Project Report

1. Introduction

- Project Title: AI-Powered Code Analysis & Generator
- Team Leader: Ramya M
- Team Members: Harish Balaji G, Thamizh Selvi K, Lavanya A

2. Project Overview

- Purpose: The purpose of this project is to develop an AI-powered application that can analyze software requirement documents and automatically generate functional code in multiple programming languages. This tool assists software developers, students, and project teams in reducing development time and ensuring accurate requirement analysis.
- By leveraging Large Language Models (LLMs) and Natural Language Processing (NLP), the system can process requirement documents (PDF or text), extract key specifications, and generate optimized code snippets or full modules.
- Features: Requirement Analysis, Code Generation, PDF/Text Input Support, Interactive UI (Gradio), LLM-Powered Responses.

3. Architecture

- Frontend (Gradio UI): Provides an interactive user interface with two main modules: Requirement Analysis and Code Generation.
- Backend (FastAPI): Manages input processing, integrates with the AI model, and serves outputs to the frontend.
- LLM Integration (IBM Granite): Handles natural language understanding and generation for both analysis and code generation.
- ML Modules: Support classification of requirements and contextual understanding.

4. Setup Instructions

- Prerequisites: Python 3.9+, pip, virtual environment tools, Transformers, Torch, Gradio, PyPDF2 libraries, IBM Granite model access.
- Installation: Clone the repository, install dependencies from requirements.txt, run the backend server, launch Gradio app, upload requirement documents or type requirements.

5. Folder Structure

- app/ – Backend logic
- ui/ – Gradio frontend components
- model_integration.py – Handles Granite LLM queries
- pdf_reader.py – Extracts text from PDFs
- requirement_analyzer.py – Classifies requirements

- code_generator.py – Generates code in selected languages

6. Running the Application

- Start FastAPI backend server.
- Run Gradio app frontend.
- Navigate through Requirement Analysis and Code Generation tabs.
- Upload PDF documents or enter requirements.
- View extracted requirements and generated code in real time.

7. API Documentation

- POST /analyze-requirements – Extracts and classifies requirements
- POST /generate-code – Generates code for given requirements
- POST /upload-pdf – Uploads and extracts text from PDF

8. Authentication

- Currently open for testing.
- Future support: JWT tokens, role-based access (Admin, Developer, Tester).

9. User Interface

- Tabbed layout: Requirement Analysis, Code Generation.
- Upload/manual text input, real-time outputs, PDF report download capability.

10. Testing

- Unit Testing: Verified requirement extraction and code generation.
- API Testing: Using Swagger and Postman.
- Manual Testing: Checked Gradio interface.
- Edge Cases: Large PDFs, incomplete requirements.

11. Screenshots

- To be added after implementation.

12. Known Issues

- Generated code may need manual optimization.
- Large PDFs may take longer to process.

13. Future Enhancements

- Support more programming languages.
- Integrate with GitHub for version control.
- Enable real-time team collaboration.
- Deploy on cloud for scalability.