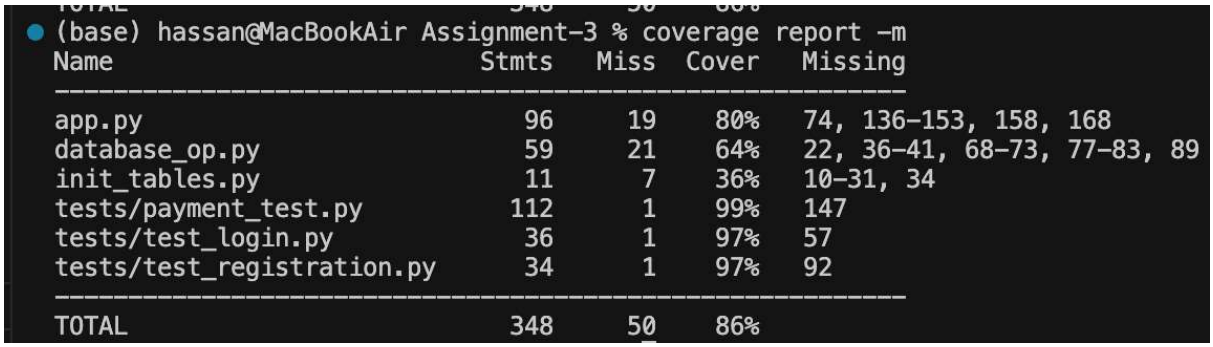
CISC 327 GROUP-51

Code Coverage Analysis - A4

Coverage for Assignment 3 before any updates:



```
(base) hassan@MacBookAir Assignment-3 % coverage report -m
Name                          Stmts   Miss  Cover   Missing
-----------------------------------------------------------
app.py                           96     19    80%   74, 136-153, 158, 168
database_op.py                   59     21    64%   22, 36-41, 68-73, 77-83, 89
init_tables.py                   11      7    36%   10-31, 34
tests/payment_test.py           112      1    99%   147
tests/test_login.py              36      1    97%   57
tests/test_registration.py       34      1    97%   92
-----------------------------------------------------------
TOTAL                           348     50    86%
```

1. **Missing statements and their reasoning**:
    a. ***note**: for all the files tested, the last missing statements are not meant to be tested, as it is used to just call the main function or run the app.route functions in app.py's case. Thus these lines would not be mentioned in the sections below (e.g. app.py line 168, database_op.py line 89, init_tables.py line 34, test_login.py line 63, test_payment line 147, test_registration line 92)

    b. app.py
        i. line 74: missing test case to check if the root route would redirect you to the registration page
        ii. lines 136-153: although the payment test cases tests the validate_payment function, it does not test request.form.get function used to retrieve data from the form in payments.html file
        iii. line 158: it does not check whether redirect for login is properly retrieved

    c. database_op.py and init_tables.py: Initially, there had been no testing files that addressed the database functions, leading to a large amount of missing coverage for database_op.py and init_tables.py
    database_op.py:
        i. line 22: check_exist() is not tested. This function checks to see if the users table exists, if not, it creates one.
        ii. lines 36-41: user_exists() is not tested. This function checks if the user already exists in the database.

iii. lines 68-73: get_all_users() is not tested. This function is responsible for retrieving all users for either testing or verification purposes
iv. lines 77-83: register_user() function is not tested.

init_tables.py:

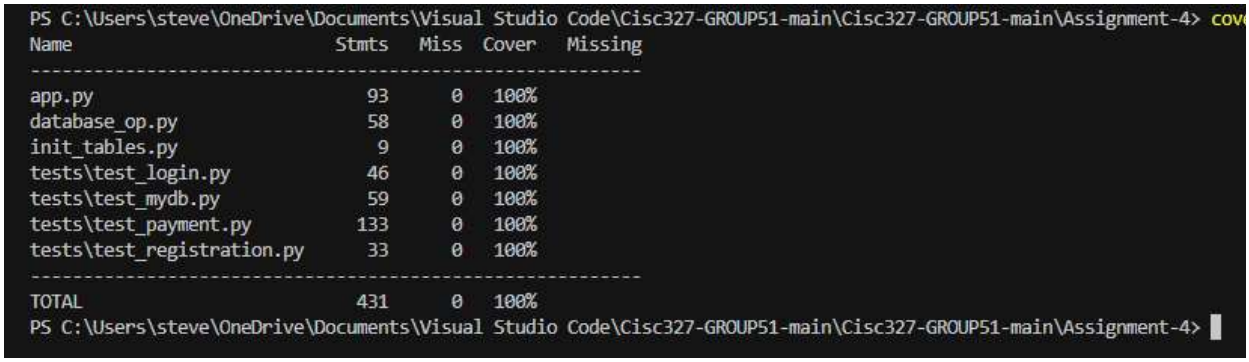
i. lines 10-31: create_user() is not tested

2. Writing additional test scripts and comments (New test scripts at the end of this file for better presentation)

a. ***note**: for all the last missing statements mentioned in part 2 are modified to be exempt from coverage testing with **# pragma: no cover** as for the previously stated reasons (2) and will not be mentioned in this section for changes made
b. **test_payment.py**: added testing mode config for app routing in order to test retrieving data in the form response as well as adding in checking response for clicking the cancel button to make sure it properly calls redirect to /login page
c. **test_login.py**: added a test to test that the root route loads the registration page successfully
d. **test_mydb.py**: was added in order to address the lack of coverage for database focused functions: namely database_op.py and init_tables. Initially, there had been no testing files that addressed the database functions.
   i. **init_tables.py**: is tested through running the corresponding function to the file and validating whether the 'users' table exists. Note: this testing case is a safety net to the coverage module as we already have a function (check_exist in database_op.py). Subsequently, the check_exists function is also covered as well.
   ii. **database_op.py**: given that there was initially no testing file for each of the functions from database_op.py, testing functions for all the features:
      1. **register_user**: The insertion function is tested by attempting to add entries into the database - checks if new records with specific data points can be added successfully.
      2. **check_exist**: This function is tested by verifying that specific entries, once created, are correctly identified as existing in the database. This validation is crucial for confirming database integrity.
      3. **delete_user**: The deletion function is tested by removing particular entries and confirming they no longer exist. This ensures that records can be purged from the database as expected.

4. **Validation**: The validation functions (create table if missing, if user not found, etc) are further tested by verifying various database states and conditions, ensuring that only accurate and consistent data persists across operations.
5. **get_all**: The mass retrieval function is tested by fetching multiple records simultaneously. The test ensures that the data returned in batch queries matches expected entries, confirming the reliability of the retrieval function for large data sets.

Coverage report after adding coverage oriented test cases:



- Upon adding/updating test cases that can be found in test_login.py, test_payment.py and test_mydb.py, all statements were run in the coverage module without any misses, thus giving us a full 100% coverage.

Screenshots of additional scripts:

1. test_payment.py:

```python
@classmethod
def setUpClass(cls):
    # Load payment data from the database
    cls.payment_data = load_payment_data()
    app.config['TESTING'] = True  # Enable testing mode
    app.config['WTF_CSRF_ENABLED'] = False  # Disable CSRF for testing

def test_process_payment_success(self):
    with app.test_client() as client:
        response = client.post('/process_payment', data={
            'email': 'test@example.com',
            'method': 'debit',
            'card_number': '1111111111111111',
            'expiration': '0125',
            'cvc': '001',
            'name_on_card': 'johnsmith',
            'country': 'CA'
        }, follow_redirects=True)
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Payment of CAD $1244.13 successful!', response.data)

def test_process_payment_insufficient_funds(self):
    with app.test_client() as client:
        response = client.post('/process_payment', data={
            'email': 'test@example.com',
            'method': 'debit',
            'card_number': '5555555555555555',
            'expiration': '0529',
            'cvc': '005',
            'name_on_card': 'johnsmith',
            'country': 'CA'
        }, follow_redirects=True)
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Insufficient funds!', response.data)

def test_process_payment_invalid_details(self):
    with app.test_client() as client:
        response = client.post('/process_payment', data={
            'email': 'test@example.com',
            'method': 'debit',
            'card_number': '9999999999999999',  # Invalid card number
            'expiration': '0125',
            'cvc': '001',
            'name_on_card': 'johnsmith',
            'country': 'CA'
        }, follow_redirects=True)
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Invalid payment details!', response.data)

def test_cancel_payment_redirect(self):
    with app.test_client() as client:
        response = client.get('/cancel_payment')
        self.assertEqual(response.status_code, 302)  # HTTP status for redirect
        self.assertIn('/login', response.location)  # Check redirection to /login
```

2. test_login.py:

```
# Test the root route to ensure it loads the registration page successfully
def test_home_page_loads(self):
    response = self.app.get('/')
    self.assertEqual(response.status_code, 200)
    self.assertIn(b'Creating your account', response.data)
```

3. For test_mydb.py, all the test cases are new so you can refer to the file itself for the new scripts since it was created from scratch.