Building the CRM App: Unit Testing the app-part-II

Topic we will Cover:

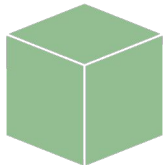
Unit testing Tickets controller

Integration testing for Auth route

Integration testing for Ticket route

Integration testing for User route

Create db.js



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.

Unit testing Tickets controller:

```
const {mockRequest, mockResponse} = require("../interceptor")
const {createTicket, getAllTickets, getOneTicket, updateTicket} =
require('../../controllers/ticket.controller')
const User = require("../../models/user.model");
const Ticket = require("../../models/ticket.model");
const client = require("../../utils/NotificationClient").client;

const db = require('../db')
beforeAll(async () => await db.connect())
afterEach(async () => await db.clearDatabase())
afterAll(async () => await db.closeDatabase())
```

Unit test for success in Create

- Create the test payload we will need while testing

```
const userTestPayload = {
   userType:"CUSTOMER",
   password:"12345678",
   name: "Test",
   userId: 1,
   email: "test@relevel.com",
   ticketsCreated: [],
   ticketsAssigned: [],
   save: jest.fn()
}
```

```
const ticketCreateTestPayload ={
   title: "Test",
   ticketPriority: 4,
   description: "Test",
   status: "OPEN",
   reporter: 1,
   assignee: 1,
   createdAt: Date.now(),
   updatedAt: Date.now()
}
```

Unit test for checking success in creating ticket

```
it("should pass", async () =>{
       const userSpy = jest.spyOn(User, 'findOne').mockReturnValue(Promise.resolve(userTestPayload));
       const clientSpy = jest.spyOn(client,'post').mockImplementation((url, args, cb) => cb("Test", null));
       const req = mockRequest();
       const res = mockResponse();
       req.body = ticketTestPayload;
       req.userId = 1;
       await createTicket(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(clientSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(201);
       expect(res.send).toHaveBeenCalledWith(
           expect.objectContaining({
               assignee: 1,
               description: "Test",
               reporter: 1,
               status: "OPEN",
               ticketPriority: 4,
               title: "Test",
           })
       );
   })
```

Unit test for checking failure in creating ticket

```
it('should fail', async () => {
       const userSpy = jest.spyOn(User,
'findOne').mockReturnValue(Promise.resolve(userTestPayload));
       const ticketSpy = jest.spyOn(Ticket, 'create').mockImplementation(cb => cb(new
Error("This is an error."), null));
       const req = mockRequest();
       const res = mockResponse();
       req.body = ticketTestPayload;
       await createTicket(req, res);
       expect(userSpy).toHaveBeenCalled();
       expect(ticketSpy).toHaveBeenCalled();
       expect(res.status).toHaveBeenCalledWith(500);
       expect(res.send).toHaveBeenCalledWith({
           message: "Some internal server error"
       });
   })
```

Unit testing for checking updating ticket success

```
it("should pass", async () =>{
        const userSpy = jest.spyOn(User, 'findOne').mockReturnValue(Promise.resolve(userTestPayload));
        const clientSpy = jest.spyOn(client,'post').mockImplementation(() => {});
        const ticketSpy = jest.spyOn(Ticket, 'findOne').mockReturnValue(Promise.resolve(ticketTestPayload));
        const req = mockRequest();
        const res = mockResponse();
        req.body = ticketTestPayload;
        req.userId = 1;
        req.param = {id:1};
        await updateTicket(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(clientSpy).toHaveBeenCalled();
        expect(ticketSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.send).toHaveBeenCalledWith(
            expect.objectContaining({
                assignee: 1,
                description: "Test",
                reporter: 1,
                status: "OPEN",
                ticketPriority: 4,
                title: "Test",
            })
        );
    })
```

Unit test for checking failure in updating ticket

```
it('should fail', async () => {
        ticketTestPayload.assignee = 5;
        const userSpy = jest.spyOn(User, 'findOne').mockReturnValue(Promise.resolve(userTestPayload));
        const ticketSpy = jest.spyOn(Ticket,
'findOne').mockReturnValue(Promise.resolve(ticketTestPayload));
        const req = mockRequest();
        const res = mockResponse();
        req.params = {id:1};
        req.userId = 1;
        await updateTicket(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(ticketSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(401);
        expect(res.send).toHaveBeenCalledWith({
            message: "Ticket can be updated only by the customer who created it"
        });
    })
```

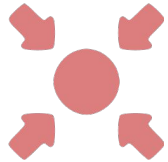
Unit test for Get All Ticket success for engineer

```
it("should pass for engineer", async () =>{
        userTestPayload.userType = "ENGINEER";
        const userSpy = jest.spyOn(User, 'findOne').mockReturnValue(Promise.resolve(userTestPayload));
        const ticketSpy = jest.spyOn(Ticket, 'find').mockReturnValue(Promise.resolve([ticketCreateTestPayload]));
        const req = mockRequest();
        const res = mockResponse();
        req.query = {status:"OPEN"};
        req.userId = 1;
        await getAllTickets(req, res);
        expect(userSpy).toHaveBeenCalled();
        expect(ticketSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.send).toHaveBeenCalledWith(
            expect.arrayContaining([
                expect.objectContaining({
                    assignee: 1,
                    description: "Test",
                    reporter: 1,
                    status: "OPEN",
                    ticketPriority: 4,
                    title: "Test"
                })
            ])
        );
    })
```

Unit test for checking failure in creating ticket

```
it("should pass with data", async () =>{
        const ticketSpy = jest.spyOn(Ticket, 'findOne').mockReturnValue(Promise.resolve(ticketCreateTestPayload)
        const req = mockRequest();
        const res = mockResponse();
        req.params = {id:1};
        await getOneTicket(req, res);
        expect(ticketSpy).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.send).toHaveBeenCalledWith(
            expect.objectContaining({
                assignee: 1,
                description: "Test",
                reporter: 1,
                status: "OPEN",
                ticketPriority: 4,
                title: "Test"
            })
        );
    })
```

Integration test of routes using supertest



Supertest:
SuperTest is a Node.js library that helps developers test APIs. It extends another library called superagent, a JavaScript HTTP client for Node.js and the browser. Developers can use SuperTest as a standalone library or with JavaScript testing frameworks like Mocha or Jest.

To install supertest:
npm install supertest

How to use supertest:
It is used to test HTTP servers by sending a method like GET, PUT, POST or DELETE, Supertest can test any HTTP requests. Even multipart file uploads like the one my API has. For example:
So, here we are sending request to the app which is our app server. And we are using post method to send the payload.

```
request(app)
.post(api_endpoint + 'auth/signup')
.send(testPayload)
```

Integration test for routes

```
const request = require('supertest');
const app = require("../../server");
const db = require('../db')
beforeAll(async () => await db.clearDatabase())
afterAll(async () => {
   await db.closeDatabase();
   app.close();
})
const api_endpoint = "/crm/api/v1/";
const testPayload = {
   userType:"CUSTOMER",
   password:"12345678",
   name: "Test",
   userId: 2,
   email: "test@relevel.com",
   userStatus: "PENDING",
   ticketsCreated: [],
   ticketsAssigned: []
}
```

Integration test for auth.routes for signup

```
describe('Post Signup Endpoints', () => {
   it('should signUp', async () => {
       const res = await request(app)
           .post(api_endpoint + 'auth/signup')
           .send(testPayload);
       expect(res.statusCode).toEqual(201);
       expect(res.body).toEqual(
           expect.objectContaining({
               "email": "test@relevel.com",
               "name": "Test",
               "userId": "2",
               "userStatus": "APPROVED",
               "userTypes": "CUSTOMER"
           })
       );
   })
})
```

Integration test for auth.routes for signin

```
describe('Post Signin Endpoints', () => {
   it('should signin', async () => {
       const res = await request(app)
           .post(api_endpoint + 'auth/signin')
           .send(testPayload);
       expect(res.statusCode).toEqual(200);
       expect(res.body).toEqual(
           expect.objectContaining({
               "email": "test@relevel.com",
               "name": "Test",
               "userId": "2",
               "userStatus": "APPROVED",
               "userTypes": "CUSTOMER"
           })
       );
   })
})
```

Integration Test for User.routes

```
const request = require('supertest');

const User = require('../../models/user.model');
const app = require("../../server");
var jwt = require("jsonwebtoken");
const config = require("../../configs/auth.config");

const db = require('../db')
beforeAll(async () => {
   await db.clearDatabase();
   await User.create({
       name: "Vishwa",
       userId: 1, // It should be atleat 16, else will throw error
       email: "Kankvish@gmail.com",  // If we don't pass this, it will throw the error
       userType: "ADMIN",
       password :"Welcome1",
       userStatus: "APPROVED"

   });

})
afterAll(async () => {
   await db.closeDatabase();
   app.close();
})
```

Integration test for user.routes for Find By Id

```
describe('Find By Id Endpoints', () => {
   var token = jwt.sign({ id: 1 }, config.secret, {
       expiresIn: 120 // 2 minutes
     });
   it('should find by id', async () => {
       const res = await request(app)
           .get(api_endpoint + 'users/1')
           .set("x-access-token", token)
           .field("userId",1)
       console.log(res.error)
       expect(res.statusCode).toEqual(200);
       expect(res.body).toEqual(
           expect.arrayContaining([
               expect.objectContaining({
                   "email": "kankvish@gmail.com",
                   "name": "Vishwa",
                   "userId": "1",
                   "userStatus": "APPROVED",
                   "userType": "ADMIN"})
           ])
       );
   })
})
```

Integration test for user.routes for find all

```
describe('Find All Endpoints', () => {

   var token = jwt.sign({ id: 1 }, config.secret, {
       expiresIn: 120 // 2 minutes
     });
   it('should Find All', async () => {
       const res = await request(app)
           .get(api_endpoint + 'users')
           .set("x-access-token", token)
           .query({userType:"ADMIN"})
           .field("userId",1)
       expect(res.statusCode).toEqual(200);
       expect(res.body).toEqual(
           expect.arrayContaining([
               expect.objectContaining({
                   "email": "kankvish@gmail.com",
                   "name": "Vishwa",
                   "userId": "1",
                   "userStatus": "APPROVED",
                   "userType": "ADMIN"})
           ])
       );
   })
})
```

Integration test for user.routes for Update

```
describe('PUT Update Endpoints', () => {

   var token = jwt.sign({ id: 1 }, config.secret, {
       expiresIn: 120 // 2 minutes
     });
   it('should Update', async () => {
       const res = await request(app)
           .put(api_endpoint + 'users/1')
           .set("x-access-token", token)
           .field("userType","ADMIN")
           .field("userId",1)
       console.log(res.error)
       expect(res.statusCode).toEqual(200);
       expect(res.body).toEqual({
           "message": "User record has been updated successfully"
       });
   })
})
```

Integration Test for Ticket.routes

```
const request = require('supertest');

const User = require('../../models/user.model');
const app = require("../../server");
var jwt = require("jsonwebtoken");
const config = require("../../configs/auth.config");
const client = require("../../utils/NotificationClient").client;

const db = require('../db')
beforeAll(async () => {
   await db.clearDatabase();
   await User.create({
       name: "Vishwa",
       userId: 1, // It should be atleat 16, else will throw
error
       email: "Kankvish@gmail.com",  // If we don't pass this, it
will throw the error
       userType: "ENGINEER",
       password :"Welcome1",
       userStatus: "APPROVED"

   });

})
```

```
afterAll(async () => {
   await db.closeDatabase();
   app.close();
})

const api_endpoint = "/crm/api/v1/";

const ticketTestPayload ={
   title: "Test",
   ticketPriority: 4,
   description: "Test",
   status: "OPEN",
   assignee: 1
}
let updateId;
```

Integration test for ticket.routes for Find By Id

```
describe('Post Tickets Endpoints', () => {
   jest.spyOn(client,'post').mockImplementation((url, args, cb) => cb("Test", null));

   var token = jwt.sign({ id: 1 }, config.secret, {
       expiresIn: 120 // 2 minutes
     });
   it('should create', async () => {
       const res = await request(app)
           .post(api_endpoint + 'tickets/')
           .set("x-access-token", token)
           .send(ticketTestPayload);
       id = res.body.id;
       expect(res.statusCode).toEqual(201);
       expect(res.body).toEqual(
           expect.objectContaining({
               "assignee": 1,
               "description": "Test",
               "reporter": 1,
               "status": "OPEN",
               "ticketPriority": 4,
               "title": "Test",
           })
       );
   })
})
```

Integration test for ticket.routes for update

```
describe('Put Tickets Endpoints', () => {
   jest.spyOn(client,'post').mockImplementation((url, args, cb) => cb("Test", null));

   var token = jwt.sign({ id: 1 }, config.secret, {
       expiresIn: 120 // 2 minutes
     });
   it('should update', async () => {
       const res = await request(app)
           .put(api_endpoint + 'tickets/' + id)
           .set("x-access-token", token)
           .send(ticketTestPayload);
       expect(res.statusCode).toEqual(200);
       expect(res.body).toEqual(
           expect.objectContaining({
               "assignee": 1,
               "description": "Test",
               "reporter": 1,
               "status": "OPEN",
               "ticketPriority": 4,
               "title": "Test",
           })
       );
   })
})
```

Integration test for ticket.routes for get all tickets

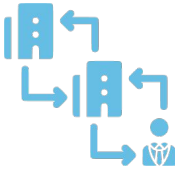
```
describe('Get All Tickets Endpoints', () => {
   jest.spyOn(client,'post').mockImplementation((url, args, cb) => cb("Test", null));
   var token = jwt.sign({ id: 1 }, config.secret, {
       expiresIn: 120 // 2 minutes
     });
   it('should get all', async () => {
       const res = await request(app)
           .get(api_endpoint + 'tickets')
           .set("x-access-token", token)
           .send(ticketTestPayload);
       updateId = res.body[0].id;
       expect(res.statusCode).toEqual(200);
       expect(res.body).toEqual(
           expect.arrayContaining([
               expect.objectContaining({
                   "assignee": 1,
                   "description": "Test",
                   "reporter": 1,
                   "status": "OPEN",
                   "ticketPriority": 4,
                   "title": "Test",
               })
           ])
       );
   }))
```

Integration test for ticket.routes for get One tickets

```
describe('Get One Tickets Endpoints', () => {
   jest.spyOn(client,'post').mockImplementation((url, args, cb) => cb("Test", null));

   var token = jwt.sign({ id: 1 }, config.secret, {
       expiresIn: 120 // 2 minutes
     });
   it('should get one', async () => {
       const res = await request(app)
           .get(api_endpoint + 'tickets/' + updateId)
           .set("x-access-token", token)
           .send();
       expect(res.statusCode).toEqual(200);
       expect(res.body).toEqual(
           expect.objectContaining({
               "assignee": 1,
               "description": "Test",
               "reporter": 1,
               "status": "OPEN",
               "ticketPriority": 4,
               "title": "Test",
           })
       );
   })
})
```

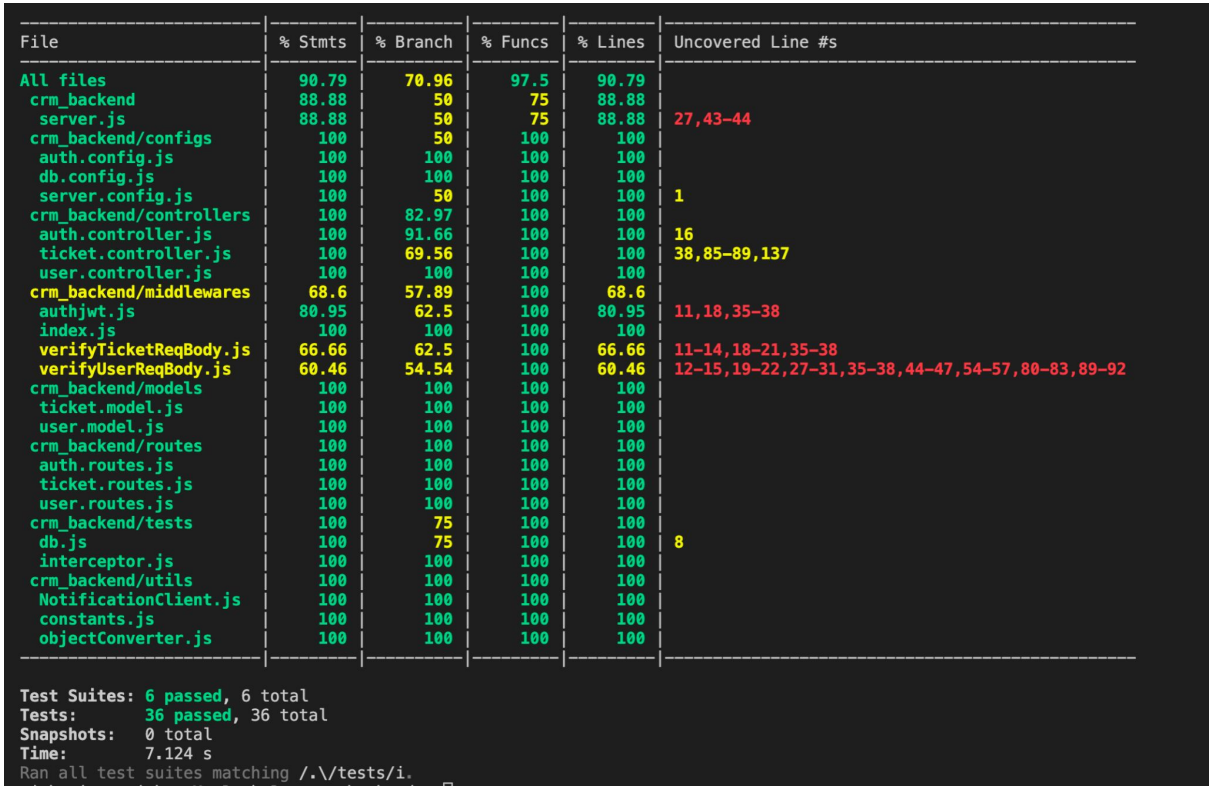
Demonstrating code coverage



Run the below command in cmd

npm test

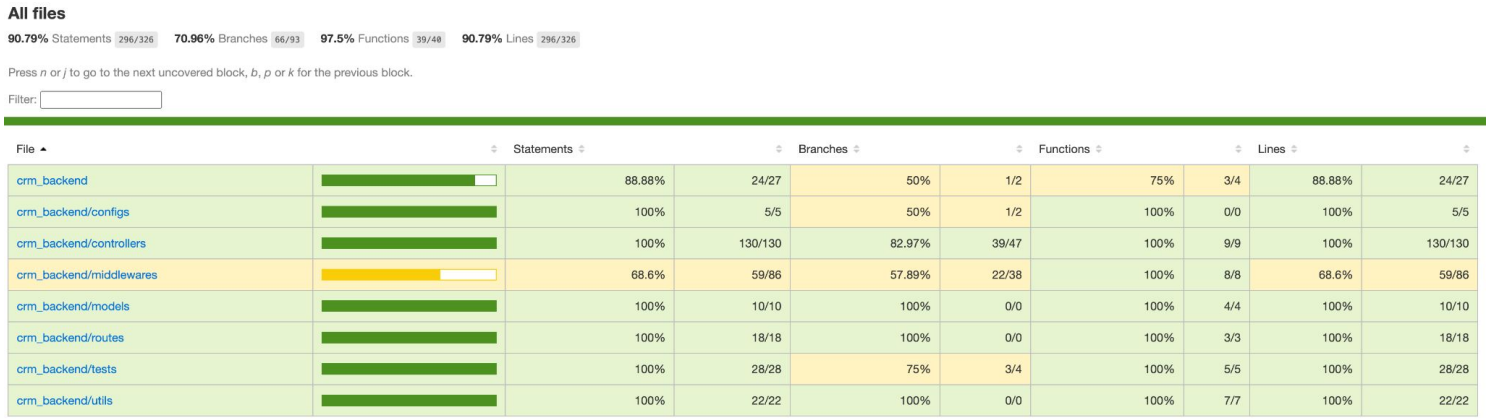
This will execute the unit tests and also gives the code coverage of our application.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	90.79	70.96	97.5	90.79	
crm_backend	88.88	50	75	88.88	
server.js	88.88	50	75	88.88	27,43-44
crm_backend/configs	100	50	100	100	
auth.config.js	100	100	100	100	
db.config.js	100	100	100	100	
server.config.js	100	50	100	100	1
crm_backend/controllers	100	82.97	100	100	
auth.controller.js	100	91.66	100	100	16
ticket.controller.js	100	69.56	100	100	38,85-89,137
user.controller.js	100	100	100	100	
crm_backend/middlewares	68.6	57.89	100	68.6	
authjwt.js	80.95	62.5	100	80.95	11,18,35-38
index.js	100	100	100	100	
verifyTicketReqBody.js	66.66	62.5	100	66.66	11-14,18-21,35-38
verifyUserReqBody.js	60.46	54.54	100	60.46	12-15,19-22,27-31,35-38,44-47,54-57,80-83,89-92
crm_backend/models	100	100	100	100	
ticket.model.js	100	100	100	100	
user.model.js	100	100	100	100	
crm_backend/routes	100	100	100	100	
auth.routes.js	100	100	100	100	
ticket.routes.js	100	100	100	100	
user.routes.js	100	100	100	100	
crm_backend/tests	100	75	100	100	
db.js	100	75	100	100	8
interceptor.js	100	100	100	100	
crm_backend/utils	100	100	100	100	
NotificationClient.js	100	100	100	100	
constants.js	100	100	100	100	
objectConverter.js	100	100	100	100	

Test Suites: 6 passed, 6 total
Tests: 36 passed, 36 total
Snapshots: 0 total
Time: 7.124 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

90.79% Statements `296/326` **70.96%** Branches `66/93` **97.5%** Functions `39/40` **90.79%** Lines `296/326`

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	
crm_backend	88.88%	24/27	50%	1/2	75%	3/4	88.88%	24/27
crm_backend/configs	100%	5/5	50%	1/2	100%	0/0	100%	5/5
crm_backend/controllers	100%	130/130	82.97%	39/47	100%	9/9	100%	130/130
crm_backend/middlewares	68.6%	59/86	57.89%	22/38	100%	8/8	68.6%	59/86
crm_backend/models	100%	10/10	100%	0/0	100%	4/4	100%	10/10
crm_backend/routes	100%	18/18	100%	0/0	100%	3/3	100%	18/18
crm_backend/tests	100%	28/28	75%	3/4	100%	5/5	100%	28/28
crm_backend/utils	100%	22/22	100%	0/0	100%	7/7	100%	22/22

MCQs

1. _____ is used to add headset in supertest

A. setHeader()
B. All the options
C. set()
D. Header()

2. _____ is used to set value to direct child of request payload.

A. Query
B. Body
C. Field
D. None of the above

3. How to end the express server.

A. app.end()
B. app.close()
C. app.stop()
D. None of the above

4. Which will not be returned by require?

A. exports

B. module.exports

C. Both

D. None of the above

5. jest.fn().mockImplemtaion() can be used to mock

A. save()

B. findOneAndUpdate()

C. exec()

D. All of the above

Practice/HW

- Write unit tests for verifyTicketReqBody method by yourself.
- Write unit tests for verifyUserReqBody method by yourself.

Thank You!