**Full Stack Development with MERN Project Documentation format**

# 1. Introduction

* **Project Title:** **Smart SDLC -AI- Enhanced Software Development Life Cycle**
* **Team Members:**
* G K N V S Ragavendra (Code)
* Koyya Ramyasri (Documentation)
* Ambati Naveen Raju (Demo Video)

# 2. Project Overview

* **Purpose:**
* The purpose of the Smart SDLC – AI-Enhanced Software Development Life Cycle project is to revolutionize traditional software development practices by integrating artificial intelligence at each phase of the SDLC. This project aims to increase efficiency, accuracy, and agility in software development by automating routine tasks, enhancing decision-making, and reducing human error.

**Goals:**

* **Automate Repetitive Tasks:** Use AI to streamline tasks such as code generation, testing, bug detection, and documentation.
* **Improve Decision-Making:** Leverage predictive analytics and machine learning to support planning, risk assessment, and resource allocation.
* **Enhance Code Quality and Security:** Implement AI-driven tools to detect vulnerabilities and suggest improvements early in the development process.
* **Accelerate Development Cycles:** Reduce time-to-market by optimizing workflows and minimizing bottlenecks through intelligent automation.
* **Enable Continuous Feedback and Learning:** Integrate AI to learn from historical data and user behavior to improve future project outcomes.

Overall, the project seeks to create a more intelligent, adaptive, and efficient software development framework that aligns with modern technological and business needs

* **Features:**

1. **AI-Driven Requirements Analysis:**

**Functionality**: Automatically interprets natural language requirements and generates structured user stories and specifications.

**2. AI-Based Code Generation:**

**Functionality**: Converts design models or requirements into boilerplate or functional code across multiple programming languages.

**3. Smart Testing and Bug Detection:**

**Functionality**: Generates test cases, predicts bugs, performs automated testing (unit, integration, regression), and offers real-time quality feedback.

**4. Optimized CI/CD Pipelines:**

**Functionality**: Enhances continuous integration and deployment workflows using intelligent decision-making to prioritize builds and tests.

**5.** **Real-Time Monitoring and Predictive Maintenance:**

**Functionality**: Monitors application performance, detects anomalies, and predicts system failures before they occur.

# 3. Architecture

* **Frontend:**
* Framework: React (with TypeScript for type safety)
* State Management: Redux Toolkit or Zustand (for managing global application state)
* Routing: React Router DOM (for SPA navigation)
* Styling: Tailwind CSS / Styled Components (for modular styling)
* API Communication: Axios or Fetch (with middleware for auth/error handling)
* Real-Time Communication: WebSockets / Socket.io (for live monitoring and feedback)
* AI Integration Layer: Abstraction layer for calling AI-powered backend services (via REST or GraphQL)
* Build Tooling: Vite / Webpack (for optimized bundling)
* **Backend:**
* Runtime Environment: Node.js
* Web Framework: Express.js
* Database:
* Relational: PostgreSQL (for structured data like users, projects, tasks)
* NoSQL: MongoDB (for unstructured data like logs, AI outputs)
* **Database:** Detail the database schema and interactions with MongoDB.

MongoDB is used to handle unstructured and semi-structured data that is dynamic, AI-generated, or non-relational in nature. This complements a relational database like PostgreSQL, which handles structured core data (users, roles, projects, etc.).

* AI Output Storage:

NLP-parsed requirements, AI-generated code, test recommendations, bug predictions.

* Logs and Activity Traces:

Real-time logs from CI/CD processes, test runs, and user actions.

* Monitoring Data:

System metrics, anomaly detection outputs, performance snapshots.

* Historical Data & Versioning:

Stores past versions of requirements, generated code, and test results for learning and rollback.

# 4. Setup Instructions

* **Prerequisites:**
* Node.js
* MongoDB
* Python
* Hugging Face Transformers
* React.js
* Express.js
* **Installation:**

1. Clone Repository

2. Install Backend Dependencies

3. Set Up Python Environment

4. Install Frontend Dependencies

5. Create Environment Variable Files

6. Start MongoDB and Redis

7. Run Backend Server

8. Run Python AI Service

9. Run Frontend Server

# 5. Folder Structure

* **Client:**

Describe the structure of the React frontend.

Organized into folders for components, pages, API calls, state management, styles, and utilities, with a public folder for static files and an entry point in index.js.

* **Server:**

Structured with folders for config, controllers, middleware, models, routes, services, and utilities. The app is initialized in app.js and started via server.js.

# 6. Running the Application

* **Frontend:**

cd client

npm install

npm start

* **Backend:**

cd server

npm install

npm start

# 7. API Documentation

* **Register User**

POST /api/auth/register

Response:

201 Created

* **Login User**

POST /api/auth/login

Response:

200 OK

* **Get Projects**

GET /api/projects

Headers:

Authorization: Bearer <token>

Response:

200 OK

* **Create Project**

POST /api/projects

Headers:

Authorization: Bearer <token>

Response:

201 Created

* **AI Analysis**

POST /api/ai/analyze

Headers:

Authorization: Bearer <token>

Response:

200 OK

# 8. Authentication

1. **Authentication (Identity Verification)**

* JWT (JSON Web Tokens): Used for stateless user login. Tokens are issued upon successful login and attached to each request.
* Token Expiry & Refresh: Short-lived access tokens with refresh tokens for session continuity.

1. **Authorization (Access Control)**

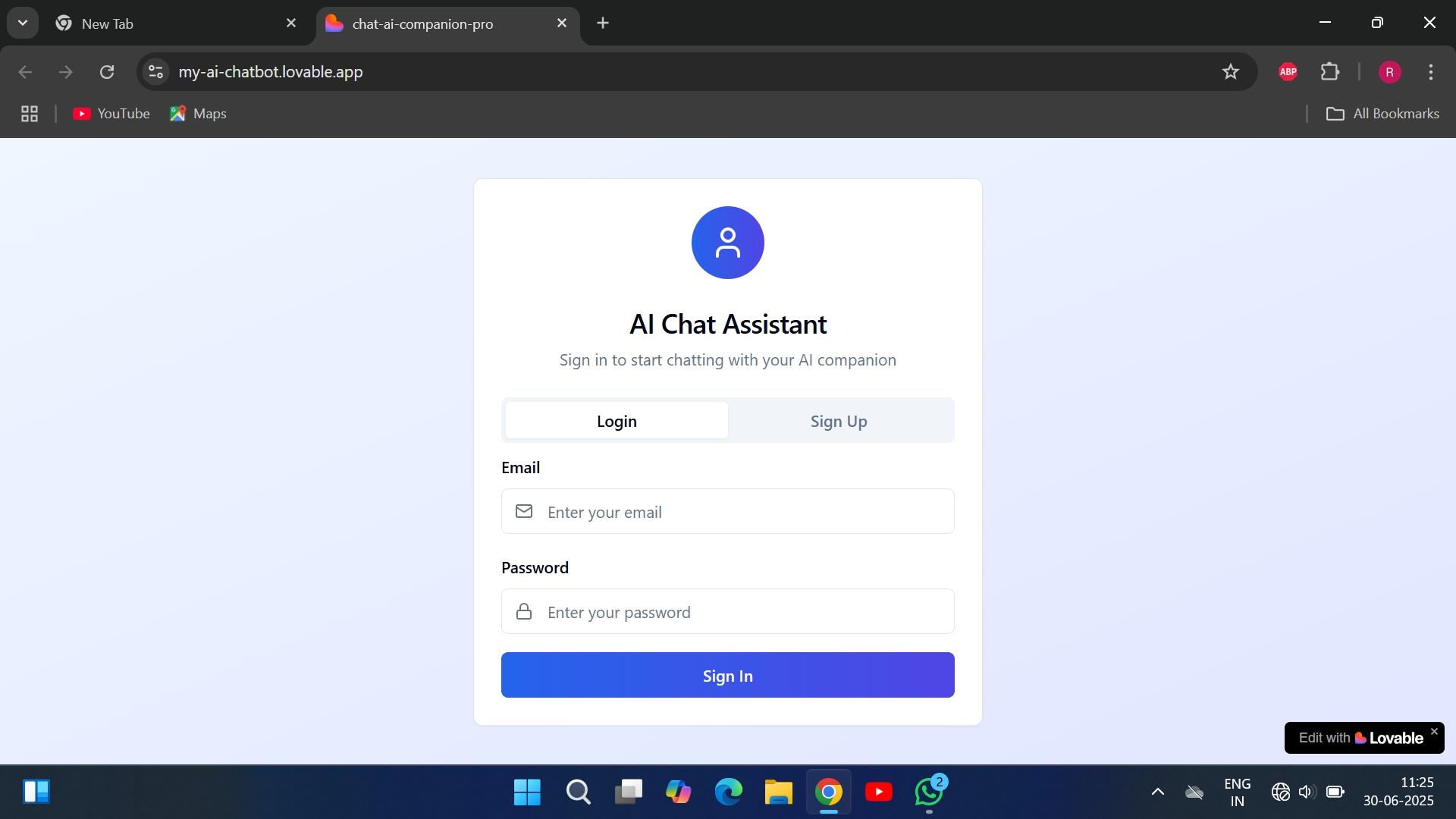
* RBAC (Role-Based Access Control):
* Roles like Admin, Developer, QA, Manager determine access.
* Fine-Grained Permissions: Specific rights assigned for actions like editing tasks, deploying models, etc.

1. **Session Management**

* Tokens: Access tokens are used for each request; refresh tokens extend sessions.
* Logout & Token Revocation: Invalidate tokens on logout or misuse.
* Session Timeout: Auto logout after inactivity.

1. **Security Enhancements**

* HTTPS, rate limiting, audit logs, and secure cookies ensure system safety.

1. **User Interface**
2. **Testing** 
   * **Unit Testing** – Validates individual components

Tools: Junit, PyTest, Mockito

* + **AI/ML Model Testing** – Checks model accuracy, fairness, and drift

Tools: MLflow, AIF360, DeepChecks, Evidently AI

* + **Integration Testing** – Ensures modules and tools work together

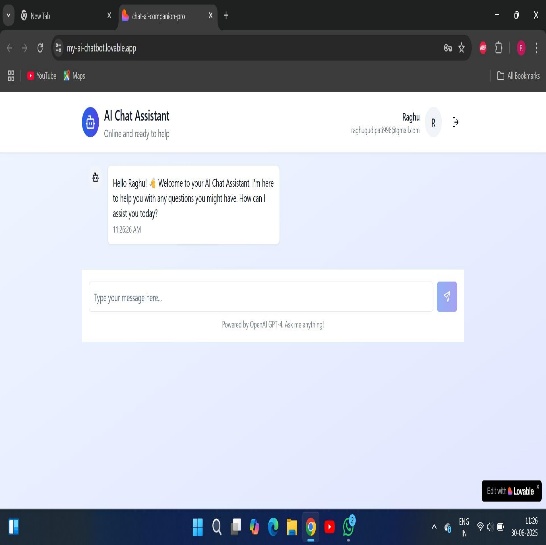
Tools: Postman, REST Assured, Docker Compose

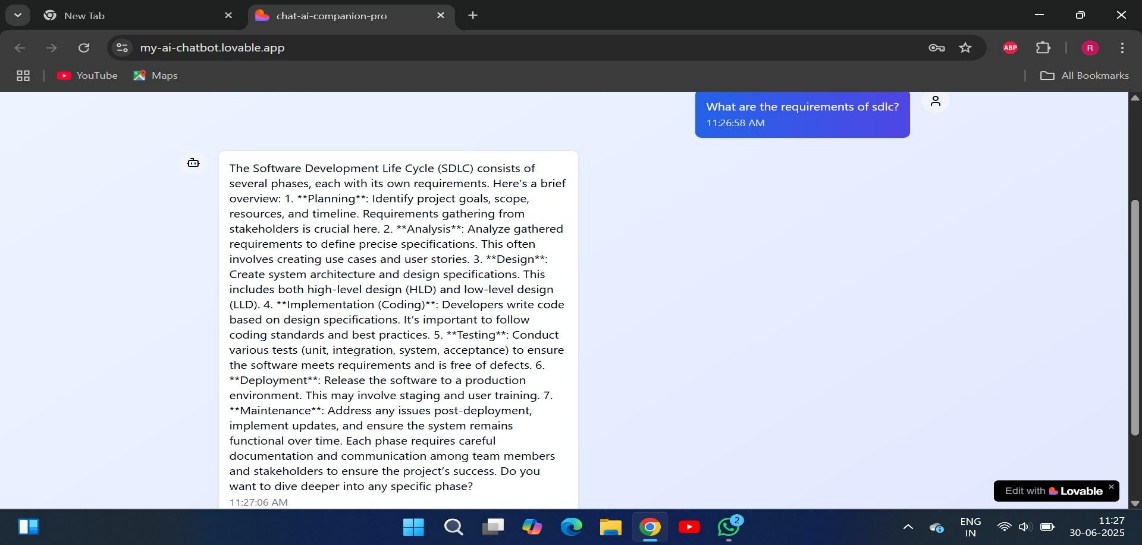
* + **System Testing** – Verifies end-to-end workflows

Tools: Selenium, Robot Framework, Cypress

* + **Performance Testing** – Measures speed and load capacity

Tools: JMeter, Locust

1. **Screenshots or Demo** 



# 12. Future Enhancements

 **AI Code Generation:** Auto-create code and tests from specs.

 **Predictive Planning:** Forecast project risks and delays.

 **Smart Test Automation:** Prioritize and generate tests using AI.

 **AI Code Review:** Intelligent feedback on code quality and security.

 **Continuous Model Learning:** Update AI models with real-time data.