PA2550: Seminar Series in Software Engineering

Group Project: Quality Assurance Activity Report, including unit tests

14-10-2024

Student 1	**Name**	Makarudze, Anna
	Email	anmk24@student.bth.se
	Swedish ID	860630T441
Student 2	**Name**	Mahapatra, Payel
	Email	pamh24@student.bth.se
	Swedish ID	930618T729
Student 3	**Name**	Kotapati, Jammithri
	Email	jako22@student.bth.se
	Swedish ID	20010610T223
Student 4	**Name**	Rokaya, Sanjeeb
	Email	saro24@student.bth.se
	Swedish ID	010702T339
Student 5	**Name**	Arakkal Sudesh, Meenakshi
	Email	meaa24@student.bth.se
	Swedish ID	951124R029

1. **How many unit tests have been written for the project code units? Explain by giving numbers and details about code units and unit tastings. <u>(The code of at least 2 of your unit tests should be part of this report)</u>**

Code for unit testing Login Functionality

```
@pytest.fixture
def valid_login_form(db, guest):
   form = {"username": "elsa@test.com", "password": "pass1234"}
   return form


@pytest.fixture
def missing_email(db):
   form = {"username": "", "password": "pass1234"}
   return form


@pytest.fixture
def missing_password(db):
   form = {"username": "test@123.com", "password": ""}
   return form

def test_missing_email(manager_client, missing_email):
   """Test login with invalid credentials"""
   form = LoginForm(data=missing_email)
   assert form.is_valid() is False
   assert form.errors == {"username": ["This field is required."]}


def test_missing_password(manager_client, missing_password):
   """Test login with invalid credentials"""
   form = LoginForm(data=missing_password)
   assert form.is_valid() is False
```

```
    assert form.errors == {"password": ["This field is required."]}


def test_valid_login_form(valid_login_form, manager_client):
    """Test login with valid details"""
    form = LoginForm(data=valid_login_form)
    # assert form.is_valid()
    assert form.errors == {}
    def test_valid_login(manager_client, valid_login_form, client):

    """Test login with valid credentials"""
    response = client.get(reverse("accounts:login"))
    assert response.status_code == 200
    response = client.post(reverse("accounts:login"), data=valid_login_form)
    assert response.status_code == 302
```

Unit Test for Add Room Functionality by Manager and Front desk staff role

```
@pytest.fixture
def valid_room_form(db):
    return {
        "name": "Queen Room",
        "description": "A nice room to share.",
        "bed_type": "Single",
        "number_of_beds": 4,
        "room_type": "Queen",
        "bathroom":  "Ensuite",
        "room_capacity": 4,
        "price":              500.0,
        "can_be_rented": True,
    }
@pytest.fixture
def front_desk_client(client, front_desk):
```

```
    """Fixture to create a client to simulate what front desk staff can view while browsing the website."""
    client.force_login(front_desk)
    return client


@pytest.fixture
def manager_client(client, manager):
    """
    Fixture to create a client to simulate what the manager/admin can do while browsing the website.
    This user should be able to access everything.
    """
    client.force_login(manager)
    return client



def test_add_room_view_guest(guest_client):
    response = guest_client.get(reverse("rooms:add_room"))
    assert response.status_code == 403


def test_add_room_view_manager(manager_client, valid_room_form):
    response = manager_client.get(reverse("rooms:add_room"))
    assert response.status_code == 200
    response = manager_client.post(reverse("rooms:add_room"), data=valid_room_form)
    assert response.status_code == 302
def test_add_room_view_staff(front_desk_client, valid_room_form):
    response = front_desk_client.get(reverse("rooms:add_room"))
    assert response.status_code == 200
    response = front_desk_client.post(reverse("rooms:add_room"), data=valid_room_form)
    assert response.status_code == 302
```

Test Add Event Functionality by Manager and Front desk staff

```
@pytest.fixture
def valid_event_form(db):
  return {
     "name": "Concert Day",
     "description": "Concert by Ed Sheeran",
     "host": "Payel",
     "venue": "Multisalen",
     "start_date": "2024-12-05",
     "end_date": "2024-12-06",
     "min_participants": 100,
     "max_participants": 150,
     "num_participants": 130,
     "fully_booked": True,
     "price": 560,
     "age_restrictions": "No age restrictions",
     "additional_information": "No food allowed in concert hall",
  }
def test_add_event_view_guest(guest_client):
  response = guest_client.get(reverse("events:add_event"))
  assert response.status_code == 403


def test_add_event_view_staff(front_desk_client, valid_event_form):
  response = front_desk_client.get(reverse("events:add_event"))
  assert response.status_code == 200
  response = front_desk_client.post(
     reverse("events:add_event"), data=valid_event_form
  )
```

```
    assert response.status_code == 302


def test_add_event_view_manager(manager_client, valid_event_form):
    response = manager_client.get(reverse("events:add_event"))
    assert response.status_code == 200
    response = manager_client.post(reverse("events:add_event"), data=valid_event_form)
    assert response.status_code == 302
```

Test Make Room Reservation by a Guest

```
@pytest.fixture
def room(db):
    return Room.objects.create(
        name="Acacia",
        description="A nice single room.",
        photo="image.jpeg",
        bed_type="Double",
        number_of_beds=1,
        room_type="Single",
        bathroom="Shared",
        price=1000.00,
    )
@pytest.fixture
def valid_reservation_rooms(db, room, guest):
    """Fixture where rooms are missing"""
    return {
        "user": guest,
        "number_of_adults": 2,
        "number_of_children": 1,
        "check_in_date": "2024-12-10",
        "check_out_date": "2024-12-12",
        "rooms": [room.id],
```

```
    }
@pytest.mark.django_db
def test_make_reservation_view(guest_client, valid_reservation_rooms):
    response = guest_client.get(reverse("website:make_reservation"))
    assert response.status_code == 200
    response = guest_client.post(
        reverse("website:make_reservation"), data=valid_reservation_rooms
    )
    assert response.status_code == 200
```

Test Case ID	TC_001	**Test Case Description**	Login Functionality in BnB Reservation System		
Created By	Payel	**Reviewed By**	Anna	**Version**	1
QA Tester's Log	All the test cases are passed.				
Tester's Name	Payel	**Date Tested**	6-10-2024	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	Prerequisites:
1	Account should be created for Guest, Manager, and Front-desk Staff.
2	
3	

S #	Test Data
1	Username = elsa@test.com Password = pass1234
2	Username = "" Password = pass1234
3	Username = elsa@test.com Password = ""

Test Scenario	Test the login functionality of the website with different test data.

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Unit Test with valid username and password	200 OK	Assert True	Pass
2	Unit Test with empty username and a valid password	Error Message	Assert True	Pass
3	Unit Test with valid username	Error Message	Assert True	Pass

	and empty password			

Test Case ID	TC_002	**Test Case Description**	Add room functionality by user having Manager and Front-desk Staff Privileges		
Created By	Anna	**Reviewed By**	Jammithri	**Version**	1
QA Tester's Log	All the test cases are passed.				
Tester's Name	Anna	**Date Tested**	6-10-2024	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**
1	Manager and Front-desk Staff should be logged into the system to add rooms

S #	**Test Data**
1	"name": "Queen Room" "description": "A nice room to share" "bed_type": "Single" "number_of_beds": 4 "room_type": "Queen" "bathroom": "Ensuite" "room_capacity": 4 "price": 500.0 "can_be_rented": True

Test Scenario	Test the add room functionality by different users having different privileges.

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed /Suspended**
1	Unit test to add room by a user having the Guest privileges	403 Forbidden	Assert True	Pass
2	Unit Test to add room by a user having Manager privileges	200 OK	Assert True	Pass

3	Unit test to add room by a user having Front-desk Staff privileges	200 OK	Assert True	Pass

Test Case ID	TC_003	**Test Case Description**	Add event functionality by user having Manager and Front-desk staff Privileges		
Created By	Jammithri	**Reviewed By**	Meenakshi	**Version**	1
QA Tester's Log	All the test cases are passed.				
Tester's Name	Jammithri	**Date Tested**	8-10-2024	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**
1	Manager and Front-desk Staff should be logged into the system to add events

S #	**Test Data**
1	"name": "Concert Day" "description": "Concert by Ed Sheeran" "host": "Payel" "venue": "Multisalen" "start_date": "2024-12-05" "end_date": "2024-12-06" "min_participants": 100 "max_participants": 150 "num_participants": 130 "fully_booked": True "price": 560 "age_restrictions": "No age restriction" "additional_information": "No food allowed in concert hall"

Test Scenario	Test the add event functionality by different users having different privileges.

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Unit test to add event by a user having the Guest privileges	403 Forbidden	Assert True	Pass
2	Unit Test to add event by a user having Manager privileges	200 OK	Assert True	Pass

3	Unit test to add event by a user having Front-desk Staff privileges	200 OK	Assert True	Pass

Test Case ID	TC_004	**Test Case Description**	Make room reservation by the Guest		
Created By	Meenakshi	**Reviewed By**	Sanjeeb	**Version**	1
QA Tester's Log	All the test cases are passed.				
Tester's Name	Meenakshi	**Date Tested**	10-10-2024	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**
1	The guest should be signed up and logged before making a room reservation

S #	**Test Data**
1	"user": guest "number_of_adults": 2 "number_of_children": 1 "check_in_date": "2024-12-10" "check_out_date": 2024-12-12 "rooms": [room.id] Room = {name="Acacia" description="A nice single room" photo="image.jpeg" bed_type="Double" number_of_beds=1 room_type="Single" bathroom="Shared" price=1000.00}

Test Scenario	Test the make room reservation functionality by the Guest.

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Unit Test with valid Guest making room reservation	200 OK	Assert True	Pass

2. What percentage of code in your units has been tested by these unit tests?

We have been able to achieve 95% of code coverage for our codebase by running Python unit test cases(**pytest**).

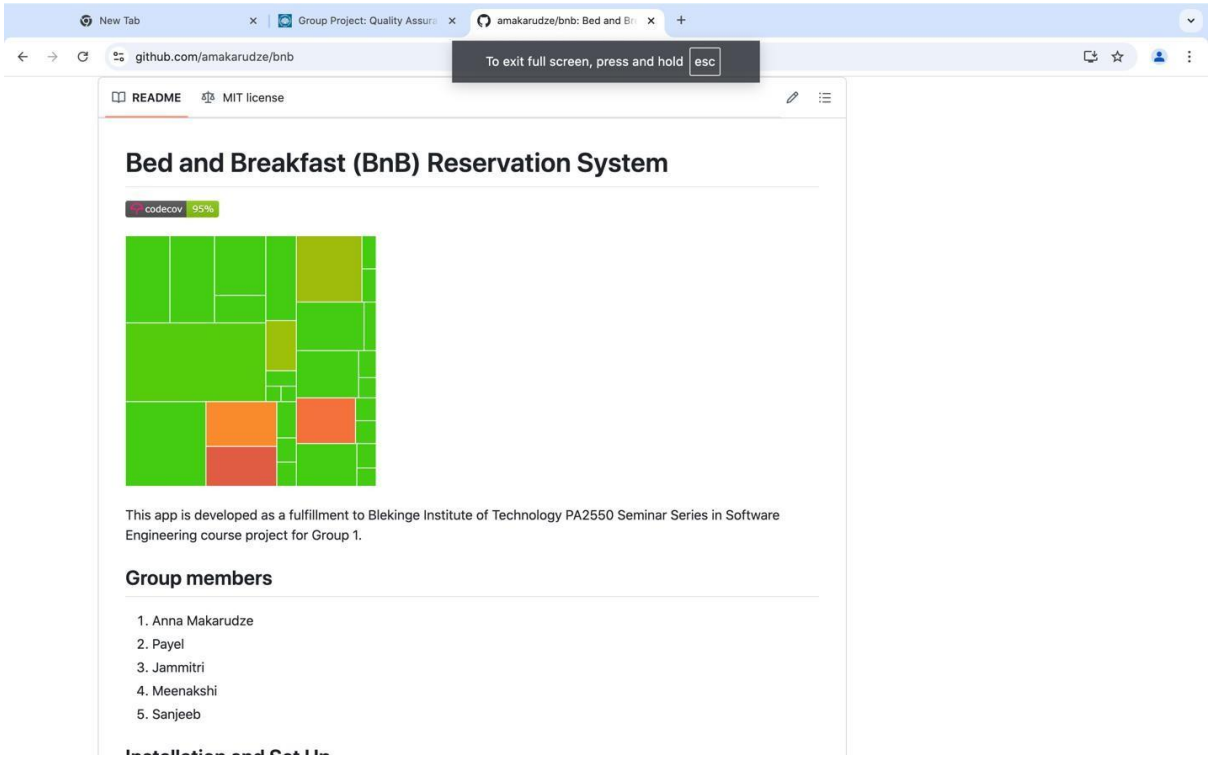

Image 1: GitHub code coverage screenshot. GitHub link: *https://github.com/amakarudze/bnb*

3. **Have you written any integration tests (tests to check the functional correctness of task two or more units as they perform a task)? If yes, how many and provide details. If not, why not?**

We have written one integration test for guests making reservations, starting when they visit the website, search for available rooms, select a room, sign up, and make a reservation.
The code for this test is shown below:

```
import pytest

from django.conf import settings
from django.core import mail
from django.test import RequestFactory
from django.urls import reverse

from pytest_django.asserts import assertRedirects

from accounts.views import send_email
from website.views import make_reservation

@pytest.mark.transactional_db(True)
def test_guest_reservation_process(
    client,
    guest,
    search_form_valid,
    valid_sign_up_form,
    valid_reservation_rooms,
    reservation,
    reservations,
    room,
    rooms,
    available_rooms,
    check_in_date,
    check_out_date,
    number_of_adults,
```

```
    number_of_children,
    upcoming_events,
):
    response = client.get(reverse("website:home"))
    assert response.status_code == 200

    response = client.get(reverse("website:search"), data=search_form_valid)
    assert response.status_code == 200
    assert "rooms" in response.context
    assert len(response.context["rooms"]) == 7

    response = client.get(reverse("website:make_reservation"))
    assert response.status_code == 302
    assertRedirects(response, "/accounts/login/?next=/make_reservation/")

    response = client.get(reverse("accounts:signup"))
    assert response.status_code == 200
    response = client.post(reverse("accounts:signup"), data=valid_sign_up_form)

    to_email = [guest.email]
    send_email(
        "Sign Up Confirmation",
        "Test signup confirmation.",
        settings.DEFAULT_FROM_EMAIL,
        to_email,
    )
    assert len(mail.outbox) == 2
    assert mail.outbox[0].subject, "Signup Confirmation"

    assert response.status_code == 200

    client.force_login(guest)
```

```
factory = RequestFactory()
request = factory.get(reverse("website:make_reservation"))
request.session = {
    "check_in_date": check_in_date,
    "check_out_date": check_out_date,
    "number_of_adults": number_of_adults,
    "number_of_children": number_of_children,
    "rooms": available_rooms,
    "events": upcoming_events,
}
request.user = guest
response = make_reservation(request)

assert response.status_code == 200

response = client.post(
    reverse("website:make_reservation"),
    data=valid_reservation_rooms,
)
assert response.status_code == 302
```

4. Have you prepared and tested any acceptance tests? If yes, provide details. If not, why not?

We have executed a few acceptance tests through manual testing till now. The details are as below.

Test Scenario	**Acceptance Criteria**	**Status**
Test User login functionality	• All users can enter their email address and password • Form validates that email is in the correct format, else displays an error popup message • The password is masked “XXX”	Pass

	• All input fields are required, if not an error message popup is displayed. • The User is logged in after clicking the Submit button	
Test search available rooms functionality	• All users can enter the check-in date and check-out date, number of guests, and number of children in the search bar that is displayed on the homepage. • All available rooms should then be displayed after clicking the Search button. • If the check-in date is earlier than the current date or the check-out date is before the check-in date, the page displays an error message	Pass
Test Make a reservation by a guest	• The guest selects an available room and clicks the Reserve button. • Guest is redirected to a signup page to enter their personal details • Mandatory fields are required to be filled before submission, else an error message pops up. • Guest clicks on the Sign-up button and they are then redirected to the View Room page with reserve button. • Signed up Guest can now click on Reserve button and they are redirected to Reservation Successful page.	Pass

5. How many bugs did you find as your group tested the code? Please share statistics. How many hours of effort during sprints have been spent on bug removal?

We have found around 5-10 bugs during unit testing of different modules. We detected these bugs by preparing thorough test case data that covered all possible scenarios of a functionality. In Python programming, we prepared "**pytest fixtures**" having test data for all possible outcomes of a functionality. We noticed while some test cases passed, others had failed with different test data and this led to us detecting the bugs and finding the resolution for it by fixing and refactoring the main code under test. We have been working in sprints

having two weeks duration (80 hours) and in each sprint, we have spent 16 hours (2 days) approximately on fixing bugs detected through unit tests.

6. **Lastly, how confident are you (as a group) that the product is bug/error-free and ready for the demonstration? (Give a brief answer)**

   Each of our python forms and views have been unit tested thoroughly with all possible test data. Our website is able to perform the tasks that is expected from it and this verification has been done manually as well as through unit testing by each of the team member. We have handled all exception scenarios by displaying proper error message whenever the system receives data or user input that is not expected as part of the requirement. Therefore, we believe our website is bug and error free and ready for demonstration. In the unlikely case that any potential bugs arise which had been missed during our testing, will be addressed further by refining our test suit.