Unit testing In White Box Testing

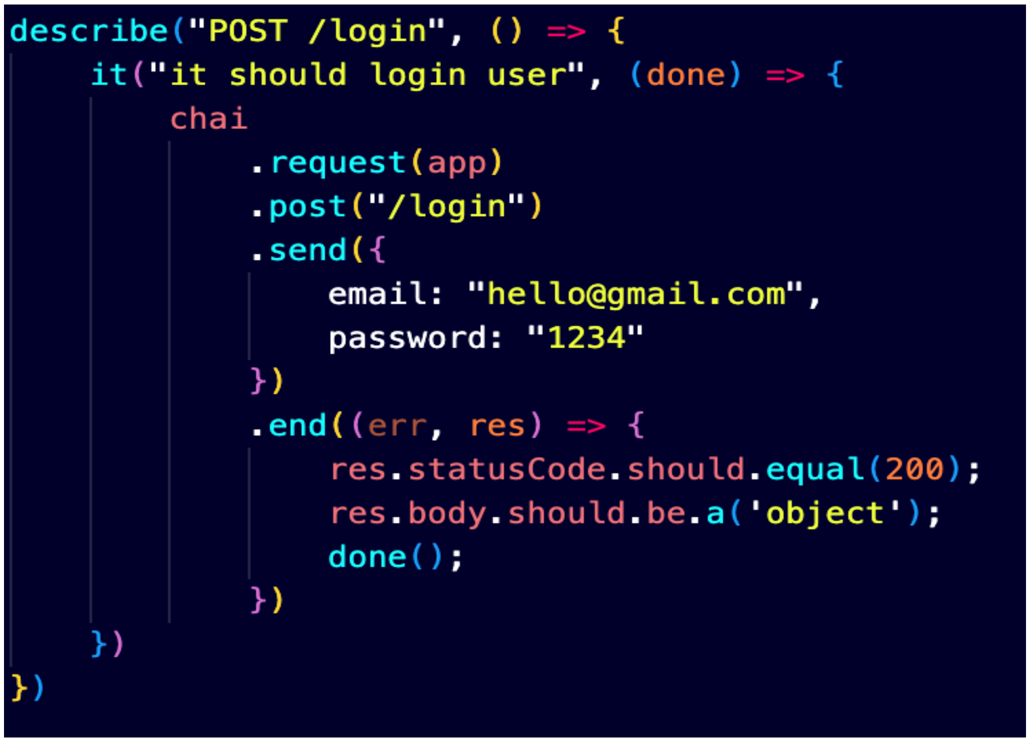
Using Chai and Mocha.js

Unit testing of the javascript code was done using the Mocha.js and Chai libraries. They provide a practical means of testing all the built-in API services and routes. During this testing, other possible issues were identified, including the database returning NULL values and taking the wrong data type into account, among others.

These mistakes were quickly fixed, which contributed to the robustness of our code.

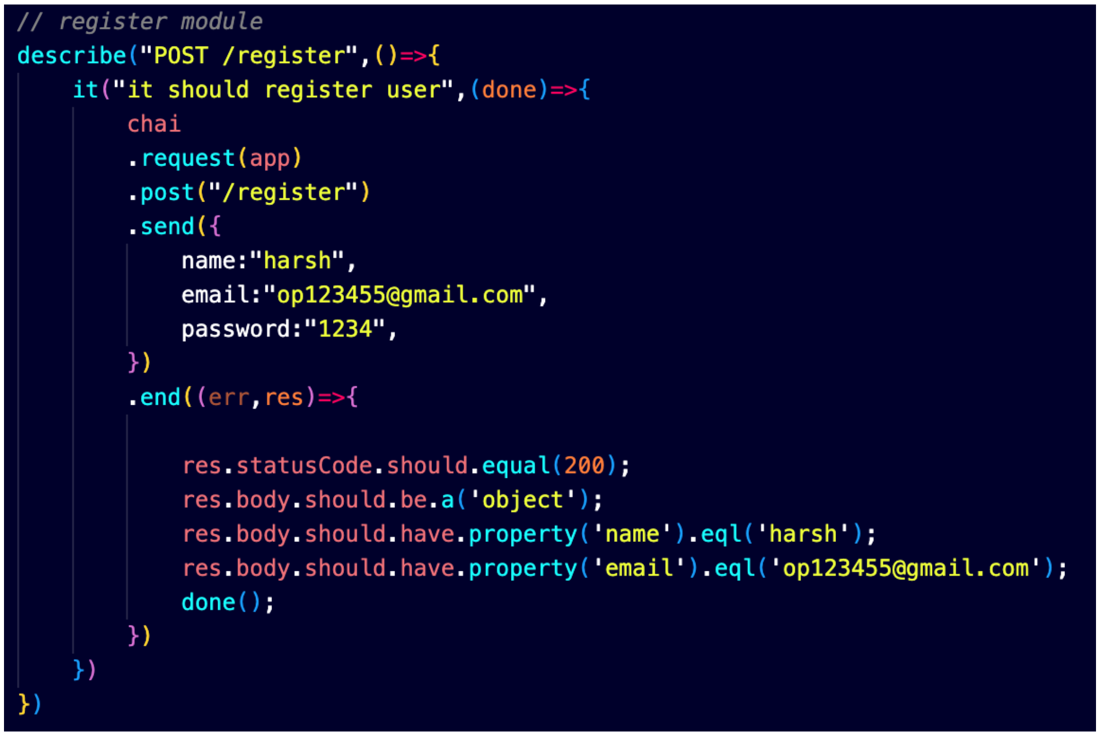
1. For User functionalities

a. Checking for successful login with valid credentials.



```
describe("POST /login", () => {
    it("it should login user", (done) => {
        chai
            .request(app)
            .post("/login")
            .send({
                email: "hello@gmail.com",
                password: "1234"
            })
            .end((err, res) => {
                res.statusCode.should.equal(200);
                res.body.should.be.a('object');
                done();
            })
    })
})
```

b. Checking for successful registration with valid inputs.

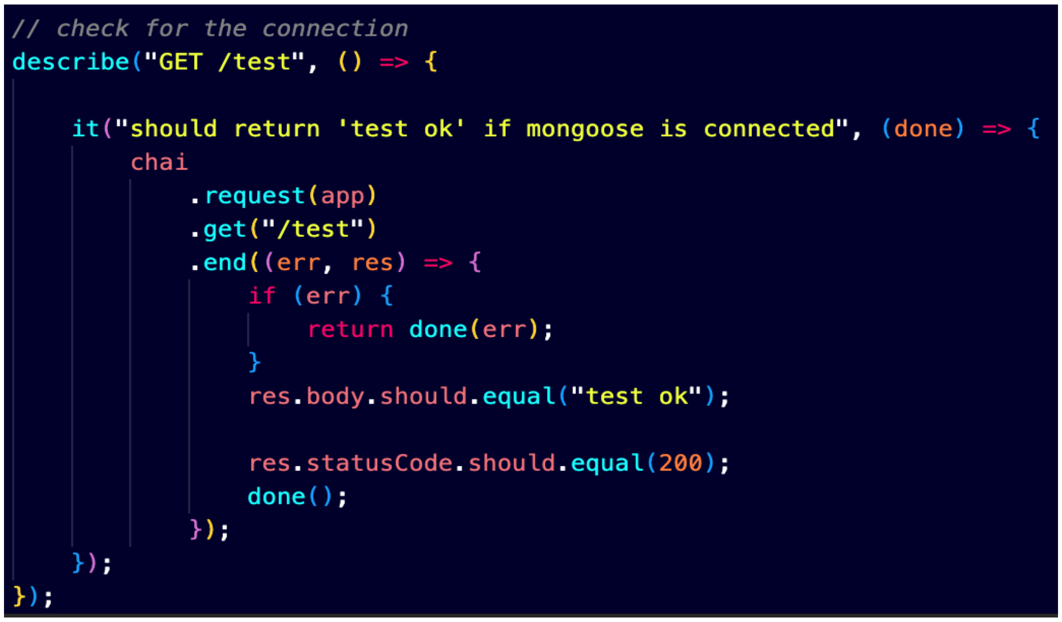


```
// register module
describe("POST /register",()=>{
    it("it should register user",(done)=>{
        chai
        .request(app)
        .post("/register")
        .send({
            name:"harsh",
            email:"op123455@gmail.com",
            password:"1234",
        })
        .end((err,res)=>{

            res.statusCode.should.equal(200);
            res.body.should.be.a('object');
            res.body.should.have.property('name').eql('harsh');
            res.body.should.have.property('email').eql('op123455@gmail.com');
            done();
        })
    })
})
```

2. For connections

a. Checking for successful connection mongoose connection with db.



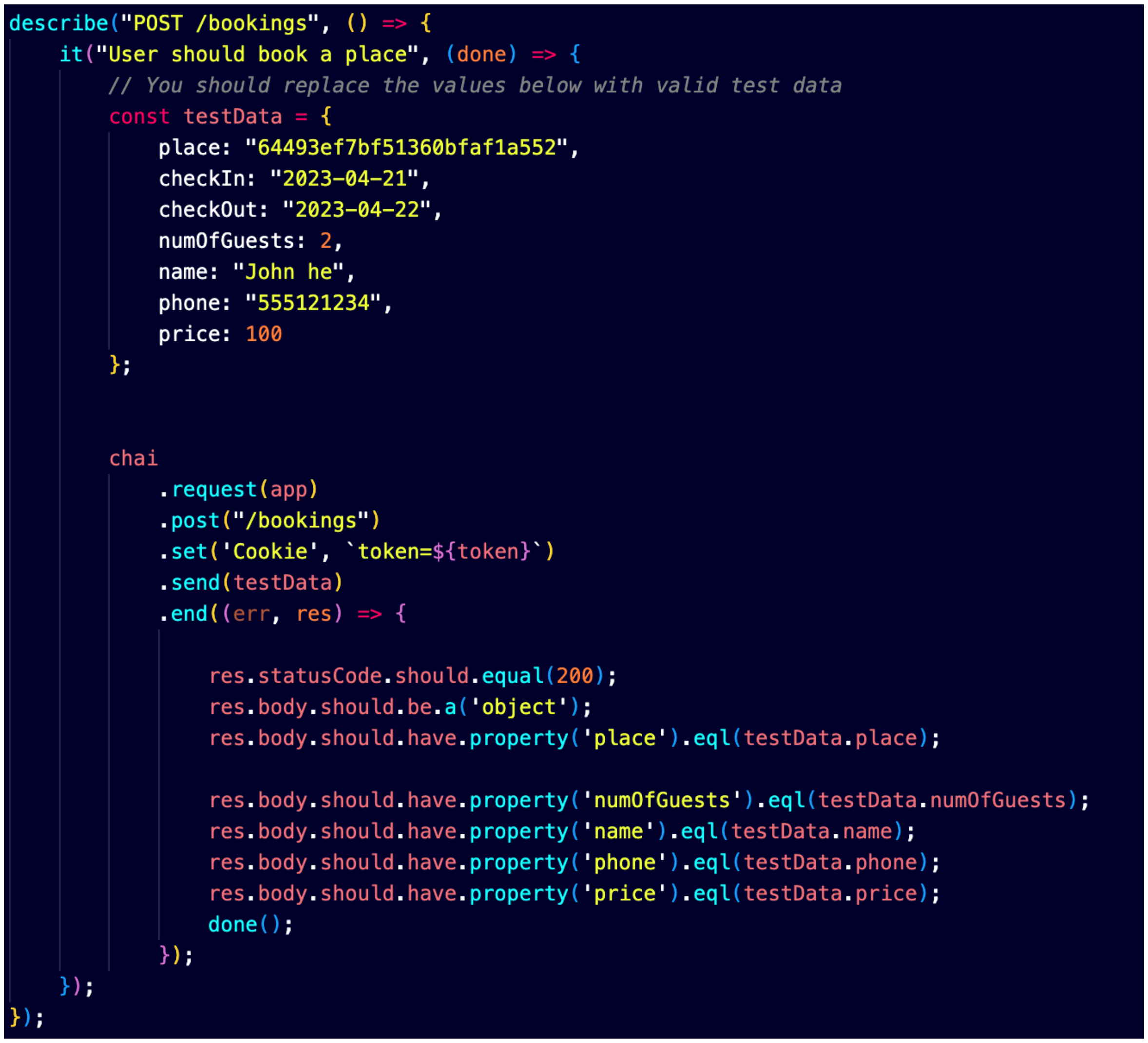
```
// check for the connection
describe("GET /test", () => {

    it("should return 'test ok' if mongoose is connected", (done) => {
        chai
            .request(app)
            .get("/test")
            .end((err, res) => {
                if (err) {
                    return done(err);
                }
                res.body.should.equal("test ok");

                res.statusCode.should.equal(200);
                done();
            });
    });
});
```

3. **For booking functionalities**

   a. Checking for successfully booking a place with valid credentials.



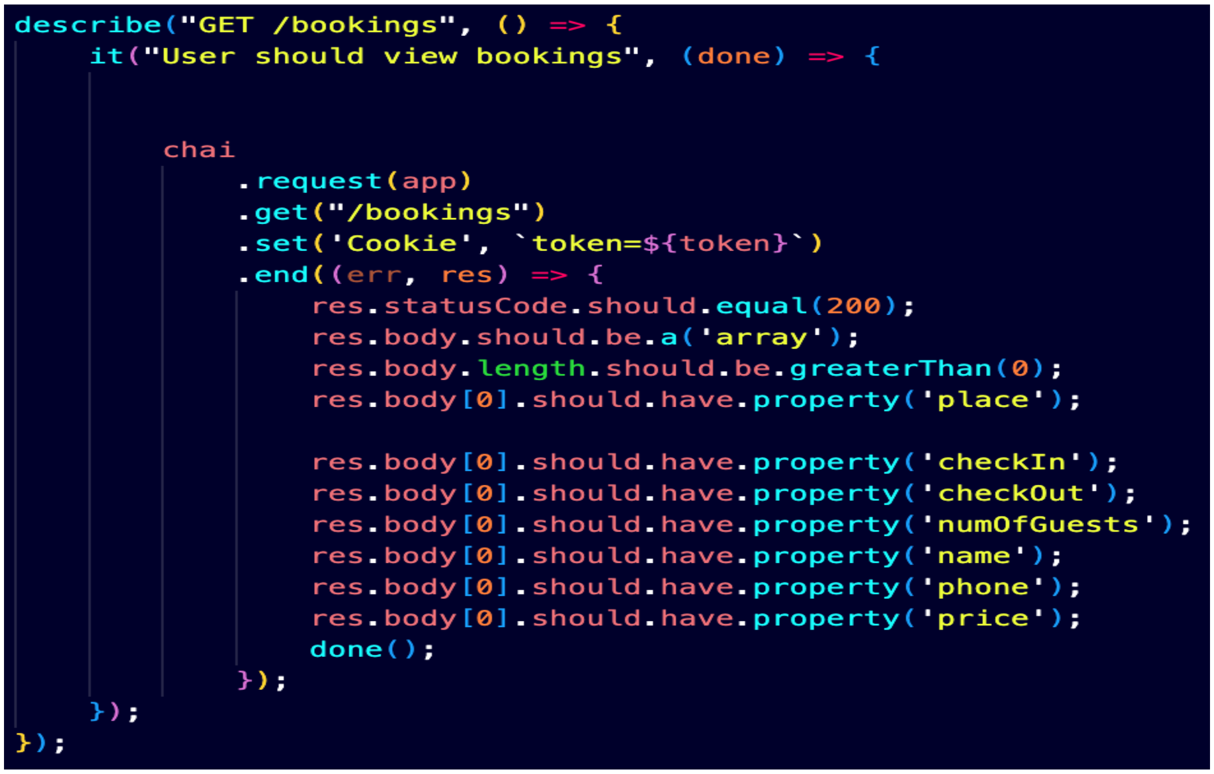
```
describe("POST /bookings", () => {
    it("User should book a place", (done) => {
        // You should replace the values below with valid test data
        const testData = {
            place: "64493ef7bf51360bfaf1a552",
            checkIn: "2023-04-21",
            checkOut: "2023-04-22",
            numOfGuests: 2,
            name: "John he",
            phone: "555121234",
            price: 100
        };


        chai
            .request(app)
            .post("/bookings")
            .set('Cookie', `token=${token}`)
            .send(testData)
            .end((err, res) => {

                res.statusCode.should.equal(200);
                res.body.should.be.a('object');
                res.body.should.have.property('place').eql(testData.place);

                res.body.should.have.property('numOfGuests').eql(testData.numOfGuests);
                res.body.should.have.property('name').eql(testData.name);
                res.body.should.have.property('phone').eql(testData.phone);
                res.body.should.have.property('price').eql(testData.price);
                done();
            });
    });
});
```

b. Checking for successfully viewing the bookings.\

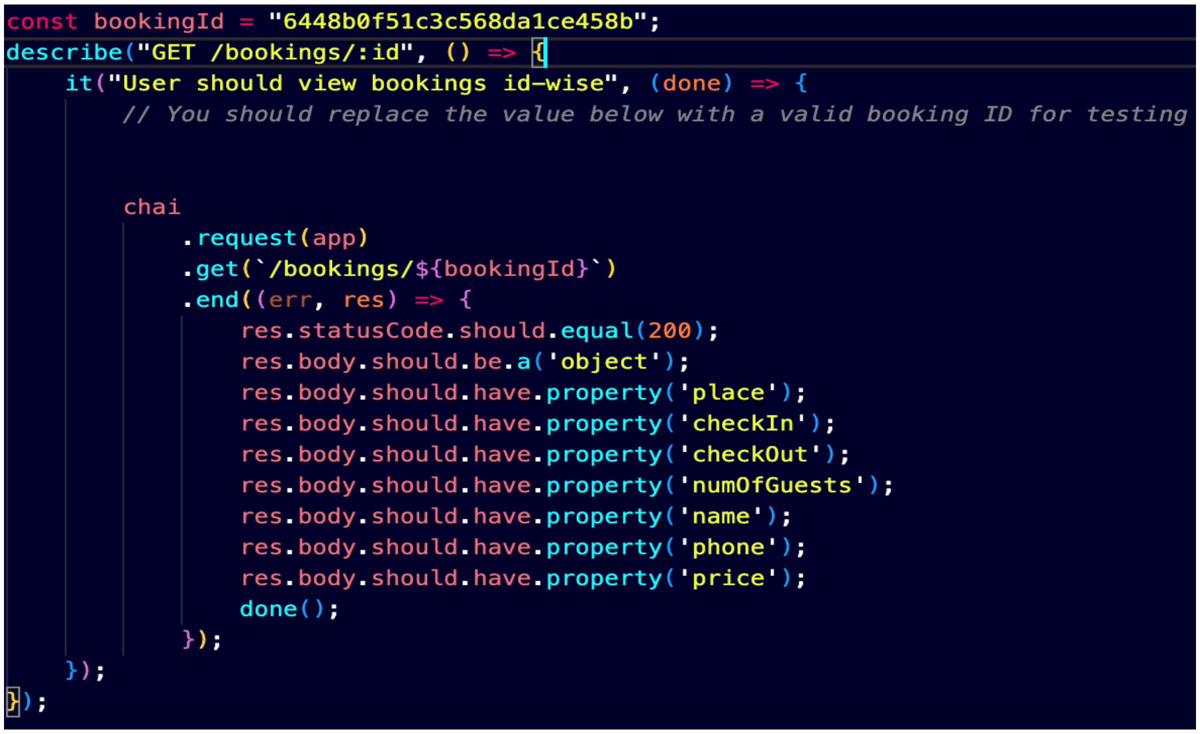


```
describe("GET /bookings", () => {
    it("User should view bookings", (done) => {


        chai
            .request(app)
            .get("/bookings")
            .set('Cookie', `token=${token}`)
            .end((err, res) => {
                res.statusCode.should.equal(200);
                res.body.should.be.a('array');
                res.body.length.should.be.greaterThan(0);
                res.body[0].should.have.property('place');

                res.body[0].should.have.property('checkIn');
                res.body[0].should.have.property('checkOut');
                res.body[0].should.have.property('numOfGuests');
                res.body[0].should.have.property('name');
                res.body[0].should.have.property('phone');
                res.body[0].should.have.property('price');
                done();
            });
    });
});
```

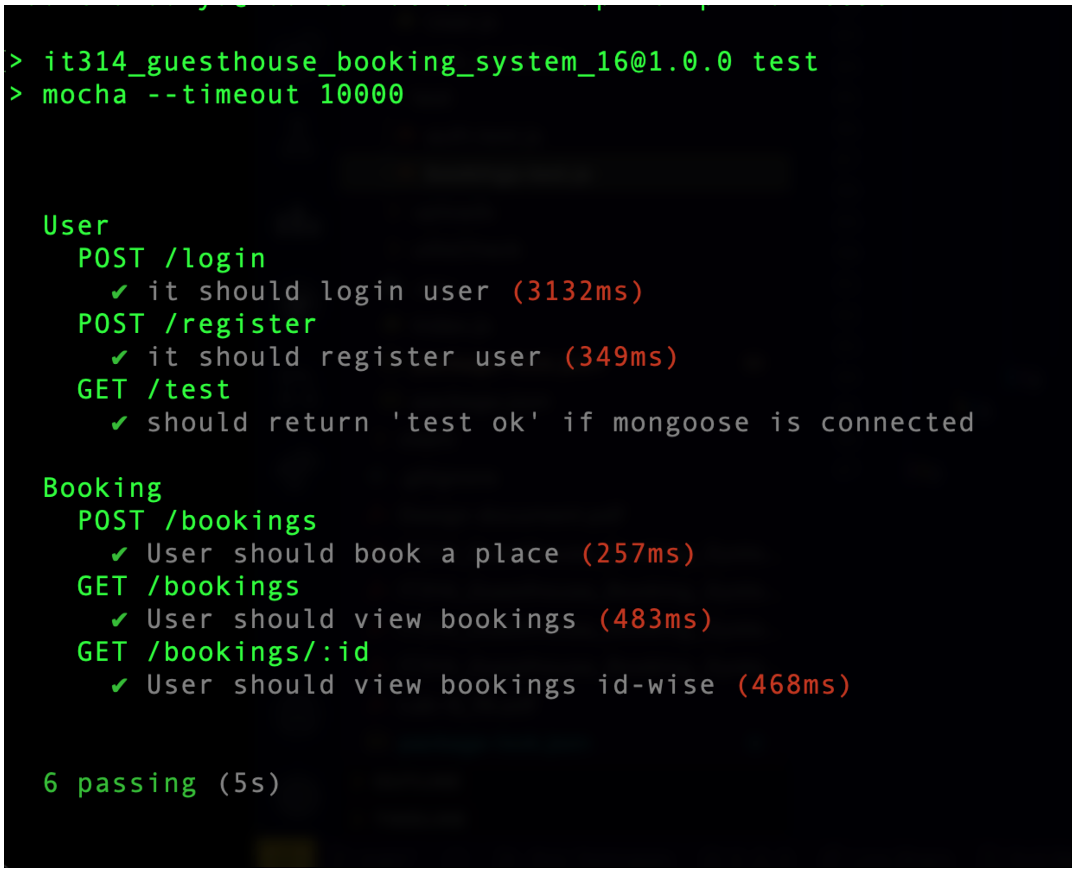
c. Checking for successfully viewing the bookings according to id.



```
const bookingId = "6448b0f51c3c568da1ce458b";
describe("GET /bookings/:id", () => {
    it("User should view bookings id-wise", (done) => {
        // You should replace the value below with a valid booking ID for testing


        chai
            .request(app)
            .get(`/bookings/${bookingId}`)
            .end((err, res) => {
                res.statusCode.should.equal(200);
                res.body.should.be.a('object');
                res.body.should.have.property('place');
                res.body.should.have.property('checkIn');
                res.body.should.have.property('checkOut');
                res.body.should.have.property('numOfGuests');
                res.body.should.have.property('name');
                res.body.should.have.property('phone');
                res.body.should.have.property('price');
                done();
            });
    });
});
```

Test Results:



```
> it314_guesthouse_booking_system_16@1.0.0 test
> mocha --timeout 10000



  User
    POST /login
      ✔ it should login user (3132ms)
    POST /register
      ✔ it should register user (349ms)
    GET /test
      ✔ should return 'test ok' if mongoose is connected

  Booking
    POST /bookings
      ✔ User should book a place (257ms)
    GET /bookings
      ✔ User should view bookings (483ms)
    GET /bookings/:id
      ✔ User should view bookings id-wise (468ms)


  6 passing (5s)
```