Test Document

for

Unified Portal For Hall Automation

Version 1.0

Prepared by

Group #: 1 **Group Name:** The Tech Titans

Soham Amit Bharambe	210264	sohamb21@iitk.ac.in
Divyansh	210355	divyansh21@iitk.ac.in
Divyansh Chhabria	210356	divyanshc21@iitk.ac.in
Jhalak Sharma	210474	jhalak21@iitk.ac.in
Kriti	210534	kriti21@iitk.ac.in
Kumar Harsh Mohan	210543	harshmohan21@iitk.ac.in
Labajyoti Das	210552	labajyoti21@iitk.ac.in
Pranjal Singh	210744	psingh21@iitk.ac.in
Rajeev Kumar	210815	rajeevks21@iitk.ac.in
Sandeep Nitharwal	210921	nsandeep21@iitk.ac.in

Course: CS253

Mentor TA: Mr Ashitosh Vankatrao More

Date: 2 April, 2023

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Soham Amit Bharambe Jhalak Divyansh Chhabria Divyansh Rajeev Kumar Sandeep Nitharwal Labajyoti Das Kumar Harsh Mohan Pranjal Singh Kriti Arora	Initial Version	02/04/2023

1 Introduction

Test Strategy

We have automated the unit testing using Django's test framework. Django provides a built-in test framework. It lets us use the same set of tests during the testing procedure, as we find mistakes and make changes to the views and models. These tests are written in python. The test results are reported systematically.

We used the Selenium library for integration testing. It lets us generate scripts that interact with websites and webpages by identifying elements by their IDs. It is a python library. We wrote scripts to test the functionalities of the portal, hosted on upha.pythonanywhere.com. As before, the scripts can be used repeatedly and help developers find and correct bugs during the testing phase.

System testing was conducted manually on the website. We tried to find exceptions and errors and simulated a large number of scenarios. We tried to check all functional and non-functional requirements that we listed in the SRS.

When was the testing conducted?

We tested the software for basic functions during the implementation phase so as to ensure the software was functional. The majority of the testing was done after the implementation phase. This made it easy to account for exceptional circumstances, as they could mostly be managed with some conditional blocks that do not disturb the software design and call for local changes only.

Who were the testers?

The testers were the same as the developers, due to the constraints on us. However, we ensured that no feature was tested by the person(s) who implemented it, as is standard practice in industry. As a large number of perspectives analysed each feature, we could identify some incorrect assumptions that some of us made as developers. In some cases, upon being shown such errors, the developers also predicted related errors that would appear under exceptional inputs.

What coverage criteria were used?

We used functional coverage for testing.

Have you used any tool for testing?

We used Django's built-in test framework for unit testing and the Selenium library in python for integration testing. System testing was greatly simplified by hosting the portal on pythonanywhere.com, which let us make multiple logins at the same time, from different or the same account depending on which feature we were testing. However, the testing was manual.

2 Unit Testing

1. Login

a. Urls Testing:

This testing will validate the correct url connecting from each path of urls.py to respective functions in the views.py

Unit Details: [
Functions: Login, Set_Password, Reset_Password ,SignUp, OTP, Logout
]
Test Owner: Sandeep Nitharwal
Test Date: [30/03/2023] - [30/03/2023]
Given Test inputs:

```
from django.test import SimpleTestCase
from django.urls import reverse,resolve
from Login.views import
Login,Set_Password,Reset_Password,SignUp,OTP,Logout

class TestUrls(SimpleTestCase):
    def test_Login(self):
        url= reverse('Login')
        self.assertEquals(resolve(url).func,Login)

    def test_Set_Password(self):
        url= reverse('Set_Password')
        self.assertEquals(resolve(url).func,Set_Password)

    def test_Reset_Password(self):
        url= reverse('Reset_Password')
        self.assertEquals(resolve(url).func,Reset_Password)

    def test_SignUp(self):
        url= reverse('SignUp')
        self.assertEquals(resolve(url).func,SignUp)

    def test_OTP(self):
```

```
        url= reverse('OTP')
        self.assertEquals(resolve(url).func,OTP)

    def test_Logout(self):
        url= reverse('Logout')
        self.assertEquals(resolve(url).func,Logout)
```

Test Result:
Found 6 test(s).
System check identified no issues (0 silenced).
.....

Ran 6 tests in 0.008s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

b. Authentication Testing

This test will check where an unauthenticated user is shown error while trying to access any url in Login
Unit Details: [
Functions: Login, Set_Password, Reset_Password ,SignUp, OTP, Logout
]
Test Owner: Sandeep Nitharwal
Test Date: [30/03/2023] - [30/03/2023]
Given Test inputs:

```
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth.models import User
from Login.models import UserManager, User_class
from Login.views import Login, Set_Password, Reset_Password, SignUp,
OTP

class TestViews(TestCase):

    def setUp(self):
```

```
        self.client = Client()

        self.user = User.objects.create_user(
            username = 'testuser',
            password= 'testuser',
            designation = 'Mess Manager',
            name = 'testuser'
        )

    def test_Login(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Set_Password'),
            data={
                'username': 'testuser',
                'password': 'testuser'
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Login.html')

    def test_Set_Password(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Set_Password'),
            data={
                'password1': 'password1',
                'password2': 'password2'
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Set_Password.html')

    def test_Reset_Password(self):
        # Submit a POST request to place an order without logging in
```

```
        response = self.client.post(
            reverse('Reset_Password'),
            data={
                'username': 'testuser'
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Reset_Password.html')

    def test_SignUp(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('SignUp'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'designation': 'Mess Manager'
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'SignUp.html')

    def test_OTP(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('OTP'),
            data={
                'OTP': '123412'
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'OTP.html')
```

Test Results:
Found 12 test(s).
System check identified no issues (0 silenced).
.....

Ran 12 tests in 0.020s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No Comments

c. **Model Object Instantiation Testing:**

This testing will validate creation of objects of each model while instantiating in the authenticated stage of correct designation.

Unit Details: [
Models: user_object, superuser_object]
Test Owner: Sandeep Nitharwal
Test Date: [30/03/2023] - [30/03/2023]
Given Test inputs:

```
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth.models import User
import datetime as datetime
from Login.models import UserManager, User_class

class TestModels(TestCase):

    def test_user_object_is_created(self):
        user_object = UserManager.create_user.create(
            username = 'username',
            name = 'name',
            designation = 'designation',
        )
        #checking if the object is created
        self.assertEqual(user_object.username, 'username')
        self.assertEqual(user_object.name, 'name')
```

```
        self.assertEqual(user_object.designation, 'designation')

    def test_superuser_object_is_created(self):
        superuser_object = UserManager.create_superuser.create(
            username = 'username1',
            name = 'name1',
            designation = 'Hall Manager',
        )
        #checking if the object is created
        self.assertEqual(superuser_object.username, 'username1')
        self.assertEqual(superuser_object.name, 'name1')
        self.assertEqual(superuser_object.designation, 'Hall Manager')
```

Test Results:

Found 14 test(s).
System check identified no issues (0 silenced).
.....

Ran 14 tests in 0.052s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

2. Home

a) Urls Testing:

This testing will validate the correct url connecting from each path of urls.py to respective functions in the views.py

Unit Details: [
Functions: Make_Homepage
]
Test Owner: Sandeep Nitharwal
Test Date: [30/03/2023] - [30/03/2023]
Given Test inputs:

```
from django.test import SimpleTestCase
from django.urls import reverse,resolve
from Home.views import Make_Homepage

class TestUrls(SimpleTestCase):
    def test_Home(self):
        url= reverse('Home')
        self.assertEquals(resolve(url).func,Make_Homepage)
```

Test Result:
Found 1 test(s).
System check identified no issues (0 silenced).
.

Ran 1 test in 0.007s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

3. Booking

a) Urls Testing:

This testing will validate the correct url connecting from each path of urls.py to respective functions in the views.py

Unit Details: [
Functions: Guestroom, Sports_equipments, courts_book, Guestroom_pb, sports_equipments_pb, courts_ub
]
Test Owner: Divyansh
Test Date: [26/03/2023] - [28/03/2023]
Given Test inputs:

```
class TestUrls(SimpleTestCase):
    def test_guestroom_url_is_resolves(self):
        url = reverse('guestroom')
        print(resolve(url))
        self.assertEquals(resolve(url).func,guestroom)


    def test_sports_equipments_url_is_resolves(self):
        url = reverse('sports_equipments')
        print(resolve(url))
        self.assertEquals(resolve(url).func,sports_equipments)


    def test_courts_url_is_resolves(self):
        url = reverse('courts_book')
        print(resolve(url))
        self.assertEquals(resolve(url).func,courts_book)
```

```
 def test_booking_manager_url_is_resolves(self):
    url = reverse('booking_manager')
    print(resolve(url))
    self.assertEquals(resolve(url).func,booking_manager)


def test_secy_request_validation_url_resolves(self):
    url = reverse('secy_request_validation')
    print(resolve(url))
    self.assertEquals(resolve(url).func,
                            secy_request_validation)




def test_secy_return_validation_url_is_resolves(self):
    url = reverse('secy_return_validation')
    print(resolve(url))
    self.assertEquals(resolve(url).func,
                        secy_return_validation)


def test_secy_add_equipment_url_is_resolves(self):
    url = reverse('secy_add_equipment')
    print(resolve(url))
    self.assertEquals(resolve(url).func,secy_add_equipment)


def test_guestroom_pb_url_is_resolves(self):
    url = reverse('guestroom_pb')
    print(resolve(url))
    self.assertEquals(resolve(url).func,guestroom_pb)


def test_courts_ub_url_is_resolves(self):
    url = reverse('courts_ub')
```

```
        print(resolve(url))
        self.assertEquals(resolve(url).func,courts_ub)


    def test_sports_equipments_pb_url_is_resolves(self):
        url = reverse('sports_equipments_pb')
        print(resolve(url))
        self.assertEquals(resolve(url).func,sports_equipments_pb)
```

Test Result:
Found 10 test(s).
System check identified no issues (0 silenced).
.....

Ran 10 tests in 0.012s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: courts_pb was not linked properly which is changed and url resolving is fixed.

b) Authentication Testing:

This test will check where an unauthenticated user is shown error while trying to access any url in Booking
Unit Details: [
Functions: Guestroom, Sports_equipments
]
Test Owner: Divyansh
Test Date: [26/03/2023] - [28/03/2023]
Given Test inputs:

```
class TestViews(TestCase):

    def setUp(self):
        self.client = Client()
        #creating user
        self.user = User_class.objects.create_user(
            username='testuser',
```

```
            password='testpassword',
            designation='Student',
            name = 'testuser',
        )
        #creating guestroom
        self.guestroom_item = Guestroom.objects.create(
            room = '101',
            checkin_date = datetime.date(2020, 1, 1),
            checkout_date = datetime.date(2020, 1, 2),
            price = '100',
            name = 'testuser',
            username = 'testuser',
        )

    #creating authentication in guestroom booking
    def test_guestroom_booking_not_authenticated(self):
            response = self.client.post(
            reverse('Guestroom_Booking'),
            data={
                'room': '101',
                'checkin_date': datetime.date(2020, 1, 1),
                'checkout_date': datetime.date(2020, 1, 2),
               'price': '100',
                'name': 'testuser',
                'username': 'testuser',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Error.html')

    def test_sports_equipment_booking_not_authenticated(self):
            response = self.client.post(
            reverse('Sports_Equipment_Booking'),
            data={
                'name': 'testuser',
                'username': 'testuser',
```

```
                'item': 'CB',
                'quantity': 1,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Error.html')

    def test_court_booking_not_authenticated(self):
            response = self.client.post(
            reverse('Court_Booking'),
            data={
                'name': 'testuser',
                'username': 'testuser',
                'court': 'Badminton',
                'date': datetime.date(2020, 1, 1),
                'time_of_checkin': '10:00',
                'time_of_checkout': '11:00',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Error.html')
```

Test Result:
Found 9 test(s).
System check identified no issues (0 silenced).
.....

Ran 9 tests in 0.048s

OK
Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

c) Model Objects Instantiation Testing:

This testing will validate creation of objects of each model while instantiating in the authenticated stage of correct designation.

Unit Details: [
Models: Guestroom, Sports_equipments_store, Sports_equipments_registered, Sports_equipments_request, Courts
]
Test Owner: Divyansh
Test Date: [27/03/2023] - [29/03/2023]

```
class TestModels(TestCase):

   def Guestroom_object_is_created(self):
       Guestroom_object = Guestroom.objects.create(
           room = '101',
           checkin_date = datetime.date(2020, 1, 1),
           checkout_date = datetime.date(2020, 1, 2),
           price = '100',
           name = 'testuser',
           username = 'testuser',
       )
       #checking if the object is created
       self.assertEqual(Guestroom_object.room, '101')
       self.assertEqual(Guestroom_object.checkin_date,
                                datetime.date(2020, 1, 1))
       self.assertEqual(Guestroom_object.checkout_date,
                                datetime.date(2020, 1, 2))
       self.assertEqual(Guestroom_object.price, '100')
       self.assertEqual(Guestroom_object.name, 'testuser')
       self.assertEqual(Guestroom_object.username, 'testuser')

   def sports_equipments_registered_object_is_created(self):
       sports_equipments_registered_object =
          sports_equipments_registered.objects.create(
```

```
            item = 'CB',
            quantity = 1,
        )
        #checking if the object is created
        self.assertEqual(sports_equipments_registered_object.item,
                                                    'CB')
        self.assertEqual(sports_equipments_registered_object.quantity,
1)

    def sports_equipments_request_object_is_created(self):
        sports_equipments_request_object =
          sports_equipments_request.objects.create(
            name = 'testuser',
            item = 'CB',
            quantity = 2,
        )
        #checking if the object is created
        self.assertEqual(sports_equipments_request_object.name,
                                                        'testuser')
        self.assertEqual(sports_equipments_request_object.item, 'CB')
        self.assertEqual(sports_equipments_request_object.quantity, 2)

    def sports_equipmnents_store_object_is_created(self):
        sports_equipmnents_store_object =
        sports_equipmnents_store.objects.create(
            item = 'CB',
            quantity = 1,
        )
        #checking if the object is created
        self.assertEqual(sports_equipmnents_store_object.item, 'CB')
        self.assertEqual(sports_equipmnents_store_object.quantity, 1)

    def Courts_object_is_created(self):
        courts_object = Courts.objects.create(
            name = 'testuser',
            sports = 'Cricket',
            date = datetime.date(2020, 1, 1),
```

```
            time_of_start = datetime.time(10, 00),
            time_of_end = datetime.time(11, 00),
        )
        #checking if the object is created
        self.assertEqual(courts_object.name, 'testuser')
        self.assertEqual(courts_object.sports, 'Cricket')
        self.assertEqual(courts_object.date, datetime.date(2020, 1, 1))
        self.assertEqual(courts_object.time_of_start,
                                    datetime.time(10, 00))
        self.assertEqual(courts_object.time_of_end,
                                    datetime.time(11, 00))
```

Test Result:
Found 14 test(s).
System check identified no issues (0 silenced).
.....

Ran 14 tests in 0.087s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

4. Mess

a. Urls Testing:

This testing will validate the correct url connecting from each path of urls.py to respective functions in the views.py

Unit Details: [
Functions: Student_Regular_Menu, Student_Book_Extras, Student_Booked_Extras, Student_Apply_For_Rebate, Student_Order_History, Manager_Extra_Items, Manager_Modify_Menu, Manager_View_Feedback, Manager_Rebate_Requests, Manager_Students_Bills, Manager_View_Orders, Manager_Modify_BDMR]
Test Owner: Divyansh
Test Date: [25/03/2023] - [27/03/2023]
Given Test inputs:

```
class TestUrls(SimpleTestCase):

  def test_Student_Regular_Menu_url_is_resolves(self):
      url = reverse('Student_Regular_Menu')
      print(resolve(url))
      self.assertEquals(resolve(url).func,Student_Regular_Menu)

  def test_Student_Book_Extras_url_is_resolves(self):
      url = reverse('Student_Book_Extras')
      print(resolve(url))
      self.assertEquals(resolve(url).func,Student_Book_Extras)

  def test_Student_Booked_Extras_url_is_resolves(self):
      url = reverse('Student_Booked_Extras')
      print(resolve(url))
      self.assertEquals(resolve(url).func,Student_Booked_Extras)

  def test_Student_Apply_For_Rebate_url_is_resolves(self):
      url = reverse('Student_Apply_For_Rebate')
      print(resolve(url))
      self.assertEquals(resolve(url).func,Student_Apply_For_Rebate)
```

```
    def test_Student_Order_History_url_is_resolves(self):
        url = reverse('Student_Order_History')
        print(resolve(url))
        self.assertEquals(resolve(url).func,Student_Order_History)

    def test_Manager_Extra_Items_url_is_resolves(self):
        url = reverse('Manager_Extra_Items')
        print(resolve(url))
        self.assertEquals(resolve(url).func,Manager_Extra_Items)

    def test_Manager_Modify_Menu_url_is_resolves(self):
        url = reverse('Manager_Modify_Menu')
        print(resolve(url))
        self.assertEquals(resolve(url).func,Manager_Modify_Menu)

    def test_Manager_View_Feedback_url_is_resolves(self):
        url = reverse('Manager_View_Feedback')
        print(resolve(url))
        self.assertEquals(resolve(url).func,Manager_View_Feedback)

    def test_Manager_Rebate_Requests_url_is_resolves(self):
        url = reverse('Manager_Rebate_Requests')
        print(resolve(url))
        self.assertEquals(resolve(url).func,Manager_Rebate_Requests)

    def test_Manager_Students_Bills_url_is_resolves(self):
        url = reverse('Manager_Students_Bills')
        print(resolve(url))
        self.assertEquals(resolve(url).func,Manager_Students_Bills)

    def test_Manager_View_Orders_url_is_resolves(self):
        url = reverse('Manager_View_Orders')
        print(resolve(url))
        self.assertEquals(resolve(url).func,Manager_View_Orders)

    def test_Manager_Modify_BDMR_url_is_resolves(self):
        url = reverse('Manager_Modify_BDMR')
```

```
        print(resolve(url))
        self.assertEquals(resolve(url).func,Manager_Modify_BDMR)
```

Test Results:
Found 12 test(s).
System check identified no issues (0 silenced).
....

Ran 12 tests in 0.068s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

b. Authentication Testing:

This testing will validate whether the user is authenticated or not while sending a request to the respective function of views

Unit Details: [
Functions: Student_Regular_Menu, Student_Book_Extras, Student_Booked_Extras, Student_Apply_For_Rebate, Student_Order_History, Manager_Extra_Items, Manager_Modify_Menu, Manager_View_Feedback, Manager_Rebate_Requests, Manager_Students_Bills, Manager_View_Orders, Manager_Modify_BDMR]
Test Owner: Divyansh
Test Date: [25/03/2023] - [27/03/2023]
Given Test inputs:

```
class TestViews(TestCase):

    def setUp(self):
        self.client = Client()

        self.user = User.objects.create_user(
            username = 'testuser',
            password= 'testuser',
            designation = 'Mess Manager',
            name = 'testuser',
        )

    def test_student_regular_menu_is_authenticated(self):
```

```
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Student_Regular_Menu'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'Day': 'Monday',
                'Breakfast': 'Bread',
                'Lunch': 'Rice',
                'Dinner': 'Dal',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Student_Regular_Menu.html')

    def student_book_extras_is_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Student_Book_Extras'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'quantity': 1,
                'identity': '24',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Student_Book_Extras.html')

    def student_apply_for_rebate(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Student_Apply_For_Rebate'),
            data={
                'username': 'testuser',
```

```
                'name': 'testuser',
                'from': '2020-01-01',
                'to': '2020-01-02',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response,
'Student_Apply_For_Rebate.html')

    def manager_modify_menu_is_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Manager_Modify_Menu'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'Day': 'Monday',
                'meal': 'Breakfast',
                'item': 'Bread',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Manager_Modify_Menu.html')


    def manager_extra_item_is_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Manager_Extra_Item'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'meal_dated': '2020-01-01',
                'meal': 'Breakfast',
                'item': 'Bread',
```

```
                'capacity': 1,
                'price': 100,
                'booking_date': '2020-01-01',
                'start_time': '10:00',
                'end_time': '11:00',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Manager_Extra_Item.html')

    def manager_rebate_request_is_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Manager_Rebate_Request'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'idt': '24',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response,
'Manager_Rebate_Request.html')

    def manager_student_bill_is_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Manager_Student_Bill'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'key': '24',
            }
        )
        # Check that the response status is 200 OK
```

```
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Manager_Student_Bill.html')

    def manager_modify_BDMR_is_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Manager_Modify_BDMR'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'date': '2020-01-01',
                'BDMR': 'Breakfast',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Manager_Modify_BDMR.html')
```

Test Results:
Found 20 test(s).
System check identified no issues (0 silenced).
....

Ran 20 tests in 0.078s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

c. Model Testing:

This will check the correct instantiation of the classes, whether the object is being created or not.

Unit Details: [
Models: Regular_menu, Extras, Orders, Datewise_BDMR, Bill, Rebate, Rating_Regular
]
Test Owner: Divyansh
Test Date: [25/03/2023] - [27/03/2023]
Given Test inputs:

```
class TestModels(TestCase):
  def test_regular_menu_object_is_created(self):
      regular_menu_object = Regular_menu.objects.create(
          Day = 'Mon',
          Meal = 'Breakfast',
          Item = 'Poha',
          Rating = 4.0,
      )
      #checking if the object is created
      self.assertEqual(regular_menu_object.Day, 'Mon')
      self.assertEqual(regular_menu_object.Meal, 'Breakfast')
      self.assertEqual(regular_menu_object.Item, 'Poha')
      self.assertEqual(regular_menu_object.Rating, 4.0)

  def test_extras_object_is_created(self):
      extra_object = Extras.objects.create(
          Meal_Date = '2020-01-01',
          Item_Name = 'Poha',
          Start_time = '10:00',
          End_time = '11:00',
          Capacity = 10,
          Available_Orders = 10,
      )
      #checking if the object is created
```

```
        self.assertEqual(extra_object.Meal_Date, '2020-01-01')
        self.assertEqual(extra_object.Item_Name, 'Poha')
        self.assertEqual(extra_object.Start_time, '10:00')
        self.assertEqual(extra_object.End_time, '11:00')
        self.assertEqual(extra_object.Capacity, 10)
        self.assertEqual(extra_object.Available_Orders, 10)

    def test_orders_object_is_created(self):
        orders_object = Orders.objects.create(
            Order_Date_Time = datetime.datetime.now(),
            Meal_Date = '2020-01-01',
            Item_Name = 'Poha',
            User_name = 'testuser',
            Quantity = 1,
            Price = 15,
            Amount = 15,
            History_Status = False,
        )
        #checking if the object is created
        self.assertEqual(orders_object.Order_Date_Time,
datetime.datetime.now())
        self.assertEqual(orders_object.Meal_Date, '2020-01-01')
        self.assertEqual(orders_object.Item_Name, 'Poha')
        self.assertEqual(orders_object.User_name, 'testuser')
        self.assertEqual(orders_object.Quantity, 1)
        self.assertEqual(orders_object.Price, 15)
        self.assertEqual(orders_object.Amount, 15)
        self.assertEqual(orders_object.History_Status, False)

    def test_datewise_bdmr_object_is_created(self):
        datewise_bdmr_object = Datewise_BDMR.objects.create(
            Date = '2020-01-01',
            BDMR = 100,
        )
        #checking if the object is created
        self.assertEqual(datewise_bdmr_object.Date, '2020-01-01')
        self.assertEqual(datewise_bdmr_object.BDMR, 100)
```

```
    def test_bill_object_is_created(self):
        bill_object = Bill.objects.create(
            Bill_Month = 2,
            Month_Name = 'February',
            Total_Days = 28,
            rebate_days = 0,
            User_name = 'testuser',
            Basic_Amount = 100,
            Extras_Amount = 0,
            Mess_Bill = 100,
        )
        #checking if the object is created
        self.assertEqual(bill_object.Bill_Month, 2)
        self.assertEqual(bill_object.Month_Name, 'February')
        self.assertEqual(bill_object.Total_Days, 28)
        self.assertEqual(bill_object.rebate_days, 0)
        self.assertEqual(bill_object.User_name, 'testuser')
        self.assertEqual(bill_object.Basic_Amount, 100)
        self.assertEqual(bill_object.Extras_Amount, 0)
        self.assertEqual(bill_object.Mess_Bill, 100)

    def test_rebate_object_is_created(self):
        rebate_object = Rebate.objects.create(
            Date_From = '2020-01-01',
            Date_To = '2020-01-31',
            Rebate_Days = 0,
            User_Name = 'testuser',
            Status = 0,
        )
        #checking if the object is created
        self.assertEqual(rebate_object.Date_From, '2020-01-01')
        self.assertEqual(rebate_object.Date_To, '2020-01-31')
        self.assertEqual(rebate_object.Rebate_Days, 0)
        self.assertEqual(rebate_object.User_Name, 'testuser')
        self.assertEqual(rebate_object.Status, 0)
```

```
    def test_rating_regular_object_is_created(self):
        rating_regular_object = Rating_Regular.objects.create(
            Meal = 'Breakfast',
            Day = 'Mon',
            User_Name = 'testuser',
            Rating_Value = 4.0,
        )
        #checking if the object is created
        self.assertEqual(rating_regular_object.Meal, 'Breakfast')
        self.assertEqual(rating_regular_object.Day, 'Mon')
        self.assertEqual(rating_regular_object.User_Name, 'testuser')
        self.assertEqual(rating_regular_object.Rating_Value, 4.0)
```

Test Results:
Found 24 test(s).
System check identified no issues (0 silenced).
....

Ran 24 tests in 0.108s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

5. Cleaning

a. Urls Testing:

This testing will validate the correct url connecting from each path of urls.py to respective functions in the views.py

Unit Details: [
Functions: Pending_Request, Past_Request, Lodge_Request, Cleaning_hall
]
Test Owner: Sandeep Nitharwal
Test Date: [28/03/2023] - [28/03/2023]
Given Test inputs:

```
from django.test import SimpleTestCase
from django.urls import reverse,resolve
from Cleaning.views import
Pending_Request,Past_Request,Lodge_Request,Cleaning_hall

class TestUrls(SimpleTestCase):
    def test_Pending_Request(self):
        url= reverse('Pending_Request')
        self.assertEquals(resolve(url).func,Pending_Request)

    def test_Past_Request(self):
        url= reverse('Past_Request')
        self.assertEquals(resolve(url).func,Past_Request)

    def test_Lodge_Request(self):
        url= reverse('Lodge_Request')
        self.assertEquals(resolve(url).func,Lodge_Request)

    def test_Cleaning_hall(self):
        url= reverse('Cleaning_hall')
        self.assertEquals(resolve(url).func,Cleaning_hall)
```

Test Results:

Found 4 test(s).
System check identified no issues (0 silenced).
....

Ran 4 tests in 0.008s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

b. Authentication Testing:

This test will check where an unauthenticated user is shown error while trying to access any url in Login
Unit Details: [
Functions: Pending_Request, Past_Request, Lodge_Request, Cleaning_hall
]
Test Owner: Sandeep Nitharwal
Test Date: [28/03/2023] - [28/03/2023]
Given Test inputs:

```
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth.models import User
from Login.models import UserManager, User_class
from Cleaning.models import Cleaning_Request
from Cleaning.views import Past_Request, Pending_Request,
Lodge_Request, Cleaning_hall,

class TestViews(TestCase):

    def setUp(self):
        self.client = Client()
        self.user = User.objects.create_user(
            username = 'testuser',
            password= 'testuser',
            designation = 'student',
            name = 'testuser',
        )
```

```
    def test_student_Past_Request_is_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Past_Request'),
            data={
                'username': 'testuser',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Past_Request.html')


    def student_Pending_Request_is_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Pending_Request'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'identity': '28',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Pending_Request.html')


    def student_Lodge_Request(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Lodge_Request'),
            data={
                'username': 'testuser',
                'name': 'testuser',
                'place': 'place',
```

```
                'room': 'D-343',
                'comments': 'Done',
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Lodge_Request.html')


    def test_Cleaning_hall(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Cleaning_hall'),
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'Cleaning_hall.html')
```

Test Results:
Found 9 test(s).
System check identified no issues (0 silenced).
.....

Ran 9 tests in 1.02s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

c. Model Testing:

This will check the correct instantiation of the classes, whether is object is being created or not.

Unit Details: [
Models: Cleaning_Request]
Test Owner: Sandeep Nitharwal
Test Date: [15/03/2023] - [17/03/2023]
Given Test inputs:

```
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth.models import User
import datetime as datetime
from Login.models import UserManager, User_class
from Cleaning.models import Cleaning_Request


class TestModels(TestCase):

    def test_user_object_is_created(self):
        Cleaning_Request = Cleaning_Request.objects.create(
            User_Name = 'username',
            Room = 'D-343',
            Done = 1,
            Place = 'Room',
            comments = 'Nope',
        )
        #checking if the object is created
        self.assertEqual(Cleaning_Request.User_Name, 'username')
        self.assertEqual(Cleaning_Request.Room, 'D-343')
        self.assertEqual(Cleaning_Request.Done, 1)
        self.assertEqual(Cleaning_Request.Place, 'Room')
        self.assertEqual(Cleaning_Request.comments, 'Nope')
```

Test Results:
Found 10 test(s).
System check identified no issues (0 silenced).
....

Ran 10 tests in 1.108s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

6. Canteen

a) Urls Testing:

This testing will validate the correct url connecting from each path of urls.py to respective functions in the views.py

Unit Details: [
Functions: Student_Place_Order, Student_Cart, Student_Pending_Order, Student_Orders_History, Owner_New_Order, Owner_Pending_Order, Owner_Modify_Menu, Owner_Students_Bill
]
Test Owner: Rajeev Kumar
Test Date: [28/03/2023] - [28/03/2023]
Given Test inputs:

```
from django.test import SimpleTestCase
from django.urls import reverse, resolve
from Canteen.views import Student_Place_Order, Student_Cart,
Student_Pending_Order, Student_Orders_History, Owner_New_Order,
Owner_Pending_Order, Owner_Modify_Menu, Owner_Students_Bill

class TestUrls(SimpleTestCase):

    def test_Student_Place_Order_url(self):
        url = reverse('Student_Place_Order')
        self.assertEquals(resolve(url).func, Student_Place_Order)

    def test_Student_Cart_url(self):
        url = reverse('Student_Cart')
        self.assertEquals(resolve(url).func, Student_Cart)

    def test_Student_Pending_Order_url(self):
        url = reverse('Student_Pending_Order')
        self.assertEquals(resolve(url).func, Student_Pending_Order)

    def test_Student_Orders_History_url(self):
        url = reverse('Student_Orders_History')
```

```
        self.assertEquals(resolve(url).func, Student_Orders_History)

    def test_Owner_New_Order_url(self):
        url = reverse('Owner_New_Order')
        self.assertEquals(resolve(url).func, Owner_New_Order)

    def test_Owner_Pending_Order_url(self):
        url = reverse('Owner_Pending_Order')
        self.assertEquals(resolve(url).func, Owner_Pending_Order)

    def test_Owner_Modify_Menu_url(self):
        url = reverse('Owner_Modify_Menu')
        self.assertEquals(resolve(url).func, Owner_Modify_Menu)

    def test_Owner_Students_Bill_url(self):
        url = reverse('Owner_Students_Bill')
        self.assertEquals(resolve(url).func, Owner_Students_Bill)
```

Test Results:
Found 8 test(s).
System check identified no issues (0 silenced).
.....

Ran 8 tests in 0.023s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

b) Authentication Testing

This test will check where an unauthenticated user is shown error while trying to access any url in Canteen
Unit Details: [
Functions: Student_Place_Order, Student_Cart, Student_Pending_Order, Student_Orders_History, Owner_New_Order, Owner_Pending_Order, Owner_Modify_Menu, Owner_Students_Bill
]
Test Owner: Rajeev Kumar
Test Date: [28/03/2023] - [30/03/2023]
Given Test inputs:

```
from django.test import TestCase, Client
from django.urls import reverse
from Canteen.models import Menu, Bill, Order
from django.contrib.auth.models import User
from Login.models import UserManager, User_class

class TestView(TestCase):

    def setUp(self):
        # Create a student user
        self.student = User_class.objects.create_user(
            username='student',
            name='Student',
            password='1234@Div',
            designation='Student'
        )
        # Create a non-student user
        self.canteen_manager = User_class.objects.create_user(
            username='canteen_manager',
            name='Canteen Manager',
            password='1234@Div',
            designation='Canteen Manager'
        )
        # Create a menu item
        self.item = Menu.objects.create(
            Item_Name='item',
            Price=20
        )

    def tearDown(self):
        self.student.delete()
        self.canteen_manager.delete()
        self.item.delete()

    def test_Student_Place_Order_not_authenticated(self):
        # Submit a POST request to place an order without logging in
        response = self.client.post(
            reverse('Student_Place_Order'),
```

```
            data={
                'submit': self.item.id,
                'quantity': 2
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Student_Place_Order_not_student(self):
        # Login as the non-student user
        self.client.login(username='canteen_manager', password='1234@Div')
        # Submit a POST request to place an order
        response = self.client.post(
            reverse('Student_Place_Order'),
            data={
                'submit': self.item.id,
                'quantity': 2
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Student_Pending_Order_not_authenticated(self):
        # Submit a GET request to get the set of pending orders without
logging in
        response = self.client.get(reverse('Student_Pending_Order'))
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Student_Pending_Order_not_student(self):
        # Login as the non-student user
        self.client.login(username='canteen_manager', password='1234@Div')
        # Submit a GET request to get the set of pending orders
```

```
        response = self.client.get(reverse('Student_Pending_Order'))
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Student_Orders_History_not_authenticated(self):
        # Submit a GET request to get the set of past orders without logging
in
        response = self.client.get(reverse('Student_Orders_History'))
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Student_Orders_History_not_student(self):
        # Login as the non-student user
        self.client.login(username='canteen_manager', password='1234@Div')
        # Submit a GET request to get the set of past orders
        response = self.client.get(reverse('Student_Orders_History'))
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Student_Cart_remove_item_not_authenticated(self):
        # Submit a POST request to remove an item from cart without logging
in
        response = self.client.post(
            reverse('Student_Cart'),
            data={
                'order_validation1': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')
```

```
    def test_Student_Cart_confirm_order_not_authenticated(self):
        # Submit a POST request to confirm an order without logging in
        response = self.client.post(
            reverse('Student_Cart'),
            data={
                'order_validation2': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Student_Cart_remove_item_not_student(self):
        # Login as the non-student user
        self.client.login(username='canteen_manager', password='1234@Div')
        # Submit a POST request to remove an item from cart
        response = self.client.post(
            reverse('Student_Cart'),
            data={
                'order_validation1': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Student_Cart_confirm_order_not_student(self):
        # Login as the non-student user
        self.client.login(username='canteen_manager', password='1234@Div')
        # Submit a POST request to confirm the order
        response = self.client.post(
            reverse('Student_Cart'),
            data={
                'order_validation2': self.item.id,
            }
        )
        # Check that the response status is 200 OK
```

```
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_New_Order_accept_order_not_authenticated(self):
        # Submit a POST request to accept an order without logging in
        response = self.client.post(
            reverse('Owner_New_Order'),
            data={
                'accepted': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_New_Order_reject_order_not_authenticated(self):
        # Submit a POST request to reject an order without logging in
        response = self.client.post(
            reverse('Owner_New_Order'),
            data={
                'rejected': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_New_Order_accept_order_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to accept an order without logging in
        response = self.client.post(
            reverse('Owner_New_Order'),
            data={
                'accepted': self.item.id,
            }
```

```
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_New_Order_reject_order_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to reject an order
        response = self.client.post(
            reverse('Owner_New_Order'),
            data={
                'rejected': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def
test_Owner_Pending_Order_change_served_status_not_authenticated(self):
        # Submit a POST request to change served status of an order without
logging in
        response = self.client.post(
            reverse('Owner_Pending_Order'),
            data={
                'served': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Pending_Order_change_paid_status_not_authenticated(self):
        # Submit a POST request to change paid status of an order without
logging in
```

```
        response = self.client.post(
            reverse('Owner_Pending_Order'),
            data={
                'paid': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Pending_Order_remove_order_not_authenticated(self):
        # Submit a POST request to remove an order without logging in
        response = self.client.post(
            reverse('Owner_Pending_Order'),
            data={
                'remove': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Pending_Order_served_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to change served status an order
        response = self.client.post(
            reverse('Owner_Pending_Order'),
            data={
                'served': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')
```

```
    def test_Owner_Pending_Order_paid_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to change paid status of an order
        response = self.client.post(
            reverse('Owner_Pending_Order'),
            data={
                'paid': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Pending_Order_remove_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to remove an order
        response = self.client.post(
            reverse('Owner_Pending_Order'),
            data={
                'remove': self.item.id,
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Modify_Menu_edit_item_not_authenticated(self):
        # Submit a POST request to edit an item without logging in
        response = self.client.post(
            reverse('Owner_Modify_Menu'),
            data={
                'editable_mode': 1
            }
        )
        # Check that the response status is 200 OK
```

```
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Modify_Menu_add_item_not_authenticated(self):
        # Submit a POST request to add an item without logging in
        response = self.client.post(
            reverse('Owner_Modify_Menu'),
            data={
                'add_hidden_item': 1
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Modify_Menu_delete_item_not_authenticated(self):
        # Submit a POST request to delete an item without logging in
        response = self.client.post(
            reverse('Owner_Modify_Menu'),
            data={
                'delete': self.item.id
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Modify_Menu_edit_item_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to edit an item
        response = self.client.post(
            reverse('Owner_Modify_Menu'),
            data={
                'editable_mode': 1
            }
```

```
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Modify_Menu_add_item_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to add an item
        response = self.client.post(
            reverse('Owner_Modify_Menu'),
            data={
                'add_hidden_item': 1
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Modify_Menu_delete_item_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to delete an item
        response = self.client.post(
            reverse('Owner_Modify_Menu'),
            data={
                'delete': self.item.id
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Students_Bill_not_authenticated(self):
        # Submit a POST request to view students' bill without logging in
        response = self.client.post(reverse('Owner_Students_Bill'))
```

```
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Owner_Students_Bill_not_canteen_manager(self):
        # Login as the student user
        self.client.login(username='student', password='1234@Div')
        # Submit a POST request to modify students' bill
        response = self.client.post(
            reverse('Owner_Students_Bill'),
            data={
                'order_validation1': 100,
                'submit': 'student'
            }
        )
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')
```

Test Results:
Found 28 test(s).
System check identified no issues (0 silenced).
.....

Ran 28 tests in 15.396s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

c) Model Testing:

This will check the correct instantiation of the classes, whether is object is being created or not.

Unit Details: [
Models: Menu, Bill, Order]
Test Owner: Rajeev Kumar
Test Date: [30/03/2023] - [31/03/2023]
Given Test inputs:

```
from django.test import TestCase
from Canteen.models import Menu, Bill, Order
from datetime import datetime

class TestModels(TestCase):

    def test_Menu_object(self):
        # Create an object of class Menu
        Menu_object = Menu.objects.create(
            Item_Name='item',
            Price=20
        )
        # Check if the object is created
        self.assertEqual(Menu_object.Item_Name, 'item')
        self.assertEqual(Menu_object.Price, 20)

    def test_Bill_object(self):
        # Create an object of class Bill
        Bill_object = Bill.objects.create(
            User_Name='student',
            Name='Student',
            Amount=1000
        )
        # Check if the object is created
        self.assertEqual(Bill_object.User_Name, 'student')
        self.assertEqual(Bill_object.Name, 'Student')
```

```
        self.assertEqual(Bill_object.Amount, 1000)

    def test_Order_object(self):
        # Create an object of class Bill
        Order_object = Order.objects.create(
            User_Name='student',
            Name='Student',
            Item_Name='item',
            Price=20,
            Quantity=2,
            Amount=40,
            Order_Date_Time =datetime.now(),
            Cart_Status=0,
            Processing_Status=0,
            Accepted_Status=1,
            History_Status=0,
            Served_Status=1,
            Payment_Status=0,
        )
        # Check if the object is created
        self.assertEqual(Order_object.User_Name, 'student')
        self.assertEqual(Order_object.Name, 'Student')
        self.assertEqual(Order_object.Item_Name, 'item')
        self.assertEqual(Order_object.Price, 20)
        self.assertEqual(Order_object.Quantity, 2)
        self.assertEqual(Order_object.Amount, 40)
        self.assertEqual(Order_object.Cart_Status, 0)
        self.assertEqual(Order_object.Processing_Status, 0)
        self.assertEqual(Order_object.Accepted_Status, 1)
        self.assertEqual(Order_object.History_Status, 0)
        self.assertEqual(Order_object.Served_Status, 1)
        self.assertEqual(Order_object.Payment_Status, 0)
```

Test Results:

Found 3 test(s).

System check identified no issues (0 silenced).

...

Ran 3 tests in 0.063s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

7. My Account

a) Urls Testing:

This testing will validate the correct url connecting from each path of urls.py to respective functions in the views.py

Unit Details: [
Functions: Mess, Canteen
]
Test Owner: Rajeev Kumar
Test Date: [30/03/2023] - [30/03/2023]
Given Test inputs:

```
from django.test import SimpleTestCase
from django.urls import reverse, resolve
from My_Account.views import Mess, Canteen

class TestUrls(SimpleTestCase):

    def test_Mess_url(self):
        url = reverse('Mess')
        self.assertEquals(resolve(url).func, Mess)

    def test_Canteen_url(self):
        url = reverse('Canteen')
        self.assertEquals(resolve(url).func, Canteen)
```

Test Results:
Found 2 test(s).
System check identified no issues (0 silenced).
..

Ran 2 tests in 0.020s

OK

Structural Coverage: Code Coverage :100% and Branch Coverage :100%

Additional Comments: No comments

b) Authentication Testing:

This test will check where an unauthenticated user is shown error while trying to access any url in My Account

Unit Details: [
Functions: Mess, Canteen
]
Test Owner: Rajeev Kumar
Test Date: [30/03/2023] - [30/03/2023]
Given Test inputs:

```
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth.models import User
from Login.models import UserManager, User_class

class TestView(TestCase):

    def setUp(self):
        # Create a student user
        self.student = User_class.objects.create_user(
            username='student',
            name='Student',
            password='1234@Div',
            designation='Student'
        )
        # Create a non-student user
        self.canteen_manager = User_class.objects.create_user(
            username='canteen_manager',
            name='Canteen Manager',
            password='1234@Div',
            designation='Canteen Manager'
        )

    def tearDown(self):
```

```
        self.student.delete()
        self.canteen_manager.delete()

    def test_Mess_not_authenticated(self):
        # Submit a POST request to view mess bill without logging in
        response = self.client.post(reverse('Mess'))
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Mess_not_student(self):
        # Login as the non-student user
        self.client.login(username='canteen_manager',
password='1234@Div')
        # Submit a POST request to view mess bill
        response = self.client.post(reverse('Mess'))
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Canteen_not_authenticated(self):
        # Submit a POST request to view canteen bill without logging
in
        response = self.client.post(reverse('Canteen'))
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')

    def test_Canteen_not_student(self):
        # Login as the non-student user
        self.client.login(username='canteen_manager',
password='1234@Div')
        # Submit a POST request to view canteen bill
        response = self.client.post(reverse('Canteen'))
        # Check that the response status is 200 OK
        self.assertEqual(response.status_code, 200)
```

```
        # Check that the rendered template is Error.html
        self.assertTemplateUsed(response, 'Error.html')
```

Test Results:
Found 4 test(s).
System check identified no issues (0 silenced).
....

Ran 4 tests in 2.884s

OK

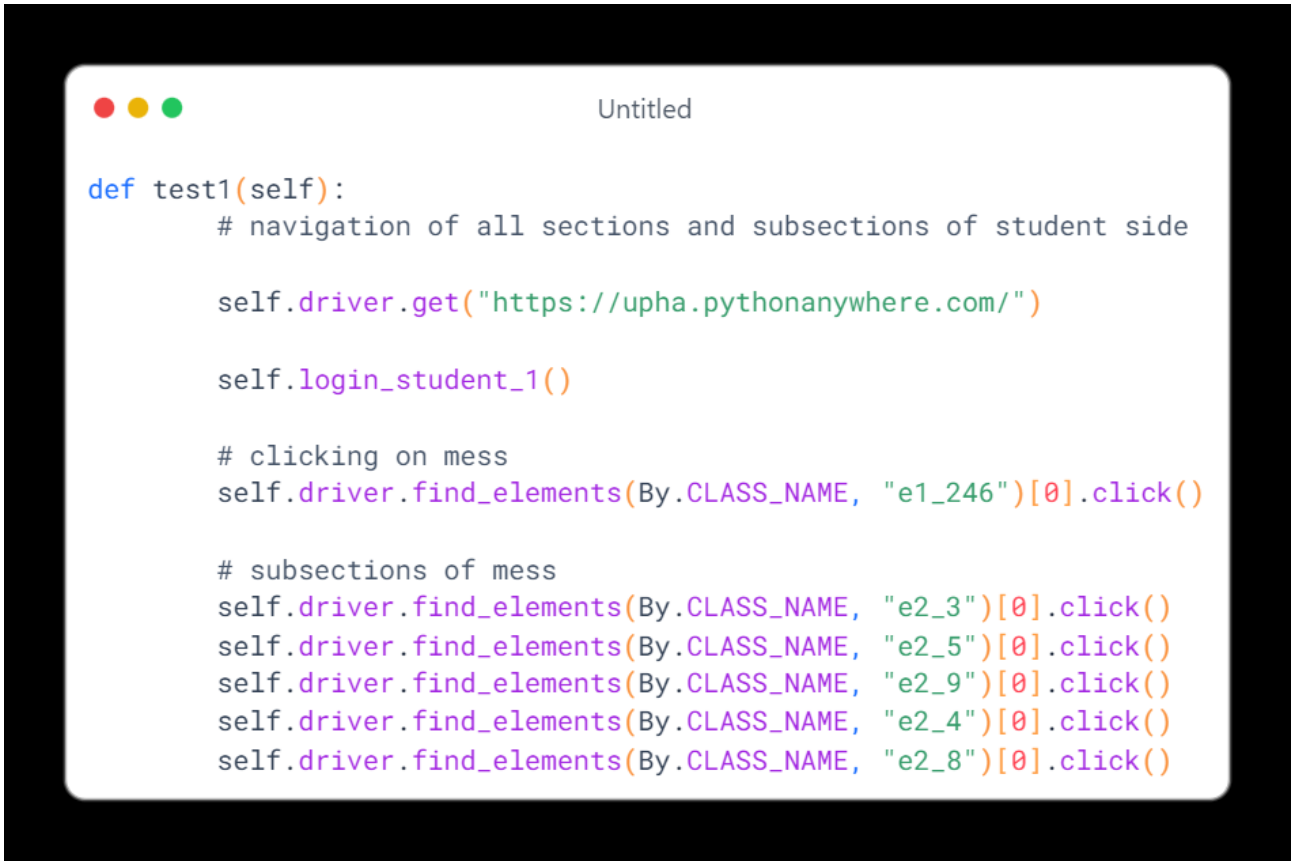
Structural Coverage: Code Coverage :100% and Branch Coverage :100%
Additional Comments: No comments

3 Integration Testing

1 Test1

Tested the navigation of all sections and subsections of student side.

Module Details: This unit contains all classes, templates, and functions necessary to navigate through the section and the subsection from student's side.
Test Owner: Divyansh Chhabria.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to open various sections and subsections from the student's side and check their navigation. All the navigations were performed successfully.
Commands: python test_frontend.py



Untitled

```
# clicking on canteen
        self.driver.find_elements(By.CLASS_NAME, "e1_242")[0].click()

        # subsections of canteen
        self.driver.find_elements(By.CLASS_NAME, "e2_3")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_9")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()

        # clicking on booking
        self.driver.find_elements(By.CLASS_NAME, "e1_234")[0].click()

        # subsections of booking
        self.driver.find_elements(By.CLASS_NAME, "e2_3")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_9")[0].click()

        # clicking on cleaning
        self.driver.find_elements(By.CLASS_NAME, "e1_238")[0].click()

        # subsections of cleaning
        self.driver.find_elements(By.CLASS_NAME, "e2_3")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_9")[0].click()

        # clicking on my account
        self.driver.find_elements(By.CLASS_NAME, "e1_230")[0].click()

        # subsections of my account
        self.driver.find_elements(By.CLASS_NAME, "e2_3")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

        # clicking on home through title
        self.driver.find_elements(By.CLASS_NAME, "e1_261")[0].click()

        # clicking on canteen
        self.driver.find_elements(By.CLASS_NAME, "e1_242")[0].click()

        # clicking on home
        self.driver.find_elements(By.CLASS_NAME, "e1_250")[0].click()

        # logging out
        self.logout()
```

2 Test2

Tested the navigation of all sections and subsections of Mess Manager.

Module Details: This unit contains all classes, templates, and functions necessary to navigate through the section and the subsection from Mess Manager's side.
Test Owner: Jhalak Sharma.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to open various sections and subsections from the Mess Manager's side and check their navigation. All the navigations were performed successfully**.**
Command: python test_frontend.py

test_frontend.py

```
def test2(self):
    # navigation of all sections and subsections of Mess Manager
    self.driver.get("https://upha.pythonanywhere.com/")

    self.login_mess_manager()

    # clicking on mess
    self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

    # subsections of mess
    self.driver.find_elements(By.CLASS_NAME, "e2_3")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_31")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_32")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_33")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_34")[0].click()

    # clicking on home
    self.driver.find_elements(By.CLASS_NAME, "e1_250")[0].click()

    # logging out
    self.logout()
```

3 Test3

Tested the navigation of all sections and subsections of Canteen Manager.

Module Details: This unit contains all classes, templates, and functions necessary to navigate through the section and the subsection from Mess Manager's side.
Test Owner: Soham Bharambe.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to open various sections and subsections from the Canteen Manager's side and check their navigation. All the navigations were performed successfully**.**
Command: python test_frontend.py

test_frontend.py

```
def test3(self):
    # navigation of all sections and subsections of Canteen Manager
    self.driver.get("https://upha.pythonanywhere.com/")

    self.login_canteen_manager()

    # clicking on canteen
    self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

    # subsections of canteen
    self.driver.find_elements(By.CLASS_NAME, "e2_3")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_31")[0].click()

    # clicking on home
    self.driver.find_elements(By.CLASS_NAME, "e1_250")[0].click()

    # logging out
    self.logout()
```

4 Test4

Tested the navigation of all sections and subsections of Hall Manager.

Module Details: This unit contains all classes, templates, and functions necessary to navigate through the section and the subsection from Hall Manager's side.
Test Owner: Divyansh Chhabria
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to open various sections and subsections from the Hall Manager's side and check their navigation. All the navigations were performed successfully**.**
Command: python test_frontend.py

test_frontend.py

```
def test4(self):
    # navigation of all sections and subsections of Hall Manager
    self.driver.get("https://upha.pythonanywhere.com/")

    self.login_hall_manager()

    # clicking on hall
    self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

    # subsections of hall
    self.driver.find_elements(By.CLASS_NAME, "e2_3")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

    # clicking on home
    self.driver.find_elements(By.CLASS_NAME, "e1_250")[0].click()

    # logging out
    self.logout()
```

5 Test5

Tested the navigation of all sections and subsections of Sports Secy.

Module Details: This unit contains all classes, templates, and functions necessary to navigate through the section and the subsection from Sports Secy's side.
Test Owner: Jhalak Sharma
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.

Additional Comments: Using the selenium framework, code was written to open various sections and subsections from the Sports Secy's side and check their navigation. All the navigations were performed successfully.
Command: python test_frontend.py

test_frontend.py

```
def test5(self):
    # navigation of all sections and subsections of Sports secy
    self.driver.get("https://upha.pythonanywhere.com/")

    self.login_sports_secy()

    # clicking on sports
    self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

    # subsections of sports
    self.driver.find_elements(By.CLASS_NAME, "e2_3")[0].click()
    self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

    # clicking on home
    self.driver.find_elements(By.CLASS_NAME, "e1_250")[0].click()

    # logging out
    self.logout()
```

6 Test6

Tested the modify menu page of the Mess manager's side and its functionality.

Module Details: This unit contains all the classes, templates and functions necessary for modifying the menu from Mess Manager's side.
Test Owner: Soham Bharambe.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to edit, add and delete the items in the menu.
Command: python test_frontend.py

test_frontend.py

```
        # adding item
        self.driver.find_elements(By.NAME, "add_hidden_item")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "a")[1].click()
        self.driver.find_elements(By.NAME, "a2")[1].click()
        self.driver.find_elements(By.CLASS_NAME, "b")[1].click()
        self.driver.find_elements(By.NAME, "b2")[1].click()
        self.driver.find_elements(By.CLASS_NAME, "c")[1].send_keys("Pav Bhaji")

        # submitting
        self.driver.find_elements(By.NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Changes made successfully.")
```

test_frontend.py

```
 # deleting
        self.driver.find_elements(By.NAME, "delete")[1].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Deleted Successfully.")
```

test_frontend.py

```
        # editting
        self.driver.find_elements(By.NAME, "edit")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "a")[1].click()
        self.driver.find_elements(By.NAME, "a0")[1].click()
        self.driver.find_elements(By.CLASS_NAME, "b")[1].click()
        self.driver.find_elements(By.NAME, "b0")[1].click()
        self.driver.find_elements(By.CLASS_NAME, "c")[1].clear()
        self.driver.find_elements(By.CLASS_NAME, "c")[1].send_keys("Sandwitch")

        # submitting
        self.driver.find_elements(By.NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Changes made successfully.")
```

7 Test7

Tested the Extra menu page of the Mess manager's side and its functionality.

Module Details: This unit contains all the classes, templates, and functions necessary for adding a modifying the Extra's menu from Mess Manager's side.
Test Owner: Divyansh Chhabria.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to edit, add and delete the items in the menu.

Command: python test_frontend.py

test_frontend.py

```
# adding item
self.driver.find_elements(By.NAME, "add_hidden_item")[0].click()
self.driver.find_elements(By.CLASS_NAME, "a")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "a")[0].send_keys("2023-04-30")
self.driver.find_elements(By.CLASS_NAME, "b")[0].click()
self.driver.find_elements(By.NAME, "b2")[0].click()
self.driver.find_elements(By.CLASS_NAME, "c")[0].send_keys("Masala Dosa")
self.driver.find_elements(By.CLASS_NAME, "d")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "d")[0].send_keys("250")
self.driver.find_elements(By.CLASS_NAME, "e")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "e")[0].send_keys("34")
self.driver.find_elements(By.CLASS_NAME, "f")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "f")[0].send_keys("02-04-002023T00:00")
self.driver.find_elements(By.CLASS_NAME, "g")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "g")[0].send_keys("30-03-002023T00:00")

# submitting
self.driver.find_elements(By.NAME, "submit")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Changes made successfully.")
```

test_frontend.py

```
# deleting
self.driver.find_elements(By.NAME, "delete")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Deleted Successfully.")
```

test_frontend.py

```
# editting item
self.driver.find_elements(By.CLASS_NAME, "a")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "a")[0].send_keys("2023-04-30")
self.driver.find_elements(By.CLASS_NAME, "b")[0].click()
self.driver.find_elements(By.NAME, "b2")[0].click()
self.driver.find_elements(By.CLASS_NAME, "c")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "c")[0].send_keys("Masala Dosa")
self.driver.find_elements(By.CLASS_NAME, "d")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "d")[0].send_keys("250")
self.driver.find_elements(By.CLASS_NAME, "e")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "e")[0].send_keys("34")
self.driver.find_elements(By.CLASS_NAME, "f")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "f")[0].send_keys("30-03-002023T00:00")
self.driver.find_elements(By.CLASS_NAME, "g")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "g")[0].send_keys("02-04-002023T00:00")

# submitting
self.driver.find_elements(By.NAME, "submit")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Deleted Successfully.")
```

8 Test8

Tested the Enter BDMR functionality of the Mess manager's side.

Module Details: This unit contains all the classes, templates, and functions necessary for adding a modifying the BDMR and entering the BDMR date from Mess Manager's side.
Test Owner: Jhalak Sharma
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to enter the BDMR, the date for the BDMR and finally submitting the information.
Command: python test_frontend.py

test_frontend.py

```
def test8(self):
        # enter BDMR

        # mess manager extra menu
        self.driver.get("https://upha.pythonanywhere.com/")

        self.login_mess_manager()

        # clicking on mess
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_34")[0].click()
        self.driver.find_elements(By.NAME, "BDMR_Date")[0].send_keys("2023-03-30")
        self.driver.implicitly_wait(1)
        self.driver.find_elements(By.NAME, "BDMR")[0].send_keys("63.89")
        self.driver.find_elements(By.NAME, "bdmr_submit")[0].click()

        # logging out
        self.logout()
```

9 Test9

Testing the interaction between student and Mess Manager's side of Modification of menu and Rating functionality.

Module Details: This unit contains all the classes, templates, and functions necessary for adding a modifying the Mess Menu from and viewing the rating by students from Manager's side and viewing the mess menu and rating functionality from Student's side.
Test Owner: Soham Bharambe.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to add, edit and delete an item in the mess menu and view the ratings from Manager's side and entering the ratings and viewing the Mess menu from the student's side.
Command: python test_frontend.py

test_frontend.py

```
        # mess manager login
        self.login_mess_manager()

        # clicking on mess
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_31")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "Thursday")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Lunch")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "Rajma Chawal")
        assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "☆ 3.00")

        # logging out
        self.logout()
```

10 Test10

Testing the interaction between student and Mess Manager's side of Modification of Extra's menu.

Module Details: This unit contains all the classes, templates, and functions necessary for adding a modifying the Extra's Menu from Manager's side and viewing the extra's menu from Student's side.

Test Owner: Divyansh Chhabria.

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: Using the selenium framework, code was written to add, edit and delete an item in the extra's menu and view the ratings from Manager's side and viewing the extra's menu from the student's side.

Command: python test_frontend.py

test_frontend.py

```
# student checking subsections of mess
self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "Masala Dosa")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "April 30, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "Dinner")
assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "March 30, 2023, midnight")
assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "April 2, 2023, midnight")
assert(self.driver.find_elements(By.CLASS_NAME, "g")[0].text == "34")

self.driver.find_elements(By.NAME, "quantity")[0].clear()
self.driver.find_elements(By.NAME, "quantity")[0].send_keys("3")
self.driver.find_elements(By.NAME, "submit")[0].click()
```

test_frontend.py

```
# hall manager checking subsections of mess
self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "April 30, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "s1")
assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "Dinner")
assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "Masala Dosa")
assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "3")
```

test_frontend.py

```
        # student login
        self.login_student_1()

        # clicking on mess
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_9")[0].click()

        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "Masala Dosa")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "3")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "April 30, 2023")
        assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "Dinner")

        self.driver.find_elements(By.NAME, "order_validation")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()

        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Dinner")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "Masala Dosa")
        assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "3")
        assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "34")
        assert(self.driver.find_elements(By.CLASS_NAME, "f")[0].text == "102")
```

11 Test11

Testing the functionality of setting the Start time and End time for Extra item's booking from Mess Manager' side.
Module Details: This unit contains all the classes, templates, and functions necessary for logging in, setting the Start time and End time.
Test Owner: Jhalak Sharma
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to login the Mess Manager and entering the Start time and End time and finally submitting all the information.
Command: python test_frontend.py

test_frontend.py

```
# adding item
        self.driver.find_elements(By.NAME, "add_hidden_item")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "a")[0].clear()
        self.driver.find_elements(By.CLASS_NAME, "a")[0].send_keys("2023-04-30")
        self.driver.find_elements(By.CLASS_NAME, "b")[0].click()
        self.driver.find_elements(By.NAME, "b2")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "c")[0].send_keys("Masala Dosa")
        self.driver.find_elements(By.CLASS_NAME, "d")[0].clear()
        self.driver.find_elements(By.CLASS_NAME, "d")[0].send_keys("250")
        self.driver.find_elements(By.CLASS_NAME, "e")[0].clear()
        self.driver.find_elements(By.CLASS_NAME, "e")[0].send_keys("34")
        self.driver.find_elements(By.CLASS_NAME, "f")[0].clear()
        self.driver.find_elements(By.CLASS_NAME, "f")[0].send_keys("02-04-002023T00:00")
        self.driver.find_elements(By.CLASS_NAME, "g")[0].clear()
        self.driver.find_elements(By.CLASS_NAME, "g")[0].send_keys("30-03-002023T00:00")

        # submitting
        self.driver.find_elements(By.NAME, "submit")[0].click()

        assert(self.driver.find_elements(By.CLASS_NAME, "error")[0].text == "Start time must be less than end time.")
```

12 Test12

Testing the Extra's booking from student's interface and testing whether the software acknowledges the time constraints.
Module Details: This unit contains all the classes, templates, and functions necessary for logging in, setting the Start time and End time from the Manager's side and booking functionality from the student's side.
Test Owner: Soham Bharambe
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using the selenium framework, code was written to login the Mess Manager and entering the Start time and End time and finally submitting all the information and booking an item at a time after the end time, so that the system gives error.
Command: python test_frontend.py

test_frontend.py

```
        # student login
        self.login_student_1()

        # clicking on mess
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

        self.driver.find_elements(By.NAME, "quantity")[0].clear()
        self.driver.find_elements(By.NAME, "quantity")[0].send_keys("2")
        self.driver.find_elements(By.NAME, "submit")[0].click()

        assert(self.driver.find_elements(By.CLASS_NAME, "error")[0].text == "Order cannot be booked now")
        # logging out
        self.logout()
```

13 Test13

Testing the Applying for Rebate functionality from student's interface.
Module Details: This unit contains all the classes, templates, and functions necessary for logging in, entering the To and From date and finally submitting all the information.
Test Owner: Divyansh Chhabria.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: The code also gives a popup showing that the request is successfully submitted.
Command: python test_frontend.py

test_frontend.py

```
        # student login
        self.login_student_1()

        # clicking on mess
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_8")[0].click()

        self.driver.find_elements(By.NAME, "from")[0].clear()
        self.driver.find_elements(By.NAME, "from")[0].send_keys("2023-03-31")
        self.driver.find_elements(By.NAME, "to")[0].clear()
        self.driver.find_elements(By.NAME, "to")[0].send_keys("2023-04-01")
        self.driver.find_elements(By.NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your rebate request has been sent to the mess manager.
        You will soon receive a confirmation email.")
```

14 Test14

Testing the software's acknowledgement of date constraints when a student applies for rebate.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, entering the To and From date and finally submitting all the information.

Test Owner: Jhalak Sharma

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: A "From-Date" late than the "To-Date" was entered using the selenium framework and software gave an error popup of "From-Date must be less than To-Date"

Command: python test_frontend.py

test_frontend.py

```
def test14(self):
    # student apply for rebate
    self.driver.get("https://upha.pythonanywhere.com/")

    # student login
    self.login_student_1()

    # clicking on mess
    self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

    # subsections of mess
    self.driver.find_elements(By.CLASS_NAME, "e2_8")[0].click()

    self.driver.find_elements(By.NAME, "from")[0].clear()
    self.driver.find_elements(By.NAME, "from")[0].send_keys("2023-04-30")
    self.driver.find_elements(By.NAME, "to")[0].clear()
    self.driver.find_elements(By.NAME, "to")[0].send_keys("2023-04-01")
    self.driver.find_elements(By.NAME, "submit")[0].click()
    assert(self.driver.find_elements(By.CLASS_NAME, "error")[0].text == "From-Date must be less
    than equal to To-Date")
    self.logout()
```

15 Test15

Testing the interaction between student' side and mess manager side regarding applying for rebate functionality.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, entering the To and From date and finally submitting all the information and logging in and displaying the rebate requests and also the functionality to accept or reject them from Mess Manger's side.

Test Owner: Soham Bharambe

Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: A rebate request was made from student's side and that request was displayed on the Mess Manager's side and they accepted the request. A message confirming the accepting was displayed and a mail regarding the same was received by the student.
Command: python test_frontend.py

test_frontend.py

```
# student login
self.login_student_1()

# clicking on mess
self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

# subsections of mess
self.driver.find_elements(By.CLASS_NAME, "e2_8")[0].click()

self.driver.find_elements(By.NAME, "from")[0].clear()
self.driver.find_elements(By.NAME, "from")[0].send_keys("2023-03-31")
self.driver.find_elements(By.NAME, "to")[0].clear()
self.driver.find_elements(By.NAME, "to")[0].send_keys("2023-04-01")
self.driver.find_elements(By.NAME, "submit")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your rebate request has been
sent to the mess manager. You will soon receive a confirmation email.")

self.logout()
```

test_frontend.py

```
# mess manager login
self.login_mess_manager()

# clicking on mess
self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

# subsections of mess
self.driver.find_elements(By.CLASS_NAME, "e2_32")[0].click()

self.driver.find_elements(By.NAME, "accept")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Rebate Request Accepted.")
self.logout()
```

16 Test16

Testing the interaction between student' side and mess manager side regarding applying for rebate functionality.
Module Details: This unit contains all the classes, templates, and functions necessary for logging in, entering the To and From date and finally submitting all the information and

logging in and displaying the rebate requests and also the functionality to accept or reject them from Mess Manger's side.
Test Owner: Divyansh Chhabria.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: A rebate request was made from student's side and that request was displayed on the Mess Manager's side and they rejected the request. A message confirming the rejecting was displayed and a mail regarding the same was received by the student.

test_frontend.py

```
        # student login
        self.login_student_1()

        # clicking on mess
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_8")[0].click()

        self.driver.find_elements(By.NAME, "from")[0].clear()
        self.driver.find_elements(By.NAME, "from")[0].send_keys("2023-03-31")
        self.driver.find_elements(By.NAME, "to")[0].clear()
        self.driver.find_elements(By.NAME, "to")[0].send_keys("2023-04-01")
        self.driver.find_elements(By.NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your rebate request has
        been sent to the mess manager. You will soon receive a confirmation email.")

        self.logout()
```

test_frontend.py

```
        # mess manager login
        self.login_mess_manager()

        # clicking on mess
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_32")[0].click()

        self.driver.find_elements(By.NAME, "reject")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Rebate Request Rejected.")
        self.logout()
```

17 Test17

Testing the interaction between student' side and mess manager's side regarding the

displaying combined bill and updating BDMR and submitting the final bill on both the sides.
Module Details: This unit contains all the classes, templates, and functions necessary for logging in, adding the item in the extras, along with adding the Start and End time, entering the BDMR for each date and accepting or rejecting functionality from the manager's side. Also includes all the classes, templates, and functions for displaying the extra's menu, functionality of booking an extra item and applying for rebate request from student's side.
Test Owner: Jhalak Sharma.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using selenium framework an item was added in the extra's menu by the Mess Manager, that item was booked by a called student_1 and it's price was added to their final bill. The mess manager also entered the BDMR value for certain dates. The same student applied for rebate and the respective amount was deducted from his final bill. Finally, the final bill is displayed on student's interface.
Command: python test_frontend.py

test_frontend.py

```
# student login
self.login_student_1()

# clicking on mess
self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

# subsections of mess
self.driver.find_elements(By.CLASS_NAME, "e2_8")[0].click()

# applying for rebate
self.driver.find_elements(By.NAME, "from")[0].clear()
self.driver.find_elements(By.NAME, "from")[0].send_keys("2023-03-31")
self.driver.find_elements(By.NAME, "to")[0].clear()
self.driver.find_elements(By.NAME, "to")[0].send_keys("2023-04-01")
self.driver.find_elements(By.NAME, "submit")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your rebate request has
been sent to the mess manager. You will soon receive a confirmation email.")

self.logout()
```

test_frontend.py

```
# student login
        self.login_student_1()

        # clicking on my account
        self.driver.find_elements(By.CLASS_NAME, "e1_230")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "171.48")
        assert(self.driver.find_elements(By.CLASS_NAME, "e")[1].text == "157.19")

        # logging out
        self.logout()
```

test_frontend.py

```
        # mess manager login
        self.login_mess_manager()

        # clicking on mess
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_32")[0].click()

        self.driver.find_elements(By.NAME, "accept")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Rebate Request Accepted.")

        # subsections of mess
        self.driver.find_elements(By.CLASS_NAME, "e2_33")[0].click()

        assert(self.driver.find_elements(By.CLASS_NAME, "g")[0].text == "171.48")
        assert(self.driver.find_elements(By.CLASS_NAME, "g")[3].text == "157.19")
        self.logout()
```

18 Test18

Testing the interaction between student' side and hall manager's side regarding the lodging the cleaning complaints from student's interface and displaying those requests on hall manager' interface.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, lodging the request by entering the room number, and other specifications. Also displaying those requests on the hall manager's interface

Test Owner: Soham Bharambe.

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: Using selenium framework several requests were lodged and those requests were visible to the hall manager.
Command: python test_frontend.py

test_frontend.py

```
assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "s1")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "March 31, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "D-213")
assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "Room")
assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "Room check")
assert(self.driver.find_elements(By.CLASS_NAME, "a")[1].text == "s1")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[1].text == "March 31, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "c")[1].text == "F-211")
assert(self.driver.find_elements(By.CLASS_NAME, "d")[1].text == "Toilet")
assert(self.driver.find_elements(By.CLASS_NAME, "e")[1].text == "Toilet check")
assert(self.driver.find_elements(By.CLASS_NAME, "a")[2].text == "s1")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[2].text == "March 31, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "c")[2].text == "D-343")
assert(self.driver.find_elements(By.CLASS_NAME, "d")[2].text == "Corridor")
assert(self.driver.find_elements(By.CLASS_NAME, "e")[2].text == "Corridor check")
assert(self.driver.find_elements(By.CLASS_NAME, "a")[3].text == "s1")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[3].text == "March 31, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "c")[3].text == "F-413")
assert(self.driver.find_elements(By.CLASS_NAME, "d")[3].text == "Others")
assert(self.driver.find_elements(By.CLASS_NAME, "e")[3].text == "Others check")
```

test_frontend.py

```
# checking the data for student 1

assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Others")
self.driver.find_elements(By.NAME, "identity")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Corridor")
self.driver.find_elements(By.NAME, "identity")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Toilet")
self.driver.find_elements(By.NAME, "identity")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Room")
self.driver.find_elements(By.NAME, "identity")[0].click()
```

test_frontend.py

```
# Past requests of student 1 assessment

self.driver.find_elements(By.CLASS_NAME, "e2_9")[0].click()

assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Others")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[1].text == "Corridor")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[2].text == "Toilet")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[3].text == "Room")
```

19 Test19

Testing the functionalities of modify menu from canteen manager's side.
Module Details: This unit contains all the classes, templates, and functions necessary for logging in, adding an item, editing, or deleting the item and finally submitting all the changes from canteen manager's side.
Test Owner: Soham Bharambe.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using selenium framework several items were added in the menu from canteen owner's side, also these items were edited and deleted, finally the framework logged out.
Command: python test_frontend.py

test_frontend.py

```
# clicking on canteen
self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

# subsections of canteen
self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()

# adding item
self.driver.find_elements(By.NAME, "add_hidden_item")[0].click()
self.driver.find_elements(By.CLASS_NAME, "a")[0].send_keys("Veg. Maggi")
self.driver.find_elements(By.CLASS_NAME, "b")[0].clear()
self.driver.find_elements(By.CLASS_NAME, "b")[0].send_keys("30")
```

test_frontend.py

```
# editting
self.driver.find_elements(By.NAME, "edit")[0].click()
self.driver.find_elements(By.CLASS_NAME, "a")[1].clear()
self.driver.find_elements(By.CLASS_NAME, "a")[1].send_keys("Sandwitch")
self.driver.find_elements(By.CLASS_NAME, "b")[1].clear()
self.driver.find_elements(By.CLASS_NAME, "b")[1].send_keys("25")

# submitting
self.driver.find_elements(By.NAME, "submit")[0].click()
```

test_frontend.py

```
# deleting
self.driver.find_elements(By.NAME, "delete")[1].click()

# deleting
self.driver.find_elements(By.NAME, "delete")[0].click()
```

20 Test20

Testing the functionalities of modify menu from canteen manager's side and "adding to" and "removing from" cart functionality from student's side.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, adding an item to the menu from canteen manager's side and adding certain item from menu to the cart and removing that item from the cart from student's side.

Test Owner: Divyansh Chhabria.

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: Using selenium framework several items were added in the menu from canteen owner's side, these items were added to cart from student's side and then removed.

Command: python test_frontend.py

test_frontend.py

```
# login student 1
        self.login_student_1()

        # clicking on canteen
        self.driver.find_elements(By.CLASS_NAME, "e1_242")[0].click()

        self.driver.find_elements(By.NAME, "quantity")[0].clear()
        self.driver.find_elements(By.NAME, "quantity")[0].send_keys("4")
        self.driver.find_elements(By.NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your order has been successfully added to cart")

        # going to cart

        # subsections of canteen
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

        # delete the item
        self.driver.find_elements(By.NAME, "order_validation1")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your Cart Item has been successfully removed")
```

21 Test21

Testing the functionalities of modify menu and rejecting the order from canteen manager's side and "adding to" cart and book order functionality from student's side.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, adding an item to the menu and rejecting the order from canteen manager's side and adding certain item from menu to the cart and finally booking the order from student's side.

Test Owner: Jhalak Sharma.

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: Using selenium framework several items were added in the menu from canteen owner's side, these items were added to cart and booked from student's side. Finally, the order was rejected and the notification and mail of rejection was sent successfully to the student.

Command: python test_frontend.py

test_frontend.py

```
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your order has been successfully added to cart")

        # going to cart

        # subsections of canteen
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

        # delete the item
        self.driver.find_elements(By.NAME, "order_validation2")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Order request has been made to Canteen Manager")

        self.logout()
```

test_frontend.py

```
        self.login_canteen_manager()

        # clicking on canteen
        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()

        # rejecting order
        self.driver.find_elements(By.NAME, "rejected")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "error")[0].text == "You rejected the order")

        self.logout()
```

test_frontend.py

```
        # checking in student order history

        self.login_student_1()

        # clicking on canteen
        self.driver.find_elements(By.CLASS_NAME, "e1_242")[0].click()

        # subsection:Order history
        self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Veg. Maggi")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "4")
        assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "30")
        assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "120")
        assert(self.driver.find_elements(By.CLASS_NAME, "f")[0].text == "Failed")

        self.logout()
```

22 Test22

Testing the functionalities of modify menu and accepting the orders from canteen owner's side and the whole functionality of serving and keeping track of the payment from canteen owner's side. Also, several functionalities from student's side such as adding to cart, book an order and view the canteen dues were tested.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, adding an item to the menu and accepting the order from canteen manager's side and adding certain item from menu to the cart and finally booking the order from student's side.

Test Owner: Soham Bharambe.

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: Using selenium framework several items were added in the menu from canteen owner's side, these items were added to cart and booked from student's side. Finally, the order was accepted and the served and payment done clarification were made from the canteen owner's side. On checking upon my accounts section from student's side the section did not show any additional dues. Hence, successfully executing all the targeted the functionalities.

Command: python test_frontend.py

test_frontend.py

```
        # accepting the order
        self.driver.find_elements(By.NAME, "accepted")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "You accepted the order")

        # changing subsection
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

        # served
        self.driver.find_elements(By.NAME, "served")[0].click()

        # paid
        self.driver.find_elements(By.NAME, "paid")[0].click()

        assert(len(self.driver.find_elements(By.NAME, "served")) == 0)
```

test_frontend.py

```
# checking in student order history

        self.login_student_1()

        # clicking on canteen
        self.driver.find_elements(By.CLASS_NAME, "e1_242")[0].click()

        # subsection:Order history
        self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Veg. Maggi")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "4")
        assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "30")
        assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "120")
        assert(self.driver.find_elements(By.CLASS_NAME, "f")[0].text == "Sucessful")

        # clicking on my account
        self.driver.find_elements(By.CLASS_NAME, "e1_230")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()
        assert(self.driver.find_elements(By.NAME, "show")[0].text == "Your Canteen bill is Rs. 0")

        # logging out
        self.logout()
```

23 Test23

Testing the functionalities of modify menu and accepting the orders from canteen owner's side and the whole functionality of serving and keeping track of the payment from canteen owner's side. Also, several functionalities from student's side such as adding to cart, book an order and view the canteen dues were tested.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, adding an item to the menu and accepting the order from canteen manager's side and adding certain item from menu to the cart and finally booking the order from student's side.

Test Owner: Divyansh Chhabria.

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: Using selenium framework several items were added in the menu from canteen owner's side, these items were added to cart and booked from student's side. Finally, the order was accepted and the served clarification was made from the canteen owner's side. On checking upon my accounts section from student's side the section showed the required amount there. Hence, successfully executing all the targeted the functionalities.

Command: python test_frontend.py

test_frontend.py

```
        # checking in student order history
        self.login_student_1()
        # clicking on canteen
        self.driver.find_elements(By.CLASS_NAME, "e1_242")[0].click()

        # subsection:Order history
        self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Veg. Maggi")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "4")
        assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "30")
        assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "120")
        assert(self.driver.find_elements(By.CLASS_NAME, "f")[0].text == "Sucessful")

        # clicking on my account
        self.driver.find_elements(By.CLASS_NAME, "e1_230")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()
        assert(self.driver.find_elements(By.NAME, "show")[0].text == "Your Canteen bill is Rs. 120")

        # logging out
        self.logout()
```

24 Test24

Testing the functionalities of clear dues from canteen owner's side.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, adding an item to the menu and accepting the order from canteen manager's

side and adding certain item from menu to the cart and finally booking the order from student's side.
Test Owner: Jhalak Sharma.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Using selenium framework several items were added in the menu from canteen owner's side, these items were added to cart and booked from student's side. Finally, the order was accepted and the served clarification was made from the canteen owner's side. On checking upon my accounts section from student's side the section showed the required amount there. Later upon payment, the dues were cleared from the canteen owner's side.
Command: python test_frontend.py

25 Test25

Testing the functionalities of guest room booking from student's and hall manager's side.
Module Details: This unit contains all the classes, templates, and functions necessary for logging in, entering the check in and check out date and viewing the status of booking from student's side. From hall manager's side, the unit contains all the functions, classes and templates necessary for viewing the requests and rejecting/ accepting them.
Test Owner: Soham Bharambe.
Test Date: [31/03/2023] - [31/03/2023]
Test Results: All tests passed successfully.
Additional Comments: Several request for booking the guest room were made from the student's side for testing. All the request were sent successfully to the hall manager and they could successfully reject/ accept them.

Command: python test_frontend.py

test_frontend.py

```
# booking a guest room
self.driver.find_elements(By.NAME, "checkin_date")[0].send_keys("2023-04-03")
self.driver.find_elements(By.NAME, "checkout_date")[0].send_keys("2023-04-05")
self.driver.find_elements(By.NAME, "submit")[0].click()
assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your request for this
room has been sent to the Hall manager, please contact him for further proceedings.
UPHA team will communicate their confirmation to you.")
```

test_frontend.py

```
# checking guestroom bookings
self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "1")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "April 3, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "April 5, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "Rs.900")
assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "Pending...")
```

test_frontend.py

```
# hall manager checking
assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "s1")
assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "1")
assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "April 3, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "April 5, 2023")
```

test_frontend.py

```
# hall manager accepting the request
self.driver.find_elements(By.CLASS_NAME, "approve")[0].click()

# checking the message and the updated request
assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "The booking has been validated")
assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "s1")
assert(self.driver.find_elements(By.CLASS_NAME, "f")[0].text == "1")
assert(self.driver.find_elements(By.CLASS_NAME, "g")[0].text == "April 3, 2023")
assert(self.driver.find_elements(By.CLASS_NAME, "h")[0].text == "April 5, 2023")
```

test_frontend.py

```
        # student checking if booked
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "1")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "April 3, 2023")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "April 5, 2023")
        assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "Rs.900")
        assert(self.driver.find_elements(By.CLASS_NAME, "e")[0].text == "Booked")
```

26 Test26

Testing the functionalities of booking sports equipment from student's and sport secy's side.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, adding the sports equipment for availability and accepting/ rejecting the borrow request or return request from sports secy's side. For student's side, this unit consists of functions, classes and templates for requesting the borrow/ return of an item and viewing the status of the request.

Test Owner: Divyansh Chhabria.

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: Several request for booking the sports equipment were made from the student's side for testing. All the request were sent successfully to the sports secy's and they could successfully reject/ accept them.

Command: python test_frontend.py

test_frontend.py

```
        # student requesting for equipment
        self.driver.find_elements(By.NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your request
                                        for this item has been sent to the secretary.")
        self.driver.find_elements(By.NAME, "submit")[1].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your request for
                                         returning this item has been sent to the secretary.")
```

test_frontend.py

```
        # student checking if requested
        self.driver.find_elements(By.CLASS_NAME, "e2_9")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "Cricket Bat")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Pending...")
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[1].text == "Badminton")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[1].text == "Pending...")
```

test_frontend.py

```
        # Validating and rejecting
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "s1")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "s1")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "Cricket Bat")
        self.driver.find_elements(By.CLASS_NAME, "reject")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "The request has been rejected")
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "s1")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "s1")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "Badminton")
        self.driver.find_elements(By.CLASS_NAME, "accept")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "The request has been validated")
```

test_frontend.py

```
        # checking the reduce in availability
        self.driver.find_elements(By.CLASS_NAME, "e1_234")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_9")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "2")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "Not available")
        assert(self.driver.find_elements(By.CLASS_NAME, "f")[0].text == "2")

        # testing return of item
        self.driver.find_elements(By.CLASS_NAME, "e2_4")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "Badminton")
        self.driver.find_elements(By.NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "Your request for returning this item has been sent to the secretary.")
        self.driver.find_elements(By.NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "You have already requested for return of this item.")
```

test_frontend.py

```
        # logging into sports secy
        self.login_sports_secy()

        self.driver.find_elements(By.CLASS_NAME, "e1_246")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_5")[0].click()

        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "s1")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "s1")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "Badminton")
        self.driver.find_elements(By.CLASS_NAME, "submit")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "success")[0].text == "The return has been validated")

        self.logout()
```

27 ***Test27***

Testing the functionalities of booking sports court from student's side.

Module Details: This unit contains all the classes, templates, and functions necessary for logging in, entering the check in and check out time and viewing the status of booking of the court from student's side.

Test Owner: Soham Bharambe.

Test Date: [31/03/2023] - [31/03/2023]

Test Results: All tests passed successfully.

Additional Comments: A request for booking the court was made from the student's side for testing. The court was booked successfully and the student could see the status of the booked court.

Command: python test_frontend.py

test_frontend.py

```
        # logging into student_1
        self.login_student_1()

        # booking menu
        self.driver.find_elements(By.CLASS_NAME, "e1_234")[0].click()
        self.driver.find_elements(By.CLASS_NAME, "e2_8")[0].click()

        self.driver.find_elements(By.NAME, "sport")[0].click()
        self.driver.find_elements(By.ID, "CB")[0].click()
        self.driver.find_elements(By.NAME, "checkin_time")[0].send_keys("14:00")
        self.driver.find_elements(By.NAME, "checkout_time")[0].send_keys("16:00")
        self.driver.find_elements(By.NAME, "date")[0].send_keys("2023-04-02")
        self.driver.find_elements(By.NAME, "submit").click()

        # checking
        self.driver.find_elements(By.CLASS_NAME, "e2_19")[0].click()
        assert(self.driver.find_elements(By.CLASS_NAME, "a")[0].text == "Cricket")
        assert(self.driver.find_elements(By.CLASS_NAME, "b")[0].text == "April 2, 2023")
        assert(self.driver.find_elements(By.CLASS_NAME, "c")[0].text == "2 p.m.")
        assert(self.driver.find_elements(By.CLASS_NAME, "d")[0].text == "4 p.m.")
```

4 System Testing

1. **Requirement:** Users can create accounts on the Sign Up page.
   Test Owner: Kriti Arora
   Test Date: 28 March 2023
   Test Results: Test Passed.
   Additional Comments: Only one Canteen Manager, Hall Manager, Mess Manager and Sports Secretary account could be created. The portal did not permit creation of an account with an already taken username.
2. **Requirement:** Users can log in to the portal.
   Test Owner: Kriti Arora
   Test Date: 28 March 2023
   Test Results: Test passed.
   Additional Comments: Users can log in from multiple devices. This is permitted as some users would want to use it from their phone and laptop for using different sections.
3. **Requirement:** If a user has forgotten his/her password, he/she can change the password and log in using the forgot password functionality.
   Test Owner: Kriti Arora
   Test Date: 28 March 2023
   Test Results: Test passed.
   Additional Comments: OTP was successfully sent by the portal to the IIT-K email account of the user. The Resend OTP button was tested.
4. **Requirement:** Users can log out from any page on the portal.
   Test Owner: Kriti Arora
   Test Date: 28 March 2023
   Test Results: Test passed.
   Additional Comments: N/A
5. **Requirement:** A canteen menu is maintained in real-time.
   Test Owner: Pranjal Singh
   Test Date: 29 March 2023
   Test Results: Test passed.
   Additional Comments: Incompletely described dishes (without a name or valid price) are not accepted. If the manager opens a different page instead of submitting the changed menu from the edit mode, a blank entry appears on the manager's interface, to be edited/deleted but it is not shown to students.
6. **Requirement:** Students can add items to their cart and place an order for the dishes in the cart. Orders appear on the incoming orders page on the manager's interface.
   Test Owner: Pranjal Singh
   Test Date: 29 March 2023
   Test Results: Test passed.
   Additional Comments: N/A

7. **Requirement:** If an order is rejected, the student is sent an email and the status as seen on the past orders page is “Failed”. If an order is accepted, it is moved to the pending orders pages of the student and canteen manager and remains there till the order is served.
   Test Owner: Pranjal Singh
   Test Date: 29 March 2023
   Test Results: Test Passed.
   Additional Comments: N/A
8. **Requirement:** The system keeps track of orders till they are served.
   Test Owner: Pranjal Singh
   Test Date: 29 March 2023
   Test Results: Test passed.
   Additional Comments: The order status as visible to students ensures that the canteen owner cannot mark an order as served without the knowledge of the student.
9. **Requirement:** Students can pay for orders on-the-spot.
   Test Owner: Pranjal Singh
   Test Date: 29 March 2023
   Test Results: An error arises when the dues from which the payment amount was deducted is less than the amount to be deducted. But this is unlikely as the dues contain the amount to be deducted, which is added when the order is accepted.
   Additional Comments: The remove button against a pending order need not be clicked only if the payment made button was clicked by the canteen manager, else the remove button needs to be clicked to communicate that the student has decided not to pay immediately.
10. **Requirement:** Canteen dues are maintained as per the policy and can be cleared on payment to the canteen manager.
    Test Owner: Pranjal Singh
    Test Date: 29 March 2023
    Test Results: An error arises when the canteen manager enters an amount more than the dues.
    Additional Comments: N/A
11. **Requirement:** Mess regular menu is showing the menu given by manager
    Test Owner: Labajyoti Das
    Test Date: 28 March 2023
    Test Results: Portal is showing the menu as provided by the manager and students are also able to rate the items provided and it is properly reflected on the manager's side.
    Additional Comments: Students can view the average rating and rate menu items. If they have already rated an item, then making a rating overwrites the previous rating.
12. **Requirement:** Orders are placed upon booking through the portal.
    Test Owner: Pranjal Singh
    Test Date: 28 March 2023
    Test Results: Test passed.
    Additional Comments: The order can only get booked if it is ordered in the time frame for ordering which is given in the page.
13. **Requirement:** Validate order section is working properly and reflecting on the manager's side.

Test Owner: Pranjal Singh
Test Date: 28 March 2023
Test Results: Student is able to click on the ok button on receiving the extra item he or she ordered and it is reflected on the managers side and is moved to the order history section.
Additional Comments: NA

14. Requirement: Order history is updated after the user has validated the order.
Test Owner: Pranjal Singh
Test Date: 28 March 2023
Test Results: Test passed/
Additional Comments: NA

15. Requirement: Application for rebate request.
Test Owner: Labajyoti Das
Test Date: 28 March 2023
Test Results: Students are able to apply for rebate by giving the dates between which the user wants to apply for rebate and can apply for rebate and it is reflected on the manager's side well.
Additional Comments: We also checked for cases like unusual dates like negative difference between to and from dates and it was showing an error message as expected.

16. Requirement: Mess manager can modify the menu.
Test Owner: Labajyoti Das
Test Date: 28 March 2023
Test Results: Manager can successfully update the menu as well as add new items clicking on the respective buttons and as mentioned previously it was properly reflected on the students side.
Additional Comments: The manager can renew the rating of any item by deleting and reading the item.Other than this we all the features of this page and deleting button and submit and they worked as expected.

17. Requirement: Mess manager can modify the extras menu.
Test Owner: Labajyoti Das
Test Date: 29 March 2023
Test Results: Manager can successfully update the menu as well as add new items along with their prices by clicking on the respective buttons and as mentioned previously it was properly reflected on the students side.The time between which the student can order the items is also given here and it is working as previously mentioned.
Additional Comments:NA

18. Requirement: Mess manager can see the items ordered by the students in the extras.
Test Owner: Labajyoti Das
Test Date: 29 March 2023
Test Results:As previously mentioned the view orders page gets reflected on the students giving orders on the extra items.
Additional Comments:The items get removed on the student clicking the ok button on their side from this page as expected.

19. Requirement: Mess manager can see the rating given by the students in the feedback section.

Test Owner: Labajyoti Das
Test Date: 29 March 2023
Test Results:The feedback section is giving the average rating given by the students .
Additional Comments:Please note well that the manager can renew the rating of any item by deleting and reading the item as previously mentioned.

20. Requirement: Managing the rebate requests given by students.
Test Owner: Labajyoti Das
Test Date: 29 March 2023
Test Results:The manager is able to accept as well reject the requests.
Additional Comments:In both accepting as well as rejecting the requests a mail is successfully sent to the student as expected also the mess bill was not taken from the student at that time as shown in the students bill section.

21. Requirement: Manager is able to clear as well as view dues.
Test Owner: Labajyoti Das
Test Date: 29 March 2023
Test Results:Test passed
Additional Comments:We checked that the dues was updated on the students bill section on updating the BDMR as well as on ordering extras as expected.Also on clearing dues it also got accordingly updated on the accounts section.

22. Requirement: Mess Manager is able to clear as well as view dues.
Test Owner: Labajyoti Das
Test Date: 29 March 2023
Test Results:Test passed
Additional Comments:We checked that the dues was updated on the students bill section on updating the BDMR as well as on ordering extras as expected.Also on clearing dues it also got accordingly updated on the accounts section.

23. Requirement: Mess Manager is able to set BDMR.
Test Owner: Labajyoti Das
Test Date: 29 March 2023
Test Results:Test passed
Additional Comments:We checked that the dues was updated on the students bill section on updating the BDMR as previously mentioned and it was also updated on the students accounts section.

24. Requirement: Guest Room Booking request by student
Test Owner: Pranjal Singh
Test Date: 29 March 2023
Test Results: Students are able to book a guest room if the room is available on the requested dates and after a successful request the booking request is sent to the hall manager for approval and is also displayed in the previous booking section of student with correct status . If

a student tries to book a room that is already booked for the requested dates then an error message is displayed.
Additional Comments: We checked all the possible dates and room clashes and the system worked correctly for all scenarios.

25. **Requirement:** Manager's response to requested booking requests.
Test Owner:Pranjal Singh
Test Date: 29 March 2023
Test Results: Manager is able to accept or reject the booking requests. On accepting the booking status is changed to booked and mail is sent to the student and on rejecting also mail is sent to the student.
Additional Comments: NA
26. **Requirement:** Sports Equipment booking requests by student.
Test Owner: Harsh Mohan
Test Date: 29 March 2023
Test Results: Students are able to request for booking the available equipment and the request is sent to the sports secy for approval.
Additional Comments: NA
27. **Requirement:** Sports secy approval for the pending requests.
Test Owner: Harsh Mohan
Test Date: 29 March 2023
Test Results: Sports secy is able to approve/reject the incoming requests. On approval the equipment is added to the Booked equipment section on the students side . After successful issuing of equipment the available quantity is also changed in the booking side.
Additional Comments: NA
28. **Requirement:** Returning of sports equipment
Test Owner: Harsh Mohan
Test Date:29 March 2023
Test Results: Student is able to request to return the equipment and the request is sent to validate return in sports secy's account for his approval. On approval of sports secy the equipment is removed from the booked equipment section of the student.
Additional Comments: NA
29. **Requirement:** Sports court booking
Test Owner: Harsh Mohan
Test Date: 29 March 2023
Test Results:Students are able to book sports courts if it is available for the requested time slot.
Additional Comments: NA
30. **Requirement:** Students can lodge cleaning requests and a list of pending requests is available to the manager.
Test Owner: Kriti Arora
Test Date: 29 March 2023
Test Results: All requests lodged by students were visible to the manager till they were marked completed. Pending and completed requests were visible to students.

Additional Comments: The character limit (200) for the comment was not present and the HTML template was changed.

31. **Requirement:** Students can mark cleaning requests as complete. The request is moved to the past requests page and the current page reloads without the completed cleaning request.
    Test Owner: Kriti Arora
    Test Date: 29 March 2023
    Test Results: Test passed.
    Additional Comments: NA

5 Conclusion

As far as we can see, the testing was sufficiently exhaustive and we do not expect a significant number of errors to arise if the system were deployed. The number of bugs we found and corrected was also significant, indicating that the testing was effective.

There is some scope for further testing in the booking section.

The testing process can be improved by having a larger number of students in the team and dividing the team into groups, so as to reduce the amount of communication between the developers and testers of each feature. There is also a conflict of interest in the testing process if the testers and developers are part of the same group. Completely separating the two groups and the assessments of their work would incentivise testers to think as end-users rather than custodians of the software. In fact, this is standard practice in the certification of safety-critical software, i.e. independent regulators certify systems as safe.
Giving more time to the testing process would also have improved the testing process. However, as we had limited time, implementation was given priority.

Appendix A - Group Log

S No	Date	Agenda	Duration	Venue
1	18/03/23	Meet to discuss the work allocation of Testing and User-Manual.	60 min	RM Building
2	26/03/23	Discussed the work of testing completed along with changes that needed to be made in User-Manual. Also reviewed the details and any doubts of the Test Document.	90 min	RM Building
3	30/03/23	Finalized the User-Manual.	60 min	RM Building
4	01/04/23	Formatted and Merged the Testing Document.	100 min	RM Building