Low Level Design (LLD)

Shopping Cart

Ayushi Raj

Abstract

This project abstract outlines the development of a Shopping Cart Application using React, a popular JavaScript library for building dynamic web interfaces. The application provides users with features such as product browsing, cart management, secure checkout, and real-time updates. By emphasizing responsive design, state management, and performance optimization, this project serves as a practical example of how to leverage React to create a modern, user-friendly e-commerce solution. Whether you're a novice or an experienced developer, this project offers valuable insights into building effective web applications with React.

1 Introduction

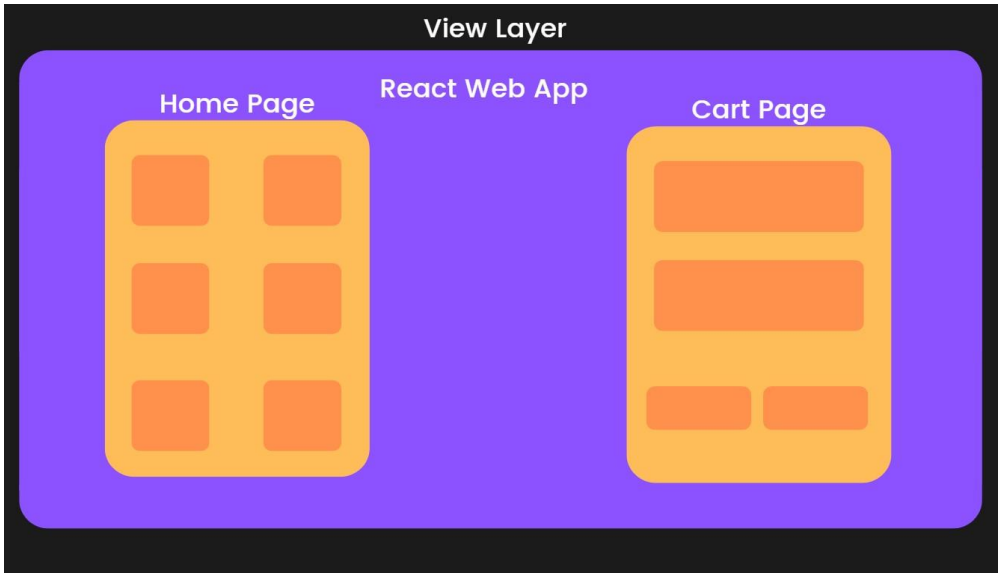
1.1 Why this Low-Level Design Document?

In the development process of our Shopping Cart Application using React, this Low-Level Design Document plays a pivotal role in guiding the implementation phase. While the abstract provided an overview of the project's objectives and key features, this document delves deeper into the technical intricacies of our application. It serves several crucial purposes:

1. **Technical Blueprint:** This document serves as the technical blueprint for the project, outlining the architecture, component interactions, and data flow within our React-based application. It provides a granular understanding of how different parts of the application work together.
2. **Development Reference:** For developers and team members involved in the project, this document acts as a reference point. It provides detailed information on the coding standards, patterns, and best practices to be followed during the implementation phase.
3. **Troubleshooting Aid:** In case of issues or bugs during development, this document can be used as a troubleshooting aid. It helps identify potential sources of problems and provides insights into how different components are expected to function.
4. **Enhancement Planning:** As development progresses, this document becomes a valuable asset for planning enhancements and new features. It provides a clear picture of the existing architecture, making it easier to identify areas for improvement.
5. **Collaboration and Communication:** Effective collaboration among team members is essential for project success. This document facilitates communication by offering a shared understanding of the application's low-level design, ensuring that everyone is on the same page.
6. **Quality Assurance:** Quality assurance and testing teams can use this document to create test cases, ensuring that the application meets the specified requirements and functions as intended.

By providing this Low-Level Design Document, we aim to enhance transparency, efficiency, and the overall quality of our development process. It's a vital resource that complements the project's

overarching goals and ensures that our Shopping Cart Application with React is not only feature-rich but also robust and maintainable.



2. System Architecture

2.1 High-Level Architecture

The Shopping Cart Application with React is structured as a single-page web application (SPA) that follows the client-server architecture. It utilizes React for the client-side user interface, Node.js with Express.js for the serverside, and a relational database (e.g., PostgreSQL) to store product and user data.

3. Component Design

3.1 App Component

Purpose: The App component serves as the main entry point for the application.

Interfaces: It encapsulates the entire application structure, including routing, navigation, and context setup.

Technologies/Libraries Used: React, React Router.

3.2 Navbar Component

Purpose: The Navbar component is responsible for rendering the application's navigation bar.

Interfaces: It provides navigation links for users to access different sections of the application.

Technologies/Libraries Used: React.

3.3 Shop Component

Purpose: The Shop component represents the main shopping page where users can browse and select products.

Interfaces: It fetches product data from the server and displays it to users.

Technologies/Libraries Used:** React, Context API.

3.4 Cart Component

Purpose: The Cart component manages the user's shopping cart, allowing them to add or remove items and view the total price.

Interfaces: It interacts with the shopping cart context to retrieve and update cart data.

Technologies/Libraries Used: React, Context API, LocalStorage (for temporary cart storage).

4. Data Flow and Workflow

4.1 Data Flow

1. Users interact with the product catalog by browsing and adding items to their carts.
2. The client-side application sends requests to the server for product data and cart updates.
3. The server processes requests, communicates with the database, and sends responses back to the client.
4. The client updates its user interface based on the server's responses.

4.2 Workflow

User Adding a Product to the Cart:

1. User clicks "Add to Cart" on a product in the Shop component.
2. The Shop component sends a request to the Cart component to add the item to the cart.
3. The Cart component updates the shopping cart context with the new item.
4. The Cart component sends a request to the server to update the user's cart in the database.
5. The server processes the request, updates the database, and sends a success response.

6. The Shop component updates the UI to reflect the changes in the cart.

5. Error Handling and Logging

5.1 Error Handling

Errors in the client and server code are handled using try-catch blocks. Relevant error messages are sent as responses to assist with debugging and user notifications.

5.2 Logging

Server-side logging is implemented using a logging library (e.g., Winston) to record errors and application events for debugging purposes.

6. Performance Optimization

6.1 Strategies

Caching: Product data is cached on the client side to reduce unnecessary requests to the server.

Lazy Loading: Images and resources are loaded asynchronously to enhance page load times.

6.2 Performance Testing

Performance testing was conducted using tools such as Lighthouse and GTmetrix to ensure that the application meets optimal loading times and responsiveness.