AEROASPIRE INTERN ASSESSMENT-03
KARTHIK TL

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

To test how the application handles error flows when the client sends invalid data, I create test cases with deliberately malformed or missing inputs to simulate real-world mistakes. For example, during API testing, I would send JSON with missing required fields or invalid types and verify that the server returns appropriate error responses like HTTP 400 with descriptive messages. This helps confirm that validation and error handling work correctly and clients get meaningful feedback

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

Regarding Flask's exception flow, if an unhandled exception occurs during a request, Flask by default returns a generic 500 Internal Server Error response, often with a stack trace in debug mode. To improve this, I usually set up custom error handlers using the @app.errorhandler decorator, so I can respond with structured JSON error messages and appropriate HTTP status codes depending on the error type. This makes the API more predictable and user-friendly

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

CORS (Cross-Origin Resource Sharing) is a security mechanism that restricts how resources are shared between different origins (domains). Browsers block cross-origin requests to prevent malicious sites from accessing resources without permission. To allow cross-origin requests in Flask, I use the Flask-CORS extension which can be configured globally, per resource, or on specific routes to enable the necessary HTTP headers that instruct browsers to permit requests from approved origins.

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

The data flow of a network request from React to Flask generally goes like this: The React app issues a fetch or Axios request with any payload and headers to the Flask backend URL. Flask receives this request, processes it (e.g., reads JSON, interacts with the database), then sends back a response. On React's side, I handle the promise that fetch/Axios returns and check the response. For error handling, I catch exceptions or non-OK HTTP status codes to display error messages or take corrective action.

5. How to log or debug failed requests.

For logging and debugging failed requests, I add logging statements in Flask using Python’s logging module or Flask's built-in logger to record error details along with request context. On the client, I inspect network requests in browser developer tools and handle errors in React by showing friendly messages and optionally logging details for diagnostics. Debugging tools and logs help track down issues in the request lifecycle efficiently