SmartSDLC - AI-Powered Software Development Lifecycle Automation

1. Introduction

Project Title: SmartSDLC - AI-Powered Software Development Lifecycle Automation
Team Members:
- Project Manager: [rithwik sir] – Oversees the entire project, ensures deadlines are met.
- AI Engineer: [Kolusu Surendra Kumar] – Develops and integrates the AI models (NLP, WatsonX) for automating the SDLC processes.
- Backend Developer: [Prakash Tammireddi] – Responsible for backend API development, integrating with WatsonX and MongoDB.
- Frontend Developer: [Mungamuri Yamini] – Develops the user interface using React, ensuring smooth interaction with the backend.
- QA Engineer: [Koram Prasanna] – Focuses on testing and ensuring the functionality of each feature, bug-free experience.
- Documentation Specialist: [] – Creates and maintains all project documentation.

2. Project Overview

Purpose:
SmartSDLC is an AI-powered platform designed to fully automate the Software Development Lifecycle (SDLC). It leverages Natural Language Processing (NLP) and WatsonX to automate processes like converting unstructured requirements into actionable user stories, generating code, fixing bugs, creating test cases, and even providing chatbot support for developers. This leads to better accuracy, faster development, and improved collaboration within development teams.

Features:
- Requirement Parsing: SmartSDLC uses NLP to convert unstructured requirements (e.g., plain text) into user stories, making it easier to translate customer needs into software features.
- Code Generation: Based on user stories, the platform generates code snippets automatically, speeding up development.
- Bug Fixing: The system identifies bugs in the code and fixes them automatically by utilizing AI-powered analysis.
- Test Case Generation: The platform generates unit tests and integration tests to ensure the generated code works as expected.
- Code Summarization: The platform summarizes large code blocks, making it easier for developers to understand and maintain them.

- AI Chatbot Support: The system includes a chatbot that helps developers by answering questions and guiding them through the SDLC processes.

3. Architecture

Frontend:
Built using React.js for an interactive and responsive user interface. Allows users to input requirements, view user stories, generate code, and interact with the AI chatbot.

Backend:
The backend is implemented using Node.js and Express.js to handle API requests, user interactions, and integrate with WatsonX to process the SDLC tasks. Interacts with MongoDB to store user stories, generated code, bugs, and test cases. AI models (via WatsonX) process input and generate required outputs such as code and bug fixes.

Database:
MongoDB is used to store unstructured requirements, user stories, generated code, bugs, test cases, and chatbot interactions.
Key collections include:
- Requirements: Stores unstructured data entered by the user.
- User Stories: Stores structured user stories derived from the requirements.
- Generated Code: Stores code snippets generated from user stories.
- Bugs: Stores bugs found in the generated code and their resolutions.
- Test Cases: Stores automated test cases for generated code.

4. Setup Instructions

Prerequisites:
- Node.js: Version 16 or higher.
- MongoDB: Install locally or use a cloud-based MongoDB service (e.g., MongoDB Atlas).
- WatsonX API Credentials: You'll need to sign up on IBM Cloud to get the API credentials for WatsonX.

Installation:
1. Clone the repository:
   git clone <repo_url>
2. Install dependencies for both frontend and backend:
   npm install in both the client and server directories.
3. Set up environment variables:
- Create a .env file in both the frontend and backend directories to include your MongoDB connection string and WatsonX API key.

4. Start the application:
- In the client directory, run:
  npm start
- In the server directory, run:
  npm start

5. Folder Structure

Client:
Contains all frontend components and assets:
- src/: Contains React components such as Home.js, UserStories.js, CodeEditor.js, and Chatbot.js.
- public/: Contains static assets like index.html and images.

Server:
Contains backend logic:
- controllers/: Handles API requests for generating user stories, code, and managing bugs.
- routes/: API endpoints like /generate-user-story, /generate-code.
- models/: MongoDB schema models for storing data like requirements, user stories, and generated code.

6. Running the Application

Frontend:
- Navigate to the client directory:
  cd client
- Run the frontend:
  npm start

Backend:
- Navigate to the server directory:
  cd server
- Start the backend:
  npm start

7. API Documentation

POST /generate-user-story
Request:
  { "requirements": "The user needs a login page." }

Response:
{ "user_story": "As a user, I want a login page so that I can log in to the system." }

POST /generate-code
Request:
{ "user_story": "As a user, I want a login page." }
Response:
{ "code": "const loginPage = () => { return <form>...</form> }" }

8. Authentication

Method:
- JWT-based Authentication for securing API endpoints.
- Users log in using a username and password, and receive a JWT token that is used for subsequent API requests in the Authorization header.

9. User Interface

The UI showcases:
- A dashboard where users can input requirements.
- A user story generation screen that displays structured user stories.
- Code generation screens where developers can view the code generated for a user story.
- An interactive chatbot for instant developer support.

10. Testing

Strategy:
- Unit testing for backend functions using Jest to verify code logic, especially the AI features like bug fixing and test case generation.
- End-to-end testing using Cypress for the frontend to ensure the user experience flows smoothly from requirement input to code generation.

Tools:
- Jest: For backend testing.
- Cypress: For UI testing and ensuring the React components function as expected.

11. Screenshots or Demo

Link to Demo: A hosted demo or video link showcasing the system can be shared here.

12. Known Issues

Known Bugs:

- Sometimes, WatsonX can take time to process large user stories, causing delays in generating code.
- API keys might expire if not updated.

13. Future Enhancements

New AI Models:

- Introduce AI models for code refactoring to enhance code quality.

CI/CD Integration:

- Seamless integration with CI/CD pipelines to automate the deployment process based on generated code.

Cross-Language Support:

- Extend code generation to other programming languages like Java and C#.