COLLEGE CODE: 9528

COLLEGE NAME: SCAD College of Engineering and Technology

DEPARTMENT: CSE

STUDENT NM ID: d99d3571e0c45cea55047bbb81583279

ROLL NO: 95282331104133

DATE: 26/09/2005

Completed the project named as phase ___ TECHNOLOGY PROJECT

NAME: SINGLE PAGE APPLICATION

SUBMITED BY,

NAME: K. Roslin Nisha

MOBILE NO:8220425705

1.Tech stack selection

For building the Single Page Application (SPA), the following technology stack is selected:

Layer	Technology	Reason for Selection
Frontend	React.js	Fast rendering with Virtual DOM, component-based structure, reusable components.
Backend	Node.js with Express.js	Lightweight, scalable, supports REST APIs, works well with JavaScript.
Routing	React Router	Client-side routing for SPA navigation without reloading pages.
State Management	Redux / Context API	Centralized state management to handle shared data efficiently.
Styling	Tailwind CSS / Bootstrap	Fast UI styling with responsive design.
Database	MongoDB	Flexible NoSQL database, easy JSON data handling.
API Format	REST API / GraphQL	Structured data fetching, better integration.
Version Control	Git / GitHub	Collaboration and version management.

2. UI Structure / API Schema Design

UI Structure

The UI of SPA will have a modular layout with reusable components:
- Header — Navigation menu
- Sidebar — Optional menu for dashboard navigation
- Main Content Area — Dynamic content loading without page refresh
- Footer — Contact information and links

Page Views: Home, About, Services, Dashboard, Contact

API Schema Design

Example API schema for a “Products” module:
GET /api/products
POST /api/products
GET /api/products/:id
PUT /api/products/:id
DELETE /api/products/:id

Product Object Structure:

```
{
 "id": "string",
 "name": "string",
 "description": "string",
 "price": "number",
 "category": "string",
 "imageUrl": "string",
 "stock": "number"
}
```

3. Data Handling Approach

Data handling in SPA requires efficient state management and API communication.

- Client-side State Management: Redux or Context API for maintaining application-wide state.
- API Calls: Axios or Fetch API for communication with backend.
- Data Caching: Store frequently used data locally for faster access.
- Error Handling: Show appropriate messages when API calls fail.

Example:

```
import axios from "axios";

const fetchProducts = async () => {
 try {
  const response = await axios.get("/api/products");
  return response.data;
 } catch (error) {
  console.error("Error fetching products", error);
 }
};
```

4. Component / Module Diagram

Here's the conceptual diagram for the SPA:

App
│
├── Header
├── Sidebar
├── Routes
│ ├── HomePage
│ ├── AboutPage
│ ├── DashboardPage
│ ├── ContactPage
└── Footer

Each page will have sub-components for modular design.

5. Basic Flow Diagram

User Interaction Flow:
User → App (React Router)
 ├── Click Navigation Link
 ├── Route changes without page refresh
 ├── Component loads
 ├── API request sent if needed
 ├── Data loaded & rendered dynamically
 ├── User interacts with page

Flow Description:
1. Initial Load → SPA loads index.html and main JavaScript bundle.
2. Navigation → React Router changes views without reload.
3. API Call → Fetches data only for that component.
4. Render Data → Data is displayed instantly in the component.