TEST SUITE

for

Travel in good health management system

Version 1.0 approved

Prepared by

Abhishek Gandhi | 19CS10031
Shashvat Gupta | 19CS30042
Sajal Chhamunya | 19CS10051

Instructors :

Prof. Sourangshu Bhattacharya
Prof. Abir Das
Omprakash Chakraborty

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

18 March 2021

Table of Content

Introduction:

For project TGHM: (Travel in Good Health Management System)

Some cases are tested manually on the GUI while for some we use Python Automation mainly while testing backend and database.

Testing Classes

1. Test case for Account Class

----------ConstructorTesting----------

Given

Name = “Abhishek”

Date = datatime.date(2021,01,27)

Type = AccountType(0)

Password = “hiii_new_user”

Then

check all assigned values like name, id, type by checking each attribute.

----------Method Testing----------

1. getID()

Given

Three object of Account class A1, A2, A3

Then

assert that the IDs of all the objects are unique and the ID of the first constructed object is 1.

2. getName()

Given

Two object of Account class A1, A2

Then

assert that the name returned of both objects is the same as given during construction.

3. getType()

Given

Two object of Account class A1, A2

Then

assert that the type returned of both objects is the same as given during construction.

4. getOpenDate()

Given

Two object of Account class A1, A2

Then

assert that the date returned of both objects is the same as given during construction.

5. changePassword()

Given

Old password

New password

Then

IF oldPassword != __password
    Assert “Invalid old password, please try again”
IF oldPassword == __password
    Assert that password has changed to new password.

2. Test case for Management Class

----------Constructor Testing----------

Given

Get two instance from GetInstance()

Then

Their address must be same as it is a singleton class
And check all assigned values like name, id, type by checking each attribute.

----------Method Testing----------

1. Add_Train()

Given

Add two train object

Then

Assert that the two objects are added in train_list

2. Remove_Train()

Given

Remove any one object

Then

Check Train-list list for required changes

3. Add_Station()

Given

Add two station object

Then

Assert that the two objects are added in station_list

4. Remove_Station()

Given

Remove any one object

Then

Check Station_list list for required changes

5. **Get_Application()**

Given

Create a restaurant add it to application

Then

It must return a list consisting of that restaurant

3. Test case for Station Class

----------Constructor Testing----------

Given

Name = “Kharagpur Station”

Then

check all assigned values like name, id, type by checking each attribute.

----------Method Testing----------

1. **Add_Restaurant()**

Given

Add two restaurants using these functions.

Then

Assert that the two objects are added in __Restaurant.

2. **Get_Restaurant()**

Given

Station class with two added restaurant

Then

Assert that two restaurants must be on the returned list.

4. Test case for Item Class

----------Constructor Testing----------

Given

Name = “fries”
Price = 100

Then

check all assigned values like name and price by checking each attribute.

5. Test case for Menu Class

----------Constructor Testing----------

Given

Create a Menu class

Then

check the condition of __Item list(It must be initialized empty)

----------Method Testing----------

1. Add_Item_by_obj()

Given

Add two Item objects I1, I2

Then

Assert that the two objects are added in __Item list.

2. Add_Item()

Given

S = “cheese”
Value = 5

Then

Check for an Item with given name and value in __Item list.

3. return_price()

Given

S = “cheese”

Then

It must return 5.

4. return_price()

Given

S = “cheese”

Then

assert that the integer returned is the same as the amount given at the time of Adding Cheese.

5. return_menu()

Given

Object of Menu class

Then

assert that the list return must contain all the added Items.

6. Test case for Location Class

----------Constructor Testing----------

Given

X = 50
Y = -100
Landmark = “Near Hospital 1”

Then

check all assigned values like X, Y and Landmark by checking each attribute.

7. Test case for Restaurant Class

----------Constructor Testing----------

Given

Name = “sajal dhaba”
Password = “best_of_best”

Then

check all assigned values like Name, Type, Password by checking each attribute.

----------Method Testing----------

1. Add_Item()

Given

S = “fries”
Value = 50

Then

Assert that a new item is added in the self.Menu object.

2. Add_Order()

Given

An Order Object.

Then

Assert that the added object is in Order_List and the status of that order is pending.

3. Update_Order_status()

Given

An Order Object already present in Order_list and status

Then

If status = Accepted
Order must get an Agent assigned. And that same agent must get an order assigned. And Order.status must change to accepted
If status = declined
Order must get removed from Order_List and Order.status changes to Declined
If status = cooking
Assert that order status must change to cooking
If status = On-way
Assert that order status must change to On-way

4. Add_Agent()

Given

name= “ramu”
Password = “ramu_rocks”

Then

Assert that an agent object is in Agent_List with the given name and password.

5. **Remove_Agent()**

Given

An Agent Object already present in the Agent list.

Then

Assert that the agent given is removed from the agent list.

8. Test case for Agent Class

----------Constructor Testing----------

Given

name= “ramu”
Password = “ramu_rocks”

Then

check all assigned values like Name, Type, Password by checking each attribute.

----------Method Testing----------

1. **update_location()**

Given

X = 30
Y = 20

Then

Assert that values of X and Y are changed in __Location object.

2. **return_location()**

Given

An Agent Object

Then

Assert that it returns a string with x value 30 and y value 20.

3. update_alloted_order()

Given

An Order object

Then

Assert that __ALloted_order is changed to new provided class

4. Update_order_status()

Given

Current status of order

Then

If status = cooking
    Assert that order status must change to cooking
If status = On-way
    Assert that order status must change to On-way
If status = Delivered
    Assert that order status must change to Delivered

9. Test case for Order Class

----------Constructor Testing----------

Given

An customer class, train class and string seat_no

Then

Assert that customer, train and seat_no are assigned.

----------Method Testing----------

1. update_restaurant()

Given

A restaurant class

Then

Check that restaurant is assigned to restaurant

2. update_item()

Given

item

Then

Asset that a new item is added to the item list and price is increased by equivalent amount.

3. remove_item()

Given

Item already present in item list

Then

Assert that the item is removed from the item list.

4. Update_order_status()

Given

Current status of order

Then

If status = Payment_Done

Assert that order status must change to Payment_Done and that order is added into order_list of that selected restaurant

10. Test case for Customer Class

----------Constructor Testing----------

Given

name= “Abhishek Gandhi”
Password = “lets_give_it_our_best”
Train = Train class with name “gitanjali”

Seat_no = “A201”

Then

check all assigned values like Name, Type, Password, Train and seat_no by checking each attribute.

----------Method Testing----------

1. **Add_Order()**

Given

A customer class

Then

Assert that the returned class is an order class with the same customer object.

2. **Remove_Order()**

Given

A order class already present in order list

Then

Remove order from order_list and add it in cancel_order list in restaurant class and order status must change to canceled.

3. **Get_Receipt()**

Given

A order class already present in order list

Then

Returns a string representing order details.

11. Test Cases for Authorization Blueprint

----------Login ----------

GIVEN:

login credentials

URL of login

WHEN:

Management/Customer/Agent tries to login

THEN:

Response should be appropriate with correct status code and RESPONSE HTML

Case 0:

username doesn't pass username validation test
username: abhigandhi29
password:babababababa
Input:
POST METHOD
Response
Assert response.status == 401
Assert "invalid username" in response.data

Case 1:

Login credentials are correct
username: abhigandhi29
password:babababababa
Input: POST METHOD
On submitting
Response:
Assert response.status == 200
Assert "Successfully logged in" in response.HTML

Case 2: Login credentials are wrong

username: abhigandhi29
password:babababababa
Input: POST METHOD with credentials
Response: Assert response.status == 401
Assert "Incorrect password" in response.HTML

Case 3: username doesn't exist

username: abhigandhi29
password:babababababa
Input:
POST METHOD
Response: Assert response.status == 404
Assert "username not registered" in response.HTML

---------- Register----------

GIVEN:

Restaurant details,
Agent details,
Customer details

WHEN:

Restaurant tries to create a new account for the employee,

Restaurant Register,
Customer Register

THEN:

Response should have its status code 200 if user is created
Response should have its status code as 403 if user is already created
Response should have its status code as 404 if username validation fails or any other form validation fails

Feature Testing:

Feature - 1: Database Management System

Testing with Python Automate:

- **Setup:**
    a. Create Mock Stations, Trains, Restaurant with Mock Menu with some Items in it as well as a Mock Agent affiliated to it and a Mock Passenger.
    b. Add all these to the test Database
- **Fetch_And_Display:**
    a. Fetch all created objects in the setup i.e. Station, Train, Restaurant, Menu, Items, Agent, and Passenger.
    b. Use assertEqual and check with the originally created object instances to check if the created objects were stored in the database properly.
        ■ Golden Output: True (for all objects fetched)
    c. In case of a False Output the test case fails.
- **Edit_And_Store:**
    a. Edit Train Route by adding a created Station to the train route.
    b. Assert the added Station to be the same as the station in Train's Route.
        ■ Golden Output: True
    c. Add an item to the Restaurant Menu.
    d. Assert the added Item to be the item in the Menu
        ■ Golden Output: True
    e. Remove the item and assert using finding the item in the Menu.
        ■ Golden Output: False (Not in Menu)
- **Show_Nearby_Stations**
    a. Add another station and restaurant with restaurant options pertaining to delivery to the said Station.
    b. Call function to show nearby restaurants from the Customer by putting its nearest station as the newly added station.
    c. Assert the Restaurant received to be the newly added restaurant.

        - Golden Output: True
    d. Get Menu of the same Restaurant and assert the same with the Menu added to the said Restaurant
        - Golden Output: True
- **Get_Statistics**
    a. Call the function in Management instance to get statistics of all orders/stations/restaurants/Agents
    b. Assert the statistics
        - Golden Output: True
- **Remove**
    a. Remove all stations, restaurants, menus, etc.
    b. Find all the objects one by one and assert if they exist
        - Golden Output: False

Feature - 2: Placing Orders

Testing with Python Automate:

- **Setup:**
    a. Create a Restaurant, Agent affiliated to the same Restaurant, and a Passenger in a Train to a station where the restaurant delivers.
    b. Fetch these from the database to assert their successful creation
- **Issue_Order:**
    a. Create an Order object in the name of the created Passenger Class to the earlier created Restaurant object.
    b. Fetch the order from the restaurant's queued-up orders. Assert the presence of the Order in the Restaurant's queue.
        - Golden Output: True
    c. Also, check the status of the order and assert it.
        - Golden Output: Pending
- **Restaurant_Accepts:**
    a. This is for testing the scenario that the Restaurant accepts the Order. After accepting the order it appoints an Agent to deliver the order.
    b. Accept the order on behalf of the restaurant and assert the Status of the order.
        - Golden Output: Accepted
    c. Appoint the created Agent to the order and assert the Status of the Order.
        - Golden Output: Cooking
    d. The order showed up in the allotted order for the agent. Assert the allotted order of the agent is the same as the Order.
        - Golden Output: True

e. The agent then picks up the order and this changes the status of the Order. Assert the Status of the Order.
    - Golden Output: On-Way
f. The location of the agent is tracked all the time. Assert the location of the agent.
g. On Delivering the Order, the Status of the order changes. Assert the status of the order.
    - Golden Output: Delivered

- **Restaurant_Declines:**
    a. This is for testing the scenario that the Restaurant declines the Order. After declining the order on behalf of the Restaurant assert the status of the Order.
        - Golden Output: Cancelled

Testing using GUI:

- Log in as a Passenger and order from a restaurant.
    - Response: If the restaurant accepts, the order receipt is shown
    - Response: If the restaurant declines, we see an apology message
- Log in as Restaurant:
    - We receive an order request, which can be accepted or declined.
    - If we accept,
        - Response: Choose a Delivery Agent
    - If we reject,
        - Response: No response
- Login as Agent:
    - We receive allotted order, which needs to be picked up
    - After picking up, the status changes to On-Way
    - After delivery, status changes to Delivered

Feature - 3: User Interface

Testing with Python Automate:

- **Setup:**
    a. Create a User account one of each Passenger, Restaurant, and Agent with username, password, and all other necessary data.
    b. Fetch these from the database to assert their successful creation
- **Wrong_UserName_OR_Password:**
    a. This is to test the scenario where a user puts in the wrong Username or password.
    b. Try to login using the wrong username and assert the authentication status.

        - Golden Output: False
    c. Try to login using the wrong password and assert the authentication status.
        - Golden Output: False
    d. Repeat the same with all User type
- **Right_UserName_AND_Password:**
    a. This is to test the scenario where the user puts in the correct username and password.
    b. Try to login using the right username and password and assert the authentication status.
        - Golden Output: True
    c. Repeat the same with all the user types.
- **Forgot_Password:**
    a. This takes in the phone number and sends a validation OTP to the phone number. Validate the OTP. If correct, assert the validation of the OTP.
        - Golden Output: True
    b. Change the account password to a new input password. Assert the new password to be the same as the input string.
        - Golden Output: True

Test Using GUI:

- Create a User of each user type from the respective signup pages:
    - Response: Shows success of account creation
- Try Login by putting in the wrong username:
    - Response: Shows could not log in message
- Try Login by putting the wrong password:
    - Response: Shows could not log in message
- Try login with the correct username and password:
    - Response: Shows Dashboard for the respective account type
- Click the Forgot Password link:
    - Response: Asks for phone Number
    - Response: Asks for OTP
        - Response::Correct OTP: Change Password page
        - Response::Wrong OTP: Wrong OTP Page