Name: Sanjay Khanna S

Emp Id: 12124

Jest & JUnit Testing Lab Assessment

Perform the below activities for each Requirements

General Instructions:

1. Use the existing **React – Spring Boot REST** Application (Created for Manual Testing)
2. Perform the **W3H Analysis** on the Unit Testing Scenarios

(Use Manual Testing Doc as Reference)

3. Prepare a **Test Document** which contains the Unit Test Cases which can be performed in front-end and back-end.
4. Use **JUnit** for testing your Spring Boot REST Code
    a. Create a Test Suite
    b. Create Test Classes for each Java Class which are supposed to be unit tested

(Test the service layer and repository layer classes)

5. Use **JEST** for testing your React Front End Code

Notes:

1. Perform **Unit Testing** after developing the Test Cases & Test Scenarios
2. Documents to be Submitted:
    a. W3H Analysis Document
    b. Test Case Document
    c. JUnit Classes
    d. Jest Tests
    e. Test Success /Failure Screens

W3H Analysis: Bank Management System

What	How
What are user stories that are present here? **Ans** • Create/Update/Delete Payee • Money transaction & record the histories of transaction **<u>Front End</u>** **What are components present in my user stories?** **Ans** • Home Component ○ Display all payee ○ Add payee ○ Delete payee ○ Edit payee • Transfer Money • Transaction Histories • Utilities Components ○ Output ○ Search **What is the testing type for testing the above components?** **Ans** Unit & Component Testing Using Jest Framework **What are the details which need to be tested in home component?**	**How were we going to perform the jest testing to frontend part which made using React JS?** **Ans** • Testing through pages • Testing through Services • Testing through Components **How will the component be tested?** **Ans** • Testing only Text • Testing only Input fields • Testing only Buttons • All these above **How will the Headers of page be tested?** **Ans** • By using the toHaveTextConent() • By using the toBeInTheDocument() • By using both the method **How to test the table were present in the listing page?** **Ans** • By assigning a role as 'Table' • By assigning a title as 'Name_Table' • By assigning a title as 'Table'

Ans

- Display all payee record
  - Header of list
  - Displaying Table
  - Payee Record fetching
- Add Payee
  - All Input fields
  - Add Button
  - Record insertion
- Edit Payee
  - All Input fields
  - Edit Button
  - Record updating
- Delete Payee
  - Delete Button
  - Record deletion

What are the details which need to be tested in Transfer Money component?
Ans

- All Input Fields
- Header of Transfer Money Page
- Transfer Button
- Functionality of money transferring

What are the details which need to be tested in Transaction History component?

- Header of list
- Displaying Table
- Payee Record fetching

<u>Backend</u>

What is the testing type for testing the backend?

How to test the button in all the components?
Ans

- By assigning test id as unique
- By assigning role as ‘button’
- By assigning title as ‘button’

How to test the input fields in the components?
Ans

- By using getByTestId
- By using getByLabel
- By using getByRole

How can we perform unit testing such that test cases to be tested for each functionality in JUnit testing?
Ans

- Testing with only success cases
- Testing with only failure cases
- Testing with both Cases

How can we test the methods in the service class?
Ans

- By Instantiating service class object
- By Declaring ‘@Autowired’ annotation on service class reference
- By Using Constructor injection to inject service class reference

How can we perform testing for the insert/Update functionality?
Ans

- By using assertEquals method
- By using assertNull method

Ans Unit Testing using JUnit framework **What were we going to test in backend?** **Ans** • Functionality of Each Service Method ○ Insert ○ Delete ○ Update ○ Find ○ Find all • Dataflow **What are services classes presented for performing the unit testing?** **Ans** • Payee Service • Transaction Service	• By using assertTrue method **How can we perform testing for the Delete functionality?** **Ans** • By using assertNull method • By using assertEquals method • By using assertTrue method **How can we perform testing for the find functionality?** **Ans** • By using assertNotNull method • By using assertEquals method • By using assertTrue method

Why	**Why Not**
How were we going to perform the jest testing to frontend part which made using React JS? **Ans** • Testing through Components (Because Unit testing is specified for the component such that component set of code will different functionality) **How will the component be tested?** **Ans** • All these above (Because these field will present in the UI Page, as developer we need check	**How were we going to perform the jest testing to frontend part which made using React JS?** **Ans** Because Unit testing is specified for the component such that pages and service cannot be tested **How will the component be tested?** **Ans** Because these field will present in the UI Page, as developer we do not test any specific functionality

that all mentioned components are presented) **How will the Headers of page be tested?** **Ans** • By using both the method (Because we need to check both feature through specified test id) **How to test the table were present in the listing page?** **Ans** • By assigning a title as ‘Name_Table’ (Because the method should be ambiguous while selecting using the common name as ‘Table’) **How to test the button in all the components?** **Ans** • By assigning test id as unique (Because to avoid the ambiguity while testing so safer side defining the test id is a better option) **How to test the input fields in the components?** **Ans** • By using getByLabel (Because in my situation I'm using Material UI component for input fields such that using label means more effective) **How can we perform unit testing such that test cases to be tested for each functionality in JUnit testing?**	**How will the Headers of page be tested?** **Ans** Because toBeInTheDocument() is for checking that is present or not, and toHaveTextContent() is for checking that value of the header so we can’t come with only one method **How to test the table were present in the listing page?** **Ans** Because there will be chance of ambiguity while selecting using the common name as ‘Table’ so we should avoid unnecessary. **How to test the button in all the components?** **Ans** Because there will be chance of ambiguity when selecting the buttons **How to test the input fields in the components?** **Ans** Because in my situation I'm using Material UI component for input field such that role and title doesn’t have that much advantage **How can we perform unit testing such that test cases to be tested for each functionality in JUnit testing?** **Ans** Because we cannot test only with specific cases **How can we test the methods in the service class?**

Ans • Testing with both Cases (Because testing should ensure all cases to produce a quality product) **How can we test the methods in the service class?** **Ans** • By Declaring '@Autowired' annotation on service class reference (Because using the autowired annotation object were implicitly instantiated so easy to be used) **How can we perform testing for the insert/Update functionality?** **Ans** • By using assertEquals method (Because this method will check or compare the string message return form the insert/Update Method) **How can we perform testing for the Delete functionality?** **Ans** • By using assertEquals method (Because this method will check or compare the string message return form the delete Method) **How can we perform testing for the find functionality?** **Ans** • By using assertNotNull method (Because Find method will return an object so that assertNotNull will be the exact one)	**Ans** Because constructor injecting will not support while performing JUnit testing And Hard coding should be avoided while creating a spring boot application to maintain the clean code standard **How can we perform testing for the insert/Update functionality?** **Ans** Because this method will check or compare the string message return form the insert/Update Method **How can we perform testing for the Delete functionality?** **Ans** Because this method will check or compare the string message return form the delete Method **How can we perform testing for the find functionality?** **Ans** • By using assertNotNull method Because other method is not defined to checking the nullability of method returning object

Jest Test Cases:

Home.test.js:

```
import '@testing-library/jest-dom';
import { render, screen } from '@testing-library/react';
import renderer from 'react-test-renderer';
import Home from '../Components/MainComponents/Home';
import { doGetAllPayee } from '../Components/Services/Services';

describe('Checking for Home Page Rendering', () => {

  test('should render home page', () => {
    render(<Home />)
    expect(screen.getByText('List of Payee')).toBeInTheDocument();
  });

  test('should check the header', () => {
    render(<Home />)
    expect(screen.getByTestId('homeHeader')).toBeInTheDocument();
    expect(screen.getByTestId('homeHeader')).toHaveTextContent('List of Payee');
  });

  test('should check rendering the table', () => {
    render(<Home />)
    expect(screen.getByTitle('payeeTable')).toBeInTheDocument();
  });

  test('should return payee list', () => {
    const result = doGetAllPayee();
    expect(result).toBeDefined();
    result.then(res => {
      expect(res.status).toEqual(200);
      expect(res.data).toEqual([]);
    });
  });

  test('Snapshot Testing for Home Page', () => {
    expect(renderer.create(<Home />).toJSON()).toMatchSnapshot();
  });
});
```

AddPayee.test.js:

```
import '@testing-library/jest-dom';
import { render, screen } from '@testing-library/react';
import AddPayee from '../Components/MainComponents/AddPayee';
import { doAdd } from '../Components/Services/Services';

const Flags = {
    isAddable: true,
}

describe('Checking Add page is Rendering', () => {

    test('Should have add fields', () => {
        render(<AddPayee Flags={Flags} />)
        expect(screen.getByLabelText("Full Name")).toBeInTheDocument();
        expect(screen.getByLabelText("Nick Name")).toBeInTheDocument();
        expect(screen.getByLabelText("Account Number")).toBeInTheDocument();
    })


    test('should insert the data with mock data', () => {
        const mockData = {
            payeeId: 11,
            payeeName: "Test010",
            nickName: "Test010",
            account: {
                accountNumber: "1234567821",
            }
        }
        const result = doAdd(mockData);
        expect(result).toBeDefined();
        result.then(res => {
            expect(res.status).toEqual(200);
            expect(res.data).toEqual("success");
        });

    });
});
```

EditPayee.test.js:

```
import '@testing-library/jest-dom';
import { render, screen } from '@testing-library/react';
import EditPayee from '../Components/MainComponents/EditPayee';
import { doGetAllPayee, doUpdate } from '../Components/Services/Services';

const Flags = {
    isEditable: true,
}



describe('Checking Edit page is Rendering', () => {
    test('Should have add fields', () => {
        render(<EditPayee Flags={Flags} id={1} />)
        expect(screen.getByLabelText("Full Name")).toBeInTheDocument();
        expect(screen.getByLabelText("Nick Name")).toBeInTheDocument();
    })




    test('should return a payee data', () => {
        const result = doGetAllPayee(10);
        expect(result).toBeDefined();
        result.then(res => {
            expect(res.status).toEqual(200);
            expect(res.data.payeeName).toEqual("Test009");
            expect(res.data.nickName).toEqual("Test009");
        }
        );
    });

    test('should update the data with mock data', () => {
        const mockData = {
            payeeId:10,
            payeeName: "Test111",
            nickName: "Test0111",
            account: {
                accountNumber: "1234567888",
            }
        }
        const result = doUpdate(mockData);
        expect(result).toBeDefined();
        result.then(res => {
            expect(res.status).toEqual(200);
            expect(res.data).toEqual("success");
        });
    });
});
```

DeletePayee.test.js:

```
import '@testing-library/jest-dom';
import { render, screen } from '@testing-library/react';
import DeletePayee from '../Components/MainComponents/DeletePayee';
import { doDelete } from '../Components/Services/Services';

const Flags = {
    isDeletable: true
}

describe('Checking the Delete page components', () => {

    test('should render the delete page', () => {
        render(<DeletePayee Flags={Flags} id={1} />);
        expect(screen.getByText("Are you confirm to
delete ?")).toBeInTheDocument();
    });

    test('should have the Button and Content', () => {
        render(<DeletePayee Flags={Flags} id={1} />);
        expect(screen.getByTestId("warning")).toBeInTheDocument();
        expect(screen.getByTestId("confirmBtn")).toBeInTheDocument();
        expect(screen.getByTestId("cancelBtn")).toBeInTheDocument();
    });

    test('should delete a payee data', () => {
        const result = doDelete(22);
        expect(result).toBeDefined();
        result.then((res) => {
            expect(res.data).toEqual("success");
            expect(res.status).toEqual(200);
        })
    });




});
```

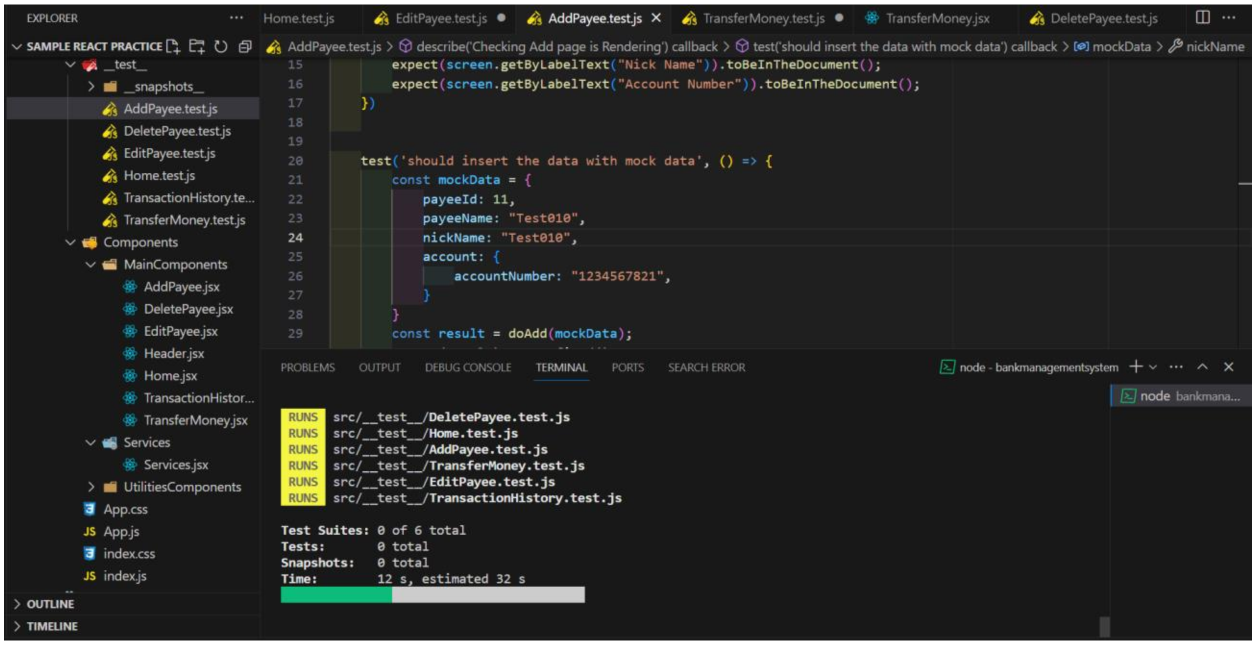
TransactionHistory.test.js:

```
import '@testing-library/jest-dom';
import { render, screen } from '@testing-library/react';
import renderer from 'react-test-renderer';
import TransactionHistory from '../Components/MainComponents/TransactionHistory';
import { doGetAllTransaction } from '../Components/Services/Services';

describe('Checking Transaction History Page', () => {

    test('should render the Transaction History Page', () => {
        render(<TransactionHistory />)
        expect(screen.getByText("History Of Transaction")).toBeInTheDocument();
    });

    test('should check rendering the table', () => {
        render(<TransactionHistory />)
        expect(screen.getByTestId("historyTable")).toBeDefined();
    });

    test('should return history of transaction list', () => {
        const result = doGetAllTransaction();
        expect(result).toBeDefined();
        result.then(res => {
            expect(res.status).toEqual(200);
            expect(res.data).toEqual([]);
        });
    });

    test('Snapshot Testing for History Page', () => {
        expect(renderer.create(<TransactionHistory
/>).toJSON()).toMatchSnapshot();
    });
});
```

TransferMoney.test.js:

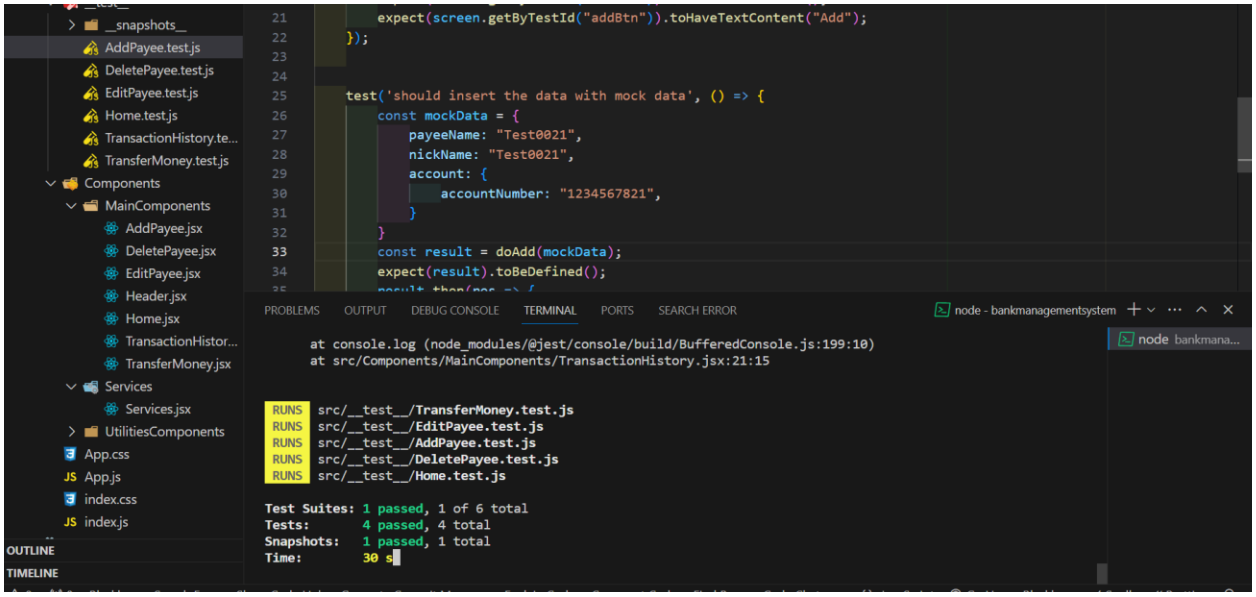
```
import '@testing-library/jest-dom';
import { render, screen } from '@testing-library/react';
import renderer from 'react-test-renderer';
import TransferMoney from '../Components/MainComponents/TransferMoney';
import { doSave } from '../Components/Services/Services';
jest.mock('react-router-dom')

describe('Checking the Transfer Money Component', () => {

    test('should render the Transfer Money Component', () => {
        render(<TransferMoney />)
        const transferMoney = screen.getByText('Enter details to Transfer
Money');
        expect(transferMoney).toBeInTheDocument();
    });

    test('should render the Header of transfer money component', () => {
        render(<TransferMoney />)
        expect(screen.getByTestId('transferHeader')).toBeInTheDocument();
        expect(screen.getByTestId('transferHeader')).toHaveTextContent('Enter
details to Transfer Money');
    });

    test('should check the field in transfer money component', () => {
        render(<TransferMoney />)
        expect(screen.getByLabelText("amount")).toBeInTheDocument();
        expect(screen.getByLabelText("payeeId")).toBeInTheDocument();
    });

    test('should perform the transfer method', () => {
        const data = {
            amount: 2000,
            payee: {
                payeeId: 10
            }
        }
        const result = doSave(data)
        result.then(res => {
                expect(res.status).toBe(200);
                expect(res.data).toEqual("success");
                })
    });


    test('should render the Transfer Money Component', () => {
        expect(renderer.create(<TransferMoney />).toJSON()).toMatchSnapshot();
    });
});
```

Output for Jest Testing:



```
        expect(screen.getByLabelText("Nick Name")).toBeInTheDocument();
        expect(screen.getByLabelText("Account Number")).toBeInTheDocument();
    })

    test('should insert the data with mock data', () => {
        const mockData = {
            payeeId: 11,
            payeeName: "Test010",
            nickName: "Test010",
            account: {
                accountNumber: "1234567821",
            }
        }
        const result = doAdd(mockData);
```

```
 RUNS  src/__test__/DeletePayee.test.js
 RUNS  src/__test__/Home.test.js
 RUNS  src/__test__/AddPayee.test.js
 RUNS  src/__test__/TransferMoney.test.js
 RUNS  src/__test__/EditPayee.test.js
 RUNS  src/__test__/TransactionHistory.test.js

Test Suites: 0 of 6 total
Tests:       0 total
Snapshots:   0 total
Time:        12 s, estimated 32 s
```

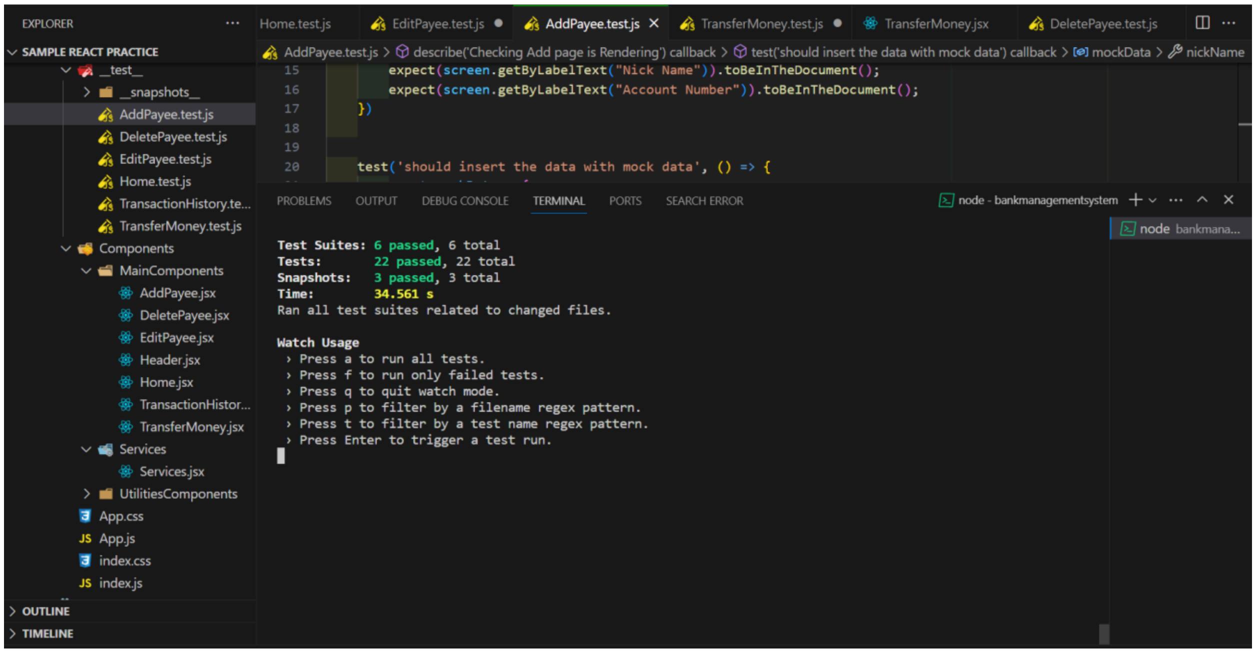


```
__snapshots__
AddPayee.test.js
DeletePayee.test.js
EditPayee.test.js
Home.test.js
TransactionHistory.te...
TransferMoney.test.js
Components
  MainComponents
    AddPayee.jsx
    DeletePayee.jsx
    EditPayee.jsx
    Header.jsx
    Home.jsx
    TransactionHistor...
    TransferMoney.jsx
  Services
    Services.jsx
  UtilitiesComponents
App.css
App.js
index.css
index.js
OUTLINE
TIMELINE
```

```js
        expect(screen.getByTestId("addBtn")).toHaveTextContent("Add");
    });

    test('should insert the data with mock data', () => {
        const mockData = {
            payeeName: "Test0021",
            nickName: "Test0021",
            account: {
                accountNumber: "1234567821",
            }
        }
        const result = doAdd(mockData);
        expect(result).toBeDefined();
        result.then(res => {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR — node - bankmanagementsystem

```
      at console.log (node_modules/@jest/console/build/BufferedConsole.js:199:10)
      at src/Components/MainComponents/TransactionHistory.jsx:21:15


 RUNS  src/__test__/TransferMoney.test.js
 RUNS  src/__test__/EditPayee.test.js
 RUNS  src/__test__/AddPayee.test.js
 RUNS  src/__test__/DeletePayee.test.js
 RUNS  src/__test__/Home.test.js

Test Suites: 1 passed, 1 of 6 total
Tests:       4 passed, 4 total
Snapshots:   1 passed, 1 total
Time:        30 s
```



EXPLORER — SAMPLE REACT PRACTICE

Home.test.js | EditPayee.test.js | AddPayee.test.js | TransferMoney.test.js | TransferMoney.jsx | DeletePayee.test.js

AddPayee.test.js > describe('Checking Add page is Rendering') callback > test('should insert the data with mock data') callback > mockData > nickName

```
__test__
  __snapshots__
  AddPayee.test.js
  DeletePayee.test.js
  EditPayee.test.js
  Home.test.js
  TransactionHistory.te...
  TransferMoney.test.js
Components
  MainComponents
    AddPayee.jsx
    DeletePayee.jsx
    EditPayee.jsx
    Header.jsx
    Home.jsx
    TransactionHistor...
    TransferMoney.jsx
  Services
    Services.jsx
  UtilitiesComponents
App.css
App.js
index.css
index.js
OUTLINE
TIMELINE
```

```js
        expect(screen.getByLabelText("Nick Name")).toBeInTheDocument();
        expect(screen.getByLabelText("Account Number")).toBeInTheDocument();
    })


    test('should insert the data with mock data', () => {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR — node - bankmanagementsystem

```
Test Suites: 6 passed, 6 total
Tests:       22 passed, 22 total
Snapshots:   3 passed, 3 total
Time:        34.561 s
Ran all test suites related to changed files.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```


JUnit Testing:

TestPayeeService.js:

```
package com.bms.service;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import com.bms.entity.Account;
import com.bms.entity.Payee;

@SpringBootTest
@TestMethodOrder(OrderAnnotation.class)
class TestPayeeService {

@Autowired
private PayeeService payeeService;

@Test
@Order(1)
public void testAddPayee1() {
Payee payee = new Payee();
Account account = new Account();
account.setAccountNumber(1234567890);
payee.setAccount(account);
payee.setPayeeId(1);
payee.setPayeeName("Sanjay Khanna");
payee.setNickName("sk");
assertEquals("success", payeeService.addPayee(payee));
}
```

```java
@Test
@Order(2)
public void testAddPayee2() {
Payee payee = new Payee();
Account account = new Account();
account.setAccountNumber(1234567880);
payee.setAccount(account);
payee.setPayeeId(2);
payee.setPayeeName("Sanjay Khanna");
payee.setNickName("s");
assertEquals("success", payeeService.addPayee(payee));
}

@Test
@Order(2)
public void testAddPayee3() {
Payee payee = new Payee();
Account account = new Account();
account.setAccountNumber(1234567899);
payee.setAccount(account);
payee.setPayeeId(3);
payee.setPayeeName("Sanjay Khanna");
payee.setNickName("san");
assertEquals("success", payeeService.addPayee(payee));

}

@Test
@Order(5)
public void testAddPayee4() {
Payee payee = new Payee();
payee.setPayeeName("Sanjay Khanna");
payee.setNickName("sk");
assertEquals("Failed to add Payee", payeeService.addPayee(payee));

}

@Test
@Order(5)
```

```
public void testAddPayee5() {
Payee payee = new Payee();
Account account = new Account();
account.setAccountNumber(1234567890);
payee.setAccount(account);
payee.setNickName("sk");
assertEquals("Failed to add Payee", payeeService.addPayee(payee));

}

@Test
@Order(6)
public void testAddPayee6() {
Payee payee = new Payee();
payee.setPayeeName("Sanjay Khanna");
Account account = new Account();
account.setAccountNumber(1234567890);
payee.setAccount(account);
assertEquals("Failed to add Payee", payeeService.addPayee(payee));

}

@Test
@Order(7)
public void testDeletePayee1() {
assertEquals("success", payeeService.deletePayee(1));
}

@Test
@Order(8)
public void testDeletePayee2() {
assertEquals("success", payeeService.deletePayee(2));
}

@Test
@Order(9)
public void testDeletePayee3() {
assertEquals("Failed to delete Payee", payeeService.deletePayee(1));
}
```

```java
@Test
@Order(10)
public void testDeletePayee4() {
assertEquals("Failed to delete Payee", payeeService.deletePayee(2));
}

@Test
@Order(11)
public void testGetPayee1() {
assertNotNull(payeeService.getPayee(3));
}

@Test
@Order(12)
public void testGetPayeeList1() {
assertEquals(true, payeeService.getPayeeList().size() > 0);
}

@Test
@Order(13)
public void testGetPayeeList2() {
assertEquals(false, payeeService.getPayeeList().size() == 0);
}

@Test
@Order(14)
public void testGetPayeeList3() {
assertNotNull(payeeService.getPayeeList());
}

@Test
@Order(15)
public void testUpdatePayee1() {
Payee payee = new Payee();
Account account = new Account();
account.setAccountNumber(1234567800);
payee.setAccount(account);
payee.setPayeeName("Sanjay");
payee.setNickName("sks");
assertEquals("success", payeeService.updatePayee(payee));

}

@Test
@Order(16)
public void testUpdatePayee2() {
Payee payee = new Payee();
Account account = new Account();
account.setAccountNumber(1234567800);
payee.setAccount(account);
payee.setPayeeName("Sanjay");
payee.setNickName("s");
assertEquals("Failed to update Payee",
payeeService.updatePayee(payee));

}




}
```

TestTransactionService.js

package com.bms.service;

import static org.junit.jupiter.api.Assertions.*;

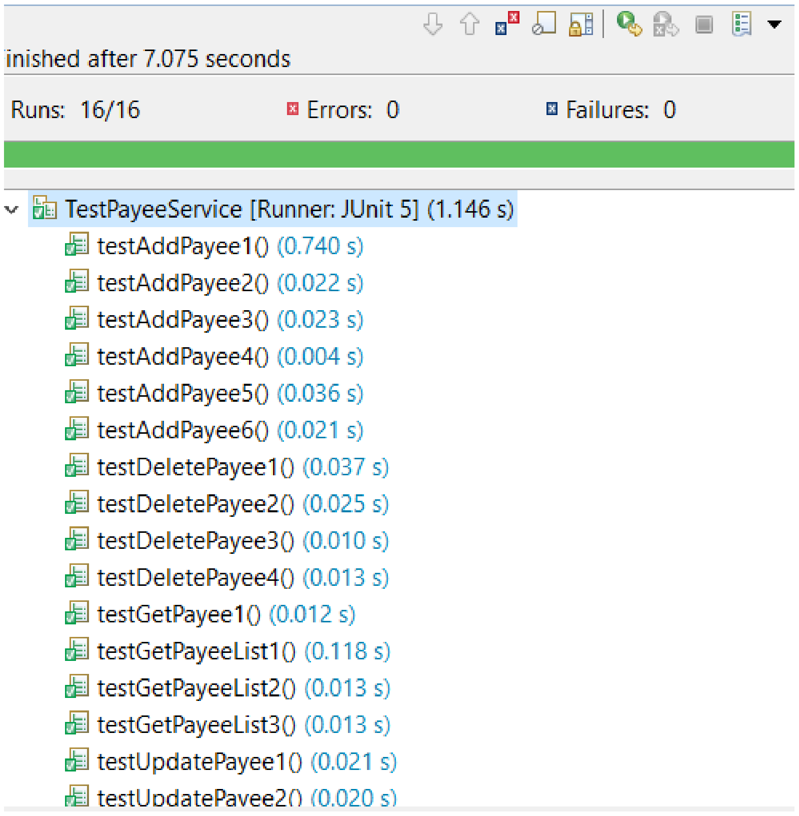
import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

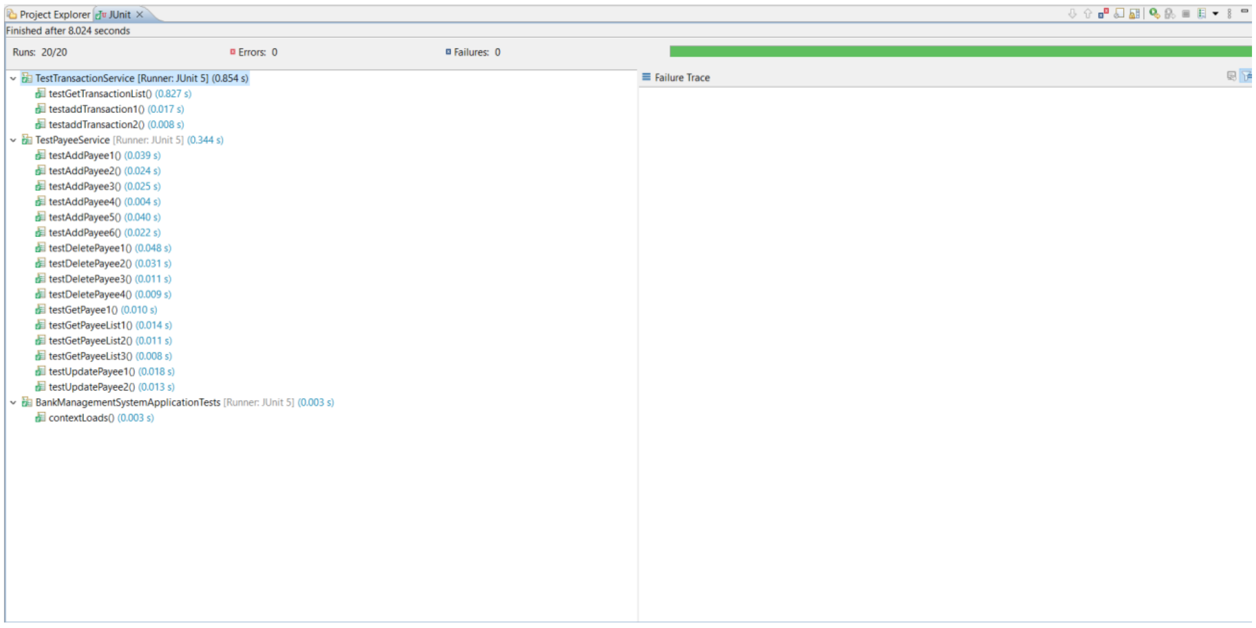
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

```
import org.springframework.boot.test.context.SpringBootTest;

import com.bms.entity.Payee;
import com.bms.entity.Transaction;

@SpringBootTest
@TestMethodOrder(OrderAnnotation.class)
class TestTransactionService {

private TransactionService transactionService;

@Test
void testaddTransaction1() {
Payee payee = new Payee();
payee.setPayeeId(3);
Transaction transaction = new Transaction();
transaction.setAmount(2000);
transaction.setPayee(payee);
assertEquals("success",
transactionService.addTransaction(transaction));
}

@Test
void testaddTransaction2() {
Payee payee = new Payee();
payee.setPayeeId(5);
Transaction transaction = new Transaction();
transaction.setAmount(2000);
transaction.setPayee(payee);
assertEquals("Failed",
transactionService.addTransaction(transaction));
}

@Test
void testGetTransactionList() {
assertNotNull(transactionService.getTransactionList());
}

}
```

Output:



```
inished after 7.075 seconds

Runs:  16/16          Errors:  0          Failures:  0

TestPayeeService [Runner: JUnit 5] (1.146 s)
    testAddPayee1() (0.740 s)
    testAddPayee2() (0.022 s)
    testAddPayee3() (0.023 s)
    testAddPayee4() (0.004 s)
    testAddPayee5() (0.036 s)
    testAddPayee6() (0.021 s)
    testDeletePayee1() (0.037 s)
    testDeletePayee2() (0.025 s)
    testDeletePayee3() (0.010 s)
    testDeletePayee4() (0.013 s)
    testGetPayee1() (0.012 s)
    testGetPayeeList1() (0.118 s)
    testGetPayeeList2() (0.013 s)
    testGetPayeeList3() (0.013 s)
    testUpdatePayee1() (0.021 s)
    testUpdatePayee2() (0.020 s)
```



```
Finished after 8.024 seconds

Runs:  20/20          Errors:  0          Failures:  0

TestTransactionService [Runner: JUnit 5] (0.854 s)
    testGetTransactionList() (0.827 s)
    testaddTransaction1() (0.017 s)
    testaddTransaction2() (0.008 s)
TestPayeeService [Runner: JUnit 5] (0.344 s)
    testAddPayee1() (0.039 s)
    testAddPayee2() (0.024 s)
    testAddPayee3() (0.025 s)
    testAddPayee4() (0.004 s)
    testAddPayee5() (0.040 s)
    testAddPayee6() (0.022 s)
    testDeletePayee1() (0.048 s)
    testDeletePayee2() (0.031 s)
    testDeletePayee3() (0.011 s)
    testDeletePayee4() (0.009 s)
    testGetPayee1() (0.010 s)
    testGetPayeeList1() (0.014 s)
    testGetPayeeList2() (0.011 s)
    testGetPayeeList3() (0.008 s)
    testUpdatePayee1() (0.018 s)
    testUpdatePayee2() (0.013 s)
BankManagementSystemApplicationTests [Runner: JUnit 5] (0.003 s)
    contextLoads() (0.003 s)
```