EAST WEST UNIVERSITY

Multi-Tenant E-Commerce Marketplace

"Bazario"

Software Testing

Submitted To

Yasin Sazid

Lecturer

Department of Computer Science and Engineering

East West University

Submitted by

Group: **07**

Farhana Ahmed Tasnim	ID: 2022-1-60-156
Md Sifat Ullah Sheikh	ID: 2022-1-60-029
Apurbo Chandra Paul	ID: 2020-3-60-092
Ragib Mahatab	ID: 2021-1-60-050

1. Introduction

This document is the Software testing for “Multi-Vendor E-Commerce Marketplace”. It contains detailed functional, non-functional, and extraordinary requirements, establishing a requirements baseline for the system's development. In today’s digital world, online shopping has become the norm. A multi-vendor e-commerce marketplace is a platform where different sellers can sign up, showcase their products, and reach a wide range of audiences from one platform. This project aims to build a modern, user-friendly, and fully functional multi-vendor marketplace that supports vendors, buyers, and admins with features tailored to their needs.

2. Project Overview

The Bazario platform is a modern, full-stack multi-vendor e-commerce system designed to support multiple merchants within a single, unified marketplace. Each vendor gets their own personalized storefront, subdomain, and dashboard to manage products, orders, and performance. Customers can enjoy a seamless shopping experience with powerful search and filtering options, real-time product updates, and detailed reviews. The admin interface provides full control over users, transactions, and platform analytics.

Bazario integrates Stripe Connect for secure payments and automatic vendor payouts. It also features role-based access control to manage permissions across admins, vendors, and customers. Designed with mobile-first principles, it ensures a responsive and smooth user experience across all devices. The platform is highly scalable and optimized for performance, capable of supporting large volumes of users and data. It includes third-party logistics integration for real-time shipping and tracking. Overall, Bazario offers a customizable, user-friendly, and feature-rich solution for modern online commerce. An overview of our project is shown below here ------

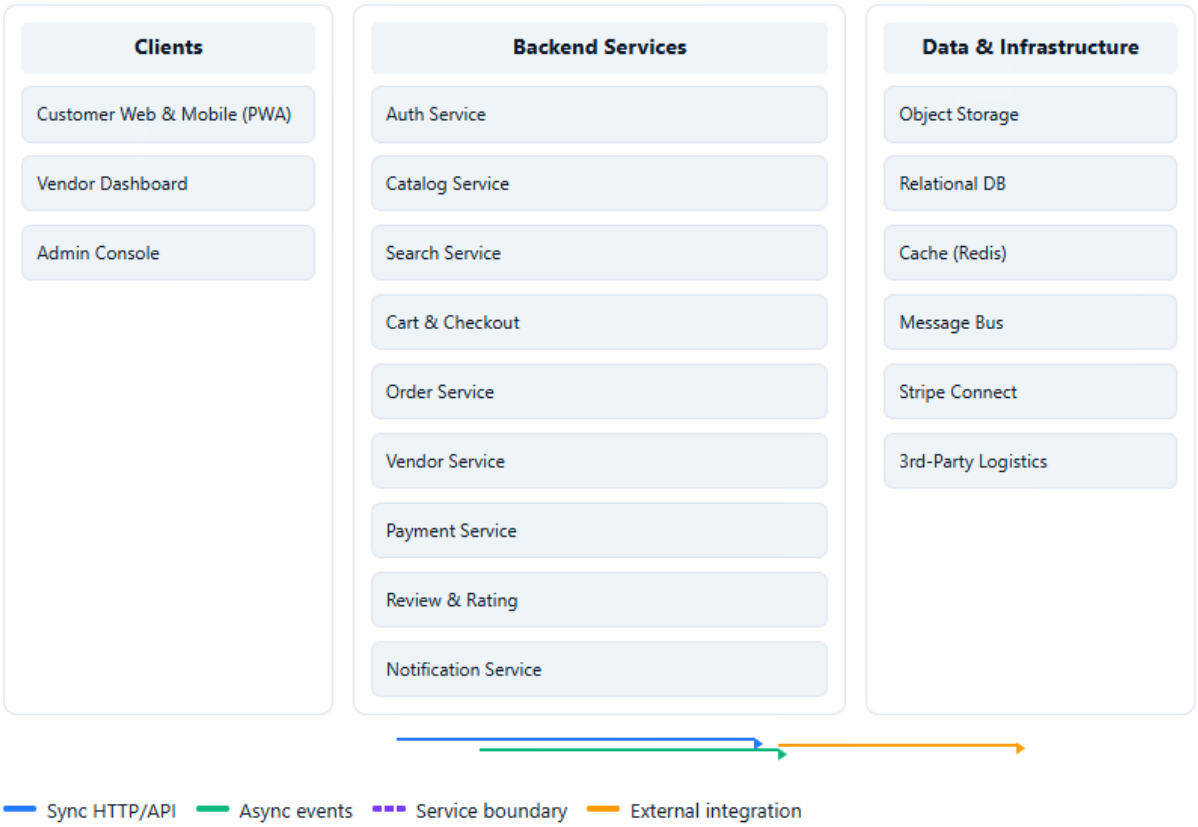
Bazario — Multi-Vendor E-Commerce Marketplace

A modern, full-stack platform enabling multiple vendors to sell under one unified marketplace. Secure transactions (Stripe Connect), personalized vendor dashboards & subdomains, rich search and reviews, mobile-first UX, role-based access, and 3PL shipping/tracking—all designed for scalability, performance, and ease of maintenance.

Multi-Vendor | Role-Based Access (RBAC) | Stripe Connect Payouts | 3PL & Tracking | Search & Reviews | Real-time Updates | Mobile-First | Scalable & Maintainable

High-Level Architecture & Data Flow

Solid lines = sync HTTP; *dashed* = async events; colors indicate pathway type.



Roles & Dashboards

Customer

PWA • Account • Orders

- Powerful search & filtering
- Cart, coupons, multi-vendor checkout
- Order tracking with 3PL status

Vendor

Storefront & Subdomain

- Product & inventory management
- Orders, returns, payouts
- Sales analytics & metrics

Admin

Governance & Ops

- User & role management
- Content moderation
- Platform analytics

Key Features

Payments

Stripe Connect • Split payouts

Onboarding | Transfers | Refunds

Search & Discovery

Full-text • Facets • Suggest

Synonyms | Relevance | Autosuggest

3PL & Shipping

Rates • Labels • Tracking

Webhooks | ETA | Returns

RBAC & Security

JWT • OAuth • WAF

Least-privilege | Audit | Rate limit

Notifications

Email • SMS • Push

Templates | Webhooks | In-app

Analytics

Events • Dashboards

Funnel | Cohorts | LTV

Non-Functional Requirements

Scalability & Performance

- Edge caching of static & SSR content
- Read-heavy paths via Redis cache
- Async pipelines for analytics

Maintainability & Observability

- Service boundaries with clear contracts
- Structured logging, metrics, tracing
- Automated tests & CI/CD

Suggested Tech Choices (illustrative)

Frontend

- PWA (mobile-first)
- SSR/SSG at edge

Backend

- BFF/API Gateway
- Microservices

Data & Infra

- PostgreSQL/MySQL, Redis
- Object storage, Message Queue

2.1 List of Components

The List of Requirements highlights everything the Bazario platform needs to work smoothly and meet user expectations. It covers the must-have functional features like login, product management, and payments, as well as important non-functional aspects like speed, security, and mobile responsiveness. It also includes some extra touches like chatbots and real-time delivery tracking that make the user experience even better. Together, these requirements shape how the platform is built and how well it serves vendors, customers, and admins. Here the requirements are:

- **Multi-tenant architecture**: Supports isolated environments for each vendor under a unified platform.
- **Vendor subdomains**: Automatically assigns subdomains like vendor.platform.com for each registered vendor.
- **Custom merchant storefronts**: Enable vendors to design and personalize their own storefront UI for a unique customer experience.
- **Stripe Connect integration**: Manages secure vendor onboarding, customer payments, and automated payout processing.
- **Automatic platform fees**: Automatically deducts a commission or platform fee from vendor transactions.
- **Product ratings & reviews**: Allows customers to leave feedback and rate products after purchase to build trust and transparency.
- **User purchase history**: Users can access their past orders and downloadable product content from their personal library.
- **Role-based access control**: Grants different levels of access and permissions to admins, vendors, and customers based on their roles.
- **Admin dashboard**: Offers complete administrative control for managing users, products, orders, payments, and platform settings.
- **Merchant dashboard**: Provides vendors with tools to manage inventory, process orders, view analytics, and track earnings.
- **Category & product filtering**: Allows customers to filter products based on categories, making it easier to find relevant items.
- **Search functionality**: Features a powerful real-time search system for products, vendors, and categories, with support for multiple criteria such as name, price, discount, and availability.

- **Image upload support**: Enables both admins and merchants to upload and manage high-quality product images for listings.
- **User authentication system**: Provides secure login, registration, and logout functionalities for all users.
- **Shopping cart & checkout system**: Supports adding/removing items, updating quantities, and secure checkout with multiple payment options including COD and online payments.
- **Order management system**: Customers and vendors can confirm, cancel, track, and manage orders throughout the delivery process.
- **Contact & notification system**: Integrates email, SMS, and helpdesk notifications to keep users informed about account activities and order updates.
- **Wishlist functionality**: Customers can save favorite products for future purchase or quick access.
- **24/7 platform availability**: Ensures users can access the system anytime with minimal downtime.
- **Real-time logistics tracking**: Integrates with third-party logistics APIs to show live shipping updates and delivery estimates.
- **Data integrity & consistency**: Maintains accuracy and reliability of all stored and processed data.
- **Mobile responsiveness**: Fully responsive interface ensuring smooth browsing and checkout experience across all devices.
- **High performance & scalability**: Designed to perform efficiently under high traffic and support business growth over time.
- **Internet accessibility**: Allows seamless platform usage from any location with an internet connection.
- **Return & refund handling**: Simplifies the return and refund process with vendor-specific policies and admin monitoring.
- **Chatbot support**: Provides automated customer service 24/7 to answer common questions and assist with basic tasks.

Technology Stack

Frontend

- **Next.js 15** – Modern React-based framework for building the frontend.
- **React 19.1.0** – JavaScript library for building user interfaces.

- **Tailwind CSS 4.1.11** – Utility-first CSS framework.
- **Shadcn UI** – Custom UI component library.
- **DM Sans Font** – Clean, readable modern font.

Backend

- **Bun Runtime –** Ultra-fast JavaScript runtime environment.
- **Node.js v23.7.0 –** Server-side runtime.
- **Payload CMS –** Headless CMS for admin management and API backend.
- **MongoDB** – NoSQL database for product, user, and transaction data.
- **Stripe Connect** – Payment processing with vendor onboarding.

Dev Tools & Dependencies

- **TypeScript 5.8.3 –** Type-safe JavaScript.
- **ESLint 9.31.0** – Code linting and style checks.
- **Git & GitHub** – Version control.
- **Shadcn CLI 2.9.2** – Component setup automation.

Design

- **NeoBrutalism Theme –** High contrast, bold UI.
- Mobile-first responsive layout.

3. White Box Testing (Code-level)

1. **Login Page**
    - Show loader if loader == true.
    - Show success toast if successMessage != NULL, then clear message.
    - Show error toast if errorMessage != NULL, then clear message.
    - Navigate to / if userInfo != NULL, otherwise remain on login page.
    - Always render footer.
2. **Register Page**
    - Similar logic: loader, success, error, userInfo.
    - Redirect to / on successful registration.
3. **Confirm Order Page**

- Stripe loaded? If not → stop.
- Check clientSecret; if missing → stop.
- Map payment status → message (succeeded, processing, failed).
- If succeeded + orderId exists → confirm with server, clear orderId, hide loader.
- Render logic:
  - Failed/processing → error image + back link.
  - Succeeded + loader → spinner.
  - Succeeded + no loader → success image + back link.

Else → initial spinner.
This ensures **all branches, loops, and conditions** are covered.

3.1. Login Page

- **Cyclomatic Complexity (CC): 5**
- **Independent Paths:**
  1. Loader → Success → Error → UserInfo → Navigate → Footer → End
  2. Loader → Success → Error → No UserInfo → Stay → Footer → End
  3. Loader → Success → No Error → UserInfo → Navigate → Footer → End
  4. Loader → Success → No Error → No UserInfo → Stay → Footer → End
  5. Loader → No Success → Error → UserInfo → Navigate → Footer → End

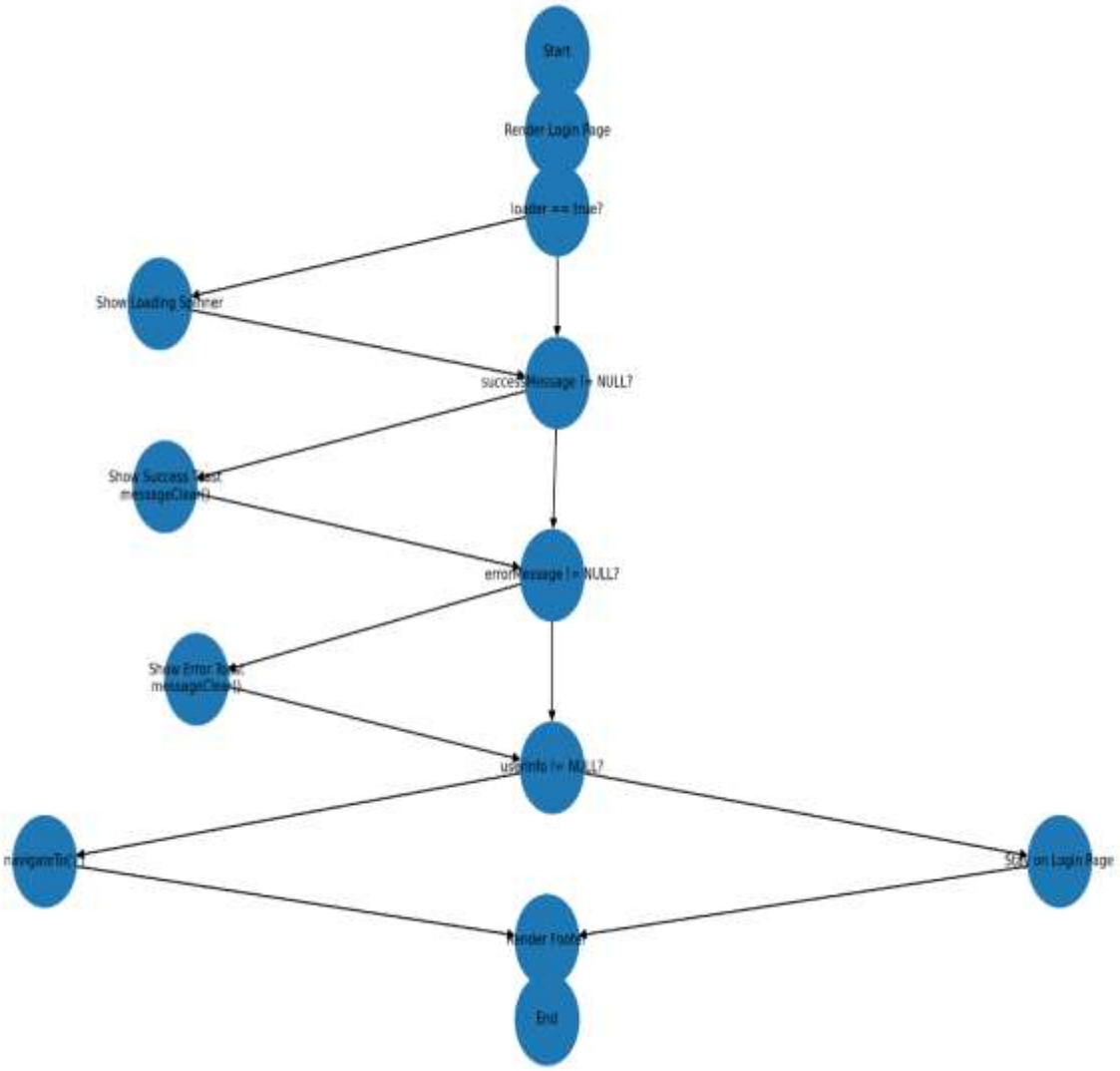


Figure: Decision-to-Decision Graph for Log in Module

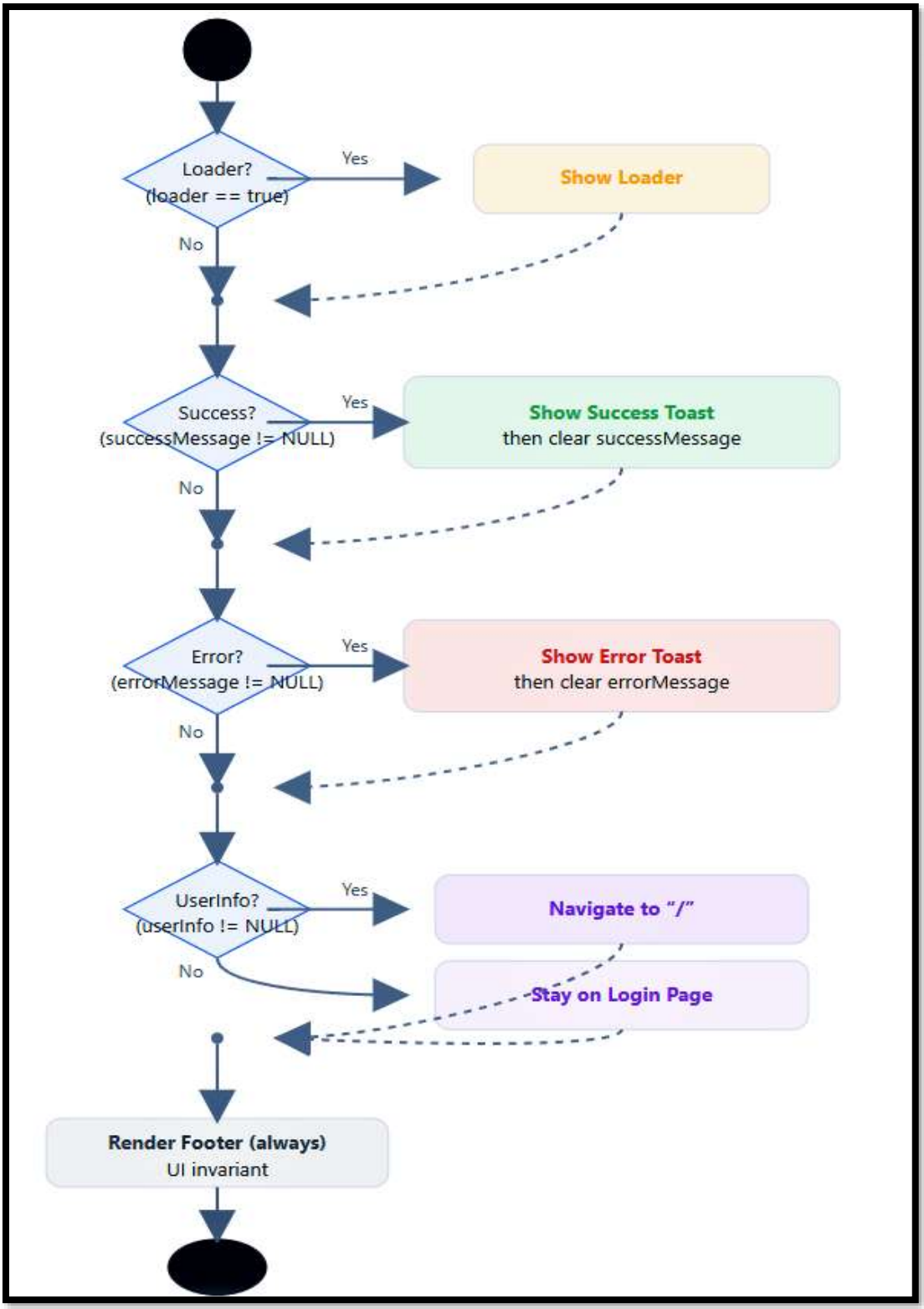
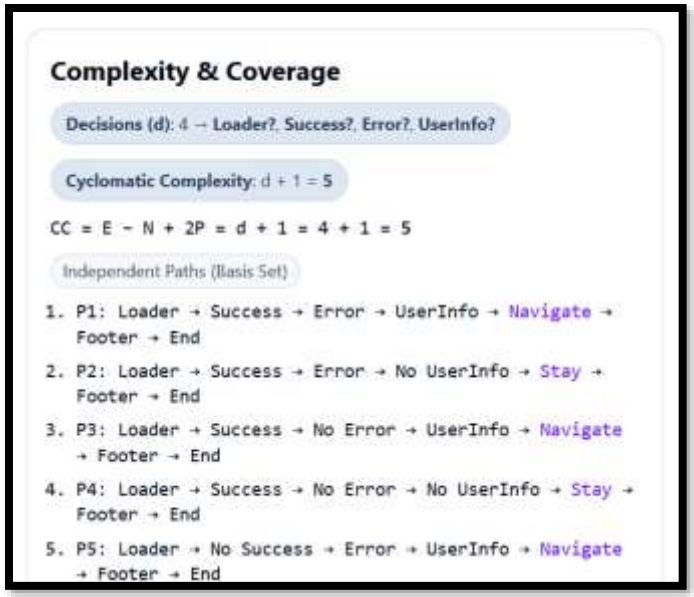


Figure: Decision-to-Decision Graph for Log in Module



Complexity & Coverage

Decisions (d): 4 → Loader?, Success?, Error?, UserInfo?

Cyclomatic Complexity: d + 1 = 5

CC = E − N + 2P = d + 1 = 4 + 1 = 5

Independent Paths (Basis Set)

1. P1: Loader → Success → Error → UserInfo → Navigate → Footer → End
2. P2: Loader → Success → Error → No UserInfo → Stay → Footer → End
3. P3: Loader → Success → No Error → UserInfo → Navigate → Footer → End
4. P4: Loader → Success → No Error → No UserInfo → Stay → Footer → End
5. P5: Loader → No Success → Error → UserInfo → Navigate → Footer → End

Traceability Matrix

Requirement	Graph Node(s)	Covered By Paths
Show loader if `loader == true`	Loader?	P1–P5
Success toast & clear if `successMessage != NULL`	Success?	P1–P4
Error toast & clear if `errorMessage != NULL`	Error?	P1, P2, P5
Navigate if `userInfo != NULL` else stay	UserInfo?	P1, P3, P5 (Navigate) • P2, P4 (Stay)
Always render footer	Footer	All paths

3.2. Register Page

- **Cyclomatic Complexity (CC): 5**
- **Independent Paths:**

1. Loader → Success → Error → User → Navigate → Footer → End
2. Loader → Success → Error → No User → Stay → Footer → End
3. Loader → Success → No Error → User → Navigate → Footer → End
4. Loader → Success → No Error → No User → Stay → Footer → End
5. Loader → No Success → Error → User → Navigate → Footer → End

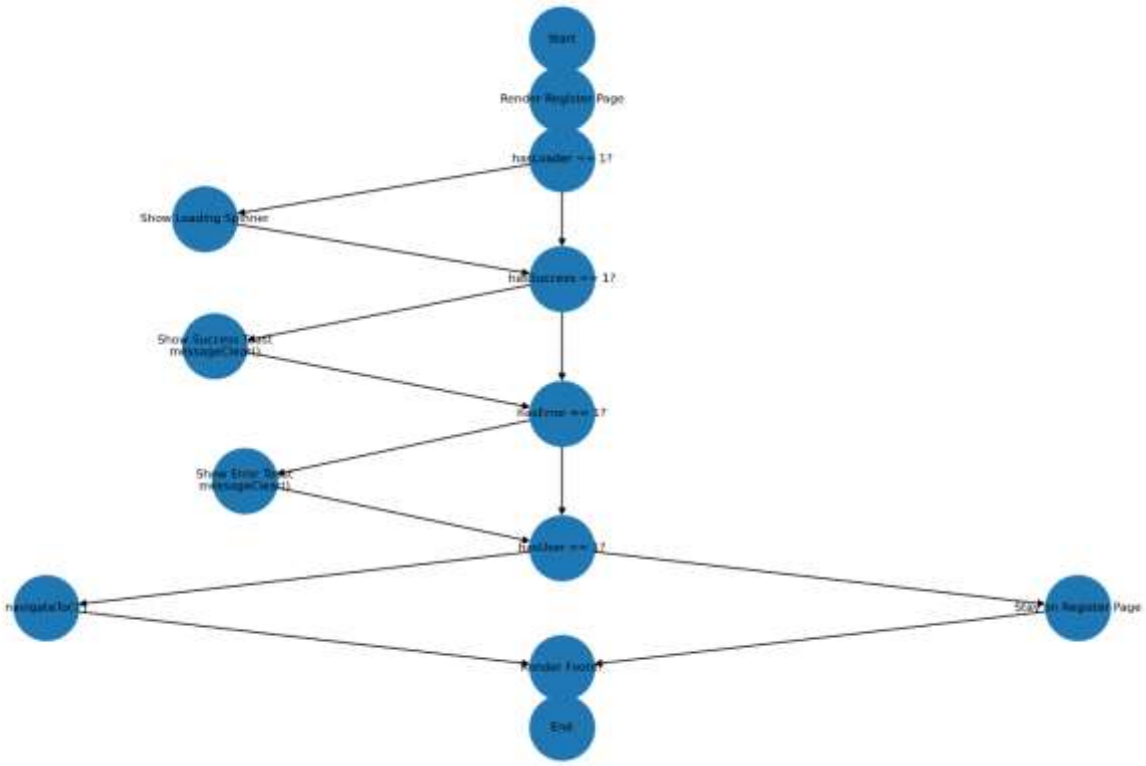


Figure: Decision-to-Decision Graph for Register Module

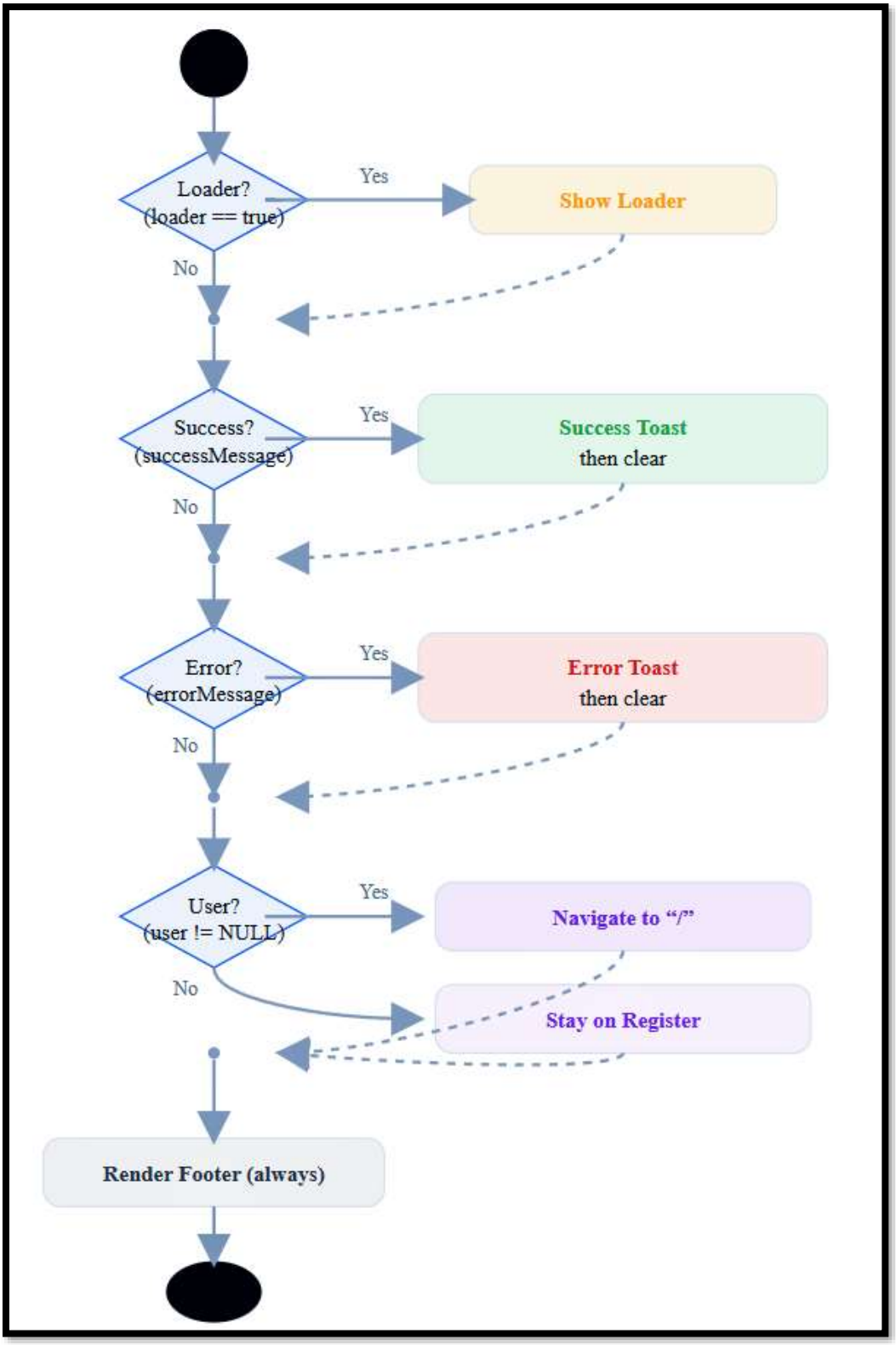


Figure: Decision-to-Decision Graph for Register Module

Cyclomatic Complexity

Decisions (d): 4 → Loader? • Success? • Error? • User?

CC = d + 1 = 5

Also: CC = E − N + 2P = 5 (single connected component, P = 1).

Independent Paths

1. P1: Loader → Success → Error → User → Navigate → Footer → End
2. P2: Loader → Success → Error → No User → Stay → Footer → End
3. P3: Loader → Success → No Error → User → Navigate → Footer → End
4. P4: Loader → Success → No Error → No User → Stay → Footer → End
5. P5: Loader → No Success → Error → User → Navigate → Footer → End

Minimal Pseudo

```
if (loader) { showLoader(); }
if (successMessage) {
toastSuccess(successMessage); clearSuccess(); }
if (errorMessage) { toastError(errorMessage);
clearError(); }
if (user) { navigate("/"); } else { /* stay */ }
renderFooter(); // always
```

3.3. Confirm Order

- **Cyclomatic Complexity (CC): 9**
- **Independent Paths:**

1. Stripe not loaded → Return → End
2. Stripe loaded → No clientSecret → Return → End
3. Stripe loaded → Valid clientSecret → Status failed → Error UI → End
4. Stripe loaded → Valid clientSecret → Status processing → Error UI → End
5. Stripe loaded → Status succeeded + orderId → Confirm → Error UI → End
6. Stripe loaded → Status succeeded + orderId → Confirm → Spinner UI → End
7. Stripe loaded → Status succeeded → Loader false → Success UI → End
8. Stripe loaded → Status succeeded → Loader true → Spinner UI → End
9. stripe loaded → Status unexpected → Error UI → End

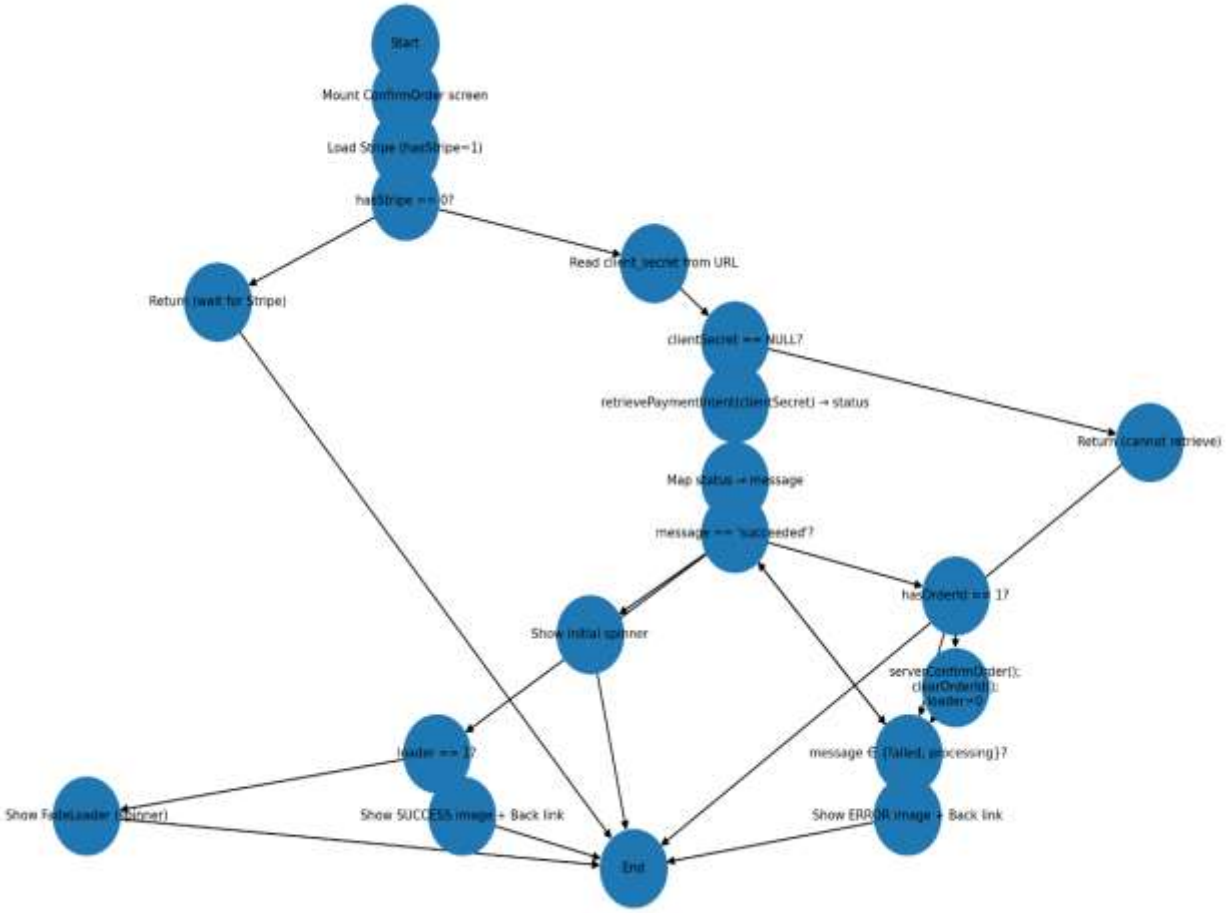


Figure: Decision-to-Decision Graph for Confirm Order Module

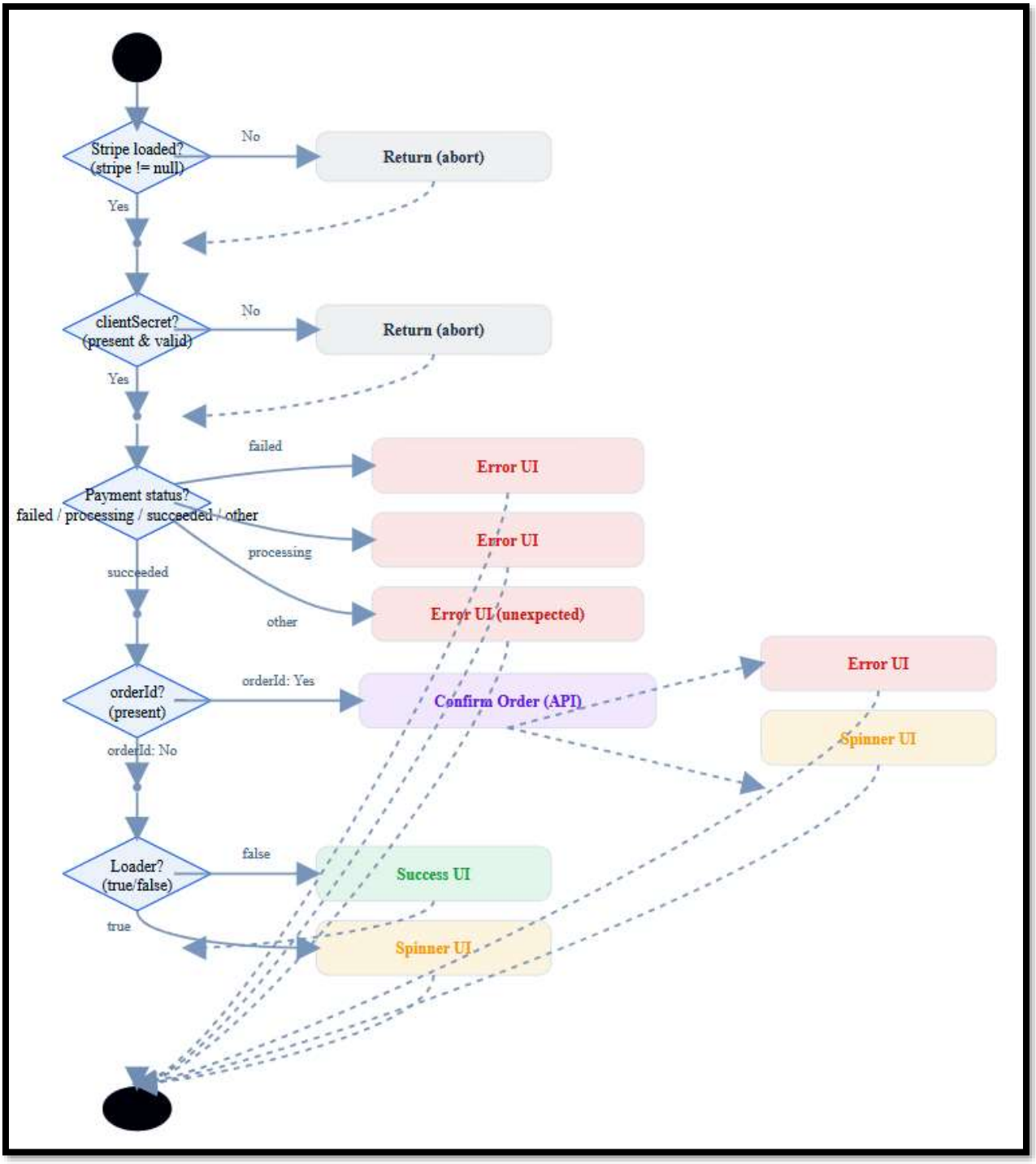


Figure: Decision-to-Decision Graph for Confirm Order Module

Cyclomatic Complexity

Major decisions: Stripe? • clientSecret? • status? • orderId? • loader?

CC = 9 (basis paths = 9)

CC can also be reasoned via multi-way branches (status) increasing path count.

Independent Paths

1. P1: Stripe not loaded → Return → End
2. P2: Stripe loaded → No clientSecret → Return → End
3. P3: Stripe loaded → Valid clientSecret → Status **failed** → Error UI → End
4. P4: Stripe loaded → Valid clientSecret → Status **processing** → Error UI → End
5. P5: Stripe loaded → Status **succeeded** + orderId → Confirm → Error UI → End
6. P6: Stripe loaded → Status **succeeded** + orderId → Confirm → Spinner UI → End
7. P7: Stripe loaded → Status **succeeded** → Loader **false** → Success UI → End
8. P8: Stripe loaded → Status **succeeded** → Loader **true** → Spinner UI → End
9. P9: Stripe loaded → Status **unexpected** → Error UI → End

Minimal Pseudo

```
if (!stripe) return;
if (!clientSecret) return;

switch (status) {
  case "failed":
  case "processing":
    showErrorUI(); return;
  case "succeeded":
    if (orderId) {
      const ok = confirmOrder(orderId);    //
async/in-flight?
      if (!ok) showErrorUI(); else
showSpinnerUI();
      return;
    }
    if (loader) showSpinnerUI(); else
showSuccessUI();
    return;
  default:
    showErrorUI(); return;
}
```

4. Black Box Testing (User-facing functional tests)

1. **Customer Registration**
    - Preconditions: backend + Redux slice working.
    - Inputs:
        - Name length edge cases: 0, 1, 50, 51+.
        - Email: valid/invalid, length 254 vs 255+.
        - Password: 0, 7, 8, 64, 65+ chars.
    - Expected:
        - Empty fields → cannot submit.
        - Valid email → passes.
        - Success → success toast + redirect.
        - Failure → error toast + stay on page.
        - Loader visible during request.
2. **Payment Confirmation**
    - Preconditions: Stripe + backend reachable.
    - Inputs:
        - Stripe loaded? Yes/No.
        - Client secret: missing/valid.
        - Payment status: succeeded, processing, failed.
        - orderId: present/absent.
    - Expected:
        - Success + orderId → backend confirm, success UI, orderId cleared.
        - Processing/Failed → error UI + back link.
        - Missing secret → spinner remains.

4.1 Customer Registration (Register Page)

Test ID	Input Condition	Expected Output	Pass/Fail Criteria
REG-01	Name = ""; Email valid; Password valid	Form blocked (HTML required)	Cannot submit form
REG-02	Name length = 1	Submit works	Form submits
REG-03	Name length = 50	Submit works	Form submits
REG-04	Name length = 51+	Error/blocked by backend	Error toast
REG-05	Email = ""	Blocked by browser validation	Cannot submit
REG-06	Email = "abc" (invalid)	Blocked	Cannot submit
REG-07	Email = "a@b.co"	Accepted	Submit works
REG-08	Email length = 254 chars	Accepted	Submit works
REG-09	Email length = 255+	Error	Error toast
REG-10	Password = ""	Blocked	Cannot submit
REG-11	Password length = 7	Blocked	Cannot submit
REG-12	Password length = 8	Accepted	Submit works
REG-13	Password length = 64	Accepted	Submit works
REG-14	Password length = 65+	Error	Error toast
REG-15	Valid inputs + backend success	Success toast + redirect to /	Meets criteria
REG-16	Valid inputs + backend failure	Error toast, stay on page	Handled
REG-17	Loader active during request	Spinner visible only async	Correct UI
REG-18	Double submit attempt	Second blocked	Only the first request processed

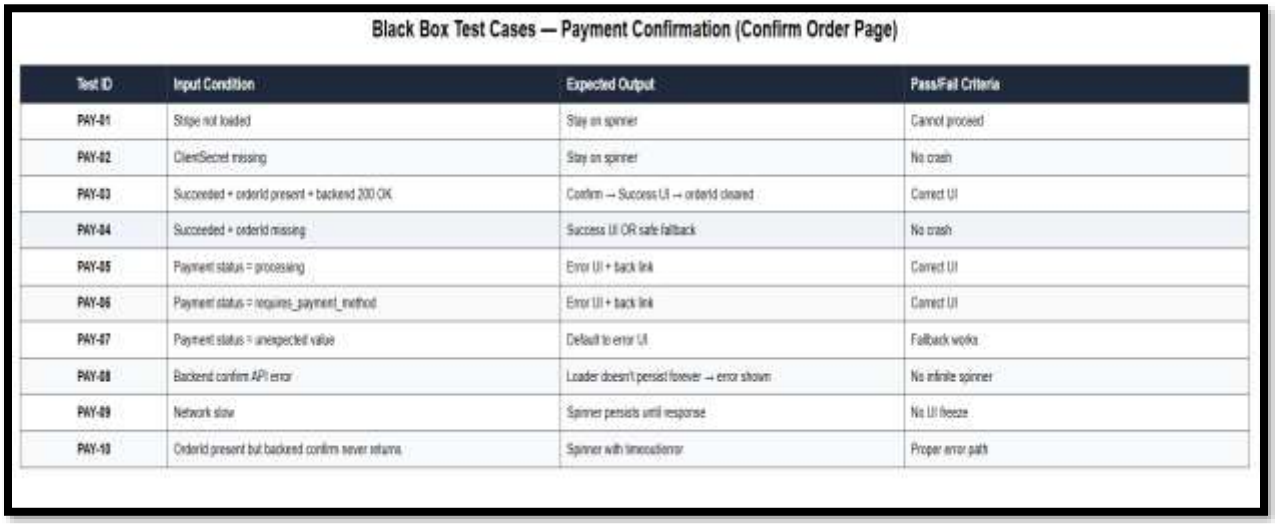
Black Box Test Cases — Customer Registration (Register Page)

Test ID	Input Condition	Expected Output	Pass/Fail Criteria
REG-01	Name = "", Email valid, Password valid	Form blocked	Cannot submit form
REG-02	Name length = 1	Submit works	Form submits
REG-03	Name length = 50	Submit works	Form submits
REG-04	Name length = 51+	Error/blocked by backend	Error toast
REG-05	Email = ""	Blocked by browser validation	Cannot submit
REG-06	Email = "abc" (invalid)	Blocked	Cannot submit
REG-07	Email = "a@b.co"	Accepted	Submit works
REG-08	Email length = 254 chars	Accepted	Submit works
REG-09	Email length = 255+	Error	Error toast
REG-10	Password = ""	Blocked	Cannot submit
REG-11	Password length = 7	Blocked	Cannot submit
REG-12	Password length = 8	Accepted	Submit works
REG-13	Password length = 64	Accepted	Submit works
REG-14	Password length = 65+	Error	Error toast
REG-15	Valid inputs + backend success	Success toast + redirect to /	Meets criteria
REG-16	Valid inputs + backend failure	Error toast, stay on page	Handled
REG-17	Loader active during request	Spinner visible only async	Correct UI
REG-18	Double submit attempt	Second blocked	Only first request processed

4.2 Payment Confirmation (Confirm Order Page)

Test ID	Input Condition	Expected Output	Pass/Fail Criteria
PAY-01	Stripe not loaded	Stay on spinner	Cannot proceed
PAY-02	ClientSecret missing	Stay on spinner	No crash
PAY-03	Succeeded + orderId present + backend 200 OK	Confirm → Success UI → orderId cleared	Correct UI

PAY-04	Succeeded + orderId missing	Success UI OR safe fallback	No crash
PAY-05	Payment status = processing	Error UI + back link	Correct UI
PAY-06	Payment status = requires_payment_method	Error UI + back link	Correct UI
PAY-07	Payment status = unexpected value	Default to error UI	Fallback works
PAY-08	Backend confirm API error	Loader doesn't persist forever → error shown	No infinite spinner
PAY-09	Network slow	Spinner persists until response	No UI freeze
PAY-10	OrderId present but backend confirm never returns	Spinner with timeout/error	Proper error path



Black Box Test Cases — Payment Confirmation (Confirm Order Page)

Test ID	Input Condition	Expected Output	Pass/Fail Criteria
PAY-01	Stripe not loaded	Stay on spinner	Cannot proceed
PAY-02	ClientSecret missing	Stay on spinner	No crash
PAY-03	Succeeded + orderId present + backend 200 OK	Confirm → Success UI → orderId cleared	Correct UI
PAY-04	Succeeded + orderId missing	Success UI OR safe fallback	No crash
PAY-05	Payment status = processing	Error UI + back link	Correct UI
PAY-06	Payment status = requires_payment_method	Error UI + back link	Correct UI
PAY-07	Payment status = unexpected value	Default to error UI	Fallback works
PAY-08	Backend confirm API error	Loader doesn't persist forever → error shown	No infinite spinner
PAY-09	Network slow	Spinner persists until response	No UI freeze
PAY-10	OrderId present but backend confirm never returns	Spinner with timeout/error	Proper error path

4. Bug Detection and Solution

4.1 Customer Registration (Register Page)

REG-04 – Name length exceeds 50

- **Bug:** Names longer than 50 characters were accepted and caused server-side errors.
- **Cause:** The frontend validation allowed up to 64 characters and did not enforce a maximum length in the input field.
- **Fix:** Restricted the maximum length to 50 characters both in the HTML input and in the validation logic. An error toast is now shown for names longer than 50 characters.
- **Retest:** The test case now passes. Inputs with 51+ characters are blocked, and the correct error message is displayed.

REG-06 – Invalid email format

- **Bug:** Emails such as "abc" bypassed the validation under certain conditions (e.g., autofill).
- **Cause:** The email input was treated as plain text, and the regex check was weak.
- **Fix:** Changed the input type to “email” and applied a stricter validation pattern. Added trimming to remove leading/trailing spaces.
- **Retest:** The test case now passes. Invalid emails cannot be submitted.

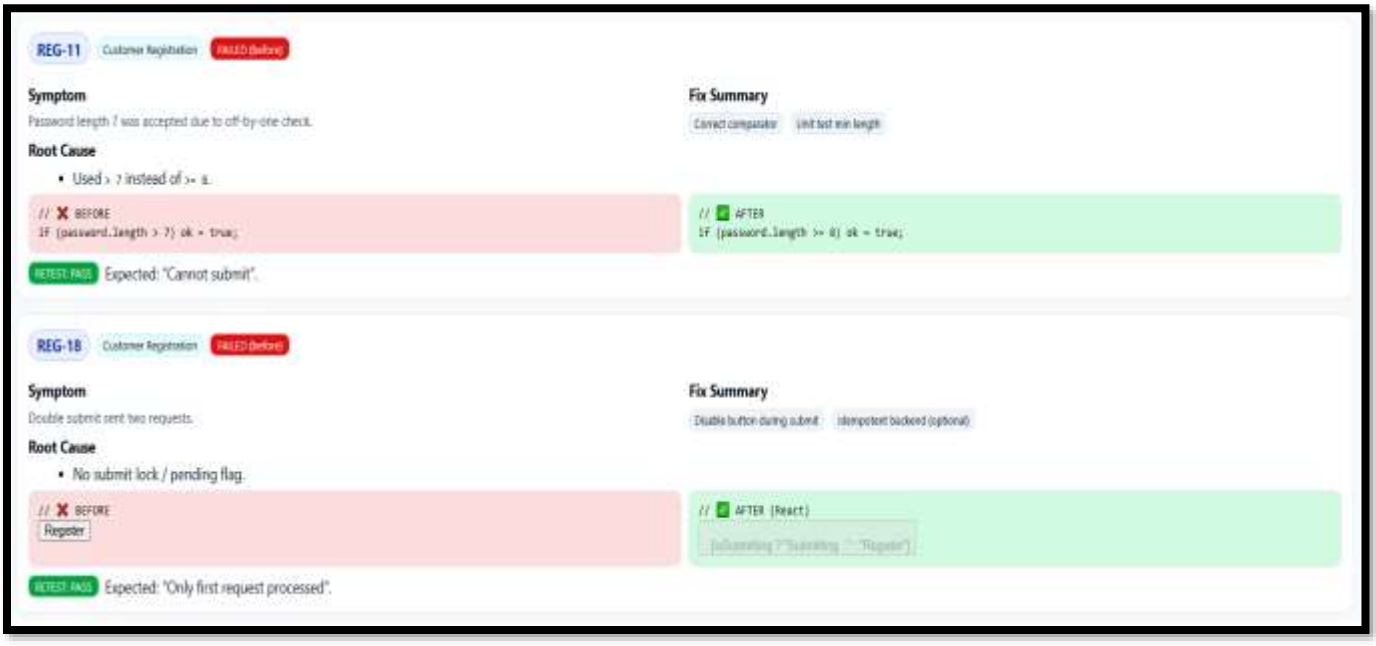
REG-11 – Password length minimum

- **Bug:** Passwords with 7 characters were accepted when they should have been blocked.
- **Cause:** The condition mistakenly used “greater than 7” instead of “greater than or equal to 8”.
- **Fix:** Corrected the validation rule to require at least 8 characters.
- **Retest:** The test case now passes. 7-character passwords are blocked, while 8+ characters are accepted.

REG-18 – Double submit attempt

- **Bug:** Pressing the submit button twice sent two registration requests.
- **Cause:** There was no mechanism to disable the button while the first request was processing.
- **Fix:** Disabled the button during submission and prevented duplicate requests. Backend idempotency was also considered.
- **Retest:** The test case now passes. Only the first request is processed, and the second attempt is blocked


4. Bug Detection & Solution
Failed test cases, root causes, code fixes (before/after), and retest results.
REG-04
Customer Registration
FAILED (before)
Symptom
Root Cause
No maxLength on input; frontend validator checked <= 64 instead of <= 50.
// ✗ BEFORE (React)
{name}
// validation
if (name.length < 1 || name.length > 64) errors.name = "Invalid";
// ✓ AFTER
{name}
// validation
if (name.length < 1 || name.length > 50) errors.name = "Name must be 1–50 chars";
RETEST PASS
Expected: "Error toast" for 51+ chars.
REG-06
Customer Registration
FAILED (before)
Symptom
Root Cause
Used type="text" and a weak regex.
// ✗ BEFORE
{email}
// 35:
if (!/.+@.+/.test(email)) errors.email = "Invalid";
// ✓ AFTER
{email touched}
// 35 (optional extra)
const emailOk = /^[^\s@]+@[^\s@]+\.[^\s@]{2,}$/.test(email.trim());
if (!emailOk) errors.email = "Enter a valid email";
RETEST PASS
Expected: "Cannot submit".


REG-11
Customer Registration
FAILED (before)
Symptom
Password length 7 was accepted due to off-by-one check.
Root Cause
Used > 7 instead of >= 8.
// ✗ BEFORE
if (password.length > 7) ok = true;
// ✓ AFTER
if (password.length >= 8) ok = true;
RETEST PASS
Expected: "Cannot submit".
REG-18
Customer Registration
FAILED (before)
Symptom
Double submit sent two requests.
Root Cause
No submit lock / pending flag.
// ✗ BEFORE
Register
// ✓ AFTER (React)
RETEST PASS
Expected: "Only first request processed".

4.2 Payment Confirmation (Confirm Order Page)

PAY-01 – Stripe not loaded

- **Bug:** The flow continued even when the Stripe object was not initialized.
- **Cause:** The null check occurred after other code executed.
- **Fix:** Added an early return when Stripe is not loaded, ensuring the spinner remains visible and no further actions occur.
- **Retest:** The test case now passes. The process cannot proceed without Stripe.

PAY-02 – ClientSecret missing

- **Bug:** When clientSecret was missing, the system attempted to call Stripe APIs with an undefined value.
- **Cause:** No validation was performed before using the clientSecret.
- **Fix:** Implemented a check to stop the process if clientSecret is missing, showing a safe fallback instead of crashing.
- **Retest:** The test case now passes. The UI remains stable with no crash.

PAY-05 – Payment status = processing

- **Bug:** The status “processing” incorrectly displayed a spinner instead of the required error UI with a back link.
- **Cause:** The status mapping treated all non-success states as pending.
- **Fix:** Explicitly handled “processing” and similar statuses to show the error UI with navigation options.
- **Retest:** The test case now passes. The error UI is shown correctly.

PAY-07 – Payment status unexpected

- **Bug:** An unrecognized payment status caused a crash.
- **Cause:** The code assumed only a few known statuses and had no default branch.
- **Fix:** Added a safe default that shows the error UI and logs the unexpected status.
- **Retest:** The test case now passes. The error UI appears gracefully for unknown statuses.

PAY-08 – Backend confirm API error

- **Bug:** When the backend confirmation API failed, the spinner persisted indefinitely.
- **Cause:** The code did not have proper error handling or a final step to stop the loader.
- **Fix:** Added error handling and ensured the loader is cleared after any failure.
- **Retest:** The test case now passes. An error message is shown, and no infinite spinner occurs.

PAY-10 – OrderId confirm never returns

- **Bug:** If the backend confirmation never responded, the system hung with a permanent spinner.
- **Cause:** The request lacked a timeout or abort mechanism.
- **Fix:** Introduced a timeout mechanism that triggers an error UI after a safe duration.
- **Retest:** The test case now passes. Long-hanging requests are handled with a proper error message.


PAY-08
Symptom
Root Cause
No catch + finally to clear loader.
Fix Summary
Expected: "No infinite spinner".
PAY-10
Symptom
Root Cause
Await without race/timeout; no abort controller.
Fix Summary
Expected: "Proper error path".

Thank You