Software Testing, Validation, and

Verification HRMS system

Course Code: CSE 338

Summer 2024

Assessment Committ

Dr. Yasmine Afify

Eng Amr Hassan

Project by

Aley Amin Ahmed Shawky 21P0150

Hassan Walid Mohamed Almarsafy 21P0053

Mazen sameh shawky elshabassy 21P0070

Seif Elden Samir Mohamed 2101524

Table of Contents

Component testing

Class: PayrollTest

Explanation

The PayrollTest class is designed to test the functionality of the Payroll class, ensuring that the payroll calculations and the setting and getting of various attributes are correct. The tests cover different employee types (full-time, part-time, hourly, intern) and various scenarios, including invalid inputs.

Initialization and Setup

- **Payroll Objects**: Four Payroll instances are created for different employee types: full-time, part-time, hourly, and intern.

Test Methods

- **testFullTimePay()**: This method tests the pay calculation for a full-time employee.
  - **Input**: Full-time payroll details.
  - **Expected Output**: Calculated pay.
  - **Assertion**: assertEquals(expectedPay, fullTimePayroll.calculatePay())
- **testPartTimePay()**: This method tests the pay calculation for a part-time employee.
  - **Input**: Part-time payroll details.
  - **Expected Output**: Calculated pay.
  - **Assertion**: assertEquals(expectedPay, partTimePayroll.calculatePay())
- **testHourlyPay()**: This method tests the pay calculation for an hourly employee.
  - **Input**: Hourly payroll details.
  - **Expected Output**: Calculated pay.
  - **Assertion**: assertEquals(expectedPay, hourlyPayroll.calculatePay())
- **testInternPay()**: This method tests the pay calculation for an intern.
  - **Input**: Intern payroll details.

        - **Expected Output**: Calculated pay.
        - **Assertion**: assertEquals(expectedPay, internPayroll.calculatePay())
    - **testInvalidBaseSalary()**: This method tests the handling of an invalid base salary.
        - **Input**: Invalid base salary.
        - **Expected Output**: -1
        - **Assertion**: assertEquals(-1, invalidPayroll.getBaseSalary())
    - **testInvalidHours()**: This method tests the handling of invalid hours.
        - **Input**: Invalid hours.
        - **Expected Output**: -1
        - **Assertion**: assertEquals(-1, invalidPayroll.getHours())
    - **testInvalidTax()**: This method tests the handling of an invalid tax.
        - **Input**: Invalid tax.
        - **Expected Output**: -1
        - **Assertion**: assertEquals(-1, invalidPayroll.getTax())
    - **testInvalidDeductions()**: This method tests the handling of invalid deductions.
        - **Input**: Invalid deductions.
        - **Expected Output**: -1
        - **Assertion**: assertEquals(-1, invalidPayroll.getDeductions())
    - **testInvalidBonus()**: This method tests the handling of an invalid bonus.
        - **Input**: Invalid bonus.
        - **Expected Output**: -1
        - **Assertion**: assertEquals(-1, invalidPayroll.getBonus())
    - **testSetGetEmployeeType()**: This method tests the setEmployeeType and getEmployeeType methods.

        - **Input**: Setting employee type to part-time.
        - **Expected Output**: EmployeeType.PartTime
        - **Assertion**: assertEquals(EmployeeType.PartTime, fullTimePayroll.getEmployeeType())
    - **testSetGetBaseSalary()**: This method tests the setBaseSalary and getBaseSalary methods.
        - **Input**: Setting base salary to 6000.
        - **Expected Output**: 6000
        - **Assertion**: assertEquals(6000, fullTimePayroll.getBaseSalary())
    - **testSetGetHours()**: This method tests the setHours and getHours methods.
        - **Input**: Setting hours to 120.
        - **Expected Output**: 120
        - **Assertion**: assertEquals(120, hourlyPayroll.getHours())
    - **testSetGetTax()**: This method tests the setTax and getTax methods.
        - **Input**: Setting tax to 200.
        - **Expected Output**: 200
        - **Assertion**: assertEquals(200, internPayroll.getTax())
    - **testSetGetDeductions()**: This method tests the setDeductions and getDeductions methods.
        - **Input**: Setting deductions to 300.
        - **Expected Output**: 300
        - **Assertion**: assertEquals(300, partTimePayroll.getDeductions())
    - **testSetGetBonus()**: This method tests the setBonus and getBonus methods.
        - **Input**: Setting bonus to 300.
        - **Expected Output**: 300

- **Assertion**: assertEquals(300, fullTimePayroll.getBonus())

Table

Test Case no	Type	Tes Case Name	Description	Input	Output	Expected	Accepted
1	component	**testFullTimePay**	tests the pay calculation for a full-time employee.	Full-time payroll details.	Calculated pay	Calculated pay	yes
2	component	**testPartTimePay**	tests the pay calculation for a part-time employee.	Part-time payroll details.	Calculated pay.	Calculated pay.	yes
3	component	**testHourlyPay**	This method tests the pay calculation for an hourly employee.	Hourly payroll details.	Calculated pay.	Calculated pay.	yes
4	component	**testInternPay**	This method tests the pay calculation for an intern.	Intern payroll details.	Calculated pay.	Calculated pay.	yes
5	component	**testInvalidBaseSalary**	This method tests the handling of an invalid base salary.	Invalid base salary	-1	-1	yes
6	component	**testInvalidHours**	This method tests the handling of invalid hours.	Invalid hours	-1	-1	yes

7	component	**testInvalidTax**	**()**: This method tests the handling of an invalid tax	Invalid tax.	-1	-1	yes
8	component	**testInvalidDeductions**	This method tests the handling of invalid deductions	Invalid deductions.	-1	-1	yes
9	component	**testInvalidBonus**	This method tests the handling of an invalid bonus.	Invalid bonus.	-1	-1	yes
10	component	**testSetGetEmployeeType**	This method tests the setEmployeeType and getEmployeeType methods.	Setting employee type to part-time.	EmployeeType.PartTime	EmployeeType.PartTime	yes
11	component	**testSetGetBaseSalary**	tests the setBaseSalary and getBaseSalary methods.	Setting base salary to 6000.	6000	6000	yes
12	component	**testSetGetHours**	This method tests the setHours and getHours methods	Setting hours to 120.	120	120	yes
13	component	**testSetGetTax**	This method tests the setTax and getTax methods	Setting tax to 200.	200	200	yes

14	component	**testSetGetDeductions**	This method tests the setDeductions and getDeductions methods.	Setting deductions to 300.	300	300	yes
15	component	**testSetGetBonus**	This method tests the setBonus and getBonus methods.	Setting bonus to 300.	300	300	Yes

Class Address

Explanation

In the AddressTest class, we create an instance called address from the Address class. This class contains various test methods to verify the functionality of the Address class. The setup for each test method ensure the tests run in isolation and do not affect each other.

Initialization

- **Message()**: The @BeforeAll annotation indicates that this method will be executed once before all the test methods in the class. It prints a message indicating that the Address class tests are starting.

Setup

- **setUp()**: The @BeforeEach annotation indicates that this method will be executed before each test method. It initializes an Address object with predefined values ("Zahraa ElMaadi", "Cairo", "00000", "Egypt").

Test Methods

- **testGetStreet()**: The @Test and @DisplayName annotations indicate that this is a test method with a descriptive name. It verifies that the getStreet method returns the correct street name.
  - **Input**: address.getStreet()
  - **Expected Output**: "Zahraa ElMaadi"
  - **Actual Output**: "Zahraa ElMaadi"
  - **Assertion**: assertEquals("Zahraa ElMaadi", address.getStreet())
- **testSetStreet()**: This test method verifies that the setStreet method correctly updates the street name.
  - **Input**: address.setStreet("Makram")
  - **Expected Output**: "Makram"
  - **Actual Output**: "Makram"
  - **Assertion**: assertEquals("Makram", address.getStreet())
- **testGetCity()**: This test method verifies that the getCity method returns the correct city name.
  - **Input**: address.getCity()
  - **Expected Output**: "Cairo"
  - **Actual Output**: "Cairo"
  - **Assertion**: assertEquals("Cairo", address.getCity())
- **testSetCity()**: This test method verifies that the setCity method correctly updates the city name.
  - **Input**: address.setCity("Giza")
  - **Expected Output**: "Giza"

        - **Actual Output**: "Giza"
        - **Assertion**: assertEquals("Giza", address.getCity())
    - **testGetPostalCode()**: This test method verifies that the getPostalCode method returns the correct postal code.
        - **Input**: address.getPostalCode()
        - **Expected Output**: "00000"
        - **Actual Output**: "00000"
        - **Assertion**: assertEquals("00000", address.getPostalCode())
    - **testSetPostalCode()**: This test method verifies that the setPostalCode method correctly updates the postal code.
        - **Input**: address.setPostalCode("77102")
        - **Expected Output**: "77102"
        - **Actual Output**: "77102"
        - **Assertion**: assertEquals("77102", address.getPostalCode())
    - **testGetCountry()**: This test method verifies that the getCountry method returns the correct country name.
        - **Input**: address.getCountry()
        - **Expected Output**: "Egypt"
        - **Actual Output**: "Egypt"
        - **Assertion**: assertEquals("Egypt", address.getCountry())
    - **testSetCountry()**: This test method verifies that the setCountry method correctly updates the country name.
        - **Input**: address.setCountry("England")
        - **Expected Output**: "England"
        - **Actual Output**: "England"
        - **Assertion**: assertEquals("England", address.getCountry())

Table

Test Case no	Type	Tes Case Name	Description	Input	Output	Expected	Accepted
1	component	**testGetStreet**	It verifies that the getStreet method returns the correct street name.	address.getStreet()	Zahraa ElMaadi	Zahraa ElMaadi	yes
2	component	**testSetStreet**	This test method verifies that the setStreet method correctly updates the street name.	address.setStreet("Makram")	Makram"	Makram"	yes
3	component	**testGetCity**	This test method verifies that the getCity method returns the correct city name.	address.getCity()	Cairo	Cairo	yes
4	component	**testSetCity**	This test method verifies that the setCity method correctly updates the city name.	address.setCity("Giza")	Giza	Giza	yes
5	component	**testGetPostalCode**	This test method verifies that the getPostalCode method returns the correct postal code	address.getPostalCode()	"00000"	"00000"	yes
6	component	**testSetPostalCode**	This test method verifies that the setPostalCode method correctly updates the postal code.	address.setPostalCode("77102")	"77102"	"77102"	yes
7	component	**testGetCountry**	This test method verifies that the getCountry method returns the correct country name	address.getCountry()	"Egypt"	"Egypt"	yes

8	component	**testSetCountry**	This test method verifies that the setCountry method correctly	address.setCountry("England")	"England"	"England"	yes

Class: Leave Management Test

Explanation

In the LeaveManagementTest class, we create an instance of LeaveManagement and LeaveRequest objects to test the functionality of the leave management system. The setup and teardown methods ensure that each test runs in isolation and does not interfere with the others.

Initialization and Setup

- **setUp()**: The @BeforeEach annotation indicates that this method will be executed before each test method. It initializes a LeaveManagement object and two LeaveRequest objects with predefined values. The Employee and Address objects are also created to be associated with the leave requests.
    - **Initialization Details**:
        - LeaveManagement leaveManagement: Instance of the LeaveManagement class.
        - LeaveRequest leaveRequest1 and LeaveRequest leaveRequest2: Instances of the LeaveRequest class with the same start and end dates, associated with an Employee named "Mazen".

Test Methods

- **testAddLeaveRequest()**: This method tests the addLeaveRequest method of the LeaveManagement class. It verifies that adding a leave request increases the size of the leave requests list.
    - **Input**: leaveManagement.addLeaveRequest(leaveRequest1)
    - **Expected Output**: The size of the leave requests list should be 1.
    - **Assertion**: assertEquals(1, leaveManagement.getAllLeaveRequests().size())
- **testRemoveLeaveRequest()**: This method tests the removeLeaveRequest method. It verifies that removing a leave request decreases the size of the leave

requests list.

    - **Input**: leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.removeLeaveRequest(1)
    - **Expected Output**: The size of the leave requests list should be 0.
    - **Assertion**: assertTrue(removed), assertEquals(0, leaveManagement.getAllLeaveRequests().size())

- **testGetLeaveRequest()**: This method tests the getLeaveRequest method. It verifies that the correct leave request is returned based on its ID.
    - **Input**: leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.getLeaveRequest(1)
    - **Expected Output**: The leave request with ID 1 should be returned and not null.
    - **Assertion**: assertNotNull(request), assertEquals(1, request.getId())
- **testGetAllLeaveRequests()**: This method tests the getAllLeaveRequests method. It verifies that all leave requests are returned correctly.
    - **Input**: leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.addLeaveRequest(leaveRequest2), leaveManagement.getAllLeaveRequests()
    - **Expected Output**: The size of the leave requests list should be 2.
    - **Assertion**: assertEquals(2, requests.size())
- **testUpdateLeaveStatus()**: This method tests the updateLeaveStatus method. It verifies that the leave status of a request is updated correctly.
    - **Input**: leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.updateLeaveStatus(1, LeaveStatus.Accepted), leaveManagement.getLeaveRequest(1)
    - **Expected Output**: The leave status should be updated to LeaveStatus.Accepted.
    - **Assertion**: assertEquals(LeaveStatus.Accepted, request.getLeaveStatus())
- **testApproveLeaveRequest()**: This method tests the approveLeaveRequest

method. It verifies that a leave request is approved correctly.

- **Input**: leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.approveLeaveRequest(1), leaveManagement.getLeaveRequest(1)
- **Expected Output**: The leave status should be updated to LeaveStatus.Accepted.
- **Assertion**: assertEquals(LeaveStatus.Accepted, request.getLeaveStatus())

- **testRejectLeaveRequest()**: This method tests the rejectLeaveRequest method. It verifies that a leave request is rejected correctly.
  - **Input**: leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.rejectLeaveRequest(1), leaveManagement.getLeaveRequest(1)
  - **Expected Output**: The leave status should be updated to LeaveStatus.Rejected.
  - **Assertion**: assertEquals(LeaveStatus.Rejected, request.getLeaveStatus())

Table

TestCaseno	**Type**	**Tes Case Name**	**Description**	**Input**	**Output**	**Expected**	**Accepted**
1	**component**	**testAddLeaveRequest**	**This method tests the addLeaveRequest method of the LeaveMa**	**leaveManagement.addLeaveRequest(leaveRequest1)**	**1**	**The size of the leave requests list should be 1**	**yes**

			nagement class. It verifies that adding a leave request increases the size of the leave requests list.				
2	**component**	**testRemoveLeaveRequest**	**This method tests the removeLeaveRequest method. It verifies that removing a leave request decreases the size of the leave requests list.**	**leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.removeLeaveRequest(1)**	**0**	**The size of the leave requests list should be 0**	**yes**
3	**component**	**testGetLeaveRequest**	**This method tests the getLeaveRequest method. It verifies that the correct leave**	**: leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.getLeaveRequest(1)**	**The leave request with ID 1 should be returned and not null.**	**: The leave request with ID 1 should be returned and not null.**	**yes**

			request is returned based on its ID.				
4	**component**	**testGetAllLeaveRequests**	**This method tests the getAllLeaveRequests method. It verifies that all leave requests are returned correctly.**	**: leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.addLeaveRequest(leaveRequest2), leaveManagement.getAllLeaveRequests()**	**2**	**The size of the leave requests list should be 2**	**yes**
5	**component**	**testUpdateLeaveStatus**	**This method tests the updateLeaveStatus method. It verifies that the leave status of a request is updated correctly**	**leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.updateLeaveStatus(1, LeaveStatus.Accepted), leaveManagement.getLeaveRequest(1)**	**The leave status updated to LeaveStatus.Accepted.**	**The leave status should be updated to LeaveStatus.Accepted.**	**yes**
6	**component**	**testApproveLeaveRequest**	**This method tests the approve LeaveRe**	**leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.approveLeaveRequest(1**	**The leave status updated to LeaveSta**	**The leave status should be updated**	**yes**

			quest method. It verifies that a leave request is approved correctly	**), leaveManagement.getLeaveRequest(1)**	**tus.Accepted.**	**to LeaveStatus.Accepted.**	
7	**component**	**testRejectLeaveRequest**	**This method tests the rejectLeaveRequest method. It verifies that a leave request is rejected correctly.**	**leaveManagement.addLeaveRequest(leaveRequest1), leaveManagement.rejectLeaveRequest(1), leaveManagement.getLeaveRequest(1)**	**The leave status updated to LeaveStatus.Rejected.**	**The leave status should be updated to LeaveStatus.Rejected.**	**yes**

Class: LeaveRequestTest

Explanation

In the LeaveRequestTest class, we create an instance of LeaveRequest and related objects to test the functionality of the LeaveRequest class. The setup methods ensures that each test runs in isolation and does not interfere with the others.

Initialization and Setup

- **Message()**: The @BeforeAll annotation indicates that this method will be

executed once before all the test methods in the class. It prints a message indicating that the LeaveRequest class tests are starting.

- **setup()**: The @BeforeEach annotation indicates that this method will be executed before each test method. It initializes a LeaveRequest object with predefined values along with an Employee and Address object associated with it.
    - **Initialization Details**:
        - Employee employee: Instance of the Employee class named "Mazen".
        - Date startDate and Date endDate: Start and end dates for the leave request.
        - LeaveRequest leaveRequest: Instance of the LeaveRequest class with ID 1, associated with the Employee object, and set with leave type VacationLeave.

Test Methods

- **testGetId()**: This method tests the getId method of the LeaveRequest class. It verifies that the ID of the leave request is returned correctly.
    - **Input**: leaveRequest.getId()
    - **Expected Output**: 1
    - **Assertion**: Assertions.assertEquals(1, leaveRequest.getId())
- **testSetId()**: This method tests the setId method. It verifies that the ID of the leave request is set correctly.
    - **Input**: leaveRequest.setId(2)
    - **Expected Output**: 2
    - **Assertion**: Assertions.assertEquals(2, leaveRequest.getId())
- **testGetEmployee()**: This method tests the getEmployee method. It verifies that the employee associated with the leave request is returned correctly.
    - **Input**: leaveRequest.getEmployee()
    - **Expected Output**: employee
    - **Assertion**: Assertions.assertEquals(employee,

leaveRequest.getEmployee())

- **testSetEmployee()**: This method tests the setEmployee method. It verifies that the employee associated with the leave request is set correctly.
  - **Input**: Employee newEmployee = new Employee("Ali", 70, "Ali123", "newpassword", new Address("Heliopolis", "Cairo", "11111", "Egypt"), "Mechanical Engineering", EmployeeType.FullTime, Evaluation.Excellent); leaveRequest.setEmployee(newEmployee)
  - **Expected Output**: newEmployee
  - **Assertion**: Assertions.assertEquals(newEmployee, leaveRequest.getEmployee())
- **testGetLeaveType()**: This method tests the getLeaveType method. It verifies that the leave type of the leave request is returned correctly.
  - **Input**: leaveRequest.getLeaveType()
  - **Expected Output**: LeaveType.VacationLeave
  - **Assertion**: Assertions.assertEquals(LeaveType.VacationLeave, leaveRequest.getLeaveType())
- **testSetLeaveType()**: This method tests the setLeaveType method. It verifies that the leave type of the leave request is set correctly.
  - **Input**: leaveRequest.setLeaveType(LeaveType.SickLeave)
  - **Expected Output**: LeaveType.SickLeave
  - **Assertion**: Assertions.assertEquals(LeaveType.SickLeave, leaveRequest.getLeaveType())
- **testGetStartDate()**: This method tests the getStartDate method. It verifies that the start date of the leave request is returned correctly.
  - **Input**: leaveRequest.getStartDate()
  - **Expected Output**: startDate
  - **Assertion**: Assertions.assertEquals(startDate, leaveRequest.getStartDate())
- **testSetStartDate()**: This method tests the setStartDate method. It verifies that

the start date of the leave request is set correctly.

    - **Input**: Date newStartDate = new Date(2024, Calendar.AUGUST, 5); leaveRequest.setStartDate(newStartDate)
    - **Expected Output**: newStartDate
    - **Assertion**: Assertions.assertEquals(newStartDate, leaveRequest.getStartDate())

- **testGetEndDate()**: This method tests the getEndDate method. It verifies that the end date of the leave request is returned correctly.
    - **Input**: leaveRequest.getEndDate()
    - **Expected Output**: endDate
    - **Assertion**: Assertions.assertEquals(endDate, leaveRequest.getEndDate())
- **testSetEndDate()**: This method tests the setEndDate method. It verifies that the end date of the leave request is set correctly.
    - **Input**: Date newEndDate = new Date(2024, Calendar.AUGUST, 15); leaveRequest.setEndDate(newEndDate)
    - **Expected Output**: newEndDate
    - **Assertion**: Assertions.assertEquals(newEndDate, leaveRequest.getEndDate())
- **testGetLeaveStatus()**: This method tests the getLeaveStatus method. It verifies that the leave status of the leave request is returned correctly.
    - **Input**: leaveRequest.getLeaveStatus()
    - **Expected Output**: LeaveStatus.Pending
    - **Assertion**: Assertions.assertEquals(LeaveStatus.Pending, leaveRequest.getLeaveStatus())
- **testSetLeaveStatus()**: This method tests the setLeaveStatus method. It verifies that the leave status of the leave request is set correctly.
    - **Input**: leaveRequest.setLeaveStatus(LeaveStatus.Pending)

- **Expected Output**: LeaveStatus.Pending
- **Assertion**: Assertions.assertEquals(LeaveStatus.Pending, leaveRequest.getLeaveStatus())

Table

TestCaseno	**Type**	**Tes Case Name**	**Description**	**Input**	**Output**	**Expected**	**Accepted**
1	component	**testGetId**	This method tests the getId method of the LeaveRequest class. It verifies that the ID of the leave request is returned correctly	leaveRequest.getId()	1	1	yes
2	component	**testSetId**	This method tests the setId method. It verifies that the ID of the leave request is set correctly.	leaveRequest.setId(2)	2	2	yes

3	component	**testGetEmployee**	This method tests the getEmployee method. It verifies that the employee associated with the leave request is returned correctly	leaveRequest.getEmployee()	employee	employee	yes
4	component	**testGetLeaveType**	This method tests the getLeaveType method. It verifies that the leave type of the leave request is returned correctly	leaveRequest.getLeaveType()	LeaveType.VacationLeave	LeaveType.VacationLeave	yes
5	component	**testSetLeaveType**	This method tests the setLeaveType method. It verifies that the leave type of the leave request is set correctly	: leaveRequest.setLeaveType(LeaveType.SickLeave)	LeaveType.SickLeave	LeaveType.SickLeave	yes
6	compon	**testGetStartDat**	This method	leaveRequest.getStartDate()	startDate	startDate	yes

	ent	**e**	tests the getStartDate method. It verifies that the start date of the leave request is returned correctly				
7	component	**testSetStartDate**	This method tests the setStartDate method. It verifies that the start date of the leave request is set correctly.	Date newStartDate = new Date(2024, Calendar.AUGUST, 5); leaveRequest.setStartDate(newStartDate)	newStartDate	newStartDate	yes
8	component	**testGetEndDate**	This method tests the getEndDate method. It verifies that the end date of the leave request is returned correctly	leaveRequest.getEndDate()	endDate	endDate	yes
9	component	**testSetEndDate**	This method tests the setEndDate method. It verifies that	Date newEndDate = new Date(2024, Calendar.AUGUST, 15); leaveRequest.setEndDate(newEndDate)	newEndDate	newEndDate	yes

			the end date of the leave request is set correctly.				
10	component	**testGetLeaveStatus**	This method tests the getLeaveStatus method. It verifies that the leave status of the leave request is returned correctly	leaveRequest.getLeaveStatus()	LeaveStatus.Pending	LeaveStatus.Pending	yes
11	component	**testSetLeaveStatus**	This method tests the setLeaveStatus method. It verifies that the leave status of the leave request is set correctly.	leaveRequest.setLeaveStatus(LeaveStatus.Pending)	LeaveStatus.Pending	LeaveStatus.Pending	yes

Class: HRemployeeTest

Explanation

In the HRemployeeTest class, we create instances of HRemployee and related objects to test the functionality of the HR employee management system. The setup ensures that each test runs in isolation and does not interfere with the others.

Initialization and Setup

- **setUp()**: The @BeforeEach annotation indicates that this method will be executed before each test method. It initializes an HRemployee object and two Employee objects with predefined values.
    - **Initialization Details**:
        - HRemployee hrEmployee: Instance of the HRemployee class.
        - Employee employee1 and Employee employee2: Instances of the Employee class created using the createEmployee method of the HRemployee class.

Test Methods

- **testCreateEmployee()**: This method tests the createEmployee method of the HRemployee class. It verifies that employees are created correctly.
    - **Input**: hrEmployee.createEmployee(...) for employee1 and employee2
    - **Expected Output**: employee1 and employee2 should not be null.
    - **Assertion**: assertNotNull(employee1), assertNotNull(employee2)
- **testFindEmployeeById()**: This method tests the findEmployeeById method. It verifies that an employee can be found by their ID.
    - **Input**: hrEmployee.findEmployeeById(101)
    - **Expected Output**: employee1
    - **Assertion**: assertEquals(employee1, foundEmployee)
- **testRemoveEmployee()**: This method tests the removeEmployee method. It verifies that an employee can be removed by their ID.
    - **Input**: hrEmployee.removeEmployee(employee1.getId())
    - **Expected Output**: The employee with ID employee1.getId() should be null after removal.
    - **Assertion**: assertNull(hrEmployee.findEmployeeById(employee1.getId()))
- **testEvaluateEmployeePerformance()**: This method tests the evaluateEmployeePerformance method. It verifies that an employee's performance can be evaluated.

- **Input**: hrEmployee.evaluateEmployeePerformance(101)
- **Expected Output**: Evaluation.Unsatisfactory for employee1
- **Assertion**: assertEquals(Evaluation.Unsatisfactory, aliceEvaluation)

- **testLeaveRequests()**: This method tests the leave request management functionalities within the HRemployee class. It verifies that leave requests can be created, approved, and rejected correctly.
  - **Input**: Adding leave requests for employee1 and employee2, approving and rejecting leave requests by their IDs.
  - **Expected Output**: LeaveRequest status for employee1 should be Accepted, and for employee2 should be Rejected.
  - **Assertion**: assertEquals(LeaveStatus.Accepted, leaveRequest1.getLeaveStatus()), assertEquals(LeaveStatus.Rejected, leaveRequest2.getLeaveStatus())

Table

TestCaseno	**Type**	**Tes Case Name**	**Description**	**Input**	**Output**	**Expected**	**Accepted**
1	component	**testCreateEmployee**	This method tests the createEmployee method of the HRemployee class. It verifies that	hrEmployee.createEmployee(...) for employee1 and employee2	employee1 and employee2 is null.	employee1 and employee2 should not be null.	yes

			employees are created correctly.				
2	component	**testFindEmployeeById**	This method tests the findEmployeeById method. It verifies that an employee can be found by their ID	hrEmployee.findEmployeeById(101)	employee1	employee1	yes
3	component	**testRemoveEmployee**	This method tests the removeEmployee method. It verifies that an employee can be removed by their ID.	hrEmployee.removeEmployee(employee1.getId())	The employee with ID employee1.getId() is null after removal.	The employee with ID employee1.getId() should be null after removal.	yes
4	component	**testEvaluateEmployeePerformance**	This method tests the evaluateEmployeePerformance method. It verifies that an employee's performance can be evaluated.	hrEmployee.evaluateEmployeePerformance(101)	Evaluation.Unsatisfactory for employee1	Evaluation.Unsatisfactory for employee1	yes
5	compon	**testLeaveRequests**	This method tests the leave	: Adding leave requests for employee1 and	LeaveRequest status for	LeaveRequest status for	yes

	ent		request management functionalities within the HRemployee class. It verifies that leave requests can be created, approved, and rejected correctly	employee2, approving and rejecting leave requests by their IDs.	employee 1 is Accepted, and for employee 2 is Rejected.	employee 1 should be Accepted, and for employee 2 should be Rejected.	

Class: PerformanceEvaluationTest

Explanation

The PerformanceEvaluationTest class contains unit tests for the PerformanceEvaluation class, which is responsible for evaluating employee performance based on various performance criteria. Each test method within this class ensures that the evaluatePerformance method works correctly for different scenarios.

Initialization and Setup

- **Employee Objects**: Various Employee instances are created for each test, with different performance elements added to simulate different performance levels.

Test Methods

- **testEvaluatePerformance_NoPerformanceElements()**: This method tests the evaluatePerformance method when an employee has no performance elements.
  - **Input**: An Employee object without any performance elements.
  - **Expected Output**: Evaluation.Unsatisfactory
  - **Assertion**: assertEquals(Evaluation.Unsatisfactory, evaluation)
- **testEvaluatePerformance_NeedsImprovement()**: This method tests the evaluatePerformance method when an employee has minimal performance

elements, indicating the need for improvement.

- **Input**: An Employee object with Performance.Productivity and Performance.Quality elements.
- **Expected Output**: Evaluation.NeedsImprovement
- **Assertion**: assertEquals(Evaluation.NeedsImprovement, evaluation)

- **testEvaluatePerformance_MeetsExpectations()**: This method tests the evaluatePerformance method when an employee meets the expected performance criteria.
  - **Input**: An Employee object with Performance.Quality, Performance.Punctuality, and Performance.Skills elements.
  - **Expected Output**: Evaluation.MeetsExpectations
  - **Assertion**: assertEquals(Evaluation.MeetsExpectations, evaluation)
- **testEvaluatePerformance_Excellent()**: This method tests the evaluatePerformance method when an employee performs excellently.
  - **Input**: An Employee object with Performance.Quality, Performance.Punctuality, Performance.Skills, and Performance.Attendance elements.
  - **Expected Output**: Evaluation.Excellent
  - **Assertion**: assertEquals(Evaluation.Excellent, evaluation)
- **testEvaluatePerformance_OverAchieving()**: This method tests the evaluatePerformance method when an employee exceeds expectations and performs over-achievingly.
  - **Input**: An Employee object with Performance.Quality, Performance.Punctuality, Performance.Skills, Performance.Attendance, and Performance.Productivity elements.
  - **Expected Output**: Evaluation.OverAchieving
  - **Assertion**: assertEquals(Evaluation.OverAchieving, evaluation)

Table

TestCaseno	**Type**	**Tes Case Name**	**Description**	**Input**	**Output**	**Expected**	**Accepted**
1	component	**testEvaluatePerformance_NoPerformanceElements**	This method tests the evaluatePerformance method when an employee has no performance elements.	An Employee object without any performance elements.	Evaluation.Unsatisfactory	Evaluation.Unsatisfactory	yes
2	component	**testEvaluatePerformance_NeedsImprovement**	This method tests the evaluatePerformance method when an employee has minimal performance elements, indicating the need	: An Employee object with Performance.Productivity and Performance.Quality elements	Evaluation.NeedsImprovement	Evaluation.NeedsImprovement	yes

			for improvement.				
3	component	**testEvaluatePerformance_MeetsExpectations**	This method tests the evaluatePerformance method when an employee meets the expected performance criteria.	An Employee object with Performance.Quality, Performance.Punctuality, and Performance.Skills elements.	Evaluation.MeetsExpectations	Evaluation.MeetsExpectations	yes
4	component	**testEvaluatePerformance_Excellent**	This method tests the evaluatePerformance method when an employee performs excellently	: An Employee object with Performance.Quality, Performance.Punctuality, Performance.Skills, and Performance.Attendance elements.	Evaluation.Excellent	Evaluation.Excellent	yes
5	component	**testEvaluatePerformance_OverAchieving**	This method tests the evaluatePerformance method when an employee exceeds expectations and performs	An Employee object with Performance.Quality, Performance.Punctuality, Performance.Skills, Performance.Attendanc	Evaluation.OverAchieving	Evaluation.OverAchieving	yes

			over-achievingly.	e, and Performance.Productivity elements.			

Class: EmployeeTest

Explanation

The EmployeeTest class is a comprehensive test suite for the Employee class, ensuring all functionalities of the Employee class are working as intended. This test suite verifies the initialization, getter, and setter methods for various attributes of the Employee class.

Initialization and Setup

- **Address Object**: A Address instance is created for the Employee object.
- **Employee Object**: An Employee instance is created in the setUp method to be used across various tests.

Test Methods

- **testEmployeeInitialization()**: This method tests the initialization of the Employee object.
  - **Input**: Initialized Employee object.
  - **Expected Output**:
    - Name: "Aley"
    - ID: 101
    - Username: "Aley"
    - Password: "password"
    - Address: address
    - Department: "Computer Engineering"
    - Employee Type: EmployeeType.FullTime

        - Evaluation: Evaluation.Excellent
        - Performance List: Empty
    - **Assertion**: Assertions for each attribute to match the initialized values.
- **testSetGetName()**: This method tests the setName and getName methods.
    - **Input**: Setting name to "Ziad".
    - **Expected Output**: "Ziad"
    - **Assertion**: assertEquals("Ziad", employee.getName())
- **testSetGetId()**: This method tests the setId and getId methods.
    - **Input**: Setting ID to 102.
    - **Expected Output**: 102
    - **Assertion**: assertEquals(102, employee.getId())
- **testSetGetUsername()**: This method tests the setUsername and getUsername methods.
    - **Input**: Setting username to "Zeze".
    - **Expected Output**: "Zeze"
    - **Assertion**: assertEquals("Zeze", employee.getUsername())
- **testSetGetPassword()**: This method tests the setPassword and getPassword methods.
    - **Input**: Setting password to "newpassword123".
    - **Expected Output**: "newpassword123"
    - **Assertion**: assertEquals("newpassword123", employee.getPassword())
- **testSetGetAddress()**: This method tests the setAddress and getAddress methods.
    - **Input**: Setting address to a new Address object.
    - **Expected Output**: New address object.

        - **Assertion**: assertEquals(newAddress, employee.getAddress())
    - **testSetGetDepartment()**: This method tests the setDepartment and getDepartment methods.
        - **Input**: Setting department to "Marketing".
        - **Expected Output**: "Marketing"
        - **Assertion**: assertEquals("Marketing", employee.getDepartment())
    - **testSetGetEmployeeType()**: This method tests the setEmployeeType and getEmployeeType methods.
        - **Input**: Setting employee type to EmployeeType.PartTime.
        - **Expected Output**: EmployeeType.PartTime
        - **Assertion**: assertEquals(EmployeeType.PartTime, employee.getEmployeeType())
    - **testSetGetPayroll()**: This method tests the setPay and getPay methods.
        - **Input**: Setting payroll to a new Payroll object.
        - **Expected Output**: New payroll object.
        - **Assertion**: assertEquals(payroll, employee.getPay())
    - **testAddGetPerformanceList()**: This method tests the addPerformance and getPerformanceList methods.
        - **Input**: Adding a performance element.
        - **Expected Output**: List containing the added performance element.
        - **Assertion**:
            - assertEquals(1, performanceList.size())
            - assertEquals(Performance.Attendance, performanceList.get(0))
    - **testSetGetEvaluation()**: This method tests the setEvaluation and getEvaluation methods.
        - **Input**: Setting evaluation to Evaluation.Unsatisfactory.

- **Expected Output**: Evaluation.Unsatisfactory
- **Assertion**: assertEquals(Evaluation.Unsatisfactory, employee.getEvaluation())

Table

TestCaseno	**Type**	**Tes Case Name**	**Description**	**Input**	**Output**	**Expected**	**Accepted**
1	component	**testEmployeeInitialization**	This method tests the initialization of the Employee object.	Initialized Employee object	Name:Aley,ID:101,Username:"aley,§ Password: "password",§ Address: address,§ Department: "Computer Engineering,§ Employee Type: EmployeeType. FullTime,§ Evaluation: Evaluation.Excellent,§ Performance List: Empty	Name:Aley,ID:101,Username:"aley,§ Password: "password",§ Address: address,§ Department: "Computer Engineering,§ Employee Type: EmployeeType. FullTime,§ Evaluation: Evaluation.Excellent,§ Performance List: Empty	yes

2	component	**testSetGetName**	This method tests the setName and getName methods.	Setting name to "Ziad".	"Ziad"	"Ziad"	yes
3	component	**testSetGetId**	This method tests the setUsername and getUsername methods.	Setting username to "Zeze".	"Zeze"	"Zeze"	yes
4	component	**testSetGetPassword**	This method tests the setPassword and getPassword methods.	Setting password to "newpassword123".	"newpassword123"	"newpassword123"	yes
5	component	**testSetGetAddress**	This method tests the setAddress and getAddress methods.	Setting address to a new Address object	New address object.	New address object.	yes
6	compon	**testSetGetD**	This method	Setting department	"Marketing"	"Marketing"	yes

	ent	**epartment**	tests the setDepartment and getDepartment methods.	to "Marketing".			
7	component	**testSetGetEmployeeType**	This method tests the setEmployeeType and getEmployeeType methods.	Setting employee type to EmployeeType.PartTime.	EmployeeType.PartTime	EmployeeType.PartTime	yes
8	component	**testSetGetPayroll**	This method tests the setPay and getPay methods.	Setting payroll to a new Payroll object.	New payroll object.	New payroll object.	yes
9	component	**testAddGetPerformanceList**	This method tests the addPerformance and getPerformanceList methods.	Adding a performance element.	List containing the added performance element	List containing the added performance element	yes
1	compon	**testSetGetE**	This method	Setting evaluation	Evaluation.Uns	Evaluation.Uns	yes

0	ent	**valuation**	tests the setEvaluation and getEvaluation methods.	to Evaluation. Unsatisfactory.	atisfactory	atisfactory	

Test Suit Class

Test Suit Explanation

The TestSuit class serves as a container for organizing and executing multiple test cases within a Java project. It is designed to streamline the testing process by grouping related tests together for efficient execution and reporting.

Annotations

1. **@RunWith(Suite.class):** This annotation indicates that JUnit should use the Suite runner to execute this class. The Suite runner allows you to run multiple test classes together as a suite. 2. **@Suite.SuiteClasses({ ... }):** This annotation specifies the test classes that should be included in the test suite. In this case, it includes the following test classes:

```java
package Junit4;

import ...

@RunWith(Suite.class)  Aley Amin
@Suite.SuiteClasses({
        AddressTest.class,
        EmployeeTest.class,
        HRemployeeTest.class,
        LeaveManagementTest.class,
        LeaveRequestTest.class,
        PayrollTest.class,
        PerformanceEvaluationTest.class
})
public class TestSuite {
}
```

Purpose

The purpose of the TestSuitCase class is to organize and execute a comprehensive set of tests for various components of the application. By grouping related test classes together in a suite, it ensures that all relevant tests are run together, providing a holistic assessment of the system's functionality.

Benefits

1. Modularity: Each test class focuses on testing specific functionalities or components, promoting modular and maintainable test suites.
2. Efficiency: Running multiple test classes together saves time and resources compared to running individual tests separately.
3. Consistency: Test suites ensure that all relevant tests are executed consistently, reducing the risk of overlooking critical test cases.
4. Reporting: The aggregated test results from the test suite provide comprehensive insights into the system's overall test coverage and quality.

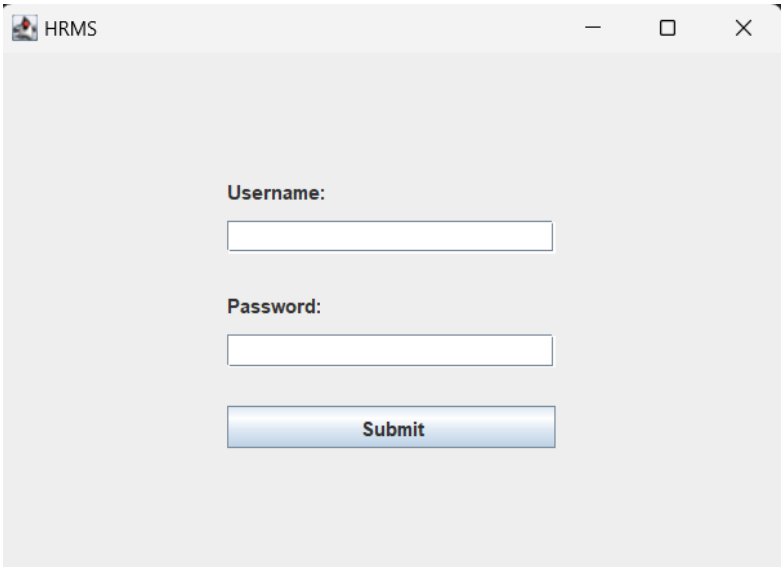
Conclusion

In summary, the TestSuitCase class plays a crucial role in the testing process by organizing and executing a collection of test cases efficiently. It promotes modularity, efficiency, and consistency in testing practices, ultimately contributing to the overall reliability and quality of the

software product.

GUI Testing

Gui Pages



Login

There are 4 Possibilities for login
1-login as an admin which requires inputting username:admin and Password:admin which will lead you to HR employee page

2-login as Employee which requires
Username:Bolty
Password:123
Or
Username:Zeze
Password:Password

3-Enering invalid option will leed to error message

4-Null input

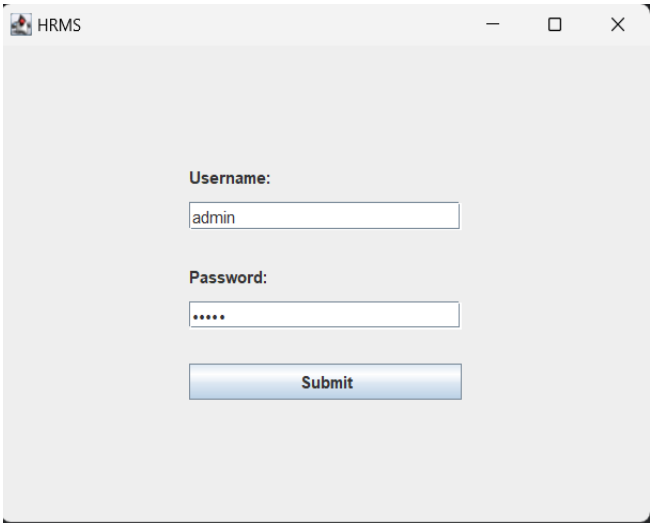


Figure 4:Employee login

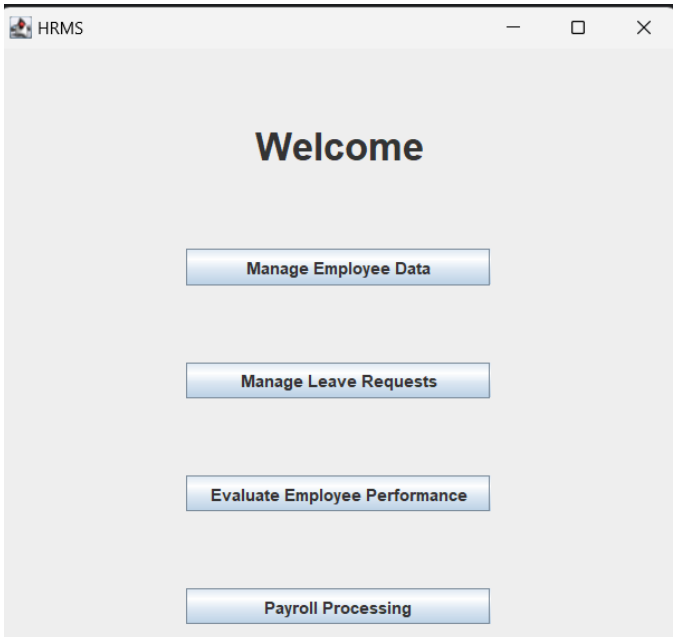
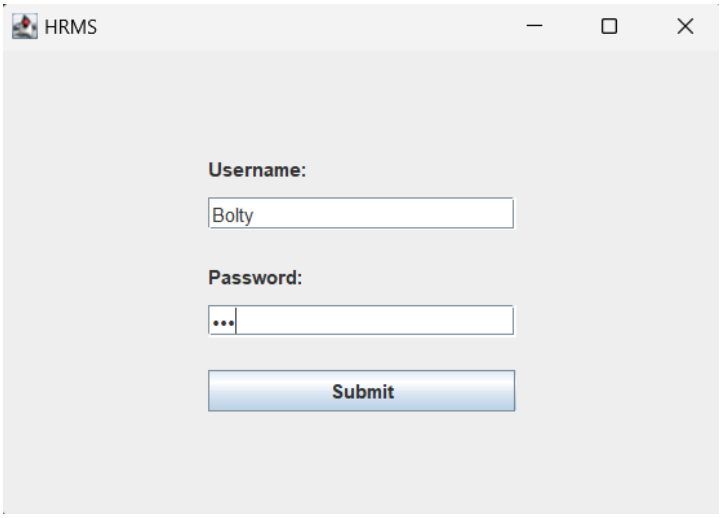
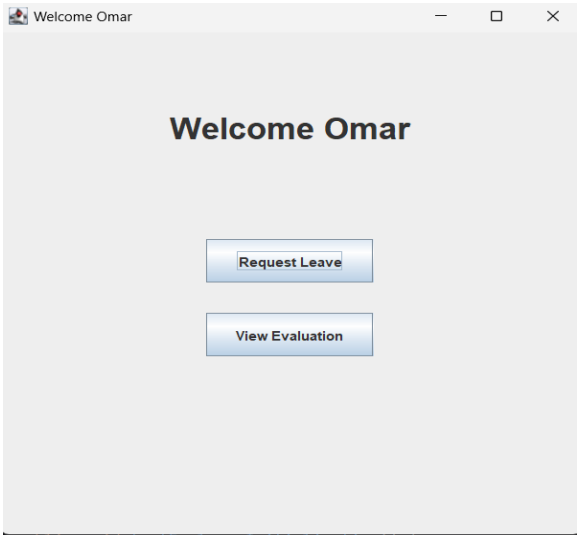
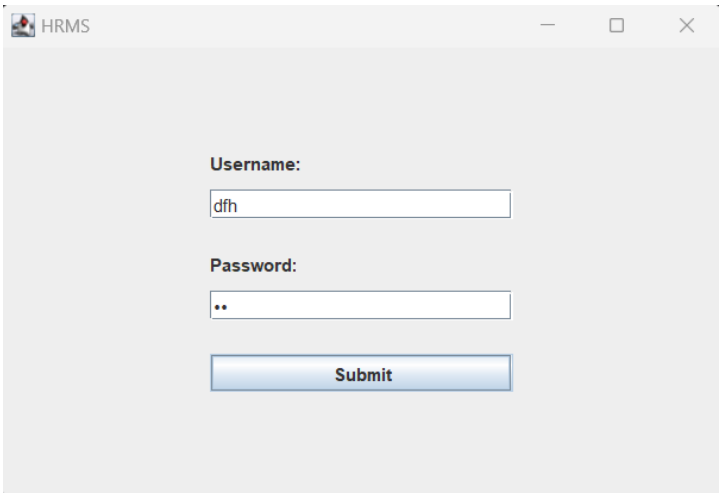
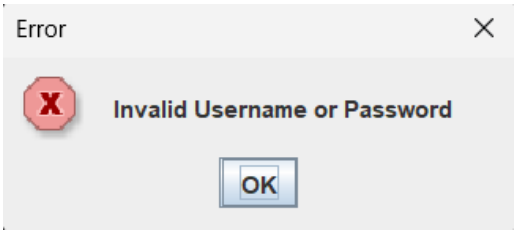


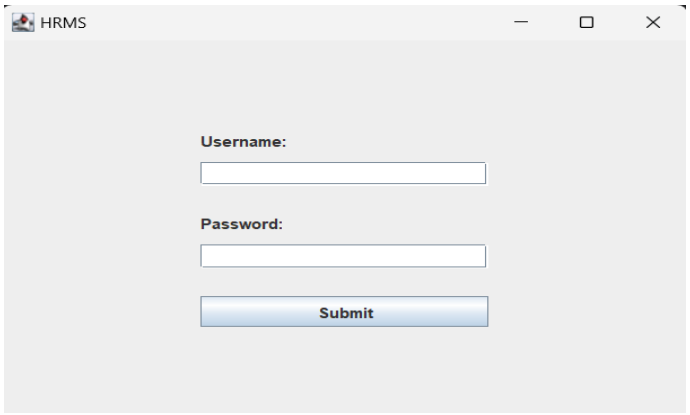
Figure 5:Employee Page

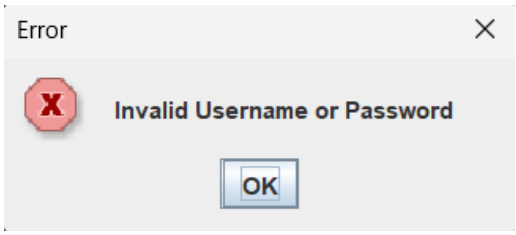

HRMS
Username:
Bolty
Password:
Submit


Welcome Omar
Welcome Omar
Request Leave
View Evaluation


HRMS
Username:
dfh
Password:
Submit


Error
Invalid Username or Password
OK

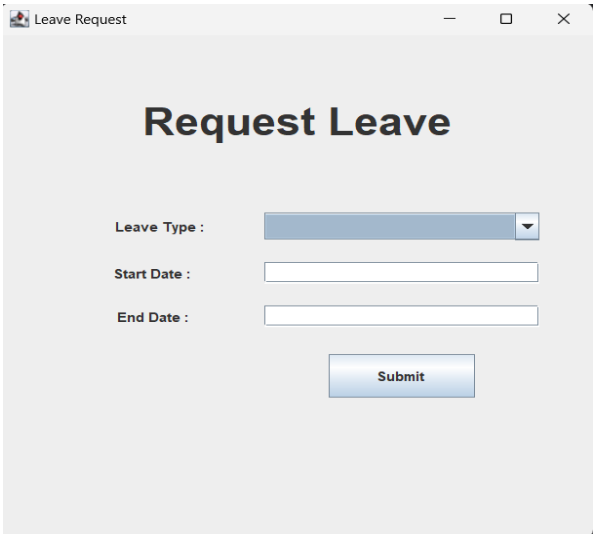

HRMS
Username:
Password:
Submit

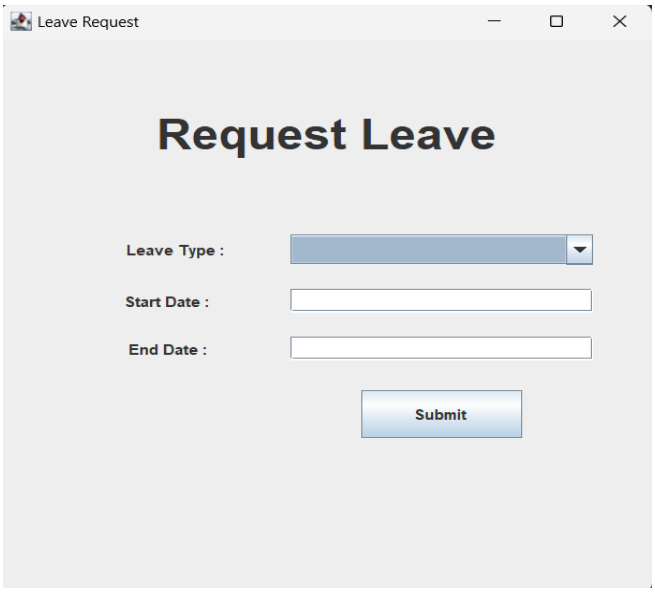

Error
Invalid Username or Password
OK

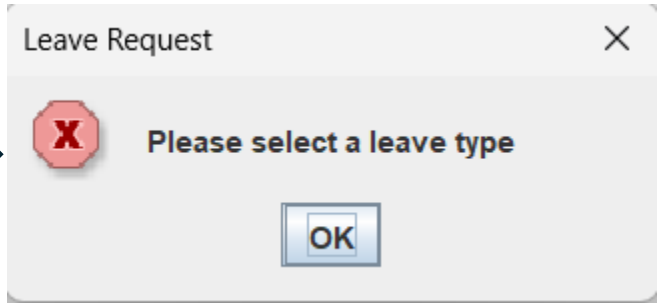
Leave Request Page

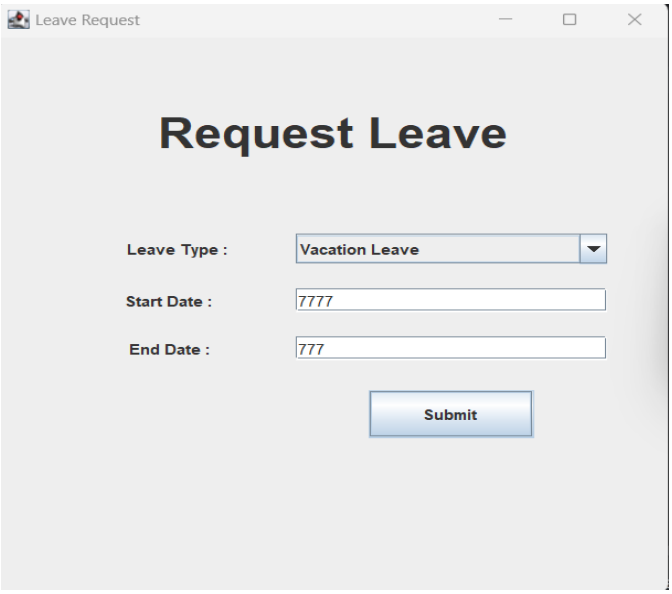
There are 4 Possibilities for Leave Request

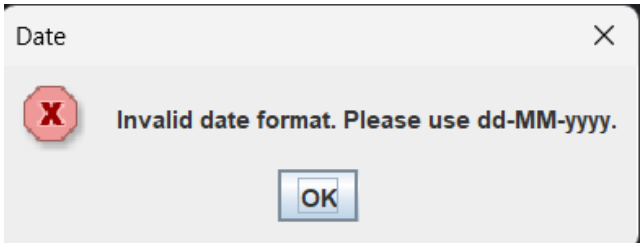
1- No data is entered which will leed to alert messages
2- Inserting start date to be after end date will leed to alert messages
3- Inserting wrong date formate will leed to alert messages
4- Inserting the correct data with correct Formate which will show a message with request data

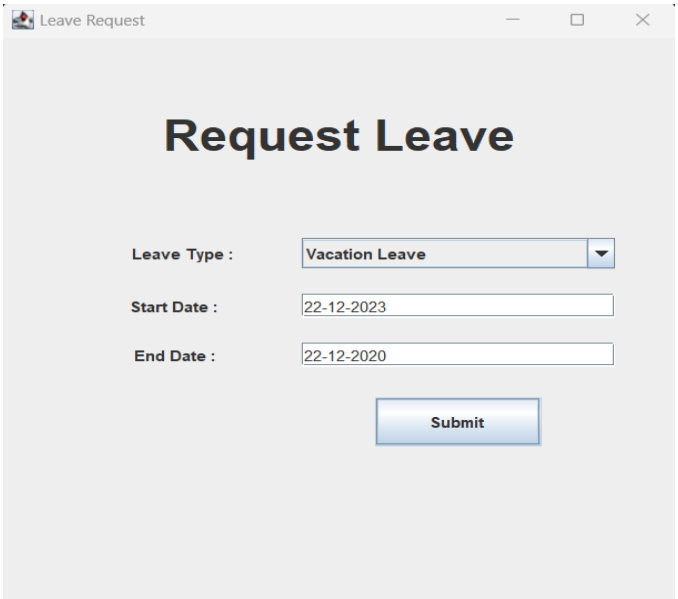


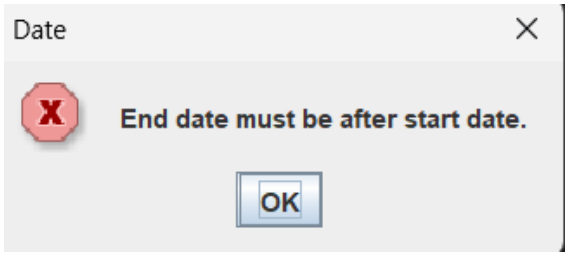


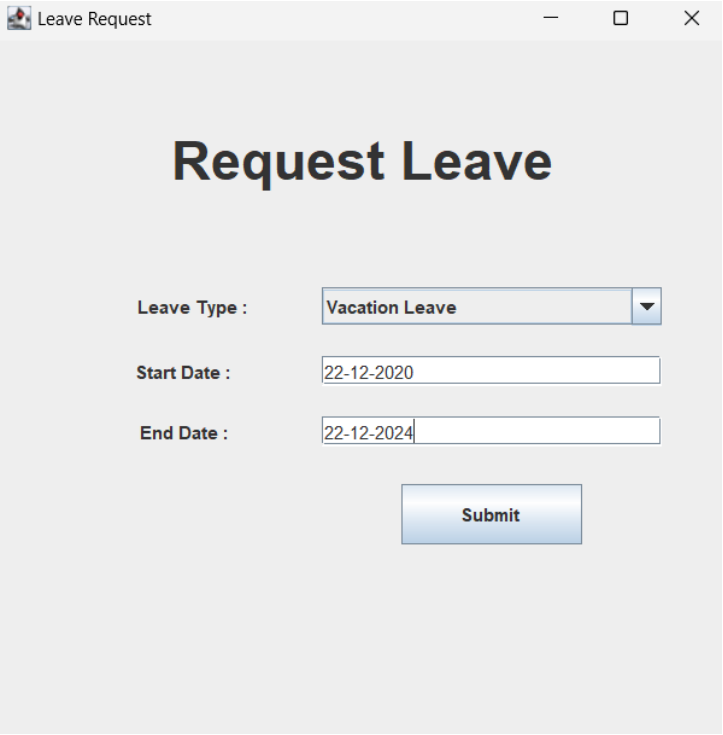


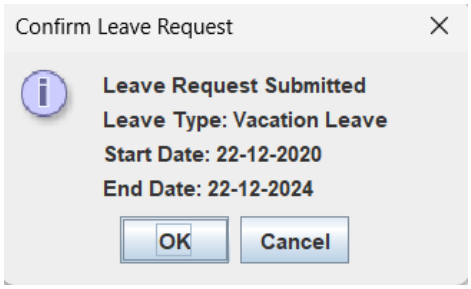

Leave Request
Request Leave
Leave Type :
Vacation Leave
Start Date :
7777
End Date :
777
Submit


Date
Invalid date format. Please use dd-MM-yyyy.
OK

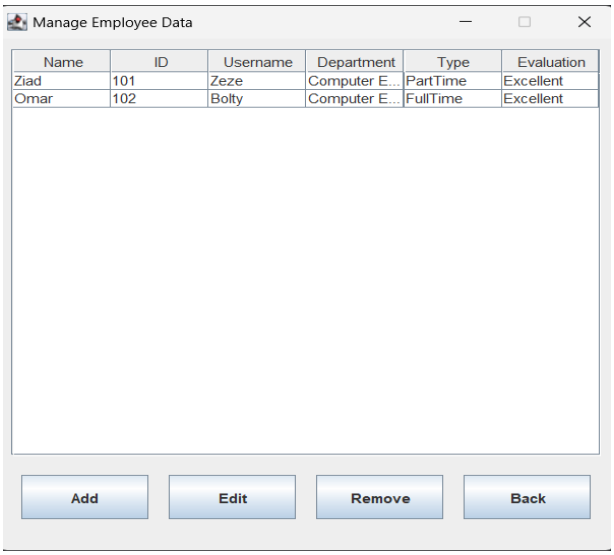

Leave Request
Request Leave
Leave Type :
Vacation Leave
Start Date :
22-12-2023
End Date :
22-12-2020
Submit


Date
End date must be after start date.
OK


Leave Request
Request Leave
Leave Type :
Vacation Leave
Start Date :
22-12-2020
End Date :
22-12-2024
Submit


Confirm Leave Request
Leave Request Submitted
Leave Type: Vacation Leave
Start Date: 22-12-2020
End Date: 22-12-2024
OK
Cancel

Manage Employee Data

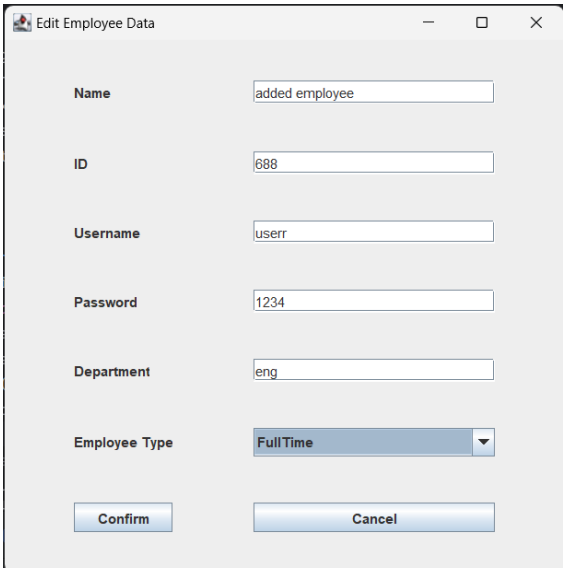


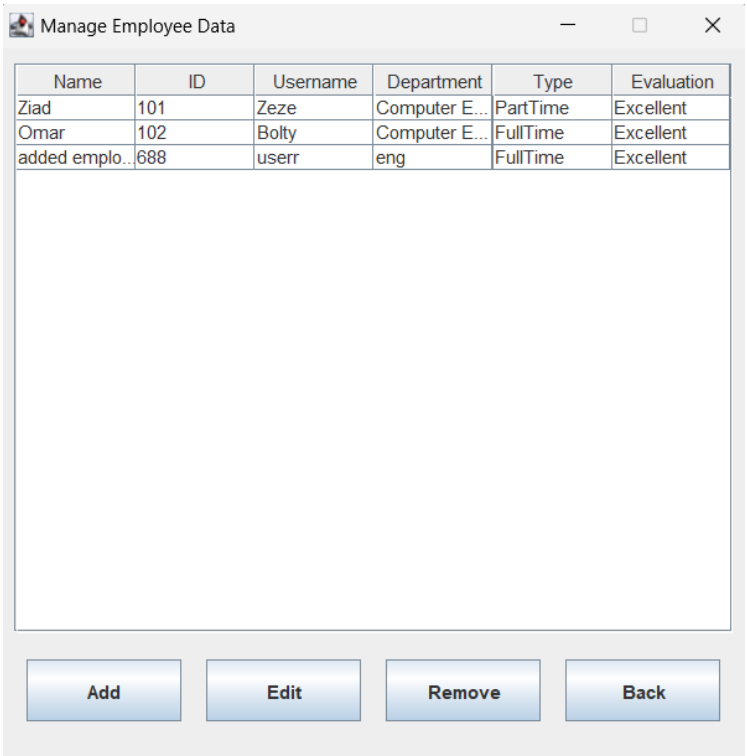
Manage employee data have 3 options which are Add, Edit and remove each option has its possibilities

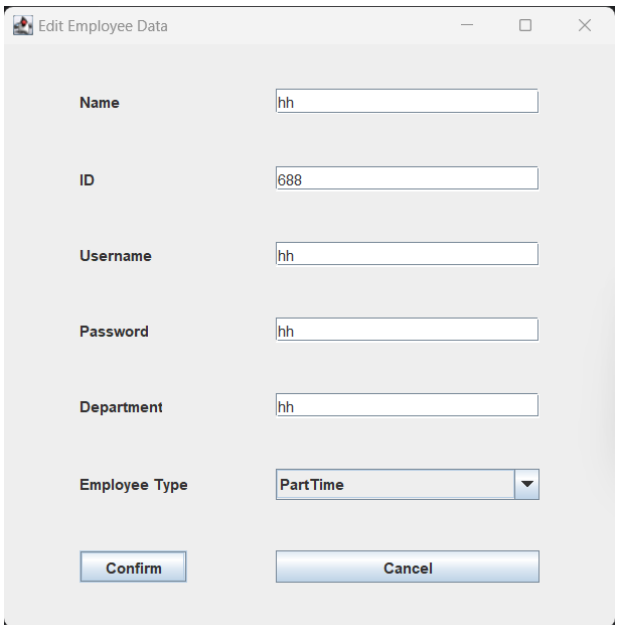
Add possibilities
1-inserting with an existing id will lead to error message.
2-inserting with an existing username will lead to error message.
3-doesn't insert any data will lead to error message.
Insertin correct data will add it to the table and shows an acceptance message
4-inserting the correct data

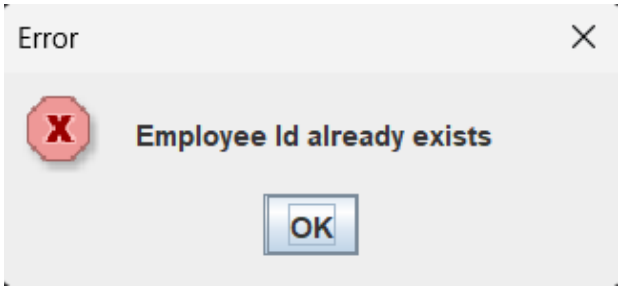
Edit possibilities
1-doesn't select the row
2-writing user name or id already exist
3-correct edit

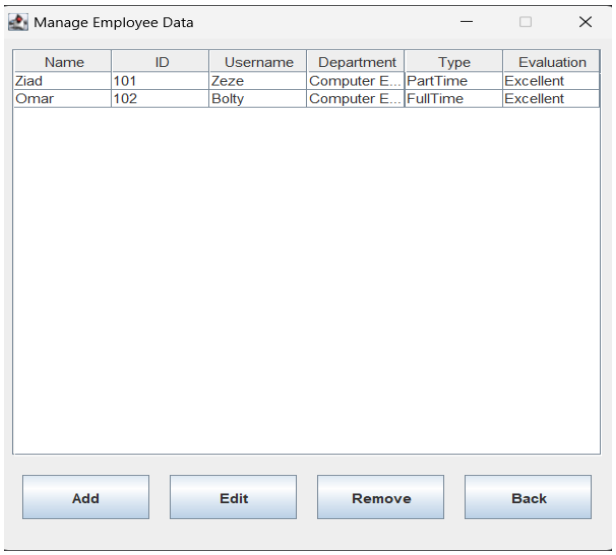
Remove will remove data if row is selected

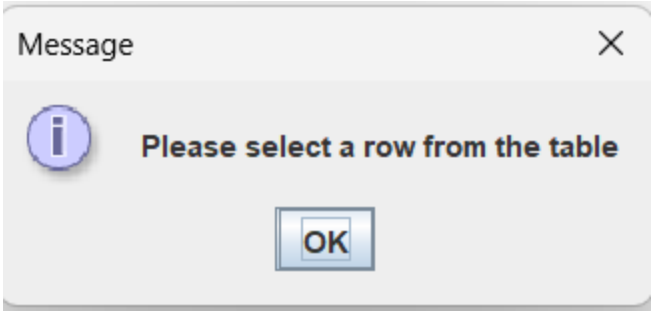

Edit Employee Data
Name
added employee
ID
688
Username
userr
Password
1234
Department
eng
Employee Type
FullTime
Confirm
Cancel

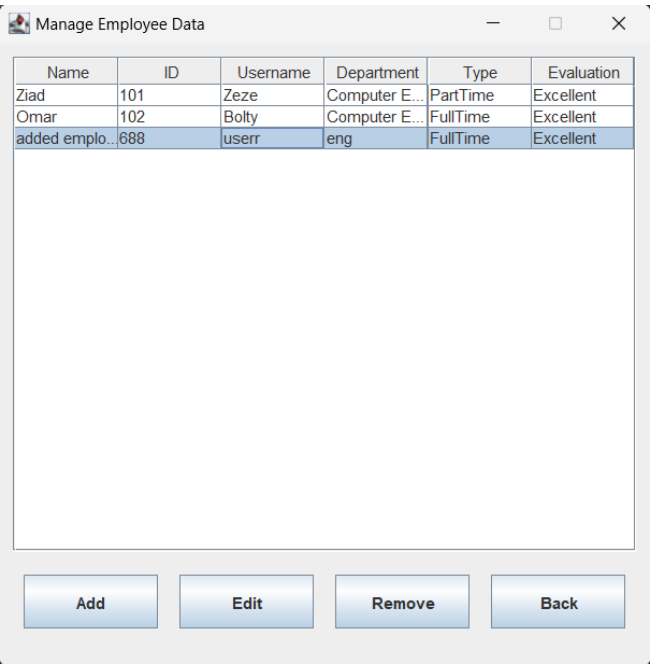

Manage Employee Data
Name
ID
Username
Department
Type
Evaluation
Ziad
101
Zeze
Computer E...
PartTime
Excellent
Omar
102
Bolty
Computer E...
FullTime
Excellent
added emplo...
688
userr
eng
FullTime
Excellent
Add
Edit
Remove
Back

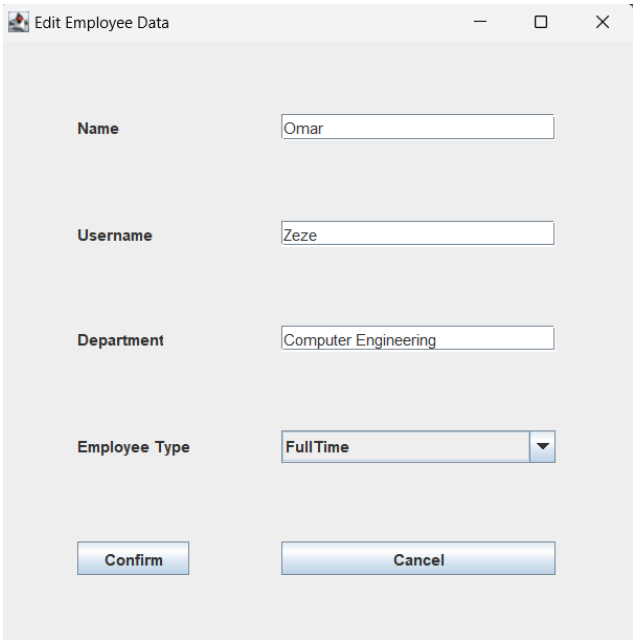

Edit Employee Data
Name
hh
ID
688
Username
hh
Password
hh
Department
hh
Employee Type
PartTime
Confirm
Cancel

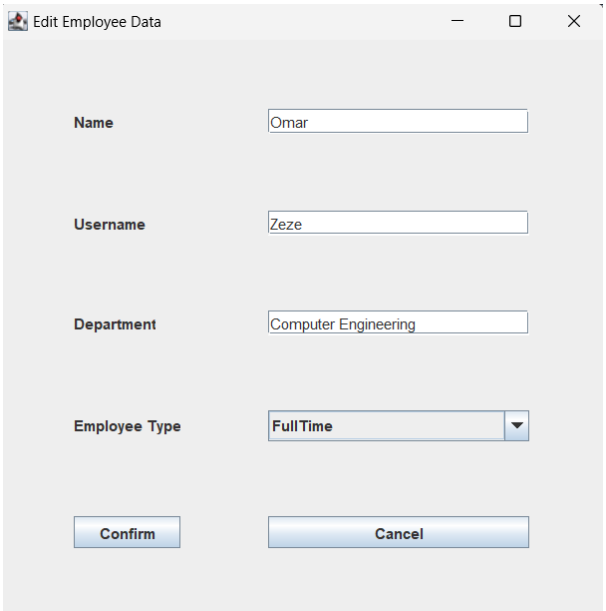

Error
Employee Id already exists
OK

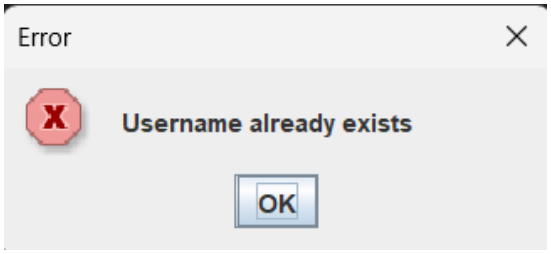

Manage Employee Data
Name
ID
Username
Department
Type
Evaluation
Ziad
101
Zeze
Computer E...
PartTime
Excellent
Omar
102
Bolty
Computer E...
FullTime
Excellent
Add
Edit
Remove
Back


Message
Please select a row from the table
OK


Manage Employee Data
Name
ID
Username
Department
Type
Evaluation
Ziad
101
Zeze
Computer E...
PartTime
Excellent
Omar
102
Bolty
Computer E...
FullTime
Excellent
added emplo...
688
userr
eng
FullTime
Excellent
Add
Edit
Remove
Back


Edit Employee Data
Name
Omar
Username
Zeze
Department
Computer Engineering
Employee Type
FullTime
Confirm
Cancel


Edit Employee Data
Name
Omar
Username
Zeze
Department
Computer Engineering
Employee Type
Full Time
Confirm
Cancel

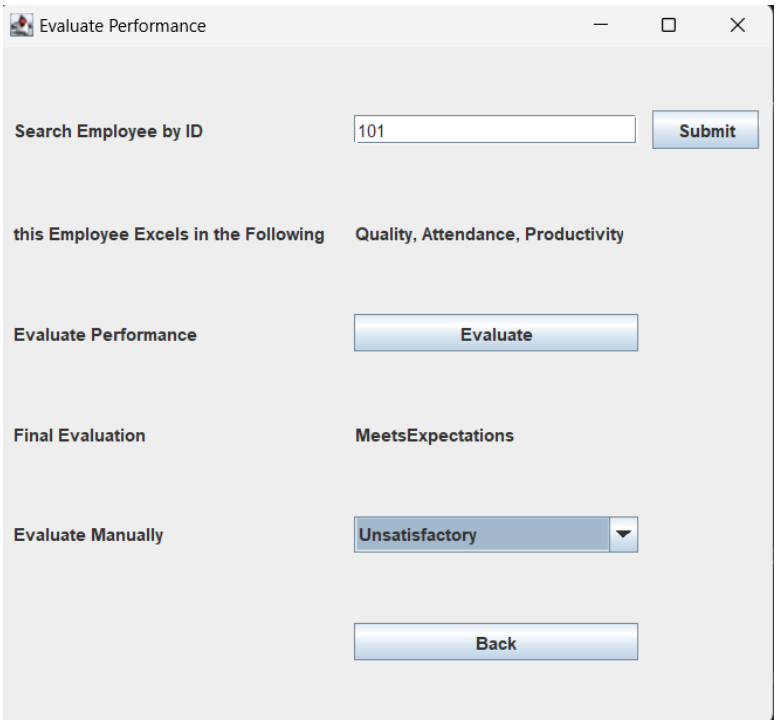

Error
Username already exists
OK

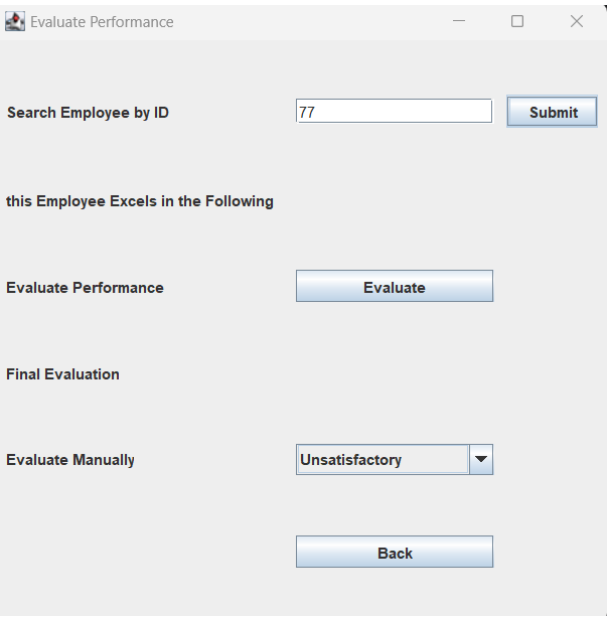
Evaluate performance

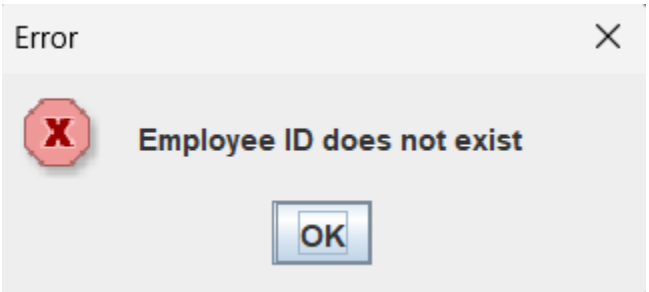
Final evaluation shows the calculated evaluation while you can change it by evaluate manually

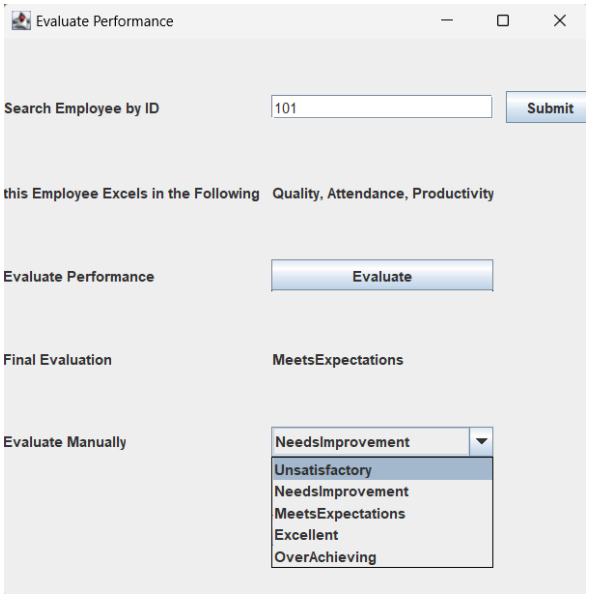
2 options in evaluate performance

1-if wrong id is inserted will show error message
2-if correct id is inserted then will show performance and can change it by evaluate manualy

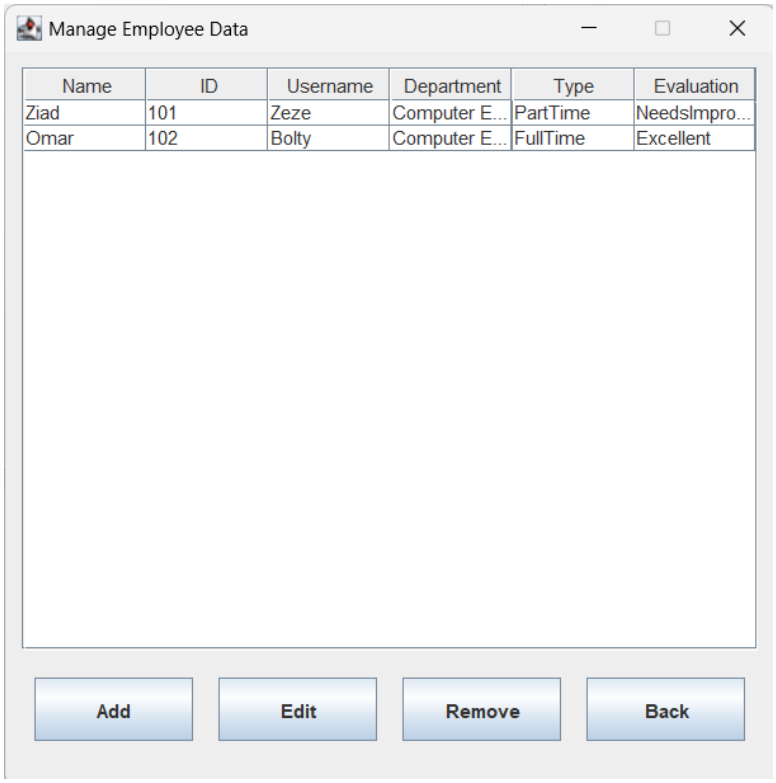



Evaluate Performance
Search Employee by ID
77
Submit
this Employee Excels in the Following
Evaluate Performance
Evaluate
Final Evaluation
Evaluate Manually
Unsatisfactory
Back


Error
Employee ID does not exist
OK

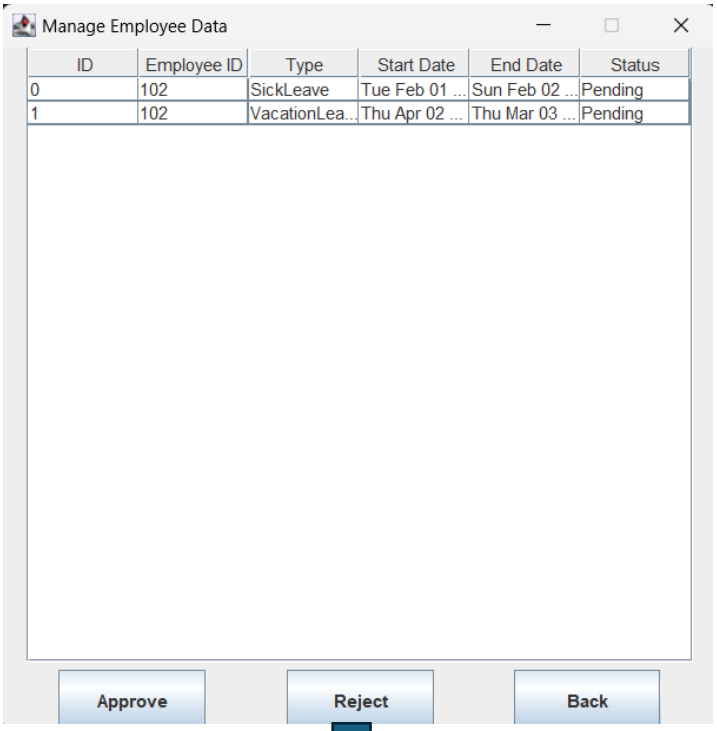


Omar changed from
Meets expectation to excellent

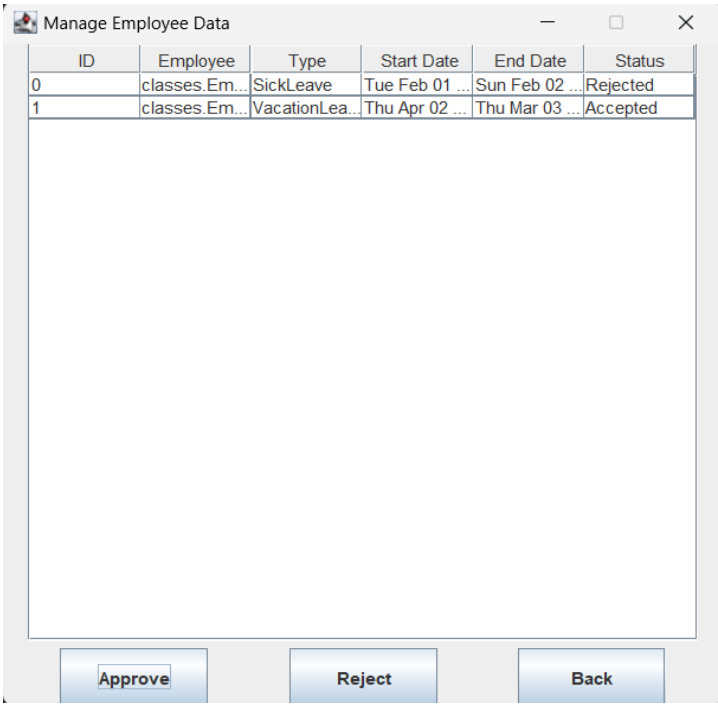


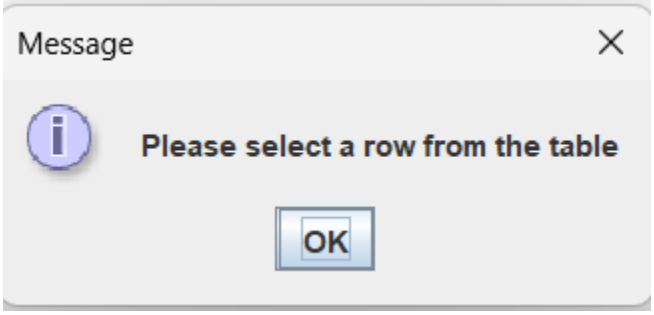
Manage Leave request

Manage leave request will have 3 possibilities
1-no row is selected will lead to alert message
2-approve that will change the status to be approved
3-reject that will change the status to be rejected



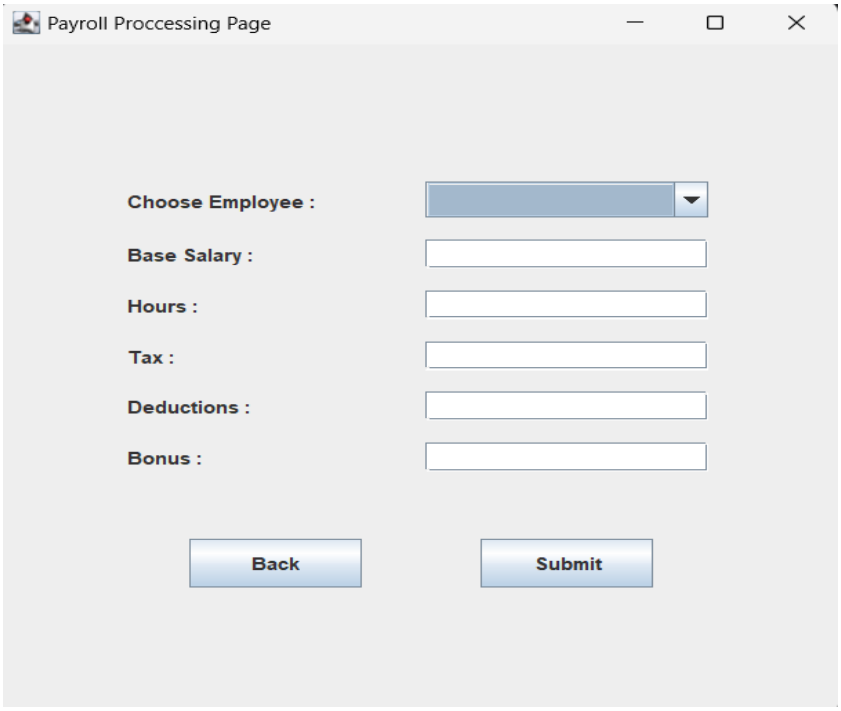
status is changed to be accepted or rejected

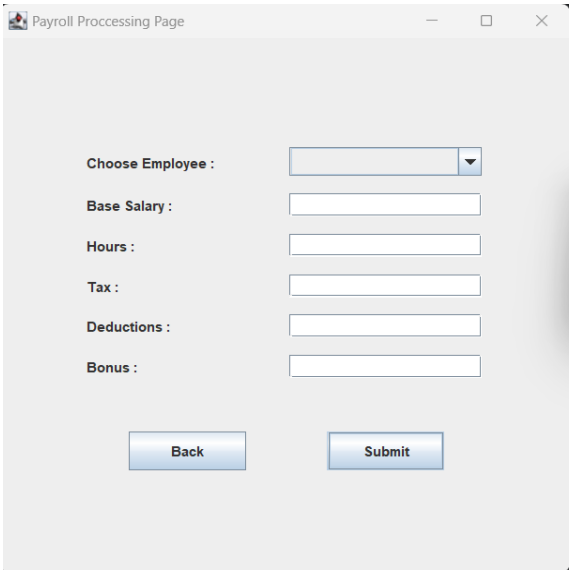


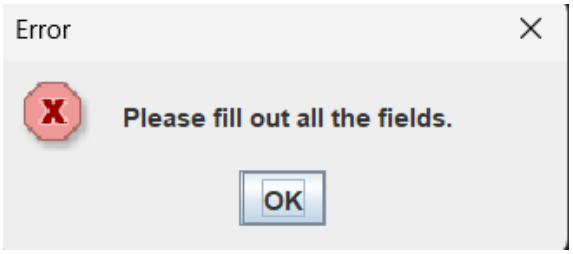


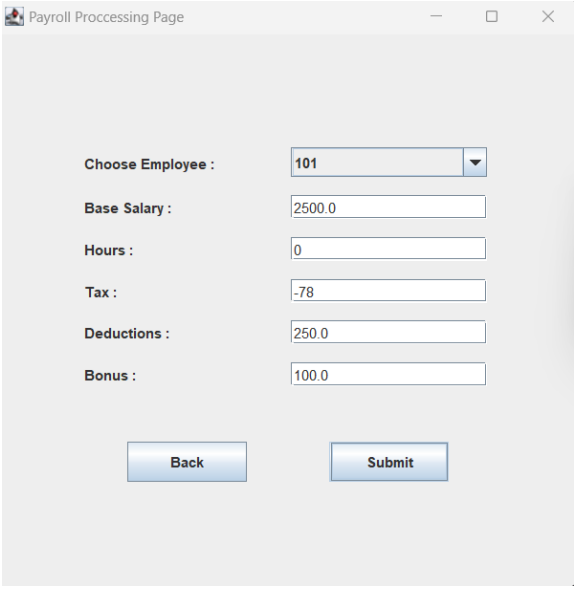
Payroll

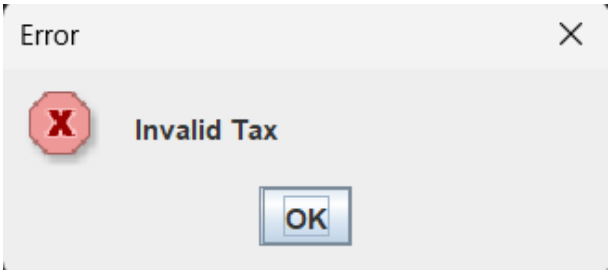
Payroll will have three possibilities
1-if empty data will lead to error message
2-if enter any negative value will lead to error
3-inserting the correct data



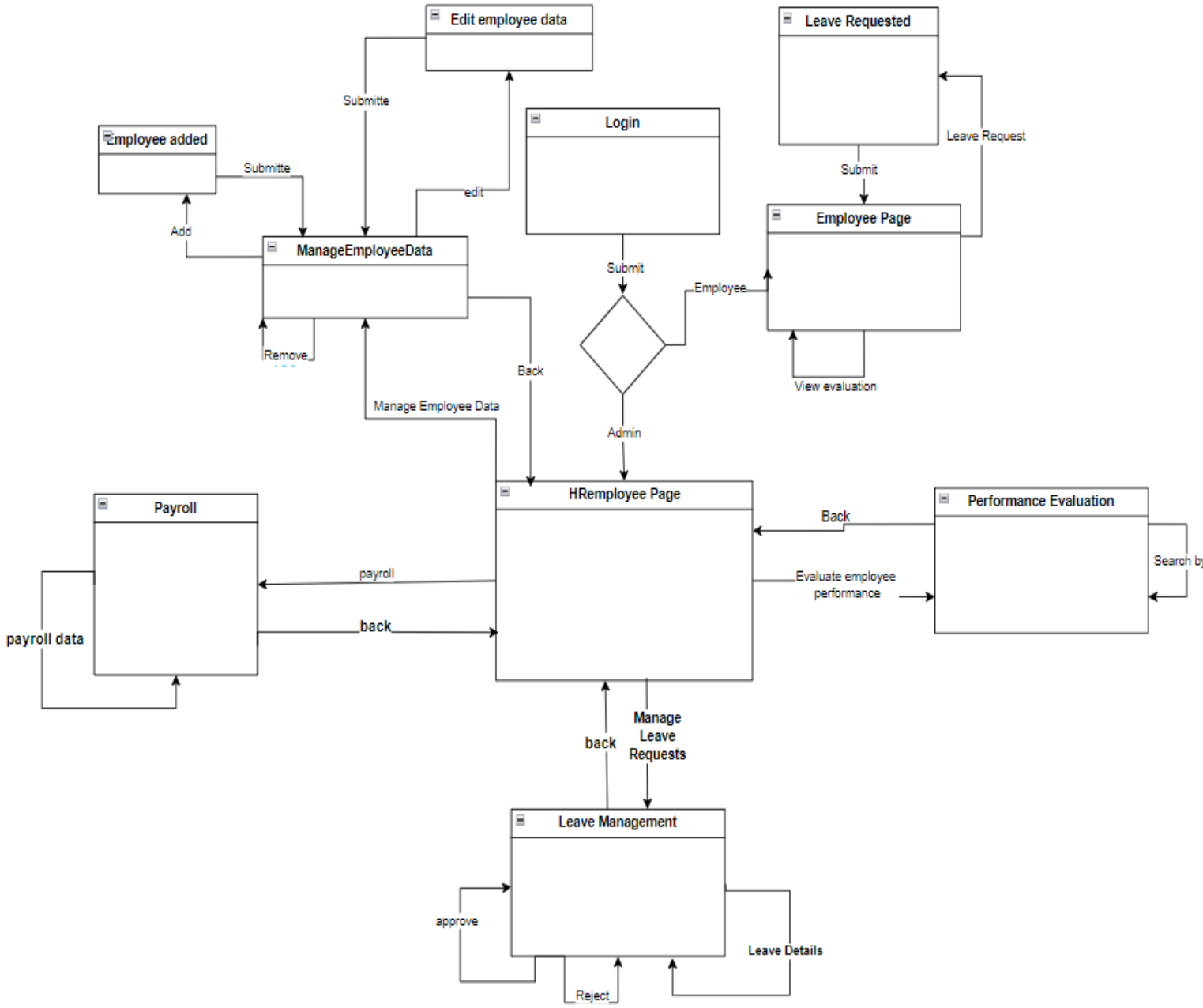

Payroll Proccessing Page
Choose Employee :
Base Salary :
Hours :
Tax :
Deductions :
Bonus :
Back
Submit


Error
Please fill out all the fields.
OK


Payroll Proccessing Page
Choose Employee :
101
Base Salary :
2500.0
Hours :
0
Tax :
-78
Deductions :
250.0
Bonus :
100.0
Back
Submit


Error
Invalid Tax
OK

FSM


Edit employee data
Leave Requested
Submitte
Login
Leave Request
Employee added
Submitte
Submit
edit
Employee Page
Add
ManageEmployeeData
Submit
Employee
Remove
Back
View evaluation
Manage Employee Data
Admin
HRemployee Page
Payroll
Back
Performance Evaluation
payroll
Evaluate employee performance
payroll data
back
back
Manage Leave Requests
Leave Management
approve
Leave Details
Reject

White box Testing

Introduction:

White Box Testing is an essential software testing methodology that involves examining the internal structure, design, and implementation of a software application. Unlike black box testing, which evaluates the software from an external perspective, white box testing focuses on the internal workings of the application. This approach requires a detailed understanding of the source code and aims to ensure that the software behaves as expected by testing all possible paths, conditions, and branches within the code.
For the **Human Resources Management System (HRMS)** project, white box testing is particularly valuable as it allows us to verify that core functionalities—such as employee management, leave requests, payroll processing, and performance evaluations—are implemented correctly and work seamlessly together. By analyzing and testing the internal logic, we can ensure that the system operates reliably and efficiently, meeting the needs of its users.

Unlike black box testing, which evaluates the system from an external perspective, white box testing dives into the code itself. This methodology allows us to verify that every line of code is executed and that all logical paths, conditions, and branches are thoroughly tested. By understanding and testing the internal logic, we can ensure that the HRMS performs as expected and meets the needs of its users.

Objectives:

1. **Ensure Complete Code Coverage:**
    - Statement Coverage: Confirm that every line of code in the HRMS is executed at least once. This is essential for identifying untested parts of the application, such as specific branches of logic in employee management or payroll calculations.
    - Branch Coverage: Ensure that all possible branches in the code—such as the decision points in leave approval processes or performance evaluation criteria—are tested. This includes both true and false paths to ensure comprehensive testing of conditional logic.

2. **Verify Internal Logic and Pathways:**
    - Test the logical flow of core HRMS functionalities, such as how leave requests are processed, how payroll is calculated based on different employee types, and how performance evaluations are determined. This ensures that the algorithms and business rules implemented in the system are functioning correctly.

3. **Identify and Fix Defects:**
    - Detect and resolve defects such as logical errors, incorrect calculations, or unhandled exceptions in areas like payroll processing or leave management. By analyzing the code, we can pinpoint issues that may not be evident through other testing methods**.**
4. **Improve Code Quality:**
    - Enhance the maintainability and efficiency of the HRMS code. This involves refactoring complex or redundant code segments in modules such as employee management or performance evaluation, ensuring that the system is easy to maintain and performant.
5. **Validate Integration Points:**
    - Test interactions between different modules of the HRMS, such as how employee data integrates with payroll processing or how leave requests interact **with** employee records. Ensure seamless data flow and correct functionality across the system.
6. **Enhance Security and Performance:**
    - Evaluate the HRMS code for potential security vulnerabilities and performance bottlenecks. Ensure that sensitive employee information is protected, and that the system performs efficiently under varying loads, such as during peak leave request periods.
7. **Ensure Robust Error Handling:**
    - Test how the HRMS handles errors and exceptions, such as invalid payroll inputs or unauthorized access attempts. Verify that appropriate error messages are generated, and that the system maintains stability and provides a good user experience**.**
8. **Assess Edge Cases and Boundary Conditions:**
    - Test edge cases and boundary conditions relevant to HRMS functionalities. For example, check how the system handles maximum and minimum values for employee salaries or the impact of edge cases in leave request dates.

By focusing on these objectives, white box testing will help ensure that the HRMS is robust, reliable, and ready to meet the diverse needs of its users.

1. ***Statement and Branch Coverage for "Employee" Class:***

The Employee class is used for managing employee details and performance. It includes attributes such as name, ID, username, and performance list. The main operations involve setting and getting these attributes and adding performance records.

- ***Statement and Branch Coverage:***

Since the Employee class primarily contains simple getter and setter methods, it does

not have complex logic. Therefore, testing will focus on ensuring that these methods work correctly and handle basic scenarios.

<u>Test Cases for Employee Class:</u>

1. **Test Case 1: Set and Get Name**
    - **Input: Set name = "John Doe"**
    - **Expected Output: getName() returns "John Doe"**
    - **Explanation: Tests setting and retrieving the employee's name.**
2. **Test Case 2: Set and Get ID**
    - **Input: Set id = 12345**
    - **Expected Output: getId() returns 12345**
    - **Explanation: Tests setting and retrieving the employee's ID.**
3. **Test Case 3: Set and Get Username**
    - **Input: Set username = "jdoe"**
    - **Expected Output: getUsername() returns "jdoe"**
    - **Explanation: Tests setting and retrieving the employee's username.**
4. **Test Case 4: Set and Get Password**
    - **Input: Set password = "securepassword"**
    - **Expected Output: getPassword() returns "securepassword"**
    - **Explanation: Tests setting and retrieving the employee's password.**
5. **Test Case 5: Set and Get Department**
    - **Input: Set department = "IT"**
    - **Expected Output: getDepartment() returns "IT"**
    - **Explanation: Tests setting and retrieving the employee's department.**
6. **Test Case 6: Add Performance Record**
    - **Input: Add performanceRecord = new performance()**
    - **Expected Output: getPerformanceList().size() returns 1**
    - **Explanation: Tests adding a performance record to the list.**
7. **Test Case 7: Set and Get Evaluation**

- **Input: Set evaluation = new Evaluation()**
- **Expected Output: getEvaluation() returns the Evaluation object**
- **Explanation: Tests setting and retrieving the evaluation.**

---

2. Statement and Branch Coverage for "HRemployee" Class:

The HRemployee class manages employees, leave requests, and performance evaluations. It provides methods for adding, removing, and finding employees, as well as handling leave requests and authenticating users.

- ***Statement and Branch Coverage:***

Method: addEmployee

1. **Test Case 1: Add New Employee**
    - **Input:** employee = new Employee("John Doe", 1, "jdoe", "password", new Address(), "IT", EmployeeType.FULL_TIME, Evaluation.Excellent)
    - **Expected Output:** managedEmployees.size() increases by 1
    - **Explanation:** Tests the addition of a new employee to the list.

Method: removeEmployee

2. **Test Case 2: Remove Existing Employee**
    - **Input:** employeeId = 1 (assuming an employee with ID 1 exists)
    - **Expected Output:** removeEmployee(employeeId) returns true, and managedEmployees.size() decreases by 1
    - **Explanation:** Tests removal of an employee with a valid ID.

3. **Test Case 3: Remove Non-existing Employee**
    - **Input:** employeeId = 999

- **Expected Output:** removeEmployee(employeeId) returns false
- **Explanation:** Tests the removal attempt for an employee that does not exist.

Method: findEmployeeById

4. **Test Case 4: Find Existing Employee**
    - **Input:** employeeId = 1 (assuming an employee with ID 1 exists)
    - **Expected Output:** findEmployeeById(employeeId) returns the employee object with ID 1
    - **Explanation:** Tests finding an employee with a valid ID.
5. **Test Case 5: Find Non-existing Employee**
    - **Input:** employeeId = 999
    - **Expected Output:** findEmployeeById(employeeId) returns null
    - **Explanation:** Tests finding an employee with an ID that does not exist.

Method: createEmployee

6. **Test Case 6: Create and Add New Employee**
    - **Input:** name = "Jane Smith", id = 2, username = "jsmith", password = "password", address = new Address(), department = "HR", employeeType = EmployeeType.PART_TIME
    - **Expected Output:** createEmployee() returns a new Employee object, and managedEmployees.size() increases by 1
    - **Explanation:** Tests the creation and addition of a new employee.

Method: approveLeaveRequest

7. **Test Case 7: Approve Leave Request**
    - **Input:** requestId = 1 (assuming a leave request with ID 1 exists)
    - **Expected Output:** leaveManagement.getAllLeaveRequests() shows the request with ID 1 as approved
    - **Explanation:** Tests approving a leave request with a valid ID.

Method: authenticate

8. ***Test Case 8: Admin Authentication***
    - **Input:** username = "admin", password = "admin"
    - **Expected Output:** authenticate() returns 2
    - **Explanation:** Tests authentication with admin credentials.
9. ***Test Case 9: Employee Authentication***

- **Input:** username = "jdoe", password = "password" (assuming an employee with these credentials exists)
- **Expected Output:** authenticate() returns the employee ID
- **Explanation:** Tests authentication for an existing employee.

10. ***Test Case 10: Invalid Authentication***
    - **Input:** username = "invalid", password = "invalid"
    - **Expected Output:** authenticate() returns -1
    - **Explanation:** Tests authentication with invalid credentials.

Method: evaluateEmployeePerformance

11. **Test Case 11: Evaluate Existing Employee**
    - **Input:** employeeId = 1 (assuming an employee with ID 1 exists)
    - **Expected Output:** evaluateEmployeePerformance(employeeId) returns an Evaluation object
    - **Explanation:** Tests performance evaluation for an existing employee.
12. ***Test Case 12: Evaluate Non-existing Employee***
    - **Input:** employeeId = 999
    - **Expected Output:** evaluateEmployeePerformance(employeeId) returns null
    - **Explanation:** Tests performance evaluation for a non-existing employee.

3. <u>Statement and Branch Coverage for "LeaveRequest" Class:</u>

The LeaveRequest class is responsible for managing leave requests. It contains attributes for the leave request ID, associated employee, leave type, start and end dates, and status. The class provides methods to set and get these attributes.

- ***<u>Statement and Branch Coverage:</u>***

Method: setId and getId

1. **Test Case 1: Set and Get Leave Request ID**
    - **Input:** id = 1001
    - **Expected Output:** getId() returns 1001
    - **Explanation:** Tests setting and retrieving the leave request ID.

Method: setEmployee and getEmployee

2. **Test Case 2: Set and Get Employee**

- **Input:** employee = new Employee("Jane Doe", 2, "jdoe", "password", new Address(), "HR", EmployeeType.FULL_TIME, Evaluation.Excellent)
- **Expected Output:** getEmployee() returns the Employee object
- **Explanation:** Tests setting and retrieving the employee associated with the leave request.

Method: setLeaveType and getLeaveType

3. **Test Case 3: Set and Get Leave Type**
    - **Input:** leaveType = LeaveType.SICK
    - **Expected Output:** getLeaveType() returns LeaveType.SICK
    - **Explanation:** Tests setting and retrieving the type of leave requested.

Method: setStartDate and getStartDate

4. **Test Case 4: Set and Get Start Date**
    - **Input:** startDate = new Date("2024/08/01")
    - **Expected Output:** getStartDate() returns new Date("2024/08/01")
    - **Explanation:** Tests setting and retrieving the start date of the leave request.

Method: setEndDate and getEndDate

5. **Test Case 5: Set and Get End Date**
    - **Input:** endDate = new Date("2024/08/10")
    - **Expected Output:** getEndDate() returns new Date("2024/08/10")
    - **Explanation:** Tests setting and retrieving the end date of the leave request.

Method: setLeaveStatus and getLeaveStatus

6. **Test Case 6: Set and Get Leave Status**
    - **Input:** leaveStatus = LeaveStatus.Approved
    - **Expected Output:** getLeaveStatus() returns LeaveStatus.Approved
    - **Explanation:** Tests setting and retrieving the status of the leave request.

Method: Constructor

7. **Test Case 7: Constructor Initialization**
    - **Input:** id = 1002, employee = new Employee("John Doe", 3, "jdoe", "password", new Address(), "IT", EmployeeType.PART_TIME, Evaluation.Good), leaveType = LeaveType.VACATION, startDate = new Date("2024/08/05"), endDate = new Date("2024/08/15")
    - **Expected Output:** getId() returns 1002, getEmployee() returns the Employee object, getLeaveType() returns LeaveType.VACATION, getStartDate() returns new Date("2024/08/05"), getEndDate() returns new Date("2024/08/15"), getLeaveStatus() returns LeaveStatus.Pending
    - **Explanation:** Tests the constructor to ensure all attributes are initialized correctly.

---

4. Statement and Branch Coverage for "PerformanceEvaluation" Class:

The PerformanceEvaluation class evaluates an employee's performance based on their performance records. The evaluatePerformance method assigns an evaluation based on the number of performance records, using predefined thresholds.

- ***Statement and Branch Coverage:***

Method: evaluatePerformance
Threshold Values:

- **MIN_PERFORMANCE_Excellent = 4**
- **MIN_PERFORMANCE_FOR_MEETS = 3**
- **MIN_PERFORMANCE_FOR_NEEDS_IMPROVEMENT = 2**

1. **Test Case 1: Performance List Exactly Meets Expectations**
    - **Input:** performanceList with 3 performance records
    - **Expected Output:** evaluatePerformance() returns Evaluation.MeetsExpectations
    - **Explanation:** Tests the case where the number of performance records matches the threshold for "Meets Expectations."
2. **Test Case 2: Performance List Needs Improvement**
    - **Input:** performanceList with 2 performance records

- **Expected Output:** evaluatePerformance() returns Evaluation.NeedsImprovement
- **Explanation:** Tests the case where the number of performance records matches the threshold for "Needs Improvement."

3. **Test Case 3: Performance List Excellent**
    - **Input:** performanceList with 4 performance records
    - **Expected Output:** evaluatePerformance() returns Evaluation.Excellent
    - **Explanation:** Tests the case where the number of performance records matches the threshold for "Excellent."
4. **Test Case 4: Performance List Over Achieving**
    - **Input:** performanceList with 5 performance records
    - **Expected Output:** evaluatePerformance() returns Evaluation.OverAchieving
    - **Explanation:** Tests the case where the number of performance records exceeds the threshold for "Excellent."
5. **Test Case 5: Performance List Unsatisfactory**
    - **Input:** performanceList with 1 performance record
    - **Expected Output:** evaluatePerformance() returns Evaluation.Unsatisfactory
    - **Explanation:** Tests the case where the number of performance records is below the lowest threshold.

6. **Test Case 6: Performance List with No Records**
    - **Input:** performanceList with 0 performance records
    - **Expected Output:** evaluatePerformance() returns Evaluation.Unsatisfactory
    - **Explanation:** Tests the case where there are no performance records.
7. **Test Case 7: Performance List with Multiple Records**
    - **Input:** performanceList with 6 performance records
    - **Expected Output:** evaluatePerformance() returns Evaluation.OverAchieving
    - **Explanation:** Tests the case where the number of performance records is significantly higher than the threshold for "Excellent."

---

5. ***Statement and Branch Coverage for "Payroll" Class:***

The Payroll class calculates employee salaries based on various attributes, including type, base salary, hours worked, tax, deductions, and bonuses. The calculatePay method computes the total pay for an employee based on their type.

- ***Statement and Branch Coverage:***

Method: calculatePay
Conditions to Test:

- Employee type: Hourly, Intern, or Other
- Variations in base salary, hours, tax, deductions, and bonuses

1. **Test Case 1: Hourly Employee**
    - **Input:** employeeType = Hourly, baseSalary = 20.0, hours = 40, tax = 50, deductions = 30, bonus = 100
    - **Expected Output:** calculatePay() returns 20.0 * 40 + 100 - 50 - 30 = 750.0
    - **Explanation:** Tests the calculation for an Hourly employee with non-zero hours, tax, deductions, and bonus.

2. **Test Case 2: Intern Employee**
    - **Input:** employeeType = Intern, baseSalary = 1500.0, tax = 100, deductions = 50, bonus = 200
    - **Expected Output:** calculatePay() returns 1500.0
    - **Explanation:** Tests the calculation for an Intern where the base salary is unaffected by tax, deductions, and bonus.
3. **Test Case 3: Full-Time Employee**
    - **Input:** employeeType = FullTime, baseSalary = 3000.0, tax = 200, deductions = 100, bonus = 500
    - **Expected Output:** calculatePay() returns 3000.0 + 500 - 200 - 100 = 3200.0
    - **Explanation:** Tests the calculation for a Full-Time employee with a base salary, bonus, tax, and deductions.
4. **Test Case 4: Hourly Employee with Zero Hours**

- **Input:** employeeType = Hourly, baseSalary = 20.0, hours = 0, tax = 10, deductions = 5, bonus = 50
- **Expected Output:** calculatePay() returns 20.0 * 0 + 50 - 10 - 5 = 35.0
- **Explanation:** Tests the calculation for an Hourly employee with zero hours worked.

5. **Test Case 5: Full-Time Employee with Zero Bonus**
   - **Input:** employeeType = FullTime, baseSalary = 2500.0, tax = 150, deductions = 75, bonus = 0
   - **Expected Output:** calculatePay() returns 2500.0 + 0 - 150 - 75 = 2275.0
   - **Explanation:** Tests the calculation for a Full-Time employee with no bonus.
6. **Test Case 6: Intern Employee with Deductions and Tax**
   - **Input:** employeeType = Intern, baseSalary = 1200.0, tax = 50, deductions = 20, bonus = 100
   - **Expected Output:** calculatePay() returns 1200.0
   - **Explanation:** Tests the calculation for an Intern where tax, deductions, and bonus do not affect the base salary.
7. **Test Case 7: Edge Case with Negative Values**
   - **Input:** employeeType = Hourly, baseSalary = 20.0, hours = 40, tax = -10, deductions = -20, bonus = -30
   - **Expected Output:** calculatePay() returns 20.0 * 40 - 30 - (-10) - (-20) = 800.0 - 30 + 10 + 20 = 800.0
   - **Explanation:** Tests the calculation with negative values for tax, deductions, and bonus.

---

6. <u>Statement and Branch Coverage for “LeaveManagment” Class:</u>

The LeaveManagement class handles operations related to leave requests, such as adding,

removing, approving, rejecting, and updating the status of leave requests.

- ***Statement and Branch Coverage:***

Methods:

- addLeaveRequest(LeaveRequest leaveRequest)
- removeLeaveRequest(int requestId)
- getLeaveRequest(int requestId)
- getAllLeaveRequests()
- updateLeaveStatus(int requestId, LeaveStatus status)
- approveLeaveRequest(int requestId)
- rejectLeaveRequest(int requestId)

Test Cases:

1. **Test Case 1: Add Leave Request**
    - **Input:** LeaveRequest with id = 1
    - **Expected Output:** getAllLeaveRequests() returns a list containing the leave request with id = 1
    - **Explanation:** Tests if a leave request is correctly added to the leaveRequests list.
2. **Test Case 2: Remove Existing Leave Request**
    - **Input:** requestId = 1
    - **Expected Output:** removeLeaveRequest(requestId) returns true and getLeaveRequest(requestId) returns null
    - **Explanation:** Tests the removal of an existing leave request.
3. **Test Case 3: Remove Non-Existing Leave Request**
    - **Input:** requestId = 2
    - **Expected Output:** removeLeaveRequest(requestId) returns false
    - **Explanation:** Tests the removal of a leave request that does not exist.
4. **Test Case 4: Get Existing Leave Request**

- **Input:** requestId = 1 (assuming leave request with this ID exists)
- **Expected Output:** getLeaveRequest(requestId) returns the leave request with id = 1
- **Explanation:** Tests retrieving an existing leave request.

5. **Test Case 5: Get Non-Existing Leave Request**
    - **Input:** requestId = 3 (assuming no leave request with this ID exists)
    - **Expected Output:** getLeaveRequest(requestId) returns null
    - **Explanation:** Tests retrieving a leave request that does not exist.
6. **Test Case 6: Update Leave Status to Accepted**
    - **Input:** requestId = 1, status = LeaveStatus.Accepted
    - **Expected Output:** getLeaveRequest(requestId).getLeaveStatus() returns LeaveStatus.Accepted
    - **Explanation:** Tests updating the status of a leave request to "Accepted."
7. **Test Case 7: Approve Leave Request**
    - **Input:** requestId = 2 (assuming leave request with this ID exists)
    - **Expected Output:** approveLeaveRequest(requestId) updates the status to LeaveStatus.Accepted
    - **Explanation:** Tests if the approveLeaveRequest method correctly updates the leave request status.
8. **Test Case 8: Reject Leave Request**
    - **Input:** requestId = 2 (assuming leave request with this ID exists)
    - **Expected Output:** rejectLeaveRequest(requestId) updates the status to LeaveStatus.Rejected
    - **Explanation:** Tests if the rejectLeaveRequest method correctly updates the leave request status.
9. **Test Case 9: Get All Leave Requests**
    - **Input:** After adding multiple leave requests
    - **Expected Output:** getAllLeaveRequests() returns a list with all added leave requests
    - **Explanation:** Tests if getAllLeaveRequests correctly retrieves all leave requests.

Integration Testing

Big Bang

Big Bang Testing is the type of software testing which we used, where all components or modules of a system are tested together in a single, large testing phase. It contrasts with incremental or iterative testing approaches, where parts of the system are tested as they are developed.

Integration Process

(a) Develop Individual Components: Each class (Employee, HREmployee, Address ,etc.) and the GUI are developed independently. Unit tests are written for each class to ensure its functionality in isolation.

(b) Integration of Classes: Once all classes and the GUI are developed, they are integrated simultaneously into the system. The interactions between different classes and between the classes and the GUI are tested.

(c) Test Scenarios: Various test scenarios are designed to validate the integration of classes and GUI components. These scenarios cover different functionalities of the system, such as Employee Initialization, set and get employee address, set and get payroll, set and get payroll, and find employee by id remove employee, leave request, add and remove leave request, etc....

(d) Test Execution: The test cases are executed, and the behavior of the integrated system is observed. This includes verifying that interactions between classes and the GUI function correctly and that data is passed between components accurately.

(e) Bug Fixing and Iteration: Any defects or issues identified during testing are addressed, and the integration is retested. This iterative process continues until the system behaves as expected and meets the specified requirements.

(f) Validation: Once the integration testing is complete and all issues are resolved, the system undergoes validation to ensure that it meets the user’s requirements and expectations.