Project Implementation and Testing

Date	8th July 2024
Team ID	SWTID1720074725
Project Name	CodeXchange: An AI-Powered Code Translator Tool

Implementation

The implementation of CodeXchange involves several key steps to ensure a seamless, efficient, and user-friendly code translation experience. The project leverages Google Gemini AI for the core translation functionality and Streamlit for the user interface. Below are the main components and steps of the implementation:

1. Environment Setup

- **Install Dependencies**: Use a `requirements.txt` file to install necessary Python libraries such as `streamlit`, `google-generativeai`, and `python-dotenv`.
- **Configuration**: Set up environment variables using a `.env` file to securely manage the Google Gemini AI API key.

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

2. Google Gemini AI Integration

- **Configure API**: Load the API key from the environment variables and configure the Google Gemini AI API for use.
- **Model Selection**: Choose the appropriate model (`chat-bison-001`) for code translation tasks.
- **Translation Function**: Develop a function to interact with the AI model, sending the code snippet and receiving the translated code.

```
from dotenv import load_dotenv
import google.generativeai as palm
import os

```

```
load_dotenv()
gemini_api_key = os.getenv('GEMINI_API_KEY')
palm.configure(api_key=gemini_api_key)
model_name = "models/chat-bison-001"

def translate_code(code_snippet, target_language):
    instruction = "Give fully runnable code. Include syntax highlighting. Give only the translated code."
    prompt = f"Translate the following code to {target_language}:\n\n{code_snippet}"
    response = palm.chat(model=model_name, messages=[instruction, prompt])
    return response.candidates[0]["content"]
```

3. Streamlit User Interface

- **Setup UI Components**: Use Streamlit to create the web interface for user inputs and display outputs.
- **User Inputs**: Provide text areas and dropdown menus for users to input code snippets and select the target programming language.
- **Display Results**: Show the translated code with syntax highlighting and formatting.

```
import streamlit as st

def main():
    st.title("CodeXchange: AI-Powered Code Translation Tool")
    source_code_snippet = st.text_area("Enter the code snippet you want to translate:")
    languages = ["Python", "Java", "C++"]
    target_language = st.selectbox("Select the target programming language:", languages)

    if st.button("Translate"):
        if source_code_snippet.strip():
            with st.spinner("Translating code..."):
                try:
                    translated_code = translate_code(source_code_snippet, target_language)
                    cleaned_code = extract_code(translated_code)
                    st.success("Code translation complete!")
                    st.code(cleaned_code, language=target_language.lower())
                except Exception as e:
                    st.error(f"Error translating code: {e}")
        else:
            st.error("Please enter a code snippet.")

```

```
if __name__ == "__main__":
    main()
```

4. Command-Line Interface (CLI)

- **CLI Support**: Implement a command-line interface for users who prefer to work directly from the terminal, allowing for flexible and quick translations.

```
def commandLine(targetLanguage):
    inputFileName = outputFileName = ""
    if len(sys.argv) < 4:
        outputFileName = "converted.txt"
    else:
        outputFileName = sys.argv[3]
    inputFileName = sys.argv[2]
    code = ""
    with open(inputFileName,"r") as inputFile:
        code = inputFile.read()

    with open(outputFileName,"w") as outputFile:
        print("\rConverting...")
        translated_code = extractCode(translate_code(code,targetLanguage))
        print("\rConverted...")
        print("\rSaving...")
        outputFile.write(translated_code)
        print("\rOutput Saved...\n\n")
        print(f"{targetLanguage.title()} Code:")
                print(translated_code)
```

The implementation of CodeXchange involves setting up the environment, integrating with Google Gemini AI for translation, building a user-friendly interface with Streamlit, and providing a command-line interface for flexibility. This comprehensive approach ensures accurate, efficient, and accessible code translations for developers.