# *Project Title*

Smart SDLC - AI-Enhanced Software Development Lifecycle

1.Introduction

* **Team Members:**

• Sneka.R

• Srivalli.R

• Swetha.S

• Vaishnavi.A

• Vennila.M

## 2. Project Overview

* Purpose :

The Smart SDLC project enhances the Software Development Lifecycle by integrating AI-powered requirement analysis and automated code generation. It provides developers and project teams with tools for analyzing requirement documents, organizing them into structured categories, and generating functional code in multiple programming languages. This system saves time, ensures quality, and supports collaborative development.

* + Features:
* Requirement Analysis Assistant – Extracts functional, non-functional, and technical requirements from PDFs or text.
* AI-based Code Generation – Generates production-ready code in multiple programming languages (Python, Java, JavaScript, etc.).
* Download History – Saves past analysis and code generation results into a .txt file.
* Code Syntax Highlighting – Displays generated code in a syntax-highlighted format (gr.Code).
* Clear Buttons – Reset outputs for new analysis and code generation.
* History Storage – Maintains both requirement analysis and code generation history.
* Model Settings Panel – Adjustable creativity (temperature) and output length via sliders.
* Soft Theme Support – Clean and modern interface for accessibility.

## 3. Architecture

Frontend (Gradio): Provides a tabbed interface for requirement analysis, code generation, and history download with sliders for model settings.  
  
Backend (Torch + HuggingFace Granite): Loads IBM Granite LLM for requirement analysis and code generation.  
  
Modules:  
- Requirement Analysis: Extracts key requirements from PDFs or text input.  
- Code Generator: Produces functional code in user-selected language.  
- History Manager: Stores analysis and code results for later download.  
  
Download Manager: Exports all results into timestamped .txt files.

## 4. Setup Instructions

* Prerequisites:  
  • Python 3.9+  
  • pip and virtual environment  
  • HuggingFace Transformers library  
  • PyTorch (GPU support optional)  
  • Gradio  
  • PyPDF2  
    
  Installation:  
  1. Clone the repository  
  2. Install dependencies from requirements.txt  
  3. Run the application using: python app.py  
  4. Access the Gradio link provided in the terminal

## 5. Folder Structure

smart\_sdlc/  
│── app.py # Main entry file with Gradio app  
│── requirements.txt # Python dependencies  
│── modules/  
│ ├── analyzer.py # Requirement analysis logic  
│ ├── codegen.py # Code generation logic  
│ ├── history.py # History storage and download manager  
│── docs/ # Documentation and policy files  
│── outputs/ # Saved histories and exports

## 6. Running the Application

1. Launch the Gradio application with: python app.py  
2. Open the provided localhost or shareable link.  
3. Navigate to tabs:  
 - Code Analysis  
 - Code Generation  
 - Download History  
4. Upload PDFs or enter requirements manually.  
5. View results, copy generated code, and download history.

## 7. API Documentation

* Although the app is primarily UI-based, the following internal functions are available:  
  - requirement\_analysis(pdf\_file, prompt\_text, max\_len, temp): Extracts and categorizes requirements.  
  - code\_generation(prompt, language, max\_len, temp): Generates code for given requirements.  
  - download\_history(): Exports all past results into a .txt file.  
  These can be extended into REST APIs in the future.

## 8. Authentication

The current version runs in open demo mode without authentication. For secure deployment, options include:  
• API key-based access  
• JWT tokens  
• Role-based authentication (Admin, Developer, Tester)  
• OAuth2 integration for enterprise environments

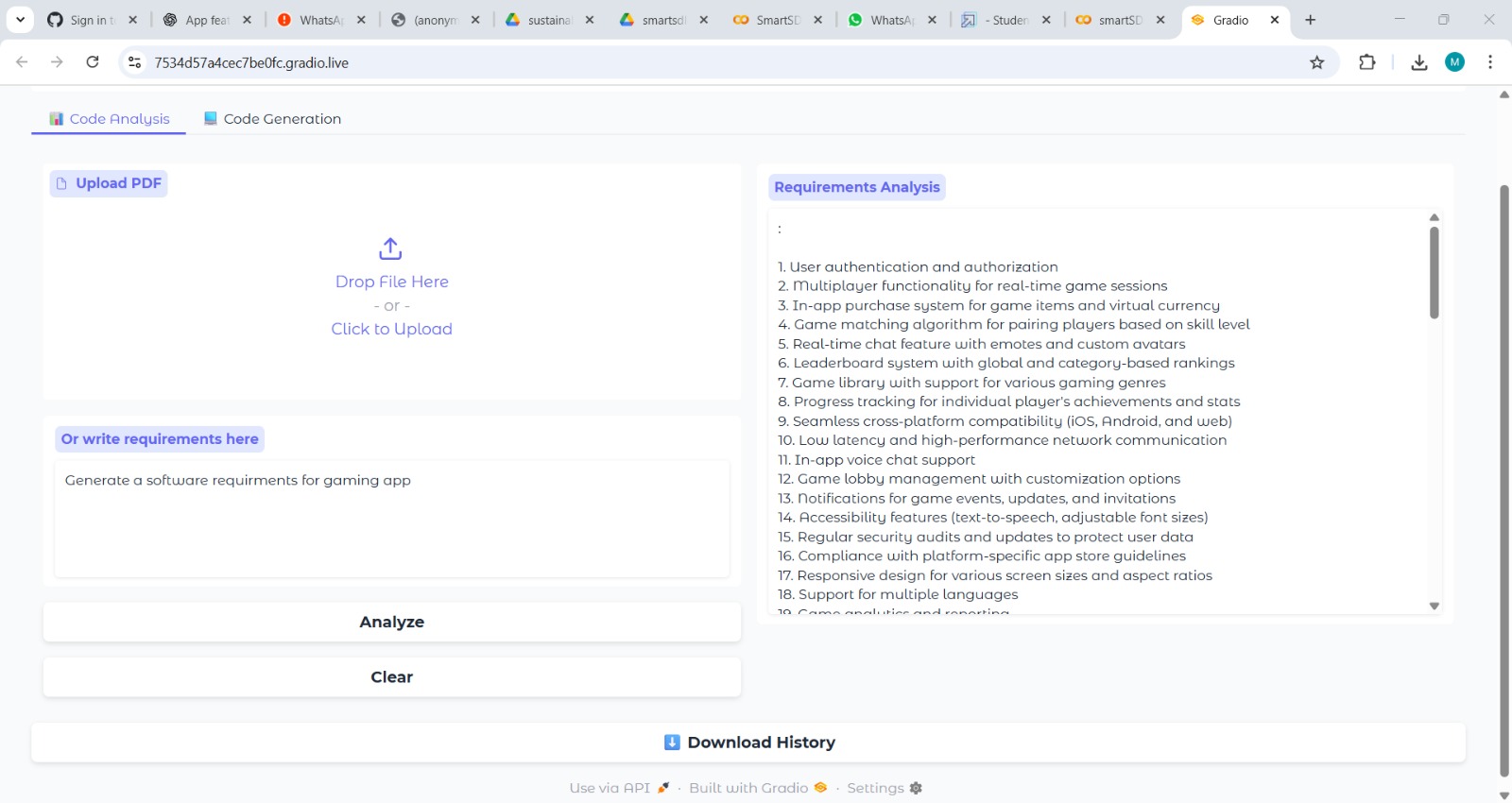
## 9. User Interface

* The UI is built using Gradio with the following elements:  
  • Tabs: Code Analysis, Code Generation, Download History  
  • Accordions: Model Settings (temperature, max length)  
  • Sliders: Adjustable creativity and output length  
  • Code Box: Syntax-highlighted output (gr.Code)  
  • Download Button: Save history into .txt files  
  • Clear Buttons: Reset outputs instantly

## 10. Testing

* Testing was conducted as follows:  
  • Unit Testing: Functions for requirement analysis, code generation, and history export.  
  • Manual Testing: Uploading different PDFs, generating multi-language code.  
  • API Testing: Verifying internal functions with sample prompts.  
  • Edge Case Handling: Empty PDFs, long text input, invalid file formats.  
  • Performance Testing: Running with large requirements documents.

## 11. Screenshots



## 12. Known Issues

• Large PDFs may slow down requirement analysis.  
• GPU recommended for faster model inference, CPU is slower.  
• Limited offline use without HuggingFace model download.  
• Generated code may require manual optimization.  
• Currently no integrated database for storing history.

## 13. Future Enhancement

• Extend into REST API endpoints for external integration.  
• Add database support for persistent storage of analysis history.  
• Support advanced IDE plugins (VS Code, IntelliJ) for direct integration.  
• Improve UI with chart-based visualizations of requirements.  
• Enable team collaboration and version control features.  
• Add multilingual support for requirement analysis.