Lab 11 :
Part A:

Test Plan
Feature 1:
Login Page
Sample Cases for UAT:

- Redirects to home page when user information is verified
- displays message error, and re-render the login page if information is not verified

acceptance criteria:

- user cannot log in without filling out all required fields (username and password)
- user cannot log in with incorrect information

description of test data:

description of test environment:

description of test results:

Feature 2:
Register Page
Sample Cases for UAT:

- if password confirmation does not match, display error message, and then re-render register page
- if register is successful then redirect to the login page

acceptance criteria:

- user must fill out all fields (username, password)
- will not accept username that already exists
- information is entered into the user database

description of test data:

description of test environment:

description of test results:

Feature 3:
Logout Functionality
Sample Cases for UAT:

- when user chooses to logout, their session ends, and redirect to the landing page

acceptance criteria:

- once logged out, message that user has been logged out is displayed
- landing page rendered correctly after this
- user session ended

description of test data:

description of test environment:

description of test results:

Part B:

```
describe('*Login*', () => {
 // Sample test case given to test / endpoint.
 it('Negative : /login. Checking empty name', done => {
   chai
     .request(index)
     .get('/login')
     .end((err, res) => {
       expect(res).to.have.status(200);
       expect(res.body.status).to.equals('Username cannot be empty');
       assert.notEqual(res.body.username, null);
       done();
     });
     it('Positive: /login. username in database', done => {
     chai
     .request(index)
     .get('/login')
     .end((err, res) => {
       expect(res).to.have.status(200);
       expect(res.body.status).to.equals('success');
       done();
     });
 });
```

```
it('positive: /login - Login successful', async () => {
  const uniqueUser = `TestUser_${Date.now()}`;
  const registrationRes = await chai
    .request(server)
    .post('/register')
    .send({ username: uniqueUser, password: 'test' });
   expect(registrationRes).to.have.status(200);
  expect(registrationRes.body.status).to.equals('success');
  assert.strictEqual(registrationRes.body.message, 'User registered successfully.');
   const loginRes = await chai
    .request(server)
```

```
    .post('/login')
    .send({ username: uniqueUser, password: 'test' });
   expect(loginRes).to.have.status(200);
  expect(loginRes.body.status).to.equals('success');
  assert.strictEqual(loginRes.body.message, 'Welcome!');
   await dbt.none('DELETE FROM users WHERE username = $1', [uniqueUser]);
});
 it('negative: /login - User not found', async () => {
  const nonExistentUser = `NonExistentUser_${Date.now()}`;
  const response = await chai
    .request(server)
    .post('/login')
    .send({ username: nonExistentUser, password: 'password' });
   expect(response).to.have.status(400);
  expect(response.body.status).to.equals('error');
  assert.strictEqual(response.body.error, 'Incorrect username or password. If you do
not have an account, please register.');
});
 it('positive: /register - Successful registration', async () => {
  const uniqueUser = `TestUser_${Date.now()}`;
  const response = await chai
    .request(server)
    .post('/register')
    .send({ username: uniqueUser, password: 'test' });
   expect(response).to.have.status(200);
  expect(response.body.status).to.equals('success');
  assert.strictEqual(response.body.message, 'User registered successfully.');
   await dbt.none('DELETE FROM users WHERE username = $1', [uniqueUser]);
});
 it('negative: /register - Duplicate username', async () => {
  const duplicateUser = 'DuplicateUser';
  // Register the user once
  const firstRegistration = await chai
    .request(server)
    .post('/register')
    .send({ username: duplicateUser, password: 'password' });
   const secondRegistration = await chai
    .request(server)
    .post('/register')
    .send({ username: duplicateUser, password: 'password' });
   expect(secondRegistration).to.have.status(400);
  expect(secondRegistration.body.status).to.equals('error');
```

```
   assert.strictEqual(secondRegistration.body.message, 'Registration failed. Username
already exists.');
    await dbt.none('DELETE FROM users WHERE username = $1', [duplicateUser]);
   });

});
```