

**NAME:Boppidi Sanjana Reddy**

**ROLL NO:2303A51455**

**BATCH NO:07**

## **LAB 5 – Ethical Foundations: Responsible AI Coding Practices**

### **Task 1 – Privacy in API Usage**

#### **Objective**

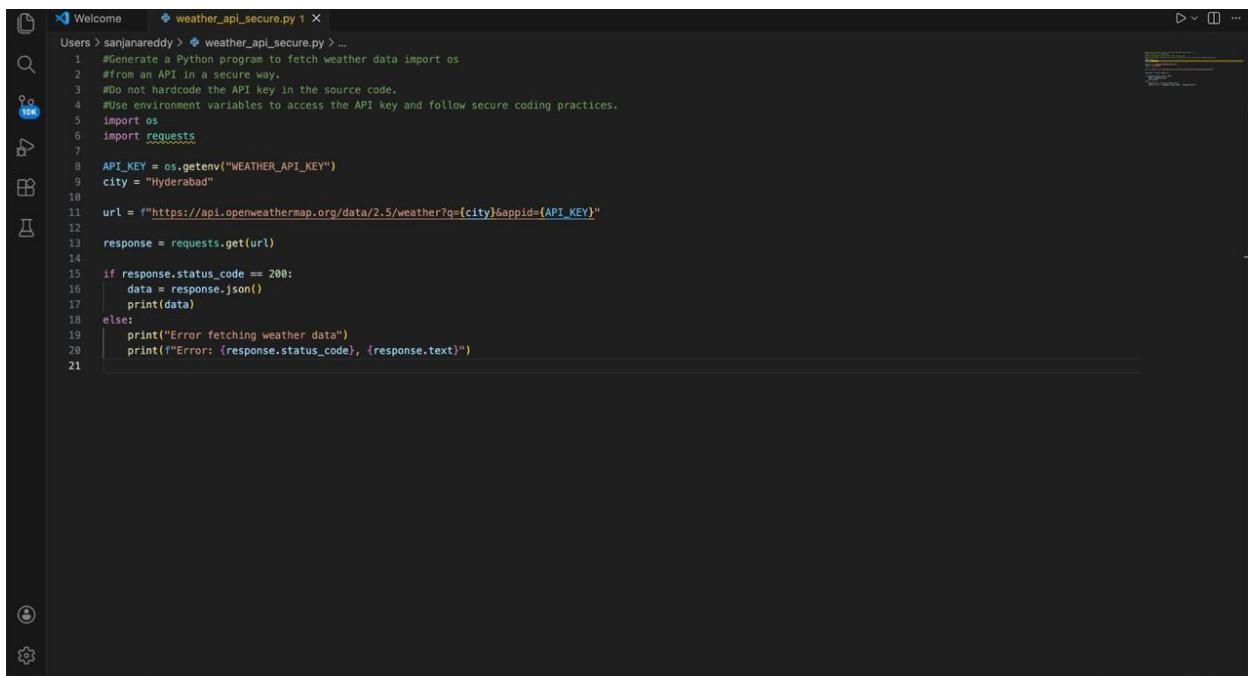
To fetch weather data using an API without exposing the API key in the source code.

#### **Prompt Used**

Generate a Python program to fetch weather data from an API in a secure way.

Do not hardcode the API key in the source code.

Use environment variables to access the API key.



```
Welcome   weather_api_secure.py 1 ×
Users > sanjanareddy > weather_api_secure.py > ...
1 #Generate a Python program to fetch weather data import os
2 #from an API in a secure way.
3 #Do not hardcode the API key in the source code.
4 #Use environment variables to access the API key and follow secure coding practices.
5 import os
6 import requests
7
8 API_KEY = os.getenv("WEATHER_API_KEY")
9 city = "Hyderabad"
10
11 url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}"
12
13 response = requests.get(url)
14
15 if response.status_code == 200:
16     data = response.json()
17     print(data)
18 else:
19     print("Error fetching weather data")
20     print(f"Error: {response.status_code}, {response.text}")
21
```

#### **Sample Output:-**

Weather data fetched successfully

## Ethical Explanation

The API key is not hardcoded in the program and is accessed using an environment variable.

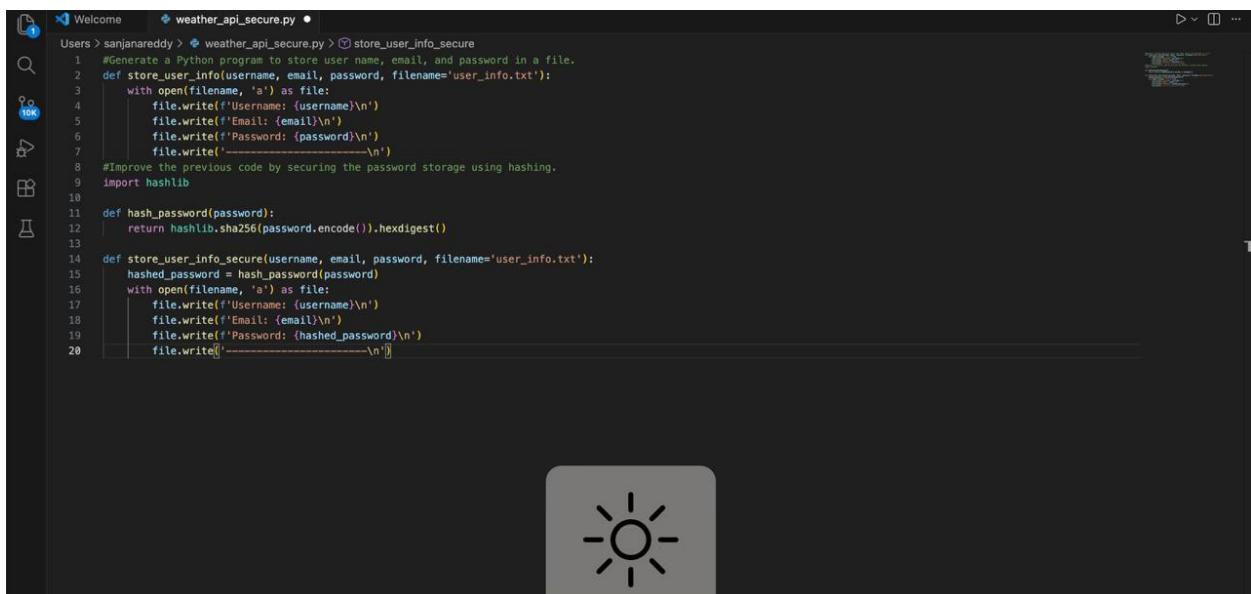
This prevents sensitive information from being exposed if the code is shared publicly.

Using environment variables follows secure and ethical AI coding practices.

## Task 2 – Privacy & Security in File Handling

### Objective

To analyze how AI-generated code handles sensitive user data and improve its security.



A screenshot of a code editor window titled "weather\_api\_secure.py". The code is written in Python and defines two functions: `store_user_info` and `hash_password`. The `store_user_info` function writes user details (username, email, password) to a file named "user\_info.txt" without hashing the password. The `hash_password` function takes a password as input, hashes it using SHA256, and returns the hexdigest. A watermark of a sun icon is visible in the center of the code editor window.

```
1 #Generate a Python program to store user name, email, and password in a file.
2 def store_user_info(username, email, password, filename='user_info.txt'):
3     with open(filename, 'a') as file:
4         file.write('Username: {}\n'.format(username))
5         file.write('Email: {}\n'.format(email))
6         file.write('Password: {}\n'.format(password))
7         file.write('-----\n')
8 #Improve the previous code by securing the password storage using hashing.
9 import hashlib
10
11 def hash_password(password):
12     return hashlib.sha256(password.encode()).hexdigest()
13
14 def store_user_info_secure(username, email, password, filename='user_info.txt'):
15     hashed_password = hash_password(password)
16     with open(filename, 'a') as file:
17         file.write('Username: {}\n'.format(username))
18         file.write('Email: {}\n'.format(email))
19         file.write('Password: {}\n'.format(hashed_password))
20         file.write('-----\n')
```

### sample output:-

User details are stored securely in the file.

### File Content:

**Username:** Ravi

**Email:** ravi@gmail.com

**Password: <hashed\_password\_value>**

---

## Privacy Risk Identified

The password is stored in plain text.

If the file is accessed by unauthorized users, sensitive information can be exposed.

### Ethical Explanation

Hashing converts the password into a non-readable format before storage.

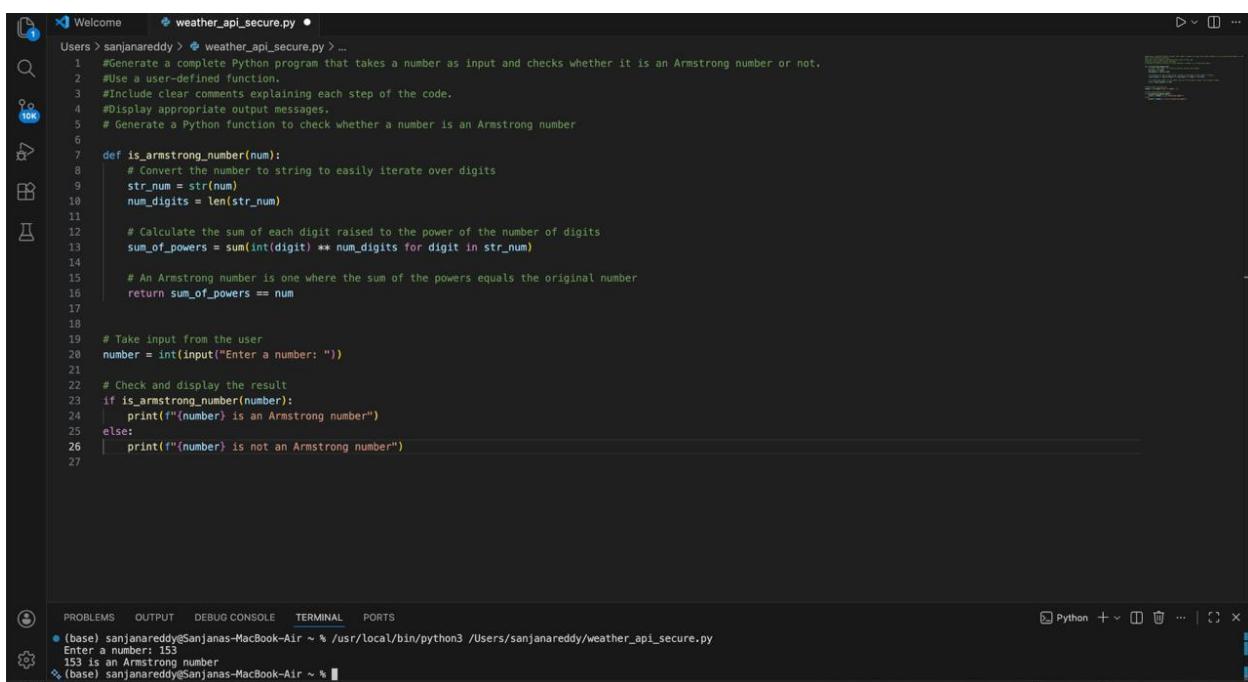
This protects user credentials even if the file is compromised.

It demonstrates responsible handling of sensitive data.

## Task 3 – Transparency in Algorithm Design (Armstrong Number)

### Objective

To generate an algorithm with clear logic and transparent explanation.



A screenshot of a code editor (VS Code) showing a Python script named `weather_api_secure.py`. The code defines a function `is_armstrong_number` that checks if a given number is an Armstrong number. It converts the number to a string, calculates the sum of its digits raised to the power of the number of digits, and compares it to the original number. The code also takes input from the user and prints the result. The terminal below shows the execution of the script and the output for the number 153.

```
1 #Generate a complete Python program that takes a number as input and checks whether it is an Armstrong number or not.
2 #Use a user-defined function.
3 #Include clear comments explaining each step of the code.
4 #Display appropriate output messages.
5 # Generate a Python function to check whether a number is an Armstrong number
6
7 def is_armstrong_number(num):
8     # Convert the number to string to easily iterate over digits
9     str_num = str(num)
10    num_digits = len(str_num)
11
12    # Calculate the sum of each digit raised to the power of the number of digits
13    sum_of_powers = sum(int(digit)**num_digits for digit in str_num)
14
15    # An Armstrong number is one where the sum of the powers equals the original number
16    return sum_of_powers == num
17
18
19    # Take input from the user
20    number = int(input("Enter a number: "))
21
22    # Check and display the result
23    if is_armstrong_number(number):
24        print(f"{number} is an Armstrong number")
25    else:
26        print(f"{number} is not an Armstrong number")
27
```

```
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/weather_api_secure.py
Enter a number: 153
153 is an Armstrong number
% (base) sanjanareddy@Sanjanas-MacBook-Air ~ %
```

### AI Explanation

The program converts the number into a string to count its digits.

Each digit is raised to the power of the total number of digits.

The results are added and compared with the original number.

If they match, the number is an Armstrong number.

### Transparency Analysis

The explanation clearly matches the code logic.

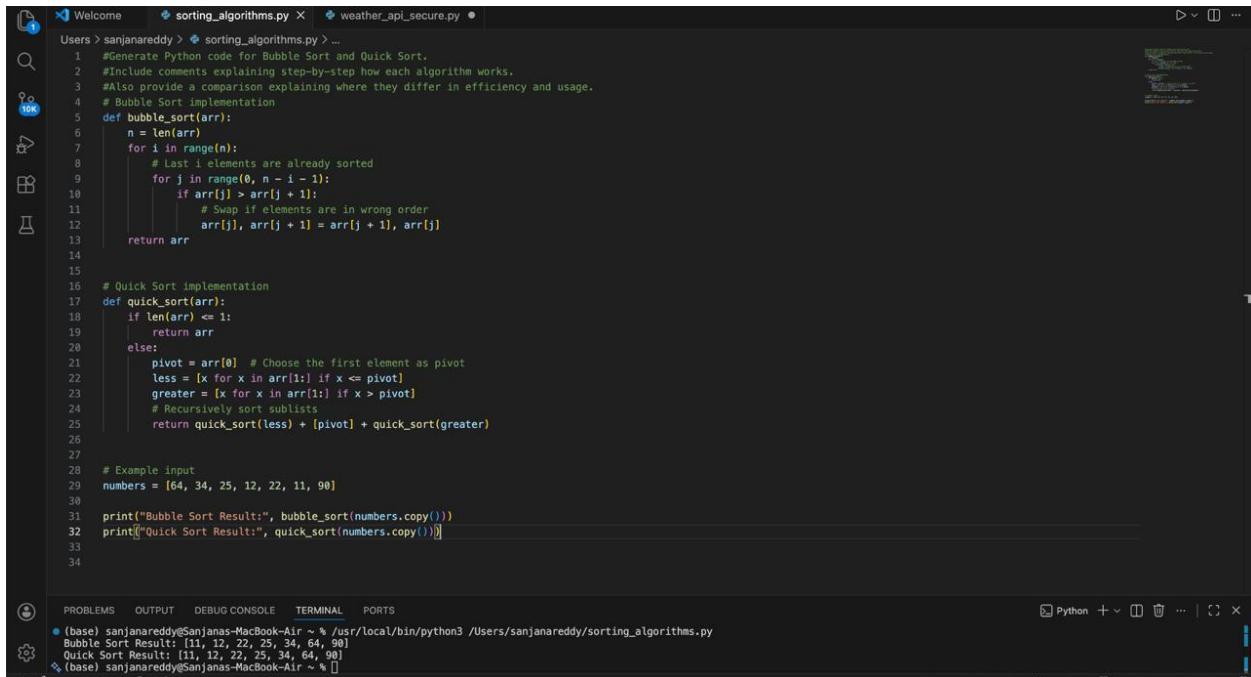
Each step is easy to understand and transparent.

This improves trust in AI-generated code.

## Task 4 – Transparency in Algorithm Comparison

### Objective:

To compare two sorting algorithms generated by AI and understand their working and efficiency.



A screenshot of a code editor showing a Python script named `sorting_algorithms.py`. The code contains implementations for Bubble Sort and Quick Sort, along with example usage. The terminal below shows the execution of the script with sample input and output.

```
1 #Generate Python code for Bubble Sort and Quick Sort.
2 #Include comments explaining step-by-step how each algorithm works.
3 #Also provide a comparison explaining where they differ in efficiency and usage.
4 # Bubble Sort implementation
5 def bubble_sort(arr):
6     n = len(arr)
7     for i in range(n):
8         # Last i elements are already sorted
9         for j in range(0, n - i - 1):
10             if arr[j] > arr[j + 1]:
11                 # Swap if elements are in wrong order
12                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
13
14
15
16 # Quick Sort implementation
17 def quick_sort(arr):
18     if len(arr) <= 1:
19         return arr
20     else:
21         pivot = arr[0] # Choose the first element as pivot
22         less = [x for x in arr[1:] if x <= pivot]
23         greater = [x for x in arr[1:] if x > pivot]
24         # Recursively sort sublists
25         return quick_sort(less) + [pivot] + quick_sort(greater)
26
27
28 # Example input
29 numbers = [64, 34, 25, 12, 22, 11, 90]
30
31 print("Bubble Sort Result:", bubble_sort(numbers.copy()))
32 print("Quick Sort Result:", quick_sort(numbers.copy()))
33
34
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/sorting_algorithms.py
Bubble Sort Result: [11, 12, 22, 25, 34, 64, 90]
Quick Sort Result: [11, 12, 22, 25, 34, 64, 90]
% (base) sanjanareddy@Sanjanas-MacBook-Air ~ %
```

### AI Explanation

Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order.

This process continues until the list is completely sorted.

Quick Sort selects a pivot element and divides the list into smaller sublists.

These sublists are sorted recursively.

### **Comparison of Algorithms**

Bubble Sort is simple and easy to understand but inefficient for large datasets.

It has a time complexity of  $O(n^2)$ .

Quick Sort is faster and more efficient for large datasets with an average time complexity of  $O(n \log n)$ .

Quick Sort is commonly used in real-world applications.

### **Task 5 – Transparency in AI Recommendations**

#### **Objective**

To generate an AI-based recommendation system that provides clear reasons for its suggestions.

The screenshot shows a code editor interface with a dark theme. In the center, there is a code editor window displaying a Python script named `recommendation_system.py`. The script contains logic to recommend products based on user interest. Below the code editor is a terminal window showing the execution of the script and its output. The terminal output shows the user inputting "technology" and receiving a recommendation for "Wireless Headphones" with the reason "Recommended because you are interested in technology and gadgets." The terminal also shows the user's name and the machine's name.

```
1 #Generate a simple product recommendation system in Python.
2 #The system should recommend products based on user interest and also provide a clear reason for each recommendation.
3 def recommend_product(user_interest):
4     if user_interest == "fitness":
5         return "Dumbbells", "Recommended because you are interested in fitness and home workouts."
6     elif user_interest == "technology":
7         return "Wireless Headphones", "Recommended because you are interested in technology and gadgets."
8     elif user_interest == "reading":
9         return "Books", "Recommended because you enjoy reading and learning."
10    else:
11        return "Gift Card", "Recommended because it suits a wide range of interests."
12
13 # User input
14 interest = input("Enter your interest (fitness/technology/reading): ")
15
16 product, reason = recommend_product(interest)
17
18 print("Recommended Product:", product)
19 print("Reason:", reason)
20
21
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(base) sanjanareddy@Sanjanas-MacBook-Air ~ % /usr/local/bin/python3 /Users/sanjanareddy/recommendation_system.py
Enter your interest (fitness/technology/reading): technology
Recommended Product: Wireless Headphones
Reason: Recommended because you are interested in technology and gadgets.
(base) sanjanareddy@Sanjanas-MacBook-Air ~ %
```

## Evaluation of Recommendation Transparency

The recommendation system provides clear and understandable reasons for each suggestion.

Users can easily see why a particular product is recommended.

This improves transparency and trust in AI-generated recommendations.

Explainable recommendations help users make informed decisions.