Master Test Plan

Overview

1. Test Plan Identifier - OSP_TEST_ASSGN5
2. Authors – Group 22 (Animesh Jha, Nisarg Upadhyaya, Pranav Rajput)
3. References
    a. SRS submitted by Group 22 as a part of Assignment 4
    b. Project Description
    c. Assignment 3 of Software Engineering Theory
4. Introduction - This is the Master Test Plan for the Online Sales Portal (OSP) submitted by Group 22. This plan will address testing of all items and elements that are related to the OSP, 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. Address Class
        i. Constructor
    b. User Class
        i. Constructor
        ii. verifyPassword()
        iii. getInfo()
    c. Manager Class
        i. audit()
        ii. manageBuyer()
        iii. manageSeller()
        iv. removeItem()
        v. negotiation()
        vi. getInfo()
    d. Customer Class
        i. Constructor
    e. Buyer Class
        i. Constructor
        ii. getInfo()
        iii. generateBuyRequest()
    f. Seller Class
        i. Constructor
        ii. uploadItem()
        iii. removeItem()
        iv. modifyRequest()

        v. approvePayment()
        vi. getInfo()
    g. BuyRequest Class
        i. Constructor
        ii. validateRequest()
        iii. modifyStatus()
        iv. negotiation()
    h. Item Class
        i. Constructor
        ii. getInfo()
        iii. search()
    i. Category Class
        i. Constructor
        ii. getType()
        iii. getID()
6. Features to Be Tested
    a. Negotiation Interface
    b. Login Interface
    c. Sign-Up Interface
    d. Audit Interface
    e. Items add/removal Interface
    f. Buy Interface
7. Features Not Being Tested
    a. All HTML, CSS and Bootstrap 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
    d. The database deployed via MongoDB would 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 another 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

Address Class

1. house_no → String → should contain only numbers, dash, slash, commas and letters (non-empty)
2. street → String → same as house_no (can be empty)
3. locality → String → same as house_no
4. city → String → only letters
5. state → String → only letters
6. pin code → String → 6 digit only numbers, 1st digit cannot be 0

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 constructor with proper parameters and print the attributes
2. Call constructor with incorrect parameters

User Class

1. user_id → unique
2. telephone_number → 10 digit, only numbers shouldn't start with 0
3. email → should be a valid email address
4. name → should not be empty

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

Plan:

1. Call constructor with proper parameters and print the attributes. Print how many objects exist in the database with the given user_id.
2. Call constructor with incorrect parameters

Manager Class

1. Inherits User, so should pass proper parameters to super
2. address → should be a valid address
3. gender → nonempty
4. date_of_birth → should be a valid date that can parsed by the python datetime module

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 incorrect parameters
3. Check manage-buyer and seller functionality by attempting to remove existing buyers/sellers and those not present in the system

Customer Class

1. Inherits User, so should pass proper parameters to super
2. city → shouldn't be empty and should contain only letters

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 incorrect parameters

Buyer Class

1. Inherits Customer, so should pass proper parameters to super
2. requests → array of BuyRequest

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. With a Buyer object, call generateBuyRequest with proper parameters and print the status of requests
3. Call constructor with incorrect parameters.

Seller Class

1. Inherits Customer, so should pass proper parameters to super
2. requests → array of BuyRequest

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 uploadItem with proper parameters and print the status of items_list
3. With a Buyer object, call generateBuyRequest with the above Item as a parameter, after this print the Seller objects' requests data member.
4. Call modifyRequestStatus and pass the buyrequest generated in 3, modify it and print the status
5. Call approvePaymentStatus and pass the buyrequest (now modified) and then print the status if the buyrequest
6. Call removeItem with proper parameters and remove the item uploaded in 2, after this print the status of items list.
7. Call constructor with incorrect parameters.

Buy Request Class

1. BuyRequests with status approved cannot be negotiated on
2. The constructor should check that the city of Buyer and the city of Seller should be same, if not it should raise an exception.

Plan

1. Create a BuyRequest R1 and R2 from a Buyer B to Seller S.
2. While calling the constructor for a buy request, the program will check if there are enough items in the database, if not then the constructor would raise an exception.
3. Print the attributes of R1 and R2.
4. B would call function negotiate of R1, should lead to notifications to S and any random Manager. S will then call modifyRequest and modify R1 and print the attributes of R1.
5. B would call function validateRequest of R2, on doing this S would be notified after which it will call approvePaymentStatus and then print the attributes of R2.
6. B would call function negotiate of R2 and this should lead to an exception as approved buyrequests can't be negotiated on.

Category Class

1. name → String → should be non-empty and unique
2. id → integer → should be unique

Plan

1. Call constructor with incorrect parameters, should raise an exception
2. Call constructor with correct parameters and create object C1.
3. Call constructor again with correct parameters and create object C2.
4. Print attributes of C1 and C2, the static member categories_list should now contain both.
5. Check that C1.id != C2.id

6. Try to create a category with same name as some existing category.

Items Class

1. Non empty field → id, name, category, seller, price, company, city, photo, info, is_heavy
2. constructor should check that category provided actually exists
3. Add/remove item interface would simply be a list operation on the items_list static member.

Plan

1. Call constructor with incorrect parameters, should raise an exception.
2. Call constructor with correct parameters to create object I1
3. Call constructor with correct parameters to create object I2.
4. Search for the Items created above and search for an item not in the database.
5. Print attributed of I1 and I2, static member items_list should contain both.

Login Interface

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

1. Checking username
2. Checking password (tests the verifyPassword() of User class)
3. Checking correct login type (i.e., to filter if Buyer is trying to login as Manager and other such cases)

Signup Interface

The validation of information is done in the constructors of User class and its children.

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, Seller)
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.

Negotiation Interface

Covered in the BuyRequest Class plan

Audit Interface

1. Managers can access all details of all items and categories and can remove items and add/remove categories
2. Removal of Category will lead to recategorization of all items of that category to UNK (unknown)
3. Removal of an Item would invalidate all pending BuyRequests of that Item

Plan

1. Delete a category, print all attributes of all items in items_list
2. Delete an item, print all attributes of all items in items_list and all BuyRequests