CSCN8010
GROUP#4'S FINAL PROJECT
AI-ENHANCED SELF-SERVICE PORTAL FOR STUDENT AFFAIRS

- Aiswarya Thekkuveettil - 8993970
- Rohit Krishnamurthy Iyer - 8993045
- Vishnu Sivaraj - 9025320
- Cemil Caglar Yapici - 9081058

Project Purpose and Context

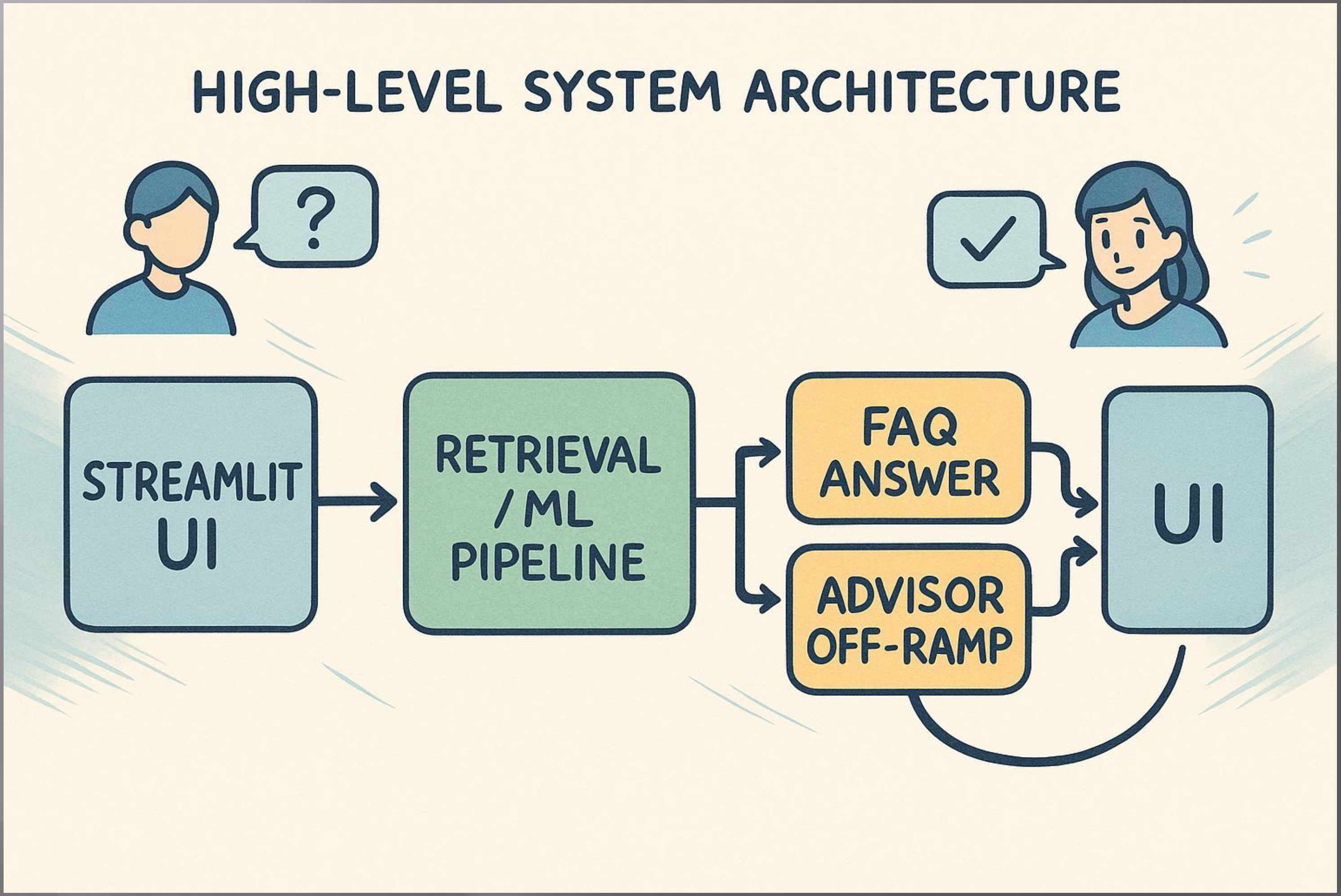
- As it knows that Student Success Advisors (SSAs) are overwhelmed by repetitive, low-complexity inquiries, especially after recent staff reductions. Students often ask the same FAQ-type questions, consuming advisor time.
- Now our goal is: Develop a self-service **AI chatbot** to handle these frequently asked questions automatically, reducing SSA workload and wait times for students.
- Scope: The chatbot will answer common Student Affairs queries (e.g. enrollment, fees, course changes) and **escalate to a human advisor** for complex or sensitive issues (a safety “off-ramp”). This ensures students still get personal help when needed.

Solution Overview & System Architecture

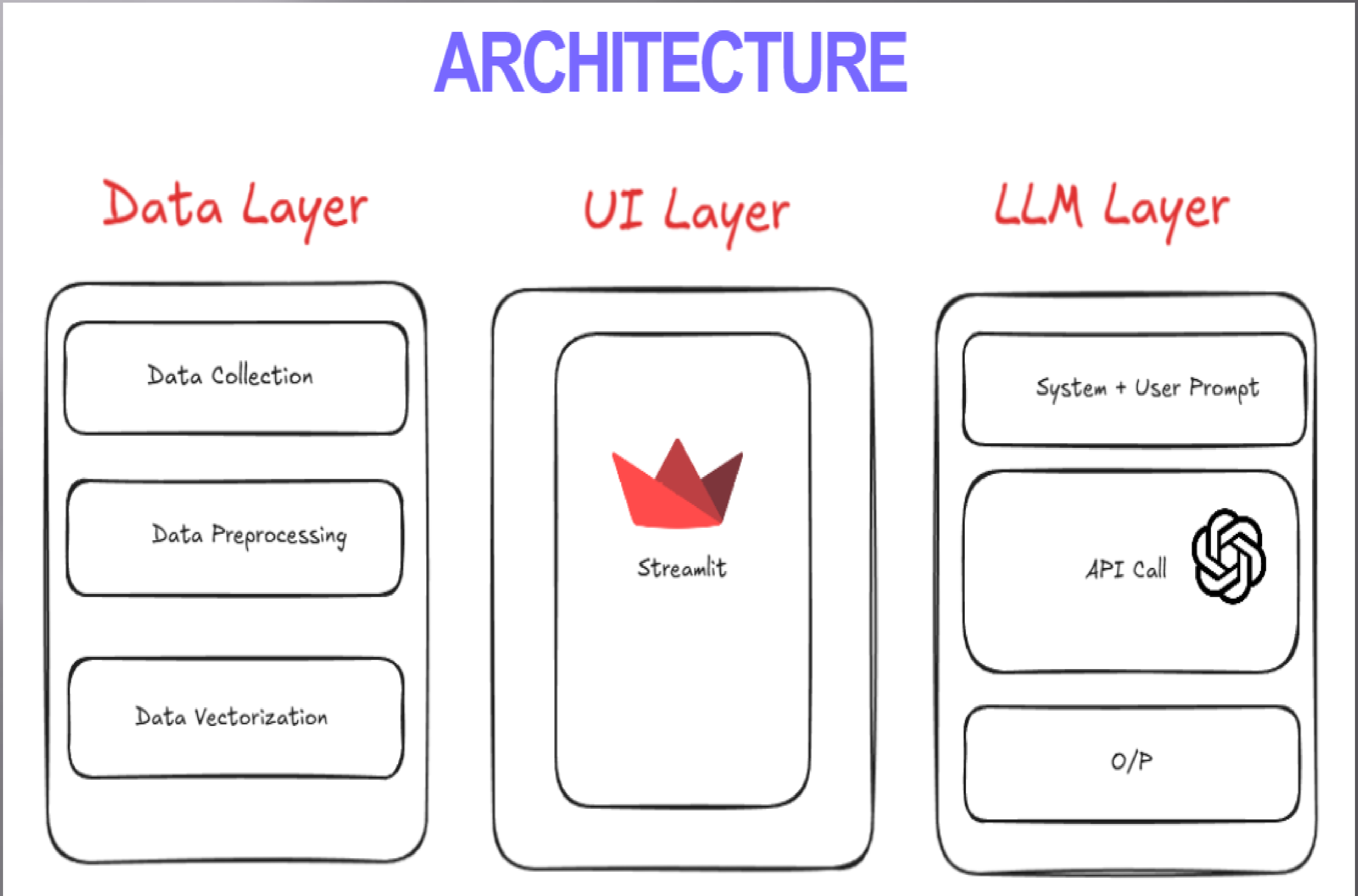
- **Solution:** A **web-based chatbot portal** where students ask questions and receive instant answers. Built with a friendly UI (using Streamlit) on the front-end and an intelligent FAQ retrieval system on the back-end.

- **Architecture Highlights:** Student questions flow from the UI to an NLP-based engine that finds answers in an FAQ knowledge base. If an inquiry triggers certain keywords or appears too complex (e.g. emotional distress), the system will prompt the student to contact an advisor (escalation).

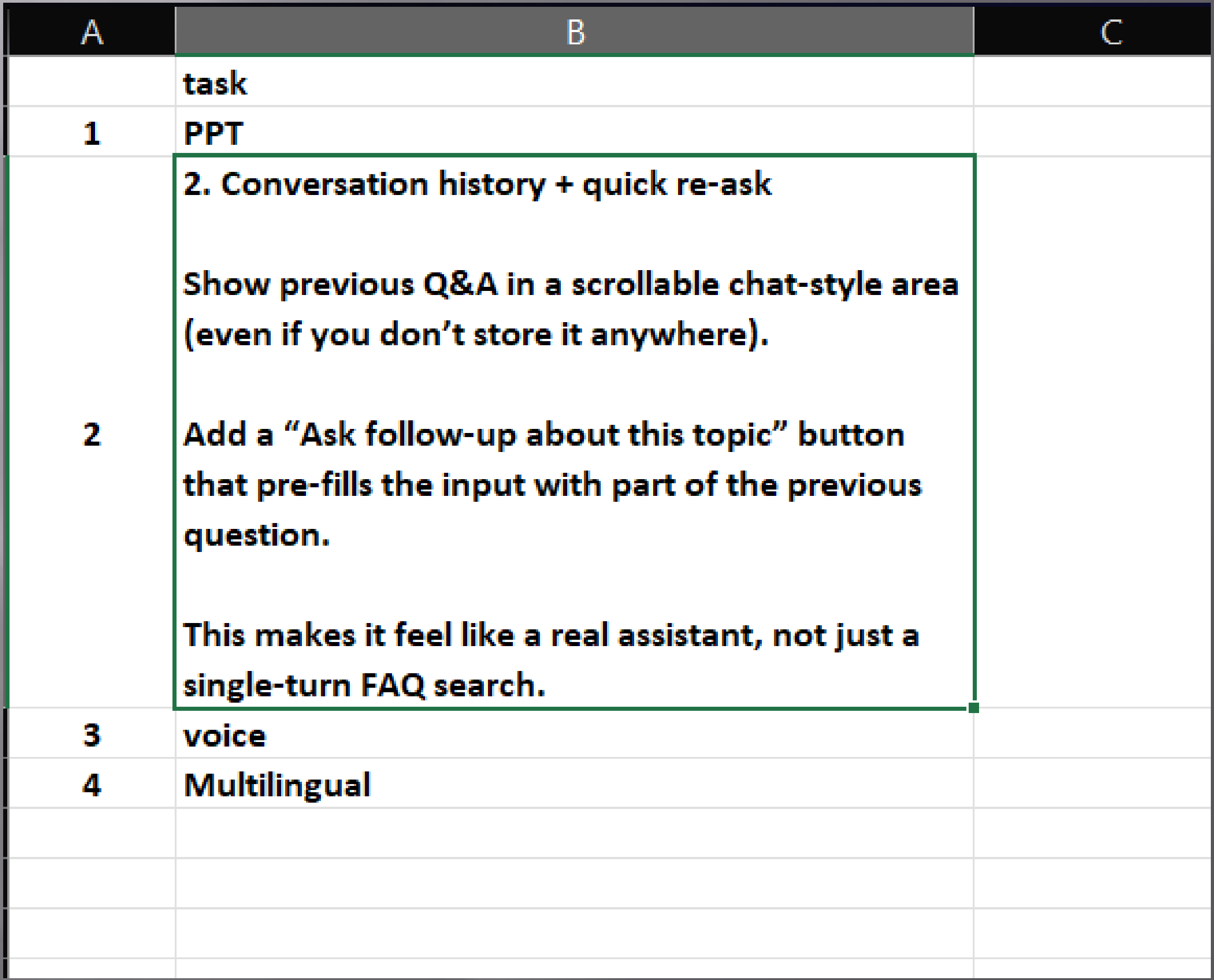
Architecture diagram
(Made by AI for AI project ☺)



Good thing is: Streamlit is included VS Code/Python by .venv and it can be run locally also.

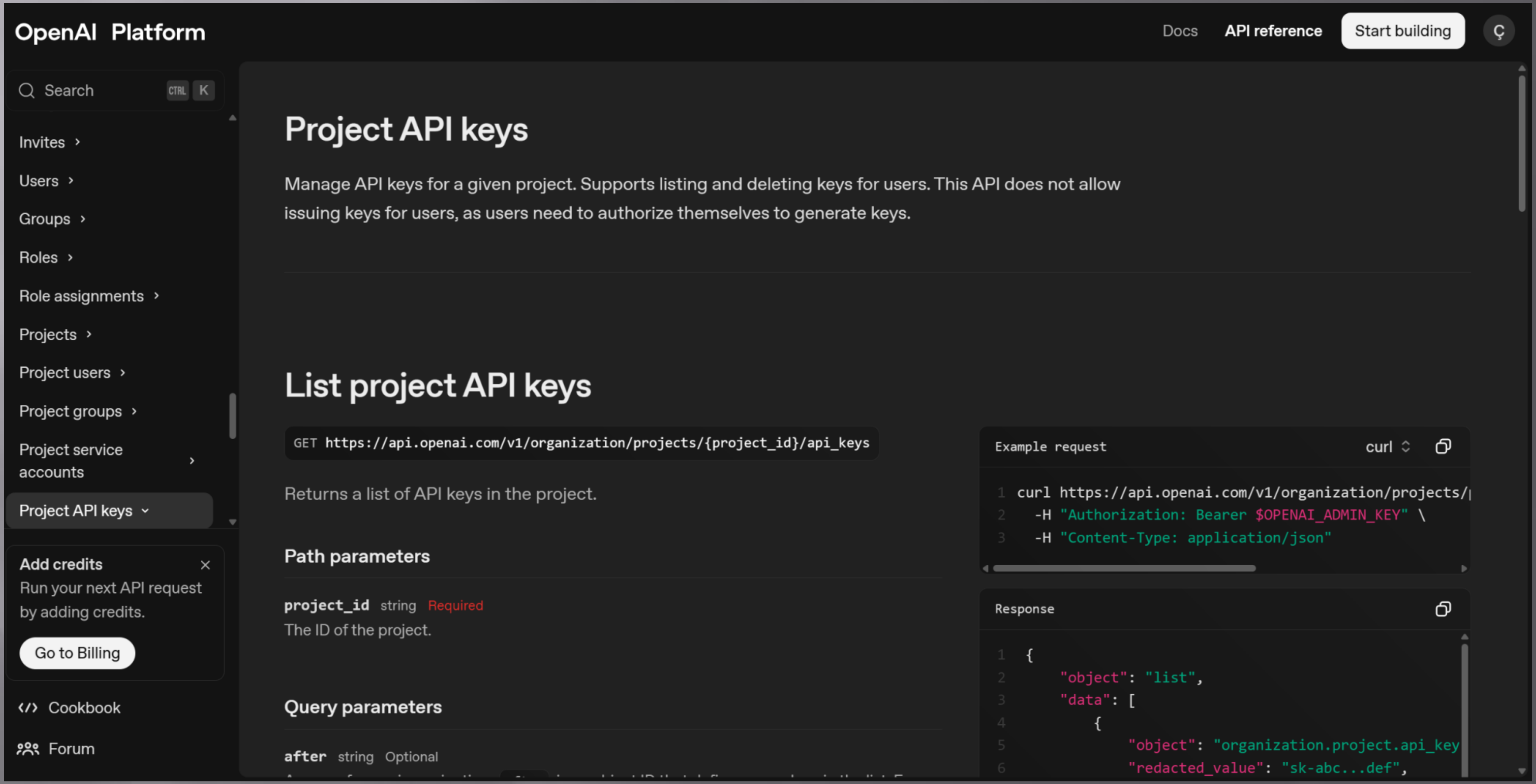


Main Tasks to Do



A	B	C
	task	
1	PPT	
2	2. Conversation history + quick re-ask Show previous Q&A in a scrollable chat-style area (even if you don’t store it anywhere). Add a “Ask follow-up about this topic” button that pre-fills the input with part of the previous question. This makes it feel like a real assistant, not just a single-turn FAQ search.	
3	voice	
4	Multilingual	

OpenAI API KEY



Project API keys

Manage API keys for a given project. Supports listing and deleting keys for users. This API does not allow issuing keys for users, as users need to authorize themselves to generate keys.

List project API keys

```
GET https://api.openai.com/v1/organization/projects/{project_id}/api_keys
```

Returns a list of API keys in the project.

Path parameters

project_id string Required

The ID of the project.

Query parameters

after string Optional

Example request curl

```
curl https://api.openai.com/v1/organization/projects/
  -H "Authorization: Bearer $OPENAI_ADMIN_KEY" \
  -H "Content-Type: application/json"
```

Response

```
{
    "object": "list",
    "data": [
        {
            "object": "organization.project.api_key
            "redacted_value": "sk-abc...def",
```

Main Components

- Python:

Core language for development (backend logic, data processing, and integration of ML/NLP components).

- CSV:

(for main datasets' format, also known as excel files)

- UI & Visualization:

Streamlit web app where students type questions and view answers. Used to create the interactive web application interface for the chatbot (provides a quick way to build a frontend in Python)

Main Components

- Jupyter Notebook:

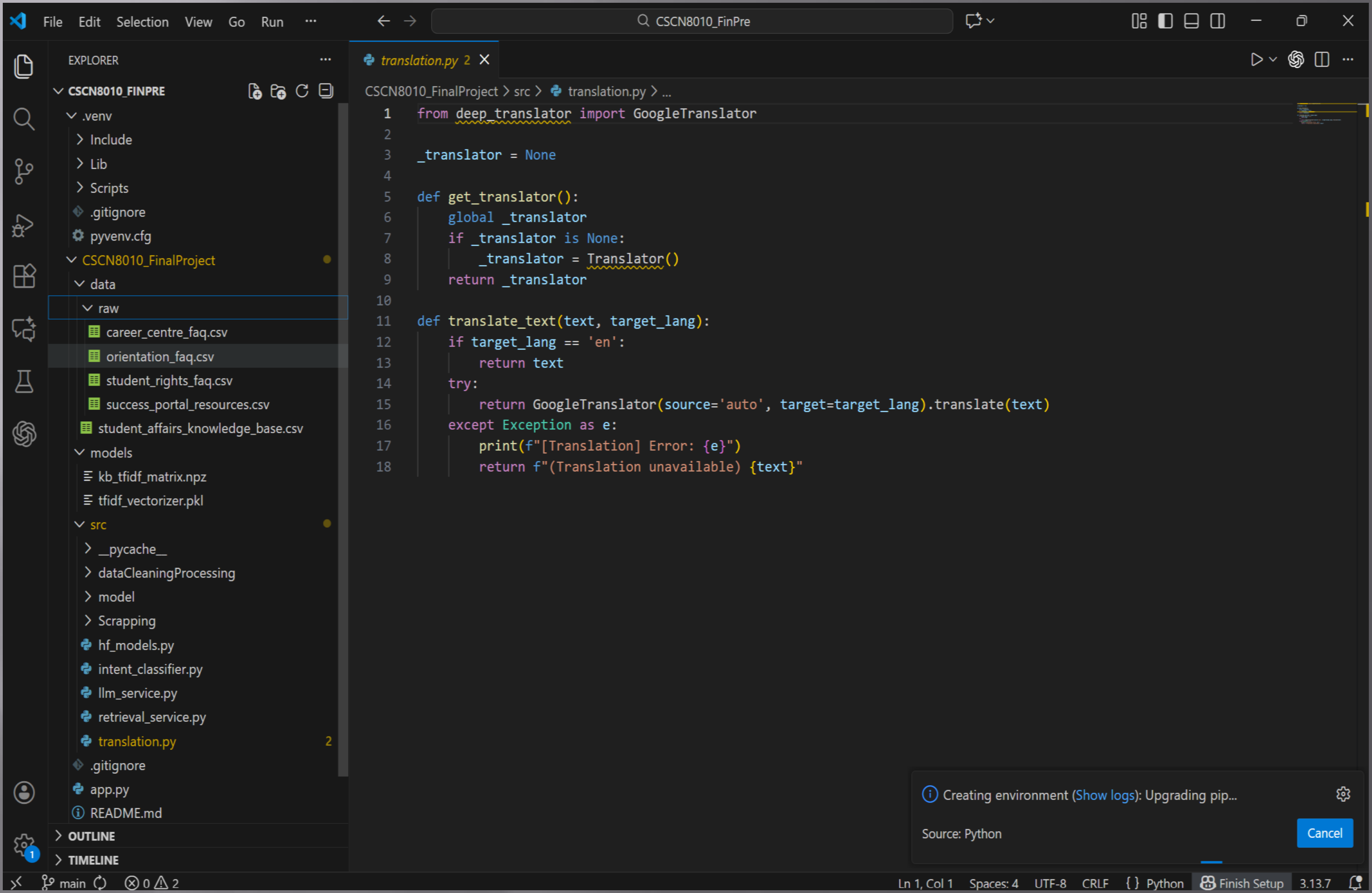
Utilized during development for prototyping and experimenting with NLP models and data (initial model training, testing queries).

Note: For Visualization Jupyter Notebook is our 2nd option like alternative if streamlit causes issues.

- VS Code:

Primary code editor/IDE for the project, for implementing the application modules (helped in writing Python scripts, managing the project repository).

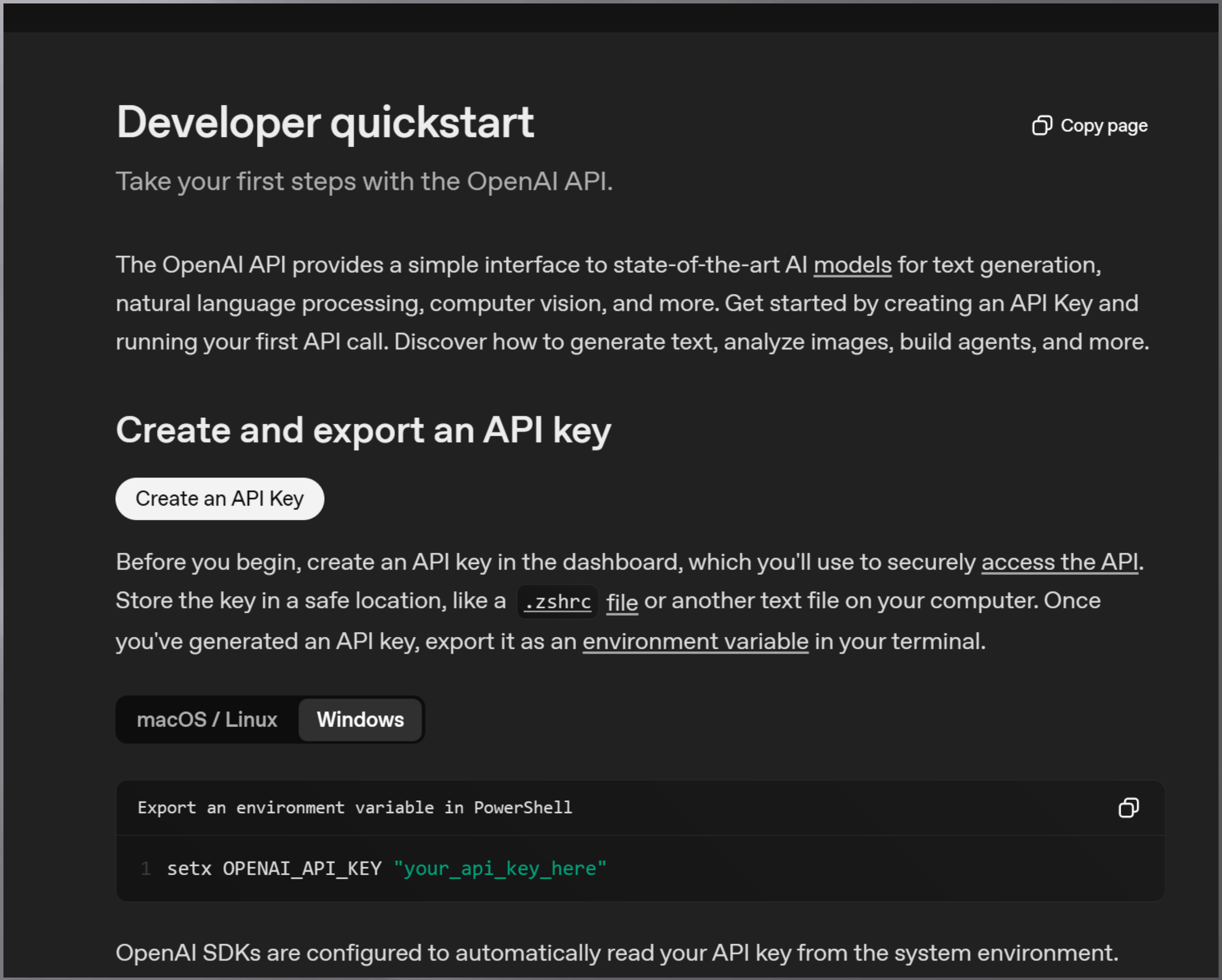
Main Screenshots from our project



```python
from deep_translator import GoogleTranslator

_translator = None

def get_translator():
    global _translator
    if _translator is None:
        _translator = Translator()
    return _translator

def translate_text(text, target_lang):
    if target_lang == 'en':
        return text
    try:
        return GoogleTranslator(source='auto', target=target_lang).translate(text)
    except Exception as e:
        print(f"[Translation] Error: {e}")
        return f"(Translation unavailable) {text}"
```

Creating environment (Show logs): Upgrading pip...

Source: Python



Developer quickstart

Take your first steps with the OpenAI API.

The OpenAI API provides a simple interface to state-of-the-art AI models for text generation, natural language processing, computer vision, and more. Get started by creating an API Key and running your first API call. Discover how to generate text, analyze images, build agents, and more.

Create and export an API key

Before you begin, create an API key in the dashboard, which you'll use to securely access the API. Store the key in a safe location, like a `.zshrc` file or another text file on your computer. Once you've generated an API key, export it as an environment variable in your terminal.

Export an environment variable in PowerShell

```
setx OPENAI_API_KEY "your_api_key_here"
```

OpenAI SDKs are configured to automatically read your API key from the system environment.

Knowledge Base (Data Sources)

As we mentioned on previous pages we have datasets that we had taken from different sources in different formats. Let us explain:

- Student FAQ Documents (PDFs):

We gathered official FAQs from Student Affairs departments – e.g. a Registrar Office FAQ and a Student Fees FAQ – which contain common questions and answers. These PDFs cover topics like fee payments, course registration, OSAP, withdrawals, etc., forming the content the chatbot can draw from.

- CSV Datasets:

The team extracted and compiled Q&A pairs from these PDFs into structured CSV files (included in the project zip). The CSV format makes it easy to load data into the chatbot and perform searches

Knowledge Base (Data Sources)

- Data Usage:

At runtime, the chatbot indexes this content (using TF-IDF vectorization on questions). When a student asks something, the system compares the query against all known questions to retrieve the best match and its answer.

- Maintenance:

The design allows updating the knowledge base by simply adding new entries to the CSV or updating the source documents, so the chatbot can stay up-to-date as policies change.

Sources

- Openai.com
- VSCode documents.
- Stackoverflow
- Github