IT 313 Software Engineering

Hospital Management System

Unit testing

Group: 15

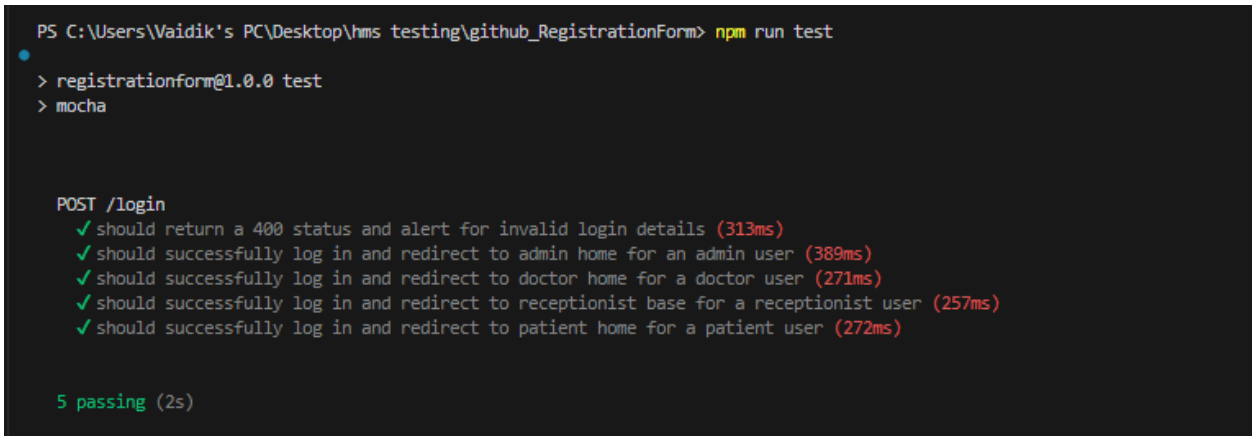
Instructor: Prof. Saurabh Tiwari

Teaching Assistant: Harshita Tripathi

Unit testing using mocha and chai:

Mocha is a widely used JavaScript test framework running on NodeJS and browsers. Chai is an assertion library that is used chiefly alongside Mocha.

1. Login



Test cases:

1. If the inserted password is wrong, it shows an alert and the user won't be able to log in.
2. The system checks the database. If the user is saved as an admin, it will redirect to the admin home page.
3. The system checks the database. If the user is saved as a doctor, it will redirect to the doctor home page.
4. The system checks the database. If the user is saved as a receptionist, it will redirect to the receptionist home page.
5. The system checks the database. If the user is saved as a patient, it will redirect to the patient home page.

Code

```
const chai = require('chai');
const chaiHttp = require('chai-http');
const expect = chai.expect;
chai.use(chaiHttp);
```

```
describe('POST /login', () => {
  const host = http://localhost:3000;

  it('should return a 400 status and alert for invalid login details',
async () => {
    const res = await chai
      .request(host)
      .post('/login')
      .send({ email: 'invalid-email@example.com', password:
'invalid-password' });

    expect(res).to.have.status(400);
    expect(res.text).to.include('Invalid login details');
  });
  it('should successfully log in and redirect to admin home for an
admin user', async () => {
    const res = await chai
      .request(host)
      .post('/login')
      .send({ email: '202101428@daiict.ac.in', password: 'Vaidik@1234' });

    expect(res).to.redirect;
    expect(res.redirects[0]).to.include('admin_home');
  });

  it('should successfully log in and redirect to doctor home for a
doctor user', async () => {
    const res = await chai
      .request(host)
      .post('/login')
      .send({ email: 'vaidikp9@gmail.com', password: 'Vaidik@1234' });

    expect(res).to.redirect;
    expect(res.redirects[0]).to.include('new_doc_home');
  });

  it('should successfully log in and redirect to receptionist base for a
receptionist user', async () => {
    const res = await chai
```

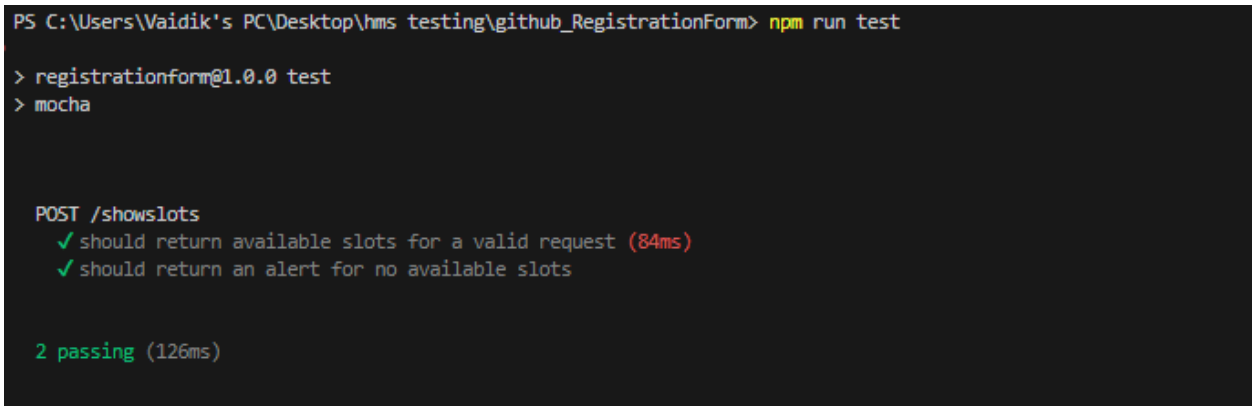
```
      .request(host)
      .post('/login')
      .send({ email: 'scpatel507@gmail.com', password: 'Vaidik@1234' });

    expect(res).to.redirect;
    expect(res.redirects[0]).to.include('receptionist_base');
  });

  it('should successfully log in and redirect to patient home for a
patient user', async () => {
    const res = await chai
      .request(host)
      .post('/login')
      .send({ email: '202101217@daiict.ac.in', password: 'Ab@12345' });

    expect(res).to.redirect;
    expect(res.redirects[0]).to.include('patient_home');
  });
});
```

2. Appointment booking



Test cases:

1. If slots are available for the selected doctor, the patient will be able to book an appointment.
2. If slots are not available, it will show an alert for no available slots.

Code

```
const chai = require('chai');
const chaiHttp = require('chai-http');
const expect = chai.expect;
chai.should();
chai.use(chaiHttp);

describe('POST /showslots', () => {
  const host = http://localhost:3000;

  it('should return available slots for a valid request', async () => {
    // Mock data for a valid request
    const mockToken =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTY4M2U5
NjA1MGQwOTgwY2M0M2VjYzUiLCJpYXQiOjE3MDEzMzMxMTN9.Y
T7zBYftspFAiJ_vdUi9C88wsga7piVg5GEkIJ6Vi14'; // Replace with a
valid JWT token
    const mockUser = { _id: '65683e96050d0980cc43ecc5' };
    const mockDate = '2023-12-01';
    const mockDoctor = '65684063t0';
```

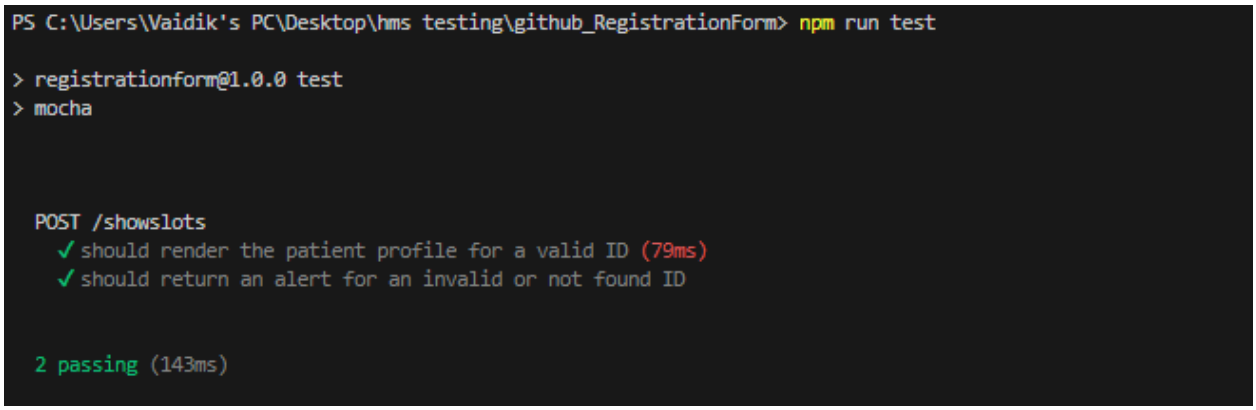
```
    chai
      .request(host)
      .post('/showslots')
      .set('Cookie', jwt=${mockToken})
      .send({ date: mockDate, doctor: mockDoctor })
      .end((err, res) => {
        expect(res).to.have.status(400); // Assuming a successful
response status code
            });
  });

  it('should return an alert for no available slots', async () => {
    // Mock data for a request with no available slots
    const mockToken =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTY4M2U5
NjA1MGQwOTgwY2M0M2VjYzUiLCJpYXQiOjE3MDEzMzMxMTN9.Y
T7zBYftspFAiJ_vdUi9C88wsga7piVg5GEkIJ6Vi14';

  const mockUser = { _id: '65683e96050d0980cc43ecc5' };
  const mockDate = '2023-12-01';
  const mockDoctor = '65684063t0';
    chai
      .request(host)
      .post('/showslots')
      .set('Cookie', jwt=${mockToken})
      .send({ date: mockDate, doctor: mockDoctor })
      .end((err, res) => {
        expect(res).to.have.status(400); // Assuming a status code
indicating no available slots
        expect(res.text).to.include('No Slots are available on this
date');

      });
  });
});
```

3. Search the patient profile



Test case:

1. If the user inserts a valid patient ID, the patient's profile will be shown.
2. If the inserted ID is wrong, an alert will be shown for an invalid ID.

Code

```
const chai = require('chai');
const chaiHttp = require('chai-http');
const expect = chai.expect;
chai.should();
chai.use(chaiHttp);

describe('POST /doc_search_patient', () => {
  const host = http://localhost:3000;

  it('should render the patient profile for a valid ID', async () => {
    // Mock data for a valid request
    const mockUser = { /* mock doctor user data */ };
    const mockPatientId = 'g65683e961';

    chai
      .request(host)
      .post('/doc_search_patient')
      .set('Cookie',
'jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTY4NDA2MzVkOGQ4M
```

```
Dk0NWQ1YTVmNWIiLCJpYXQiOjE3MDEzMzMxMTJ9.QvHaiAcirJ0F_CDuUDmJC9H8
PZMShqVFH6D4Poi5M_k') // Replace with a valid doctor JWT token
      .send({ id: mockPatientId })
      .end((err, res) => {
        expect(res).to.have.status(200); // Assuming a successful response status code
        expect(res.text).to.include('doc_search_profile'); // Assuming the profile
template is rendered
     });

  it('should return an alert for an invalid or not found ID', async () => {
    // Mock data for a request with an invalid or not found ID
    const mockUser = { /* mock doctor user data */ }; // Replace with mock doctor
user data
    const mockInvalidPatientId = 'invalidPatientId'; // Replace with an invalid patient
ID

    chai
      .request(host)
      .post('/doc_search_patient')
      .set('Cookie',
'jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTY4NDA2MzVkOGQ4M
Dk0NWQ1YTVmNWIiLCJpYXQiOjE3MDEzMzMxMTJ9.QvHaiAcirJ0F_CDuUDmJC9H8
PZMShqVFH6D4Poi5M_k')
      .send({ id: mockInvalidPatientId })
      .end((err, res) => {
        expect(res).to.have.status(400); // Assuming a status code indicating ID not
found
        expect(res.text).to.include('ID Not Found');
          });
  });

});
```

4. Apply for leave

```
PS C:\Users\Vaidik's PC\Desktop\hms testing\github_RegistrationForm> npm run test

> registrationform@1.0.0 test
> mocha



  POST /receptionist_leave
    ✓ should apply for leave successfully with valid dates (94ms)
    ✓ should return an alert for invalid dates


  2 passing (138ms)
```

Test case:

1. If the dates are valid then the leave application will be successfully applied.
2. If the leave end date comes before the leave beginning date an alert message will be shown for the invalid dates.

Code

```
const chai = require('chai');
const chaiHttp = require('chai-http');
const expect = chai.expect;
chai.should();
chai.use(chaiHttp);

describe('POST /receptionist_leave', () => {
  const host = http://localhost:3000;

  it('should apply for leave successfully with valid dates', async () => {
    // Mock data for a valid request
    const mockUser = { _id: '656847915d8d80945d5a5f7b', Name: 'sapana', ID:
'D656847911' }; // Replace with mock receptionist user data
    const mockStartDate = '2023-12-01';
    const mockEndDate = '2023-12-05';
    const mockReason = 'Vacation';

    chai
      .request(host)
      .post('/receptionist_leave')
```

```
      .set('Cookie',
jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTY4NDc5MTVkO
GQ4MDk0NWQ1YTVmN2IiLCJpYXQiOjE3MDEzMzMxMTN9.O0nf_jFIqVf_l56H
_YKzZY7kl5_gTqnDe4Hl4OHZXbw) // Replace with a valid receptionist JWT
token
      .send({ startDate: mockStartDate, endDate: mockEndDate, reason:
mockReason })
      .end((err, res) => {
        expect(res).to.have.status(400);
        expect(res.text).to.include('Applied successfully.');

      });
  });

  it('should return an alert for invalid dates', async () => {
    // Mock data for a request with invalid dates
    const mockUser = { _id: '656847915d8d80945d5a5f7b', Name: 'sapana', ID:
'D656847911' }; // Replace with mock receptionist user data
    const mockInvalidStartDate = '2023-12-05';
    const mockInvalidEndDate = '2023-12-01';
    const mockReason = 'Vacation';
    chai
      .request(host)
      .post('/receptionist_leave')
      .set('Cookie',
jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTY4NDc5MTVkO
GQ4MDk0NWQ1YTVmN2IiLCJpYXQiOjE3MDEzMzMxMTN9.O0nf_jFIqVf_l56H
_YKzZY7kl5_gTqnDe4Hl4OHZXbw)
    .send({ startDate: mockInvalidStartDate, endDate: mockInvalidEndDate,
reason: mockReason })
      .end((err, res) => {
        expect(res).to.have.status(400); // Assuming a status code indicating
invalid dates
        expect(res.text).to.include('Please Enter Valid Date.');
            });
  });


});
```