Software Engineering (IT314)

Lab-8

Grp-21

Renting System

Lab Task: Choosing the module, Writing the tests, Our Learnings

Choosing the module :

The backend signup (login) code we are using to build the tests is our own, and we wrote it since it would help us test the project and utilise it to advance it.

Node JS file (class code)

```
const Lender = require('../models/Lender.js');
const express = require('express');
const router = express.Router();
const { check, validationResult } =
require('express-validator');

router.post(
    '/register',
    [
        check('firstname', 'Enter a valid
Firstname').not().isEmpty(),
        check('lastname', 'Enter a valid
Lirstname').not().isEmpty(),
        check('email', 'Enter a valid Email Address').isEmail(),
        check('phoneno', 'Enter a valid Phone Number').isLength
({min: 10}),
        check('address', 'Enter a valid
Address').not().isEmpty(),
```

```
        check('password', 'Enter a valid Firstname').isLength
({min: 8})
    ],

    async (req, res) => {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.send({
                error: true,
                msg: errors.errors[0].msg,
            }).status(600);
        }

        try {
            let lender = await Lender.findOne({
                email: req.body.email,
            });
            if (lender) {
                return res.send({
                    error: true,
                    msg: 'User already exists',
                });
            }

            lender = new Lender({
                firstname: req.body.firstname,
                lastname: req.body.lastname,
                email: req.body.email,
                phone_no: req.body.phone_no,
                address: req.body.address,
                password: req.body.password,
                username: '',
                requestforaddress: [],
                productlist: [],
```

```
                myorder: [],
            });

            const salt = await bcrypt.genSalt(10);
            const hashedPassword = await
bcrypt.hash(req.body.password, salt);

            lender.password = hashedPassword;

            const savedLender = await lender.save();
            res.send({
                error: false,
                userid: lender._id,
            }).status(400);
        } catch (err) {
            console.log(err);
            res.send({
                error: true,
                msg: err.message,
            }).status(501);
        }
    }

);

router.post(
    '/login',
    [
        check('email', 'Enter a valid Email Address').isEmail(),
        check('password', 'Enter a valid Firstname').isLength
({min: 8})
    ],

    async (req, res) => {
```

```
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.send({
                error: true,
                msg: errors.errors[0].msg,
            }).status(601);
        }

        try {
            const lender = await Lender.findOne({
                email: req.body.email,
                phone_no: req.body.phone_no,
            });
            if (!lender) {
                return res.send({
                    error: true,
                    msg: 'User is not registered',
                });
            }

            const validPassword = await bcrypt.compare(
                req.body.password,
                lender.password
            );
            if (!validPassword) {
                return res.send({
                    error: true,
                    msg: 'Password is not valid',
                }).status(300);
            }

            const token = jwt.sign(
                {
                    _id: lender._id,
```

```
                },
                process.env.TOKEN_SECRET
            );
            res.header('auth_token', token).send({
                error: false,
                auth_token: token,
            }).status(400);
        } catch (err) {
            console.log(err);
            res.send({
                error: true,
                msg: err.message,
            }).status(501);
        }
    }
);

module.exports = router;
```

Test Code Code:

```
const chai = require("chai")
const chaiHttp = require("chai-http")
const app = require("../routes/LenderTest");
const expect = chai.expect;
chai.use(chaiHttp);
//Test case 1
describe('localhost:3001',()=>{
    describe('POST /register',()=>{
        it('Case 1',async()=>{
            const res = await
chai.request(app).post("http://localhost:3001/lender/register");
            expect(res).to.have.status(600);
        })
    })
```

```
})

//Test case 2
describe('localhost:3001',()=>{
    describe('POST /register',()=>{
        it('Case 2',async()=>{
            const res = await
chai.request(app).post('http://localhost:3001/lender/register').
send({

firstname:"John",email:"John@email.com",phoneno:1234567890,addes
s:"DAIICT",password:"PASSWORD"
            });
            expect(res).to.have.status(200);
            done();
        })
    })
})

//Test case 3
describe('localhost:3001',()=>{
    describe("Login with invalid inputs",()=>{
        it('Case 3',async()=>{
            const res = await
chai.request(app).post('http://localhost:3001/register');
            expect(res).to.have.status(601);
            done();
        })
    })
})

//Test case 4
describe('localhost:3001',()=>{
```

```
    describe("Login with valid inputs but incorrect
credentials",()=>{
        it('Case 4',async()=>{
            const res = await
chai.request(app).post('http://localhost:3001/register').send({e
mail:"John@email.com",password:"password"});
            expect(res).to.have.status(300);
            done();
        })
    })
})

//Test cases
describe('localhost:3001',()=>{
    describe("Login with valid inputs and correct
credentials",()=>{
        it('Case 5',async()=>{
            const res = await
chai.request(app).post('http://localhost:3001/register').send({e
mail:"John@email.com",password:"PASSWORD"});
            expect(res).to.have.status(200);
            done();
        })
    })
})
```

Results:

```
  http://10.200.8.251:8080
    POST /signin
      1) Should success if credential is valid
    POST /signin
      2) Should success if credential is valid
    POST /signup
      3) Should create new User


  0 passing (6s)
  3 failing

  1) http://10.200.8.251:8080
       POST /signin
         Should success if credential is valid:
     Error: Timeout of 2000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a Promise, ensure it resolves.
m_6\temp\mochatest\test\appTest.js)
      at listOnTimeout (node:internal/timers:573:17)
      at process.processTimers (node:internal/timers:514:7)

  2) http://10.200.8.251:8080
       POST /signin
         Should success if credential is valid:
     Error: Timeout of 2000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a Promise, ensure it resolves.
m_6\temp\mochatest\test\appTest.js)
      at listOnTimeout (node:internal/timers:573:17)
      at process.processTimers (node:internal/timers:514:7)

  3) http://10.200.8.251:8080
       POST /signup
         Should create new User:
     Error: Timeout of 2000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a Promise, ensure it resolves.
m_6\temp\mochatest\test\appTest.js)
      at listOnTimeout (node:internal/timers:573:17)
      at process.processTimers (node:internal/timers:514:7)
```

```
> mochatest@1.0.0 test
> mocha



  http://10.200.8.251:8080
    POST /signin
      ✓ Should succeed if credential is valid
    POST /signin
      ✓ Should succeed if credential is valid
    POST /signup
      ✓ Should create new user


  3 passing (6ms)
```