|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** | | | | 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) | | | | | | |
| **Course Code** | | | 24CS002PC215 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week9 - Thursday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | |  | | | |
| **AssignmentNumber:18.4**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab 18 – API Integration: Connecting to External Services with Error Handling**  **Lab Objectives:**   * Learn how to integrate Python programs with external REST APIs. * Understand API request/response handling using requests or similar libraries. * Implement proper error handling for failed API calls (timeouts, invalid responses, rate limits). * Practice extracting and displaying meaningful information from API responses.   **Task 1: Connect to a Public API**   * **Instructions:**   + Use Python (or Node.js/JavaScript) to connect to a public API (e.g., OpenWeatherMap or JSONPlaceholder).   + Send a simple GET request to retrieve data.   + Display the response in a readable format (pretty JSON). * **Expected Output:**   1. Program prints API response (e.g., weather data or posts from JSONPlaceholder).   2. Output should be properly formatted JSON (not raw).   **Prompt: write a python function that display weather details of a city using weather api without error handling. Display weather details as JSON output**  **To use this code:**   1. **First, you'll need to sign up for a free API key at Open Weather Map:**[**https://openweathermap.org/api**](vscode-file://vscode-app/c:/Users/CAIDL/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) 2. **Replace 'YOUR\_API\_KEY' in the code with your actual API key** 3. **Install the required requests package by running:**   **pip install requests**  **You can then run the script and enter any city name to get its current weather details. The output will look something like this:**  **CODE:**    OUTPUT:      **Task 2: Add Error Handling for Invalid API Calls**   * **Instructions:**   + Modify your code from Task 1 to handle errors.   + Include try/except (Python) or try/catch (JavaScript) blocks.   + Handle cases like:     - Invalid URL     - Network timeout     - Wrong API key (if required)   + Print user-friendly error messages. * **Expected Output:**   1. If the API works, the result is shown as in Task 1.   2. If there’s an error, output:   **Error: Could not connect to API. Check your API key or network connection.**  **Prompt: write a python function that display weather details of a city using weather api with error handling. Display weather details as JSON output**  **To use this code:**   1. **First, you'll need to sign up for a free API key at Open Weather Map:**[**https://openweathermap.org/api**](vscode-file://vscode-app/c:/Users/CAIDL/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) 2. **Replace 'YOUR\_API\_KEY' in the code with your actual API key** 3. **Install the required requests package by running:**   **CODE:**    **OUTPUT:**    **Task 3: Extract and Display Specific Data**   * **Instructions:**   1. From the API response (e.g., weather API), extract specific fields (temperature, humidity, description).   2. Display them in a user-friendly format (not raw JSON). * **Expected Output:** * City: London * Temperature: 18°C * Humidity: 60%   Weather: Clear sky  CODE:    OUTPUT:    **Task 4: Build a Function with Parameters**   * **Instructions:**   + Write a function that accepts a parameter (e.g., city name for weather API).   + The function should call the API dynamically based on user input.   + Include error handling if the city is invalid. * **Expected Output:**   1. Input: "New York"   2. Output:   3. City: New York   4. Temperature: 22°C   5. Humidity: 55%   6. Weather: Few clouds   7. Input: "xyz123"   8. Output:   **Error: City not found. Please enter a valid city.**  **CODE:**    **OUTPUT:**    **Prompt: write a python function that display weather details of a city using weather api with error handling. Display weather details in user friendly format**  **Task 5: Store API Results Locally**   * **Instructions:**   + Extend your function from Task 4.   + Save the extracted API results into a local file (results.json or results.txt).   + Each new request should append results without overwriting old ones**.** * **Expected Output:**   1. Console still shows formatted output.   2. A local file results.json is created/updated with stored responses like:   3. [   4. {"city": "London", "temp": 18, "humidity": 60, "weather": "Clear sky"},   5. {"city": "New York", "temp": 22, "humidity": 55, "weather": "Few clouds"}   ]  CODE:      **OUTPUT:**    **Prompt: write a python function that display weather details of a city using weather api with error handling. Display weather details as JSON output. Store the weather details in current directory as text file, every run output will append**  ✅ Deliverables (For All Tasks)   1. AI-generated prompts for code and test case generation. 2. At least 3 assert test cases for each task. 3. AI-generated initial code and execution screenshots. 4. Analysis of whether code passes all tests. 5. Improved final version with inline comments and explanation. 6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.   Top of Form | | | | | | Week9 - Thursday |  |