**ClickUp Doc: Error Handling Strategy**

**I. Overview**

* **Purpose:** This document defines the strategies for handling errors within the DACS application. It outlines how to detect, log, report, and recover from potential issues, ensuring a stable and reliable application. This plan should cover both the front end and the back end parts of the app.
* **Goals:**
  + Prevent application crashes due to unexpected errors.
  + Provide meaningful error messages to users and developers.
  + Log detailed error information for debugging and analysis.
  + Implement graceful degradation when errors occur.
  + Follow consistent error handling approaches throughout the application.
* **Scope:** This document applies to all parts of the DACS application, including:
  + Python backend code (Flask, data generation, API calls).
  + JavaScript front-end code (UI interactions, API calls).
  + Interaction with external services (Gemini API, ClickUp API).
  + General configuration and setup errors.

**II. General Principles for Error Handling:**

* **Graceful Degradation:** When an error occurs, the application should not crash or leave the user in a broken state. Instead, it should provide a clear error message and, where possible, continue to operate.
* **Informative Error Messages:** Error messages should be informative for both users and developers, they should clearly indicate what went wrong, and guide the user to the correct way of handling the problem if possible.
* **Clear Error Log:** All errors, including all details, should be logged for debugging and monitoring.
* **Centralized Handling (Where Possible):** Handle errors in centralized locations (such as a global handler in Flask or a shared module for API calls) when appropriate to maintain consistency.
* **Specific Error Handling:** Handle specific errors when needed to implement special logic for common issues.
* **Security:** All sensitive information should not be shown directly to the user through the errors (for example, API keys or internal file structures).
* **User Feedback:** For user facing errors, provide a prompt that is easy to understand for people with no technical background.

**III. Error Handling in Python (Backend)**

* **Core Strategy:** Use try...except blocks to handle potential exceptions in your functions.
* **Specific Error Handling:**
  + **Gemini API Errors:**
    - Use try-except blocks to catch exceptions from the google-generativeai library.
    - Handle specific API errors like invalid requests, rate limits, or authorization errors. Log the specific type of error, the error message, and the code that failed.
    - If the error cannot be recovered gracefully, return a generic error to the user: (e.g., "There was an error processing your request, please try again later")
  + **Data Generation Errors:**
    - Handle errors that can occur during data generation with pandas (e.g., issues reading, formatting).
    - Log the type of error and details about which part of the function failed.
  + **Flask Errors:** Use flask error handlers to catch unhandled exceptions on your app. Use the logging library to log the errors before sending the response to the user.
* **Logging Implementation:**
  + Use Python's logging module to log all errors, and important actions during the execution of the application.
  + Configure a logging file app.log to store errors:  
    python import logging # Configure logging logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", # Save logs to a file datefmt="%Y-%m-%d %H:%M:%S" )
  + Set different log levels (e.g., INFO, ERROR) depending on the severity of the event.
  + Add specific log messages that clearly indicate what action was being performed at the time of the error, and the specific messages given by the libraries.
* Include as much relevant information to every log as possible.
  + Example:
  + def generate\_scenario(domain):
  + logging.info(f"Generating scenario for domain: {domain}")
  + try:
  + # ...Your code here
  + except Exception as error:
  + logging.error(f"Error generating scenario: {error}", exc\_info=True)

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

**IV. Error Handling in JavaScript (Front-End)**

* **Core Strategy:** Use try...catch blocks to handle JavaScript exceptions.
* **Specific Error Handling:**
  + **API Request Errors:** Catch errors from the fetch function for API calls to /generate\_scenario and /download\_data, etc.
  + **JSON Parsing Errors:** Add try catches to handling your API responses, and errors from parsing JSON responses.
  + **General Exceptions:** Catch errors related to DOM manipulation and other javascript code.
* **User Feedback:**
  + Use JavaScript's alert() function to display a message when an error occurs. The message should be clear, concise, and user-friendly. Do not provide detailed technical information to the user, instead, say "An error has occurred, please try again later".
* **Logging:**
  + You can use the browsers console to display console errors when there is no way to catch them.

**V. Specific Error Messages:**

* **Gemini API Failure:** "There was an error in the API call, please try again".
* **Data Generation Failure:** "There was an error generating the data, please try again later".
* **CSV Download Failure:** "There was an error when attempting to download the data, please try again later".
* **Generic Error:** "An error has occurred, please try again later".

**VI. Testing Error Handling:**

* **Simulate Errors:** Intentionally cause errors to test your error handling (e.g., wrong API keys, invalid parameters, missing resources, invalid data).
* **Verify Logging:** Ensure that all errors are correctly logged in the app.log file.
* **User Feedback:** Confirm that error messages are being properly displayed in the front end.

**VII. Logging Levels:**

* logging.info(): Record normal events, such as a new scenario being generated, a dataset being downloaded, the code starting or stopping.
* logging.error(): Record an error that was handled, but that could impact the application in some way, also record the details of what exactly failed.
* logging.exception(): Record a general error, including its traceback. This should be included when you handle an exception.
* console.log() and console.error(): Use the browsers console to log information about your javascript application.