Master Test Plan

Overview

1. **Test Plan Identifier** - VRS_TEST_ASSGN6
2. **Authors** – Group 27 (Vishal Ravipati, Monish Natarajan, Rohit Ranjan)
3. **References** –
    a. SRS submitted by Group 27 in of Assignment 5
    b. Project Description
4. **Introduction** - This is the Master Test Plan for the Video Rental System (VRS) submitted by Group 27. This plan will address testing of all items and elements that are related to the VRS, both directly and indirectly. The project will have unit testing, the details for each test are addressed in the appropriate section.
5. **Test Items**
    a. **Customer Class**
        i. Constructor
        ii. addBalance()
        iii. purchaseMovie()
        iv. viewOrders()
        v. returnMovie()
        vi. viewDeadlines()
    b. **Staff Class**
        i. Constructor
        ii. viewDatabase()
        iii. restockMovies()
        iv. viewCustomerPurchases()
    c. **Manager Class**
        i. Constructor
        ii. audit()

iii. editMovieDatabase()
iv. editUserDatabase()
v. editStaffDatabase()

d. **Movies Class**
i. Constructor
ii. displayDetails()
iii. enterFeedback()

e. **Order Class**
i. Constructor
ii. ShowOrderDetails()

6. Features to Be Tested

a. Login Interface
b. Search Interface
c. SignUp Interface
d. Manager Audit Interface
e. Staff Interface
f. Customer Interface

7. Features Not Being Tested

a. All HTML and CSS will NOT be tested
b. All client-side scripts written in JavaScript and Vue.Js would NOT be tested
c. The server-side code written using Flask and Jinja would NOT be tested extensively. A small unit test file will be provided for the basic login.
d. ElasticSearch API for searching will NOT be tested.

8. **Pass/Fail Criteria** - We will be providing golden outputs for all unit tests, which need to be matched for a unit test to be considered as “Passed”, any other result would be treated as “Failed”.

Test Plans and Scenarios

NOTE: Printing of all attributes would use the corresponding get functions, for attributes which do not have a get function, printing would be via accessing the member via the object

Customer Class

1. customer_id → unique
2. username → unique string
3. password → string
4. email → should be a valid email address
5. time_active → should indicate the time since the object has been created
6. balance → should be non-negative

On calling the constructor, the program will check that the email is valid, if not it should raise an exception and print the error. The customer_id should be unique, therefore only one user should be returned on querying the database.

Plan:

1. Call the constructor with proper parameters and print the attributes. Print how many objects exist in the database with the given customer_id/username.

2. Call addBalance() function and check that the change in balance is reflected in the customer object

3. Call purchaseMovie() function with insufficient funds and check whether error is thrown

4. Call purchaseMovie() function with sufficient funds and check whether balance is deducted

5. Call viewDeadlines() function to check whether movie purchased above has its deadline correctly printed

6. Call viewOrders() function to check whether the movie purchased is visible

7. Call returnMovie() function to check whether the movie can be returned

8. Re-call viewDeadlines() and viewOrders() to check that the movie is removed in the first case but not in the second

9. Call the constructor with incorrect/invalid parameters

Staff Class

1. staff_id → should be unique

2. username → should be unique
3. password → string
4. email → should be a valid address
5. time_active → should indicate the time since the creation of object

On calling the constructor, the program will check that the parameters are valid, if not they should raise an exception and print the error.

Plan:

1. Call the constructor with proper parameters and print the attributes. Print the number of objects with the staff

2. Call the updateStock() function to increase the stock of the movies and check whether the balance is updated

3. Call the updateStock() function to reduce the stock more than what is possible

4. Call the viewCustomerPurchases() to check the purchase made by the above customer

5. Call constructor with incorrect parameters

Manager Class

1. Inherits staff class so pass appropriate parameters to super constructor
2. manager_id → should be a unique string
3. username → should be a unique string
4. password → string
5. email → String → should be a valid email address

On calling the constructor, the program will make that the input conforms to the above conditions, if it doesn't then the constructor should raise an exception and print the error.

Plan:

1. Call the constructor with proper parameters and print the attributes. Check the number of managers with the given id.

2. Perform the same tests as staff to check correct inheritance of staff class.

3. Call the editUserDatabase() function for viewing, adding and deleting entries from the user database and checking for updates in the database.

4. Call the editStaffDatabase() function for viewing, adding and deleting entries from the staff database and checking for updates in the database.

5. Call the editMoviesDatabase() function for viewing, adding, editing and deleting movie entries in the movie database and checking for updates in the database.

6. Call the audit() function and check for the proper functioning mentioned in the audit interface.

7. Call the constructor with incorrect parameters

Movies Class

1. movie_id → 4 digit numeric value unique to each movie in the database
2. name → name of the movie
3. description → brief description of the movie plot
4. genre → genre the movie belongs too
5. rating → value between 0.1 to 10
6. price → non-negative numeric value
7. quantity → non-negative integer value

On calling the constructor, the program will check that the parameters are valid, if not they should raise an exception and print the error. Each movie_id should be unique to prevent clashes in the database, other entries must obey the scheme specified above, else an error message is raised.

Plan:

1. Call constructor with proper parameters and print the attributes.

2. Call constructor with incorrect parameters

3. Call displayDetails() function to test out correct display of movie details when searched

Order Class

1. order_id → unique to each order
2. movie_id → unique to each movie, obtained from value entered by user
3. user_id → unique to each user, obtained from id of the calling user_object
4. Total_price → non-negative, obtained from matching
5. start_date → date value, system date while placing order
6. end_date → return days duration over start_date system date
7. return_status → “Yes”/ “No”

Order object is created when a customer exercises the purchaseMovie() function. On calling the constructor, the program will check that the parameters are valid, if not they should raise an exception and print the error.

Plan:

1. Call constructor with proper parameters and print the attributes.

2. Call constructor with non-existent movie id, program should raise “Movie does not exist!” error message

Login Interface

Following are features of the login interface

1. username - unique to each entity (Staff, Manager, Customer)

2. password - value stored after hashing into user database

3. different entity login options - login as Staff, Manager, or as User

4. Register Customer - button to add a user to the database.

A user will be allowed to access the system interface only after a match of username and password details stored in the SQL database, according to the type of the entity. Also provided is an option to register a new customer.

Plan:

1. Select a drop-down list to select entity type (Staff/Customer/User)

2. Logging-in as a user with incorrect details (eg. Using Manager login with customer login details or Manager login with correct username but incorrect password, else login with incorrect username)

3. Call Register() option to create a new user.

4. Call Register() option to register Staff/Manager with incorrect System Password

Search Movies Interface

Following are features of the login interface

1. Text box to search movie by name/genre/id

2. Movie details, display description, pricing, and other movie details after selection

Plan

1. Call SearchMovie() to search movies by name, genre, index

2. Call SearchMovie() with incorrect details

Signup Interface

The sign-up interface is displayed when the user selects a Register() function. Registering a Staff/Manager requires an additional System Password to be entered for authentication.

The following scenarios are checked for while considering the signup functionality:

1. Signup of new user with a new email ID in some category (Buyer, Manager, Staff)
2. Signup of new user with the same email ID in some different category in which the user has not registered before.
3. Signup of new user with the same email ID in some category the user has already registered for before.
4. Check correctness of System Password while registering Staff and Manager

Customer Interface

The Customer interface is displayed on successful Customer login.

1. ViewAccountDetails(), to view username, password, and email id
2. viewOrder(), view list of all orders
3. searchMovies(), search movies from database
4. addFunds(), add funds to users wallet.

Plan

Refer Customer Class section for detailed plan

Manager Audit Interface

The Manager interface is displayed on successful manager login.

1. displayUsers(), displayStaff() : Managers can view a list of all Staff and Customers.
2. Managers can view the database of all movies: displayMovies()

3. editMoviesDatabase() - to add new movies/ change of existing movie details.

Plan

1. Display of other entities and movies
2. Delete a movie from the database and re-display the updated database
3. Change movie details and re-display movies list
4. Trying to change non-existent movie details

Staff Interface

The Staff interface is displayed on successful Staff login.

1. Staff can view the database of all movies: displayMovies()
2. addStock(), to add stock of particular movie based on movie_id

Plan

1. Call displayMovies(), to view all movies
2. Call addStock(), with correct movie id
3. Call addStock(), with incorrect movie id should display error