Name:B.Nithin H.NO:2303A51803 BATCH:26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Dr. Rishabh Mittal		
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju		
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week10 – Friday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number:20.5(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 20 – Security Testing: Identifying Vulnerabilities in AI-Generated Code**	Week10 - Friday

Lab Objectives:

- Understand how to test AI-generated code for common security vulnerabilities.
- Learn to apply secure coding principles while analyzing AI outputs.
- Practice detecting risks such as SQL injection, XSS, hardcoded credentials, and weak encryption.
- Enhance code reliability and safety by using AI for secure refactoring.

Task 1 – Password Storage Vulnerability Check

Task:

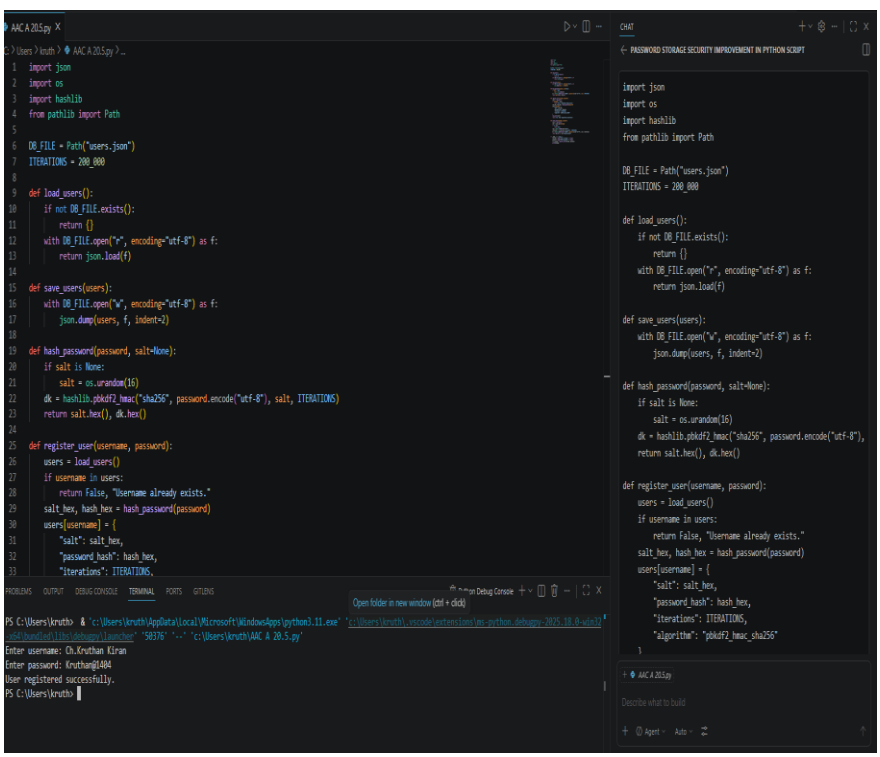
Analyze an AI-generated signup script for insecure password storage.

Instructions:

- Prompt AI to generate a Python user registration program.
- Check whether passwords are stored in plain text.
- Improve the script using hashing (bcrypt or hashlib).

Expected Output:

- A secure registration script with hashed password storage.



```python
def hash_password(password, salt=None):
    return salt.hex(), dk.hex()

def register_user(username, password):
    users = load_users()
    if username in users:
        return False, "Username already exists."
    salt_hex, hash_hex = hash_password(password)
    users[username] = {
        "salt": salt_hex,
        "password_hash": hash_hex,
        "iterations": ITERATIONS,
        "algorithm": "pbkdf2_hmac_sha256"
    }
    save_users(users)
    return True, "User registered successfully."

def verify_user(username, password):
    users = load_users()
    user = users.get(username)
    if not user:
        return False
    salt = bytes.fromhex(user["salt"])
    iterations = int(user.get("iterations", ITERATIONS))
    dk = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"), salt, iterations)
    return dk.hex() == user["password_hash"]

if __name__ == "__main__":
    username = input("Enter username: ").strip()
    password = input("Enter password: ").strip()
    ok, message = register_user(username, password)
    print(message)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

Python Debug Console

```
PS C:\Users\kruth>  & "c:\Users\kruth\AppData\Local\Microsoft\WindowsApps\python3.11.exe" "c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32
-x64\bundled\libs\debugpy\launcher" "50376" "--" "c:\Users\kruth\AAC A 20.5.py"
Enter username: Ch.Kruthan Kiran
Enter password: Kruthan@1404
User registered successfully.
```

```python
import json
import os
import hashlib
from pathlib import Path

DB_FILE = Path("users.json")
ITERATIONS = 200_000

def load_users():
    if not DB_FILE.exists():
        return {}
    with DB_FILE.open("r", encoding="utf-8") as f:
        return json.load(f)

def save_users(users):
    with DB_FILE.open("w", encoding="utf-8") as f:
        json.dump(users, f, indent=2)

def hash_password(password, salt=None):
    if salt is None:
        salt = os.urandom(16)
    dk = hashlib.pbkdf2_hmac("sha256", password.encode("ut
    return salt.hex(), dk.hex()

def register_user(username, password):
    users = load_users()
    if username in users:
        return False, "Username already exists."
    salt_hex, hash_hex = hash_password(password)
    users[username] = {
        "salt": salt_hex,
        "password_hash": hash_hex,
        "iterations": ITERATIONS,
        "algorithm": "pbkdf2_hmac_sha256"
    }
```

AAC A 20.5.py

Task 2 – Hardcoded Secrets Detection

Task:

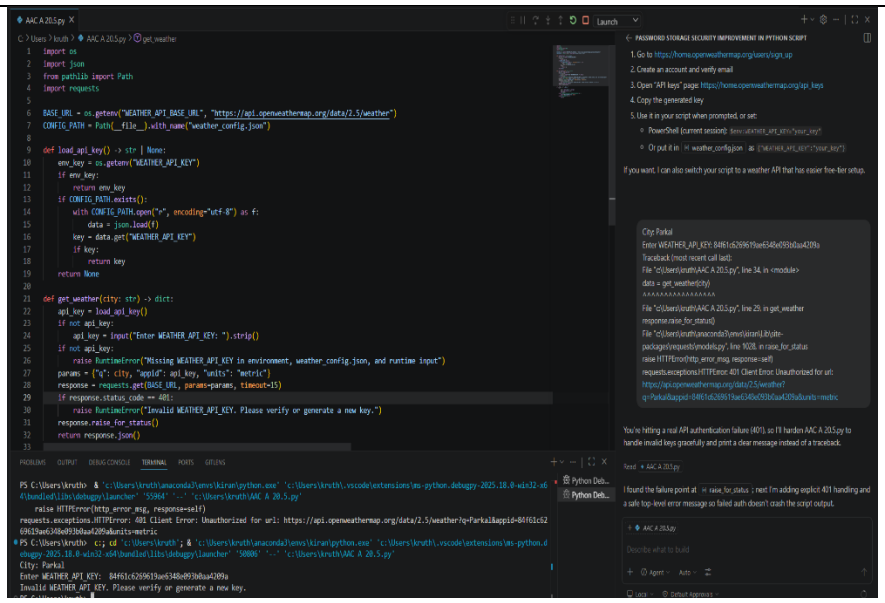
Evaluate AI-generated code for hardcoded API keys or credentials.

Instructions:

- Ask AI to generate a script that connects to an external API.
- Identify if API keys/passwords are written directly in code.
- Refactor using environment variables or config files.

Expected Output:

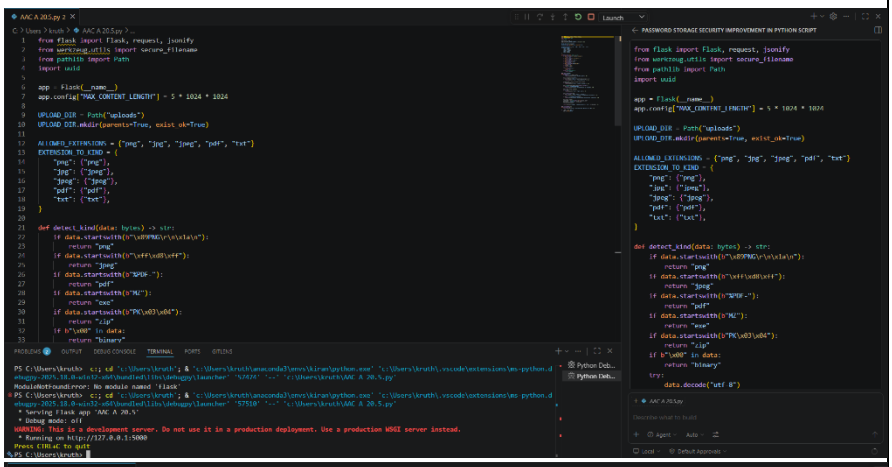
- A secure script with secrets managed properly.

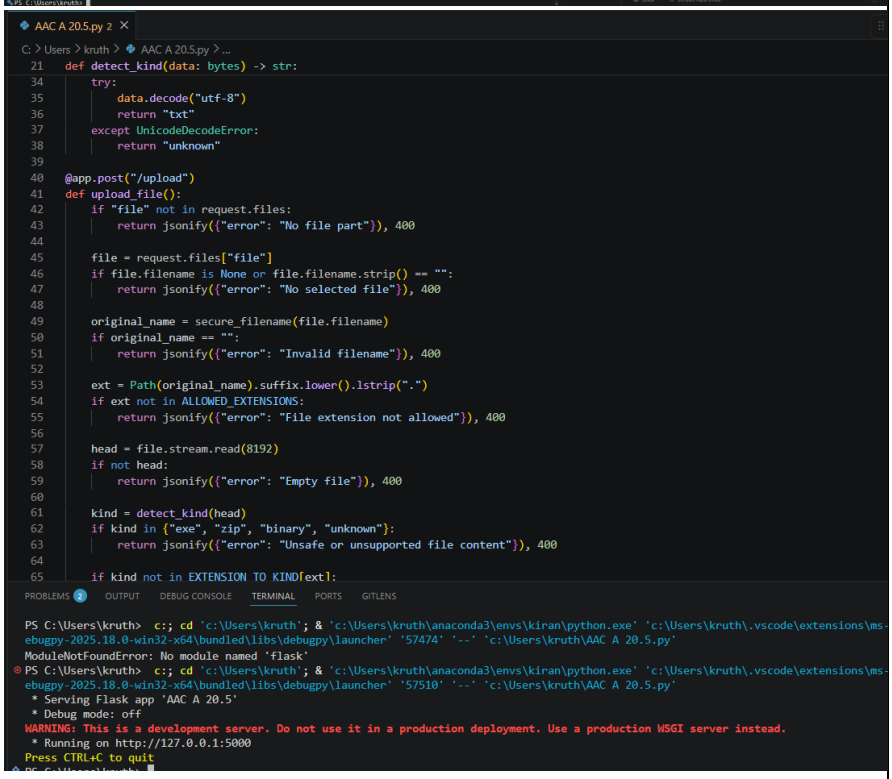

AAC A 20.5.py
C: > Users > kruth > AAC A 20.5.py > get_weather
import os
import json
from pathlib import Path
import requests
BASE_URL = os.getenv("WEATHER_API_BASE_URL", "https://api.openweathermap.org/data/2.5/weather")
CONFIG_PATH = Path(__file__).with_name("weather_config.json")
def load_api_key() -> str | None:
env_key = os.getenv("WEATHER_API_KEY")
if env_key:
return env_key
if CONFIG_PATH.exists():
with CONFIG_PATH.open("r", encoding="utf-8") as f:
data = json.load(f)
key = data.get("WEATHER_API_KEY")
if key:
return key
return None
def get_weather(city: str) -> dict:
api_key = load_api_key()
if not api_key:
api_key = input("Enter WEATHER_API_KEY: ").strip()
if not api_key:
raise RuntimeError("Missing WEATHER_API_KEY in environment, weather_config.json, and runtime input")
params = {"q": city, "appid": api_key, "units": "metric"}
response = requests.get(BASE_URL, params=params, timeout=15)
if response.status_code == 401:
raise RuntimeError("Invalid WEATHER_API_KEY. Please verify or generate a new key.")
response.raise_for_status()
return response.json()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
PS C:\Users\kruth> & 'c:\Users\kruth\anaconda3\envs\kiran\python.exe' 'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55964' '--' 'c:\Users\kruth\AAC A 20.5.py'
raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 401 Client Error: Unauthorized for url: https://api.openweathermap.org/data/2.5/weather?q=Parkal&appid=84f61c6269619ae6348e093b0aa4209a&units=metric
PS C:\Users\kruth> c:; cd 'c:\Users\kruth'; & 'c:\Users\kruth\anaconda3\envs\kiran\python.exe' 'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50806' '--' 'c:\Users\kruth\AAC A 20.5.py'
City: Parkal
Enter WEATHER_API_KEY: 84f61c6269619ae6348e093b0aa4209a
Invalid WEATHER_API_KEY. Please verify or generate a new key.
PASSWORD STORAGE SECURITY IMPROVEMENT IN PYTHON SCRIPT
1. Go to https://home.openweathermap.org/users/sign_up
2. Create an account and verify email
3. Open "API keys" page: https://home.openweathermap.org/api_keys
4. Copy the generated key
5. Use it in your script when prompted, or set:
PowerShell (current session): $env:WEATHER_API_KEY="your_key"
Or put it in weather_config.json as {"WEATHER_API_KEY":"your_key"}
If you want, I can also switch your script to a weather API that has easier free-tier setup.
City: Parkal
Enter WEATHER_API_KEY: 84f61c6269619ae6348e093b0aa4209a
Traceback (most recent call last):
File "c:\Users\kruth\AAC A 20.5.py", line 34, in <module>
data = get_weather(city)
^^^

- Check for vulnerabilities like uploading executable files.
- Add validation for file type, size, and secure filename handling.

Expected Output:

- A secure file upload implementation.





AAC A 20.5.py 2

C: > Users > kruth > AAC A 20.5.py > ...

```python
def upload_file():
    if original_name == "":
        return jsonify({"error": "Invalid filename"}), 400

    ext = Path(original_name).suffix.lower().lstrip(".")
    if ext not in ALLOWED_EXTENSIONS:
        return jsonify({"error": "File extension not allowed"}), 400

    head = file.stream.read(8192)
    if not head:
        return jsonify({"error": "Empty file"}), 400

    kind = detect_kind(head)
    if kind in {"exe", "zip", "binary", "unknown"}:
        return jsonify({"error": "Unsafe or unsupported file content"}), 400

    if kind not in EXTENSION_TO_KIND[ext]:
        return jsonify({"error": "File content does not match extension"}), 400

    file.stream.seek(0)
    stored_name = f"{uuid.uuid4().hex}_{original_name}"
    destination = UPLOAD_DIR / stored_name
    file.save(destination)

    return jsonify({"message": "Upload successful", "file": stored_name}), 201

@app.errorhandler(413)
def file_too_large(_):
    return jsonify({"error": "File too large. Max size is 5 MB"}), 413

if __name__ == "__main__":
    app.run(debug=False)
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Users\kruth>  c:; cd 'c:\Users\kruth'; & 'c:\Users\kruth\anaconda3\envs\kiran\python.exe' 'c:\Users\kruth\.vscode\ex
ebugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57474' '--' 'c:\Users\kruth\AAC A 20.5.py'
ModuleNotFoundError: No module named 'flask'
PS C:\Users\kruth>  c:; cd 'c:\Users\kruth'; & 'c:\Users\kruth\anaconda3\envs\kiran\python.exe' 'c:\Users\kruth\.vscode\ex
ebugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57510' '--' 'c:\Users\kruth\AAC A 20.5.py'
 * Serving Flask app 'AAC A 20.5'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
PS C:\Users\kruth>
```

Task 4 – Command Injection Prevention

Task:

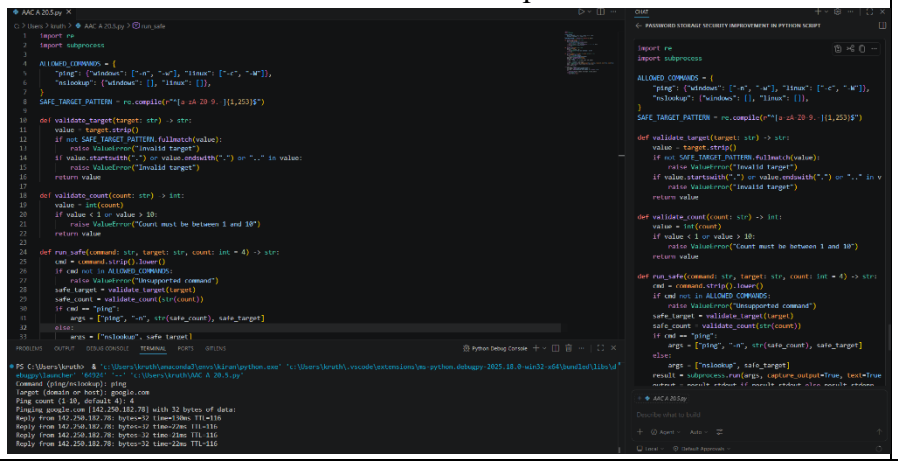
Analyze an AI-generated Python script that executes system commands.

Instructions:

- Ask AI to generate code using os.system() or subprocess.
- Identify command injection risks with user input.
- Fix using safe subprocess calls and input validation.

Expected Output:

- A secure command execution script.



AAC A 20.5.py

C: > Users > kruth > AAC A 20.5.py > run_safe

```python
def run_safe(command: str, target: str, count: int = 4) -> str:
    cmd = command.strip().lower()
    if cmd not in ALLOWED_COMMANDS:
        raise ValueError("Unsupported command")
    safe_target = validate_target(target)
    safe_count = validate_count(str(count))
    if cmd == "ping":
        args = ["ping", "-n", str(safe_count), safe_target]
    else:
        args = ["nslookup", safe_target]
    result = subprocess.run(args, capture_output=True, text=True, timeout=15, shell=False, check=False)
    output = result.stdout if result.stdout else result.stderr
    return output.strip()

if __name__ == "__main__":
    user_command = input("Command (ping/nslookup): ")
    user_target = input("Target (domain or host): ")
    user_count = input("Ping count (1-10, default 4): ").strip() or "4"
    try:
        print(run_safe(user_command, user_target, int(user_count)))
    except Exception as e:
        print(f"Error: {e}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Users\kruth>  & 'c:\Users\kruth\anaconda3\envs\kiran\python.exe' 'c:\Users\kruth\.vscode\extensions\ms-python.
ebugpy\launcher' '64924' '--' 'c:\Users\kruth\AAC A 20.5.py'
Reply from 142.250.182.78: bytes=32 time=22ms TTL=116
Reply from 142.250.182.78: bytes=32 time=21ms TTL=116
Reply from 142.250.182.78: bytes=32 time=22ms TTL=116

Ping statistics for 142.250.182.78:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 21ms, Maximum = 130ms, Average = 48ms
PS C:\Users\kruth>
```

Task 5 – Secure Input Handling in Web Forms

Task:

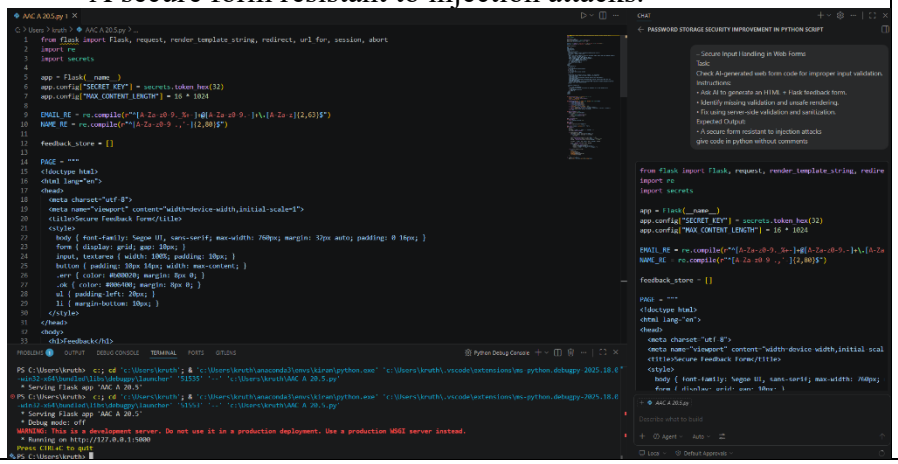
Check AI-generated web form code for improper input validation.

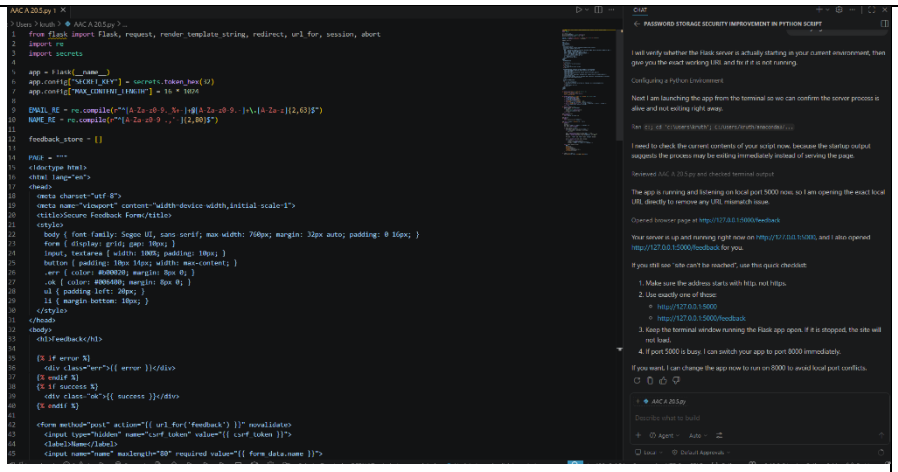
Instructions:

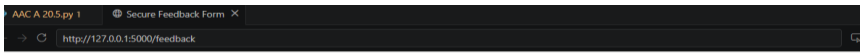
- Ask AI to generate an HTML + Flask feedback form.
- Identify missing validation and unsafe rendering.
- Fix using server-side validation and sanitization.

Expected Output:

- A secure form resistant to injection attacks.







Feedback

Name

Kruthan Kiran

Email

kruthankiran@gmail.com

Message

Hello There its Kruthan (2303A51404)

Submit

Recent Feedback

- No feedback yet.

Feedback

Feedback submitted successfully.

Name

Email

Message

Submit

Recent Feedback

- **Kruthan Kiran** (kruthankiran@gmail.com): Hello There its Kruthan (2303A51404)

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.