Advanced Web Design Project

Development of a Full-Stack Booking Application

Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Davor Georgijev 213266

Marija Stojanoska 213024

Introduction

This report provides a comprehensive overview of the development process for a full-stack booking application, detailing the technologies used, the architecture implemented, and the key functionalities integrated. The application aims to streamline the process of booking accommodations by allowing users to browse, select, and reserve accommodations seamlessly. Developed using React for the front-end and Spring Boot for the back-end, the application leverages these technologies to deliver a robust and efficient user experience.

Project Overview

The booking application serves as a centralized platform where users can explore various accommodations, view detailed information, register, log in, and make reservations. Data management is facilitated through a PostgreSQL database, ensuring reliable storage and retrieval of user and booking information. React is utilized on the front-end to create dynamic user interfaces, while Spring Boot powers the back-end with RESTful APIs for seamless communication between the client and server.

Technologies Used

Front-end:

- **React:** A JavaScript library for building user interfaces, known for its efficiency and flexibility in creating interactive UI components.
- **React Router:** A routing library for React applications, enabling navigation between different views based on the URL.
- **Axios:** A promise-based HTTP client for making asynchronous requests to the server, essential for fetching data and interacting with APIs.
- **Bootstrap:** A popular CSS framework for developing responsive and mobile-first websites, providing pre-styled components and layout utilities for rapid UI development.

Back-end:

- **Spring Boot:** An opinionated framework for building Java-based applications, offering a streamlined development experience and production-ready features out of the box.
- **Spring Data JPA:** Part of the Spring Data framework, it simplifies data access layer implementation using JPA (Java Persistence API), reducing boilerplate code.
- **PostgreSQL:** An open-source relational database management system, chosen for its robustness, SQL compliance, and scalability in handling structured data.
- **Lombok:** A Java library that reduces boilerplate code by automatically generating getters, setters, and other common methods in Java classes.

- **Spring Security:** A powerful authentication and authorization framework for securing Java applications, essential for protecting sensitive endpoints and managing user sessions securely.
-

Key Features

1. **User Registration and Login:** Provides functionalities for users to create accounts, log in securely, and manage their profiles.
2. **Accommodation Browsing and Filtering:** Allows users to search for accommodations based on criteria such as location, price range, and availability dates.
3. **Accommodation Details:** Displays comprehensive information about each accommodation, including descriptions, amenities, pricing, and images.
4. **Reservation Management:** Enables users to select accommodations and book them for specified dates, managing reservations effectively.
5. **User-Specific Reservations:** Allows users to view their current and past reservations, providing transparency and control over bookings.
6. **Date Validation and Conflict Handling:** Ensures that reservations are made for valid dates and detects any conflicts with existing bookings to maintain data integrity.

Backend Development

The back-end development focuses on implementing business logic, managing data persistence, and ensuring secure communication between the client and server. Spring Boot serves as the foundation for building RESTful APIs that handle CRUD operations and enforce business rules effectively.

Project Structure

The back-end architecture is structured into distinct layers to separate concerns and facilitate modular development:

- **Controllers:** Handle incoming HTTP requests, map them to appropriate service methods, and return HTTP responses to the client.
- **Services:** Implement business logic and orchestrate interactions between controllers and repositories.
- **Repositories:** Define interfaces for interacting with the database, using Spring Data JPA to execute database queries and transactions.
- **Models:** Represent data entities with attributes and relationships, defining the structure of data stored in the database.

```java
@RestController
@RequestMapping("/api/reservations")
@CrossOrigin("http://localhost:3000")
public class ReservationController {

    3 usages
    private final ReservationService reservationService;

    public ReservationController(ReservationService reservationService) {
        this.reservationService = reservationService;
    }

    @GetMapping("/user/{username}")
    public List<Reservation> findAllReservationsForUser(@PathVariable String username) {
        return reservationService.findAllReservationsForUser(username);
    }

    @PostMapping("reserve")
    public Reservation makeReservation(@RequestBody ReservationDto reservationDto) {
        return reservationService.makeReservation(
                reservationDto.getUsername(),
                reservationDto.getResourceId(),
                reservationDto.getDateFrom(),
```

```java
17 usages
@Entity
@Data
@NoArgsConstructor
public class Reservation {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    private User user;
    @ManyToOne
    private Resource resource;
    private LocalDate dateFrom;
    private LocalDate dateTo;

    1 usage
    public Reservation(User user, Resource resource, LocalDate dateFrom, LocalDate dateTo) {
        this.user = user;
        this.resource = resource;
        this.dateFrom = dateFrom;
        this.dateTo = dateTo;
    }
}
```

```java
6 usages
public interface ReservationRepository extends JpaRepository<Reservation, Long> {
    1 usage
    List<Reservation> findByUserUsername(String username);

    no usages
    List<Reservation> findByResourceId(Long id);
}
```

```java
3 usages
@Service
public class ReservationService {
    3 usages
    private final ReservationRepository reservationRepository;
    3 usages
    private final ResourceService resourceService;
    2 usages
    private final UserService userService;

    public ReservationService(ReservationRepository reservationRepository, ResourceService resourceService, UserS
    1 usage
    public Reservation makeReservation(String username, Long resourceId, LocalDate dateFrom, LocalDate dateTo) {
        User user = userService.findByUsername(username);
        Resource resource = resourceService.findById(resourceId);
        if (!resourceService.hasReservationForDate(resource, dateFrom, dateTo) && dateFrom.isBefore(dateTo)) {
            Reservation reservation = new Reservation();
            reservation.setUser(user);
            reservation.setResource(resource);
            reservation.setDateFrom(dateFrom);
            reservation.setDateTo(dateTo);
            return reservationRepository.save(reservation);
        } else {
            throw new IllegalArgumentException("Resource is already booked for the selected dates.");
        }
    }
```

Key Components

User Registration and Login

User authentication and authorization are crucial components of the application's security architecture:

- **User Model:** Represents user data with attributes such as ID, username, hashed password, and role.
- **UserController:** Manages HTTP endpoints for user registration, login, profile management, and authentication-related operations.
- **UserService:** Implements business logic for user registration, password encryption, authentication validation, and authorization checks.
- **Spring Security Configuration:** Configures security settings, defines authentication providers, and restricts access to protected endpoints based on user roles.

Resource Management

Managing accommodations involves handling CRUD operations for accommodation data:

- **Resource Model:** Represents accommodation details such as ID, name, location, description, amenities, price, and availability.
- **ResourceController:** Defines RESTful endpoints for fetching accommodation data, applying filters, and handling pagination.
- **ResourceService:** Implements business logic for retrieving accommodations based on user-specified criteria, ensuring data integrity and performance.
- **ResourceRepository:** Interfaces with the database using Spring Data JPA, executing queries to retrieve, create, update, and delete accommodation records.

Reservation Handling

Reservation management links users with specific accommodations and manages booking details:

- **Reservation Model:** Captures reservation details including ID, start date, end date, associated user, and accommodation.
- **ReservationController:** Exposes endpoints for creating new reservations, retrieving user-specific bookings, and validating booking requests.
- **ReservationService:** Implements business rules to validate booking dates, detect conflicts with overlapping reservations, and enforce booking constraints.
- **ReservationRepository:** Interfaces with the database to execute CRUD operations for reservation data, maintaining consistency and reliability.

Frontend Development

The front-end development focuses on creating an intuitive and responsive user interface using React components and integrating with the back-end APIs to fetch and display data dynamically.

Project Structure

The front-end architecture is organized into modular components, pages, and services to facilitate code reuse and maintainability:

- **Components:** Reusable UI elements such as buttons, forms, cards, and navigation bars used across different views.
- **Pages:** Represent major views of the application corresponding to different routes defined by React Router.
- **Services:** Manage HTTP requests to the back-end APIs using Axios, handling data fetching, submission, and error handling.

```
return (
    <Router>
        <Header currentUser={currentUser} logout={this.logout}/>
        <main>
            <Routes>
                <Route path="/" element={<Home resources={resources}/>}/>
                <Route path="/reservations/user"
                       element={<ReservationsForUser reservations={reservationsForUser}/>}/>
                <Route path="/reserve" element={<ReserveForm reserve={this.reserve} error={reserveError}
                <Route path="/resources" element={
                    <Resources
                        resources={resources}
                        reserve={this.reserve}
                        currentUser={currentUser}
                        fetchResources={this.fetchResources}
                        setResources={this.setResources}
                    />
                }/>
                <Route path="/login"
                       element={<LoginForm login={this.login} register={this.register} error={loginError
                <Route path="/resource/:id" element={<ResourceDetails getResource={this.getCurrentResour
                                                                    resource={currentResource}/>}/>
            </Routes>
        </main>
        <Footer/>
    </Router>
);
```

Key Components

Home Page

The home page serves as the entry point to the application, providing an overview of featured accommodations and navigation links to different sections:

- **Hero Section:** Highlights key features of the application and promotes featured accommodations to attract user engagement.
- **Featured Accommodations:** Displays a selection of top-rated accommodations fetched dynamically from the back-end using Axios.
- **Navigation Links:** Facilitate easy access to different sections of the application, enhancing user navigation and experience.

User Authentication

User authentication features include registration and login functionalities:

- **Login Form:** Allows existing users to enter their credentials securely and authenticate via Axios requests to the back-end.
- **Registration Form:** Enables new users to create accounts by providing required details and submitting registration requests.
- **Authentication Flow:** Manages user sessions and redirects authenticated users to the home page or their personalized dashboard based on role-based access control.

Resource Listing and Details

Users can browse and view details of available accommodations:

- **Resource List:** Displays a list of accommodations with brief summaries and thumbnail images, allowing users to filter results by location, price range, and availability.
- **Resource Details Page:** Presents comprehensive information about each accommodation, including descriptions, amenities, pricing, and high-resolution images, enhancing user decision-making.

Reservation Management

Users can manage their reservations through dedicated pages and forms:

- **Reservation Form:** Allows users to select accommodations, specify booking dates, and submit reservation requests securely.
- **Reservation Confirmation:** Provides feedback on successful reservations and handles errors gracefully, ensuring a seamless booking experience.
- **Reservation History:** Displays a list of current and past reservations for the logged-in user, showcasing booking details and status updates.

```
return (
    <div className="container mb-5">
        <div className="row justify-content-center">
            <div className="col-4 mt-5">
                {error && <div className="alert alert-danger" role="alert">{error}</div>}
                <form>
                    <div className="mb-3">
                        <label htmlFor="dateFrom" className="form-label">Date From</label>
                        <input type="date" className="form-control" id="dateFrom" value={dateFrom} onChange={
                    </div>
                    <div className="mb-3">
                        <label htmlFor="dateTo" className="form-label">Date To</label>
                        <input type="date" className="form-control" id="dateTo" value={dateTo} onChange={(e
                    </div>
                    <button type="button" className="btn" style={{backgroundColor:"#D4A373",color:"white"}} c
                </form>
            </div>
        </div>
    </div>
);
```

```
return (
    <div className="container mt-4" style={containerStyle}>
        {reservations.length === 0 ? (
            <p className="text-center">You have no reservations.</p>
        ) : (
            <>
                <Calendar
                    tileClassName={tileClassName}
                    className="reservation-calendar"
                />
                <div className="reservations-list mt-4">
                    {reservations.map((reservation, index) => (
                        <div key={index} className="reservation-card">
                            <img src={`${process.env.PUBLIC_URL}/images/${reservation.resource.imageUrl}`} alt
                            <div className="reservation-info">
                                <h5>{reservation.resource.name}</h5>
                                <p>From: {format(new Date(reservation.dateFrom), formatStr: 'MM/dd/yyyy')}</p>
                                <p>To: {format(new Date(reservation.dateTo), formatStr: 'MM/dd/yyyy')}</p>
                                <p>Total Amount: ${calculateTotalAmount(reservation.dateFrom, reservation.date
                            </div>
                        </div>
                    ))}
                </div>
            </>
        )}
    </div>
```

API Integration

Axios facilitates seamless communication between the React front-end and Spring Boot back-end, enabling efficient data exchange and synchronization of user actions across the application.

```
import axios from '../custom-axios/axios';

const Service :{...}  = {
    register: (username, password) => {
        return axios.post( url: "/users/register",  data: {
            "username": username,
            "password": password
        });
    },
    login: (username, password) => {
        return axios.post( url: "/users/login",  data: {
            "username": username,
            "password": password
        });
    },
    fetchReservationsForUser: (username) => {
        return axios.get( url: `/reservations/user/${username}`);
    },
    reserve: (username, resourceId, dateFrom, dateTo) => {
        return axios.post( url: "/reservations/reserve",  data: {
            "username": username,
            "resourceId": resourceId,
            "dateFrom": dateFrom,
            "dateTo": dateTo
        });
    },
    fetchResources: (city : string  = "", dateFrom : string  = "", dateTo : string  = "") => {
```

```
import axios from "axios";

const instance : AxiosInstance  = axios.create({
    baseURL: 'http://localhost:9090/api',
    headers: {
        'Access-Control-Allow-Origin' : '*'
    }
})


5+ usages
export default instance;
```

Requirements Gathering

- **Functional Requirements:** Identified key features such as user authentication, accommodation browsing, reservation management, and data validation.
- **Non-Functional Requirements:** Considered factors such as performance, scalability, security, and usability to ensure the application meets user expectations and business needs.

Technology Selection

- **Front-end Technology:** Chose React for its component-based architecture, virtual DOM rendering, and strong community support for building modern, responsive user interfaces.
- **Back-end Framework:** Selected Spring Boot for its ease of development, built-in features for RESTful APIs, dependency management, and integration with Spring ecosystem components like Spring Data JPA and Spring Security.
- **Database Management System:** Opted for PostgreSQL for its reliability, ACID compliance, and ability to handle structured data effectively, ensuring data integrity and robust transaction support.

Architecture Design

- **Application Architecture:** Designed a scalable and modular architecture with separate layers for presentation (front-end), business logic (back-end services), and data access (repositories), promoting code reusability, maintainability, and testability.
- **Database Schema Design:** Defined entity relationships, attributes, and constraints in the PostgreSQL database schema to support entities like Users, Accommodations, and Reservations, ensuring efficient data storage and retrieval.

Implementing the Backend

Developed back-end functionalities to support application features and business requirements:

User Management

- **User Registration:** Implemented user registration endpoint to create new accounts, validate input data, hash passwords securely using bcrypt, and persist user details in the PostgreSQL database.
- **User Authentication:** Developed authentication mechanism to validate user credentials during login, generate JSON Web Tokens (JWT) for authenticated sessions, and enforce role-based access control (RBAC) using Spring Security annotations.
- **Profile Management:** Integrated endpoints for users to update profile information, reset passwords securely, and manage account settings with appropriate validation and error handling.

Accommodation Management

- **Accommodation Retrieval:** Implemented RESTful endpoints to retrieve a list of accommodations based on user-specified filters such as location, price range, and availability dates, optimizing query performance with efficient database indexing and caching strategies.
- **Accommodation Details:** Designed endpoints to fetch detailed information about specific accommodations, including descriptions, amenities, pricing, and multimedia content, enhancing user decision-making and engagement.
- **Search and Filtering:** Implemented search functionality and filtering options to narrow down accommodation listings dynamically, providing users with relevant and personalized search results based on their preferences.

Reservation Handling

- **Reservation Creation:** Developed endpoints to create new reservations for selected accommodations, validate booking dates against availability, and enforce business rules to prevent overlapping reservations or booking conflicts.
- **Reservation Retrieval:** Implemented endpoints to fetch user-specific reservations, display detailed booking information, and manage reservation lifecycle states (e.g., pending, confirmed, canceled) with comprehensive status updates and notifications.
- **Transactional Integrity:** Ensured transactional integrity and consistency using Spring Data JPA transactions, managing database concurrency and rollback mechanisms to handle exceptional scenarios and ensure data reliability.

Implementing the Frontend

Built intuitive and responsive user interfaces using React components to enhance user interaction and visual appeal:

Home Page

- **UI Section:** Designed a visually appealing hero section with compelling visuals, promotional content, and call-to-action buttons to engage users and encourage exploration of featured accommodations.
- **Featured Accommodations:** Implemented a carousel or grid layout to showcase top-rated accommodations fetched from the back-end dynamically, enhancing user discovery and decision-making.
- **Navigation Links:** Configured navigation links and menu options to facilitate seamless navigation between different sections of the application, improving user accessibility and browsing experience.

User Authentication

- **Login Form:** Developed a user-friendly login form with input validation, error handling, and secure submission of credentials to the back-end authentication service via Axios requests, ensuring robust security and user authentication.
- **Registration Form:** Designed an intuitive registration form to capture user details, validate input data, and securely register new accounts by communicating with the back-end registration endpoint, implementing CAPTCHA or other security measures for spam prevention.
- **Authentication Flow:** Managed user sessions, stored authentication tokens securely (e.g., in local storage or cookies), and implemented redirection logic based on user authentication status (logged in/out), ensuring seamless navigation and personalized user experiences.

Accommodation Listing and Details

- **Accommodation List:** Developed a paginated or infinite scroll list of accommodations with thumbnail images, brief summaries, and interactive filters (e.g., location, price range) to enhance user search and discovery, leveraging Axios for efficient data fetching and state management (e.g., Redux) for optimized performance.
- **Accommodation Details Page:** Designed a detailed view of individual accommodations with comprehensive information, high-resolution images, user reviews, availability calendar, and booking options, enhancing user engagement and decision-making with interactive elements and dynamic content updates.

Reservation Management

- **Reservation Form:** Implemented an intuitive reservation form to capture booking details (e.g., dates, guest count), validate input data against accommodation availability, and submit reservation requests securely to the back-end reservation service via Axios, providing real-time feedback on booking status and availability.
- **Reservation Confirmation:** Designed confirmation pages or modals to notify users of successful reservations, display booking details, and handle errors gracefully with actionable feedback and error messages, ensuring a seamless booking experience and user satisfaction.
- **Reservation History:** Developed a dedicated section or dashboard for users to view their current and past reservations, access detailed booking information, manage reservation status (e.g., cancel bookings), and receive notifications or reminders for upcoming stays, enhancing user control and transparency over their booking history.

API Integration

Axios facilitated seamless integration between the React front-end and Spring Boot back-end, enabling efficient data exchange and synchronization of user actions across the application:

- **HTTP Requests:** Managed asynchronous HTTP requests using Axios to fetch data from RESTful API endpoints, handle form submissions, and interact with back-end services securely.
- **Response Handling:** Implemented error handling and response parsing with Axios interceptors to manage HTTP status codes, network errors, and server-side exceptions, ensuring robust communication and user experience.
- **State Management:** Utilized React context, Redux, or state hooks to manage application state, update UI components based on data fetched from the back-end, and synchronize user interactions with real-time updates, optimizing performance and responsiveness.

Planning and Design

The development process began with thorough planning and design to define project requirements, select appropriate technologies, and establish a scalable architecture:

Requirements Gathering

- **Functional Requirements:** Identified core features such as user authentication, accommodation browsing, reservation management, and booking validation to meet user needs and business objectives.
- **Non-Functional Requirements:** Considered factors such as performance, scalability, security, and usability to ensure the application delivers a seamless and reliable user experience across different devices and platforms.

Technology Selection

- **Front-end Framework:** Selected React for its component-based architecture, virtual DOM rendering, and extensive ecosystem of libraries and tools for building responsive, single-page applications (SPAs).
- **Back-end Framework:** Chose Spring Boot for its rapid development capabilities, built-in support for RESTful APIs, dependency management with Maven or Gradle, and integration with Spring ecosystem components like Spring Security and Spring Data JPA.
- **Database Management System:** Opted for PostgreSQL for its reliability, ACID compliance, and robust support for structured data storage, ensuring data integrity and efficient query performance for transactional operations.

Architecture Design

- **Application Architecture:** Designed a layered architecture with separation of concerns, including presentation (front-end components), business logic (back-end services), and data access (repositories and database interactions), promoting code reusability, maintainability, and scalability.
- **Database Schema Design:** Defined entity relationships, attributes, and constraints in the PostgreSQL database schema to support entities such as Users, Accommodations, Reservations, and Reviews, ensuring optimal data storage and retrieval for application functionalities.

Backend Environment Setup

- **Spring Boot Initialization:** Initialized a new Spring Boot project using Spring Initializr, configuring necessary dependencies (e.g., Spring Web, Spring Data JPA, Spring Security), and defining application properties for database connection (PostgreSQL), logging, and security settings.
- **Database Configuration:** Configured PostgreSQL database properties, created database schema, defined JPA entities with annotations (e.g., @Entity, @Table), and implemented data migration scripts (e.g., Liquibase or Flyway) to manage schema evolution and versioning.
- **Security Configuration:** Integrated Spring Security to enforce authentication and authorization rules, implementing user authentication providers, password hashing (e.g., bcrypt), CSRF protection, role-based access control (RBAC), and secure endpoints (e.g., HTTPS) to protect sensitive data and prevent unauthorized access.

Frontend Environment Setup

- **React Application Setup:** Bootstrapped a new React project using create-react-app or similar tools, configuring essential libraries (e.g., React Router, Axios) and tooling (e.g., ESLint, Prettier) for code quality and linting, ensuring consistent coding standards and best practices.
- **Component Development:** Designed and developed reusable UI components (e.g., buttons, forms, cards) with styled components (e.g., Bootstrap, Material-UI) for consistent design and responsiveness across different screen sizes and devices.
- **State Management:** Implemented state management using React context, Redux, or state hooks (e.g., useState, useEffect) to manage application state, handle asynchronous data fetching (e.g., Axios requests), and synchronize UI updates based on user interactions and data changes from the back-end.

Backend Implementation

Developed back-end functionalities to support application features and business logic:

User Management

- **User Registration:** Implemented RESTful endpoints for user registration, validating input data (e.g., email format, password strength) using Spring validators or custom validation annotations, encrypting passwords securely (e.g., bcrypt), and persisting user details (e.g., username, hashed password) in the PostgreSQL database.
- **User Authentication:** Developed authentication mechanisms to verify user credentials during login, generate JWT (JSON Web Tokens) for authenticated sessions, store tokens securely (e.g., local storage, cookies), and enforce role-based access control (RBAC) using Spring Security annotations (@Secured, @PreAuthorize) for fine-grained authorization.
- **Profile Management:** Implemented endpoints for users to update profile information (e.g., username, email), reset passwords securely via email confirmation (e.g., using JavaMail or SMTP), and manage account settings (e.g., two-factor authentication, GDPR compliance) with robust error handling and validation checks.

Accommodation Management

- **Accommodation Retrieval:** Designed RESTful endpoints to fetch a list of accommodations based on user-defined filters (e.g., location, price range, availability dates), optimizing query performance with indexed database fields, and caching strategies (e.g., Redis) for efficient data retrieval.
- **Accommodation Details:** Implemented endpoints to retrieve detailed information about specific accommodations (e.g., name, description, amenities, pricing), including multimedia content (e.g., images, videos) fetched asynchronously using Axios, enhancing user engagement and decision-making with rich visual content.
- **Search and Filtering:** Developed search functionality and dynamic filtering options (e.g., by location, price, rating) to refine accommodation listings, integrating with frontend components (e.g., input fields, dropdowns) for real-time data updates and responsive user experience.

Reservation Handling

- **Reservation Creation:** Implemented endpoints for users to create new reservations for selected accommodations, validating booking dates against availability constraints (e.g., overlapping reservations), enforcing business rules (e.g., maximum stay duration), and handling transactional operations with Spring Data JPA for data consistency.
- **Reservation Retrieval:** Designed endpoints to fetch user-specific reservations, retrieve detailed booking information (e.g., dates, guest count, status), and manage reservation lifecycle states (e.g., pending, confirmed, canceled) with comprehensive status updates and notifications for users.
- **Transactional Integrity:** Ensured transactional integrity and database consistency using Spring Data JPA transactions, implementing optimistic locking or pessimistic locking strategies to prevent concurrency issues (e.g., race conditions, stale data) and roll back transactions on failure.

Frontend Implementation

Developed responsive and interactive user interfaces using React components and integrated with backend APIs for seamless data fetching and synchronization:

Home Page

- **Hero Section:** Designed a visually appealing hero section with promotional content, featured accommodations, and call-to-action buttons to engage users and promote exploration of available listings.
- **Featured Accommodations:** Implemented carousel or grid layouts to showcase top-rated accommodations fetched dynamically from the back-end, enhancing user discovery and interaction with personalized recommendations.
- **Navigation Links:** Configured navigation links and menu options to facilitate seamless transitions between different sections of the application, improving user accessibility and navigation flow.

User Authentication

- **Login Form:** Developed a user-friendly login form with input validation, error handling, and secure submission of credentials to the back-end authentication service via Axios requests, ensuring robust security and authentication flow.
- **Registration Form:** Designed an intuitive registration form to capture user details (e.g., username, email, password), validate input data (e.g., password strength), and securely create new accounts by communicating with the back-end registration endpoint, implementing CAPTCHA or reCAPTCHA for spam prevention.
- **Authentication Flow:** Managed user sessions and stored authentication tokens securely (e.g., local storage, cookies), implemented redirection logic based on user authentication status (logged in/out), and enforced role-based access control (RBAC) to restrict access to authenticated routes and sensitive functionalities.

Accommodation Listing and Details

- **Accommodation List:** Developed paginated or infinite scroll lists of accommodations with thumbnail images, brief summaries, and interactive filters (e.g., location, price range) for dynamic search and browsing, leveraging Axios for efficient data fetching and Redux for state management to optimize performance and responsiveness.
- **Accommodation Details Page:** Designed detailed views of individual accommodations with comprehensive information (e.g., description, amenities, pricing), high-resolution images or multimedia content fetched asynchronously via Axios, and interactive elements (e.g., booking options, availability calendar) to facilitate informed decision-making and user engagement.

Reservation Management

- **Reservation Form:** Implemented intuitive reservation forms to capture booking details (e.g., dates, guest count), validate input against accommodation availability (e.g., overlapping reservations), and submit reservation requests securely to the back-end reservation service via Axios, providing real-time feedback on booking status and availability.
- **Reservation Confirmation:** Designed confirmation pages or modals to notify users of successful reservations, display booking details (e.g., dates, accommodation details), and handle errors gracefully with actionable feedback and error messages, ensuring a seamless booking experience and user satisfaction.
- **Reservation History:** Developed dedicated sections or dashboards for users to view current and past reservations, access detailed booking information (e.g., status, cancellation options), manage reservation lifecycle (e.g., cancel bookings), and receive notifications or reminders for upcoming stays, enhancing user control and transparency over their booking history.

API Integration

Leveraged Axios for seamless integration between the React front-end and Spring Boot back-end, enabling efficient data exchange, real-time updates, and synchronized user interactions across the application:

- **HTTP Requests:** Managed asynchronous HTTP requests using Axios to fetch data from RESTful API endpoints (e.g., accommodation listings, user reservations), handle form submissions (e.g., login, registration), and interact with back-end services securely with error handling and retry strategies.
- **Response Handling:** Implemented interceptors with Axios to manage HTTP status codes (e.g., 200, 400, 401), network errors, and server-side exceptions, providing consistent error messages and feedback to users, and ensuring robust communication and user experience.
- **State Management:** Utilized Redux or React context for centralized state management, updating UI components based on data fetched from the back-end (e.g., accommodation details, reservation status), and synchronizing user interactions (e.g., bookings, cancellations) with real-time updates and notifications, optimizing performance and responsiveness.

Conclusion

The Airbnb-like accommodation booking application leverages React.js and Spring Boot frameworks to deliver a seamless user experience with intuitive navigation, responsive design, and robust backend functionality. By implementing secure authentication mechanisms, dynamic accommodation search, reservation management, and integrating scalable database solutions, the application meets diverse user needs, ensures data integrity, and supports transactional operations efficiently. With continuous testing, deployment automation, and monitoring, the application maintains high performance, reliability, and scalability, while fostering collaboration and knowledge sharing to drive ongoing improvements and innovation in accommodation booking services.