🧾 Requirement Code Traceablity with Chat – Final Project Report

---

📘 1. Overview

This project is a **Module Analyzer with Chat**, a web-based application that allows users to analyze a GitHub repository, extract information about specific modules, compare it with a requirements document (PDF), and interact with the codebase via a chatbot. It integrates AI models for summarization and suggestion generation, enhancing developer understanding and productivity.

⚙️ 2. Core Features

🔍 GitHub Repository Analyzer

- Accepts a GitHub repository URL and an optional module name.
- Clones (or pulls) the repository to a local directory.
- Parses source files (`.py`, `.js`, `.java`, `.cpp`, `.ts`) to extract:
    - Functions
    - Code comments
- Matches functions and comments with the provided module name using regex-based keyword search.

📝 PDF Requirement Comparison

- Accepts a PDF file (like an SRS or design document).

- Extracts subheadings from the PDF.
- Compares these subheadings with the module-related code found in the repository.
- Displays matched and unmatched subheadings to highlight implementation gaps.

💬 AI-Powered Chatbot

- Built-in chatbot allows users to ask questions about the analyzed code.
- Uses OpenRouter API and the **Mistral** model to generate accurate, contextual answers.
- Useful for understanding code structure, design decisions, or functionality without reading the full codebase.

📄 Code Summarizer

- Automatically summarizes key parts of the codebase using the `facebook/bart-large-cnn` model from HuggingFace.
- Generates concise, human-readable summaries of large files/modules.

✨ Suggestion Generator

- Uses AI (via OpenRouter) to suggest improvements to the code.
- Suggestions include:
    - Best practices
    - Refactoring tips
    - Code clarity enhancements

🆕 3. Newly Added Features

🚨 Code Issue Generation

Purpose: Detect common code issues or "code smells."

- Activated via "Generate Code Issues" button.
- Scans each code file for:
  - Missing docstrings
  - Bad naming practices
  - Deeply nested blocks
  - Unused variables
  - Other maintainability or readability issues
- Displays a list of potential problems in the UI.
- Helps improve overall code quality.

📑 Requirement Checker (Subheading Checker)

Purpose: Map PDF-based requirements to actual code implementation.

- Parses subheadings (e.g., `Login Module:`) from uploaded PDFs.
- Checks if each subheading is represented in the analyzed module report.
- Outputs:
  - ✅ Implemented subheadings
  - ❌ Not implemented subheadings
- Ensures **requirement traceability** – especially useful in software engineering validation processes.

▶️ 4. How to Run the Application

📁 Folder Structure (Typical)

csharp
CopyEdit
```
project/
│
├── app.py                  # Flask backend
├── templates/
│   └── index.html          # Main HTML template
├── static/
│   └── style.css           # Optional styling
├── requirements.txt        # Python dependencies
```

🔧 Step-by-Step Instructions

Clone the Repository

bash
CopyEdit
```
git clone <your-repo-url>
cd <repo-folder>
```

1.

Install Dependencies Make sure you have Python 3.8+ installed.

bash
CopyEdit
```
pip install -r requirements.txt
```

2.
3. **Set API Keys**
   - You need an **OpenRouter API key** to use Mistral for AI chat and suggestions.
   - Place it in the code (`app.py`) under the `OPENROUTER_API_KEY` variable.

Run the Flask App

bash
CopyEdit
```
python app.py
```

4.
5. **Access in Browser** Open `http://127.0.0.1:5000` in your browser.

📦 5. Dependencies (requirements.txt)

Your `requirements.txt` should include:

txt
CopyEdit
```
Flask
transformers
pdfminer.six
requests
openai
huggingface_hub
torch
PyPDF2
```

(You can tailor this further based on your exact environment.)

Team Contributions

- **Raghuram**: Developed the Flask backend; implemented repository cloning and parsing; generated code suggestions and vulnerability insights.
- **Satyannarayana**: Analyzed and summarized client requirement documents.
- **Nayak**: Focused on code parsing and evaluating code optimality.
- **Raghuveer**: Created code summaries using Python's Abstract Syntax Tree (AST).
- **Sampath**: Designed the user interface and implemented bidirectional link visualization.
- **Aseem**: Conducted testing and validated traceability links.
- **Pranav**: Integrated chatbot functionality and managed project documentation.