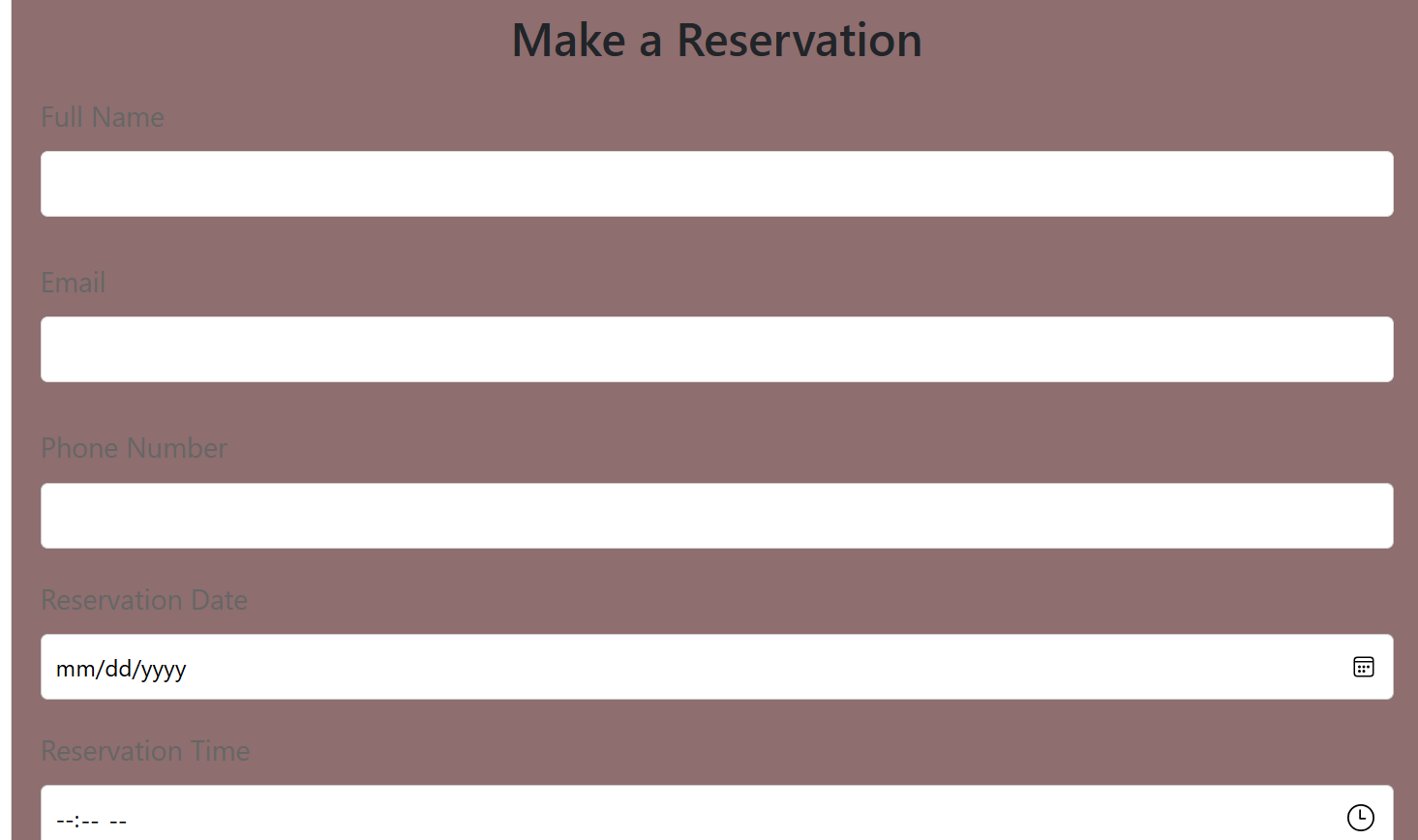
    </div>

  );

};

export default Reservation;

HERE IS A VIEW.



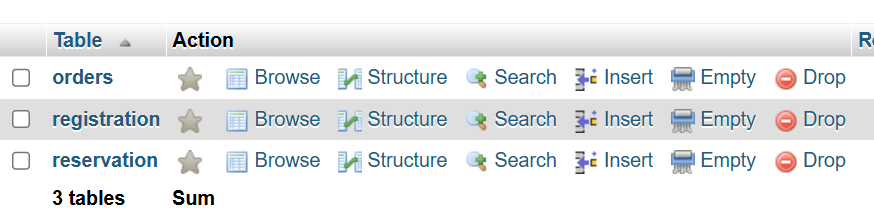
In addition I have used App.css file for styling my frontend overview, and also I created images file for all my applied images in the public folder.

BACKEND OVERVIEW OF THE SYSTEM.

My backend is built with Spring Boot, is structured to manage orders, reservation, and registration . below is the overview.

DATABASE DESIGN.

MY database includes three main tables, Order, reservation, and registration table. Below is the overview.



SPRING BOOT .

Your Spring Boot application serves as the backend, providing RESTful APIs for CRUD operations on these tables. The key components are:

Controllers for managing endpoints, repositories for interface for CRUD operations.

CONTROLLER CODES.

@RestController

@RequestMapping("/api")

@CrossOrigin

public class OrderController {

    private static final Logger logger = LoggerFactory.getLogger(OrderController.class);

    @Autowired

    private OrderRepository orderRepository;

    /\*\*

     \*

     \* @return

     \*/

    @GetMapping("/orders")

    public ResponseEntity<List<Order>> getAllOrders() {

        List<Order> orders = orderRepository.findAll();

        return new ResponseEntity<>(orders, HttpStatus.OK);

    }

    /\*\*

     \*

     \* @param

     \* @return

     \*/

    @PostMapping("/orders")

    public ResponseEntity<String> createOrder(@RequestBody Order order) {

        try {

            Order newOrder = orderRepository.save(order);

            logger.info("Order created: {}", newOrder);

            return new ResponseEntity<>("Order created successfully", HttpStatus.CREATED);

        } catch (Exception e) {

            logger.error("Error creating order", e);

            return new ResponseEntity<>("Error creating order", HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    /\*\*

     \*

     \* @param id

     \* @return

     \*/

    @GetMapping("/orders/{id}")

    public ResponseEntity<Order> getOrderById(@PathVariable Long id) {

        Optional<Order> order = orderRepository.findById(id);

        if (order.isPresent()) {

            return new ResponseEntity<>(order.get(), HttpStatus.OK);

        } else {

            logger.warn("Order with ID {} not found", id);

            return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        }

    }

    /\*\*

     \*

     \* @param id

     \* @param orderDetails

     \* @return

     \*/

    @PutMapping("/orders/{id}")

    public ResponseEntity<Order> updateOrder(@PathVariable Long id, @RequestBody Order orderDetails) {

        Optional<Order> optionalOrder = orderRepository.findById(id);

        if (optionalOrder.isPresent()) {

            Order existingOrder = optionalOrder.get();

            existingOrder.setName(orderDetails.getName());

            existingOrder.setEmail(orderDetails.getEmail());

            existingOrder.setPhoneNumber(orderDetails.getPhoneNumber());

            existingOrder.setAddress(orderDetails.getAddress());

            existingOrder.setOrderDetails(orderDetails.getOrderDetails());

            Order updatedOrder = orderRepository.save(existingOrder);

            logger.info("Order updated: {}", updatedOrder);

            return new ResponseEntity<>(updatedOrder, HttpStatus.OK);

        } else {

            logger.warn("Order with ID {} not found for update", id);

            return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        }

    }

    /\*\*

     \*

     \* @param id

     \* @return

     \*/

    @DeleteMapping("/orders/{id}")

    public ResponseEntity<String> deleteOrder(@PathVariable Long id) {

        if (orderRepository.existsById(id)) {

            orderRepository.deleteById(id);

            logger.info("Order with ID {} deleted", id);

            return new ResponseEntity<>("Order deleted successfully", HttpStatus.OK);

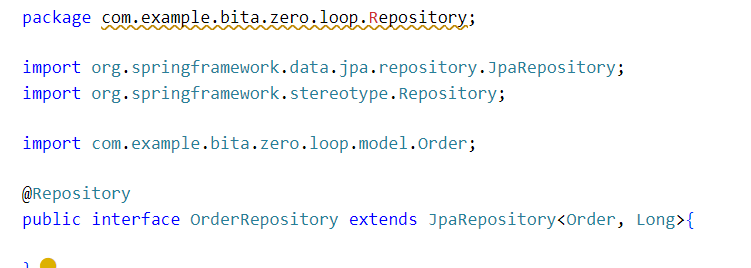
        } else {

            logger.warn("Order with ID {} not found for deletion", id);

            return new ResponseEntity<>("Order not found", HttpStatus.NOT\_FOUND);

        }

REPOSITORY CODES:



API ENDPOINS.

In my backend I used several APIendpoints to interact with the database:

Order API Endpoints,

- POST /orders: Create a new order.

- GET /orders/{id}: Retrieve a specific order by ID.

- PUT /orders/{id}: Update an order by ID.

- DELETE /orders/{id}: Delete an order by ID.

@PostMapping("/orders")

    public ResponseEntity<String> createOrder(@RequestBody Order order) {

        try {

            Order newOrder = orderRepository.save(order);

            logger.info("Order created: {}", newOrder);

            return new ResponseEntity<>("Order created successfully", HttpStatus.CREATED);

        } catch (Exception e) {

            logger.error("Error creating order", e);

            return new ResponseEntity<>("Error creating order", HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    /\*\*

     \*

     \* @param id

     \* @return

     \*/

    @GetMapping("/orders/{id}")

    public ResponseEntity<Order> getOrderById(@PathVariable Long id) {

        Optional<Order> order = orderRepository.findById(id);

        if (order.isPresent()) {

            return new ResponseEntity<>(order.get(), HttpStatus.OK);

        } else {

            logger.warn("Order with ID {} not found", id);

            return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        }

    }

    /\*\*

     \*

     \* @param id

     \* @param orderDetails

     \* @return

     \*/

    @PutMapping("/orders/{id}")

    public ResponseEntity<Order> updateOrder(@PathVariable Long id, @RequestBody Order orderDetails) {

        Optional<Order> optionalOrder = orderRepository.findById(id);

        if (optionalOrder.isPresent()) {

            Order existingOrder = optionalOrder.get();

            existingOrder.setName(orderDetails.getName());

            existingOrder.setEmail(orderDetails.getEmail());

            existingOrder.setPhoneNumber(orderDetails.getPhoneNumber());

            existingOrder.setAddress(orderDetails.getAddress());

            existingOrder.setOrderDetails(orderDetails.getOrderDetails());

            Order updatedOrder = orderRepository.save(existingOrder);

            logger.info("Order updated: {}", updatedOrder);

            return new ResponseEntity<>(updatedOrder, HttpStatus.OK);

        } else {

            logger.warn("Order with ID {} not found for update", id);

            return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        }

    }

    /\*\*

     \*

     \* @param id

     \* @return

     \*/

    @DeleteMapping("/orders/{id}")

    public ResponseEntity<String> deleteOrder(@PathVariable Long id) {

        if (orderRepository.existsById(id)) {

            orderRepository.deleteById(id);

            logger.info("Order with ID {} deleted", id);

            return new ResponseEntity<>("Order deleted successfully", HttpStatus.OK);

        } else {

            logger.warn("Order with ID {} not found for deletion", id);

            return new ResponseEntity<>("Order not found", HttpStatus.NOT\_FOUND);

        }

    }

}

Reservation API Endpoints:

- POST /reservations: Create a new reservation.

- GET /reservations/{id}: Retrieve a specific reservation by ID.

- PUT /reservations/{id}: Update a reservation by ID.

- DELETE /reservations/{id}: Cancel a reservation by ID.

@RestController

@RequestMapping("/api")

@CrossOrigin

public class ReservationController {

    private static final Logger logger = LoggerFactory.getLogger(ReservationController.class);

    @Autowired

    private ReservationRepository reservationRepository;

    /\*\*

     \*

     \*

     \* @return

     \*/

    @GetMapping("/reservations")

    public ResponseEntity<List<Reservation>> getAllReservations() {

        List<Reservation> reservations = reservationRepository.findAll();

        return new ResponseEntity<>(reservations, HttpStatus.OK);

    }

    /\*\*

     \*

     \*

     \* @param reservation

     \* @return a success

     \*/

    @PostMapping("/reservations")

    public ResponseEntity<String> createReservation(@RequestBody Reservation reservation) {

        try {

            Reservation newReservation = reservationRepository.save(reservation);

            logger.info("Reservation created: {}", newReservation);

            return new ResponseEntity<>("Reservation created successfully", HttpStatus.CREATED);

        } catch (Exception e) {

            logger.error("Error creating reservation", e);

            return new ResponseEntity<>("Error creating reservation: " + e.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

    /\*\*

     \*

     \*

     \* @param id

     \* @return

     \*/

    @GetMapping("/reservations/{id}")

    public ResponseEntity<Reservation> getReservationById(@PathVariable Long id) {

        Optional<Reservation> reservation = reservationRepository.findById(id);

        if (reservation.isPresent()) {

            return new ResponseEntity<>(reservation.get(), HttpStatus.OK);

        } else {

            return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        }

    }

    /\*\*

     \*

     \*

     \* @param id

     \* @param reservationDetails

     \* @return

     \*/

    @PutMapping("/reservations/{id}")

    public ResponseEntity<Reservation> updateReservation(@PathVariable Long id, @RequestBody Reservation reservationDetails) {

        Optional<Reservation> optionalReservation = reservationRepository.findById(id);

        if (optionalReservation.isPresent()) {

            Reservation existingReservation = optionalReservation.get();

            existingReservation.setFullName(reservationDetails.getFullName());

            existingReservation.setEmail(reservationDetails.getEmail());

            existingReservation.setPhoneNumber(reservationDetails.getPhoneNumber());

            existingReservation.setReservationDate(reservationDetails.getReservationDate());

            existingReservation.setReservationTime(reservationDetails.getReservationTime());

            existingReservation.setNumberOfGuests(reservationDetails.getNumberOfGuests());

            existingReservation.setSpecialRequests(reservationDetails.getSpecialRequests());

            Reservation updatedReservation = reservationRepository.save(existingReservation);

            return new ResponseEntity<>(updatedReservation, HttpStatus.OK);

        } else {

            return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

        }

    }

    /\*\*

     \*

     \*

     \* @param id

     \* @return

     \*/

    @DeleteMapping("/reservations/{id}")

    public ResponseEntity<String> deleteReservation(@PathVariable Long id) {

        if (reservationRepository.existsById(id)) {

            reservationRepository.deleteById(id);

            return new ResponseEntity<>("Reservation deleted successfully", HttpStatus.OK);

        } else {

            return new ResponseEntity<>("Reservation not found", HttpStatus.NOT\_FOUND);

        }

    }

}

The Spring Boot backend communicates with the React frontend via Axios or Fetch API for handling HTTP requests. enabling user interactions such as placing orders, making reservations, and logging in.

API ENDPOINTS TESTS.

