CISC327 Fall 2024 A5
Will Wu, Aidan Sibley, Youssef Elmanawy

How to Test:

1. Clone the repo
2. “cd server” in terminal to traverse to the backend
3. “pnpm install” in terminal to install node dependencies
4. “pnpm test” in terminal to run backend tests

Destination Tests (found in ./server/_tests_/destination.test.mjs):

The Destination API tests ensure that the /api/destinations endpoint behaves correctly when interacting with the database. Before the tests run, the before hook clears the destinations collection in the database to start with a clean slate. The first test sends a GET request to fetch all destinations and verifies that the response has a 200 status and the body is an array. The second test ensures that when there are no destinations in the database, the API returns an empty array with a 200 status. Finally, the after hook closes the database connection to clean up resources and exit gracefully.

Flight Tests (found in ./server/_tests_/flight.test.mjs):

The Flight API tests focus on validating the functionality of the /api/flights endpoint, particularly its ability to handle query parameters and pagination. The first test sends a GET request with specific query parameters and checks that the response includes a 200 status and an array of flights. The second test ensures the API correctly paginates flight results, verifying that the number of flights in the response does not exceed the specified limit. The third test checks that flights in the response include populated origin and destination data, ensuring the API properly resolves these relationships. Finally, the last test ensures that if no flights match the query criteria, the API returns an empty array with a 200 status. These tests together confirm the robustness and reliability of the API's interaction with the database and its ability to handle diverse scenarios.

End to End Testing (found in ./server/_e2e_tests_/e2e.flight.test.js):
The test is an End-to-End (E2E) Test designed to validate the entire flight search workflow from API interaction to frontend rendering. It starts by simulating a user selecting an origin (Toronto), destination (Paris), and departure date (2024-11-15). The test sends a GET request to /api/flights with the necessary query parameters and verifies that the API response includes a 200 status code and a correctly formatted flight object.

After validating the API response, the test mocks frontend rendering by simulating how the application would display the flight information. It creates a string representing the rendered

HTML for the flight details, including the origin, destination, departure date, and price. The test asserts that the rendered HTML contains the correct flight details as provided by the API.

This E2E test ensures that the entire system—from backend to frontend—is working cohesively, mimicking real-world user behavior. It highlights the user journey by testing both the backend API's ability to fetch correct data and the frontend's ability to display that data appropriately. By validating multiple layers of the application, it provides confidence in the seamless operation of the flight search feature.