Problem 2: Validate Solidity Code

Question: Develop an AI model that generates Solidity contracts from text descriptions.

Technologies Used: Python, Flask, Groq, Solc(Solidity Compiler), Requests

Installation Steps: pip install -r requirements.txt

Explanation Of Code:

1) **Generate Solidity Contract:**

   Endpoint: /generate-contract
   Methods: POST, GET
   Description: Generates a Solidity smart contract based on a text prompt and gives the generated code as string

2) **Validate Solidity Contract:**

   Endpoint: /validate-contract
   Methods: POST, GET
   Description: Validates syntax of solidity contract

3) **List Available Contract Template:**

   Endpoint: /list-templates
   Methods: GET
   Description: It gives list of predefined contract templates

Execution:

Run “python app.py” in terminal and it will runs flask server at port number 5000

127.0.0.1:5000

Pretty-print


```
{
  "endpoints": [
    {
      "description": "Generate a Solidity contract from text description",
      "methods": [
        "GET",
        "POST"
      ],
      "parameters": [
        "prompt"
      ],
      "path": "/generate-contract"
    },
    {
      "description": "Validate Solidity contract syntax",
      "methods": [
        "GET",
        "POST"
      ],
      "parameters": [
        "solidity_code"
      ],
      "path": "/validate-contract"
    },
    {
      "description": "List available contract templates",
      "methods": [
        "GET"
      ],
      "path": "/list-templates"
    },
    {
      "description": "Get a specific contract template",
      "methods": [
        "GET"
      ],
      "parameters": [
        "id"
      ],
      "path": "/get-template"
    }
  ],
  "service": "Solidity Contract Generator API"
}
```


2) now give “127.0.0.1:5000/generate-contract?prompt=Create a simple voting contract” => to give custom prompt

127.0.0.1:5000/generate-contract?prompt=Create%20a%20simple%20voting%20contract

Pretty-print


```
{
  "is_valid": true,
  "prompt": "Create a simple voting contract",
  "solidity_code": "```solidity\n// SPDX-License-Identifier: MIT\npragma solidity ^0.8.0;\n\nimport \"@openzeppelin/c
\"@openzeppelin/contracts/security/ReentrancyGuard.sol\";\n\ncontract SimpleVoting is Ownable, ReentrancyGuard {\n
public candidates;\n    mapping(address => bool) public voters;\n\n    event NewCandidate(uint256 indexed candidateId
_name) external onlyOwner {\n        uint256 candidateId = candidates.length;\n        candidates.push(Candidate(_nam
_candidateId) external {\n        require(!voters[msg.sender], \"You have already voted.\");\n        require(_candid
candidates[_candidateId].voteCount++;\n        voters[msg.sender] = true;\n        emit Voted(_candidateId);\n    }\n
  "validation_message": "Basic validation passed (solc compiler not available for thorough validation)"
}
```


3) Now give “127.0.0.1:5000/get-template?id=0” => to display prompt and output

127.0.0.1:5000/get-template?id=0

Pretty-print


```
{
  "output": "```solidity\n// SPDX-License-Identifier: MIT\npragma solidity ^0.8.0;\n\nimport \"@openzeppelin/contracts/toke
\"@openzeppelin/contracts/utils/math/SafeMath.sol\";\n\ncontract BurnableERC20 is ERC20 {\n    using SafeMath for uint256;\
_mint(msg.sender, initialSupply);\n    }\n\n    function burn(uint256 amount) public {\n        require(amount > 0, \"Amoun
\"Insufficient balance\");\n\n        _burn(msg.sender, amount);\n    }\n}\n```",
  "prompt": "Generate an ERC-20 token contract with burn functionality"
}
```