**Project Title: Cartify (Full-Stack Online Retail Application)**

**Objective:**  
Build a complete full-stack online retail application to manage an e-commerce store’s products, users, orders, and shopping cart. The system will enable users to browse products, add items to a shopping cart, securely log in to their accounts, and complete a checkout process with secure payment integration. The backend will handle all business logic, data management, and security, while the frontend will provide an interactive, user-friendly interface.

**Requirements:**

**1. Database Design (PostgreSQL)**

**Tables to Design:**

1. **Products:** Stores product information for the catalog.
   * **Fields:** product\_id (Primary Key), name, description, price, stock\_level, category\_id (Foreign Key), discount, is\_featured, created\_at, updated\_at.
   * **Business Rules:**
     + Only products with stock\_level > 0 should appear for purchase unless configured otherwise.
     + Products with zero stock should display as "Out of Stock."
2. **Categories:** Manages product categories to normalize and categorize products.
   * **Fields:** category\_id (Primary Key), name.
3. **Customers:** Stores registered customer information.
   * **Fields:** customer\_id (Primary Key), username, email\_address, password\_hash, shipping\_address, billing\_address, created\_at, updated\_at.
   * **Business Rules:**
     + Passwords should be stored securely using BCrypt hashing.
4. **Orders:** Records each purchase made by a customer.
   * **Fields:** order\_id (Primary Key), customer\_id (Foreign Key), order\_date, order\_total, status, shipping\_date, estimated\_delivery\_date, created\_at, updated\_at.
   * **Business Rules:**
     + Only active orders are accessible to the customer, with status tracking (e.g., "Pending," "Shipped," "Delivered").
     + Orders should only be created if items are available in stock.
5. **Order\_Items:** Tracks individual items within an order.
   * **Fields:** order\_item\_id (Primary Key), order\_id (Foreign Key), product\_id (Foreign Key), quantity, unit\_price, discounted\_price.
   * **Business Rules:**
     + Reflects final prices, accounting for discounts on each item.
6. **Shopping\_Cart:** Holds items currently in the customer’s cart.
   * **Fields:** cart\_id (Primary Key), customer\_id (Foreign Key).
   * **Business Rules:**
     + Cart contents should expire after a specified period of inactivity.
7. **Cart\_Items:** Represents items in a customer’s shopping cart.
   * **Fields:** cart\_item\_id (Primary Key), cart\_id (Foreign Key), product\_id (Foreign Key), quantity, unit\_price.
8. **Payments:** Logs payment details for orders.
   * **Fields:** payment\_id (Primary Key), order\_id (Foreign Key), amount, status, payment\_date, transaction\_id.
   * **Business Rules:**
     + Payment status should match order status.
     + Failed payments should be removed from the order flow.
9. **Reviews:** Stores customer reviews for products.
   * **Fields:** review\_id (Primary Key), product\_id (Foreign Key), customer\_id (Foreign Key), rating, comment, review\_date.
   * **Business Rules:**
     + Only customers who purchased the product can leave reviews.
     + Product pages should display average ratings.

**2. Backend (Spring Boot)**

**Endpoints and Functional Requirements:**

1. **Product Management:**
   * **Endpoint:** /api/products
   * **Operations:**
     + GET /: Retrieve all products or filter by category, with search by price range, rating, and stock availability.
     + GET /:id: Retrieve detailed information for a specific product by product\_id.
     + GET /featured: Retrieve featured products.
     + GET /:id/reviews: Retrieve reviews for a product.
     + POST (Admin only): Add new products.
     + PUT (Admin only): Update existing product details.
     + DELETE (Admin only): Remove a product.
   * **Business Rules:**
     + Only admin users can create, update, or delete products.
     + Updates to product stock should reflect in active carts with that product.
2. **User Account Management:**
   * **Endpoint:** /api/users
   * **Operations:**
     + POST /register: Register a new user account with hashed passwords.
     + POST /login: Authenticate a user and return a JWT session token.
     + GET /profile: Retrieve user details (authenticated users only).
     + PUT /profile: Update user details.
     + POST /logout: Invalidate session tokens.
   * **Business Rules:**
     + Account lockout after a configurable number of failed login attempts.
     + Tokens expire after a set duration and require re-authentication.
3. **Shopping Cart:**
   * **Endpoint:** /api/cart
   * **Operations:**
     + POST /add: Add an item to the cart.
     + DELETE /remove: Remove an item from the cart.
     + PATCH /update: Update quantities without re-adding items.
     + GET /view: View the current items in the cart.
     + POST /clear: Clear all items in the cart.
   * **Business Rules:**
     + Cart items expire after a set duration if not purchased.
     + Display updated totals, including discounts.
4. **Order Processing:**
   * **Endpoint:** /api/orders
   * **Operations:**
     + POST /checkout: Process the current cart as an order, deduct stock, calculate totals.
     + GET /history: Retrieve order history for a specific user.
     + GET /:order\_id: Retrieve details of a specific order.
   * **Business Rules:**
     + Orders only created if payment is successful and cart has items.
     + Status updates to "Pending," "Shipped," "Delivered" should be tracked.
5. **Security:**
   * **Token-Based Authentication:**
     + Implement JWT for secure token-based sessions with a set expiration.
   * **Authorization:**
     + Use role-based access control for user (Customer, Admin) roles.
   * **Password Hashing:**
     + Securely hash passwords with BCrypt.

**3. Frontend (React)**

**Functional Components and Pages:**

1. **Home Page:**
   * Display featured products with search and filter options by category, price, rating, and stock availability.
2. **Product Details Page:**
   * Show detailed product information, including description, price, stock level, and average rating.
   * Display user reviews and allow verified buyers to submit reviews.
   * Allow users to add products to the shopping cart.
3. **Shopping Cart Page:**
   * Show cart items with options to adjust quantities, remove items, and view the cart total.
   * Include estimated taxes and shipping fees, and display recommendations based on cart contents.
4. **User Authentication:**
   * Provide registration and login forms, with secure token-based sessions.
   * Show a welcome message with the user's name upon login.
5. **Checkout Page:**
   * Display a cart summary, including item breakdown, discounts, taxes, and shipping.
   * Allow users to securely enter payment and shipping information.
6. **Order History Page:**
   * Display past orders with details such as order items, date, status, and tracking information.
7. **Styling:**
   * Use Material-UI or Bootstrap for styling and layout consistency.

**4. Bonus Features**

1. **Secure Payment Integration:**
   * Use a payment gateway (like Stripe or PayPal) for handling transactions.
   * Implement webhook notifications to update order status upon payment success.
2. **Product Recommendations:**
   * Display personalized product recommendations based on viewing and purchase history.
3. **Email Notifications:**
   * Use a service like SendGrid to send emails for order confirmation, shipment updates, and other notifications.

**Technical Approach**

1. **Backend Setup:**
   * Create a Spring Boot application, connect it to PostgreSQL, and implement REST APIs.
   * Ensure endpoints are secured with Spring Security and JWT.
2. **Frontend Setup:**
   * Build the frontend in React, create components for each page, and use axios or fetch for API calls.
3. **Testing Strategy:**
   * Use Postman to test backend APIs.
   * Implement unit and integration tests for backend features, and front-end tests with Jest and React Testing Library.
   * Use Cypress for end-to-end testing of the main user workflows.

**Expected Outcome:**

By the end of this project, you will have developed a fully functional, scalable, and secure online retail application. This project showcases full-stack capabilities and is ideal for your portfolio, demonstrating skills in React, Spring Boot, and PostgreSQL.