|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **ProgramName:**B. Tech | | | | **Assignment Type: Lab** | | | **AcademicYear:**2025-2026 | | |
| **CourseCoordinatorName** | | | | Venkataramana Veeramsetty | | | | | |
| **Instructor(s)Name** | | | | |  | | --- | | Dr. V. Venkataramana (Co-ordinator) | | Dr. T. Sampath Kumar | | Dr. Pramoda Patro | | Dr. Brij Kishor Tiwari | | Dr.J.Ravichander | | Dr. Mohammand Ali Shaik | | Dr. Anirodh Kumar | | Mr. S.Naresh Kumar | | Dr. RAJESH VELPULA | | Mr. Kundhan Kumar | | Ms. Ch.Rajitha | | Mr. M Prakash | | Mr. B.Raju | | Intern 1 (Dharma teja) | | Intern 2 (Sai Prasad) | | Intern 3 (Sowmya) | | NS\_2 ( Mounika) | | | | | | |
| **CourseCode** | | | 24CS002PC215 | **CourseTitle** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week3 - Tuesday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicableto**  **Batches** | |  | | | |
| **AssignmentNumber:5.2**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***ExpectedTime***  ***to complete*** |  |
|  | 1 | Lab 5: Ethical Foundations – Responsible AI Coding Practices  **Lab Objectives:**   * To explore the ethical risks associated with AI-generated code. * To recognize issues related to security, bias, transparency, and copyright. * To reflect on the responsibilities of developers when using AI tools in software development. * To promote awareness of best practices for responsible and ethical AI coding.   **Lab Outcomes (LOs):**  After completing this lab, students will be able to:   * Identify and avoid insecure coding patterns generated by AI tools. * Detect and analyze potential bias or discriminatory logic in AI-generated outputs. * Evaluate originality and licensing concerns in reused AI-generated code. * Understand the importance of explainability and transparency in AI-assisted programming. * Reflect on accountability and the human role in ethical AI coding practices..   **Task Description#1 (Privacy and Data Security)**   * Use an AI tool (e.g., Copilot, Gemini, Cursor) to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption.   **Expected Output#1**   * Identification of insecure logic; revised secure version with proper password hashing and environment variable use.   Building a secure login system is crucial! I can help you create a Python login system, identify potential security issues like hardcoded passwords or lack of encryption, and then provide a corrected, more secure version using password hashing and environment variables.  Here is a plan to achieve this:   * **Generate initial login system code**: Write a Python code snippet that simulates a basic login system with user input for username and password. * **Identify security vulnerabilities**: Analyze the generated code for common security flaws such as hardcoded passwords, plain-text storage, or lack of encryption. * **Revise code with secure practices**: Modify the code to incorporate secure practices, including password hashing using a library like bcrypt or argon2, and using environment variables to store sensitive information like secret keys. * **Explain the security improvements**: Provide a clear explanation of the vulnerabilities identified in the initial code and how the revised code addresses them. * **Finish task**: Present the final secure code and the explanation of the security improvements.   def simple\_login():      """Simulates a basic login system with hardcoded credentials."""      correct\_username = "admin"      correct\_password = "password123"      entered\_username = input("Enter username: ")      entered\_password = input("Enter password: ")    if entered\_username == correct\_username and entered\_password == correct\_password:          print("Login successful!")      else:          print("Login failed. Invalid username or password.")  # Run the simulation  simple\_login()  import bcrypt  import os  def secure\_login(entered\_username, entered\_password):      """      Securely checks if the entered username and password match the stored credentials      retrieved from environment variables, using bcrypt for password hashing.      """      # Retrieve stored credentials from environment variables      stored\_username = os.environ.get('STORED\_USERNAME')      stored\_hashed\_password = os.environ.get('STORED\_HASHED\_PASSWORD')      # Check if environment variables are set      if not stored\_username or not stored\_hashed\_password:          print("Error: Stored username or hashed password not found in environment variables.")          return False      # Compare usernames      if entered\_username != stored\_username:          return False      # Compare passwords using bcrypt      # bcrypt.checkpw expects bytes, so encode the entered password and stored hash      try:          entered\_password\_bytes = entered\_password.encode('utf-8')          stored\_hashed\_password\_bytes = stored\_hashed\_password.encode('utf-8')          if bcrypt.checkpw(entered\_password\_bytes, stored\_hashed\_password\_bytes):              return True          else:              return False      except ValueError:          # Handle cases where the stored hash might be invalid          print("Error: Invalid stored hashed password format.")          return False  %pip install bcrypt  Please set the STORED\_USERNAME and STORED\_HASHED\_PASSWORD environment variables before running. For example, you can use os.environ['STORED\_USERNAME'] = 'your\_username' and os.environ['STORED\_HASHED\_PASSWORD'] = 'your\_hashed\_password' Enter username: admin Enter password: password123 Error: Stored username or hashed password not found in environment variables. Please set the STORED\_USERNAME and STORED\_HASHED\_PASSWORD environment variables. Secure login failed. Invalid username or password.  **Task Description#2 (Bias)**   * Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the AI-generated logic exhibits bias or differing criteria based on names or genders.   **Expected Output#2**   * Screenshot or code comparison showing bias (if any); write 3–4 sentences on mitigation techniques.   PROMPT: "I am writing a paper about algorithmic bias. Could you generate a loan approval scenario for two different people: 'John' and 'Priya'? For each person, provide a brief summary of their financial profile (e.g., credit score, income, debt) and then give a conclusion on whether their loan would be approved. I need you to evaluate whether the AI-generated logic exhibits bias or differing criteria based on their names. Please ensure the criteria you use are objective and do not favor one name over the other."  def loan\_approval\_logic(credit\_score, annual\_income, loan\_amount, debt\_to\_income\_ratio):      """      Simulates a simplified loan approval logic based on objective criteria.      Args:          credit\_score (int): The applicant's credit score.          annual\_income (float): The applicant's annual income.          loan\_amount (float): The requested loan amount.          debt\_to\_income\_ratio (float): The applicant's debt-to-income ratio.      Returns:          str: The loan approval decision ('Approved' or 'Declined').      """      # Define objective criteria for loan approval      min\_credit\_score = 650      min\_income\_multiplier = 3 # Loan amount should not exceed 3 times annual income      max\_debt\_to\_income\_ratio = 0.4 # 40%      # Evaluate criteria      if credit\_score >= min\_credit\_score and \         loan\_amount <= annual\_income \* min\_income\_multiplier and \         debt\_to\_income\_ratio <= max\_debt\_to\_income\_ratio:          return "Approved"      else:          return "Declined"  # Now let's create profiles for John and Priya and apply this logic.  # Create profiles for John and Priya with identical financial data  john\_profile = {      "name": "John",      "credit\_score": 700,      "annual\_income": 60000,      "loan\_amount": 150000,      "debt\_to\_income\_ratio": 0.35  }  priya\_profile = {      "name": "Priya",      "credit\_score": 700,      "annual\_income": 60000,      "loan\_amount": 150000,      "debt\_to\_income\_ratio": 0.35  }  # You can add more profiles here with the same financial data but different names  # to further test for name-based bias.  print("Profiles created for John and Priya.")  print("John's Profile:", john\_profile)  print("Priya's Profile:", priya\_profile)  OUTPUT:Profiles created for John and Priya. John's Profile: {'name': 'John', 'credit\_score': 700, 'annual\_income': 60000, 'loan\_amount': 150000, 'debt\_to\_income\_ratio': 0.35} Priya's Profile: {'name': 'Priya', 'credit\_score': 700, 'annual\_income': 60000, 'loan\_amount': 150000, 'debt\_to\_income\_ratio': 0.35}  # Apply the loan approval logic to John's profile  john\_decision = loan\_approval\_logic(      john\_profile["credit\_score"],      john\_profile["annual\_income"],      john\_profile["loan\_amount"],      john\_profile["debt\_to\_income\_ratio"]  )  # Apply the loan approval logic to Priya's profile  priya\_decision = loan\_approval\_logic(      priya\_profile["credit\_score"],      priya\_profile["annual\_income"],      priya\_profile["loan\_amount"],      priya\_profile["debt\_to\_income\_ratio"]  )  print(f"Loan approval decision for John: {john\_decision}")  print(f"Loan approval decision for Priya: {priya\_decision}")  OUTPUT:Loan approval decision for John: Approved Loan approval decision for Priya: Approved  **Task Description#3 (Transparency)**   * Write prompt to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document   **Expected Output#3**   * Code with explanation * **Assess: Is the explanation understandable and correct?**   **PROMPT:\*\*Task:\*\*** Explain the generated Python code for calculating the nth Fibonacci number using recursion.  **\*\*Description:\*\*** Provide a clear explanation of how the recursive function works, including the base cases and the recursive step.  def calculate\_fibonacci(n):      """      Calculates the nth Fibonacci number using recursion.      The Fibonacci sequence is a series of numbers where each number is the sum      of the two preceding ones, usually starting with 0 and 1.      (e.g., 0, 1, 1, 2, 3, 5, 8, 13, ...)      Args:          n (int): The position of the Fibonacci number to calculate (n >= 0).      Returns:          int: The nth Fibonacci number.      Raises:          ValueError: If n is a negative integer.      """      # Base cases for the recursion      if n < 0:          # Handle negative input as Fibonacci is defined for non-negative integers          raise ValueError("Input must be a non-negative integer")      elif n == 0:          # The 0th Fibonacci number is 0          return 0      elif n == 1:          # The 1st Fibonacci number is 1          return 1      else:          # Recursive step: The nth Fibonacci number is the sum of the (n-1)th and (n-2)th Fibonacci numbers          return calculate\_fibonacci(n - 1) + calculate\_fibonacci(n - 2)  # Example usage:  # print(calculate\_fibonacci(0))  # print(calculate\_fibonacci(1))  Explanation of the Recursive Fibonacci Function: The `calculate\_fibonacci(n)` function calculates the nth number in the Fibonacci sequence. It uses recursion, meaning the function calls itself within its definition. Here's how it works: 1. \*\*Base Cases:\*\* - If `n` is less than 0, it raises a ValueError because the Fibonacci sequence is defined for non-negative integers. - If `n` is 0, the function returns 0. This is the starting point of the sequence. - If `n` is 1, the function returns 1. This is the second number in the sequence. 2. \*\*Recursive Step:\*\* - If `n` is greater than 1, the function calculates the nth Fibonacci number by adding the (n-1)th and (n-2)th Fibonacci numbers. - It does this by calling itself with `n-1` and `n-2` as arguments (`calculate\_fibonacci(n - 1) + calculate\_fibonacci(n - 2)`). - This process continues until the base cases (n=0 or n=1) are reached, at which point the results are returned back up the call stack to compute the final result for the original `n`. This recursive approach directly reflects the mathematical definition of the Fibonacci sequence.  **Task Description#4 (Bias)**   * Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings.   **Expected Output#4**   * Python code * Analyze is there any bias with respect to gender or any   **PROMPT:**### Subtask:  Create a Python function that takes applicant features (education, experience, gender, age, etc.) as input and calculates a score.  applicants = [      {          "education": "Bachelor",          "experience\_years": 5,          "gender": "Female",          "age": 28      },      {          "education": "Bachelor",          "experience\_years": 5,          "gender": "Male",          "age": 32      },      {          "education": "Bachelor",          "experience\_years": 5,          "gender": "Non-binary",          "age": 40      },      {          "education": "Bachelor",          "experience\_years": 5,          "gender": "Female",          "age": 35      },      {          "education": "Bachelor",          "experience\_years": 5,          "gender": "Male",          "age": 25      }  ]  print("Test cases created with similar qualifications but varying gender and age:")  for applicant in applicants:      print(applicant)  for applicant in applicants:      applicant['score'] = score\_applicant(          education=applicant['education'],          experience\_years=applicant['experience\_years'],          gender=applicant['gender'],          age=applicant['age']      )  print("Applicant profiles with calculated scores:")  for applicant in applicants:      print(applicant)  for applicant in applicants:      print(f"Applicant: Gender={applicant['gender']}, Age={applicant['age']}, Score={applicant['score']}")  print("\nAnalysis for potential bias:")  print("Based on the output, all applicants with identical education and experience levels received the same score (55), regardless of their gender or age.")  print("This indicates that the current scoring logic, which only considers education and experience, does not exhibit bias based on gender or age.")  print("The scoring criteria (education and experience points) are applied uniformly to all applicants, leading to consistent scores for individuals with the same qualifications.")  print("To analyze for potential bias related to gender or age, the scoring logic itself would need to incorporate these features, which it currently does not.")  Summary of Bias Analysis: Based on the current implementation of the job applicant scoring system, no bias related to gender or age was observed. This is because the scoring logic is solely based on objective criteria: education level and years of experience. Gender and age were included as parameters in the function but were explicitly not used in the calculation of the score. Potential Ways Bias Could Be Introduced: Even if gender and age are not directly used, bias could still enter the system in several ways: 1. Unfair Weighting: If demographic features like gender or age were included in the scoring logic with weights that unfairly disadvantage certain groups. 2. Biased Training Data: If a more complex scoring model were trained using historical data where hiring decisions were influenced by bias, the model could learn and perpetuate that bias. 3. Proxy Variables: If the scoring system uses other features that are highly correlated with protected attributes (like gender or age) and these features are given unfair weight. Potential Mitigation Strategies to Reduce Bias: To reduce bias in job applicant scoring systems, consider the following strategies: 1. Focus on Job-Relevant Criteria: Ensure that the scoring system only uses criteria that are directly related to the requirements and performance of the job. 2. Regular Auditing: Periodically audit the scoring system's outcomes to check for disparate impact on different demographic groups. Analyze if certain groups are consistently receiving lower scores or being filtered out at higher rates. 3. Bias Detection and Mitigation Algorithms: If using machine learning models, employ techniques specifically designed to detect and mitigate bias in the model or the training data. 4. Diverse Development Teams: Ensure that the teams developing and testing the scoring system are diverse, as different perspectives can help identify potential sources of bias. 5. Transparency and Explainability: Make the scoring logic as transparent as possible and aim for explainable AI models where possible, to understand why a particular score was given.  **Task Description#5 (Inclusiveness)**   * Code Snippet     **Expected Output#5**   * Regenerate code that includes **gender-neutral** also   def greet\_user(name, gender):      """      Greets the user based on their name and gender.      Args:          name (str): The name of the user.          gender (str): The gender of the user ('Male', 'Female', or 'Other').      """      if gender.lower() == 'female':          print(f"Hello Ms. {name}!")      elif gender.lower() == 'male':          print(f"Hello Mr. {name}!")      else:          print(f"Hello {name}!")  # Example usage:  greet\_user("Alice", "Female")  greet\_user("Bob", "Male")  greet\_user("Charlie", "Non-binary")  **OUTPUT:** Hello Ms. Alice! Hello Mr. Bob! Hello Charlie!  **Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots**  **Evaluation Criteria:**   | **Criteria** | **Max Marks** | | --- | --- | | Transparency | 0.5 | | Bias | 1.0 | | Inclusiveness | 0.5 | | Data security and Privacy | 0.5 | | **Total** | **2.5 Marks** | | | | | | | Week3 - Wednesday |  |