**Tasks for AI - Phase 1 (Detailed)**

This document will provide a comprehensive view of all Phase 1 tasks, with detailed descriptions and subtasks for AI assistants.

**I. Task Definitions:**

* **Task 1: Project Setup & Environment**
  + **ID:** task1
  + **Goal:** Establish the foundational development environment for the DACS project by installing all necessary tools, libraries, and files.
  + **Subtasks:**
    - Install Python 3.x (or a specific version you have selected).
    - Create a virtual environmenat for the project.
    - Install required Python libraries: Flask, google-generativeai, Pandas, Faker.
    - Initialize a Git repository on GitHub, GitLab, or Bitbucket, and make a first commit.
    - Create a basic Flask app structure (app.py, templates, static folders).
    - Test the Flask application by visiting http://127.0.0.1:5000/
  + **Relationships:** None.
* **Task 2: Gemini API Exploration & Integration**
  + **ID:** task2
  + **Goal:** Explore the Gemini API and implement a basic integration with the python codebase for future use.
  + **Subtasks:**
    - Review the official Gemini API documentation for details on authentication, text generation, request parameters, and response format.
    - Obtain a Google Cloud API key from the Google Cloud Console and store it securely as an environment variable.
    - Install the google-generativeai Python library in the created virtual environment.
    - Create a code snippet to test basic text generation with simple prompts to understand API behavior.
    - Explore different available Gemini models to determine the best fit for scenario generation.
  + **Relationships:** task1
* **Task 3: Scenario Generation Logic with Gemini**
  + **ID:** task3
  + **Goal:** Implement the logic to generate realistic, diverse client scenarios using the Gemini API with specific data domains.
  + **Subtasks:**
    - Define a variety of prompt templates for different data domains, that include all important parts (Client profile, Business task, Data provided)
    - Experiment with prompt variations to control the output's complexity, length, and specific details.
    - Implement a Python function generate\_scenario that can: take a domain as input, selects a prompt template, calls the Gemini API to generate the scenario text, and return the text from the api response.
    - Add logic to parse Gemini's response in order to separate and retrieve the required data (client profile, business task, data available), you may use splitting or regex for this.
    - Implement robust error handling to catch potential API errors (such as timeouts, authentication, invalid requests) and log the details using a python logging module.
  + **Relationships:** task2
* **Task 4: Dataset Generation Logic**
  + **ID:** task4
  + **Goal:** Implement a function that generates realistic and diverse datasets based on a data schema, and including a series of data quality issues.
  + **Subtasks:**
    - Design data schemas (using Python dictionaries) based on common data types used in analysis (numerical, categorical, datetime, text), specifying how to create and fill those schemas.
    - Implement a Python function named generate\_dataset that takes as input the data schema you have designed and a number of rows, to create a pandas DataFrame.
    - Use the library Faker to populate the columns of the dataframe with realistic data based on their respective type and the data provided in the schema.
    - Add logic to randomly introduce missing values using probability by inserting a NaN or None value.
    - Add logic to generate outliers (using uniform or normal distributions) based on a probability to have outliers and generate new values outside of the schema.
    - Implement logic to introduce inconsistent formatting (e.g. for dates MM/DD/YYYY, YYYY-MM-DD), and mixed casing (e.g. UPPERCASE, lowercase) in strings.
  + **Relationships:** task1, task3
* **Task 5: Backend Integration - Scenario & Data**
  + **ID:** task5
  + **Goal:** Create Flask routes for the backend and connect all the python modules and create data flow.
  + **Subtasks:**
    - Create Flask routes:
      * /generate\_scenario: Handle scenario generation requests.
      * /download\_data: Handle data download requests.
  + Implement a Python function named create\_scenario\_and\_data to orchestrate the process:
    - Call the function generate\_scenario using the domain as parameter to generate the scenario.
    - Parse the scenario output to identify the data description and create the correct schema.  
      \* Pass the data\_schema and number of rows to the generate\_dataset to create a dataframe.  
      \* return both the scenario and the dataframe.
  + Implement logic within the /download\_data route that:
    - Receives the generated Pandas DataFrame.
    - Converts the DataFrame to a CSV string in memory using df.to\_csv(index=False).
    - Serves the CSV string for download, setting the appropriate HTTP headers for a downloadable file (such as, Content-Disposition and Content-Type).
  + **Relationships:** task3, task4
* **Task 6: Basic Front-End Layout**
  + **ID:** task6
  + **Goal:** Implement a basic structure to display the DACS application on the web, using HTML and CSS.
  + **Subtasks:**
    - Create an index.html file inside of the templates folder:
      * The HTML must include a title for the page.
      * A description that explains the application's purpose.
      * A button (with id= generate-button) to generate new scenarios.
      * A placeholder div (with id= scenario-output ) to display the generated text.
      * A link or button (with id= download-link) to download the data.
    - Create a CSS file called style.css inside of the static folder, and link it to your HTML document:
    - Add basic styling to center the elements of the page.
    - Apply styles to make the text readable.
    - Style the button to be clear and easy to understand.
  + **Relationships:** task1
* **Task 7: Basic Front-End JS Interaction**
  + **ID:** task7
  + **Goal:** Create the functionality for the front end of the DACS application by adding javascript to make all elements work properly.
  + **Subtasks:**
    - Create a file called script.js inside of the static folder.
    - Implement a javascript function (getScenario()) that:
      * Is triggered by the "Generate Scenario" button (with id=generate-button).
      * Makes an AJAX request to the /generate\_scenario endpoint.
      * Retrieves the generated scenario text and its corresponding data download link from the backend response.
  + Add javascript logic to display the scenario text, using the placeholder div with id (scenario-output).
  + Update the href attribute of the html element that allows for data download using the provided link from the backend response.
  + Implement error handling for API requests, and display those errors using javascript's alert functionality.
  + **Relationships:** task6, task5
* **Task 8: Testing & Refinement - Scenario & Data**
  + **ID:** task8
  + **Goal:** Thoroughly test all the DACS application's components, finding bugs and identifying areas for refinement, while focusing on the core functionalities.
  + **Subtasks:**
    - Generate multiple scenarios across different data domains and review them to see that they are realistic, diverse, and understandable.
    - Inspect the generated datasets using different tools and ensure the data types are correct, that the data quality issues are present (missing values, outliers, and inconsistent formatting), and that the data aligns with the generated scenario description.
    - Check that the button "Generate Scenario" works every time, and that it loads the new response properly.
  + Verify that the download link is working correctly by trying to download the data for several different datasets.
  + Verify that your code is displaying the error messages and that they are clear and concise.
  + **Relationships:** task7, task5
* **Task 9: Documentation & Initial Tracking Implementation**
  + **ID:** task9
  + **Goal:** Ensure that the codebase is well documented, and that all important aspects of the application are properly logged.
  + **Subtasks:**
    - Add clear code comments to all python, html, css, and javascript code, explaining the purpose and logic of all parts of the code.
    - Create a README.md file in the root of the project directory that provides: a project overview, setup instructions, basic usage instructions, and a licensing section.
    - Implement logging using python's logging module, logging successful scenario generation events, dataset generation events, and any errors that may be encountered.
  + **Relationships:** task8