AeroAspire - SDE Intern
Gokul Krishna S

Week 3 – Day 3 (October 9)

Questions/Reflections :

1. How do you test error flows (client sends invalid data)?

   To test how your backend handles invalid inputs, send intentionally wrong or malformed data from the client (React/fetch/Axios) to your Flask app—think missing fields, wrong data types, or out-of-range values. Check if the server responds with HTTP error codes like 400 (Bad Request) or 422 (Unprocessable Entity): these are "client errors" that should trigger readable error messages for the frontend.
   - Use Postman, curl, or your own React inputs to submit test cases.
   - Look for descriptive error messages and proper codes in network tab or console.
   - Examine Flask logs for traceback or error messages.

2. Describe flow of exception in Flask: what happens if an unhandled exception occurs?

   - When Flask encounters an exception that is not caught by your code, it triggers its internal error handlers.
   - If debugging is enabled, Flask shows a detailed stack trace in the browser and logs the exception.
   - In production, it sends a generic error response (like 500 Internal Server Error).
   - You can create custom handlers for specific exceptions using @app.errorhandler(ExceptionType)—these intercept errors and send custom responses.

3. What is CORS? Why browsers block cross origin requests; how to configure CORS in Flask.

    - CORS (Cross-Origin Resource Sharing) is a browser security feature that restricts web pages from making requests to a different domain than the one that served them. This protects users from malicious external scripts accessing sensitive data.
    - Browsers block requests to different origins unless the server explicitly allows it with CORS HTTP headers.
    - If not configured, you'll usually get errors like “CORS policy: No 'Access-Control-Allow-Origin' header” in your browser console.
    - To fix CORS in Flask, use the flask-cors package:

      Python
      ```bash
      from flask_cors import CORS
      ```

4. What is the flow of a fetch/Axios request from React to Flask → response → error handling.

    - React triggers a fetch or Axios call, sending an HTTP request to Flask (endpoint, path, payload).
    - Flask receives the request, validates input, and processes logic (may raise errors).
    - Flask responds—either success (2xx) or error (4xx/5xx) based on validation and outcome.
    - React inspects the response using .then() for success and .catch() for errors, displaying messages or retry logic.

5. How to log or debug failed requests.

    - Effective debugging means tracking and inspecting both frontend and backend errors:

- Backend (Flask): Use print(), Python logging, or Flask’s error logs to check request data, errors, and exceptions. Check terminal and log files for stack traces.
- Frontend (React): Use console.log(), inspect the browser's network tab, and handle errors in Axios/fetch .catch() blocks to show meaningful messages.
- For example, in Flask:

  ```
  Python
  import logging
  logging.error('Error occurred', exc_info=True)
  ```

- And in React:

  ```
  Javascript
  fetch('/api/data')
    .then(res => {/* handle success */})
    .catch(error => console.log(error));
  ```