Testing Strategy and Architecture for Distribusion Web Application

This document outlines the End-to-End (E2E) testing strategy adopted for the case study, focusing on prioritization, architecture, and anticipated challenges.

1. Testing Prioritization

The testing was prioritized based on the criticality of the user's financial transaction and core functionality.

Priority	Area	Rationale
P1 - Critical User Flows	Search, Results Selection, and Checkout Flow.	Any failure here prevents a user from making a purchase, resulting in direct revenue loss.
P2 - Data Integrity & Pricing	Price calculation, currency conversion, and journey details display.	Ensures user trust and financial accuracy. Critical for a booking platform.
P3 - Validation & Usability	Form field validation, filter functionality, and navigation integrity (back button, links).	Ensures a smooth user experience and prevents database corruption from invalid data.
P4 - Edge Cases & Robustness	Handling searches with no results, past dates, and invalid URL parameters.	Verifies system stability and prevents unexpected crashes or confusing error messages.

2. Automation Architecture and Extensibility

The project utilizes **Cypress** with a modified **Page Object Model (POM)** pattern.

Key Architectural Choices:

- **Page Object Model (POM) / Service Objects:** Test logic is separated from UI selectors and page interaction methods. This is essential for:
  - **Maintainability:** If the UI changes (e.g., a selector name), only the relevant Page Object file needs updating, not every test case.
  - **Extensibility:** New test scenarios can reuse existing interaction methods (e.g., `bookingPage.performRoundTripSearch()`).
- **Fixtures and Constants:** Critical data (cities, dates, selectors, messages) are externalized into fixtures (`destinations.json`) and dedicated constant files (`testConstants.js`, `selectors.js`). This makes test data management easier and tests cleaner.
- **Report Consolidation (Mochawesome):** Scripts are configured (`npm run test:reports`) to merge results into a single, comprehensive HTML report, demonstrating readiness for CI/CD reporting.

3. Main Challenges and Mitigation

Challenge	Strategy for Mitigation
Flaky Date Selection	**Challenge:** Cypress might fail to interact with calendar popups or select future dates reliably due to timing issues.
Dynamic Price/Currency Testing	**Challenge:** Prices and discounts change based on external APIs and currency exchange rates.
Selector Changes	**Challenge:** As a React application, selectors may frequently change (e.g., dynamic class names).
External Environment Dependencies	**Challenge:** The test success depends on external bus inventory being available for the chosen routes.