IBM-FE-Single Page Application

Phase 2 — Solution Design & Architecture

Tech Stack Selection

The following technology stack has been selected to ensure scalability, performance, and maintainability of the SPA:

- **Frontend Framework**: React.js for component-based architecture, fast rendering, and large community support.
- **State Management**: Redux or Context API to manage application-wide state.
- **Styling**: Tailwind CSS for utility-first, responsive, and modern UI design.
- **Backend**: Node.js with Express.js for handling APIs and business logic.
- **Database**: MongoDB (NoSQL) for flexibility and scalability of data storage.
- **Authentication**: JSON Web Tokens (JWT) for secure user sessions.
- **Deployment**: Docker containers with CI/CD pipeline (GitHub Actions/Jenkins).
- **Hosting**: Cloud-based (AWS or IBM Cloud) for reliability and global access.

UI Structure / API Schema Design

UI Structure (Proposed):
- **Landing Page**: Introductory page with navigation menu.
- **Login/Signup Page**: Authentication portal with validation.
- **User Dashboard**: Personalized content, user profile, and services.
- **Admin Dashboard**: Content management, user monitoring, analytics.
- **Error & Notification Pages**: Error handling and user-friendly messages.

API Schema Design (Tentative):
- **/api/auth/login** (POST) → Authenticate user credentials.
- **/api/auth/signup** (POST) → Register new users.
- **/api/users/{id}** (GET/PUT/DELETE) → Manage user profiles.
- **/api/content** (GET/POST/PUT/DELETE) → Handle dynamic content.
- **/api/analytics** (GET) → Provide performance and engagement insights.

Data Handling Approach

The SPA will follow a **client-server data exchange model** with the following principles:

- **State Management**: Application state maintained centrally using Redux/Context API.
- **Data Fetching**: Asynchronous API calls using Axios/Fetch API.
- **Data Security**: Sensitive data encrypted with HTTPS and JWT-based tokens.
- **Caching Strategy**: Frequently used data cached on the client for faster access.
- **Error Handling**: Standardized error responses with retry logic for failed requests.
- **Scalability**: Database designed with collections for users, content, and analytics.
- **Performance Optimization**: Use of lazy loading, code splitting, and efficient API design.

Component / Module Diagram

Component Breakdown:
- **App Component**: Root component handling routing.
- **Header & Footer**: Common UI elements across pages.
- **Auth Components**: Login, Signup, Logout modules.
- **Dashboard Components**: User Dashboard, Admin Dashboard, Analytics Panel.
- **Content Components**: Dynamic rendering of posts, data, and updates.
- **Notification Component**: Alerts and error messages.

Module Layering:
- **Presentation Layer**: React components and UI rendering.
- **Business Logic Layer**: State management and data transformations.
- **Data Access Layer**: API service functions connecting to backend.

Basic Flow Diagram

High-Level Workflow:

1. **User Access** → User lands on the SPA landing page.
2. **Authentication** → User logs in using credentials, validated via backend.
3. **Authorization** → Role-based access granted (User/Admin).
4. **Data Fetching** → UI requests content/data via APIs.
5. **State Update** → Application state updated dynamically.
6. **User Interaction** → User navigates seamlessly without page reloads.
7. **Analytics Logging** → Actions recorded for monitoring and insights.

This workflow ensures smooth navigation, secure access, and real-time content rendering.