9530

St. MOTHER THERESA ENGINEERING COLLEGE

COMPUTER SCIENCE ENGINEERING

NMID:EDA19B769EC5D7A864642ADBEA834557

REG NO: 953023104114

DATE:15-09-2025

Completed the project named as

Phase 2

FRONT END TECHNOLOGY

SINGLE PAGE APPLICATION

SUBMITTED BY:

SIJONE. J

9344472449

Thesis on Single Page Application (SPA) Design

1. Introduction

A Single Page Application (SPA) is a modern web application architecture where the entire application runs within a single HTML page. Instead of reloading the page on each request, only the required components and data are updated dynamically using client-side rendering. This approach provides faster navigation, improved user experience, and reduced server load.

The objective of this thesis is to design and document a SPA with clear consideration of technology stack, UI structure, API schema, data handling mechanisms, and modular architecture.

2. Technology Stack Selection

- Frontend:
• Framework: React.js
• Styling: Tailwind CSS
• Routing: React Router
• State Management: Redux Toolkit / Context API

- Backend:
• Framework: Node.js with Express.js
• Database: MongoDB

- Other Tools:
• Axios
• JWT
• Docker

3. UI Structure & API Schema Design

UI Structure

The SPA will follow a component-based architecture, with reusable and independent components.

- Layout Components: Header, Sidebar, Footer
- Feature Components:
• Dashboard
• User Profile
• Data Management (CRUD operations)
• Authentication (Login/Signup)

API Schema (REST-based)

- Authentication APIs:
• POST /api/auth/signup – Register new user
• POST /api/auth/login – Authenticate user and return token

- User APIs:

• GET /api/users/:id – Get user details
• PUT /api/users/:id – Update profile

- Data APIs:
• GET /api/data – Fetch all records
• POST /api/data – Create a new record
• PUT /api/data/:id – Update record
• DELETE /api/data/:id – Delete record

4. Data Handling Approach

- Frontend:
• Use Redux/Context API to store global state
• Lazy loading for performance
• Axios interceptors for token handling

- Backend:
• Secure endpoints with JWT
• Validation with Joi/Yup
• Mongoose ORM for MongoDB
• Pagination and filters for scalability

5. Component / Module Diagram

```
App Root
|
Layout (Header, Nav)
|
Auth | Dashboard | Profile
| | |
Login CRUD Ops User Details
```

6. Basic Flow Diagram

User Request → React Router → Component Rendering → API Call (Axios) → Backend (Express + Node.js) → Database (MongoDB) → Response (JSON) → State Update (Redux/Context) → UI Update (Virtual DOM Rendering)

7. Conclusion

This SPA design ensures:
- Scalability (modular architecture)
- Performance (client-side rendering)
- Security (JWT, validation)
- User Experience (fast navigation, responsive UI)

The documented approach forms the foundation for implementing a production-ready SPA.