 

**COLLEGE CODE : 9623**

**COLLEGE NAME:** Amrita College Of Engineering And

Technology, Erachakulam

**DEPARTMENT :** COMPUTER SCIENCE ENGINEERING

**STUDENT NM- ID :** autcs0097

**ROLL NO :** 962323104097

**DATE :**08/09/2025

**C**ompleted **T**he **P**roject **N**amed **A**s **P**hase**\_ T**echnology

**PROJECT NAME : SINGLE PAGE APPLICATION**

**S**ubmitted **B**y**,**

**NAME:** SIVA ABINANDH.R.S(962323104097)

MOBILE NO:+91 9342416708

**Phase 2- Solution Design & Architecture**

**1. Tech Stack Selection:**

**Frontend:**

* **Framework/Library**: React (or Angular/Vue as alternatives)
* **Routing**: React Router / Angular Router / Vue Router
* **State Management**: Redux Toolkit / React Context API / Pinia (Vue)
* **Styling**: Tailwind CSS or Material UI
* **Build Tool**: Vite or Webpack

**Backend:**

* **Authentication**: JWT (JSON Web Tokens)
* **API Format**: REST (GraphQL optional for flexibility)

**Deployment:**

* **Frontend**: Netlify / Vercel
* **Backend**: AWS / Render / Heroku
* **Database Hosting**: MongoDB Atlas / AWS RDS

**2. UI Structure:**

**Pages/Views:**

1. **Home Page** – Welcome info, navigation.
2. **Login/Signup** – Authentication forms.
3. **Dashboard** – List of items (with CRUD actions).
4. **Item Details** – Single item view/edit.
5. **Profile Page** – User info, settings.
6. **404 Page** – Not Found handling.

**Component Breakdown:**

**Layout Components:**

* + Header (nav bar, logo, user menu)
  + Footer

**Auth Components:**

* + LoginForm, SignupForm

**Dashboard Components:**

* + ItemList, ItemCard, ItemForm

**Utility Components:**

* + Loader, ErrorMessage, ProtectedRoute

## **UI Component Breakdown:**

### ****Global Components**** (Always Visible):

* **Header / Navbar**: App branding, user menu, search bar, notifications.
* **Sidebar / Navigation Drawer**: Links to main features (Dashboard, Profile, etc.).
* **Footer (Optional)**: Copyright, quick links.

### ****Dynamic Components (View-Specific):****

* **Dashboard** → Stats, charts, summaries.
* **Profile Page** → User info, edit forms.
* **Products / Items Page** → List view + detail view.
* **Forms** → Login, Register, Checkout, etc.
* **Settings** → Preferences, account management.

### ****Reusable Components:****

* Buttons, Modals, Cards, Dropdowns.
* Table / List Components.
* Loader / Spinner / Skeleton UI.
* Toast Notifications.

**3. API Scheme Design**

**Authentication:**

* POST /api/auth/signup → Register user
* POST /api/auth/login → Authenticate user
* GET /api/auth/logout → End session

**User:**

* GET /api/users/:id → Fetch user profile
* PUT /api/users/:id → Update profile

**Items (Example CRUD):**

* GET /api/items → Fetch all items
* POST /api/items → Create item
* GET /api/items/:id → Fetch single item
* PUT /api/items/:id → Update item
* DELETE /api/items/:id → Delete item

**4. Data Handling Approach:**

**1. Data Flow Strategy:**

* **Unidirectional Data Flow** (React/Redux, Vuex, NgRx): Ensures predictable state changes.
* **Two-Way Binding** (Vue, Angular): Useful for forms & input-heavy apps.
* Decide based on app complexity:
  + Small → Local state (React hooks, Context).
  + Medium → Lightweight store (Zustand, Pinia, Context + Reducer).
  + Large → Full-fledged state management (Redux, Vuex, NgRx).

**2. Data Fetching & APIs:**

* Use **REST APIs or GraphQL** for communication with backend.
* Adopt **asynchronous fetching** (fetch, axios, Apollo Client).
* Implement **caching & revalidation** (React Query, SWR, RTK Query).
* Use **pagination / infinite scroll** to handle large datasets.

**3. Data Storage & Persistence:**

* **In-Memory State** → Fast UI updates (temporary).
* **Local Storage / Session Storage** → Save tokens, preferences.
* **IndexedDB / WebSQL** → For large offline-capable datasets.
* **Service Workers** → Cache responses for offline-first experience.

**4. Data Synchronization:**

* **Optimistic UI Updates** → Update UI instantly, then sync with API.
* **Polling / WebSockets** → For real-time features (chat, notifications).
* **Conflict Resolution** → Handle sync issues in offline-first apps.

**5. Data Security:**

* Encrypt sensitive data before storage.
* Never store passwords or raw tokens in localStorage (prefer HttpOnly cookies or secure storage).
* Validate & sanitize all incoming/outgoing data.

**6. Error Handling & Resilience:**

* Global error boundary to catch API/UI errors.
* Retry failed requests with exponential backoff.
* Graceful fallbacks (cached data, skeleton loaders).

**7. Performance Optimization:**

* Lazy load data-heavy components.
* Use **debouncing/throttling** for search & user input.
* Minimize over-fetching (GraphQL queries, API filters).
* Batch API requests when possible.

**5. Component/Module Diagram:**

# 1. High-Level Modules

* **Presentation Layer (UI Components):**
  + Header / Navbar
  + Sidebar / Menu
  + Main Content (dynamic views: Dashboard, Profile, Products, etc.)
  + Footer
* **State Management Layer**
  + Global Store (Redux / Vuex / NgRx / Zustand)
  + Local Component State (React hooks, Angular services)
* **Service Layer (APIs & Utilities)**
  + API Service (fetch, axios, Apollo Client)
  + Auth Service (login, JWT, session)
  + Data Transformation & Validation
  + Notification/Toast Service
* **Routing Layer**
  + Client-side Router (React Router, Vue Router, Angular Router)
  + Route Guards (protected routes for authenticated users)
* **Persistence Layer**
  + Local Storage / Session Storage
  + IndexedDB (for offline use)
  + Cache Layer (React Query, SWR)

App

│

├── Layout

│ ├── Header

│ ├── Footer

│

├── Auth

│ ├── LoginForm

│ ├── SignupForm

│

├── Dashboard

│ ├── ItemList

│ │ └── ItemCard

│ ├── ItemForm

│

├── User

│ └── Profile

│

└── Shared

├── Loader

├── ErrorMessage

└── ProtectedRoute

**6. Basic Flow Diagram:**

┌─────────────────────┐

│ User Opens App │

└─────────┬───────────┘

│

┌───────────▼────────────┐

│ Authenticated? │

└───────┬─────────┬──────┘

│Yes │No

│ │

┌──────────▼───┐ ┌──▼────────────┐

│ Show Dashboard│ │ Show Login │

└───────┬──────┘ └──────┬────────┘

│ │

┌───────────▼────────────┐ │

│ User Navigates (Router)│ │

└───────────┬────────────┘ │

│ │

┌─────────▼───────────┐ │

│ Fetch Data (API) │ │

└─────────┬───────────┘ │

│ │

┌─────────▼─────────────┐ │

│ Update State & UI │ │

└─────────┬─────────────┘ │