The Pennsylvania State University

FA II - SWENG 881, Section 001: Software Testing (2025)

Final Group Project

Blazing Pizza – Testing Report

Group 4:

Dana Rashad Wilson

Liyan Alkhalidy

Instructor:

Prof. Nathalia Moraes do Nascimento

Table of Contents

Section 1: Introduction

1.1 Project Name: Blazing Pizza

GitHub Repository: **https://github.com/Liyan5092/BlazingPizza**

1.2. Summary of the Rest of the Test Plan

This test plan outlines the full testing activities conducted for the Blazing Pizza web application. It includes:

- A feature description of the description
- Assumptions and exclusions
- The complete test approach, including tools, automation, and strategies
- Formal test case documentation
- A fully detailed requirements-to-test traceability matrix
- Test environments, results, and final recommendations

This report demonstrates the use of automated testing (Playwright), NUnit backend tests, load testing, and manual exploratory testing.

Section 2: Feature Description

Blazing Pizza is a full-stack web application for ordering customizable pizzas online. Core features include:

- **User Authentication**: Users can log in, as well as register and verify their new accounts.
- **Pizza Selection**: Users browse available pizzas on the home page.
- **Pizza Customization**: Clicking a pizza opens the Configure Pizza page, allowing customization of size and toppings.
- **Order Cart**: Users can review and remove pizzas.
- **Checkout & Order Placement**: Completing an order processes it and routes the user to an order confirmation page.
- **Order Status Tracking**: Orders progress through Preparing, Out for Delivery, and Delivered.

Section 3: Assumptions

3.1 Test Case Exclusions

- **Delivery Map Integration**: Excluded because the Blazing Pizza repository uses hard-coded latitude and longitude values, meaning the map is static and cannot be validated for dynamic routing or delivery-path accuracy.
- **Load Balancing / Distributed Deployment**: Not applicable, as the application runs locally and is not deployed across multiple servers. The project supports functional load testing (e.g., adding 55 pizzas) but not infrastructure-level load balancing.
- **Full Performance Benchmarking**: While a high-volume order test (55 pizzas) was performed, formal performance profiling, such as CPU, memory, throughput, or latency metrics, was not part of the scope and therefore not included.
- **Email / SMS Notifications**: Excluded because these features do not exist in the provided Blazing Pizza codebase, and no external communication API is implemented.
- **Cross-Browser Compatibility Testing**: Only the default Playwright browser context was used. No testing was performed on legacy browsers or alternative engines (e.g., Safari, Firefox, older Edge).
- **Mobile Responsiveness**: UI interactions were explored on desktop viewport sizes, but no mobile or tablet breakpoints were tested. This includes touch interactions, mobile layout, and adaptive controls.

3.2 Test Tools, Formats, and Organizational Schemes

Testing tools - These are the tools that have been used:

- NUnit
- Playwright (UI Automation)
- Playwright CLI
- .NET CLI
- GitHub (YAML)
- Visual Studio Test Explorer
- SQLite
- Blazor - C#
- CodeGen (Running Playwright scripts)

Test Formats - How the test cases are written:

- NUnit test format; [Test], [TestFixture], [TestCaseSource]
- Playwright script format
- Manual test case format (Steps, Expected Results)

Organizational Schemes:

- The tests cases were grouped by features (Login, Pizza Order)
- Naming Convention (FunctionName_Action_WhenDo); e.g. GetBasePrice_Throws_WhenSpecialIsNull()

Section 4: Test Approach

4.1 Past Issues

The Blazing Pizza application is an open-source educational project used to teach Blazor fundamentals. Because the repository is not maintained as a production system, several issues arose when preparing it for automated UI testing and modern .NET development. Two primary problems were identified and resolved prior to executing the test plan:

1. Playwright Integration Issue

The original project did not include support for automated end-to-end testing. To enable UI automation, Playwright had to be manually installed and configured:

- Added the Playwright package:
  dotnet add package Microsoft.Playwright --version 1.55.0
- Updated the project's .csproj to reference the package
- Installed the global Playwright CLI tool
  dotnet tool install --global Microsoft.Playwright.CLI
- Installed required browser binaries
  playwright install
- Used playwright codegen to generate initial C# scaffolding for tests

Outcome: Playwright successfully integrated, allowing automated execution of UI workflows such as login, pizza configuration, checkout, and order tracking.

2. Upgrade to .NET 8 Compatibility

The repository targeted an older .NET version and required an upgrade to run correctly on .NET 8 and support modern dependencies.

The following components were installed or updated:

- .NET 8 SDK (v8.0.415)
- .NET Runtime 8.0.21
- ASP.NET Core Runtime 8.0.21

- Executed dotnet restore, dotnet build, and dotnet run to verify successful migration

Outcome: The application successfully compiled and ran under .NET 8, ensuring compatibility with Playwright and enabling stable automated testing.

Both issues, Playwright setup and .NET 8 migration, were resolved. The environment is now stable, supports end-to-end automation, and provides a reliable foundation for executing all functional, structural, and exploratory test cases defined in this report.

4.2 Special Testing Considerations

- The user is required to select a Special Pizza before proceeding with the application for testing.
- When the user logs in, they must be verified by the system, as the current application does not offer a guest checkout feature.

4.3 Test Strategy

The testing strategy for the Blazing Pizza application combines automated, manual, and exploratory techniques to ensure comprehensive coverage of both functional and non-functional requirements. Each testing area targets a different layer of the system; backend logic, UI workflows, load behavior, and real-world user interactions.
The goal is to validate correctness, reliability, performance under stress, and robustness against unexpected user behavior.

The table below summarizes the techniques, tools, scripts, and rationale used for each major testing area:

Area	Technique	Tool	Script	Rationale
Functional Backend	Unit Testing	NUnit	PizzaTests.cs OrderTests.cs	Ensures logic works
UI Workflow	Automated E2E	Playwright	UI Workflow	Fast UI testing, Realistic test cases
Load Testing	Automated UI	Playwright	PlaceOrderTests.cs	Loop – 55 Pizzas
Manual Exploratory Testing	Manual UI testing	N/A	Checkout Test Bypass Login	Tested the application manually by running it in the browser

4.4 Test Categories

To ensure comprehensive coverage, the test cases were grouped into categories that reflect different testing techniques and system behaviors. Each category targets a specific aspect of the

application, from functional correctness to workflow validation and exploratory discovery. The following table outlines these categories:

Category	Purpose	Testing Basis / Partition / Criteria Used	Description
C1 - Functional Testing: Pizza Customization & Pricing	Verify correctness of pizza customization logic and price calculation across representative combinations.	**Input Domain Modeling; Pair-Wise Coverage (PWC)** across: • **Special Pizza:** Low, Medium, High • **Size:** Small, Medium, Large • **Toppings:** None, Few, Many	Ensures the pricing engine correctly handles various pizza specials, sizes, and topping combinations. Covers baseline, mid-range, and boundary configurations for all three parameters.
C2 - Functional Testing: Ordering Pizzas & Cart Behavior	Validate system behavior for different order/cart structures.	**Input Domain Modeling; Base-Choice Coverage (BCC)** using the order-level partitions: • Empty order • Single pizza • Multiple pizzas • Uniform pizzas (identical entries) • Varied pizzas (different sizes/specials/toppings)	Ensures totals, order summaries, and cart workflows behave correctly for realistic and edge-case order compositions. Validates behavior across empty carts, simple orders, and complex multi-pizza combinations.
C3 - Workflow Path Coverage (Control-Flow Testing)	Validate the complete order-placement workflow, including authentication handling, page navigation, and order-status transitions.	**Graph-Based Modeling; All-Path Coverage** using two generated system paths: • **Path A - Logged-in User:** Order → Checkout → Order Details → Preparing → Out for Delivery → Delivered • **Path B - Not-logged-in User:** Order → Login → Checkout → Order Details → Status transitions	Ensures correct control flow for both authenticated and unauthenticated users, including redirect logic, successful login, checkout navigation, and real-time order status updates across the entire delivery lifecycle.
C4 - Non-Functional	Identify usability issues, inconsistent	**Exploratory Testing Tours + Session-Based**	Covers critical interaction areas not

Testing: Exploratory Testing	UI states, validation weaknesses, and unexpected behavior in high-interaction workflows.	**Test Management (SBTM):** • **C4.1 - Obsessive-Compulsive Tour:** UI responsiveness during rapid pizza customization • **C4.2 - Test One, Get One Free (TOGOF) Tour:** High-volume ordering scenarios • **C4.3 - Antisocial Tour:** Checkout form validation & error-handling weaknesses • **C4.4 - Collector's Tour:** Authentication, redirects, and login-bypass attempts	easily addressed by scripted tests: UI responsiveness, high-order volume edge cases, form validation gaps, redirect inconsistencies, and order-tracking anomalies. Outputs include documented observations, screenshots, exploratory logs, and defects identified for future regression cycles.

Section 5: Test Cases

5.1 Test Group and Subgroup Definition

The test groups and subgroups in this section are derived directly from the test categories defined in **Section 4.4**. Each group represents a major testing focus area, while each subgroup reflects the specific functional modules or workflows covered under that group.

- **Test Groups:** The test cases in this report are organized into four main groups, each aligned with a specific testing technique and system objective. Grouping the tests this way ensures complete coverage of functional logic, workflow behaviors, and exploratory scenarios. The table below summarizes each group and its purpose:

Group ID	Group Name	Objective
G1	Functional Testing - Pizza Customization & Pricing	Validate accuracy of pizza configuration and pricing behavior using PWC across Special category, Size, and Topping partitions.
G2	Functional Testing - Ordering & Cart Behavior	Validate order/cart functionality using Base-Choice Coverage (BCC) across Empty, Single, Multiple, Uniform, and Varied order compositions.

G3	Workflow Path Coverage (Control-Flow Testing)	Validate full ordering workflows (logged-in and not-logged-in) using Graph-Based Modeling and All-Path Coverage.
G4	Non-Functional Exploratory Testing	Identify unexpected behavior, UI inconsistencies, validation weaknesses, and interaction bottlenecks using structured exploratory tours and SBTM.

- **Subgroups:** Each subgroup corresponds to one or more parent test groups based on the testing techniques applied. The table below shows how each subgroup aligns with its associated group(s):

Subgroup Name	Parent Group(s)	Description / Scope
User Authentication	G3, G4	Covers login flows, redirect logic, accessing Checkout while authenticated or unauthenticated, and login bypass attempts.
Pizza Selection	G1, G4	Covers initial pizza choice from the home screen, loading pizza options, and navigating to the Customization page.
Pizza Customization	G1, G4	Covers selecting sizes, toppings, and specials, validating price updates, and UI responsiveness during customization.
Order Cart Management	G2, G4	Covers adding, modifying, removing items; handling Empty, Single, Multiple, Uniform, and Varied pizza configurations.
Checkout Process	G3, G4	Covers address input, form validation, submission, and correct redirection to the Order Details page.
Order Status Tracking	G3, G4	Covers status transitions (Preparing → Out for Delivery → Delivered) and validating real-time updates.

5.2 Test Cases

5.2.1 Input Domain Modeling - Pair-Wise Coverage (PWC)

5.2.1.1 Rationale for using PWC

The Pizza.GetFormattedTotalPrice() method computes the final price of a single pizza by combining:

- **Special Pizza (P1)** - determines the pizza's base price.
- **Size (P2)** - scales the pizza price by diameter.
- **Toppings (P3)** - adds customization cost based on topping count.

Because the method integrates three independent inputs, testing every possible combination (3 × 3 × 3 = 27) is unnecessary.

To reduce the input space while maintaining strong behavioral coverage, the parameters were divided into equivalence partitions. The Pair-Wise Coverage was then applied to ensure:

- Each block participates in testing
- Interactions between blocks are exercised
- Boundary, mid-range, and typical values are represented

This technique validates that Pizza.GetFormattedTotalPrice() correctly handles variations in Specials, Sizes, and Toppings.

5.2.1.2 Partitioning

P1 - Special Pizza (Base Price Tiers)

- **Low** – e.g., Margherita, $9.99
- **Medium** – e.g., Classic Pepperoni, $10.50
- **High** – e.g., Buffalo Chicken, $12.75

P2 - Size (Scaling Factor)

- **Small (9")** – minimum valid size
- **Medium (12")** – default
- **Large (17")** – upper bound

P3 - Toppings (Customization)

- **None (0)** – no toppings
- **Few (2)** – light customization

- **Many (6)** – maximum tested customization tier

5.2.1.3 PWC Test Cases Table

The following test cases were generated using Pair-Wise Coverage to validate the pricing behavior of individual pizzas across combinations of Specials, Sizes, and Toppings. Each case represents a unique interaction between the defined input partitions.

Test Case ID	Partition Values	Objective	Steps	Expected Result
TC-PWC-01	Low, Small, None	Validate lowest-tier baseline pricing	Select Low → Small → 0 toppings	Correct small scaled base price
TC-PWC-02	Low, Medium, Few	Validate low-tier moderate customization	Low → Medium → 2 toppings	Base + size + toppings correct
TC-PWC-03	Low, Large, Many	Validate low-tier upper boundary case	Low → Large → 6 toppings	Highest Low-tier price correct
TC-PWC-04	Medium, Small, Few	Validate medium-tier small pizza	Medium → Small → 2 toppings	Correct adjusted price
TC-PWC-05	Medium, Medium, Many	Validate medium-tier max toppings	Medium → Medium → 6 toppings	High topping total correct
TC-PWC-06	Medium, Large, None	Validate medium-tier plain large pizza	Medium → Large → 0 toppings	Accurate size scaling
TC-PWC-07	High, Small, Many	Validate high tier + max toppings	High → Small → 6 toppings	Premium + topping heavy total correct
TC-PWC-08	High, Medium, None	Validate high plain default pizza	High → Medium → 0 toppings	Scaled premium base correct
TC-PWC-09	High, Large, Few	Validate high large lightly customized pizza	High → Large → 2 toppings	Premium large cost correct

5.2.1.4 Test Results - Evidence of Test Cases

This section documents the execution results for all PWC test cases. A screenshot and log recorded to verify that the pizza pricing logic behaved as expected across all tested input combinations.

- ***PWC Screenshot***
- **PWC Log** (Testing Console log)

5.2.2 Input Domain Modeling - Base-Case Coverage (BCC)

5.2.2.1 Rationale for using BCC

The Order.GetFormattedTotalPrice() method calculates the final total for an entire order by summing the computed totals of all pizzas in the cart.

While PWC applies to pricing an individual pizza, order-level pricing depends on the structure of the pizza list, not the attributes of a single pizza.

Thus, Base-Choice Coverage (BCC) is the correct strategy because it:

- Chooses a realistic Base Case ("Single pizza order")
- Varies one block at a time
- Ensures efficient coverage across all list behaviors
- Captures edge cases such as:
    - Empty order
    - Uniform pizzas
    - Mixed pizzas

This allows thorough validation of list aggregation, summation logic, handling of duplicates, and mixed-pizza scenarios.

5.2.2.2 Partitioning

To apply Base-Choice Coverage (BCC) at the order level, the structure of the pizza list was divided into meaningful partitions that represent typical and edge-case ordering scenarios. Each block corresponds to a distinct order composition used to validate how Order.GetFormattedTotalPrice() aggregates multiple pizzas. The table below defines these partitions:

Block	Description	Representative Value
Empty	No pizzas	new List<Pizza>()
Single (Base)	One pizza	One Medium pizza with one topping
Multiple	Two or more pizzas	Small empty pizza + Large with two toppings
Uniform	Identical pizzas repeated	Two identical Medium Pepperoni pizzas
Varied	Different sizes, specials, toppings	Mix of Small empty pizza + Large multi-topping

5.2.2.3 BCC Test Cases Table

The following test cases apply Base-Choice Coverage to verify order-level pricing behavior. Each case varies one partition block at a time relative to the base scenario to ensure correct aggregation of pizza totals under different cart configurations.

Test Case ID	Block	Objective	Steps	Expected Result
TC-BCC-01	Empty	Validate behavior with zero pizzas	Create empty list → compute total	Total = $0.00
TC-BCC-02	Single (Base Case)	Validate standard one-pizza order	Add one pizza → compute total	Total = single pizza price
TC-BCC-03	Multiple	Validate pricing with multiple pizzas	Add more than two pizzas → compute	Sum of all pizza subtotals
TC-BCC-04	Uniform	Validate duplicate pizza handling	Add identical pizzas → compute	Total = quantity × unit price
TC-BCC-05	Varied	Validate mixed pizza aggregation	Add different pizzas → compute	Each pizza priced independently; sum correct

5.2.2.4 Test Results - Evidence of Test Cases

This section documents the execution results for all BCC test cases. A screenshot and log recorded to verify that the order-level pricing logic operated correctly across each partition scenario.

- ***BCC Test Results*** (Testing Console log)
- ***BCC Image***

5.2.3 Graph-Based Modeling - All Path Coverage (APC)

5.2.3.1 Rationale

We chose All Path Coverage because the “Place Order” process is straight-forward and linear, with only one main decision point; checking whether the user is logged in or not. This results in just two possible paths through the flow: one for logged-in users and one for users who must log in before continuing. Since the graph doesn’t contain any loops or repeating branches, it’s entirely possible to test every path from start to finish without unnecessary complexity.

Using All Path Coverage ensures that every possible user journey in this scenario is verified, giving us high confidence that both authentication and order handling behaviors work as expected

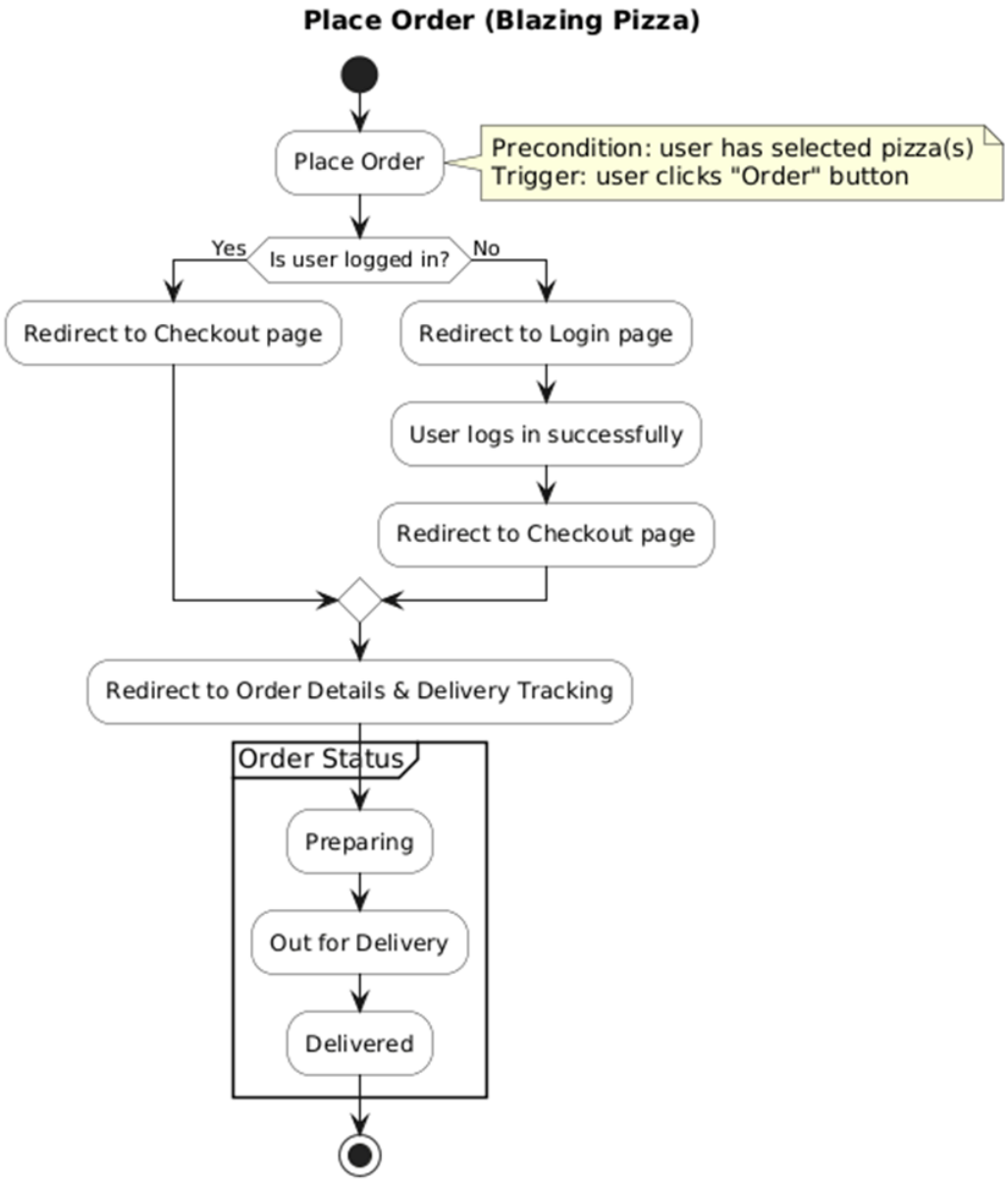
5.2.3.2 Selected Portion of the Application

The portion of the system modeled using Graph-Based Testing is the Place Order workflow, which begins when a user clicks Order and ends when the order reaches the Delivered status. This workflow includes:

- Navigation decisions
- Authentication handling
- Checkout and address submission
- Order creation
- Delivery tracking status transitions

Because this flow contains one primary decision point - *"Is the user logged in?"* - it is ideal for graph-based path analysis.

5.2.3.2 Graph Model Description



The workflow diagram above was represented as a directed graph containing:

Nodes (States):

- **N1:** User clicks *Order*
- **N2:** Authentication check
- **N3:** Login page
- **N4:** Checkout page
- **N5:** Order Details page

- **N6:** Status: Preparing
- **N7:** Status: Out for Delivery
- **N8:** Status: Delivered

Edges (Transitions):

- N1 → N2
- N2 → N4 (User is logged in)
- N2 → N3 (User not logged in)
- N3 → N4 (After successful login)
- N4 → N5 (After placing order)
- N5 → N6 → N7 → N8

This graph contains one decision point and two distinct paths, making it well-suited to full path analysis.

5.2.3.3 Selected Coverage Criterion - All Path Coverage

Criterion: All Path Coverage (APC)

Rationale: The graph for the Place Order workflow contains:

- A single decision point (logged-in vs. not logged-in)
- Two linear, loop-free paths
- Deterministic status transitions

Because the graph is simple and acyclic, **All Path Coverage** is the most appropriate criterion.

It ensures:

- Every possible user journey is tested
- Both authentication branches are validated
- All status transitions (Preparing → Out for Delivery → Delivered) occur as expected

This criterion provides complete confidence that the workflow behaves correctly under both authenticated and unauthenticated conditions.

5.2.3.4 Generated Paths

The following paths were derived from the workflow graph to represent all distinct control-flow sequences in the order-placement process. These paths form the basis for the graph-based test cases executed in this section.

Path A - Logged-In User:

1. User clicks *Order*

2. System checks authentication → Yes
3. Redirect to */checkout*
4. Submit address data
5. Redirect to *Order Details*
6. Status transitions:
    a. Preparing
    b. Out for Delivery
    c. Delivered

Expected Result: Logged-in users proceed directly to checkout without interruption.

Path B - Not Logged-In User:

1. User clicks *Order*
2. System checks authentication → No
3. Redirect to */account/login*
4. User submits valid credentials
5. Redirect to */checkout*
6. Submit address
7. Redirect to *Order Details*
8. Status transitions:
    a. Preparing
    b. Out for Delivery
    c. Delivered

Expected Result: Unauthenticated users must log in before proceeding.

5.2.3.5 Graph Based Test Cases Table

The following test cases were derived from the directed workflow graph using All-Path Coverage. Each case corresponds to a unique path through the order-placement process, ensuring both authenticated and unauthenticated user flows are fully validated.

Test Case ID	Preconditions	Steps	Expected Result
TC-GB-A1	User authenticated	Click **Order**	Navigates to **/checkout**
TC-GB-A2	At **/checkout**	Place order	Navigates to **/myorders/{id}** with Status = Preparing

TC-GB-A3	Status = Preparing	Wait/poll status	Status updates to **Out for Delivery**
TC-GB-A4	Status = Out for Delivery	Wait/poll status	Status updates to **Delivered**
TC-GB-B1	User not authenticated	Click **Order**	Redirects to **/account/login**
TC-GB-B2	On login page	Submit valid credentials	Redirects to **/checkout**
TC-GB-B3	At **/checkout**	Place order	Navigates to **/myorders/{id}**, Status = Preparing
TC-GB-B4	Status = Preparing	Wait/poll status	Status updates to **Out for Delivery**
TC-GB-B5	Status = Out for Delivery	Wait/poll status	Status updates to **Delivered**

5.2.3.6 Code Behavior Summary

This section summarizes how the application behaves along each workflow path represented in the graph. It connects the graph-based test cases to the actual code execution, highlighting how the system handles authentication, navigation, and order-status transitions.

- **TC-A1 → TC-A4** represent the **Logged-In Path**.
- **TC-B1 → TC-B5** represent the **Not-Logged-In Path**.

Each set corresponds directly to the "Yes" and "No" branches from the graph model decision point.

5.2.3.7 Identified Defects and Observations

This section documents defects, inconsistencies, and noteworthy behaviors uncovered during graph-based workflow testing. These findings provide insight into system reliability, redirect logic, and order-status handling across both authenticated and unauthenticated paths.

1. Generic Login Error Message

- Issue: Login failure shows a non-specific “Login failed”.
- Expected: Provide specific messages such as:
  - “Incorrect email or password”
  - “Account not verified”

2. Missing Registration Guidance for Non-Existent Users

- Issue: Unregistered users receive no prompt to register.
- Expected: Prompt: “Account not found. Would you like to register?”

3. No Guest Checkout Option

- Issue: Checkout requires login.
- Expected: Add Continue as Guest support.

4. Unverified Email Not Handled

- Issue: No feedback for unverified emails.
- Expected: Provide clear error: “Email not verified.”

5. Missing Address Validation

- Issue: Invalid addresses accepted during checkout.
- Expected: Validate addresses using USPS/Google Maps API.

5.2.3.8 Test Results - Evidence of Test Cases

The following recordings show how the workflow paths were tested using NUnit and Playwright.

- Logged-In User
- Not Logged-In User

5.2.4 Exploratory Testing

Exploratory Testing was conducted to uncover issues in high-interaction, error-prone areas of the Blazing Pizza application. Four areas were targeted based on risk, user workflow impact, and potential for unexpected behavior:

1. **Pizza Customization**
2. **Orders (High-Volume Ordering)**
3. **Checkout (Form Validation)**
4. **Bypass Login (Authentication & Redirect Behavior)**

Each session was structured using Session-Based Test Management (SBTM) and guided by exploratory test tours such as the Obsessive Compulsive Tour, Test One Get One Free (TOGOF) Tour, Antisocial Tour, and Collector’s Tour.

5.2.4.1 Session 1 - Pizza Customization (Obsessive Compulsive Tour)

- **Charter Table:**

Field	Details
Session	Pizza Customization
Test Tour	Obsessive Compulsive Tour
Object to Be Tested / Rough Guidance	Uncover UI responsiveness and state consistency during rapid, repeated pizza customization interactions. Validate size switching, topping add/remove behavior, duplicate toppings prevention, and UI stability during viewport changes.
JIRA Issues (Text)	**Areas to explore:** • Size selection behavior • Topping add/remove functionality • Duplicate topping prevention • Invalid topping combinations (e.g., vegetarian pizza with meat topping) • Browser resizing and viewport-dependent UI issues **Focus on:** • Price recalculation after each modification • Visual state persistence after size changes and window resizing • Edge interactions such as rapid toggling and invalid combinations
Test Duration (Execution)	15 minutes
Tester	Dana Wilson
Further Testing Opportunities	N/A

- **Execution Table:**

#	What's Done?	Status	Comment
1	Selected pizza	Success	
2	Added multiple toppings	Success	
3	Attempted to add duplicate topping	Success	UI prevented selecting the same topping twice
4	Removed toppings	Success	
5	Added toppings again	Success	
6	Changed pizza size using slider	Success	Toppings and size data were retained
7	Canceled order	Success	User returned to home page
8	Selected pizza again	Success	
9	Changed size (Small → Medium → Large)	Success	Toppings and pizza data remained intact
10	Added toppings	Success	

11	Attempted duplicate topping	Success	UI prevented duplicate selection
12	Placed order	Success	Order was placed with multiple toppings
13	Selected pizza (new test cycle)	Success	
14	Selected Small size	Success	
15	Added toppings	Success	
16	Changed size (Small → Large → Medium → Small)	Success	Toppings persisted through all size changes
17	Added additional topping	Success	
18	Placed order	Success	
19	Selected pizza	Success	
20	Added toppings	Success	
21	Placed order	Success	
22	Selected Mushroom pizza	Success	
23	Added toppings	Success	
24	Placed order	Success	
25	Selected pizza	Success	
26	Added topping	Success	
27	Resized window	Success	UI is not responsive friendly
28	Added topping again	Success	
29	Placed order	Success	
30	Selected pizza	Success	
31	Added one topping	Success	
32	Placed order	Success	
33	Selected vegetarian pizza	Success	
34	Added vegetarian topping	Success	
35	Placed order	Success	
36	Selected vegetarian pizza	Success	
37	Added meat topping	Fail	System incorrectly allowed meat on vegetarian pizza
38	Placed order	Success	System should prevent or warn about invalid topping combinations
39	Submitted order → redirected to login	Success	Expected behavior when not logged in
40	User logged in	Success	Redirected to address form
41	Entered address	Success	
42	Placed order	Success	

- **Key Findings:**
  - UI is not responsive layout breaks when the browser window is resized.
  - State persists correctly toppings and size selections remain consistent through changes.

- Duplicate toppings correctly blocked as the system prevents adding the same topping twice.
- Major defect: vegetarian pizzas can still accept meat toppings with no warning.
- Price updates correctly with every size/topping change.
- Flow remains stable as rapid toggling and repeated actions did not break customization.

- **Test Evidence:**

This video shows the exploratory testing of pizza customization, including size changes, topping interactions, and UI behavior during rapid adjustments

- Pizza Customization

5.2.4.2 Session 2 - Orders (Test One Get One Free Tour - TOGOF)

- **Charter Table:**

Field	Details
Session	Orders – High Volume
Test Tour	Test One Get One Free (TOGOF) Tour
Object to Be Tested / Rough Guidance	Validate system behavior when rapidly creating a large number of orders, focusing on performance, UI stability, and correct data persistence. Identify issues triggered by high-frequency pizza additions and order submission.
JIRA Issues (Text)	**Areas to explore:** • Fast, back-to-back pizza submissions • Different pizza complexities (simple → medium → complex toppings) • Delivery status updates after order placement **Focus on:** • Overlapping submissions (e.g., multiple orders placed quickly) • Data integrity for large orders (55 pizzas) • System performance under load (UI freeze, DB delays)
Test Duration (Execution)	15 minutes
Tester	*Liyan Alkhalidy*
Further Testing Opportunities	N/A

- **Execution Table:**

#	What's done?	Status	Comment

1	*Precondition:* User was already logged in. Placed a pizza order	Success	Precondition satisfied; no login errors.
2	Navigated to Home page	Success	Home loaded and was ready for interaction.
3	Loop-added 55 “Buffalo chicken” pizzas to the order	Success	No crashes, timeouts, or UI freezes while adding items.
4	Clicked “Order >” to proceed to checkout	Success	Correctly redirected to the checkout page.
5	Auto-filled Name, Address, City, Region, Postal Code with valid data	Success	Fields accepted valid data; no validation errors triggered.
6	Clicked “Place order”	Success	Order submitted successfully; page navigated to order details.
7	Verified Order Status = “Preparing”	Success	Status text visible; page rendered correctly.
8	Counted pizzas in order summary and asserted count ≥ 50	Success	Assertion passed; order contained at least 50 pizzas.
9	Observed system responsiveness under load	Success	Application remained stable even with 55 pizzas in one order.

- **Key Findings:**
  - System successfully handled a high-volume load of 55 pizzas without freezing or crashing.
  - Checkout fields accepted input normally, though this test did not focus on field validation.
  - Order status updated correctly (“Preparing”).
  - Performance remained stable, with no visible UI delays despite rapid additions.
  - Order summary correctly reflected the large quantity, confirming proper data persistence.
  - No major defects were observed in this session.

- **Test Evidence:**

This recording captures the high-volume test where 55 pizzas were added to evaluate system stability, responsiveness, and order-summary accuracy

  - High Order Volume

5.2.4.3 Session 3 - Checkout Form (Antisocial Tour)

- **Charter Table:**

Field	Details
Session	Checkout – Form Validation
Test Tour	Antisocial Tour

Object to Be Tested / Rough Guidance	Identify validation weaknesses in the Checkout form by intentionally entering invalid, malformed, or incomplete data. Test how the system responds to incorrect or missing inputs and whether validation prevents order submission.
JIRA Issues (Text)	**Areas to explore:** • Missing required fields • Invalid characters or formats • Empty submissions • Incorrect postal code formats **Focus on:** • Whether proper validation messages appear • Whether invalid orders can still be submitted • Whether the system prevents empty or malformed user data
Test Duration (Execution)	15 minutes
Tester	*Dana Rashad*
Further Testing Opportunities	N/A

- **Execution Table:**

#	What's Done?	Status	Comment
1	*Precondition:* User was already logged in. Placed a pizza order	Success	Redirected to "Place Order"
2	Entered valid address	Success	Order was placed and redirected to Status page
3	Placed another pizza order	Success	Redirected to "Place Order"
4	Pressed submit without entering Name and Address	Success	Required-field messages were displayed
5	Added Name and hit submit	Success	Name error cleared; form did not submit because other fields were required
6	Added invalid Address and hit submit	Fail	Required error removed but invalid data accepted; form did not submit because other fields remained required
7	Added invalid City and hit submit	Fail	City error removed; invalid data accepted; form did not submit because other fields were required
8	Added invalid Region and hit submit	Fail	Region error removed; invalid data accepted; form did not submit because other fields were required

9	Added invalid Postal Code	Fail	Postal error removed; invalid data accepted; form submitted with invalid Postal Code
10	Ordered a pizza	Success	Redirected to "Place Order"
11	Entered 100+ characters in Name field and attempted submission	Success	Max-length message displayed; form did not submit
12	Repeated Name field 100+ character test	Success	Max-length message displayed; form did not submit
13	Added 100+ characters to Address field	Success	Max-length message displayed; form did not submit
14	Added 100+ characters to City field	Success	Max-length message displayed; form did not submit
15	Added 100+ characters to Region field	Success	Max-length message displayed; form did not submit

- **Key Findings:**
  - Required field validation works, but only for detecting empty fields, not incorrect or invalid data.
  - The system accepts invalid formats for Address, City, Region, and Postal Code (e.g., numbers, symbols, incomplete data).
  - A critical defect was found: the form submitted successfully with an invalid Postal Code.
  - Maximum-length validation is implemented correctly; the form blocks submissions when fields exceed 100 characters.
  - Form behavior is inconsistent: the system removes required-field warnings even when the value entered is invalid.
  - Overall, the Checkout page lacks proper input validation, enabling users to submit incorrect delivery information.

- **Test Evidence:**

This video demonstrates checkout validation, including empty fields, invalid inputs, and maximum-length handling during form submission

  - Checkout Test

5.2.4.4 Session 4 - Login Bypass (Collector's Tour)

- **Charter Table:**

Field	Details
Session	Bypass Login
Test Tour	Collector's Tour

Object to Be Tested / Rough Guidance	Test the resilience of the login workflow, including redirects, user lookup failures, registration flow, and whether pre-login cart data persists after authentication.
JIRA Issues (Text)	**Areas to Explore:** • Cart instantiation when user is not logged in • Redirect behavior when logging in with an unknown user • Redirect chaining between Login → Register → Login • Session handoff after successful login **Focus On:** • Data persistence across redirects • Redirect loop prevention • Avoiding security leaks (messages or behaviors that reveal unnecessary system information)
Test Duration (Execution)	15 minutes
Tester	*Dana Rashad*
Further Testing Opportunities	N/A

- **Execution Table:**

#	What's Done?	Status	Comment
1	Placed a pizza order without being logged in	Success	Redirected to login
2	Entered a username not in the system; login failed	Fail	Expected redirect to registration or guest login option
3	Manually navigated to the registration page	Success	Username was not preserved; had to re-enter
4	Registered the username and confirmed the account	Success	Order was not kept
5	Logged in with the new account	Success	Order still not kept
6	Signed out of the newly created account	Success	Redirected to homepage
7	Placed a pizza order while logged out	Success	Redirect back to login again
8	Logged in with valid user on redirect	Success	
9	Entered address and completed order	Success	Order was kept after redirect with valid user

- **Key Findings:**
  - The system does not preserve the user's order when a new account is registered or after login with a newly created user.
  - Login failure for unknown users does not offer helpful options (e.g., Register, Forgot Password, Guest Checkout).

- Username is not persisted when navigating from Login → Register, creating unnecessary friction.
- Authentication redirects work correctly for valid users, and the order is properly retained in that path.
- No redirect loops or security leaks were observed.

- **Test Evidence:**

This recording shows login bypass testing, covering redirect behavior, unknown-user handling, and cart persistence after authentication

- Bypass Login Test

5.2.5 Exploratory Test Case Table

The following table consolidates all exploratory testing sessions, covering UI behavior, customization workflows, validation handling, high-volume ordering, and authentication edge cases. Each test case reflects real-world user interactions captured during structured exploratory tours and session-based testing.

TC ID	Session	Objective	High-Level Steps	Expected Outcome	Actual Outcome / Notes
TC-EXP-01	Session 1 - Pizza Customization	Validate customization workflow (toppings, sizes, UI behavior)	Select pizza → add/remove toppings → switch sizes → resize window → place order	UI remains stable; state preserved; invalid combos blocked	UI not responsive; vegetarian pizza allowed meat topping
TC-EXP-02	Session 1 - Pizza Customization	Validate duplicate & incompatible topping handling	Add duplicate toppings → add meat to vegetarian pizza	Duplicate toppings blocked; invalid combos blocked	Duplicate blocked; vegetarian restriction failed
TC-EXP-03	Session 2 - High Volume Ordering	Validate system under high load (55 pizzas)	Rapidly add 55 pizzas → proceed to checkout → place order	No UI freeze; large orders handled; status updates shown	System stable; performance acceptable; order summary correct

TC-EXP-04	Session 2 - High Volume Ordering	Validate data persistence under heavy load	Add large volume → place order → verify summary count	Order should show correct quantity	Summary correctly displayed ≥50 pizzas
TC-EXP-05	Session 3 - Checkout Validation	Identify weaknesses in form validation	Leave fields empty → enter invalid data → exceed max lengths	Proper validation; invalid data rejected	Empty fields validated; invalid formats accepted
TC-EXP-06	Session 3 - Checkout Validation	Test max-length constraints	Enter 100+ characters in each field	Form blocked with proper warnings	Max-length validation works correctly
TC-EXP-07	Session 4 - Login Bypass	Validate redirect logic for unauthenticated users	Attempt to order while logged out	Redirect to login; cart retained	Redirect correct; cart not retained
TC-EXP-08	Session 4 - Registration / Login Flow	Validate unknown-user handling & registration	Enter unknown user → navigate to registration → log in	Username persisted; order preserved	Username lost; order not preserved
TC-EXP-09	Session 4 - Post-Login Persistence	Validate order persistence after authentication	Create order → log in with valid user → enter address	Order preserved across redirect	Works only with existing accounts

5.3 Traceability Matrix

1. Functional Requirements (FR)

- FR1 - Pizza Customization:
  - FR1.1 - The system shall allow users to select a pizza from the menu.
  - FR1.2 - The system shall allow users to add toppings to a pizza.
  - FR1.3 - The system shall prevent users from adding the same topping more than once.
  - FR1.4 - The system shall update the pizza's price when toppings are added or removed.

  - FR1.5 - The system shall allow users to change the pizza size.
  - FR1.6 - The system shall preserve selected toppings and pizza configuration when the pizza size changes.
  - FR1.7 - *(Defect Requirement)* The system shall **prevent invalid topping combinations**, such as adding meat toppings to vegetarian pizzas.

- FR2 - Pizza Pricing Logic
  - FR2.1 - The system shall calculate pizza total price based on:
    - base special price
    - size multiplier
    - toppings added
  - FR2.2 - The system shall calculate order totals by summing all pizza prices in the cart.

- FR3 - Order Handling
  - FR3.1 - The system shall allow users to add one or more pizzas to an order.
  - FR3.2 - The system shall handle large orders (e.g., ≥ 50 pizzas) without performance failure.
  - FR3.3 - The system shall display a summary of all pizzas in the order.
  - FR3.4 - The system shall initiate order status tracking upon order submission.

- FR4 - Checkout Workflow
  - FR4.1 - The system shall require users to enter:
    - Name
    - Address Line 1
    - City
    - Region
    - Postal Code
  - FR4.2 - The system shall display required-field error messages when fields are empty.
  - FR4.3 - The system shall validate the correctness of user-entered address data.
  - FR4.4 - *(Defect Requirement)* The system shall prevent order submission when invalid address data is entered.
  - FR4.5 - The system shall enforce maximum character length limits for all text fields.

- FR5 - Authentication and Redirection
  - FR5.1 - The system shall require authentication before allowing users to place an order.

  - FR5.2 - If the user is not authenticated, the system shall redirect them to the Login page.
  - FR5.3 - If login fails due to unknown user, the system shall provide a path to registration.
  - FR5.4 - The system shall preserve the user's order when redirecting from Login → Checkout.
  - FR5.5 - *(Defect Requirement)* The system shall preserve cart contents after user registration and login.

- FR6 - Delivery Tracking
  - FR6.1 - After placing an order, the system shall display order status.
    FR6.2 - The system shall update the order status in sequence:
    - Preparing
    - Out for Delivery
    - Delivered

2. Non-Functional Requirements (NFR)

- NFR1 - Performance
  - NFR1.1 - The system shall process large orders (≥ 50 pizzas) without UI freezing.
  - NFR1.2 - The system shall update order status in near–real-time.

- NFR2 - Usability
  - NFR2.1 - The UI shall maintain state across size changes and topping updates.
  - NFR2.2 - The UI shall remain stable during rapid user interactions.
  - NFR2.3 - The system's layout shall be responsive across viewport sizes. *(Defect discovered)*

- NFR3 - Security
  - NFR3.1 - The system shall not disclose unnecessary error information during failed logins.
  - NFR3.2 - The system shall prevent redirect loops between Login, Register, and Checkout.

- NFR4 - Data Integrity
  - NFR4.1 - The system shall ensure all pizzas in the order are persisted correctly.
  - NFR4.2 - The system shall prevent invalid user or address data from being stored.

Traceability Matrix Table:

The traceability matrix table links all functional and non-functional requirements to their corresponding test cases, ensuring full coverage and validating that each requirement has been properly tested. The following link - **Traceability Matrix** - provides access to the complete traceability matrix image for reference.

Section 6: Test Environment

6.1 Multiple Test Environments:

Tests were executed in local development environments using two different system configurations:

Environment A: Windows 11, .NET 8.0, Visual Studio 2022

Environment B: Windows 10, .NET 8.0, VS Code 1.105

Testing across multiple environments helps ensure broader compatibility and reduces environment-specific issues. Windows 11 represents approximately 53-56% of the Windows user base, while Windows 10 accounts for roughly 40-43%, providing confidence that the application behaves consistently for most users.

6.2 Schematic Diagram

The schematic diagram below illustrates the overall test environment setup used throughout this project. It highlights the tools, configurations, and workflow components involved during test execution.
The following link – **Test Schema -** will redirect you to the full image of the test schema.

6.3 Test Architecture Overview

The test strategy for the Blazing Pizza application follows a hybrid approach, combining automated testing with structured manual exploratory sessions. All automated tests are implemented in .NET 8 using the NUnit framework. SQLite is used as the test database, ensuring fast execution with no external dependencies. Playwright serves as the UI automation tool, enabling reliable cross-browser testing and supporting both scripted and exploratory workflows.

- **Automated Components**
  - **Unit & Integration Testing:**
    Focused on pricing logic, API interactions, and data operations using NUnit. Input Domain Modeling techniques were applied, including Pairwise Coverage (PWC) for combinations of size, special, and toppings, and Base-Choice Coverage (BCC) for order-level variations.

- **End-to-End / UI Testing:**
  Playwright was used to automate end-to-end flows such as pizza configuration, login, checkout, and order tracking.

- **Manual Exploratory Testing**
  Session-based exploratory testing was conducted using structured tours such as the Obsessive-Compulsive Tour, Test-One-Get-One-Free Tour, Antisocial Tour, and Collector's Tour.

- **Execution**
  Tests were executed from Visual Studio, VS Code, and the Windows command line using → dotnet test
  This command runs both NUnit and Playwright automated test suites.

- **Artifacts Generated**
  - Screenshots (manual and Playwright-generated)
  - Video recordings (via Playwright)

These artifacts support the verification of test results and help document defects and unexpected UI behaviors.

6.4 Equipment Table

The following table lists the hardware and software configurations used during test execution. Documenting the test equipment helps ensure reproducibility and provides context for performance, compatibility, and environment-specific behaviors observed throughout the testing process.

OS	Windows 11 (Environment A) Windows 10 (Environment B)
CPU	Intel Core i9-14900 4.4GHz (Environment A) Intel Core i5-4690 3.50GHz (Environment B)
RAM	64 GB DDR5-6000 32 GB Dual Channel DDR3 @ 780MHz (Environment B)
Storage	2 TB Crucial CT2000MX500SSD1
Testing Framework	NUnit
Exploratory Testing	Playwright, Manual
UI Testing	Playwright

Section 7: Testing Results

The application performed well, our testing focused on authentication, checkout, and pizza customization. Below is a breakdown of the passing behaviors and failed behaviors.

Passing Behaviors:

- Duplicate toppings are prevented during pizza customizations. (Session 1)
- Pizza state is maintained when making changes (toppings, sizes).
- Checkout requires address in submissions. (Session 3)
- Clear error messages appear when data is missing or exceeding max character fields. (Session 3)
- Users can not order a pizza without having a valid username and password.
- Application keeps a history of completed orders.

Failed Behaviors:

- Generic “Login failed” message for unauthorized users.
- Lacks specificity invalid credentials vs unverified account.
- No navigation to registration page for non-existent user during login attempt.
- No guest checkout option, forcing login for all users.ng

Section 8: Recommendations on Software Quality

After testing the application there are a few recommendations, we have come up with. There is no address validation, at this point the application accepts incorrect address. The address validation would fix issues related to the address check process. Currently invalid (e.g. *,/,+) address characters are allowed. Validation would also enforce minimum length in the address process. Currently you can just put a single character in the application for an address and the application will accept the character as a valid address.

The specialty vegetarian pizza does not prevent or give a warning when you add meat. This is a special selection and the application should at minimum add a warning when you add meat to a vegetarian pizza.

If a user adds pizza to a cart but they are not registered user. When the user registers the account the session data is lost. For a modern application this is a weak spot. There are two recommendations create a guess login to align with modern applications. The other option is allow users register a user name. Both options should keep the session data.