**DACS Software Requirement Specification (SRS) Document**

**I. Introduction**

* **Product Overview:**
  + **Purpose:** The Data Analyst Client Simulator (DACS) is a web application designed to provide aspiring data analysts with a realistic environment to practice their skills by engaging in simulated client interactions.
  + **Product Value:** DACS offers a dynamic and challenging environment to improve data analysis skills, build portfolios, and develop communication abilities by providing practical, realistic scenarios, and addressing the need for real-world practice.
  + **Intended Audience:** Aspiring and entry-level data analysts, students in data science programs, individuals looking to build a portfolio, data analysis teams for training purposes (in later phases).
  + **Intended Use:** Users will use DACS to:
    - Generate a scenario and a corresponding dataset that resembles real-world situations.
    - Practice data analysis skills by completing tasks in the generated scenario.
    - (Future) Interact with an AI client that can provide guidance and context, and respond to questions.
    - (Future) Receive feedback from the AI and build a portfolio for display to future employers.
* **Definitions and Acronyms:**
  + **DACS:** Data Analyst Client Simulator
  + **PRD:** Product Requirements Document
  + **SRS:** Software Requirements Specification
  + **API:** Application Programming Interface
  + **UI:** User Interface
  + **CSV:** Comma-Separated Values
  + **NLP:** Natural Language Processing
  + **AI:** Artificial Intelligence
* **Table of Contents:**
  + Introduction
  + Project Brief
    - Background
    - Problem Statements
    - Goals
    - Non-Goals
    - Hypothesis
    - Vision Narrative
    - Rough Scoping & Timeline
    - Key Trade-offs & Decisions
    - Concept Mocks
  + Project Proposal
    - Proposal
    - Risks & Mitigations
    - Open Questions
  + Appendix
    - Research
    - Glossary
    - API Documentation
    - UI Documentation
  + System Requirements and Functional Requirements
  + External Interface Requirements
  + Non-Functional Requirements

**II. Project Brief**

* **Background:**
  + There's a growing need for practical experience in data analysis, but traditional educational paths don't always provide realistic, real-world-based data analysis scenarios. DACS aims to bridge this gap by providing an AI-powered platform for data analysts to practice their skills by engaging in realistic business cases.
* **Problem Statements:**
  + I am an **aspiring data analyst**. I am trying to **build a compelling portfolio**. But **I lack access to real-world data and client interactions** because **my studies and personal projects are not as complex as real jobs** which makes me feel **underprepared for professional data analysis roles**.
  + I am a **data science student**. I am trying to **refine my practical data analysis skills**. But **the scenarios and datasets given by my courses are too simple** because **they do not reflect real world data** which makes me feel **I am not getting all the skills I need to work in the field**.
* **Goals:**
  + Provide aspiring data analysts with a realistic and challenging environment to practice their skills.
  + Enable users to generate and download realistic scenarios and datasets based on different business cases.
  + (Future) Facilitate AI-powered conversations for a more dynamic learning experience.
  + (Future) Provide personalized feedback on data analysis approaches.
  + (Future) Provide portfolio-building tools and ideas.
* **Non-Goals:**
  + This initial version of DACS is not focused on user profiles, a full fledged interactive interface, complex data types (beyond CSV), or real time collaborative features.
  + Monetization is not a concern in this initial version.
* **Hypothesis:**
  + If we **create a functional application that provides realistic data analysis scenarios and downloadable datasets using AI**, then **aspiring data analysts' engagement with the tool increases** leading to **an increase in the speed of their learning and an improvement in their perceived skills**.
* **Vision Narrative:**
  + Imagine an aspiring data analyst, Alex, who has been learning data analysis concepts through online courses, and is now starting to apply for jobs in the field. He has no project that showcases the skills he has acquired in the last years of study. He discovers the DACS application online, and is thrilled at the possibilities of having some real-world scenarios to showcase. He sets out on a path to use DACS to practice analyzing multiple business cases, and creating data visualizations and reports that match the task. He completes several different projects, and builds up a portfolio of compelling projects that they can display in their resume and job interviews, which helps them land their ideal job in data analysis.
* **Rough Scoping & Timeline:**
  + **V1 (MVP):** (Estimated Completion: 2025-03-12)
    - Core functionality: Gemini-powered scenario and dataset generation.
    - Basic web UI for scenario and data download.
  + **V2 (Near Future):**
    - Domain selection and dataset size controls.
  + **V3 (Future):**
    - Interactive AI client for user interaction.
    - AI-powered feedback mechanism.
    - Portfolio building support for users.
  + **Project Size:** Initially a solo development project for personal learning and portfolio. Future versions might be scaled for broader use.
  + **Testing Plan:** Start with internal testing (user testing with you), then progressively expand to small groups of testers.
* **Key Trade-offs & Decisions**
  + **API Choice:** We decided to use the Gemini API for the generative functionalities, but we could have used other models such as OpenAI. Gemini was chosen for being free during its evaluation period.
  + **Development Approach:** An Agile approach was chosen to iterate over a functional base, rather than build a full fledged application from start to finish.
* **Concept Mocks:** *(Incorporate mockups of the basic UI layout from your previous steps here)*.

**III. Project Proposal**

* **Proposal:**
  + This document will act as a guide to build the first version of the DACS application. It contains all the instructions and requirements to complete the first phase, which involves creating a basic and functional application that fulfills all core functionalities.
* **Risks & Mitigations:**
  + **Risk 1:** Failure to understand the Gemini API.
    - **Mitigation:** Thoroughly review the documentation and experiment with basic prompts. Dedicate time for proper research.
  + **Risk 2:** Data quality is not realistic or complex.
    - **Mitigation:** Design detailed data schemas and experiment with different methods for introducing quality problems.
  + **Risk 3:** Inability to balance complexity and flexibility in the initial project.  
    \* **Mitigation:** Develop the minimum viable product first, and then progressively add complexity in future iterations. Focus on a functional core.
  + **Risk 4:** Not enough time for testing.
    - **Mitigation:** Test frequently and integrate testing with the workflow of every task.
  + **Risk 5:** Limited experience with all chosen technologies.
    - **Mitigation:** Dedicate enough time to understanding and studying the documentation of all necessary frameworks and packages (Flask, Pandas, etc.).
* **Open Questions:**
  + Specific range for row and column counts in generated datasets.
  + Acceptable scenario and dataset generation time.
  + Acceptable error rate for scenario and dataset generation.
  + Specific criteria and methods for AI feedback on user analysis (for future iterations).
  + Definition of the initial deployment environment.
  + Decision on whether a database is needed for future features.
  + Scope and duration of temporary storage for generated data.
  + Specific strategies to develop truly unpredictable elements.
  + Beta testing plan and criteria for selecting beta testers.

**IV. Appendix**

* **Research:** *(Insert links to key research documents or studies here)*.
* **Glossary:** *(Expand on terms defined above if needed)*.
* **API Documentation:** *(Include links to the documentation for Gemini and other APIs that are relevant)*
* **UI Documentation:** *(Include links to the chosen CSS framework or other UI related documentation)*

**V. System Requirements and Functional Requirements**

* **Core Functional Requirements:**  
  \* **Scenario Generation:**  
  \* The system shall generate scenarios based on several data domains.  
  \* The scenarios must contain a client profile, a business problem, and a description of data that should be available.  
  \* The scenarios must contain a full name (first string of response).  
  \* **Data Generation:**  
  \* The system shall generate a dataset, based on the data provided in the generated scenarios.  
  \* The datasets should have realistic and diverse data types.  
  \* The datasets should include data quality issues (missing values, outliers, inconsistent formatting).  
  \* **Web Application Interface:**  
  \* The UI shall provide a Generate Scenario button.  
  \* The UI shall display the generated scenario in a clear and readable format.  
  \* The UI shall provide a CSV download link.  
  \* The UI shall display error messages appropriately.  
  \* **Backend Services:**  
  \* The system shall implement all functions in Python.  
  \* The system shall use Flask for all back-end routes.  
  \* The system shall provide data in csv format.  
  \* The Gemini API shall be integrated correctly with authentication.
* **Specific Functional Requirements:**
  + **Prompt Templates:** The system shall store and reuse prompt templates for all domains.
  + **Data Schemas:** The system shall have defined schemas that it can use to build data based on requirements.
  + **Error Handling:** The system shall log and report any error while running the python application, or using the Gemini API, with descriptive error messages.
  + **Code Comments:** All code must be properly commented.
  + **Readme.md:** A proper documentation README.md shall be available in the root of the project.
  + **Logging:** All major steps in the application shall be logged for testing purposes.

**VI. External Interface Requirements**

* **User Interfaces:**
  + Simple HTML-based interface, with a button to generate, a display for the generated text, and a link to the generated data.
  + Clear and concise display.
  + No complicated navigation.
* **Software Interfaces:**
  + Google Gemini API (for scenario generation).
  + Flask (for web application).
  + Pandas (for data handling).
  + Faker (for data generation).
  + Python logging module (for log generation).
* **Communication Interfaces:**
  + HTTP requests/responses for communication between the front-end and back-end.
  + Responses from the Gemini API will be JSON files.

**VII. Non-Functional Requirements**

* **Security:**
  + API key shall not be visible in the code base, instead it will be stored as an OS environment variable.
  + Generated datasets will be handled with no permanent storage, and will be deleted as soon as possible to prevent data storage.
* **Capacity:**
  + The system should generate datasets that are a realistic size.
  + The system should be able to generate new datasets at any time.
* **Compatibility:**
  + The system must work on a basic linux environment.
  + The system must work on at least Python 3.10 and above.
  + The system must use the specific versions of flask, pandas, faker and google-generativeai listed in the environment setup.
* **Reliability and Availability:**
  + The system should be able to generate datasets at all times and function correctly.
  + The system should log errors when they occur to help debug them.
* **Scalability:**
  + Code should be structured in modules to be easy to read, and ready for any future scaling process.
* **Maintainability:**
  + The codebase should be well-structured, readable and have clear comments.
* **Usability:**
  + The user interface should be clear, concise, and easy to use for a first-time user.