# Smart SDLC Change File – Theory Based Explanation

## **Introduction**

The Software Development Life Cycle (SDLC) is a structured process used for planning, developing, testing, deploying, and maintaining software systems. It ensures that software projects are completed in a systematic and efficient manner. For developing large-scale systems such as a Smart City Assistant, SDLC provides a roadmap that defines each stage of development clearly.

The Smart SDLC Change File described here is designed for creating an **AI-driven Sustainable Smart City Assistant**. The purpose of this assistant is to support sustainable urban development by combining artificial intelligence, natural language processing, forecasting, anomaly detection, and citizen feedback mechanisms. By following the SDLC model, the system ensures scalability, security, and long-term usability. The following sections explain each stage of this Smart SDLC in detail.

# Team Members:

# ARUN P

# PRAKASH M

# ABDUL RAHUMAN I

# LOGESWARAN G

## **1. Requirement Analysis**

Requirement analysis is the first and most important step in the software development life cycle. The purpose of this project is to empower cities with an AI-driven Sustainable Smart City Assistant. This assistant will help citizens, city officials, and researchers by providing real-time insights and recommendations for sustainable development.

The stakeholders include citizens, city officials, and researchers. Citizens can use the assistant for eco-friendly tips and issue reporting. City officials will benefit from policy summarization, anomaly detection, and forecasting to support decision-making. Researchers can analyze city data for long-term planning.

The functional requirements include conversational chatbot interface, summarization of policies, forecasting of urban growth indicators, anomaly detection in data, eco-tip generation, and feedback loops. The non-functional requirements focus on scalability to handle large data, security of sensitive city information, usability for diverse users, and real-time processing of data for quick decision-making.

## **2. System Design**

System design defines the architecture of the Smart City Assistant. The frontend is designed using Streamlit, which provides user-friendly dashboards and chat interfaces. The backend is built with FastAPI, a modern and fast web framework for creating APIs. For natural language processing, the system integrates IBM Watsonx Granite LLM to handle tasks like text summarization, question answering, and conversational interaction. A Vector Database such as Pinecone is used for semantic search, enabling the system to retrieve information intelligently from stored city documents.

## **3. Module Design**

The system is divided into smaller modules for better clarity and maintainability. The Embedder Module converts input text into vector embeddings, which allows semantic search and efficient retrieval of documents. The Forecaster Module predicts future trends of key city parameters like pollution levels, energy usage, and water demand. The Anomaly Checker Module identifies unusual behaviors, such as sudden spikes in pollution or abnormal energy consumption, which helps in early warning and preventive action.

## **4. Implementation**

The implementation stage translates the design into working code. The system is developed using Python 3.9 or above. FastAPI is used for backend API services, and Streamlit is used for creating the frontend dashboard. For machine learning tasks such as anomaly detection and forecasting, Scikit-learn is used. IBM Watsonx Granite serves as the natural language engine, while Pinecone is used for semantic data storage and search.

The project follows a structured folder system: app/ for backend application and API logic, ui/ for the Streamlit-based user interface, and scripts/ for additional utilities like preprocessing and training scripts.

## **5. API Development**

APIs act as the bridge between the frontend and backend. The Smart City Assistant exposes multiple endpoints to perform key tasks. For example, /chat/ask is used for interacting with the chatbot, /upload-doc allows the upload of official city documents, /search-docs enables searching within stored documents, /get-eco-tips generates environment-friendly suggestions, and /submit-feedback collects citizen feedback. These endpoints ensure modularity and easy access to different functionalities.

## **6. Testing**

Testing ensures the reliability and accuracy of the system. Unit testing is performed to verify prompt functions, embeddings, and forecasting modules. API testing is done using Swagger UI and Postman to confirm the correctness of endpoints. In addition, manual validation is carried out to check if the chatbot provides accurate and relevant answers. Edge case testing is also included to handle unexpected scenarios such as invalid input files or vague user queries.

## **7. Deployment**

Deployment refers to making the system accessible to its users. The steps include cloning the repository, installing dependencies, configuring environment variables, and running both FastAPI and Streamlit services. The system is cloud-ready and can be deployed on IBM Cloud for LLM services and Pinecone Cloud for vector database storage. This ensures global accessibility, high availability, and scalability.

## **8. Authentication**

Security is crucial in any smart city solution. Authentication is provided using JWT (JSON Web Tokens) to ensure secure communication. OAuth2 is integrated to allow login through external providers such as Google or Microsoft. Role-based access control ensures different permissions for different users: citizens have limited access, officials have access to forecasting and policy insights, and administrators enjoy full system privileges.

## **9. Maintenance**

Maintenance is a continuous activity after deployment. The system requires regular monitoring for uptime, performance, and errors. An anomaly detection mechanism is integrated for continuous surveillance of city trends. A feedback loop is also included so citizens can report issues or improvements. Proper logging and monitoring allow developers to address errors quickly and plan for future updates.

## **10. Future Enhancements**

The Smart City Assistant is designed with scope for future improvements. Possible enhancements include support for user sessions to save history and personalize experiences, advanced analytics for policymakers, and integration with IoT devices like pollution meters and traffic sensors. The system can also be scaled to multiple cities with larger populations, making it a comprehensive smart city solution.

## **Conclusion**

This Smart SDLC Change File provides a structured, scalable, and secure methodology for developing a Smart City Assistant. Each stage of the SDLC—from requirement analysis to maintenance—ensures that the system is citizen-friendly, data-driven, and future-ready.