**ClickUp Space: Data Analyst Client Simulator (DACS)**

**List 1: Phase 1 - Core Functionality - Scenario & Dataset Generation MVP**

|  |  |
| --- | --- |
| **Task Name** | **Subtasks** |
| **Project Setup & Environment** | - Install Python 3.x<br>- Create virtual environment<br>- Install Flask<br>- Install google-generativeai<br>- Install Pandas<br>- Install Faker<br>- Initialize Git repository<br>- Create basic Flask app structure |
| **Gemini API Exploration & Integration** | - Review Gemini API documentation<br>- Obtain Google Cloud API key<br>- Install google-generativeai library<br>- Test basic text generation with simple prompts<br>- Explore different Gemini models |
| **Scenario Generation Logic with Gemini** | - Define prompt templates for different data domains<br>- Experiment with prompt variations to control scenario output<br>- Implement Python function to call Gemini API and retrieve scenario text<br>- Parse Gemini output to extract scenario components (Business Task, Data Provided)<br>- Handle potential API errors |
| **Dataset Generation Logic** | - Design data schemas based on common data types<br>- Implement Python function to create Pandas DataFrames<br>- Use Faker to populate DataFrame columns with realistic data<br>- Implement logic to introduce missing values<br>- Implement logic to introduce outliers<br>- Implement logic to introduce inconsistent formatting |
| **Backend Integration - Scenario & Data** | - Create Flask routes for scenario generation<br>- Create Flask routes for data download<br>- Implement Python function to orchestrate scenario generation and dataset creation<br>- Pass scenario information to dataset generation function<br>- Implement logic to generate and serve CSV file for download |
| **Basic Front-End Development** | - Create index.html file<br>- Design basic layout with title, description, and button<br>- Implement JavaScript function to trigger scenario generation on button click<br>- Implement logic to display scenario text on the page<br>- Create download link for CSV file<br>- Implement basic JavaScript alert for error handling |
| **Testing & Refinement - Scenario & Data** | - Generate multiple scenarios for different domains and review for realism<br>- Inspect generated datasets for correctness and presence of data quality issues<br>- Verify that the data aligns with the generated scenario<br>- Test the "Generate Scenario" button functionality<br>- Test the CSV download link<br>- Test error handling for different scenarios |
| **Documentation & Initial Tracking Implementation** | - Add comments to Python code explaining key logic<br>- Create a README.md file with project description and setup instructions<br>- Implement basic logging using Python's logging module to record scenario generation events |

**List 2: Phase 2 - Refinements and Preparation for Future Features**

|  |  |
| --- | --- |
| **Task Name** | **Subtasks** |
| **Enhance Scenario Variety & Realism by Domain** | - Research domain-specific terminology and common business tasks<br>- Refine Gemini prompts to incorporate domain-specific details<br>- Experiment with techniques to generate more complex and nuanced scenarios per domain<br>- Review generated scenarios for domain relevance and accuracy |
| **Improve Dataset Logic & Size Control Foundation** | - Analyze data generation logic for potential improvements in consistency<br>- Research Pandas functionalities for creating DataFrames with specific sizes (e.g., using sample)<br>- Design the structure for storing dataset size preferences (even if not user-configurable yet) |
| **Plan for Domain Selection UI** | - Research common UI patterns for category selection (dropdowns, lists)<br>- Define the list of data domains to be offered<br>- Design the data model for storing and retrieving domain information<br>- Outline the backend logic for filtering scenarios based on domain |
| **Front-End UI/UX Improvements (Basic Styling)** | - Choose a basic CSS framework or define a simple color palette<br>- Apply basic styling to improve readability and visual appeal of the title page and scenario display<br>- Ensure consistent font sizes and spacing |
| **Code Refactoring and Modularization** | - Identify logical modules within the codebase (e.g., scenario generation, dataset generation, API interaction)<br>- Refactor code into separate functions or classes<br>- Improve code organization and readability<br>- Add more detailed comments |
| **Advanced Testing & Error Handling (Phase 1 Features)** | - Implement more specific error handling for Gemini API calls<br>- Implement logging of errors and warnings<br>- Write unit tests for key functions in scenario generation<br>- Write unit tests for key functions in dataset generation |

**List 3: Phase 3 - (Future) Implementing Domain Selection and Dataset Size Control**

|  |  |
| --- | --- |
| **Task Name** | **Subtasks** |
| **Develop Front-End for Domain Selection** | - Implement HTML dropdown element for domain selection<br>- Populate dropdown with list of available domains<br>- Implement JavaScript to handle domain selection changes |
| **Develop Front-End for Dataset Size Slider** | - Implement HTML range input element (slider)<br>- Define minimum and maximum values for the slider<br>- Implement JavaScript to display the selected size |
| **Backend Integration - Domain Selection** | - Modify the Flask route to accept the selected domain as a parameter<br>- Update the scenario generation logic to filter or prioritize scenarios based on the selected domain |
| **Backend Integration - Dataset Size Control** | - Modify the Flask route to accept the desired dataset size as a parameter<br>- Update the dataset generation logic to use Pandas to create a DataFrame with the specified number of rows |
| **Test Domain Selection & Size Control** | - Test generating scenarios for each available domain<br>- Verify that the generated scenarios are relevant to the selected domain<br>- Test the dataset size slider with different values<br>- Verify that the generated datasets have the correct number of rows |

**List 4: Phase 4 - (Future) Implementing the Interactive AI Client**

|  |  |
| --- | --- |
| **Task Name** | **Subtasks** |
| **Explore Gemini for Conversational AI** | - Research Gemini's conversational API endpoints<br>- Experiment with sending user queries and receiving responses<br>- Explore options for maintaining conversation history |
| **Backend Integration - Gemini Interaction** | - Create a new Flask route to handle user queries<br>- Implement logic to send user input to the Gemini API<br>- Implement logic to receive and process Gemini's response |
| **Front-End - Chat Interface Development** | - Create HTML elements for a chat input field and message display area<br>- Implement JavaScript to send user messages to the backend<br>- Implement JavaScript to display AI responses in the chat window |
| **Implement Conversation State Management** | - Explore methods for storing conversation history (e.g., in-memory, database)<br>- Implement logic to include conversation history in prompts to Gemini for context |
| **Test Interactive AI Client** | - Test basic question-and-answer interactions with the AI client<br>- Verify that the AI client understands and responds appropriately to data analysis-related questions<br>- Test the handling of conversation history and context |

**List 5: Phase 5 - (Future) Feedback Mechanism and Portfolio Building Support**

|  |  |
| --- | --- |
| **Task Name** | **Subtasks** |
| **Define Feedback Criteria for Data Analysis** | - Define specific metrics and criteria for evaluating data cleaning, analysis techniques, interpretation, and communication<br>- Document the feedback criteria |
| **Develop Feedback Logic (Potentially with Gemini)** | - Explore Gemini's capabilities for code analysis and natural language understanding for feedback<br>- Implement logic to analyze user submissions against feedback criteria<br>- Design the structure for feedback messages |
| **Backend Integration - Feedback Mechanism** | - Create a Flask route to receive user submissions (code, reports)<br>- Implement logic to trigger the feedback analysis process<br>- Store and retrieve feedback messages |
| **Front-End - Feedback Display** | - Design UI elements to display feedback messages clearly and concisely<br>- Implement logic to associate feedback with specific parts of the user's submission |
| **Implement Portfolio Idea Generation** | - Analyze generated scenarios for potential project ideas<br>- Implement logic to suggest relevant project ideas to the user based on their activity or preferences |
| **Implement User Account Management (If Needed)** | - Design database schema for user accounts<br>- Implement user registration and login functionality<br>- Implement features for storing user progress (optional) |
| **Plan for Deployment and Scaling** | - Research different deployment options (e.g., Heroku, AWS, Google Cloud)<br>- Consider potential scaling challenges and solutions |