MoviePlug Ticketing System

Software Design Specification

Version 2

October 19, 2023

Group 5

Ruben Guillen

Justin Pudiquet

Rosa Lisa Silipino

Prepared for

CS 250- Introduction to Software Systems

Instructor: Gus Hanna, Ph.D.

Fall 2023

# Revision History

| **Date** | **Description** | **Author** | **Comments** |
| --- | --- | --- | --- |
| Oct-5 | Version 1 | Group 5 | Sect. 1.1 - 3.5, excluding 3.4 |
| Oct-19 | Version 2 | Group 5 | Updated UML, added Test Cases, and Verification Test Plan |
|  |  |  |  |
|  |  |  |  |

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

| **Signature** | **Printed Name** | **Title** | **Date** |
| --- | --- | --- | --- |
|  | Group 5 | Software Eng. | Oct 19, 2023 |
|  | Dr. Gus Hanna | Instructor, CS 250 |  |
|  |  |  |  |

# 

# 

# 

# 

# 

# Table of Contents

[**Revision History 2**](#_heading=h.7jfi8blpcpsh)

[**Document Approval 2**](#_heading=h.30j0zll)

[**Table of Contents 3**](#_heading=h.6ox891ondncb)

[**1. System Description 4**](#_heading=h.ymz0g4edge9x)

[**2. Software Architecture Overview 4**](#_heading=h.yvfo8hxbf7et)

[Architectural Diagram 4](#_heading=h.ncpjwlr786l7)

[SWA Description 5](#_heading=h.v91djrl9u0l9)

[UML Class Diagram 6](#_heading=)

[Description of Classes 6](#_heading=h.td4mgwiz1j2n)

[Description of Attributes 7](#_heading=h.naprsqemr2rk)

[Description of Operations 8](#_heading=h.3bk8v3sb100c)

[**3. Development Plan and Timeline 8**](#_heading=h.numi87g456yx)

[Partitioning of Tasks 8](#_heading=h.fiomqzdhxgl4)

[Team Member Responsibilities 9](#_heading=h.bhh4ree31tg9)

[**4. Verification Test Plan 10**](#_heading=h.dzhunj79gmob)

[Introduction 10](#_heading=h.v0fp7oye910l)

[Test Plan Scope 10](#_heading=h.j0hshpji67ha)

[Test Levels 10](#_heading=h.pu4nly864whd)

[1. Unit Testing 10](#_heading=h.xgilpxpv4elj)

[2. Functional Testing 10](#_heading=h.jiyh00vrejmq)

[3. System Testing 10](#_heading=h.rk4f5tka6k79)

[Test Cases 11](#_heading=h.d830vb984aps)

[1. Unit Tests 11](#_heading=h.6ltywqawmmy9)

[● Test Case 1: Verify System’s Ticket Limit for Ticket Purchases 11](#_heading=h.j9lh9yrwzstq)

[● Test Case 2: Verify Booking Timeout 11](#_heading=h.q10kfg9jh9dd)

[● Test Case 3:Verify Senior and Military Discounts 12](#_heading=h.szgytn2xeq2r)

[2. Functional Tests 13](#_heading=h.i0ro917bdpun)

[● Test Case 1: Verify Users Can Reset Password Using Forgot Password 13](#_heading=h.4ojpv5xvlrhe)

[● Test Case 2: Verify User Can Search For Movies 14](#_heading=h.icu4filwq4pl)

[● Test Case 3: Verify the ticket(s) purchased contain the proper theater, movie, seat information selected. 14](#_heading=h.w3ari3wbmevl)

[3. System Tests 15](#_heading=h.hm7qldqh0mgb)

[● Test Case 1: Verify Confirmation Email After Transaction is Processed 15](#_heading=h.qhmw9pwvm0cy)

[● Test Case 2: Verify User Can Login 16](#_heading=h.tsd84cjn31t0)

[**5. Test Case Samples 17**](#_heading=h.nhx4xgwlrrq9)

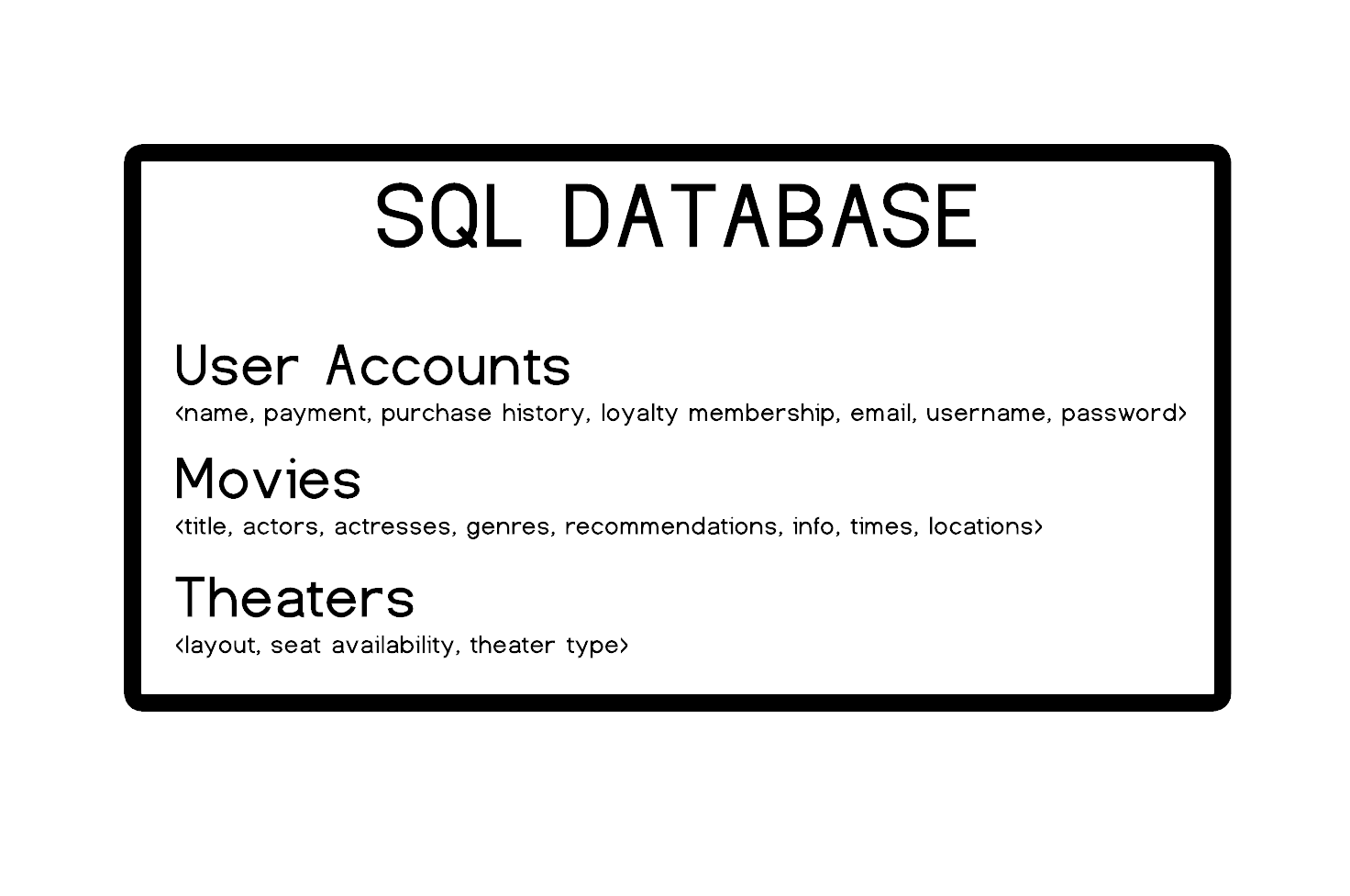
# System Description

The purpose of MoviePlug Ticketing Systemis to ease customers' movie ticket purchasing experience. By providing a convenient online web-based platform that essentially acts as a hub for customers purchasing movie tickets. It has access to 20 partnered theaters across the San Diego region offering access to both regular and deluxe theater seating reservations. The system shall interface with several APIs and a relational database that streamlines processes such as movie searches, ticket purchases, seat reservations, payment, and reviews. Customer accounts are available and loyal customers reap rewards through their loyalty points saved to their account, with their commitment to our product. Overall, the system will provide a user-friendly portal for movie-goers that offers the ability to skip long lines on movie night.

# Software Architecture Overview

## Architectural Diagram

## SWA Description



## UML Class Diagram

## Description of Classes

* *User() - Contains user information and validates information with DB.*
* *Booking() - Contains booking information like booking id, the user who is booking and screenings.*
* *Payment() - Contains user input such as card information and uses 3rd party API to process payment, and transaction information.*
* *Ticket() - Contains all ticket details such as user name, booking confirmation, and assigned seat if applicable.*
* *Seat() - Contains the user designated seat provided that the theater in question is an assigned seating theater.*
* *Theater() - Contains theater branch information such as name, theater id and capacity among other things.*
* *Screening() - Contains information in regards to actual screening date, time, theater location, film name.*
* *Movie() - Contains actual movie information such as title, genre, date released, and movie identifier (internal purposes).*

## Description of Attributes

* User()
  + user\_id - string - user display name
  + username - string user login
  + password - string user password
  + email - string email address
* Booking()
  + booking\_id - string - booking confirmation
  + user - - strings - information coming from the User() class containing user info
  + screening - multiple data types coming from the Screening() class
* Payment()
  + payment\_id - string - payment confirmation id returned from 3rd party API
  + amount - int - total amount being charged for ticket(s)
  + payment\_date - int - date transaction was made in xx/xx/xxxx xx:xx:xx format
* Ticket()
  + ticket\_id - int - ticket id unique to theater, movie, and user
  + booking - multiple data types coming from the booking class
  + seat - string - passed from the Seat() class
* Seat()
  + seat\_id - string - composed from the theater information as well as seat number
  + theater - string/int - information passed from the Theater() class
  + seat\_number - int - seat number within a given row
  + is\_occupied - bool - flag that determines if seat is already taken
* Movie()
  + movie\_id - int - unique ID that corresponds to movie within theater (internal use)
  + title - string - movie title
  + genre - string - movie genre
  + release\_date - int - contains date of when the movie will be released in theaters in xx/xx/xxxx format
* Screening()
  + screening\_id - unique ID that is composed of movie information such as location, movie title, movie times (internal use)
  + movie - string - information brought over from the Movie() class contains movie
  + theater - object - contains theater information
  + date\_time - int - Date and time of the screening
* Theater()
  + theater\_id - int - Unique identifier for the theater (internal use)
  + name - string - Name of the theater
  + capacity - int - Maximum seating capacity of the theater

## Description of Operations

* User()
  + Register() - Registers user from user input information
  + Login() - Using user input information validates credentials with DB information
* Booking()
  + Add\_Ticket() - Adds ticket to the purchase, marks seat as unavailable (if applicable), updates theater capacity, and will queue for printing at a later step.
* Payment()
  + Process\_Payment() - 3rd party API that processes payment

# Development Plan and Timeline

## 

## Partitioning of Tasks

To ensure the four month timeline is met these have been assigned to the following teams:

1. **Planning:**
   1. **Project Manager** - POC: Justine Rosario Rubenero
2. **Research**:
   1. **UX/UI Team** - POC: Shanique Doodlebob
3. **Gather Requirements:**
   1. **Project Manager** - POC: Justine Rosario Rubenero
   2. **UX/UI Team** - POC: Shanique Doodlebob
4. **Design:**
   1. **UX/UI Team** - POC: Shanique Doodlebob
5. **Build:**
   1. **Front-End Development Team** - POC: Terrance Clifford III
   2. **Back-End Development Team** - POC: Chris P. Baycoon
6. **Testing:**
   1. **Front-End Development Team** - POC: Terrance Clifford III
   2. **Back-End Development Team** - POC: Chris P. Baycoon
   3. **QA Team** - POC: Naomi Yerba
   4. **Tester Team** - POC: Wallace Steffingshire
7. **Final Review:** 
   1. **Project Manager** - POC: Justine Rosario Rubenero
   2. **Software Architect** - POC: Davy Jones
   3. **QA Team** - POC: Naomi Yerba
8. **User Testing:**
   1. **UX/UI Team** - POC: Shanique Doodlebob
   2. **QA Team** - POC: Naomi Yerba
   3. **Tester Team** - POC: Wallace Steffingshire
9. **Refining:**
   1. **Software Architect** - POC: Davy Jones
   2. **Front-End Development Team** - POC: Terrance Clifford III
   3. **Back-End Development Team** - POC: Chris P. Baycoon
10. **Launch:**
    1. **Project Manager** - POC: Justine Rosario Rubenero
    2. **Software Architect** - POC: Davy Jones

## Team Member Responsibilities

* **Project Manager**: Justine Rosario Rubenero
  + Develop a project plan and lead product planning meetings with teams.
  + Develop functional specifications.
  + Plan, estimate, and manage resources and the project budget.
  + Conduct initial interviews with stakeholders.
  + Evaluate and ensure timeline processes.
  + Launch and release.
* **UX/UI Team |*****UX/UI Lead***: Shanique Doodlebob
  + Conduct user interviews.
  + Design website wireframe.
  + Research competition.
  + Gather Requirements
* **Front-End Development Team | *FE Lead*:** Terrance Clifford III
  + Build the features laid out in the Sprint.
  + Update the status of the software project to the Project Manager and Lead.
* **Back-End Development Team** | ***BE Lead:*** Chris P. Baycoon
  + Integrate web services and APIs.
  + Build servers, frameworks, databases.
  + Integrate with front-end capabilities.
* **QA Team | *QA Lead:*** Naomi Yerba
  + Evaluate execution of processes and/or production.
  + Identify and document deviations in SOPs.
  + Provide feedback about the results of the quality assurance.
* **Tester Team | *Test Lead:*** Wallace Steffingshire
  + Understand requirements, and constraints to create and run test cases.
  + Build and run test cases to detect bugs and report them.
* **Software Architect:** Davy Jones
  + Define the technical and functional architecture of the overall system.
  + Guide developers in the design and implementation of the solution.
  + Ensure functionality of the final product.

# Verification Test Plan

## Introduction

The purpose of this Verification Test Plan is to provide an in depth overview of functional, unit and system test plans that satisfy business requirements.

## Test Plan Scope

The verification tests will target the following aspects of the software system:

* Unit Testing - Individual class methods and functions.
* Functional Testing - The interaction between different classes and functions.
* System Testing - The end-to-end behavior and performance of the software system.

## Test Levels

### Unit Testing

Unit tests analyze and evaluate at a basic level of testing individual components of MoviePlug Ticketing System software. These components can include objects and classes within the system.

### Functional Testing

Functional testing is a test level that focuses on verifying the functional aspects and features of the software application. The primary objective of functional testing is to ensure that the software behaves as expected, meets its functional requirements, and delivers the intended functionality to end-users.

### System Testing

System testing focuses on assessing the system's behavior and performance as a whole rather than individual components. System testing verifies that all the components work together seamlessly and meet the system's non-functional requirements, such as performance, security, and scalability.

## Test Cases

### Unit Tests

#### Test Case 1: Verify System’s Ticket Limit for Ticket Purchases

* + **Description:** This test will verify that the seating() class function communicates with the event management system (EMS) API to ensure conditions are set to limit the number of premium seats selected on the user's interface. The maximum number of seats a user can reserve is 20 seats utilizing MoviePlug. Upon finalizing movie, theater and showtime selection, the user will be navigated to the seat selection page if a premium theater is chosen. A graphical 2D layout of the theater allows users to select the specific seat(s) they chose. After exceeding the maximum number of seats (20) a popup error message should alert the user that the ability to choose any more seats is unable to be processed. This should mitigate surplus unfair ticket purchases. Possible faults that could occur is that the condition isn’t properly set within the external API or that seating() class doesn’t pull seating information from the database.
  + **Test Input:**
    1. Navigate to the seating selection page.
    2. Select 21 seats on the 2D-seating map.
    3. Click the "Next" button to navigate to the ticket selection page.
  + **Expected Output:** User receives an error message indicating the maximum number of seats have been selected.
  + **Pass/Fail Criteria:  
    Pass:** 
    1. System alerts number of seats selected exceed limit.
    2. Booking() function

**Fail:**

* + 1. System allows users to select over 20 seats.

#### Test Case 2: Verify Booking Timeout

* + **Description:** This test case is designed to verify the behavior of the Booking() class and its transmission to the Payment() class. When a user initiates the movie booking process but does not proceed to payment for an extended period (five minutes). The expected result is that the system should detect the booking timeout condition and display an error timeout pop up prompt.
  + **Test Input:**
  1. Open movie ticketing system application.
  2. Log in as a registered user or use a test account.
  3. Navigate to the “Movie Selection” section.
  4. Select a movie for booking.
  5. Choose the number of tickets and select seats.
  6. Proceed to the payment page.
  7. At this point, pause the test to simulate the user’s inactivity.
  8. Wait for an extended period (e.g. 10 minutes) without taking any further action.
  + **Expected Output:**
    1. After the defined waiting period, the system should detect the user’s inactivity and booking timeout.
    2. The system should display an error timeout page with a clear message informing the user that the booking session has timed out due to inactivity.
    3. The error timeout page may also provide instructions on how to restart the booking process or return to the movie selection page.
  + **Pass/Fail Criteria:**
    1. The test passes if the system correctly identifies booking timeout condition and displays the error timeout page.
    2. The error timeout message should be clear and informative.
    3. The user should be guided on how to proceed or restart the booking process.

#### Test Case 3:Verify Senior and Military Discounts

* + **Description:** The purpose of this test is to verify that Users who have been identified as either a Senior Citizen or a Military Service member or an affiliate receive a discount on the purchase of movie tickets regardless of the movie, theater, or screening selected. This test case verifies that the Payment() class receives the valid information containing the discount and that it is properly applied to the transaction.
  + **Test Input:**
    1. Go through the process of selecting a movie, screening, theater location, seat (if applicable).
    2. Just prior to confirming purchase look at the Cart Summary there should be an inline credit named "Discount" in parentheses for "$xx.xx" dollar amount.
    3. After submitting payment the proper amount is charged to the payment method.
  + **Expected Output:** Cart Summary shows "x" amount of tickets added to cart minus "$xx.xx" dollar amount for Senior/Military Discount. After purchasing, the amount reflects ticket price minus appropriate discount.

* + **Pass/Fail Criteria:**

**Pass:**

* + 1. The test passes if the total amount charged reflects the amount of a ticket or multiple tickets minus the amount of the discount. (i.e. if ticket is $15, and Military/Senior Discount is 10% then the amount charged per ticket for a customer who is discount eligible would be $13.50/ticket)

**Fail:**

* + 1. This test will fail in any event where the total amount charged is anything other than $13.50/ticket for users who are discount eligible.

### Functional Tests

#### Test Case 1: Verify Users Can Reset Password Using Forgot Password

* + - **Description**: This black box test will verify functionality of User() class function , integration with Email API, database updates as well as server communications. Main focus is to ensure the user(s) can properly reset passwords when either locked out or the password is no longer known. Forgot Password button is found on the login page, once clicked it will trigger password recovery/reset transactional email sent by API. User(s) should receive an email containing a link that will navigate them to the SSL secure password reset page and will be prompted to input a new password. Upon successful input of a valid password, the new password should be updated within MoviePlug’s internal database and authentication server and work during the next login. Possible faults could occur at the internal database with failure to update the user's password information. Process is dependent on the Email API to send email to the user for the reset process to occur.
    - **Test** **Input**:
      1. Navigate to the login page.
      2. Click the "Forgot Password" button.
      3. Enter valid email associated with account.
      4. Click the "Reset Password" button.
      5. Receive password reset email.
      6. Redirect to password reset page.
      7. Enter a valid new password and verify.
      8. Click "Confirm" Button.
      9. Return to the login page and try the new password.
    - **Expected** **Output**: User enters a valid email associated with an account to get sent a password reset link to secure SSL encrypted page to reset their password. Upon valid input of a new password that follows password requirements and user confirmation to reset password, the password associated with the account will be changed for login credentials.
    - **Pass/Fail Criteria**:

**Pass:**

1. Password meets security criteria and is accepted by authentication.
2. Database updates and saves the new password.

**Fail:**

* + - 1. User() class failed to retrieve new password from database.
      2. The internal database didn’t store the updated password.
      3. Email API didn’t successfully initiate transactional email.

#### Test Case 2: Verify User Can Search For Movies

* + - **Description**: This test case is designed to verify the functionality of the Movie() class and Screening() class search feature. The purpose is to confirm that when a user searches for a specific movie title, the system correctly displays the corresponding movie listings.
    - **Test Input**:
      1. Open the movie ticketing system application.
      2. Navigate to the “Search” or “Find Movies” section.
      3. In the search bar, enter the title of a specific movie (e.g. “The Avengers”).
      4. Initiate the search by pressing the “Search button” or hitting “Enter”.
    - **Expected Output**:
      1. The system should process the search query and retrieve movie listings matching the entered movie title.
      2. The displayed listings should include movies with titles that contain or match the search query, including exact title matches.
      3. Each movie listing should provide essential details, such as the movie title, showtimes, and theater locations
    - **Pass/Fail Criteria**:
      1. The test passes if the system successfully retrieves and displays movie listings that match the entered movie title.
      2. The displayed listings should accurately correspond to the search query.
      3. The listings should provide relevant movie details.

#### Test Case 3: Verify the ticket(s) purchased contain the proper theater, movie, seat information selected.

* + - **Description**: The purpose of this test is is to ensure that the Booking(), Ticket(), Seat(), Theater(), Movie(), and Screening() classes all function accordingly and that they all pass the proper information over to the Ticket() class which will print out the adequate data as selected and paid for by the user.

* + - **Test Input**:
      1. Search for a movie.
      2. Select a theater.
      3. Select a screening time.
      4. Select number of desired tickets.
      5. If applicable, select a seat(s).
      6. Confirm cart & proceed to finalize purchase.
      7. Confirmation page and ticket(s) will populate and prompt you to print/save.
      8. Confirm that ticket data concurs with all options chosen by the customer.
    - **Expected Output**:Printed/Saved ticket will contain all information as chosen by customer (i.e. customer chose to buy 3 tickets for movie "A" in theater "X" and seats "E8,E9,E10", printed tickets will reflect that information
    - **Pass/Fail Criteria**:

**Pass:**

1. This test case passes if the tickets printed properly reflect the users order (i.e. if User selected movie “X”, at screening time “xx:xx”, in a premium theater “T” and select seat “A1”. Then the ticket should reflect the user name with movie “X” in theater “T” allocated to seat “A1”.

**Fail:**

* + - 1. This test case fails if any component of the ticket does not match the user’s order (i.e. incorrect movie selection, screening time, theater location or seat selection *(if applicable)*).

#### Test Case 4: Verify that the movies are being shown as "available" apply to those under 10 minutes past screening time or at a later date.

* + - **Description**: The purpose of this test is to ensure that the system will check multiple classes throughout the system like the Movie(), and Screening() class to make sure that the system will allow a user to purchase a ticket for a movie that has not been playing for longer than 10 minutes past the screening time.

* + - **Test Input**:
      1. Search for a movie.
      2. Select a theater. (note that some locations will be available while others are grayed out meaning they are not available)
      3. Select a screening time, for those theaters that are marked as available.
    - **Expected Output:** Movie "A" will be available to be seen at theater "X" assuming the movie hasn't been rolling for more than 10 minutes or if it is for any screening at a later time.
    - **Pass/Fail Criteria**:

**Pass:**

1. This test case passes if the application allows the user to purchase a ticket for a screening that is either at a later time or if it has not been more than 10 minutes past the screening time. (i.e. movie starts at 4pm, system should allow user to purchase ticket until 4:10pm)

**Fail:**

* + - 1. This test case fails if the application allows the user to purchase a ticket for a screening that is more than 10 minutes past the screening time.

### System Tests

### Test Case 1: Verify Confirmation Email After Transaction is Processed

* + **Description**: This test will verify system’s communication between internal database storing user contact information, and external email API that delivers transactional email triggered by purchase from MoviePlug website. Main task is to verify Payment() class functions integrated with other class functions. After the user navigates through the payment page and payment is processed between bank and external payment API for preferred payment method, the user will automatically be navigated to the confirmation page. Upon successful payment, corresponding external QR Code and Email APIs will be triggered and users should receive a confirmation email containing the details of their purchase, along with access to their purchased ticket(s). Possible faults can be isolated to external APIs, whether it be payment status not confirmed, QR Code not generated, Email API not receiving corresponding trigger message to initiate email. User error that can occur is an incorrect email address stored in the database.
  + **Test Input**:
  1. Navigate to the payment page.
  2. Pay with preferred payment method.
  3. Confirm payment information
  4. Click "Checkout" Button
  + **Expected Output**: User receives on screen confirmation with text stating confirmation email with tickets information has been sent to email address on file. Upon checking the user checking their personal email inbox, corresponding email from MoviePlug will be delivered confirming purchase details and NFT ticket access.
  + **Pass/Fail Criteria**:

**Pass**:

* + 1. Transactions must be processed.
    2. User(s) must receive a confirmation email sent from transactional email API.

**Fail**:

1. User(s) does not receive the system's automatically generated confirmation email.

#### Test Case 2: Verify User Can Login

* + **Description**: This test case aims to verify the functionality of the software system’s login authentication process, specifically assessing the interactions with the User() class and its associated functions. The test objective is to confirm that when a user logs in with valid credentials, the User() class, in conjunction with the system, correctly authenticates the user and displays the user’s page.
  + **Test** **Input**:
  1. Open the software system application.
  2. Navigate to the “Login” or “Sign In” section.
  3. Enter valid user credentials, which include a username or email and a password.
  4. Initiate the login process by clicking the “Login” or “Submit” button.
  + **Expected** **Output**:
    1. The system should process the login credentials and perform user authentication.
    2. Upon successful authentication, the system should redirect the user to the user’s page, dashboard, or main account page.
    3. The user’s page should display relevant information, user-specific information, or a personalized dashboard.
  + **Pass/Fail Criteria**:
    1. The test passes if the system successfully authenticates the user with valid login credentials.
    2. The system should then direct the user to the user’s page.
    3. The user’s page should display user-specific information and content.
  + **Test Case 3**: **Verify Invalid User Login**
  + **Description**: The test case is designed to assess the behavior of the software system’s login authentication process when a user attempts to log in with incorrect or invalid credentials. The primary objective is to confirm that the system correctly identifies incorrect login attempts and responds by displaying an error page with clear error messages.
  + **Test Input**:
    1. Open the software system application.
    2. Navigate to the “Login” or “Sign In” section.
    3. Enter invalid user credentials, which include a username or email and a password. These credentials intentionally contain errors, such as an incorrect password.
    4. Initiate the login process by clicking the “Login” or “Submit” button.
  + **Expected Output**:
    1. The system should process the login credentials provided by the user.
    2. When incorrect credentials are detected, the system should recognize the login attempt as unsuccessful.
    3. The system should redirect the user to an error page that provides clear error messages, indicating that the login attempt has failed due to incorrect credentials.
    4. The error page may also include instructions on how to recover the account, reset the password, or return to the login page.
  + **Pass/Fail Criteria**:
    1. The test passes if the system correctly identifies incorrect login attempts and redirects the user to the error page.
    2. The error page should clearly communicate the reason for the failed login attempt (incorrect credentials).
    3. The test should ensure that the error message is user-friendly and provides guidance for the user on how to proceed.

# Test Case Samples