Building the CRM App: Unit Testing the app - Part-1

Relevel
by Unacademy

Topics to Be covered:

- Database and Code Setup for test user
- Unit testing Auth Controller
- Unit Test for User Controller
- Demonstrating code coverage

Database and Code Setup for test user:

We will need to create a new database connection with mongodb-memory-server.

Jest: Jest is a delightful JavaScript Testing Framework with a focus on simplicity.

mongodb-memory-server: It enables us to start a mongod process that stores data in memory. It is very fast as they never touch the hard-drive.

How to install:
npm install mongodb-memory-server
npm install jest

Create db.js

What is this file?

It consist of 3 function:

1. connect: Helps in creating connections.
2. closeDatabase: Helps in closing the database.
3. clearDatabase: Helps in clearing and dropping all the documents and collections from mongoDB.

Why do we need this file?

So to test we need to mock our database as well. We need to create connection, clear the values in database after every test so, tests will be independent of each other and at the end we need to close the connection with database we need few functions. So, such database related functions will be stored here.

How are we going to use it?

So, we can call all these functions in beforeAll, beforeEach and afterAll functions, as per our need.

Create db.js

```
const mongoose =  require('mongoose');
const {MongoMemoryServer} = require('mongodb-memory-
server');

let mongod;

// connect to db
module.exports.connect = async () => {
   if(!mongod){
       mongod = await MongoMemoryServer.create();
       const uri = mongod.getUri();
       const mongooseOpts = {
           useUnifiedTopology: true,
           maxPoolSize: 10
       };
       mongoose.connect(uri, mongooseOpts);
   }
}
```

```
//disconnect and close connection
module.exports.closeDatabase = async () => {
   await mongoose.connection.dropDatabase();
   await mongoose.connection.close();
   if(mongod)
       await mongod.stop();
}

// clear the db, remove all data
module.exports.clearDatabase = async () => {
   const              collections              =
mongoose.connection.collections;
   for(const key in collections){
       const collection = collections[key];
       collection.deleteMany();
   }
}
```

Unit testing Auth Controller:

```
const {mockRequest, mockResponse} = require("../interceptor")
const {signup, signin} =
require('../../controllers/auth.controller')
const User = require("../../models/user.model");
var bcrypt = require("bcryptjs");
```

What is mocking and why do we need it?
Mocking is a process used in unit testing when the unit being tested has external dependencies. The purpose of mocking is to isolate and focus on the code being tested and not on the behavior or state of external dependencies.

Interceptor.js
Interceptor.js consists the mock functions we need during our tests such as mock request and response object.

We are importing mockRequest and mockResponse from interceptor.js.
interceptor.js contains two properties/functions as below.

We are using mockRequest to mock the request for our controllers by using jest.fn().mockReturnValue(req) and store the mocked value inside request body and request params properties of the request object and then return the request object, request is generally referred as req and response is referred as res.

```
mockRequest: () => {
      const req = {}
      req.body = jest.fn().mockReturnValue(req)
      req.params = jest.fn().mockReturnValue(req)
      return req
    },

  mockResponse: () => {
      const res = {}
      res.send = jest.fn().mockReturnValue(res)
      res.status = jest.fn().mockReturnValue(res)
      res.json = jest.fn().mockReturnValue(res)
      return res
    },
```

Unit test for success in signup

- Create the test payload we will need while testing

```
const testPayload = {
  userType:"CUSTOMER",
  password:"12345678",
  name: "Test",
  userId: 1,
  email: "test@relevel.com",
  userStatus: "PENDING",
  ticketsCreated: [],
  ticketsAssigned: []
}
```

SpyOn:

Creates a mock function similar to jest.fn but also tracks number of calls to the function we called.

Returns a Jest mock function.

Unit test for success in signup

```
describe('SignUp', () => {
   it('should pass', async () => {
       const req = mockRequest();
       const res = mockResponse();
       req.body = testPayload;
       await signup(req, res);

       expect(res.status).toHaveBeenCalledWith(201);
       expect(res.send).toHaveBeenCalledWith(
           expect.objectContaining({
               email: "test@relevel.com",
               name: "Test",
               userId: "1",
               userStatus: "APPROVED",
               userTypes: "CUSTOMER"
           })
       );
   })
})
```

Unit test for Error SignUp

```
describe('SignUp', () => {
   it('should return error', async () => {
       const spy = jest.spyOn(User, 'create').mockImplementation(cb => cb(new
Error("This is an error."), null));
       const req = mockRequest();
       const res = mockResponse();
       testPayload.userType = "ENGINEER";
       req.body = testPayload;
       await signup(req, res);

       expect(res.status).toHaveBeenCalledWith(500);
       expect(res.send).toHaveBeenCalledWith({
           message: "Some internal error while inserting the element"
       });
   })
})
```

Unit testing Auth SignIn

- Create the test payload we will need while testing

```
const testPayload = {
   userType:"CUSTOMER",
   password:"12345678",
   name: "Test",
   userId: 1,
   email: "test@relevel.com",
   userStatus: "PENDING",
   ticketsCreated: [],
   ticketsAssigned: []
}
```

Unit testing Auth SignIn

```
describe('SignUp', () => {
   it('should fail due to password mismatch', async () => {
       testPayload.userStatus = "APPROVED"
       const userSpy = jest.spyOn(User, 'findOne').mockReturnValue(Promise.resolve(testPayload));
       const bcryptSpy = jest.spyOn(bcrypt, 'compareSync').mockReturnValue(false);
       const req = mockRequest();
       const res = mockResponse();
       req.body = testPayload;
       await signin(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(bcryptSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(401);
       expect(res.send).toHaveBeenCalledWith({
           accessToken: null,
           message: "Invalid Password!"
       });
   })
})
```

Unit test for should not allowed

```
describe('SignIn', () => {
    it('should not allowed', async () => {
        testPayload.userStatus = "PENDING"
        const spy = jest.spyOn(User, 'findOne').mockReturnValue(Promise.resolve(testPayload));
        const req = mockRequest();
        const res = mockResponse();
        req.body = testPayload;
        await signin(req, res);
        expect(spy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.send).toHaveBeenCalledWith({
            message: "Can't allow login as user is in statuts : [ PENDING]"
        });
    })
})
```

Unit test for Fail signin

```
it('should fail', async () => {
       const spy = jest.spyOn(User, 'findOne').mockReturnValue(null);
       const req = mockRequest();
       const res = mockResponse();
       req.body = testPayload;
       await signin(req, res);
       expect(spy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(400);
       expect(res.send).toHaveBeenCalledWith({
           message: "Failed! Userid doesn't exist!"
       });
   })
```

Unit Test for Should Pass SignUp

```
it('should pass', async () => {
       testPayload.userStatus = "APPROVED"
       const userSpy = jest.spyOn(User, 'findOne').mockReturnValue(Promise.resolve(testPayload));
       const bcryptSpy = jest.spyOn(bcrypt, 'compareSync').mockReturnValue(true);
       const req = mockRequest();
       const res = mockResponse();
       req.body = testPayload;
       await signin(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(bcryptSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(200);
       expect(res.send).toHaveBeenCalledWith(
           expect.objectContaining({
               email: testPayload.email,
               name: testPayload.name,
               userId: testPayload.userId,
               userTypes: testPayload.userType,
               userStatus: testPayload.userStatus
           })
       );
   })
```

Unit Test for User Controller

Setup:

```
const {mockRequest, mockResponse} = require("../interceptor")
const {findAll, findById, update} = require('../../controllers/user.controller')
const User = require("../../models/user.model");
```

Unit test for Update fail

```
it('should fail', async () => {
       const userSpy = jest.spyOn(User, 'findOneAndUpdate').mockReturnValue(cb =>
cb(new Error("This is an error."), null));
       const req = mockRequest();
       const res = mockResponse();
       req.params = {userId:1};
       await update(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(500);
       expect(res.send).toHaveBeenCalledWith({
           message: "Some internal error occured"
       });
   })
```

Unit test for Update pass

```
it('should pass', async () => {
        const userSpy = jest.spyOn(User, 'findOneAndUpdate').mockImplementation(()=>({
            exec: jest.fn().mockReturnValue(userTestPayload)
        }))
        const req = mockRequest();
        const res = mockResponse();
        req.params = {userId:1};
        req.body = userTestPayload;
        await update(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.send).toHaveBeenCalledWith({
            message: `User record has been updated successfully`
        });
    })
```

Unit test for Find by Id pass without data

```
it("should pass without data", async () =>{
       const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.resolve(null));
       const req = mockRequest();
       const res = mockResponse();
       req.params = {userId:1};
       await findById(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(200);
       expect(res.send).toHaveBeenCalledWith({
           message: `User with this id [1] is not present`
       });
   })
```

Unit test for Find by Id pass

```
it("should pass", async () =>{
       const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.resolve([userTestPayload]));
       const req = mockRequest();
       const res = mockResponse();
       req.params = {userId:1};
       await findById(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(200);
       expect(res.send).toHaveBeenCalledWith(
           expect.arrayContaining([
               expect.objectContaining({
                   userType:"CUSTOMER",
                   name: "Test",
                   userId: 1,
                   email: "test@relevel.com",
                   userStatus: "APPROVED"
               })
           ])
       );
   })
```

Unit test for Find All

```
it("should pass", async () =>{
        const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.resolve([userTestPayload]));
        const req = mockRequest();
        const res = mockResponse();
        req.query = {}
        await findAll(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.send).toHaveBeenCalledWith(
            expect.arrayContaining([
                expect.objectContaining({
                    email: "test@relevel.com",
                    name: "Test",
                    userId: 1,
                    userStatus: "APPROVED",
                    userType: "CUSTOMER"
                })
            ])
        );
    })
```

Unit test for Find All should pass with userStatus

```
it("should pass with userStatus", async () =>{
       const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.resolve([userTestPayload]));
       const req = mockRequest();
       const res = mockResponse();
       req.query = {userStatus: "APPROVED"}
       await findAll(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(200);
       expect(res.send).toHaveBeenCalledWith(
           expect.arrayContaining([
               expect.objectContaining({
                   email: "test@relevel.com",
                   name: "Test",
                   userId: 1,
                   userStatus: "APPROVED",
                   userType: "CUSTOMER"
               })
           ])
       );
   })
```

Unit test for Find All should pass with userType:

```
it("should pass with userType", async () =>{
        const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.resolve([userTestPayload]));
        const req = mockRequest();
        const res = mockResponse();
        req.query = {userType: "CUSTOMER"}
        await findAll(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.send).toHaveBeenCalledWith(
            expect.arrayContaining([
                expect.objectContaining({
                    email: "test@relevel.com",
                    name: "Test",
                    userId: 1,
                    userStatus: "APPROVED",
                    userType: "CUSTOMER"
                })
            ])
        );
    })
```

Unit test for Find All should pass with userType and Status

```
it("should pass with userStatus", async () =>{
       const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.resolve([userTestPayload]));
       const req = mockRequest();
       const res = mockResponse();
       req.query = {userStatus: "APPROVED"}
       await findAll(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(200);
       expect(res.send).toHaveBeenCalledWith(
           expect.arrayContaining([
               expect.objectContaining({
                   email: "test@relevel.com",
                   name: "Test",
                   userId: 1,
                   userStatus: "APPROVED",
                   userType: "CUSTOMER"
               })
           ])
       );
   })
```

Unit test for Find All should pass with name

```
it("should pass with name", async () =>{
        const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.resolve([userTestPayload]));
        const req = mockRequest();
        const res = mockResponse();
        req.query = {name: "test"
        }
        await findAll(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.send).toHaveBeenCalledWith(
            expect.arrayContaining([
                expect.objectContaining({
                    email: "test@relevel.com",
                    name: "Test",
                    userId: 1,
                    userStatus: "APPROVED",
                    userType: "CUSTOMER"
                })
            ])
        );
    })
```

Unit test for Find All should fail

```
it("should fail", async () =>{
       const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.reject(new Error("error")));
       const req = mockRequest();
       const res = mockResponse();
       req.query = {}
       await findAll(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(500);
       expect(res.send).toHaveBeenCalledWith({
           message: "Some internal error occured"
       });
   })
```

Unit test for Find All should fail with userType

```
it("should fail with userType", async () =>{
        const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.reject(new Error("error")));
        const req = mockRequest();
        const res = mockResponse();
        req.query = {userType: "CUSTOMER"}
        await findAll(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(500);
        expect(res.send).toHaveBeenCalledWith({
            message: "Some internal error occured"
        });
    })
```

Unit test for Find All should fail with name

```
it("should fail with name", async () =>{
        const userSpy = jest.spyOn(User, 'find').mockReturnValue(Promise.reject(new Error("error")));
        const req = mockRequest();
        const res = mockResponse();
        req.query = {name: "test"
        }
        await findAll(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(500);
        expect(res.send).toHaveBeenCalledWith({
            message: "Some internal error occured"
        });
    })
```

Demonstrating code coverage:

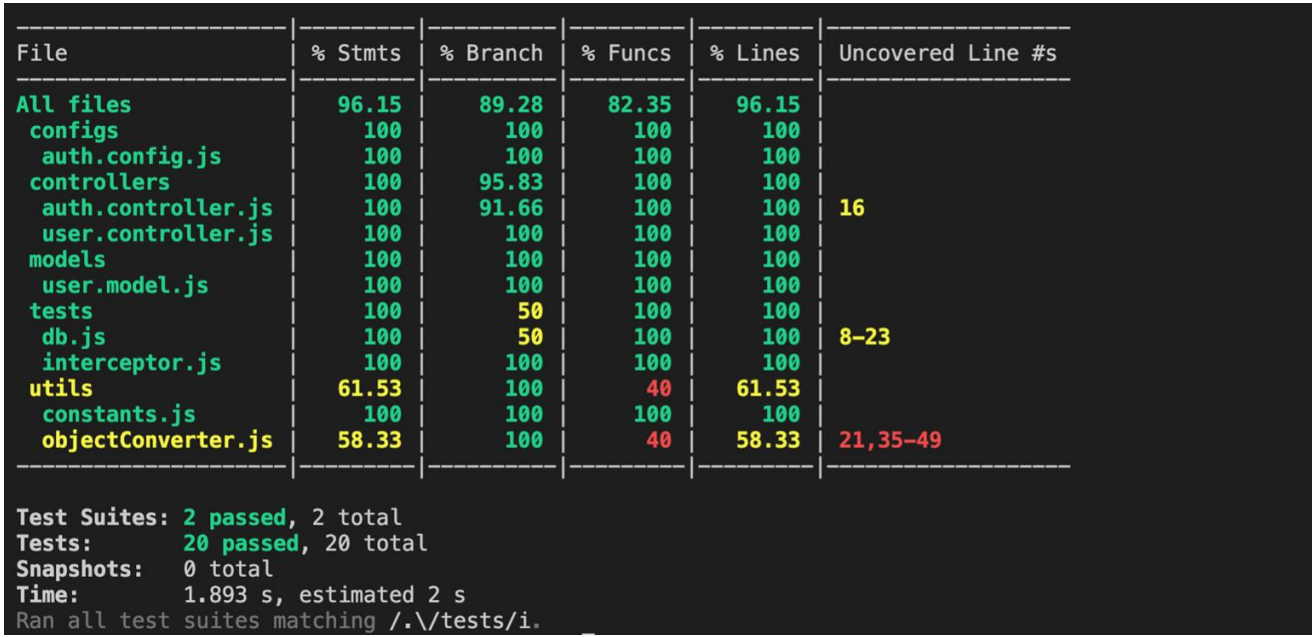
First you have to add one line in your package.json file

```
"scripts": {
   "test": "jest --testEnvironment=node --runInBand --detectOpenHandles --coverage
./tests",
   "devStart": "nodemon server.js"
 },
```

Run the below command in cmd

npm test

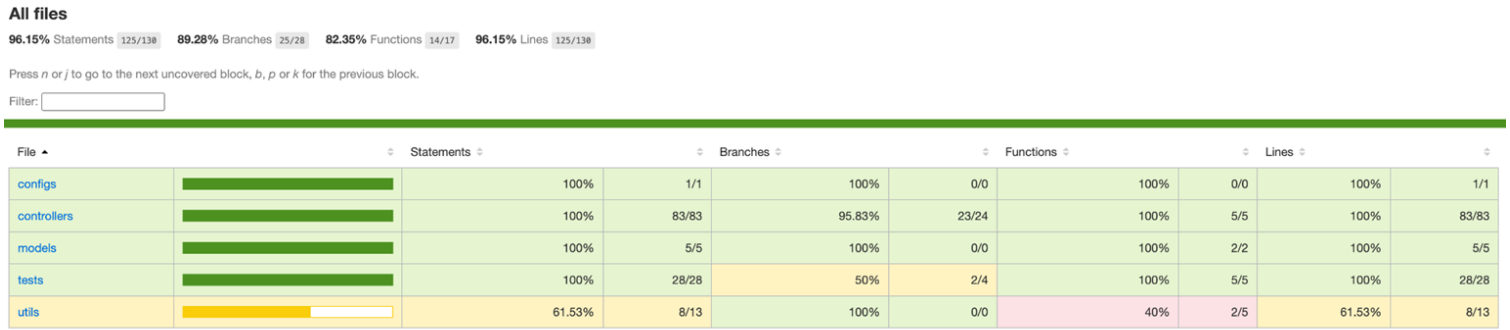
This will execute the unit tests and also gives the code coverage of our application.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	96.15	89.28	82.35	96.15	
configs	100	100	100	100	
auth.config.js	100	100	100	100	
controllers	100	95.83	100	100	
auth.controller.js	100	91.66	100	100	16
user.controller.js	100	100	100	100	
models	100	100	100	100	
user.model.js	100	100	100	100	
tests	100	50	100	100	
db.js	100	50	100	100	8-23
interceptor.js	100	100	100	100	
utils	61.53	100	40	61.53	
constants.js	100	100	100	100	
objectConverter.js	58.33	100	40	58.33	21,35-49

```
Test Suites: 2 passed, 2 total
Tests:       20 passed, 20 total
Snapshots:   0 total
Time:        1.893 s, estimated 2 s
Ran all test suites matching /.\/tests/i.
```

After this a coverage folder will be created in our app folder.
Now browse to the coverage folder and open index.html



All files

96.15% Statements `125/130` **89.28%** Branches `25/28` **82.35%** Functions `14/17` **96.15%** Lines `125/130`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
configs		100%	1/1	100%	0/0	100%	0/0	100%	1/1
controllers		100%	83/83	95.83%	23/24	100%	5/5	100%	83/83
models		100%	5/5	100%	0/0	100%	2/2	100%	5/5
tests		100%	28/28	50%	2/4	100%	5/5	100%	28/28
utils		61.53%	8/13	100%	0/0	40%	2/5	61.53%	8/13

You can click on this tests performed and can see code for which unit tests were performed

All files / controllers auth.controller.js

100% Statements 34/34 **91.66%** Branches 11/12 **100%** Functions 2/2 **100%** Lines 34/34

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1x  const User = require("../models/user.model");
1x  const { userType } = require("../utils/constants");
1x  const constants = require("../utils/constants");
1x  var bcrypt = require("bcryptjs");
1x  var jwt = require("jsonwebtoken");
1x  const config = require("../configs/auth.config");

    /**
     * Controller for the signup flow
     */
1x  exports.signup = async (req, res) => {
        /**
         * Inside the sign up call
         */
2x      var userStatus = req.body.userSatus;
2x      if(!req.body.userSatus){
2x          if(!req.body.userType || req.body.userType==constants.userTypes.customer){
1x              userStatus = constants.userStatus.approved;
            }else{
1x              userStatus = constants.userStatus.pending;
            }
        }

2x      const userObj = {
            name: req.body.name,
            userId: req.body.userId,
            email: req.body.email,
            userType: req.body.userType,
            password: bcrypt.hashSync(req.body.password, 8),
            userStatus: userStatus
        }

2x      try {
2x          const userCreated = await User.create(userObj);
1x          const postResponse = {
                name : userCreated.name,
                userId : userCreated.userId,
                email: userCreated.email,
                userTypes : userCreated.userType,
                userStatus : userCreated.userStatus,
                createdAt : userCreated.createdAt,
                updatedAt : userCreated.updatedAt
            }
1x          res.status(201).send(postResponse);
        } catch (err) {
1x          console.log("Some error while saving the user in db", err.message);
1x          res.status(500).send({
                message: "Some internal error while inserting the element"
            })
        }

    }

```

MCQs

1. _____ Accepts a value that will be returned whenever the mock function is called.
    a. mockReturnValue()
    b. All the options
    c. mockClear()
    d. mockRestore()

2. arrayContaining() is used to:

    a. To put an array in some mock Payload.

    b. Check if few values are an array.

    c. Check if few values are present in the received array.

    d. To match the exact number of values are received in an array.

3. objectContaining() is used to:

    a. To put an object in some mock Payload.
    b. Check if few values are an object.
    c. Check if few values are present in the received object.
    d. To match the exact number of values received in an object.

4. exec() is used for:

    a. To put db values to excel sheet.
    b. To execute to query.
    c. To give executive class privilege to queries.
    d. None of the above.

5. _____ enables us to start a mongod process that stores data in memory.

a. mongodb-memory-server
b. mongodb-server
c. mongodb-memory-solver
d. None of the options

Practice/HW

1. Write unit tests for ticket.route by yourself.
2. Write unit tests for auth.route by yourself.

Thank you