FlashMob - Technical Design Document

Contents

1 Introduction and System Overview

1.1 Introduction

FlashMob is a modern e-commerce platform designed to enable users to explore, purchase, and manage products across a wide range of categories. The platform offers a seamless shopping experience with features like real-time inventory updates, secure user authentication, order management, and social interactions such as product reviews and ratings.

This design document provides a comprehensive overview of the project's architecture, components, technology stack, and design decisions. It aims to serve as a guide for developers, stakeholders, and contributors.

1.2 System Overview

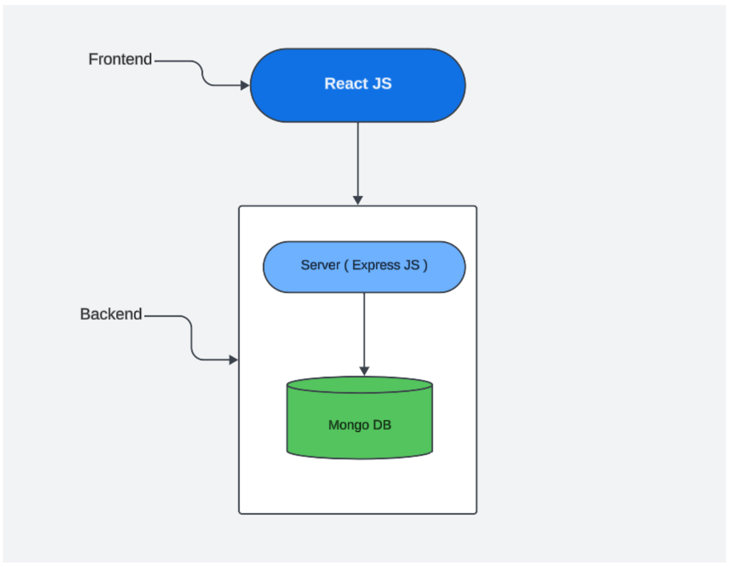
FlashMob is built using the MERN stack (MongoDB, Express.js, React, Node.js), utilizing modern web development practices. The application is structured to provide scalability, maintainability, and a responsive user experience across devices.

2 Architecture

2.1 High-Level Architecture

The system follows a three-tier architecture, comprising:

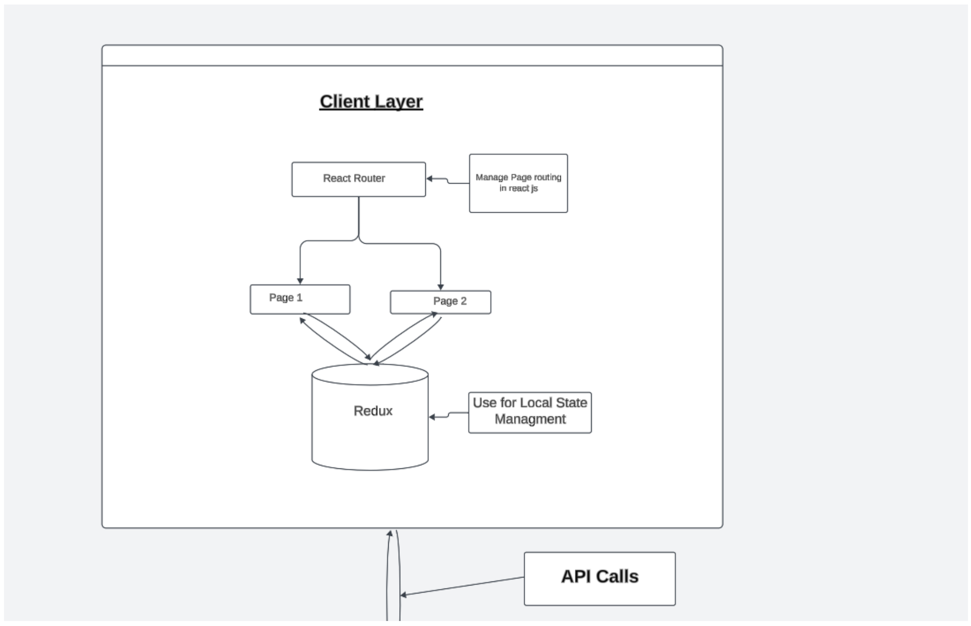
1. **Frontend:** Developed with React.js, responsible for the client-side user interface and interactions.
2. **Backend API:** Built with Express.js and Node.js, handling server-side logic, API endpoints, authentication, and business logic.
3. **Database:** Utilizes MongoDB for storing user data, products, orders, and related information.



3 Frontend Design

3.1 Technologies Used

- React.js: JavaScript library for building user interfaces.
- React Router DOM: Handling client-side routing.
- Tailwind CSS: Utility-first CSS framework for styling.
- Vite: Build tool for faster development.
- ESLint: Linting utility to maintain code quality.


Client Layer
React Router
Manage Page routing in react js
Page 1
Page 2
Redux
Use for Local State Managment
API Calls

3.2 Project Structure

- `main.jsx`: Entry point of the React application.
- `App.jsx`: Main application component.
- `components/`: Reusable UI components.
- `pages/`: Page components corresponding to routes.
- `router.jsx`: Defines client-side routing.
- `lib/`: Utility functions.
- `assets/`: Contains static assets such as images and icons.
- `index.css`: Global CSS and Tailwind directives.
- `public/`: Publicly accessible files.

3.3 Routing

Implemented using React Router:

- `/`: Home page showcasing featured products.
- `/signin`: User login page.
- `/signup`: User registration page.
- `/filter-products`: Product listing page with filtering and sorting options.
- `/product/:productId`: Product details page.

3.4 State Management

- **Local State:** Managed using React's `useState` and `useEffect` hooks.
- **Global State:** Managed using `Redux Toolkit` for scalability.
- **Authentication State:**
  - Stored in `localStorage`.
  - Accessed via custom hooks for modularity.
- **Data Fetching:**
  - Utilizes the Fetch API.
  - Implements error handling and loading states.

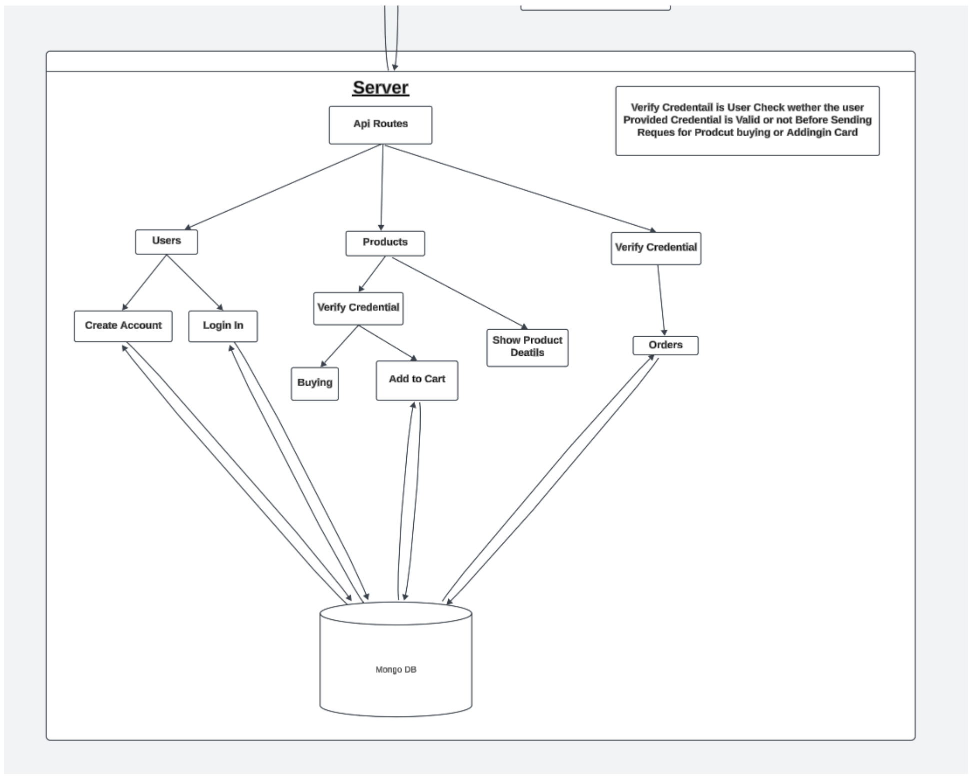
3.5 Key Components

- **NavBar:**
  - Displays navigation links and a search bar.
  - Updates options dynamically based on user authentication state.
- **ProductCard:**
  - Displays product information such as name, price, and image.
  - Includes actions like `Add to Cart`.
- **CartSummary:**
  - Shows a summary of the cart items.
  - Calculates total price dynamically.
- **OrderHistory:**
  - Displays a list of past orders with status and total amounts.

4 Backend Design

4.1 Technologies Used

- Node.js: JavaScript runtime environment.
- Express.js: Web application framework for building APIs.
- MongoDB: NoSQL database for data storage.
- Mongoose: ODM (Object Data Modeling) library for MongoDB.
- JWT: JSON Web Tokens for authentication.
- bcrypt: Library for hashing passwords.
- Nodemon: Tool for automatically restarting the server during development.


Server
Api Routes
Verify Credentail is User Check wether the user Provided Credential is Valid or not Before Sending Reques for Prodcut buying or Addingin Card
Users
Products
Verify Credential
Create Account
Login In
Verify Credential
Show Product Deatils
Orders
Buying
Add to Cart
Mongo DB

4.2 Project Structure

- `index.js`: Entry point of the server application.
- `middelwares/`: Contains middleware functions, including authentication.
- `models/`: Mongoose schemas for User, Product, and Cart models.
- `routes/`: Defines API endpoints for authentication, users, products, and cart.
- `util/`: Utility functions for various operations, including async handlers.

4.3 API Design

The backend exposes RESTful API endpoints categorized under:

- **Authentication (/api/users)**
  - POST `/register`: User registration.
  - POST `/login`: User login and JWT token issuance.
  - GET `/currentUser`: Retrieve current user profile.
  - POST `/logout`: User logout.
- **Products (/api/product)**
  - GET `/all`: Fetch a list of products with pagination and filtering options.
  - GET `/:productId`: Retrieve a specific product's details.
  - GET `/sale`: Fetch products on sale.
  - GET `/new`: Fetch new products.
- **Cart (/api/cart)**
  - POST `/add`: Add a product to the cart.
  - GET `/productIsPresent`: Check if a product is in the cart.
  - GET `/all`: Fetch all cart items for a user.

4.4 Database Schema

User Model (User.js):

- Fields:
  - `username` (`String, required`): User's full name.
  - `email` (`String, required, unique`): User's email address.
  - `password` (`String, required`): Hashed password.
  - `phonenumber` (`Number, required`): User's phone number.
- Indexes:
  - Unique index on `email`.
- Relations:
  - A user can have multiple cart items.

Product Model (Product.js):

- Fields:
  - `title` (`String, required`): Name of the product.
  - `description` (`String, required`): Product description.
  - `price` (`Number, required`): Product price.

  - `category` (`String`): Category of the product.
  - `discount` (`Number`): Discount on the product.
  - `onSale` (`Boolean, default: false`): Whether the product is on sale.
  - `createdAt` (`Date, default: Date.now`): Timestamp of creation.

- Indexes:
  - Index on `category` for filtering.
  - Index on `price` for sorting.

Cart Model (Cart.js):

- Fields:
  - `user` (`ObjectId, required`): References the User model.
  - `product_list` (`Array`): List of products in the cart.

4.5 Middlewares

- **Authentication Middleware** (`user.middelware.js`)
  - Validates JWT tokens sent in the Authorization header.
  - Attaches the authenticated user's information to the request object.
  - Protects routes that require authentication.

5 Security Considerations

- **Authentication:**
  - Securely implemented using JWT tokens.
  - Tokens are stored in `httpOnly` cookies for added security.
- **Password Security:**
  - Passwords are hashed using `bcrypt`.
  - Enforces strong password policies during registration.
- **Authorization:**
  - Role-based access control (RBAC) for admin functionalities.
  - Sensitive endpoints are protected using middleware.
- **CORS Configuration:**
  - Configured to allow requests only from trusted origins.
  - Strict headers are set for allowed methods and credentials.
- **Input Validation:**
  - All user inputs are sanitized to prevent injection attacks.
  - Validation rules are enforced on both client and server sides.

6 Deployment Plan

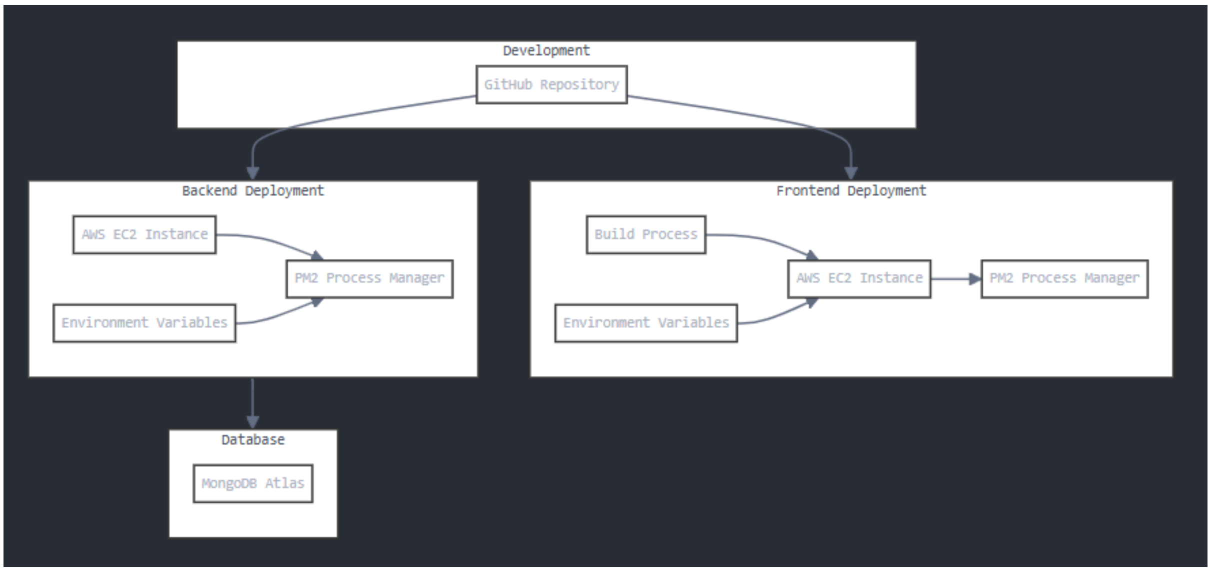
6.1 Environment Setup

- **Backend Environment Variables:**
  - `PORT`: Port number for the Node.js server.
  - `Mongoose_URL`: MongoDB connection URI.
  - `DB_NAME`: Database name.
  - `ACCESS_TOKEN_SECRET`: Secret key for signing JWTs.
  - `REFRESH_TOKEN_SECRET`: Secret key for signing refresh tokens.
- **Frontend Environment Variables:**
  - `VITE_API_URL`: Base URL for the backend API.

6.2 Deployment Steps

1. **Backend Deployment:**
   - Host on platforms like AWS, Heroku, or DigitalOcean.
   - Use `PM2` for process management.
2. **Frontend Deployment:**
   - Build using `npm run build`.
   - Host on Netlify, Vercel, or any static file hosting service.
3. **Database:**
   - Use managed MongoDB services like Atlas.
   - Enable IP whitelisting and backups.
4. **Domain and SSL:**
   - Configure custom domains and SSL certificates.
5. **CI/CD Pipelines:**
   - Automate deployments using GitHub Actions or Jenkins.

6.3 Deployment Flowchart



7 Future Enhancements

7.1 Technical Improvements

- Switch to TypeScript for better type safety.
- Integrate WebSockets for real-time updates (e.g., inventory changes).

7.2 Feature Enhancements

- Add wishlist functionality.
- Enable real-time chat support for customers.
- Implement predictive search suggestions.
- Provide analytics dashboards for users and admins.

8 Conclusion

FlashMob is a scalable, secure, and user-friendly e-commerce platform. With its robust architecture and feature-rich design, it aims to provide an exceptional shopping experience for users and comprehensive management tools for administrators.