Software Engineering and Testing

Travel Wizard Holiday Package System Software Testing

Table of Contents

[Introduction 3](#_Toc196487074)

[Testing Object 4](#_Toc196487075)

[Testing 5](#_Toc196487076)

[Black Box testing 5](#_Toc196487077)

[Equivalence Partitioning Test for Login Password Validation 5](#_Toc196487078)

[Equivalence Classes: 5](#_Toc196487079)

[Test Cases and Expected Outcomes: 5](#_Toc196487080)

[Equivalence Partitioning Testing Code 6](#_Toc196487081)

[White Box Testing 7](#_Toc196487082)

[Code 7](#_Toc196487083)

[Flow Chart 8](#_Toc196487084)

[Cyclomatic Complexity Calculation 9](#_Toc196487085)

[Basis Set of Independent Paths 9](#_Toc196487086)

[Test Cases for Each Path 10](#_Toc196487087)

# Introduction

In this testing process, we're making sure that every part of the user experience is smooth, intuitive, and responsive. We want users to be able to complete key tasks, like booking a trip, with the least amount of effort possible. Here’s a breakdown of what we’re focusing on:

1. **Minimizing Clicks**: We’re testing how many clicks it takes for a user to go from the homepage to booking confirmation. The goal is to make sure it's quick and straightforward.
2. **Clear UI for Purchases**: We're making sure the "Booking Success" button only shows up once payment is successfully processed, so users aren’t left confused about their purchase status.
3. **Form Input Validation**: We’re testing the input fields to ensure that only valid numbers are accepted, making the experience more user-friendly and secure.
4. **Responsive Layouts**: Whether on mobile or desktop, we’re checking that the purchase form looks great and works seamlessly across all devices.
5. **Error Recovery**: We’re testing how the system handles errors—like invalid login attempts—and ensuring users can recover from mistakes without losing progress, especially if they navigate away from a page and come back.
6. **Login Validation**: For security, we're making sure that passwords meet the required length and include special characters, keeping the login process safe and smooth.
7. **Basic Path Testing**: We’re checking that users can filter trips by price or destination type without any issues, making it easier to find exactly what they’re looking for.
8. **Navigation Flow**: We’re making sure the navigation bar always highlights the current page, and that clicking "Manage Trip" takes users to the correct trip details every time.
9. **Responsive Design**: We’re testing how the layout adjusts when the screen size changes, particularly on mobile, to avoid any awkward display issues like hidden or overflowing elements.
10. **Boundary Testing**: We’re ensuring the system behaves logically when users input data that falls within both valid and invalid ranges.

By covering all these areas, we’re making sure that users can have a smooth, seamless experience from start to finish whether they're booking a trip or managing one.

# Testing Object

**Testing Object: User Experience and Interface for Digital Trip Booking System**

The primary focus of our testing is on the **User Interface (UI)** and **User Experience (UX)** of the digital trip booking system. We aim to ensure that users can easily and efficiently navigate through the system from searching for trips to completing a booking. Our testing covers everything from minimizing clicks and validating form inputs to ensuring smooth error recovery and responsive design across devices. The goal is to provide a seamless, intuitive, and secure experience for users at every step of their journey, whether they're booking a trip or managing their itinerary.

This includes verifying:

* **Navigation flow** to ensure users easily find what they need.
* **Input validation** for accurate and secure form submissions.
* **Responsiveness** to make sure the system works flawlessly on mobile, tablet, and desktop.
* **Error recovery** so users don’t get stuck if they make a mistake.

Ultimately, the testing object revolves around creating a frictionless, reliable, and accessible experience for users at all stages of interacting with the system.

# Testing

## Black Box testing

### **Equivalence Partitioning Test for Login Password Validation**

This testing focuses on checking how the system handles different types of login password inputs, especially regarding their length and whether they include special characters.

### **Equivalence Classes:**

* **Valid Passwords**:
  + The password must be **exactly 9 characters long**.
  + It must also **contain at least one special character** (like !, @, #).
* **Invalid Passwords**:
  + Passwords shorter than 9 characters are **invalid**.
  + Passwords of exactly 9 characters but **without any special characters** are also **invalid**.
  + An **empty** password is **invalid**.
  + A **null input** (i.e., no password provided) is **invalid**.

### **Test Cases and Expected Outcomes:**

|  |  |  |
| --- | --- | --- |
| **Test Data** | **Expected Outcome** | **Classes Covered** |
| "abcdefg!" | Pass (T) | Valid length (exactly 9 chars) and includes special character. |
| "abc" | Fail (F) | Less than 9 characters. |
| "abdcdefgh" | Fail (F) | Exactly 9 characters but no special characters. |
| "" | Fail (F) | Empty password. |
| null | Fail (F) | No password input. |
| "abcdefg!" | Pass (T) | Valid length (exactly 9 chars) and includes special character. |
| "abc" | Fail (F) | Less than 9 characters. |

## **Equivalence Partitioning Testing Code**

}A screenshot of a computer code

AI-generated content may be incorrect.

## White Box Testing

## **Code**

A screenshot of a computer program

AI-generated content may be incorrect.

## **Flow Chart**

A diagram of a flowchart

AI-generated content may be incorrect.

**1. Steps in the Flow**

* **Step 1**: Check if $\_GET['max\_price'] is set. If it is, use it; otherwise, set it to null.
* **Step 2**: Check if $\_GET['destination'] is set. If it is, use it; otherwise, set it to null.
* **Step 3**: Loop through $trips and extract the price and locations of each trip.
* **Step 4**: Remove any formatting from the price (i.e., € ).
* **Step 5**: Check if the price is within the max\_price (if provided).
* **Step 6**: Check if the destination is part of the trip's locations (if provided).
* **Step 7**: If both conditions are met, include the trip in the filtered list.

## **Cyclomatic Complexity Calculation**

To calculate the **cyclomatic complexity** (V(G)) of the graph, we use the formula:

V(G) = E - N + 2

Where:

* **E** = Number of edges (connections between nodes)
* **N** = Number of nodes (steps in the flow)
* For this flow:
  + E = 12 edges (considering all the condition checks and branches).
  + N = 7 nodes (steps in the process).

So:

V(G) = 12 - 7 + 2 = 7

This tells us that there are **7 independent paths** through the code that we need to test.

## Basis Set of Independent Paths

Now, we determine the **basis set** of independent paths. Each path should cover a unique route through the flow graph.

**Example of the Basis Set for This Algorithm:**

1. **Path 1**: No filters applied (no max\_price and destination).
2. **Path 2**: Only max\_price is applied.
3. **Path 3**: Only destination is applied.
4. **Path 4**: Both max\_price and destination are applied, and both matched.
5. **Path 5**: Both max\_price and destination are applied, but neither matched.
6. **Path 6**: max\_price applied but no destination.
7. **Path 7**: destination applied but no price.

## **Test Cases for Each Path**

Now, you create test cases for each path. You would simulate different input scenarios that would exercise each independent path.

|  |  |  |  |
| --- | --- | --- | --- |
| Path | max\_price | destination | Expected Outcome |
| 1 | null | null | All trips are returned (no filter) |
| 2 | 100 | null | Only trips with price <= 100 are returned |
| 3 | null | "Paris" | Only trips with "Paris" in locations are returned |
| 4 | 100 | "Paris" | Only trips with price <= 100 and "Paris" in locations are returned |
| 5 | 100 | "London" | No trips are returned (neither condition is met) |
| 6 | 100 | null | Only trips with price <= 100 are returned |
| 7 | null | "London" | Only trips with "London" in locations are returned |