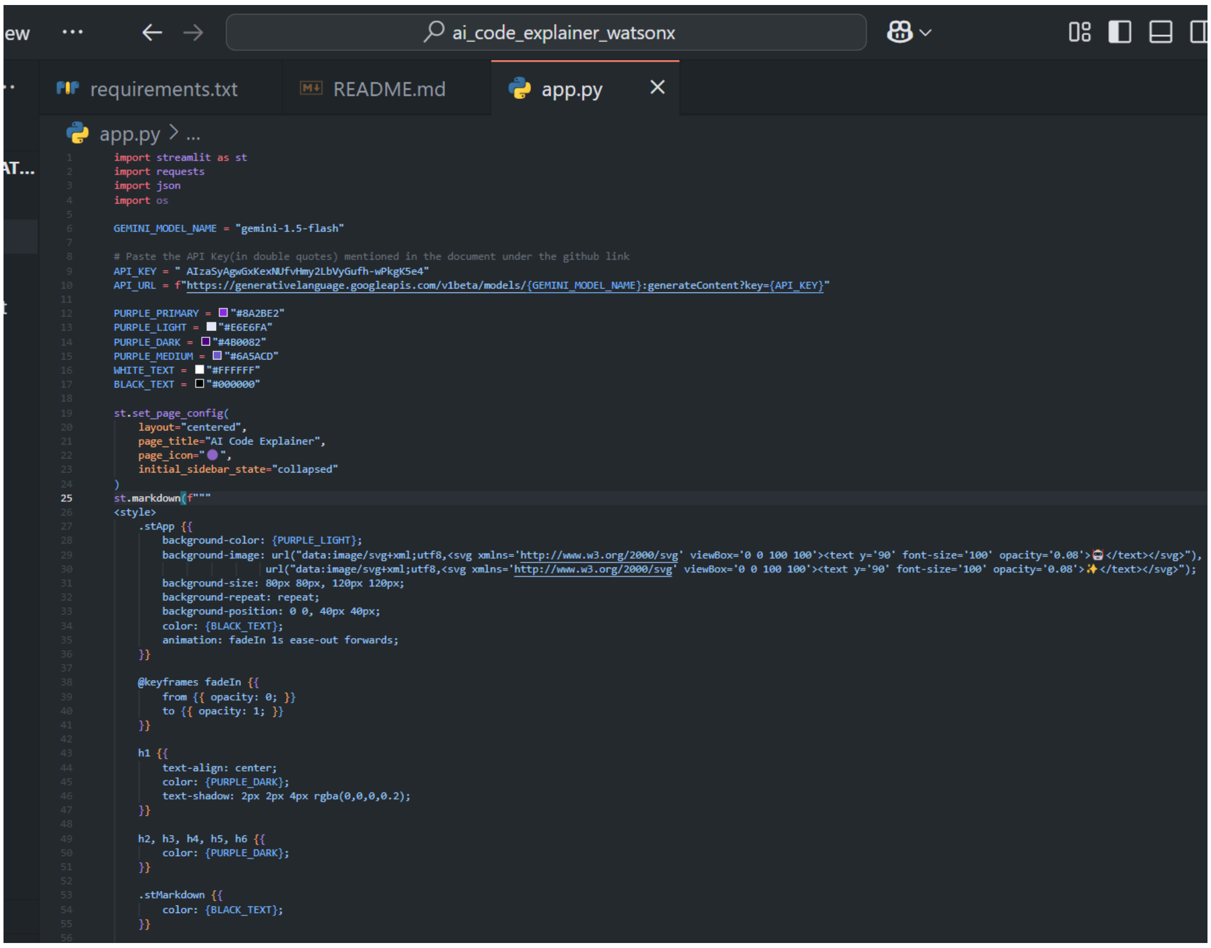
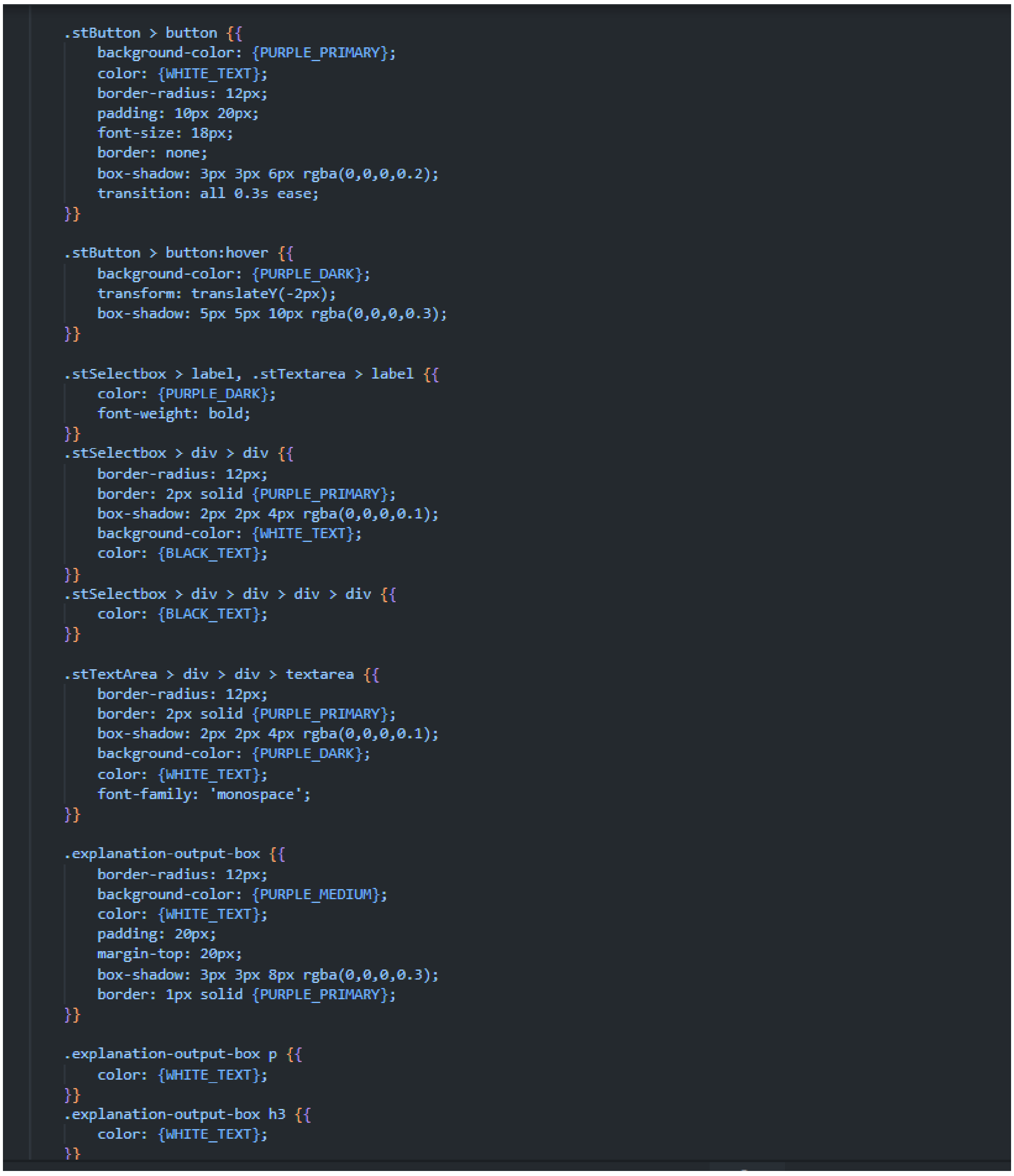
NAME: B.HARITHA

COLLEGE NAME: VIT VELLORE

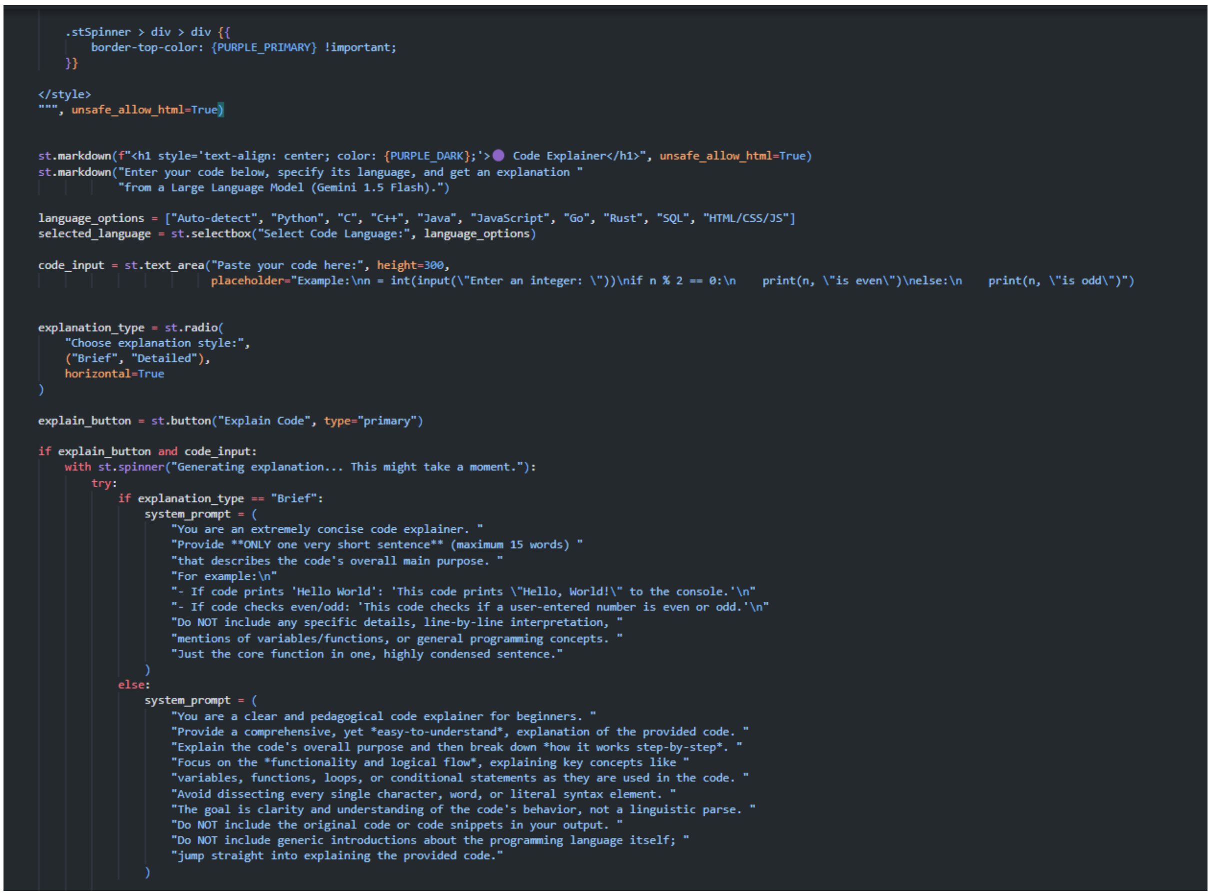
CODE



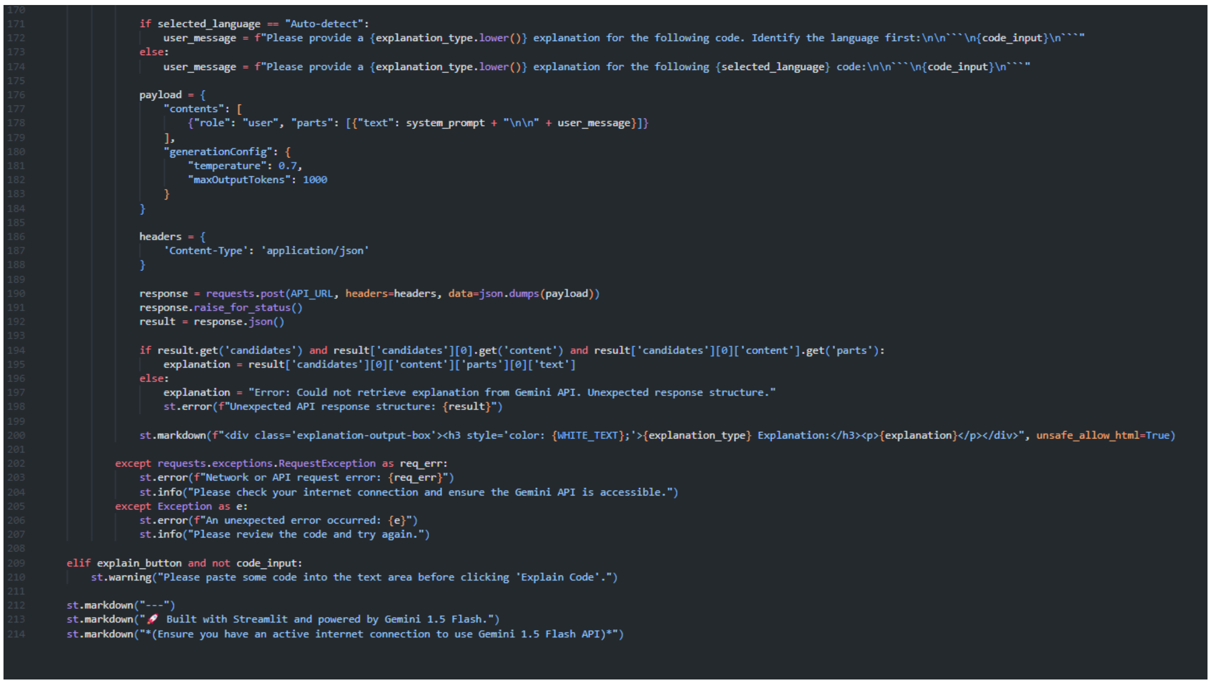
```python
import streamlit as st
import requests
import json
import os

GEMINI_MODEL_NAME = "gemini-1.5-flash"

# Paste the API Key(in double quotes) mentioned in the document under the github link
API_KEY = " AIzaSyAgwGxKexNUfvHmy2LbVyGufh-wPkgK5e4"
API_URL = f"https://generativelanguage.googleapis.com/v1beta/models/{GEMINI_MODEL_NAME}:generateContent?key={API_KEY}"

PURPLE_PRIMARY = "#8A2BE2"
PURPLE_LIGHT = "#E6E6FA"
PURPLE_DARK = "#4B0082"
PURPLE_MEDIUM = "#6A5ACD"
WHITE_TEXT = "#FFFFFF"
BLACK_TEXT = "#000000"

st.set_page_config(
    layout="centered",
    page_title="AI Code Explainer",
    page_icon="🟣",
    initial_sidebar_state="collapsed"
)
st.markdown(f"""
<style>
    .stApp {{
        background-color: {PURPLE_LIGHT};
        background-image: url("data:image/svg+xml;utf8,<svg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 100 100'><text y='90' font-size='100' opacity='0.08'>🤖</text></svg>"),
                          url("data:image/svg+xml;utf8,<svg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 100 100'><text y='90' font-size='100' opacity='0.08'>✨</text></svg>");
        background-size: 80px 80px, 120px 120px;
        background-repeat: repeat;
        background-position: 0 0, 40px 40px;
        color: {BLACK_TEXT};
        animation: fadeIn 1s ease-out forwards;
    }}

    @keyframes fadeIn {{
        from {{ opacity: 0; }}
        to {{ opacity: 1; }}
    }}

    h1 {{
        text-align: center;
        color: {PURPLE_DARK};
        text-shadow: 2px 2px 4px rgba(0,0,0,0.2);
    }}

    h2, h3, h4, h5, h6 {{
        color: {PURPLE_DARK};
    }}

    .stMarkdown {{
        color: {BLACK_TEXT};
    }}

```



```
.stButton > button {{
    background-color: {PURPLE_PRIMARY};
    color: {WHITE_TEXT};
    border-radius: 12px;
    padding: 10px 20px;
    font-size: 18px;
    border: none;
    box-shadow: 3px 3px 6px rgba(0,0,0,0.2);
    transition: all 0.3s ease;
}}

.stButton > button:hover {{
    background-color: {PURPLE_DARK};
    transform: translateY(-2px);
    box-shadow: 5px 5px 10px rgba(0,0,0,0.3);
}}

.stSelectbox > label, .stTextarea > label {{
    color: {PURPLE_DARK};
    font-weight: bold;
}}
.stSelectbox > div > div {{
    border-radius: 12px;
    border: 2px solid {PURPLE_PRIMARY};
    box-shadow: 2px 2px 4px rgba(0,0,0,0.1);
    background-color: {WHITE_TEXT};
    color: {BLACK_TEXT};
}}
.stSelectbox > div > div > div > div {{
    color: {BLACK_TEXT};
}}

.stTextArea > div > div > textarea {{
    border-radius: 12px;
    border: 2px solid {PURPLE_PRIMARY};
    box-shadow: 2px 2px 4px rgba(0,0,0,0.1);
    background-color: {PURPLE_DARK};
    color: {WHITE_TEXT};
    font-family: 'monospace';
}}

.explanation-output-box {{
    border-radius: 12px;
    background-color: {PURPLE_MEDIUM};
    color: {WHITE_TEXT};
    padding: 20px;
    margin-top: 20px;
    box-shadow: 3px 3px 8px rgba(0,0,0,0.3);
    border: 1px solid {PURPLE_PRIMARY};
}}

.explanation-output-box p {{
    color: {WHITE_TEXT};
}}
.explanation-output-box h3 {{
    color: {WHITE_TEXT};
}}
```



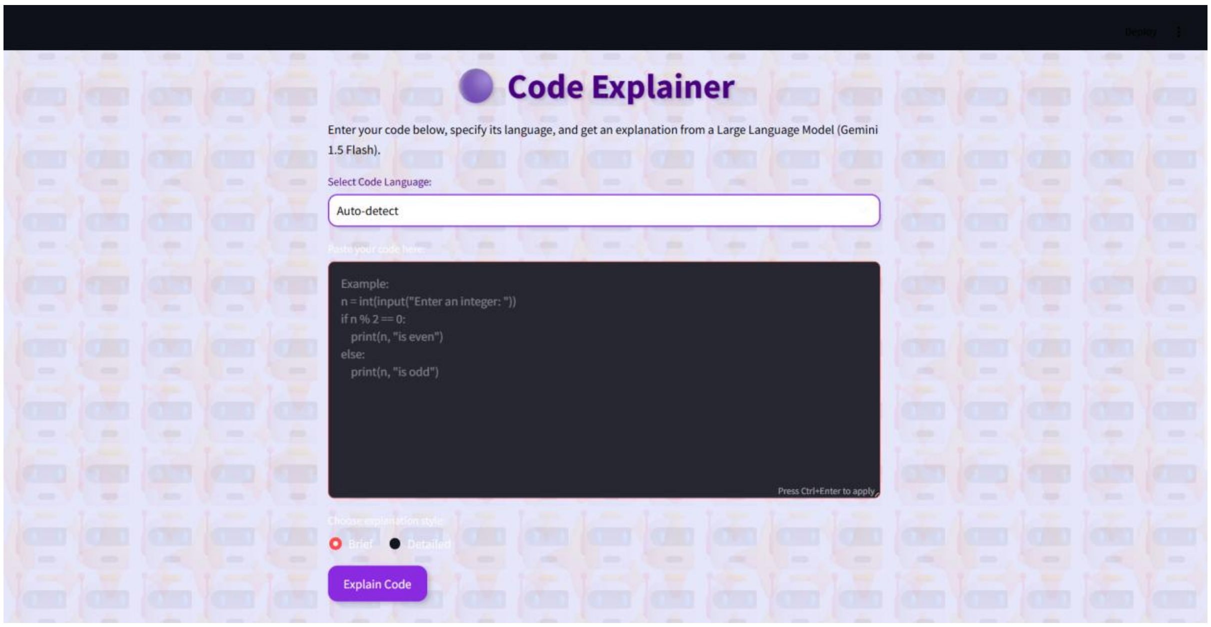
```
        .stSpinner > div > div {{
            border-top-color: {PURPLE_PRIMARY} !important;
        }}

    </style>
    """, unsafe_allow_html=True)


    st.markdown(f"<h1 style='text-align: center; color: {PURPLE_DARK};'>🟣 Code Explainer</h1>", unsafe_allow_html=True)
    st.markdown("Enter your code below, specify its language, and get an explanation "
                "from a Large Language Model (Gemini 1.5 Flash).")

    language_options = ["Auto-detect", "Python", "C", "C++", "Java", "JavaScript", "Go", "Rust", "SQL", "HTML/CSS/JS"]
    selected_language = st.selectbox("Select Code Language:", language_options)

    code_input = st.text_area("Paste your code here:", height=300,
                              placeholder="Example:\nn = int(input(\"Enter an integer: \"))\nif n % 2 == 0:\n    print(n, \"is even\")\nelse:\n    print(n, \"is odd\")")


    explanation_type = st.radio(
        "Choose explanation style:",
        ("Brief", "Detailed"),
        horizontal=True
    )

    explain_button = st.button("Explain Code", type="primary")

    if explain_button and code_input:
        with st.spinner("Generating explanation... This might take a moment."):
            try:
                if explanation_type == "Brief":
                    system_prompt = (
                        "You are an extremely concise code explainer. "
                        "Provide **ONLY one very short sentence** (maximum 15 words) "
                        "that describes the code's overall main purpose. "
                        "For example:\n"
                        "- If code prints 'Hello World': 'This code prints \"Hello, World!\" to the console.'\n"
                        "- If code checks even/odd: 'This code checks if a user-entered number is even or odd.'\n"
                        "Do NOT include any specific details, line-by-line interpretation, "
                        "mentions of variables/functions, or general programming concepts. "
                        "Just the core function in one, highly condensed sentence."
                    )
                else:
                    system_prompt = (
                        "You are a clear and pedagogical code explainer for beginners. "
                        "Provide a comprehensive, yet *easy-to-understand*, explanation of the provided code. "
                        "Explain the code's overall purpose and then break down *how it works step-by-step*. "
                        "Focus on the *functionality and logical flow*, explaining key concepts like "
                        "variables, functions, loops, or conditional statements as they are used in the code. "
                        "Avoid dissecting every single character, word, or literal syntax element. "
                        "The goal is clarity and understanding of the code's behavior, not a linguistic parse. "
                        "Do NOT include the original code or code snippets in your output. "
                        "Do NOT include generic introductions about the programming language itself; "
                        "jump straight into explaining the provided code."
                    )
```



```
                if selected_language == "Auto-detect":
                    user_message = f"Please provide a {explanation_type.lower()} explanation for the following code. Identify the language first:\n\n```\n{code_input}\n```"
                else:
                    user_message = f"Please provide a {explanation_type.lower()} explanation for the following {selected_language} code:\n\n```\n{code_input}\n```"

                payload = {
                    "contents": [
                        {"role": "user", "parts": [{"text": system_prompt + "\n\n" + user_message}]}
                    ],
                    "generationConfig": {
                        "temperature": 0.7,
                        "maxOutputTokens": 1000
                    }
                }

                headers = {
                    'Content-Type': 'application/json'
                }

                response = requests.post(API_URL, headers=headers, data=json.dumps(payload))
                response.raise_for_status()
                result = response.json()

                if result.get('candidates') and result['candidates'][0].get('content') and result['candidates'][0]['content'].get('parts'):
                    explanation = result['candidates'][0]['content']['parts'][0]['text']
                else:
                    explanation = "Error: Could not retrieve explanation from Gemini API. Unexpected response structure."
                    st.error(f"Unexpected API response structure: {result}")

                st.markdown(f"<div class='explanation-output-box'><h3 style='color: {WHITE_TEXT};'>{explanation_type} Explanation:</h3><p>{explanation}</p></div>", unsafe_allow_html=True)

            except requests.exceptions.RequestException as req_err:
                st.error(f"Network or API request error: {req_err}")
                st.info("Please check your internet connection and ensure the Gemini API is accessible.")
            except Exception as e:
                st.error(f"An unexpected error occurred: {e}")
                st.info("Please review the code and try again.")

    elif explain_button and not code_input:
        st.warning("Please paste some code into the text area before clicking 'Explain Code'.")

    st.markdown("---")
    st.markdown("🚀 Built with Streamlit and powered by Gemini 1.5 Flash.")
    st.markdown("*(Ensure you have an active internet connection to use Gemini 1.5 Flash API)*")
```

OUTPUT

This is how the interface looks:



Code Explainer

Enter your code below, specify its language, and get an explanation from a Large Language Model (Gemini 1.5 Flash).

Select Code Language:

Auto-detect

Paste your code here:

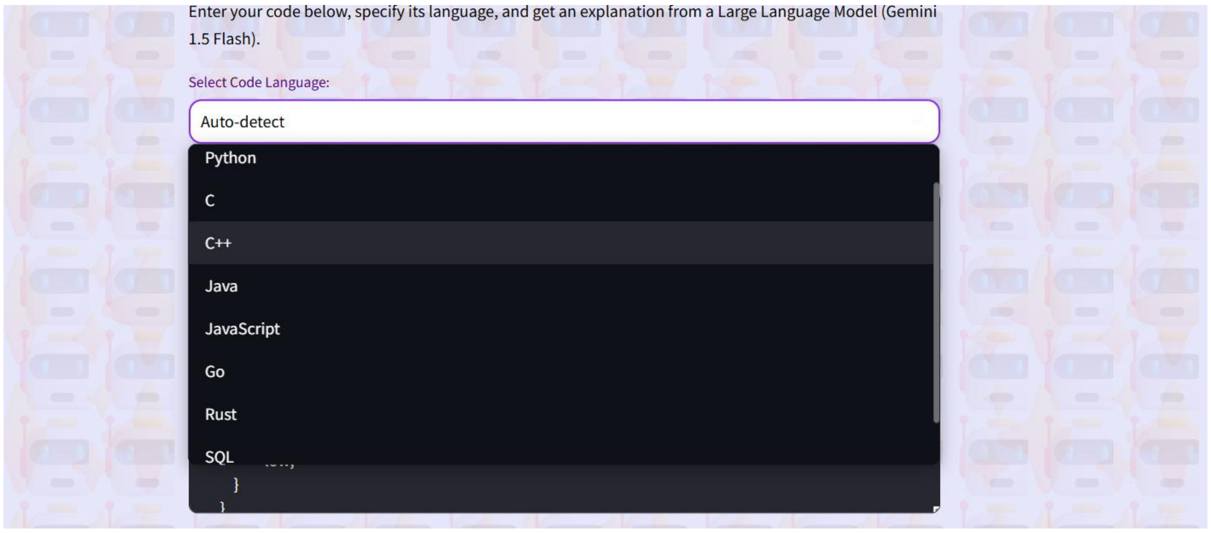
```
Example:
n = int(input("Enter an integer: "))
if n % 2 == 0:
   print(n, "is even")
else:
   print(n, "is odd")
```

Press Ctrl+Enter to apply

Choose explanation style:

[✓ Brief] Detailed

These are the programming languages that this code explainer can explain(also has auto detect)



Enter your code below, specify its language, and get an explanation from a Large Language Model (Gemini 1.5 Flash).

Select Code Language:

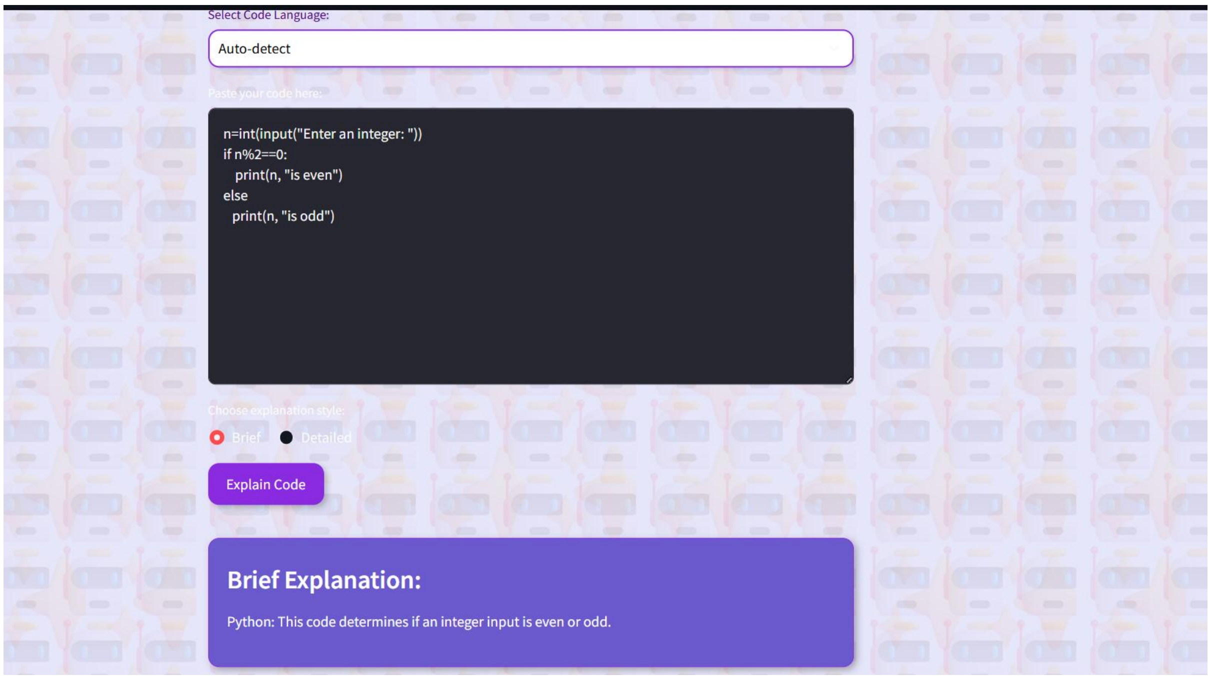
Auto-detect

Also HTML/CSS(included)

Python test:

Gives option to user to choose between Brief and Detailed explanation(according to their convenience)

1)Brief Explanation:


Select Code Language:

Auto-detect

Paste your code here:

```
n=int(input("Enter an integer: "))
if n%2==0:
   print(n, "is even")
else
  print(n, "is odd")
```

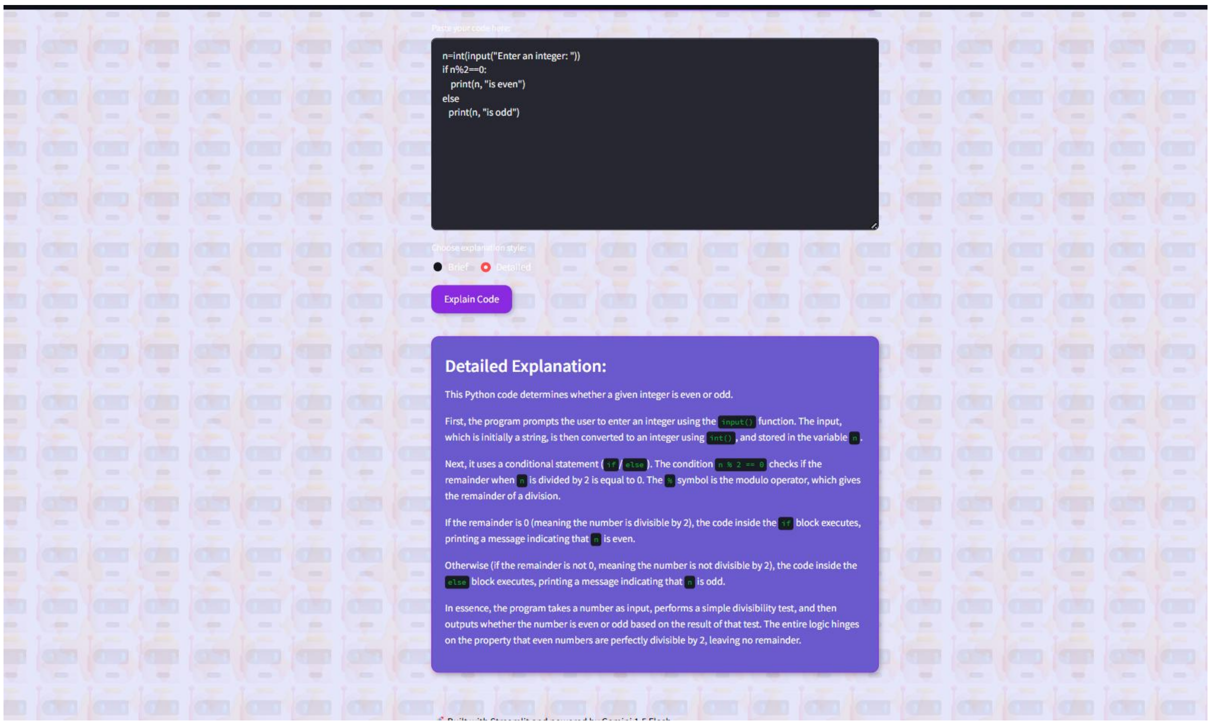
Choose explanation style:

[✓ Brief] Detailed

Brief Explanation:

Python: This code determines if an integer input is even or odd.

2)Detailed Explanation:


Paste your code here:

```
n=int(input("Enter an integer: "))
if n%2==0:
   print(n, "is even")
else
  print(n, "is odd")
```

Choose explanation style:

Brief [✓ Detailed]

Detailed Explanation:

This Python code determines whether a given integer is even or odd.

First, the program prompts the user to enter an integer using the `input()` function. The input, which is initially a string, is then converted to an integer using `int()`, and stored in the variable `n`.

Next, it uses a conditional statement (`if`/`else`). The condition `n % 2 == 0` checks if the remainder when `n` is divided by 2 is equal to 0. The `%` symbol is the modulo operator, which gives the remainder of a division.

If the remainder is 0 (meaning the number is divisible by 2), the code inside the `if` block executes, printing a message indicating that `n` is even.

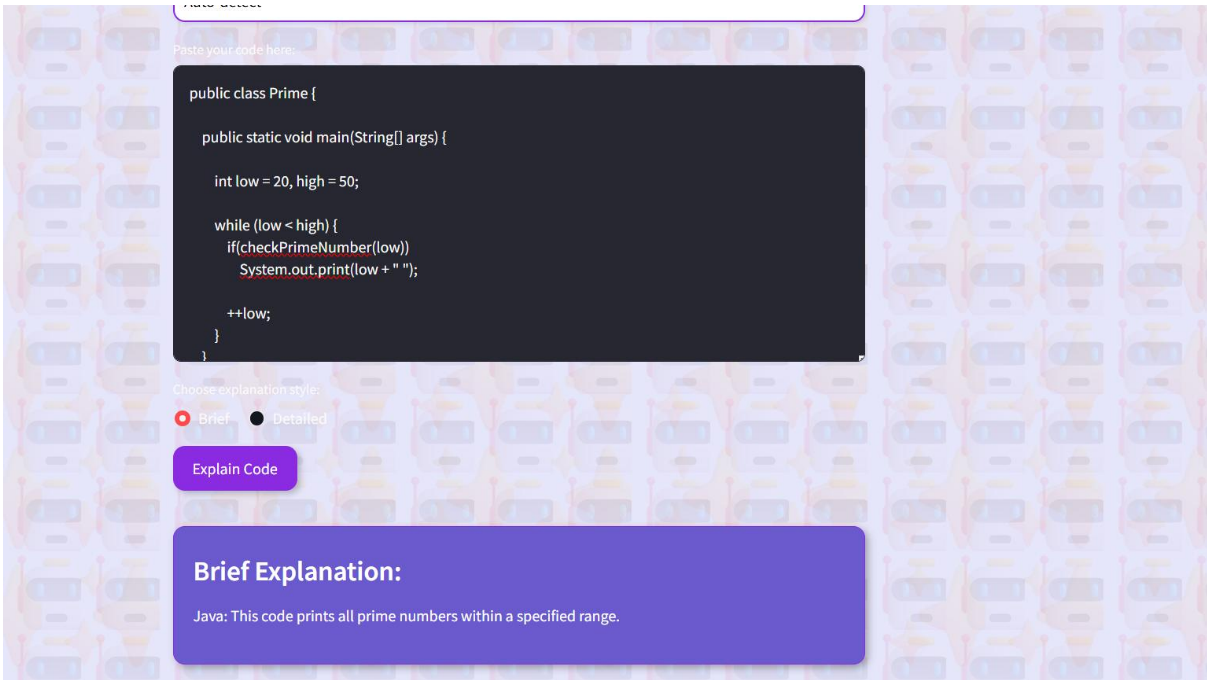
Otherwise (if the remainder is not 0, meaning the number is not divisible by 2), the code inside the `else` block executes, printing a message indicating that `n` is odd.

In essence, the program takes a number as input, performs a simple divisibility test, and then outputs whether the number is even or odd based on the result of that test. The entire logic hinges on the property that even numbers are perfectly divisible by 2, leaving no remainder.

Built with Streamlit and powered by Gemini 1.5 Flash

Java Test:

1)Brief Explanation:



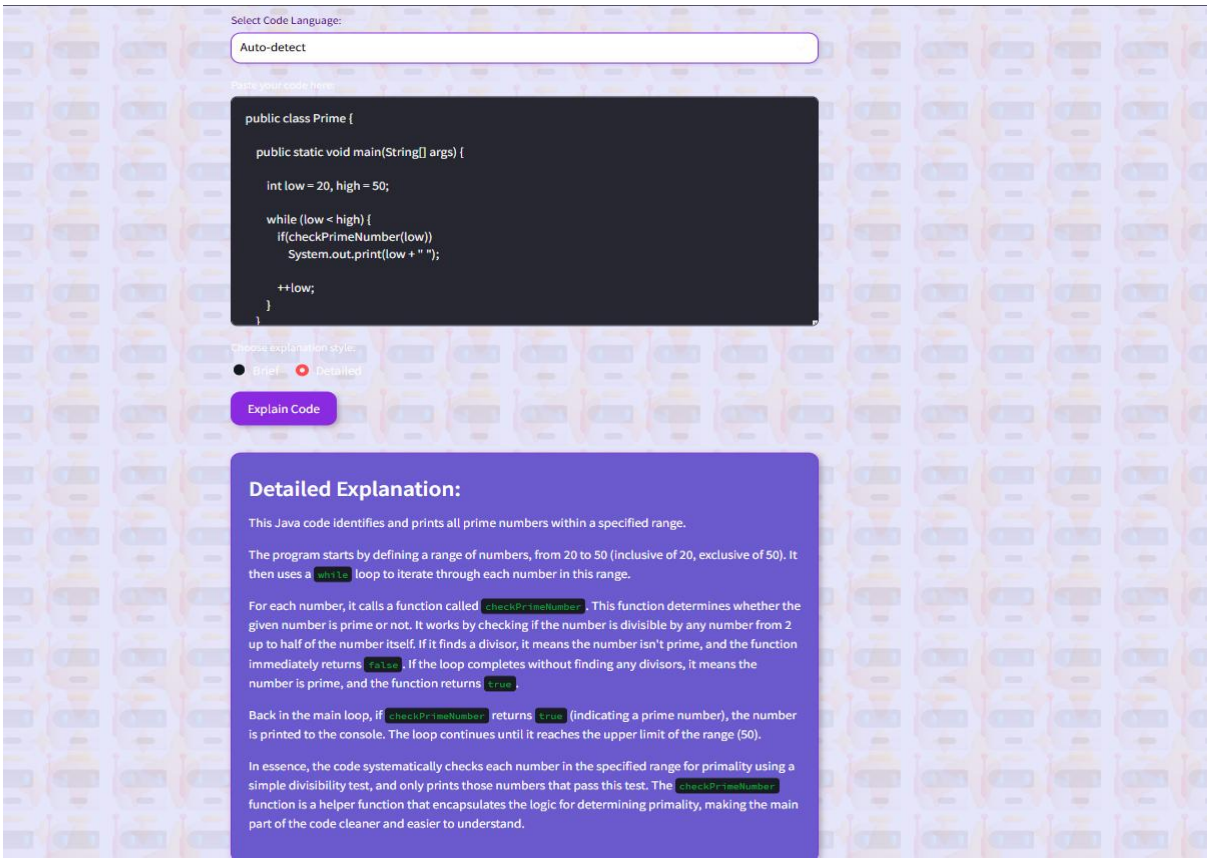
```
public class Prime {

  public static void main(String[] args) {

    int low = 20, high = 50;

    while (low < high) {
      if(checkPrimeNumber(low))
        System.out.print(low + " ");

      ++low;
    }
  }
```

Choose explanation style: [✓ Brief] Detailed

Brief Explanation:

Java: This code prints all prime numbers within a specified range.

2)Detailed Explanation:



Select Code Language: Auto-detect

```
public class Prime {

  public static void main(String[] args) {

    int low = 20, high = 50;

    while (low < high) {
      if(checkPrimeNumber(low))
        System.out.print(low + " ");

      ++low;
    }
  }
```

Choose explanation style: Brief [✓ Detailed]

Detailed Explanation:

This Java code identifies and prints all prime numbers within a specified range.

The program starts by defining a range of numbers, from 20 to 50 (inclusive of 20, exclusive of 50). It then uses a `while` loop to iterate through each number in this range.

For each number, it calls a function called `checkPrimeNumber`. This function determines whether the given number is prime or not. It works by checking if the number is divisible by any number from 2 up to half of the number itself. If it finds a divisor, it means the number isn't prime, and the function immediately returns `false`. If the loop completes without finding any divisors, it means the number is prime, and the function returns `true`.

Back in the main loop, if `checkPrimeNumber` returns `true` (indicating a prime number), the number is printed to the console. The loop continues until it reaches the upper limit of the range (50).

In essence, the code systematically checks each number in the specified range for primality using a simple divisibility test, and only prints those numbers that pass this test. The `checkPrimeNumber` function is a helper function that encapsulates the logic for determining primality, making the main part of the code cleaner and easier to understand.

Similarly all other languages are tested.