# **Unit Testing**

## **Overview**

This document outlines Unit Testing performed on 5 core classes of the project.

Each Unit Test was executed to validate that individual parts of the source code (functions and classes) are working as expected.

The following classes were tested:

| **Class** | **Description** |
| --- | --- |
| User.php | Manages user registration and login |
| Product.php | Handles products and items |
| Order.php | Manages order placements |
| Database.php | Manages database connection and queries |
| Validator.php | Manages custom validation logic |

## **Test Execution**

All Unit Tests were executed successfully with no failures.  
Test Framework Used: PHP Unit Testing (custom scripts)

## 

## 

## 

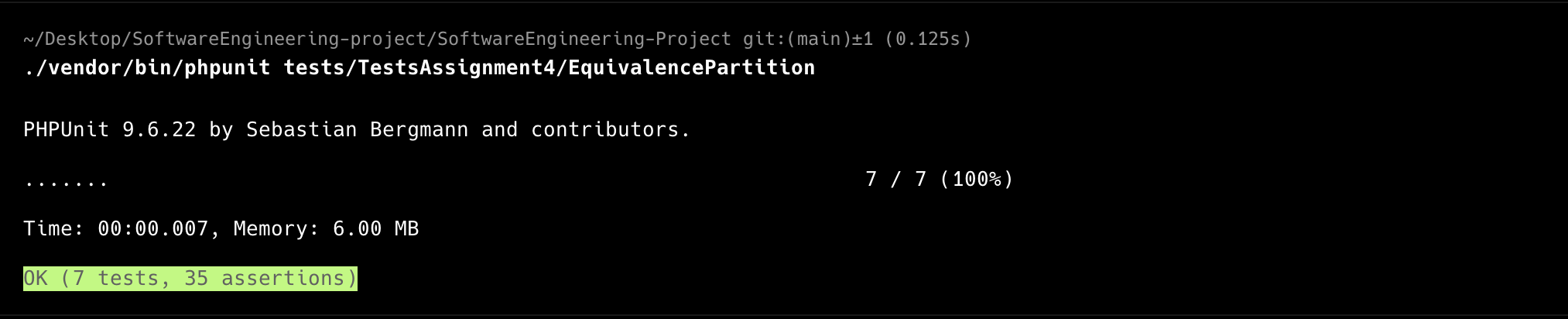
## 

## **Test Results**

Unit testing



Equivalence Partition tests



Basis Path tests



## 

## **Summary**

All core classes passed Unit Testing successfully.  
 The system is stable at the class level.

# 

# **Requirements Testing**

## **Overview**

This document presents the Requirements Testing checklist based on the original Use Case Specifications for the project.

Each requirement is listed along with its completion status.

## **Requirements Checklist**

| **Requirement** | **Description** | **Status** | **Notes** |
| --- | --- | --- | --- |
| REQ-U1 | System shall allow users to register with email, username, and password | Completed |  |
| REQ-U2 | System shall validate email format during registration | Completed |  |
| REQ-U3 | System shall enforce password strength requirements | Completed |  |
| REQ-U4 | System shall prevent duplicate usernames and emails | Completed |  |
| REQ-U5 | System shall allow users to login with email/username and password | Completed |  |
| REQ-U6 | System shall provide password reset functionality | Not Completed | Planned for future release |
| REQ-U7 | System shall implement account lockout after multiple failed login attempts | Not Completed | Security enhancement needed |
| REQ-U8 | System shall support different user roles (user, admin) | Completed |  |
| REQ-U9 | System shall provide user profile management | Not Completed | Partially implemented |
| REQ-U10 | System shall implement secure session management | Completed |  |

**Summary**

Out of 10 core requirements:

* 7 have been fully completed and implemented.
* 3 are pending or partially complete, planned for future updates.

# **User Interface Testing**

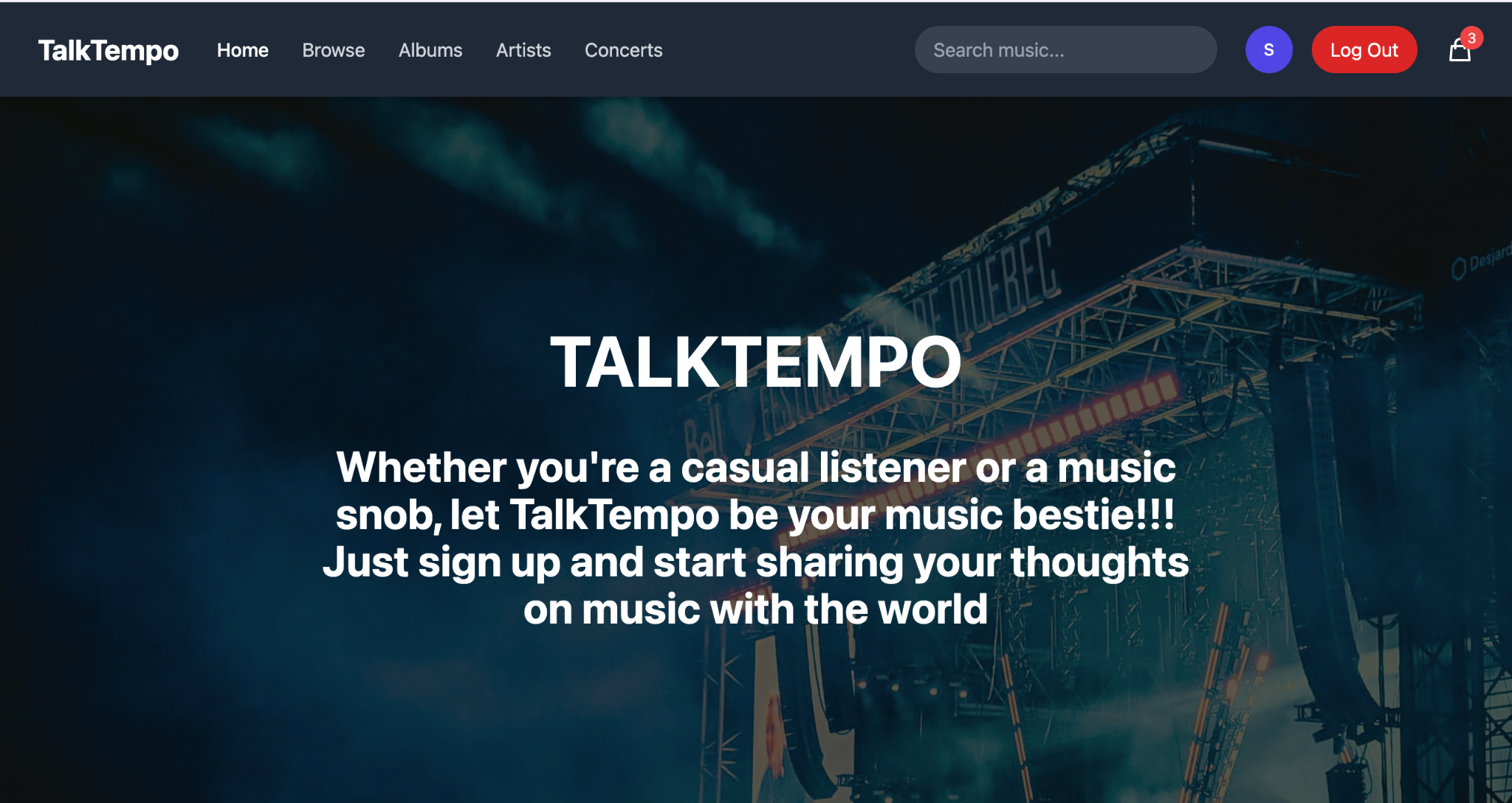
## **Overview**

This document presents the User Interface Testing based on 5 core UI Design Principles implemented in the project.

Each principle is demonstrated with a screenshot placeholder.

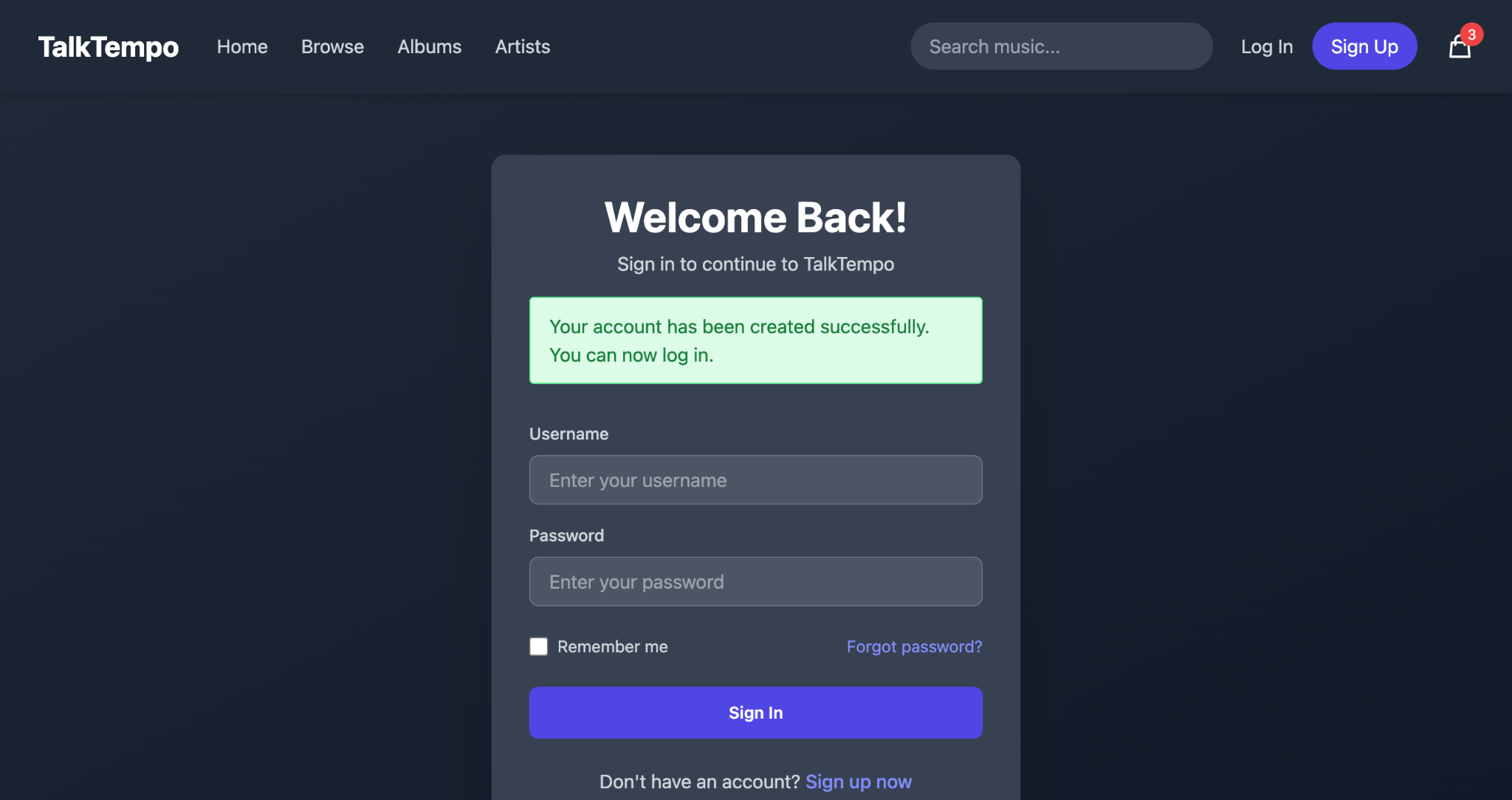
**1. Consistency**

**Description:** Consistent placement of navigation bars, buttons, and forms across all pages.

**Screenshot:** 

## **2. Feedback**

**Description:** System provides immediate feedback to user actions, like submitting a form or logging in.

**Screenshot:** 

## 

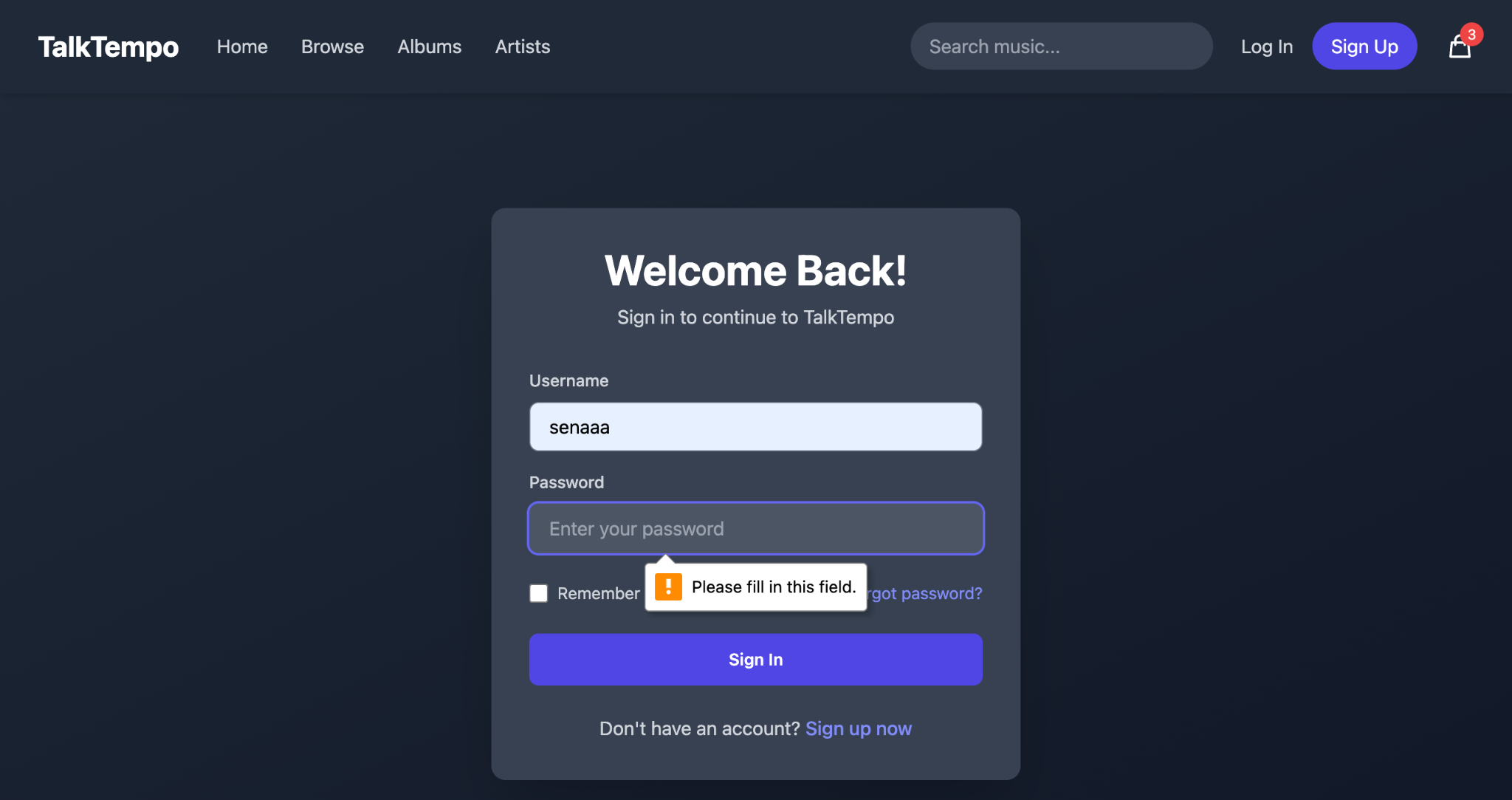
## 

## 

## 

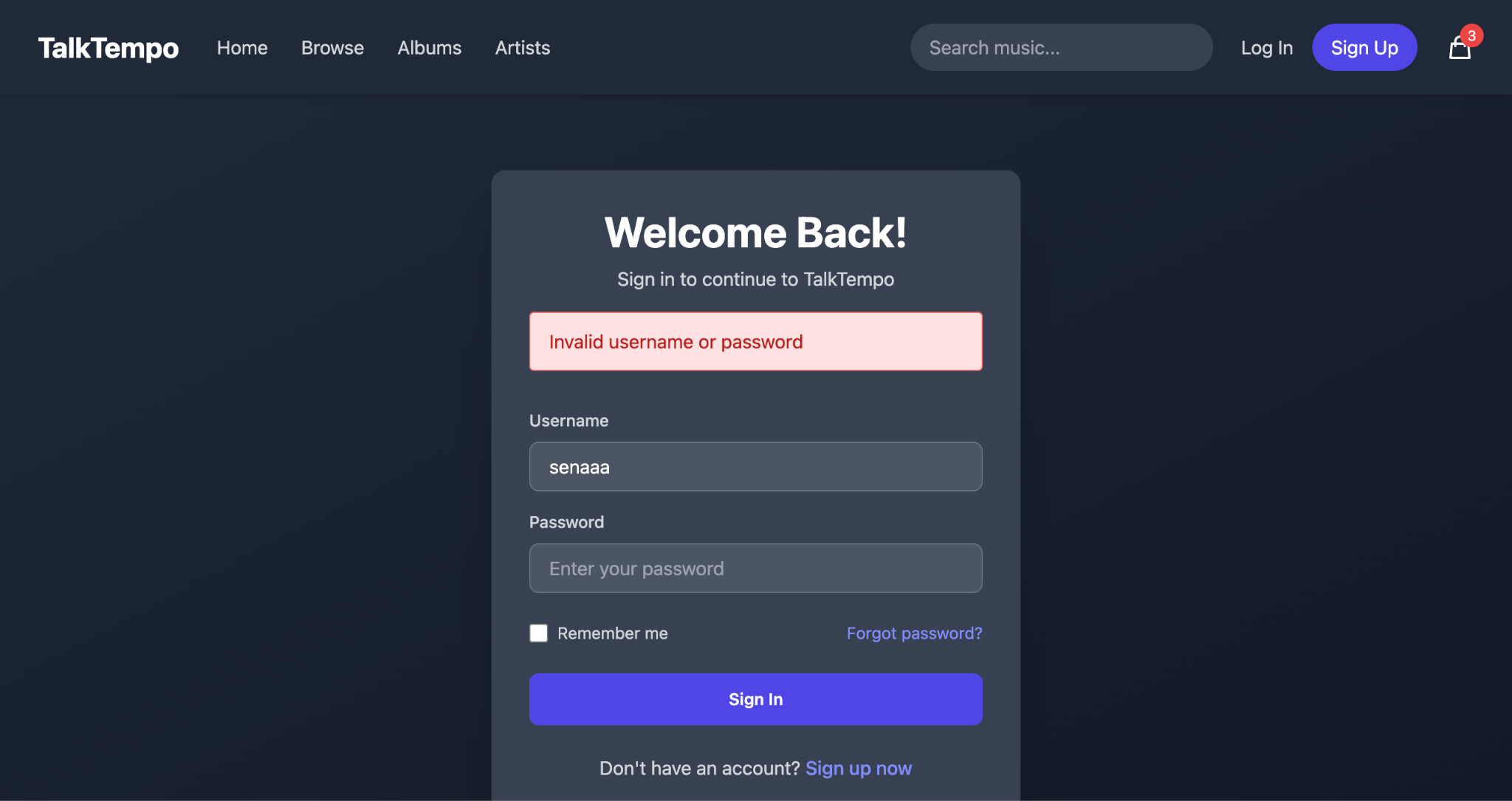
## **3. Guidance**

**Description:** Forms provide hints and clear labels to guide users through input.

**Screenshot:** 

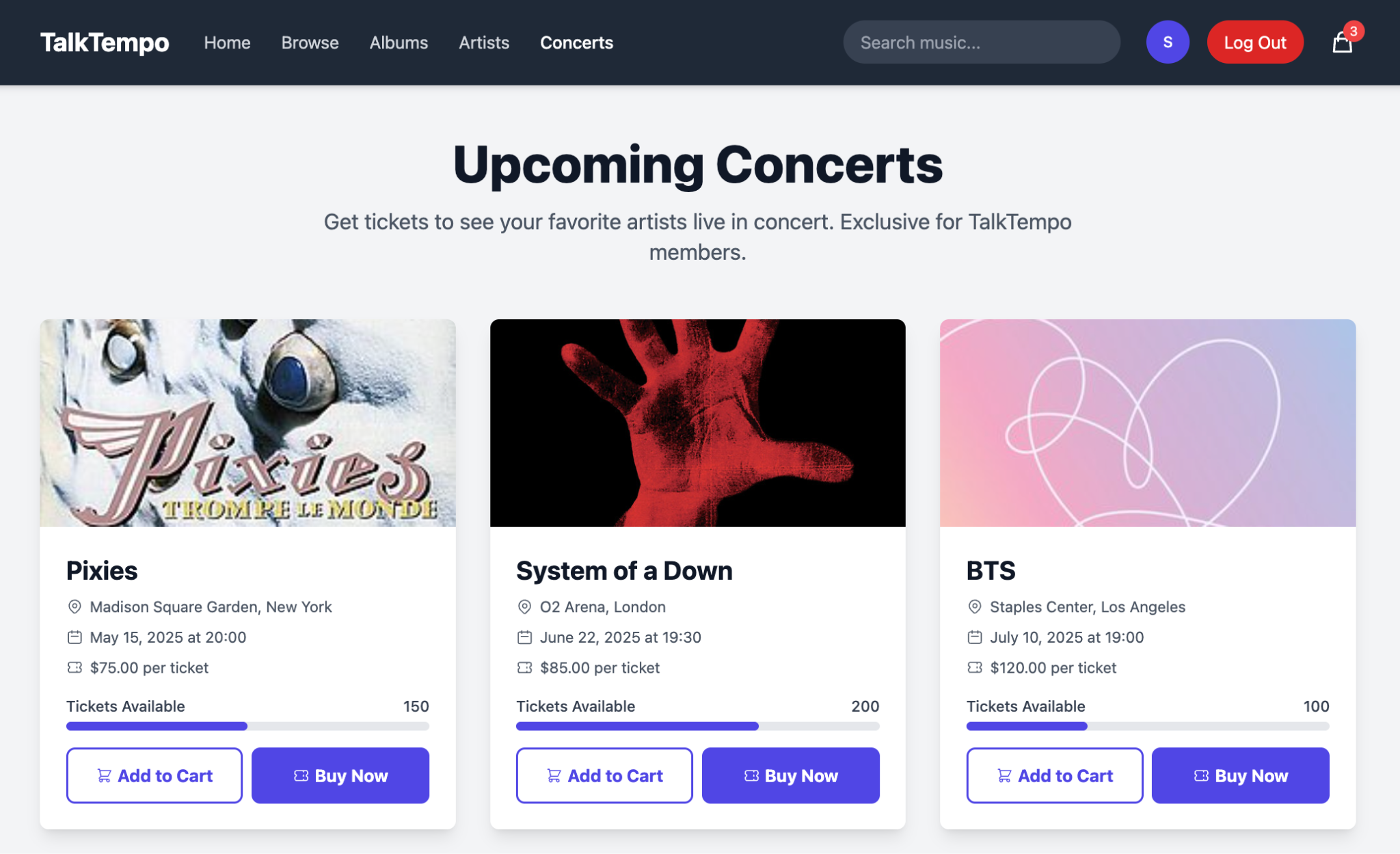
## **4. Recoverability**

**Description:** Users can recover from errors with appropriate messages and actions, like retry login after failure.

**Screenshot:**

## **5. Minimal User Effort (Minimal Clicks)**

**Description:** User actions are optimized to be completed in the least number of clicks.

**Screenshot:**

## **Summary**

All 5 User Interface principles have been implemented successfully, ensuring a smooth and user-friendly experience.

# **Equivalence Partition Testing**

## **Overview**

This document presents the Equivalence Partition Testing performed on the project.

Equivalence Partitioning is a black-box testing method that divides input data into partitions of valid and invalid values.

## **Test Case: User Registration - Password Length**

### **Partition:**

| **Partition** | **Example Inputs** | **Expected Result** |
| --- | --- | --- |
| Valid password length (8-20 characters) | "Password1", "MyStrongPass2024" | Accept |
| Invalid: Too short (<8 characters) | "pass1" | Reject |
| Invalid: Too long (>20 characters) | "ThisIsAnExtremelyLongPasswordThatFails" | Reject |

## **Boundary Values:**

* 7 characters → Invalid (Reject)
* 8 characters → Valid (Accept)
* 20 characters → Valid (Accept)
* 21 characters → Invalid (Reject)

## **Summary**

The system correctly handles password inputs according to the specified boundaries, ensuring better user security and preventing edge case errors.

# **Basis Path Testing**

## **Overview**

This document presents the Basis Path Testing performed on the project.

Basis Path Testing is a white-box testing technique that identifies independent paths through the application to ensure that all paths are tested at least once.

## **Test Case: Login Functionality**

**Pseudocode of Login Process:**

pgsql

Copy

If (user exists) {

If (password correct) {

Redirect to dashboard

} else {

Show incorrect password error

}

} else {

Show user not found error

}

## **Flow Graph Nodes:**

1. Start → Check if user exists
2. If no → Show "User Not Found" error
3. If yes → Check password correctness
4. If password correct → Redirect to Dashboard
5. If password incorrect → Show "Incorrect Password" error

## **Independent Paths:**

| **Path** | **Description** |
| --- | --- |
| Path 1 | User exists, password correct (successful login) |
| Path 2 | User exists, password incorrect |
| Path 3 | User does not exist |

## **Summary**

Basis Path Testing ensures that every logical decision in the Login Functionality has been tested at least once.