****

|  |  |
| --- | --- |
| College Name | Dinabandhu Andrews Institute of Technology and Management |
| College Code | 154 |
| University Name | Maulana Abul Kalam Azad University of Technology |
| Stream | Bachelor of Computer Application (BCA) |
| Batch | 2022 – 2025 |
| Semester | 6th Semester |
| Paper Name | Major Project & Grand Viva |
| Paper Code | BCAD681 |

|  |  |
| --- | --- |
| Project Name | FoodSphere: Food Ordering Web-Application |
| Group no. | 15 |
| Mentor Name | Amrita Nandi |
| Submission Date | 16/06/2025 |
| Submitted By | Group 15 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl.no | Name | Class Roll | University Roll no | University Registration no |
| 1 | Ripan Baidya | 08 | 15401222024 | 221541010055 |
| 2 | Torsha Dey | 44 | 15401222065 | 221541010106 |
| 3 | Sayan Mallick | 54 | 15401222053 | 221541010069 |
| 4 | Shruti Karmakar | 74 | 15401222083 | 221541010074 |
| 5 | Jit Mondal | 77 | 15401222085 | 221541010034 |
| 6 | Pronay Bhattacharya | 45 | 15401222052 | 221541010049 |

* Signature of Mentor: \_\_\_\_\_\_\_\_\_\_\_
* Signature of Project Co-ordinator: \_\_\_\_\_\_\_\_\_\_\_
* Signature of External Examiner: \_\_\_\_\_\_\_\_\_\_\_

Table Of Contents

**Content**  page

* **Title Page ……………………………………………………………………………………**
* **Introduction ………………………………………………………………………………...**
* **Architecture and Design** **……………………………………………………………….......**
* **Technologies Used ……………………………………………………………….................**
* **Features and Functionalities ………………………………………………………………**
  + User Features
  + Admin Features
* **User Interface (UI) ………………………………………………………………................**
* **Backend Implementation ……………………………………………………………….....**
  + Database Design and Schema Overview
  + API Endpoints
  + Controller Classes Overview
  + Service Classes and Business Logic
  + Repository/DAO Layer (Database Queries)
* **Frontend Implementation ………………………………………………………………....**
* React Component Structure and Hierarchy
* State Management (Redux)
* API Calls and Data Fetching
* Error Handling and Validation
* **Security Measures ………………………………………………………………………….**
* Authentication and Authorization (e.g., JWT, session management)
* **Testing ………………………………………………………………………………………**
* API Testing (Postman)
* **Deployment …………………………………………………………………………………**
* Deployment Environment (Railway, Vercel)
* Deployment Process and Steps
* **User Guide ………………………………………………………………………………….**
* **Challenges and Solutions ………………………………………………………………….**
* **Conclusion and Future Scope ……………………………………………………………..**
* **References …………………………………………………………………………………..**

**Introduction**

**Project Overview**

The **FoodSphere Food Ordering System** is a role-based web application designed to simplify the process of ordering food online. Built with a robust Spring Boot backend and a modern React frontend, FoodSphere aims to deliver a seamless, secure, and efficient platform for both end-users and restaurant administrators. The system enables users to browse food items, manage orders, and securely complete transactions, all from the convenience of their devices. Additionally, it empowers restaurant owners to manage their restaurant profiles, food menus, and operational status, ensuring smooth restaurant operations and enhanced user satisfaction.

**Purpose and Scope**

The FoodSphere system was developed as a comprehensive solution for online food ordering, catering to the needs of a college or community campus environment. With a focus on user experience, FoodSphere addresses the challenges faced by both customers and restaurant operators by providing a secure, user-friendly platform with streamlined processes for food ordering, order tracking, and payment handling.

The scope of this project includes:

* **User Functionality**: Account registration, browsing of food items, order management, and secure payment.
* **Restaurant Management**: Creation and maintenance of restaurant profiles, menu updates, and ingredient management.
* **Admin Control**: Monitoring of orders, oversight of restaurant operations, and ensuring system functionality.

This extensive feature set ensures that FoodSphere caters to a wide array of user needs, from individual consumers to restaurant administrators and system managers.

**Objectives**

The primary objectives of the FoodSphere Food Ordering System are:

* To create a reliable, secure, and intuitive online platform for food ordering.
* To provide a user-friendly interface for consumers to place orders, manage their carts, and track past orders.
* To enable restaurant owners to efficiently manage restaurant information, food items, and ingredient customization.
* To facilitate administrative oversight of orders, operational control of restaurant listings, and secure payment processing.

By achieving these objectives, FoodSphere aspires to enhance the food ordering experience, reduce friction in managing restaurant operations, and provide a secure platform for handling transactions.

**Key Technologies**

The FoodSphere Food Ordering System utilizes a combination of modern frameworks, libraries, and tools to ensure reliability, security, and ease of use:

* **Spring Boot**: Serves as the core backend framework, providing efficient handling of RESTful APIs and business logic.
* **Spring Security with JWT**: Ensures secure authentication and role-based access, leveraging JSON Web Tokens to manage user sessions and permissions.
* **Spring Data JPA**: Facilitates seamless integration with the PostgreSQL database, enabling robust data management.
* **PostgreSQL**: A relational database used to store essential application data, including user information, restaurant profiles, and order history.
* **Stripe Payment Gateway**: Integrates secure payment processing, ensuring reliable and protected online transactions for users.
* **React:** A JavaScript library for building the frontend with a component-based architecture, enabling fast rendering, efficient UI updates, and reusable, modular code for seamless user interactions.
* **Redux:** A state management tool that centralizes the application’s state, ensuring consistent data across components. It manages dynamic interactions like cart updates and order tracking, supporting complex data flows.
* **Tailwind CSS:** A utility-first CSS framework that enables a clean, responsive, and customizable UI through reusable classes, achieving consistent design across screen sizes with minimal custom CSS.

**Features Summary**

FoodSphere is equipped with a variety of features designed to meet the needs of different users within the system:

* **User Functionality**: Registration, login, food browsing, cart and order management, and order tracking.
* **Restaurant Management**: Profile creation, image uploads, and ingredient customization to enhance menu offerings.
* **Admin Functionality**: Comprehensive order monitoring, restaurant oversight, and operational control for maintaining service quality.
* **Payment Integration**: A secure and reliable online payment option using the Stripe gateway to ensure smooth, secure transactions.

**Summary**

Overall, the FoodSphere Food Ordering System provides an integrated, flexible, and secure solution for online food ordering. Through its well-designed functionalities and modern technology stack, the system not only simplifies the ordering process for users but also offers powerful tools for restaurant and administrative management.

**Architecture and Design**

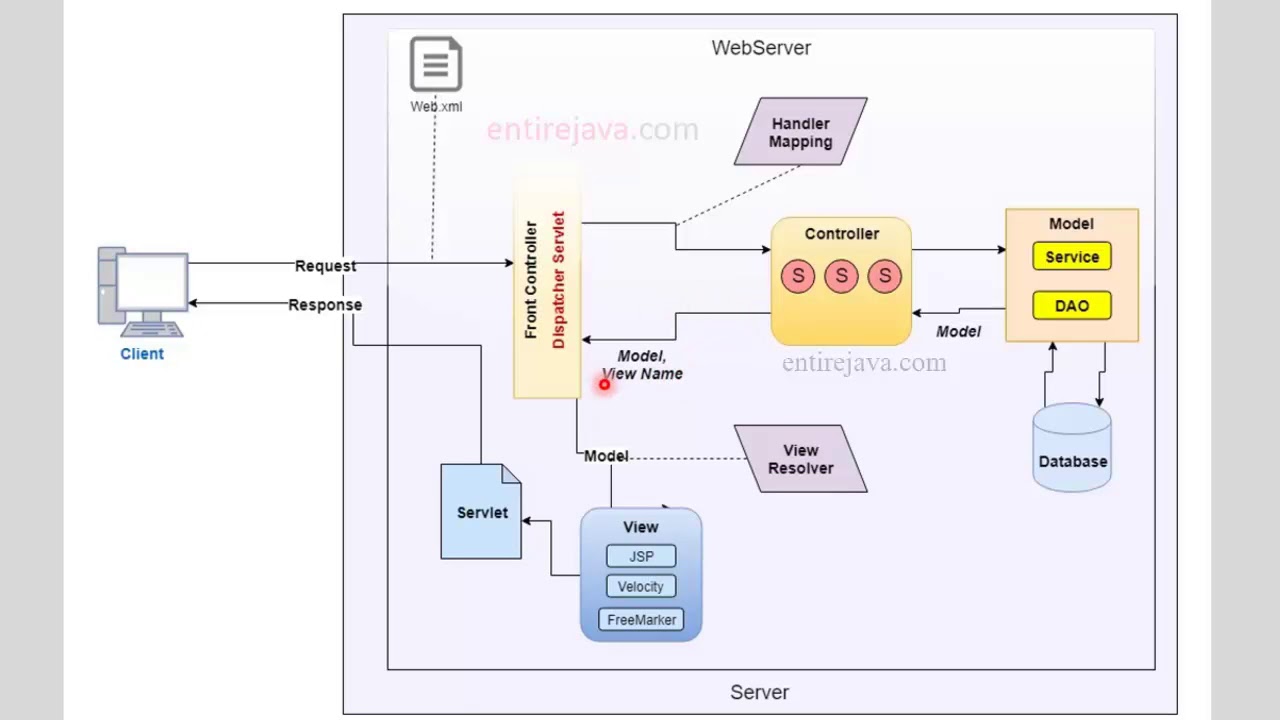
**Architecture Overview**

The **Food Ordering Web Application** is built on a **client-server architecture**, where the frontend and backend operate as separate entities that communicate over a network. This architecture promotes scalability, maintainability, and efficient data management.

* **Frontend:** Developed using **React**, the frontend is responsible for rendering the user interface and managing user interactions. It communicates with the backend via RESTful APIs to fetch +data and send requests.
* **Backend:** Powered by **Spring Boot**, the backend manages business logic, data processing, and interactions with the database. It exposes RESTful APIs that the frontend consumes.
* **Database:** The application utilizes **PostgreSQL** for persistent storage of user data, menu items, orders, and payment details.

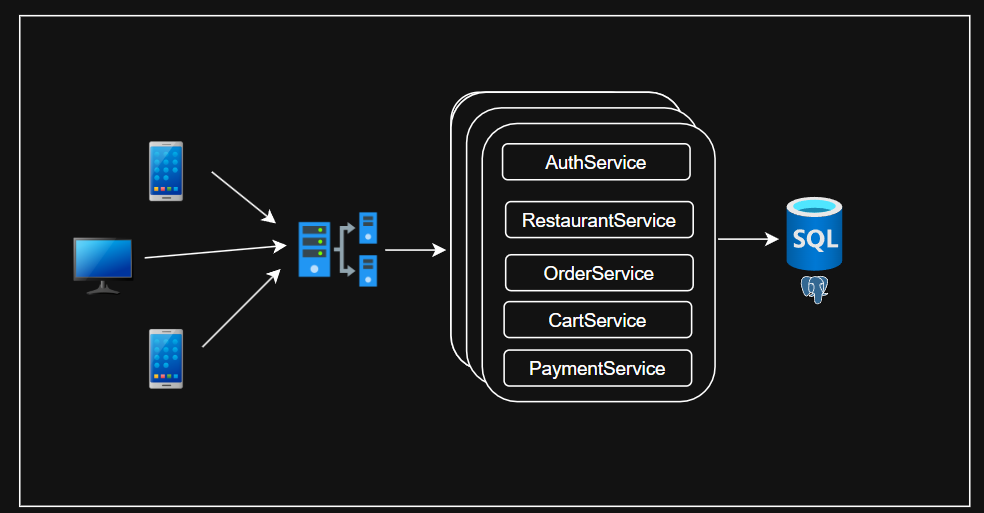
The overall architecture is illustrated below:

**Architecture of Spring MVC**



**System Design**

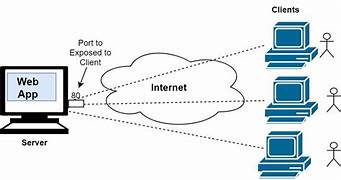
This image below illustrates a high-level overview of a microservices-based system architecture. It shows how clients (web and mobile) and administrators interact with a central UI, which communicates with an API Gateway. The API Gateway routes requests to different services such as UserService, Restaurant Service, Cart Service, Order Service, and Payment Service. These services interact with a PostgreSQL database for data persistence. The system also includes an Authentication & Authorization layer to ensure secure access and a Notification component for user updates. This design enables modular, scalable, and flexible service management, improving maintainability and performance.

****

1. **Client Request:** The client sends an HTTP request, which is intercepted by the Front Controller (Dispatcher Servlet). The web.xml file is configured to map incoming requests to this dispatcher.
2. **Handler Mapping:** The Dispatcher Servlet consults Handler Mapping to determine the appropriate Controller to handle the request.
3. **Controller:** The controller processes the request, interacts with the Model (containing business logic and database access through the Service and DAO layers), and prepares the response data.
4. **View Resolver:** The controller sends the model data and view name back to the Dispatcher Servlet, which then uses View Resolver to identify the appropriate view template (e.g., JSP, Velocity, or Free Marker).
5. **Response:** The selected view is rendered with the model data, and the final response is sent back to the client.

This MVC architecture ensures separation of concerns, making the application more modular, scalable, and maintainable.

**Client Server Architecture**

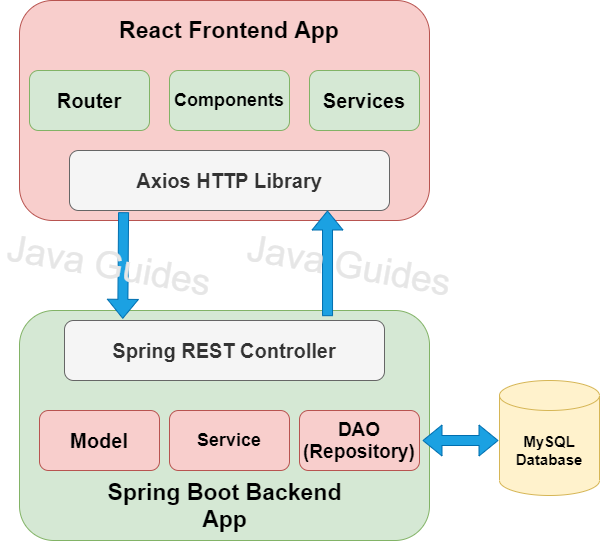


This image depicts a basic client-server architecture for a web application. Here’s a brief explanation:

1. **Server**: The web application is hosted on a server, which is connected to the internet and listens for incoming requests on a specific port (e.g., port 80 for HTTP).
2. **Internet**: The internet acts as a communication medium between the server and multiple clients.
3. Clients: Multiple clients (represented by computers and users) can access the web application via the internet by sending requests to the server's exposed port.

This setup enables clients to interact with the web application hosted on the server from different locations over the internet.

**Architecture of How react frontend and sprig-boot backend work**



The image depicts a typical architecture for a full-stack web application using ReactJS on the frontend and Spring Boot on the backend. Here's a breakdown:

**React Frontend:**

* Router: Handles navigation and routing within the application.
* Components: Reusable UI components that make up the application's interface.
* Services: Business logic and data handling for the frontend.
* Axios: An HTTP library used for making requests to the backend API.

**Spring Boot Backend**:

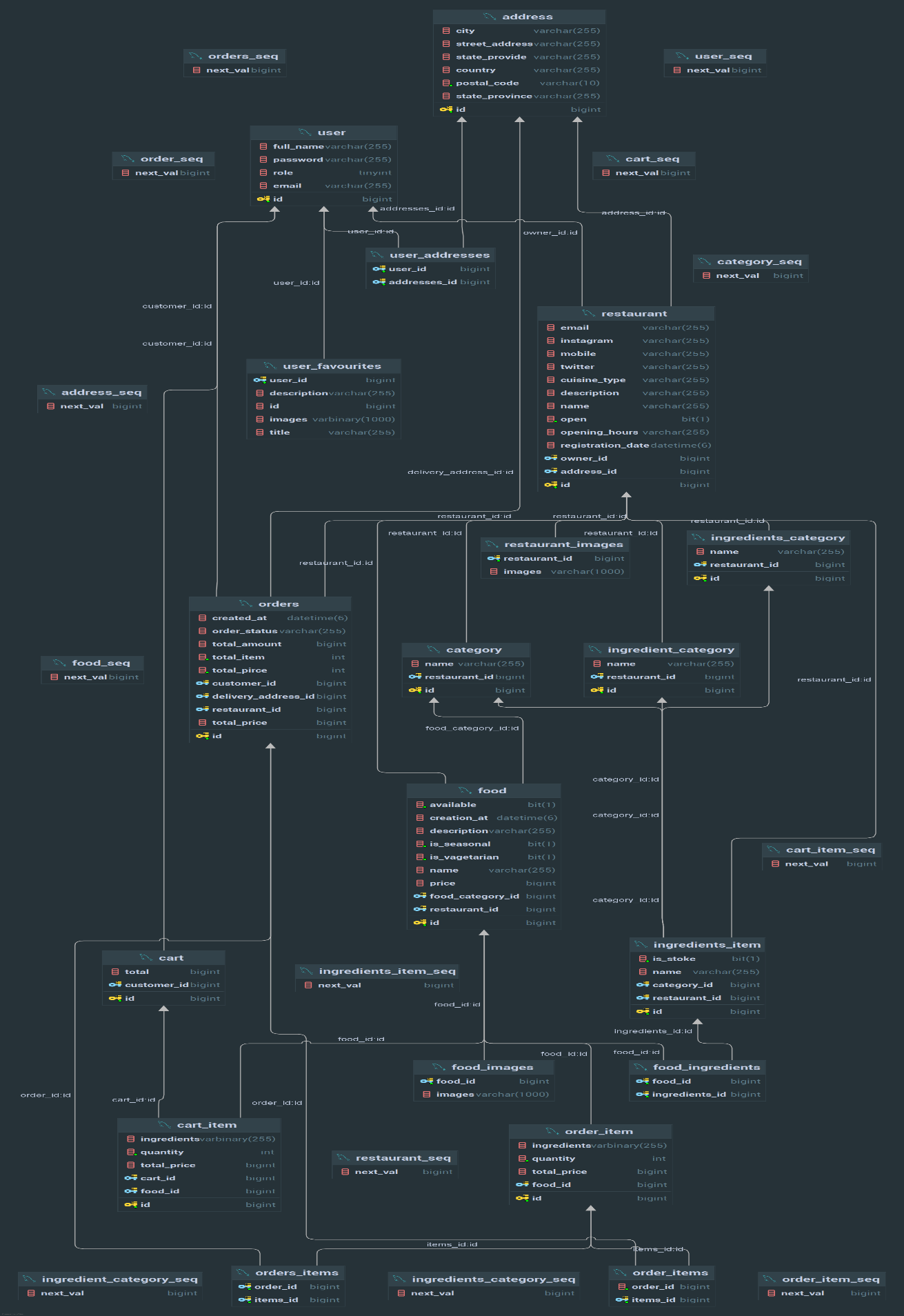
* REST Controller: Handles incoming HTTP requests from the frontend and routes them to appropriate services.
* Model: Represents the data structure for the application.
* Service: Contains business logic and data processing methods.
* DAO (Repository): Interacts with the database to retrieve and store data.
* PostgreSQL Database: Stores the application's data.

**Flow**:

1. User Interaction: User interacts with the React frontend.
2. HTTP Request: The frontend sends an HTTP request to the Spring Boot REST controller using Axios.
3. Request Handling: The REST controller receives the request and delegates it to the appropriate service.
4. Data Processing: The service handles the request, potentially interacting with the DAO to retrieve or store data from the database.
5. Response: The service returns a response to the REST controller.
6. Response to Frontend: The REST controller sends the response back to the frontend, which is then displayed to the user.

This architecture provides a clear separation of concerns between the frontend and backend, making the application more maintainable and scalable.

**Class Diagrams**



**Conclusion**

This architecture and design section outlines the foundational components of the Food Ordering Web Application. The separation of concerns between the frontend and backend enhances maintainability and scalability, ensuring a seamless user experience.

**Features and Functionalities**

**User Features**

1. **User Registration/Login**
   * Users can create an account by providing their name, email, password and mentioning role.
   * The login process includes validation and secure authentication.
2. **Menu Browsing**
   * Users can view a categorized menu of food items with images, descriptions, and prices.
   * Search functionality allows users to find specific items quickly.
3. **Ordering**
   * Users can select items, customize their orders (e.g., size, toppings), and add them to their cart.
   * A user-friendly cart interface displays selected items and total cost.
4. **Order Tracking**
   * Users can track the status of their orders in real time & order status will managed by admin.
   * Notifications are sent via email or app alerts for order updates.[notification mechanism not developed]
5. **Payment Processing**
   * Implement stripe payment gateway, user can payment through cart.
   * Secure payment processing ensures user data protection.
6. **Profile Management**
   * Users can manage their profiles, including updating personal information.
   * Order history is accessible for users to view past orders and reorder favourites.

**Admin Features**

1. **Admin Dashboard**
   * A centralized dashboard provides an overview of key metrics such as total orders, active users, and sales reports.
2. **Menu Management**
   * Admins can add, edit, or remove food items from the menu, including descriptions, prices, and availability status.
   * Category management allows grouping items for easier navigation.
3. **Order Management**
   * Admins can view and manage all orders, update statuses (e.g., preparing, delivered), and handle order cancellations or modifications.
4. **User Management**
   * Admins can view and manage user accounts, including suspending or deleting accounts if necessary.
   * Access to user activity logs for security and support purposes.
5. **Restaurant Management**
   * Admin can management the status of restaurant.
   * Can perform all the operation related to the restaurant. Like food item adding, updating, restaurant adding, removing, restaurant order status management and many more.
6. **Security Management**
   * Admins manage user roles and permissions, ensuring only authorized personnel have access to sensitive functions.

This section clearly outlines the functionalities available to both users and admins, showcasing the comprehensive nature of your food ordering web application. Adjust or expand upon any points to better fit your specific implementation!

**User Interface (UI)**

The User Interface of the **Food Ordering Web Application** is designed to provide an intuitive and seamless user experience. The following key aspects highlight the UI components and navigation flow:

**1. Login and Registration Screens**

* **Login Page**: A simple and clean design that prompts users to enter their credentials (email and password). Includes options for password recovery.
* **Registration Page**: A straightforward form for new users to sign up, requiring essential information like name, email, and password.

**2. Home Page**

* **Menu Overview**: Displays a visually appealing grid of food categories (e.g., Appetizers, Main Courses, Desserts) with enticing images and descriptions.
* **Search Bar**: Located at the top for quick item searches, enhancing user accessibility.

**3. Menu Browsing**

* **Item Details Page**: When a user clicks on a menu item, they are taken to a detailed view, showcasing the item’s image, description, customization options, and pricing.
* **Filters and Sorting**: Users can filter items by category, price range, or dietary preferences, and sort by popularity or rating.

**4. Cart Interface**

* **Cart Summary**: A dedicated cart page displaying selected items, quantities, individual prices, and the total cost. Users can modify item quantities or remove items easily.
* **Checkout Button**: Clearly visible, leading to the payment and order confirmation process.

**5. Order Tracking**

* **Order Status Page**: A user-friendly interface that provides real-time updates on the order status (e.g., Preparing, Out for Delivery) with estimated delivery times.

**6. Profile Management**

* **User Profile Page**: Allows users to update personal information, view order history, and change passwords. A clean layout ensures easy navigation through the options.

**8. Responsive Design**

* **Mobile and Desktop Compatibility**: The UI is designed to be responsive, ensuring a seamless experience across different devices and screen sizes.
* **Accessibility Features**: Colour contrast, font size adjustments, and keyboard navigation support make the application accessible to all users.

This section showcases the thoughtful design and user-friendly features of your application, highlighting how the UI contributes to an engaging food ordering experience. Feel free to adjust any descriptions or add visual aids to enhance this section further.

**Backend Implementation**

This section details the backend architecture and design of the Food Ordering Web Application, built with **Spring Boot** for robust API handling, data processing, and secure interactions with the database.

**1. Database Design and Schema Overview**

* **Database Choice**: PostgreSQL serves as the relational database, storing user profiles, food items, orders, and many more.
* **Schema Overview**: The database schema consists of tables such as Users, menu item, Orders, Order Items, ingredient item, cart item and many more, each with specific columns that reflect essential attributes and relationships.

**2. API Endpoints**

* **Endpoint Structure**: RESTful API endpoints are structured to manage the application’s core functions, with endpoints for:
  + **User Authentication** (/auth/signin, /auth/signup)
  + **Menu Browsing** (/menu/items, /menu/item/{id})
  + **Cart Management** (/cart/add, /cart/{id}/remove)
  + **Order Processing** (/order/create, /order/{id}/status)
  + **Order Status** (/order/{id}/status)

**3. Controller Classes Overview**

* **Role of Controllers**: Controllers manage API requests and responses, acting as the entry points for user actions.
* **Controller Classes**: Each module has a dedicated controller:
  + **AuthController**: Manages user registration, login, and jwt token generation.
  + **CartController**: Handles cart item retrieval and search functionality.
  + **OrderController**: Manages order creation, updates, and tracking.
* **Exception Handling**: Controllers handle exceptions to provide meaningful error messages to the client, enhancing user experience and debugging efficiency.

**4. Service Classes and Business Logic**

* **Purpose of Services**: Service classes contain business logic, coordinating between controllers and data access layers.
* **Core Services**:
  + **UserService**: Manages user-related actions like registration and profile updates.
  + **CartService**: Handles cart item listings, filtering, and searching.
  + **OrderService**: Processes order placement, updates, and tracking.
* **Dependency Injection**: Services are injected into controllers to promote modular, testable, and reusable code.

**5. Repository/DAO Layer (Database Queries)**

* **Spring Data JPA**: The Repository layer utilizes Spring Data JPA to abstract database interactions, enabling easy access to PostgreSQL tables.
* **Repositories**:
  + **User Repository**: Manages CRUD operations for user data.
  + **Cart Repository**: Handles queries cart.
  + **Order Repository**: Manages order creation, status updates, and history.
* **Custom Queries**: Some complex queries, like filtering items based on user preferences or fetching recent orders, are customized with JPQL to enhance query efficiency.

This detailed overview of backend components explains how each part of the system functions, from user actions and data flow to secure data handling and management. Adding database schemas, sample API request-response structures, and code snippets for each layer will further illustrate the backend setup and ensure complete documentation.

**Frontend Implementation**

This section covers the architecture and design of the frontend built with **React** to provide an intuitive and responsive user experience. The frontend connects to the backend APIs and manages UI state for smooth and consistent interactions.

**1. React Component Structure and Hierarchy**

* **Component Design**: The application follows a component-based architecture, where each UI element is modular, reusable, and maintainable. Major components include:
  + **Header and Navigation**: Manages site navigation, links to pages, and user login status display.
  + **MenuList and MenuItem**: Displays a list of available food items, allowing users to select items and view details.
  + **Cart and Checkout**: Manages selected items, quantity adjustments, and order submission.
  + **Order History and Tracking**: Shows past orders and allows users to track current orders.
* **Component Hierarchy**: Parent-child relationships are structured to ensure data flows correctly between components. For example, App serves as the root component, encapsulating navigation, which in turn renders pages like the menu, cart, and order tracking.

**2. State Management (e.g., Redux or Context API)**

* **Global State**: **Redux** (or **Context API** if applicable) manages the global state, handling essential data such as user login status, cart contents, and order details. This ensures that shared data is accessible across components.
* **Component State**: Local states are managed within components for individual input fields, loading states, and temporary data like modals.
* **Example**: The cart contents are stored in Redux, allowing both MenuItem and Cart components to access and update the cart seamlessly as items are added or removed.

**3. API Calls and Data Fetching**

* Using Axios: API requests are handled with **Axios** for fetching data from the backend. Axios is configured globally to include necessary headers like authentication tokens
* **Data Fetching**:
  + **Initial Load**: Fetches menu items and user details on login.
  + **Dynamic Calls**: API calls are made for actions like adding items to the cart, submitting orders, and retrieving order status.
* **Example**: When a user adds an item to the cart, an API call is triggered to update the backend and immediately reflects in the frontend’s state, ensuring consistency between the client and server.

**4. Error Handling and Validation**

* **Client-Side Validation**: Validates user inputs like email format, password strength, and required fields before submission.
* **API Response Handling**: Displays error messages for failed API calls, such as login errors or payment failures, providing feedback to users.
* **Error Display**: Errors are shown near the respective input fields or as pop-up notifications, depending on the context, to improve user experience.
* **Example**: During checkout, validation ensures all required fields are completed, and errors are displayed if the backend rejects a request due to missing information or invalid input.

This overview explains how the frontend components interact, manage state, and communicate with the backend, while ensuring a responsive and error-free experience for users. Screenshots and component flow diagrams can further clarify these points in the documentation.

**Security Measures**

This section covers the strategies and technologies used to secure the Food Ordering Web Application, ensuring safe and reliable access for users.

**1. Authentication and Authorization**

* **JWT (JSON Web Token) Authentication**: User authentication is managed using JWT, where tokens are generated upon successful login and passed with each API request to verify user identity.
  + **Token Generation**: Upon login, and at the time of registration the server generates a JWT, which is returned to the client and stored securely.
  + **Token Verification**: For each protected API request, the server verifies the token to validate the user’s session, ensuring only authenticated users access sensitive routes.
* **Role-Based Authorization**: User roles (e.g., user, admin, restaurant\_owner) are assigned to control access levels within the application.
  + **User Access**: General users have access to ordering features and personal order history.
  + **Admin Access**: Admins have additional permissions to manage menu items, view all orders, and track system status.
  + **Restaurant\_Owner**: Can add his restaurant to the website, can add food and there all information along with all necessary details of the restaurant.

**2. Session Management**

* **Session Expiry**: JWT tokens have a set expiration time, requiring users to reauthenticate after a specified period to minimize unauthorized access risks.
* **Refresh Tokens**: Optionally, refresh tokens are used to renew JWTs, enhancing session persistence without compromising security.

**3. Data Encryption**

* **Password Hashing**: User passwords are securely hashed before storage in the database using industry-standard algorithms (e.g., bcrypt), ensuring that actual passwords are never stored as plain text.
* **HTTPS Encryption**: Data transmitted between the client and server is encrypted via HTTPS, protecting sensitive information (such as login credentials) from interception.

**4. Protection Against Common Vulnerabilities**

* **SQL Injection Prevention**: The use of parameterized queries or an ORM (like Spring Data JPA) prevents SQL injection attacks by ensuring that database queries are safely handled.
* **Cross-Site Request Forgery (CSRF)**: CSRF tokens are implemented to verify that requests originate from trusted sources, adding another layer of security against unauthorized actions
* This comprehensive approach to security ensures that user data is protected, sessions are properly managed, and vulnerabilities are minimized, supporting a secure environment for users and administrators alike.

**Testing**

Thorough testing ensures that all application features and interactions work as expected, providing a smooth and reliable user experience.

**1. API Testing**

* **Objective**: API testing ensures that each backend endpoint operates as intended, responding correctly to valid and invalid requests.
* **Tools Used**: Postman is used to test and verify the RESTful API endpoints.
* **Testing Process**:
  + **Endpoint Validation**: Each API endpoint (e.g., user authentication, order submission, menu retrieval) is tested for correct responses, including status codes, headers, and data payload.
  + **Data Validation**: Testing covers data integrity by ensuring that inputs and outputs match the expected formats and data structures.
  + **Error Handling**: Tests simulate invalid requests (e.g., missing fields, incorrect data types) to verify that the API returns appropriate error messages and status codes, enhancing robustness.

**2. Integration Testing**

* **Objective**: Integration testing verifies that various components, such as the backend services, frontend UI, and database, work cohesively as a single system.
* **Testing Approach**:
  + **User Journey Tests**: Simulated user actions (like registration, login, order placement) are tested to ensure that components interact correctly, data flows smoothly, and any dependencies perform as expected.
  + **Data Consistency Checks**: Integration tests confirm that data between the frontend and backend remains consistent and accurately reflects user actions (e.g., items added to a cart, order history).
  + **Edge Cases**: Testing includes edge cases and boundary conditions, such as simultaneous order submissions or rapid data updates, ensuring the application handles these cases gracefully.

These testing processes ensure that the application performs reliably across various scenarios and maintains data integrity, providing a secure and efficient experience for both end users and administrators.

**Deployment**

This section outlines the deployment strategy and steps taken to host the **FoodSphere** Food Ordering Web Application, ensuring that it is accessible, scalable, and production-ready.

**Deployment Environment**

To achieve a seamless and reliable deployment, we utilized modern cloud-based platforms that support quick CI/CD pipelines and provide built-in scalability:

* **Backend Deployment Platform:** **Railway**
  + A developer-friendly platform that simplifies deployment of backend applications with support for containerized services, PostgreSQL integration, and environment configuration.
* **Frontend Deployment Platform:** **Vercel**
  + A powerful frontend hosting platform optimized for React applications, offering automatic builds, deployments from GitHub, and global CDN for performance.
* **Database:** **PostgreSQL** hosted and managed directly within **Railway**.

**Deployment Process and Steps**

**Step 1: Backend Deployment on Railway**

* **Project Initialization:** Connected the GitHub repository containing the Spring Boot backend to **Railway**.
* **Configuration:**
  + Defined necessary environment variables such as DB\_URL, JWT\_SECRET, and STRIPE\_API\_KEY.
  + Selected **PostgreSQL** as the database and initialized it through Railway’s managed service.
* **Deployment:**
  + Railway automatically built and deployed the project from the main branch.
  + Verified the REST API endpoints via Railway’s provided URL and integrated it with the frontend.

**Step 2: Frontend Deployment on Vercel**

* **Project Initialization:** Linked the GitHub repository containing the React frontend to **Vercel**.
* **Configuration:** Environment variables such as REACT\_APP\_BACKEND\_URL were set in the Vercel dashboard to enable API communication.
* **Deployment:** Vercel auto-deploys every commit from the main branch, providing seamless CI/CD, The app is served via Vercel’s edge network, ensuring fast loading times and availability.

**User Guide**

**1. Getting Started**

* **Accessing the Application**: Visit the application’s URL (e.g., [app-name].com) in a web browser.
* **User Registration**: New users can sign up by clicking the **Register** button, filling in required details (e.g., name, email, password), and submitting the form.
* **Logging In**: Existing users can log in with their credentials via the **Login** button. Password resets are available if needed.

**2. User Features**

* **Browsing the Menu**: After logging in, users can view the list of available food items on the main menu screen. Categories, descriptions, and prices are displayed for each item.
* **Ordering Food**:
  + Select desired items and customize them if options are available (e.g., quantity, add-ons).
  + Add items to the cart by clicking **Add to Cart**.
* **Viewing Cart and Checkout**:
  + Access the **Cart** from the navigation bar to review selected items.
  + Make adjustments as necessary and proceed to **Checkout**.
  + Enter payment details to complete the order. A confirmation message will display upon successful order placement.
* **Order Tracking**: Users can check the **Order History** section to view the status of current and past orders, with updates like "In Progress," "Out for Delivery," or "Delivered."

**3. Admin Guide**

* **Admin Login**: Authorized admins can log in via the **Admin Login** page with their credentials.
* **Managing Menu Items**:
  + Navigate to the **Menu Management** section to add, update, or remove food items.
  + Admins can update prices, descriptions, and availability status as needed.
* **Order Management**:
  + View incoming orders in real-time in the **Orders** section.
  + Update order statuses (e.g., "Preparing," "Out for Delivery") to keep customers informed.
* **User Management**:
  + Access the **User Management** section to view registered users, manage permissions, and handle account issues if needed.

**4. Frequently Asked Questions (FAQs)**

* **How to Reset Password**: Users can reset their password by clicking the **Forgot Password** link on the login page and following the instructions.
* **How to Contact Support**: For any issues, users can contact support by navigating to the **Help** or **Contact Us** section.

**5. Tips for Optimal Experience**

* **Browser Compatibility**: The application works best on modern browsers like Chrome, Firefox, and Safari.
* **Mobile-Friendly**: Access is also optimized for mobile devices, so users can order food on the go.

This user guide ensures that both users and administrators can navigate the application smoothly, with clear instructions for every major function.

**Challenges and Solutions**

**1. Challenge: Managing State Across Components**

* **Description**: With multiple user interactions (e.g., adding items to the cart, updating order status), managing the application’s state became complex.
* **Solution**: Implemented **Redux** for state management, centralizing shared data and making it easier to update and retrieve application state consistently. This approach also minimized redundant re-renders and simplified state updates across the components.

**2. Challenge: Securing User Data**

* **Description**: Ensuring secure handling of sensitive data, such as login credentials and order details, was critical, particularly given the risk of unauthorized access and data breaches.
* **Solution**: Integrated **JWT (JSON Web Token)** with **Spring Security** to enforce secure authentication and authorization. Used HTTPS for encrypted communication and implemented role-based access controls to restrict sensitive operations to authorized users only.

**3. Challenge: Efficient API Communication Between Frontend and Backend**

* **Description**: Ensuring seamless, real-time data flow between the frontend and backend without overwhelming the server was challenging, especially during peak usage.
* **Solution**: Utilized **Axios** in the React frontend to handle HTTP requests efficiently. Configured backend API endpoints to handle bulk requests and optimized query performance with Spring Data JPA, reducing the load on the server.

**5. Challenge: Deployment and Configuration**

* **Description**: Although deployment wasn't fully completed, understanding the deployment pipeline, configuring servers, and managing dependencies was initially challenging.
* **Solution**: Documented deployment steps for an Railway environment for backend and Vercel for frontend.

**6. Challenge: Database Design and Optimization**

* **Description**: Structuring the database to efficiently manage orders, user data, and menu items required careful planning to avoid redundancy and ensure fast data retrieval.
* **Solution**: Developed a relational database schema in PostgreSQL, normalized tables to reduce redundancy, and used indexes to speed up query processing. Additionally, optimized database calls in the Spring Boot backend using caching where applicable.

**Conclusion and Future Scope**

**Conclusion**

The Food Ordering Web Application successfully demonstrates the use of modern web technologies to create a seamless and efficient user experience for ordering food online. With a Spring Boot backend providing robust data management and security, paired with a React-based frontend for responsive and interactive user interfaces, the application meets the objectives of simplifying the ordering process and enhancing user convenience. Through implementing a structured architecture and handling complex data flows effectively, this project showcases best practices in full-stack development.

**Future Scope**

1. **Real-Time Order Tracking**: Implement real-time updates using WebSocket or server-sent events to keep users informed of their order status, from preparation to delivery.
2. **AI Recipe Chatbot**: Integrate a chatbot powered by AI to provide users with personalized recipe suggestions based on their preferences, enhancing engagement and user experience.
3. **Mobile Application Development**: Develop a dedicated mobile app to make the platform more accessible for users on the go, providing a seamless experience across devices.
4. **Email and Notification System**: Implement an email sender and in-app notifications to keep users updated on order confirmations, status changes, and promotions.
5. **Enhanced Payment Options**: Integrate additional payment gateways, digital wallets, and installment-based payment options to offer greater flexibility for users during checkout.

In conclusion, while the project has achieved its initial goals, there is a significant potential for future improvements that could elevate the application to an even more comprehensive, user-friendly platform. These additions would ensure continued relevance and enhance user satisfaction in a competitive market.

**References**

To build this amazing food ordering application, we utilized a variety of resources, tools, and technologies that significantly contributed to the development process. Below is a summary of the key references that guided us:

1. **Official Documentation**: Comprehensive guides from the official websites of the technologies used provided in-depth information and best practices.
   * **Spring Boot**: The [Spring Boot Documentation](https://spring.io/projects/spring-boot) offered detailed insights into setting up the backend, configuring REST APIs, and managing dependencies.
   * **React**: The React Documentation served as a foundational resource for understanding component-based architecture and state management.
   * **Redux**: The Redux Documentation helped us grasp effective state management techniques across our application.
   * **Tailwind CSS**: The [Tailwind CSS Documentation](https://tailwindcss.com/docs) was essential for implementing utility-first CSS styling and ensuring responsive design.
2. **Tutorials and Articles**: Various online articles and tutorials provided step-by-step instructions and examples that were instrumental during development.
   * **JWT Authentication**: An article on JWT Authentication in Spring Security was invaluable for implementing secure authentication features.
   * **Building REST APIs**: Resources like Building RESTful APIs with Spring Boot offered practical examples and explanations for developing our backend services.
   * **Asynchronous Data Fetching**: A guide on Handling Asynchronous Data Fetching in React with Axios helped streamline API calls and data management in the frontend.
3. **Development Tools**: Various tools and software played a crucial role in our development workflow.
   * **IntelliJ Idea & Vscode**: Use IntelliJ Idea for writing Backend code using java, spring boot and use Vs Code for building the frontend using react with other technology.
   * **Postman**: Essential for API testing and validation, ensuring that our endpoints functioned correctly.
   * **Git**: Used for version control, facilitating collaboration and tracking changes throughout the development process.
4. **AI Tools**: We leveraged AI tools like ChatGPT and Gemini to brainstorm ideas, troubleshoot coding issues, and enhance productivity throughout the development process. These tools provided quick answers, coding examples, and optimization suggestions that improved our workflow.
5. **Video Tutorials**: Numerous YouTube videos offered visual guidance and practical demonstrations on specific aspects of our tech stack, from setting up the development environment to implementing advanced features.

By combining these resources, we ensured a well-rounded development approach that incorporated best practices and innovative solutions, ultimately leading to the successful creation of our food ordering application.