

CS673 Software Engineering
Team 3 - EMS
Software Test Document

<u>Team Member</u>	<u>Role(s)</u>	<u>Signature Date</u>
Jianle Xie	Team Leader	<u>Jianle Xie 9/9/2023</u>
Yuxuan Wang	QA leader	<u>Yuxuan Wang 9/9/2023</u>
Abigya Devkota	Design and Implementation Leader	<u>Abigya Devkota</u> <u>9/9/2023</u>
Jingjing Tang	Configuration Leader	<u>Jingjing Tang 9/10/2023</u>
HaoLun Li	Requirement Leader	<u>HaoLun Li 9/11/2023</u>
WeiHao Mai	Security Leader	<u>Wei hao Mai 9/11/2023</u>
Zhiwei Lin	Backup Leader	<u>Zhiwei Lin 9/11/2023</u>

Revision history

<u>Versi on</u>	<u>Author</u>	<u>Date</u>	<u>Change</u>
<u>v2</u>	<u>Wei hao</u>	<u>10/9/23</u>	<u>Unit testing</u>
<u>v3</u>	<u>Jianle Xie</u>	<u>10/09/23</u>	<u>Acceptance Testing</u> <u>Manual Testing</u>

[Testing Summary](#)

[Manuel Tests Reports](#)

[Automated Testing Reports](#)

[Testing Metrics](#)

[References](#)

[Glossary](#)

● Testing Summary

In this section, you will summarize what was tested, who is involved in testing, testing techniques used, and testing result. You may have the following tests

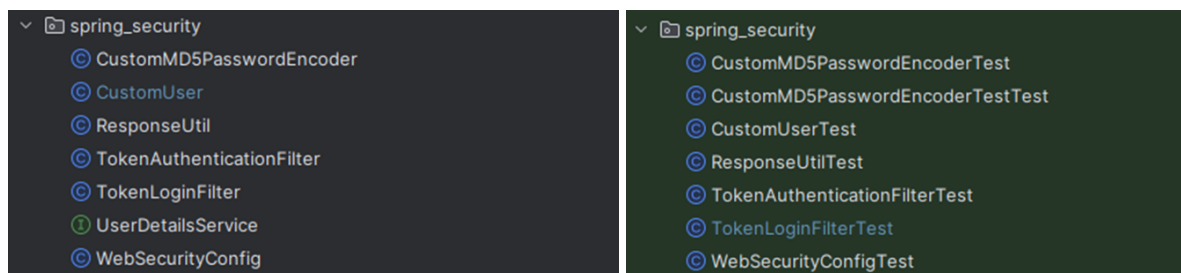
- Unit Testing
- Integration testing
- System Testing
- Acceptance Testing
- Regression Testing

Unit Testing

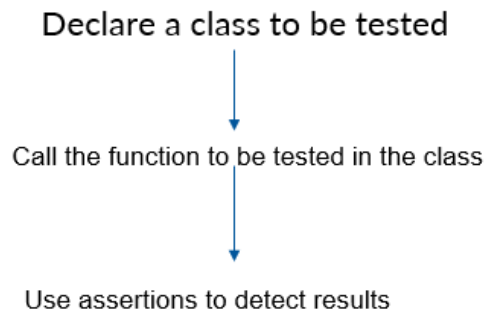
Unit testing is the process of isolating a section of code and then testing that the internals of that section work as you expect.

Mock code structure: The following is an introduction to my code structure, this unit test mainly tests the security part, for the security of each class file, write corresponding

For example, the user's class: In the CustomUser.java file, write CustomUserTest.java to test the class.



Test flow:



For each public function in each class, write the function to test as follows: First, declare a test class, and then in the Class calls the function to be tested, and finally uses the assertion as the test result, when the condition is true, the program will continue to execute; When the conditions are false, The JVM will throw an AssertionError.

Mock code example:

```
@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain, Authentication authResult)
    throws IOException, ServletException {
    CustomUser user = (CustomUser) authResult.getPrincipal();
    String token = JwtHelper.createToken(user.getCredential().getEmployeeId(), user.getCredential().getPasswordHash());

    Map<String, Object> map = new HashMap<>();
    map.put("token", token);
    ResponseUtil.out(response, Result.ok(map));
}
```

```
@Test
void testSuccessfulAuthentication_ThrowsIOException() {
    // Setup
    final HttpServletResponse response = null;
    final Authentication authResult = new TestingAuthenticationToken(principal: "user", credentials: "pass", ...authorities: "ROLE_USER");

    // Run the test
    assertThatThrownBy(() -> tokenLoginFilterUnderTest.successfulAuthentication(request: null, response, chain: null,
        authResult)).isInstanceOf(IOException.class);
}
```

Let's take a look at a code example: there is a function called successfulAuthentication, the method (pictured above)

The purpose is to generate a JWT token after the user successfully authenticates and return it to the client as part of the response.

I wrote the test code (the image below), which consists of three steps: Setup, Run the test, and verify the results. We simulate a successful authentication where the user's username is "user" and password value is "pass" and the role is "ROLE_USER". In this test, we used an empty HttpServletResponse object and applied it passed to successfulAuthentication method.

Finally, we verified the results. When validating the result, if no exception is thrown Consider the test passed.


● Manual Testing Report

In this section, you will give a detailed description of each manual test case performed and the result. If this is a previous You shall list what are existing tests developed in the previous semester and what are new tests developed currently.

Here is a sample template that can be used for each test case. For system tests or acceptance tests, you may also include some screenshots.

- Test case ID, name
- New or old:
- Test items: (what do you test)
- Test priority (high/medium/low)
- Dependencies (to other test case/requirement if any):
- Preconditions: (if any)
- input data:
- Test steps:
- Postconditions:
- Expected output:
- Actual output:
- Pass or Fail:
- Bug id/link: (this should link to your github issue id)
- Additional notes

See sheet:

 Manual Testing

● Pass or Fail: pass ● Automated Testing Report

Describe briefly the automated testing you have done, including where the test code resides in your code repository, what test frameworks are used, and the screen shots or generated testing report.

System testing (Backend-testing)

These tests reside in

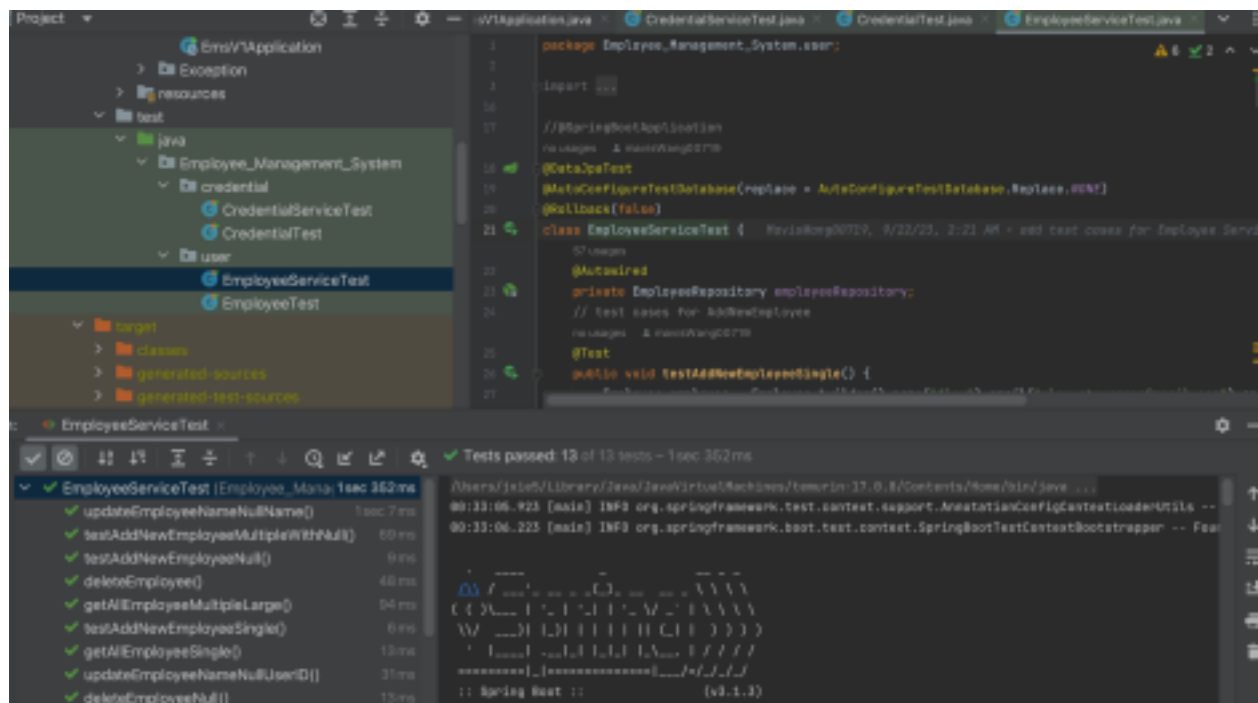
project-team-3/code/Employee_management_system/src/test/java/Employee_Management_system according to IntelliJ and under the database-frontend-backend branch. There are currently 2 folders of tests, credential and user.

CredentialServiceTest: We are testing the methods of the CredentialService entity.

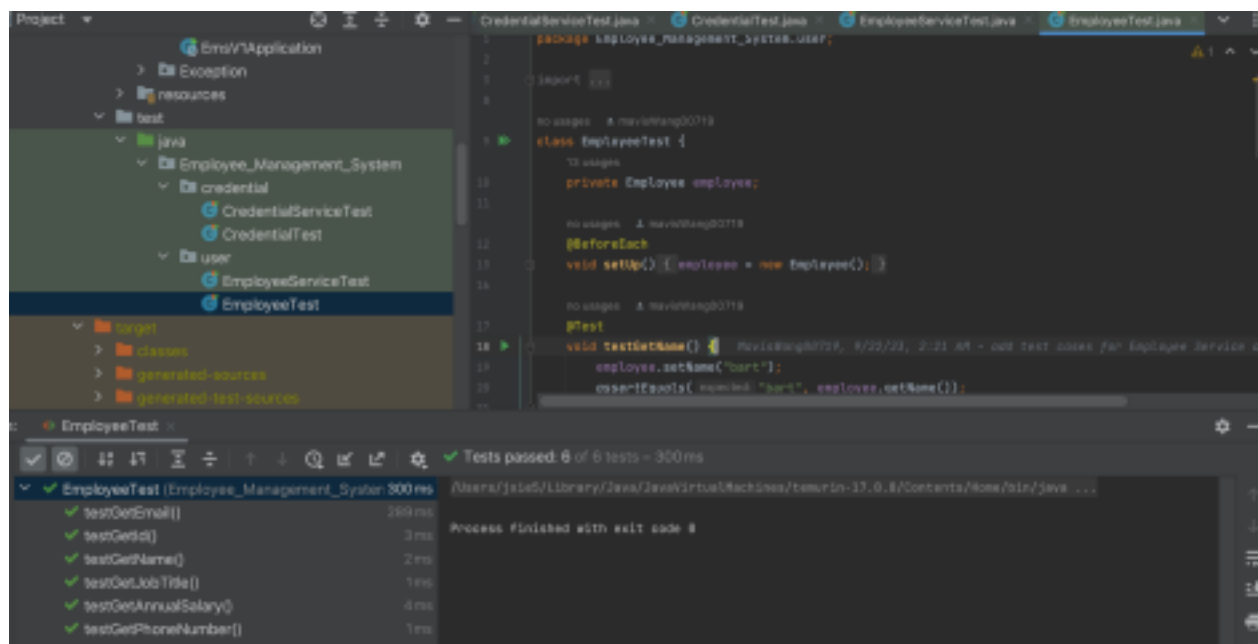
```
1 package Employee_Management_System.credential;
2
3 import org.junit.jupiter.api.Test;  Charlene1998, 9/24/23, 12:49 AM • Changes
4
5 import static org.junit.jupiter.api.Assertions.*;
6
7 no usages  ⤴ charlene1998 +1
8 class CredentialServiceTest {
9
10  no usages  ⤴ charlene1998 +1
11  @Test
12  void register() {
13
14  no usages  ⤴ charlene1998 +1
15  @Test
16  void authenticate() {
17
18  }
19 }
20 }
```

CredentialTest: Testing Credential entity

EmployeeServiceTest: Testing for SpringBoot initialization and EmployeeService class entity methods.



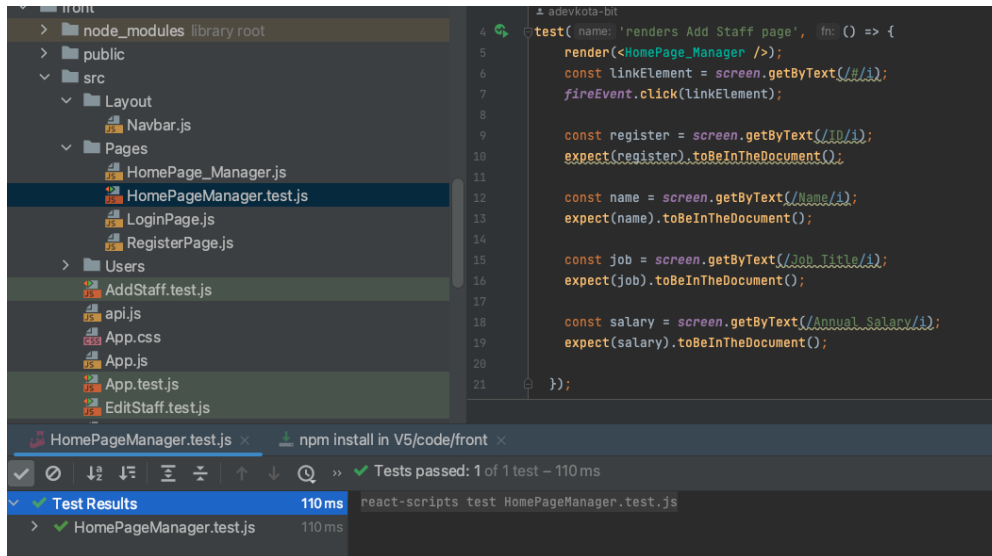
EmployeeTest: Testing for Employee entity's methods.



System testing (FrontEnd):

Front-end testing reflects the components that are being tested. All frontend test are located within /code/front/src

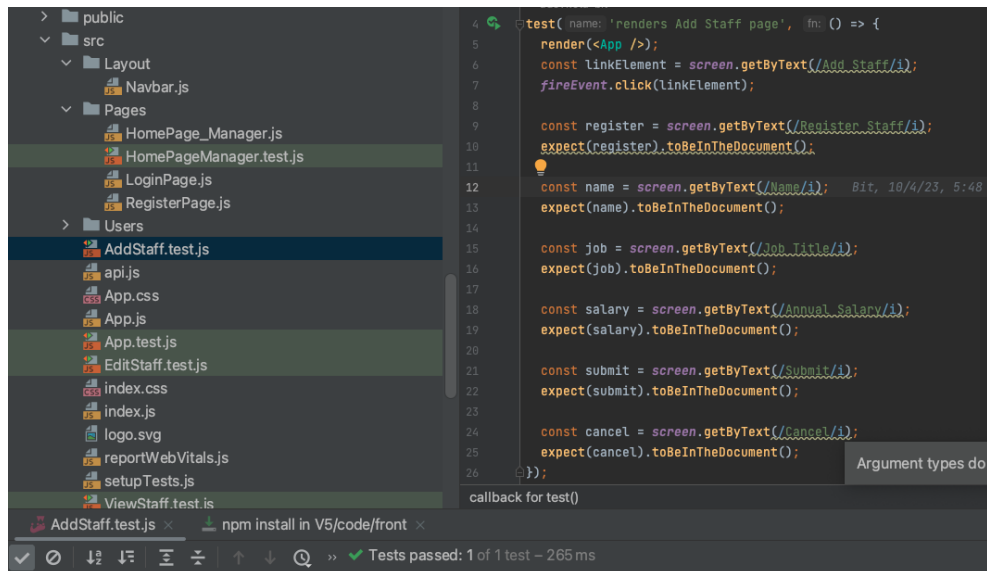
HomePage_Manager tests



```
4 test('name: 'renders Add Staff page', fn: () => {
5   render(<HomePage_Manager />);
6   const linkElement = screen.getByText(/#/i);
7   fireEvent.click(linkElement);
8
9   const register = screen.getByText(/ID/i);
10  expect(register).toBeInTheDocument();
11
12  const name = screen.getByText(/Name/i);
13  expect(name).toBeInTheDocument();
14
15  const job = screen.getByText(/Job Title/i);
16  expect(job).toBeInTheDocument();
17
18  const salary = screen.getByText(/Annual Salary/i);
19  expect(salary).toBeInTheDocument();
20
21 });
```

Test Results 110 ms
react-scripts test HomePageManager.test.js
HomePageManager.test.js 110 ms

AddStaff test



```
4 test('name: 'renders Add Staff page', fn: () => {
5   render(<App />);
6   const linkElement = screen.getByText(/Add Staff/i);
7   fireEvent.click(linkElement);
8
9   const register = screen.getByText(/Register Staff/i);
10  expect(register).toBeInTheDocument();
11
12  const name = screen.getByText(/Name/i);
13  expect(name).toBeInTheDocument();
14
15  const job = screen.getByText(/Job Title/i);
16  expect(job).toBeInTheDocument();
17
18  const salary = screen.getByText(/Annual Salary/i);
19  expect(salary).toBeInTheDocument();
20
21  const submit = screen.getByText(/Submit/i);
22  expect(submit).toBeInTheDocument();
23
24  const cancel = screen.getByText(/Cancel/i);
25  expect(cancel).toBeInTheDocument();
26  callback for test()
});
```

Test Results 265 ms
react-scripts test AddStaff.test.js
AddStaff.test.js 265 ms

Viewstaff testing

```
8 jest.mock( moduleName: 'axios', factory: () => {
9   return {
10     create: jest.fn( implementation: () => ({
11       get: jest.fn(),
12       interceptors: {
13         request: { use: jest.fn(), eject: jest.fn() },
14         response: { use: jest.fn(), eject: jest.fn() }
15       }
16     })
17   }
18 })
19
20 test( name: 'renders View Staff page', fn: async () => {
21   const fakeUser = { name: "Alice",
22     jobTitle: "QA",
23     annualSalary: "100000" };
24
25   const payload = { data: fakeUser };
26
27   axios.get = jest.fn().mockResolvedValue(payload);
28
29   render(<MemoryRouter><ViewStaff /></MemoryRouter>);
30
31
32   const staffDetails = screen.getByText(/Staff Details/i);
33   expect(staffDetails).toBeInTheDocument();
34
35   const details = screen.getByText(/Detail of user id:1/);
36   expect(details).toBeInTheDocument();
37
38   const name = screen.getByText(/Name/i);
39   expect(name).toBeInTheDocument();
40 }
```

in V5/code/front x

Tests passed: 1 of 1 test – 88ms

88ms react-scripts test ViewStaff.test.js

Acceptance Testing

User can login with username and password and then click on Login button

Add Staff

HRMasery Login

UserName

Password

Login [New user](#)

Admin user's home page can view all staff information with feature buttons to view, edit, and delete employees.

Admin user's home page also includes a Logout button to logout of the application.

HRMasery					Add Staff	Log Out
#	ID	Name	Job Title	Annual Salary		
1	1	john	teacher	\$50000	View Employee	Edit Employee Delete Employee
2	2	jack	bus driver	\$60000	View Employee	Edit Employee Delete Employee
3	3	joe	firefighter	\$70000	View Employee	Edit Employee Delete Employee

Admin user can add additional stuff via the Add Staff button and add staff via submit after entering staff information

Add Staff

Register Staff

Name

Enter name

Job Title

Enter Job Title

Annual Salary

Enter Annual Salary

SubmitCancel

SpringSecurity:

After adding SpringSecurity dependencies, even if you bypass the front-end login page and go directly to use the method url like “

http://localhost:8800/admin/system/sysRole/findAll” , you will not get the information. It will show that

```
{"code":209,"message":"No permission","data":null}
```

After adding the global variable token, prove that you are logged in. All the methods will work just as before

The screenshot shows a REST client interface with a GET request to `/admin/system/index/getUserInfo`. The request header includes a `token` with the value `eyJhbGciOiJIUzUxMiIsInppcCI6IkdkaSVAifQ.H4slAAAAAAAAAAKtWKi5NUrJScgwN8dD18Xf39NMNDXY`. The response is a JSON object with status 200, message "Success", and user details including permissions, name, and menu.

请求头	内容	操作
token	eyJhbGciOiJIUzUxMiIsInppcCI6IkdkaSVAifQ.H4slAAAAAAAAAAKtWKi5NUrJScgwN8dD18Xf39NMNDXY	删除

```
1- {
2-   "code": 200,
3-   "message": "Success",
4-   "data": {
5-     "permissions": [
6-       "bnt.sysUser.list",
7-       "bnt.sysUser.add",
8-       "bnt.sysUser.remove"
9-     ],
10-    "name": "zhangsan",
11-    "menu": [
12-      {
13-        "id": 3,
14-        "createTime": "2021-05-31 10:05:37",
15-        "updateTime": "2022-06-09 01:22:47",
16-        "isDeleted": 0,
17-        "param": {},
18-        "parentId": 2,
19-        "name": "用户管理",
20-        "type": 1,
21-        "path": "sysUser",
```

Without Token:

The screenshot shows the same REST client interface, but the `token` header is unchecked. The response is a JSON object with status 209, message "No permission", and null data.

请求头	内容	操作
<input type="checkbox"/> token	eyJhbGciOiJIUzUxMiIsInppcCI6IkdkaSVAifQ.H4slAAAAAAAAAAKtWKi5NUrJScgwN8dD18Xf39NMNDXY	删除

```
1- {
2-   "code": 209,
3-   "message": "No permission",
4-   "data": null
5- }
```

● Testing Metrics

In this section, you shall report any metrics used for the evaluation, e.g. # of test cases, test coverage, defects rate, etc.

● References

● Glossary