SmartSDLC – AI-Enhanced Software Development Lifecycle Project Documentation format

1. Introduction

Project Title:

- **Team Members:**
  - G.Vasanti: Basic coder& organize
  - M.Jayasri: Research Helper
  - T.Vennela: Design helper
  - P.Jashwanth: ideas implement

2. Project Overview:

Purpose:

The **SmartSDLC** project aims to modernize and enhance the traditional Software Development Lifecycle (SDLC) by integrating Artificial Intelligence (AI) tools and automation technologies. The primary goal is to streamline every phase of the development process—from requirement gathering to deployment—making it faster, smarter, and more efficient. It assists developers and project teams in decision-making, reducing manual tasks, improving software quality, and minimizing development time and cost.

Features:

AI-Powered Requirement Analysis:

- Uses Natural Language Processing (NLP) to interpret and analyze user requirements.
- Automatically generates user stories or software specifications.

Intelligent Project Planning:

- Generates timelines and allocates tasks based on historical data and team capacity.
- Predicts risks and recommends mitigation strategies.

Code Generation & Review:

- Integrates with AI models to assist in code suggestions and generation.
- Performs automatic code reviews and suggests improvements.

Automated Testing:

- Generates test cases based on requirements and performs test automation.
- Identifies edge cases and potential bugs using AI.

Real-Time Monitoring & Reporting:

- Tracks project progress with dashboards.
- Sends automated reports and status updates to stakeholders.

Continuous Integration & Deployment (CI/CD):

- Integrates with DevOps tools to enable seamless and automated deployment.
- Ensures version control and rollback options.

3. Architecture:

Frontend: React-based Architecture

The frontend of SmartSDLC is developed using React.js, offering a responsive and interactive user interface (UI). The architecture follows a component-based design, ensuring reusability, modularity, and maintainability.

- **Key Compnents:**
    - Dashboard: Displays project KPIs, status, and timelines.
    - RequirementForm: Form for entering and analyzing software requirements.
    - AIChatAssistant: Embedded chatbot for AI-guided interactions.
- **State Management:**
    - Managed using React Context API or Redux (if complex state management is needed).
    - Uses useEffect and useState for lifecycle and reactive UI updates.
- **API Integration:**
    - Communicates with the backend through Axios or fetch() using RESTful APIs.
    - Supports JWT-based authentication for secure access.

Backend: Node.js + Express.js Architecture

The backend is built using Node.js with Express.js, providing a lightweight and scalable REST API server. It handles the business logic, authentication, and interactions with the database and AI services.

- **API Endpoints:**

    - POST /api/requirements/analyze → Processes requirements using AI.
    - GET /api/tasks → Fetch project tasks and progress.
    - POST /api/code/review → Accepts code and returns AI review.
    - POST /api/auth/login → User authentication with JWT.
    - POST /api/deploy → Trigger CI/CD pipeline actions.
- **Middleware:**
    - Error handling middleware
    - JWT authentication middleware
    - Request logging (using morgan or similar)
- **Integration:**
    - Integrates with external AI models (e.g., IBM Watsonx, OpenAI API)
    - Interfaces with DevOps tools (like Jenkins or GitHub Actions)

4. Setup Instructions:

Prerequisites

Ensure the following software and tools are installed before setting up the project:

Dependency	**Version (Recommended)**	**Purpose**
Node.js	v18.x or later	Backend runtime environment
npm / yarn	v9.x or later	Package manager for Node.js
MongoDB	v6.x or later	NoSQL database
Git	Latest	Version control
Visual Studio Code	Latest	IDE (optional but recommended)

Installation

Follow the steps below to set up the project on your local system:

Step 1: Clone the Repository

```
git clone https://github.com/your-username/SmartSDLC.git
cd SmartSDLC
```

Step 2: Install Frontend Dependencies

```
cd client  # Navigate to the React frontend folder
npm install
```

Step 3: Install Backend Dependencies

```
cd ../server  # Navigate to the backend folder
npm install
```

Step 4: Set Up Environment Variables

Create a .env file inside the server directory

```
PORT=5000
MONGO_URI=mongodb://localhost:27017/smartsdlc
JWT_SECRET=your_jwt_secret_key
AI_API_KEY=your_ai_api_key   # Optional if using external AI
```

Step 5: Start MongoDB

```
# For Linux/macOS
mongod

# For Windows, use MongoDB Compass or Mongo shell
```

Step 6: Run Backend Server

```
cd server
npm start
```

Step 7: Run Frontend (React App)

```
cd ../client
npm start
```

5. Folder Structure (Google Colab-Based - No Folders):

```
model_selected = "Simulated Code Generator AI (Watsonx alternative)"
print(f"AI Model Selected: {model_selected}")


architecture = {
    "Frontend": "Simulated Streamlit in Colab",
    "Backend": "FastAPI logic as Python functions",
    "AI Engine": "Simulated Python logic based on prompt",
}
for layer, desc in architecture.items():
    print(f"{layer}: {desc}")


print("Environment Ready in Google Colab ")


def generate_code(prompt):
    if "login" in prompt.lower():
        return """
def login(username, password):
    if username == "admin" and password == "123":
        return "Login successful"
    else:
        return "Invalid credentials"
"""
    elif "calculator" in prompt.lower():
        return """
def calculator(a, b, operation):
    if operation == "add":
```

```
        return a + b
    elif operation == "subtract":
        return a - b
    else:
        return "Unsupported operation"
"""
    else:
        return f"# Code for prompt: {prompt}\nprint('Hello from SmartSDLC AI!')"

def api_generate_code(prompt):
    return {"generated_code": generate_code(prompt)}

print("SmartSDLC AI Code Generator")
user_prompt = input("Enter your prompt for code generation: ")
response = api_generate_code(user_prompt)
print("\n Generated Code:\n")
print(response["generated_code"])

print(" Preparing Application for Deployment...")
test_prompt = "login"
test_result = api_generate_code(test_prompt)
print("\n Test Deployment Result:")
print(test_result["generated_code"])
print(" Project Completed Successfully!")
print("SmartSDLC used simulated AI to generate Python code based on user prompt.")
print("You used Google Colab to build, test, and simulate the entire SDLC process.")
```

6. Running the Application

Follow the steps below to run the frontend and backend servers locally after completing the setup:

Frontend (React App)

Navigate to the client/ directory:

```
cd client
```

Start the frontend server:

```
npm start
```

The React app will run at:

```
http://localhost:3000
```

Backend (Node.js + Express Server)

Navigate to the server/ directory:

```
cd server
```

Start the backend server:

```
npm start
```

The backend server will run at:

```
http://localhost:5000
```

Now both the frontend and backend are running!

You can open **http://localhost:3000** in your browse and begin using the SmartSDLC platform.

7. API Documentation

The backend provides a RESTful API to support the SmartSDLC platform. Below is the detailed documentation of the available endpoints.

Authentication Routes

POST /api/auth/register

Registers a new user.

Request Body:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "123456"
}
```

Response:

```
{
  "message": "User registered successfully",
  "token": "JWT_TOKEN"
}
```

POST /api/auth/login
Logs in a user and returns a token.

- **Request Body:**

```
{
  "email": "john@example.com",
  "password": "123456"
}
```

Response:

```
{
  "message": "Login successful",
  "token": "JWT_TOKEN"
}
```

Requirements Routes

POST /api/requirements/analyze

Analyzes user-entered software requirements using AI.

- **Request Body:**

```
{
  "text": "The system should allow users to register and login."
}
```

Response:

```
{
  "analysis": {
    "features": ["User registration", "Login"],
    "actors": ["User"],
    "useCases": ["Authentication"]
  }
}
```

Common Response Codes

Code	Meaning
200	Success
201	Resource Created
400	Bad Request
401	Unauthorized
404	Not Found
500	Internal Server Error

8. Authentication

Authentication and authorization in SmartSDLC are implemented using JWT (JSON Web Tokens) to ensure secure access to protected routes and user-specific resources.

Authentication Flow

1. **User Registration/Login**
    - When a user registers or logs in, they send their credentials (email, password) to the backend (/api/auth/login or /api/auth/register).
    - The backend verifies the credentials, and upon success, generates a JWT.
2. **Token Generation**
    - A JWT token is created using the user's ID and a secret key defined in **.env:**

```
JWT_SECRET=your_jwt_secret_key
```

    - The token includes:

```
{
  "userId": "60e...",
  "email": "john@example.com",
  "role": "developer"
}
```

3. **Token Storage**

- The frontend stores the JWT token in localStorage or sessionStorage.
- For every API call to protected routes, the token is sent in the Authorization header:

```
Authorization: Bearer <token>
```

Authorization Middleware

- On the backend, protected routes use a **JWT middleware** to:

- Verify the token using jsonwebtoken.verify().
- Decode the user information and attach it to the request object (req.user).
- Reject requests with invalid or expired tokens.

Middleware (authMiddleware.js):

```
const jwt = require("jsonwebtoken");


const verifyToken = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (authHeader && authHeader.startsWith("Bearer ")) {
    const token = authHeader.split(" ")[1];
    try {
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = decoded; // Attach user info to request
      next();
    } catch (err) {
      return res.status(401).json({ message: "Invalid token" });
    }
  } else {
    return res.status(401).json({ message: "No token provided" });
  }
};
```

Feature	**Method Used**
Authentication	JWT Token
Storage	localStorage / Header
Protection	Middleware in Express
Authorization	Role-based (optional)

9. User Interface

```
AI Model Selected: Simulated Code Generator AI (Watsonx alternative)
```

```
Frontend: Simulated Streamlit in Colab
Backend: FastAPI logic as Python functions
AI Engine: Simulated Python logic based on prompt
```

```
Environment Ready in Google Colab
```

```
SmartSDLC AI Code Generator
Enter your prompt for code generation: login

 Generated Code:


def login(username, password):
    if username == "admin" and password == "123":
        return "Login successful"
    else:
        return "Invalid credentials"
```

```
 Preparing Application for Deployment...

 Test Deployment Result:

def login(username, password):
    if username == "admin" and password == "123":
        return "Login successful"
    else:
        return "Invalid credentials"
```

```
 Project Completed Successfully!
SmartSDLC used simulated AI to generate Python code based on user prompt.
You used Google Colab to build, test, and simulate the entire SDLC process.
```

10. Testing

To ensure the reliability and correctness of SmartSDLC, a structured testing strategy is used that includes unit testing, integration testing, and end-to-end (E2E) testing, across both frontend and backend components.

Testing Strategy Overview

Type	Purpose	Tools Used
Unit Testing	Test individual components or functions	Jest, Mocha
Integration Testing	Test API interactions and backend logic	Supertest, Chai
E2E Testing	Simulate full user flow from frontend to backend	Cypress (or Playwright)
Manual Testing	UI/UX testing and exploratory bug discovery	Browser-based/manual

11.Screenshots or Demo:

```
SmartSDLC AI Code Generator
Enter your prompt for code generation: login

 Generated Code:


def login(username, password):
    if username == "admin" and password == "123":
        return "Login successful"
    else:
        return "Invalid credentials"
```

```
 Project Completed Successfully!
SmartSDLC used simulated AI to generate Python code based on user prompt.
You used Google Colab to build, test, and simulate the entire SDLC process.
```

Demo Link: https://drive.google.com/file/d/14oZcYhhRg4zG-f_5Tb5sZfOtEJ693OJz/view?usp=drivesdk

12. Known Issues

Despite the overall stability of the SmartSDLC platform, there are a few known issues and limitations that users and developers should be aware of. These will be addressed in future updates.

1. Delayed AI Response on Slow Networks

- Description: The AI requirement analysis and code review features may take longer to respond on slower internet connections.
- Impact: Causes delay in UI rendering and user experience.
- Workaround: Refresh the page or retry the request after a few seconds.

2. Limited Role-Based Access Control (RBAC)

- Description: While user roles exist (e.g., developer, manager), fine-grained access control for specific routes and features is not fully enforced.
- Impact: All logged-in users may access some admin-level features.
- Workaround: Manually restrict access in the frontend (temporary solution).

3. No Password Reset or Email Verification

- Description: User accounts do not currently support password recovery or email verification.
- Impact: Users cannot reset passwords via email if forgotten.
- Workaround: Admin must reset credentials directly in the database.

4. Basic Form Validation

- Description: Some forms (e.g., task creation, project input) have only minimal frontend validation.
- Impact: Users may accidentally submit empty or malformed data.
- Workaround: Add manual checks before submission; backend rejects invalid entries.

5. AI Code Review Limited to JavaScript

- Description: The current AI integration primarily supports JavaScript code for review.

- Impact: Code written in Python, Java, or other languages may produce inaccurate suggestions.
- Workaround: Display warning to users about language support limitations.

Planned Fixes (Upcoming)

Issue	Target Version
Email verification and password reset	v1.2
Full RBAC enforcement	v1.3
Multi-language AI code review	v1.4

13. Future Enhancements

To make SmartSDLC more robust, intelligent, and user-friendly, several potential features and improvements have been identified for future development.

1. Role-Based Access Control (RBAC)

- Description: Implement detailed permission levels for different user roles like developers, testers, and managers.
- Benefit: Increases security and allows tailored feature access per role.

2. Password Reset and Email Verification

- Description: Enable users to reset forgotten passwords and verify their email upon registration.
- Benefit: Improves user account security and usability.

3. Multi-Language Code Review

- Description: Extend AI code review functionality to support languages like Python, Java, C++, etc.
- Benefit: Makes the platform usable by a wider range of developers.

4. Analytics Dashboard

- Description: Add visual analytics for team performance, task completion rates, bug trends, etc.
- Benefit: Helps project managers make data-driven decisions.

5. AI Test Case Generation

- Description: Automatically generate unit and integration test cases from requirements using AI.
- Benefit: Saves developer time and ensures better test coverage.

6. Real-Time Collaboration

- Description: Introduce real-time project editing, task updates, and comments using WebSockets.
- Benefit: Enables seamless team collaboration (like Google Docs or Trello).