AI Code & Requirement Assistant
Project Document

1.Introduction

Project title : AI Code & Requirement Assistant

Team member : RITHIKA. T

Team member : SREE LAKSHMI. M

Team member : SHARMILA. G

Team member : SHAMNI. S

---

2. Project Overview

This project is an AI-powered assistant that helps software developers and project managers streamline requirement analysis, code generation, and unit test creation. It allows users to either upload a PDF of software requirements or manually input them. The AI analyzes the requirements, categorizes them, generates code in the selected programming language and style, and even creates unit tests for the generated code.

The tool is designed with a **modern, intuitive UI** using Gradio, featuring thick-bordered cards for outputs and a user-friendly tab layout.

---

3. Problem Statement

Manual requirement analysis and code generation are time-consuming and prone to errors. Developers often spend hours interpreting documents and designing test cases.

The AI Code & Requirement Assistant solves these challenges by automating:

- Requirement extraction and classification
- Code generation in multiple languages and frameworks
- Generation of unit tests

---

4. Objectives

1. Extract and analyze software requirements from PDF documents or manual input.
2. Categorize requirements into **Functional**, **Non-Functional**, and **Technical Specifications**.
3. Generate code in multiple programming languages and frameworks.
4. Generate corresponding unit tests for the generated code.
5. Provide an interactive and visually appealing user interface for easy navigation.

---

5. Unique Selling Proposition (USP)

- **End-to-End Automation:** From requirement analysis to unit test generation.
- **Multi-Language Support:** Python, JavaScript, Java, C++, C#, PHP, Go, Rust.
- **Framework Integration:** Supports Flask, Django, React, Node.js.
- **Stylish UI:** Modern cards and thick-bordered output boxes improve readability.
- **Interactive Outputs:** Editable code blocks for real-time adjustments.

---

6. Technology Stack

Layer	**Technology**
Frontend	Gradio (Python)
Backend	Python, Transformers (HuggingFace)
AI Model	IBM Granite 3.2 2B-instruct
Libraries	PyPDF2, Torch, Transformers
Hosting	Local / Cloud (Gradio share link)

---

7. Features

1. **Requirement Analysis Tab**
    - Upload PDF documents or manually input requirements.
    - AI classifies and extracts requirements into **Functional**, **Non-Functional**, and **Technical Specifications**.
    - Output displayed in thick-bordered, easy-to-read text box.
2. **Code Generation Tab**
    - Enter code requirements.
    - Select programming language, framework, and coding style (OOP, Functional, Procedural).
    - AI generates the requested code.
    - Generates unit tests for the code in the same tab.
3. **User Interface**
    - Tab-based layout for better organization.
    - Styled with CSS for cards, shadows, rounded corners, and colored headers.
    - Interactive outputs: editable code blocks for easy copy-paste.

---

8. System Architecture

Flow:

1. User uploads PDF or inputs text.
2. AI processes text using **Granite 3.2 2B-instruct**.
3. AI outputs categorized requirements.
4. For code generation:
    - User inputs prompt + selects language/framework/style.
    - AI generates code and displays in output box.

   - Optional unit test generation for validation.

5. UI updates with thick-bordered boxes for readability.

Diagram:

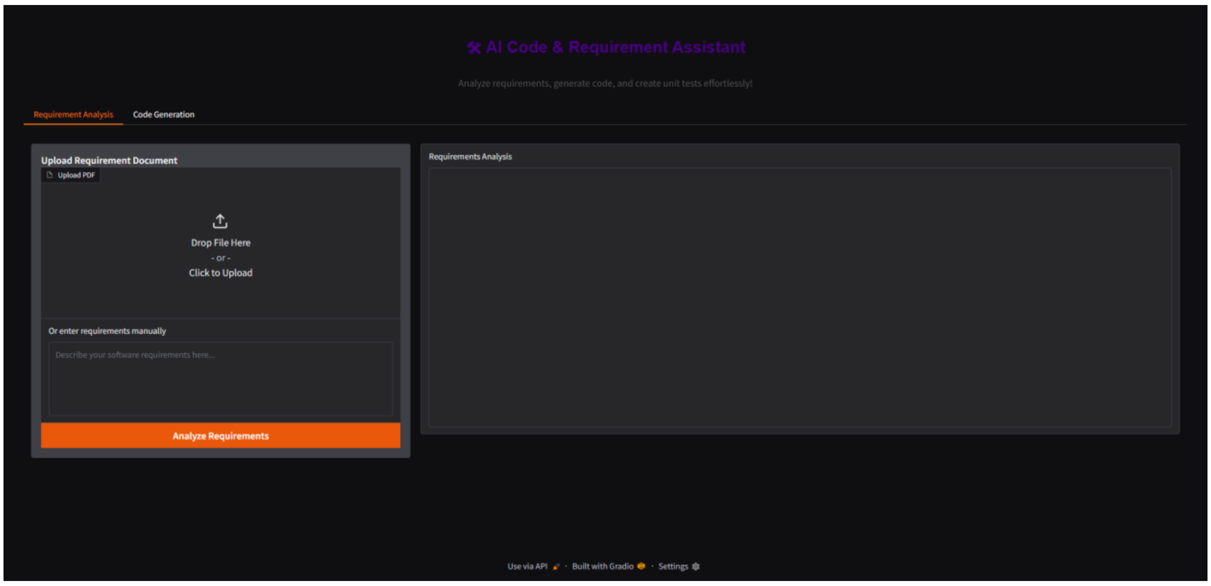
User --> Gradio UI --> Requirement Extraction --> AI Model (Granite 3.2)

--> Categorized Output --> Code Generation --> Unit Tests

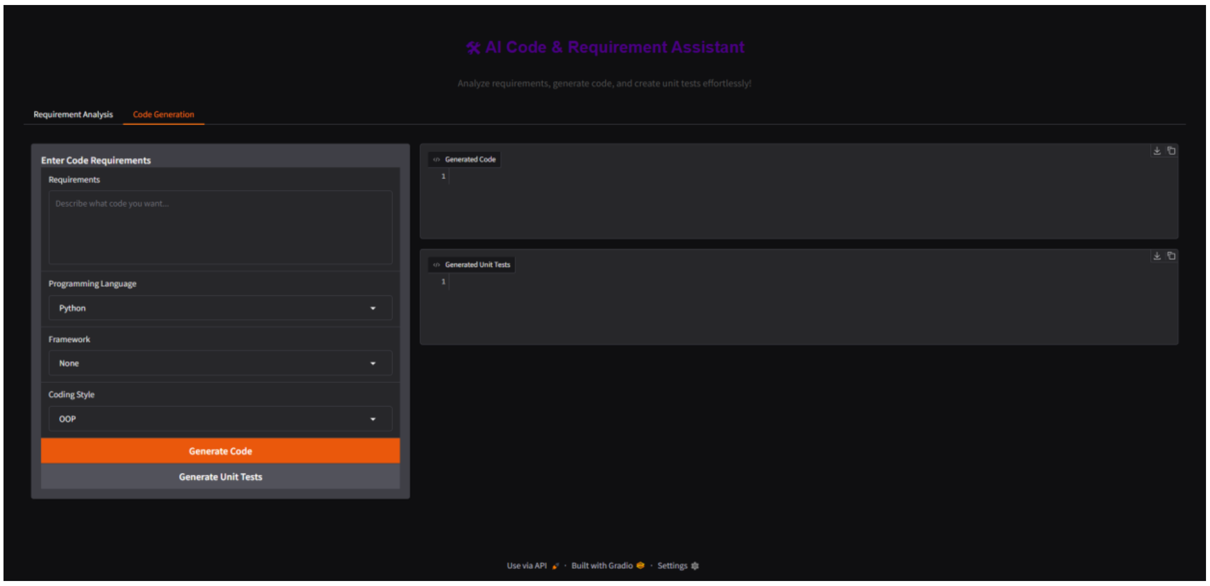
---

9. Screenshots

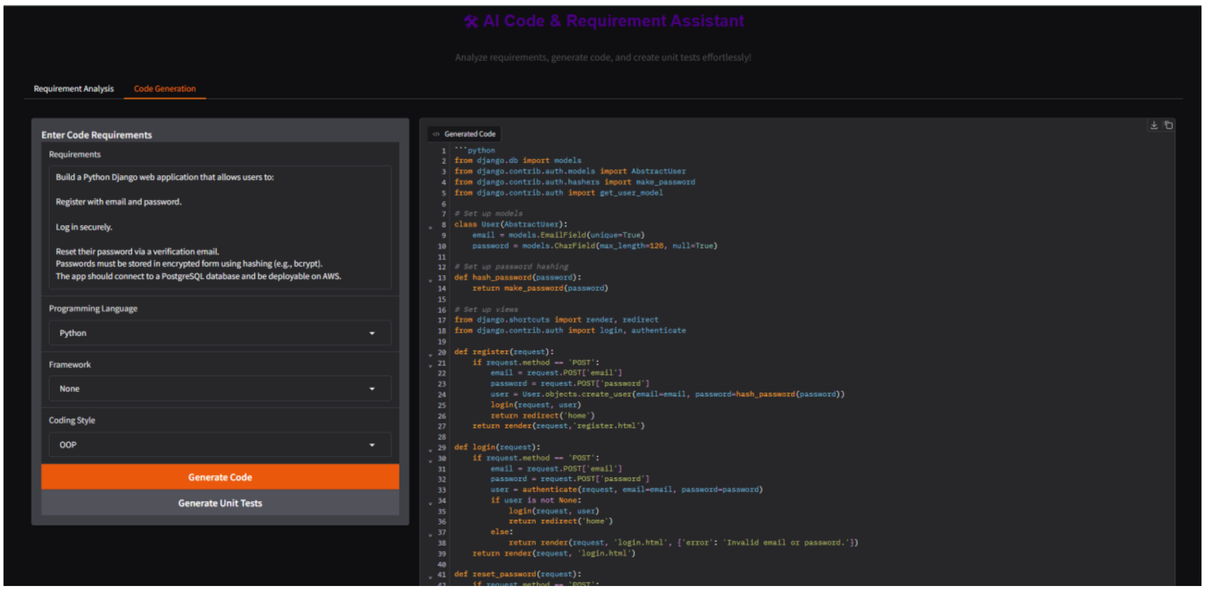
- **Requirement Analysis Tab**


AI Code & Requirement Assistant
Analyze requirements, generate code, and create unit tests effortlessly!
Requirement Analysis
Code Generation
Upload Requirement Document
Upload PDF
Drop File Here
- or -
Click to Upload
Or enter requirements manually
Describe your software requirements here...
Analyze Requirements
Requirements Analysis
Use via API · Built with Gradio · Settings

- **Code Generation Tab**



- **Generated Code & Tests Outputs**


🛠 AI Code & Requirement Assistant
Analyze requirements, generate code, and create unit tests effortlessly!
Requirement Analysis
Code Generation
Enter Code Requirements
Generate Code
Generate Unit Tests
Generated Code

Requirements

Build a Python Django web application that allows users to:

Register with email and password.

Log in securely.

Reset their password via a verification email.
Passwords must be stored in encrypted form using hashing (e.g., bcrypt).
The app should connect to a PostgreSQL database and be deployable on AWS.

Programming Language: Python

Framework: None

Coding Style: OOP

~~~
```python
from django.db import models
from django.contrib.auth.models import AbstractUser
from django.contrib.auth.hashers import make_password
from django.contrib.auth import get_user_model

# Set up models
class User(AbstractUser):
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=128, null=True)

# Set up password hashing
def hash_password(password):
    return make_password(password)

# Set up views
from django.shortcuts import render, redirect
from django.contrib.auth import login, authenticate

def register(request):
    if request.method == 'POST':
        email = request.POST['email']
        password = request.POST['password']
        user = User.objects.create_user(email=email, password=hash_password(password))
        login(request, user)
        return redirect('home')
    return render(request,'register.html')

def login(request):
    if request.method == 'POST':
        email = request.POST['email']
        password = request.POST['password']
        user = authenticate(request, email=email, password=password)
        if user is not None:
            login(request, user)
            return redirect('home')
        else:
            return render(request, 'login.html', {'error': 'Invalid email or password.'})
    return render(request, 'login.html')

def reset_password(request):
~~~

---

10. Testing

- Verified PDF parsing with multi-page documents.
- Validated code generation for Python and JavaScript.
- Unit test generation tested with sample code snippets.
- Manual input tested with edge cases like incomplete or ambiguous requirements.

---

11. Limitations

- Code generation is limited to the capabilities of the Granite 3.2 model.
- Very large PDFs may require truncation due to token limits.
- Unit tests are basic and may require human review for complex scenarios.

---

12. Future Enhancements

- Integrate **real-time collaborative editing**.
- Add **more frameworks and libraries** for each language.
- Include **requirement versioning and history**.
- Implement **advanced test generation** (integration, UI tests).
- Add **cloud hosting** for enterprise access.

---

13. References

- Gradio Documentation
- PyTorch
- HuggingFace Transformers
- PyPDF2 Documentation
