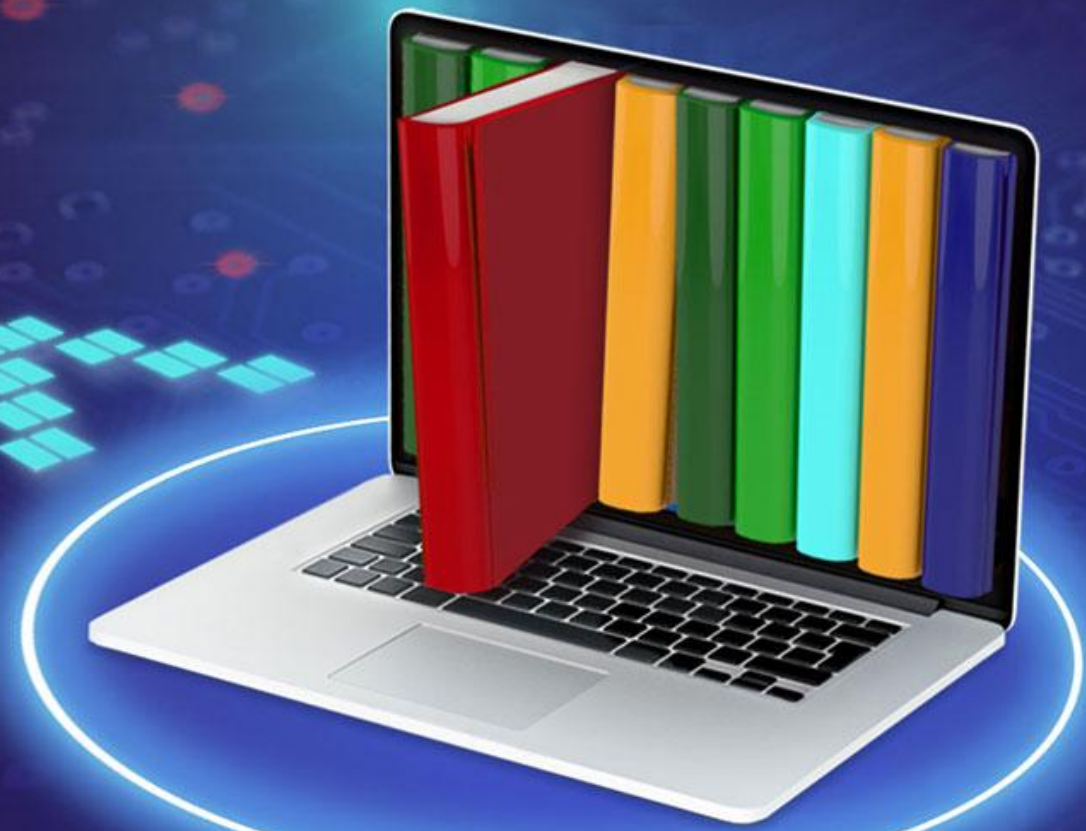
POS-System SWE-303 Project



Course: **SWE 303 – Software Testing & Quality Assurance**
Accepted by: **Ari Gjerazi, Jurgen Cama**
Group Members:
Ester Shumeli
Abdulaziz Bezan
Eridjon Krrashi
Evisa Nela

Github Repository: https://github.com/Arkbezo/Software-Testing-Project-Fall-2025

PROJECT OVERVIEW

Java-based **Point of Sale (POS)** system developed as a case study for software testing
The system manages **users, products, customers, suppliers, and transactions**

Main objective is to ensure **software quality, correctness, and reliability**

Testing activities were performed **strictly according to project guidelines**

The project applies **multiple testing levels**, including:
•Static Testing
•Testing Analysis
•Unit Testing
•Integration Testing
•System Testing

Different **testing techniques** were used, such as:
-Boundary Value Testing
-Equivalence Class & Decision Table Testing
-Code Coverage and MC/DC
-Static Code Analysis (SpotBugs & SonarQube)

Each group member contributed **individually**, with **distinct responsibilities**, ensuring:
-No duplication of work
-Clear accountability
-Full testing coverage of the system

Task Distribution Overview

01

Ester Shumeli

-Group Leader, task division, check/document overall progress
-Testing Analysis
-Integration Testing

02

Abdulaziz Bezan

-Vice Team Leader
-Testing Analysis
-System Testing

03

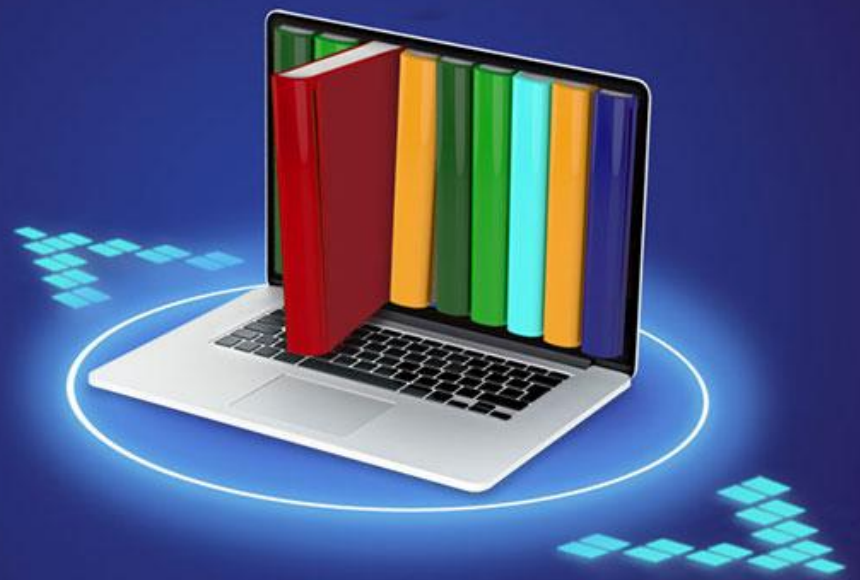
Eridjon Krrashi

-Static Testing
-Testing Analysis
-Unit Testing

04

Evisa Nela

-Testing Analysis
-Integration Testing



Point of Sale

admin Logout

- Customers
- Supplier
- Employees
- Products
- Category
- Sales
- Invoice
- Reports

Customer Information

Id	Name	Phone No
1	fawad	12345
3	fawad iqbal	1234568
6	saman	77-88-9
9	sam	9999988
10	fawad	03149972883
27	saman	77-88-9
38	fawad Iqbal	03149972883
40	wertyq	1234567890
41	fawad iqbal	1234568133
43	fawad Iqbal	03149972883
44	fawad iqbal	1234568133
45	fawad Iqbal	03149972883
46	fawad Iqbal	03149972883
47	fawad Iqbal	03149972883
48	fawad iqbal	1234568133
49	fawad Iqbal	03149972883
50	fawad Iqbal	03149972883
51	fawad Iqbal	03149972883
52	fawad Iqbal	03149972883
53	fawad Iqbal	03149972883
54	fawad Iqbal	03149972883
55	wertyq	1234567890
57	ahmad1	1234567899
58	fawad iqbal	03149972883
62	OtherName_IntTest_456e66	0690000002

Add Delete Update

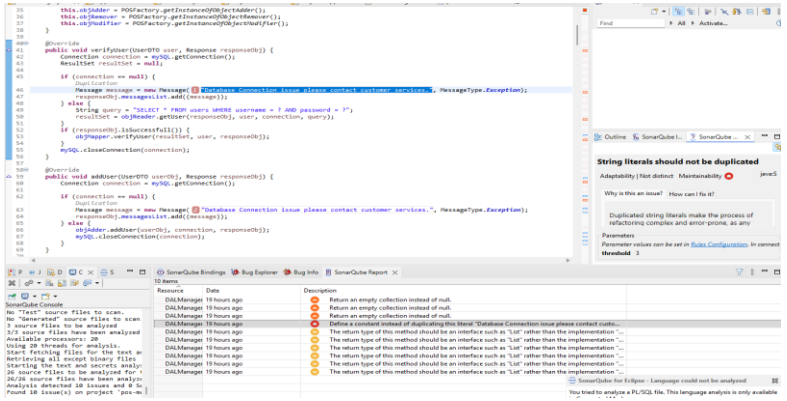
Search by Name:

Static Testing

Static testing evaluates the system without executing the code

Focus areas:
- Code quality
- Maintainability
- Potential runtime defects

Tools used:
- SonarQube
- SpotBugs
- Manual Code Review



Key Static Testing Findings
- Duplicate string literals (database error messages)
- Missing null checks on database connections
- Methods returning null instead of empty collections
- Use of concrete return types (ArrayList) instead of interfaces (List)
- DALManager identified as the **most critical class**

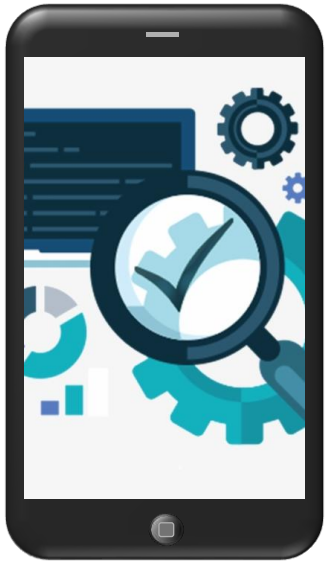
Performed by: Eridjon Krrashi

Testing Analysis

Each member performed **individual Testing Analysis** & selected **distinct methods**
Analysis techniques used:
- Boundary Value Testing (BVT)
- Equivalence Class / Decision Table Testing
- Code Coverage & MC/DC

Ester Shumeli

Methods Analyzed:
isValidPassword(...)
searchCustomersByName(...)
isValildPhoneNo(...)
Result:
All logical boundaries verified
Independent condition impact proven



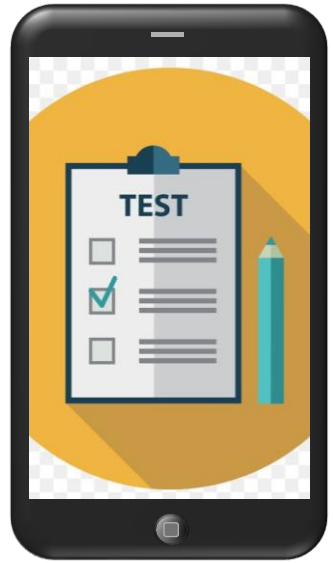
Abdulaziz Bezan

Methods Analyzed:
- calculateTotal()
- addToCartBtnActionPerformed()
- create_invoiceActionPerformed()

Focused on:
- Cart total calculation logic
- User interaction edge cases
- Identification of **missing input validation and error handling gaps**

Eridjon Krrashi

Class Analyzed: DALManager
Coverage Achieved:
100% Statement Coverage
100% Branch Coverage
100% Condition Coverage



Evisa Nela

Methods Analyzed:
isSessionExpired()
isSuccessfull()
getErrorMessages()
Result:
Time-based logic verified
Correct message filtering and output behavior

Unit Testing

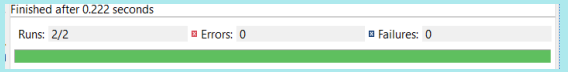
Performed by: Eridjon Krrashi

Tested **core logical classes**, including:
- DALManager (data access & search operations)
- ObjectAdder (insert operations & foreign key handling)
- ObjectModifier (update operations)
- ObjectRemover (delete operations)
- ObjectMapper (ResultSet → DTO mapping)
- CommonValidator (input validation rules)

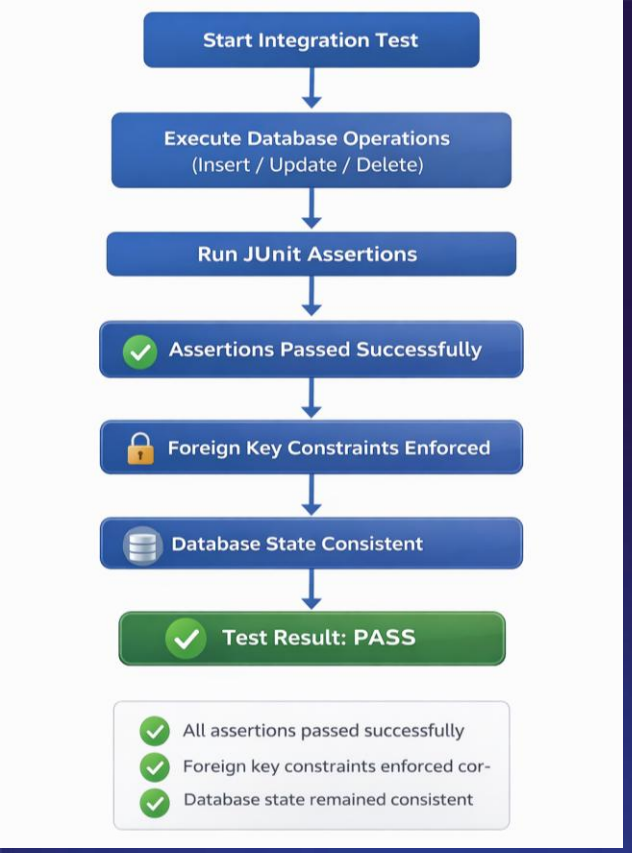
```
        // 5) VERIFY (find by id)
        EmployeeDTO found = employees.stream()
                .filter(e -> e.getId() == employeeId)
                .findFirst()
                .orElse(null);

        assertNotNull(found, "Updated employee not found in getEmployees()");

        assertEquals(updatedName, found.getName(), "Employee name was not updated correctly");
        assertEquals(updatedPhone, found.getPhoneNumber(), "Employee phoneNumber was not updated correct

        // verify old name is not present for this id
        assertNotEquals(uniqueName, found.getName(), "Employee name should not still be old value");
    }

    private int syncEmployeeIdByName(String name) {
        Response res = new Response();
        ArrayList<EmployeeDTO> employees = dalManager.getEmployees(res);
```

Unit tests covered:
- CRUD operations for all entities
- Search functionality and data retrieval
- Boundary values for IDs (−1, 0, valid ID, MAX_INT)
- Equivalence class testing (valid, invalid, null, empty inputs)
- Safe handling of null values and SQL exceptions

Results:



Over **60 unit test cases executed**
100% pass rate across all tested classes
No NullPointerExceptions or runtime crashes detected
All critical POS functionality validated in isolation



Integration Testing

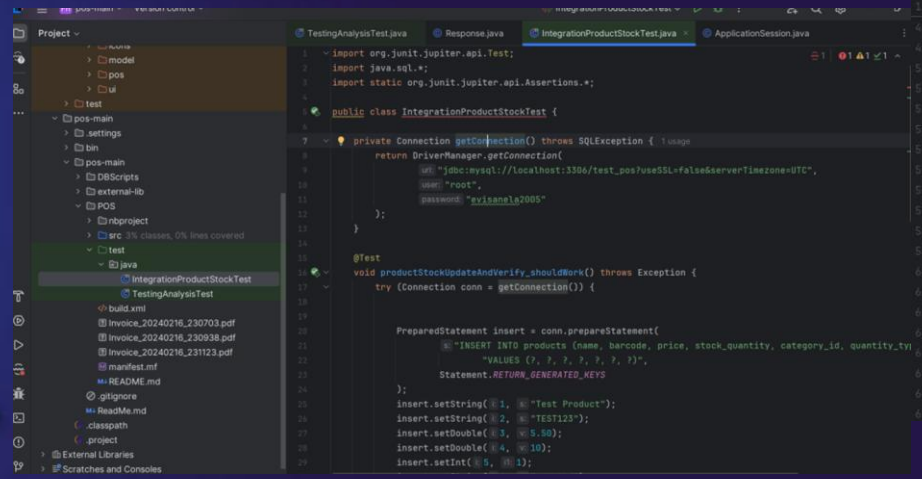
: Evisa Nela

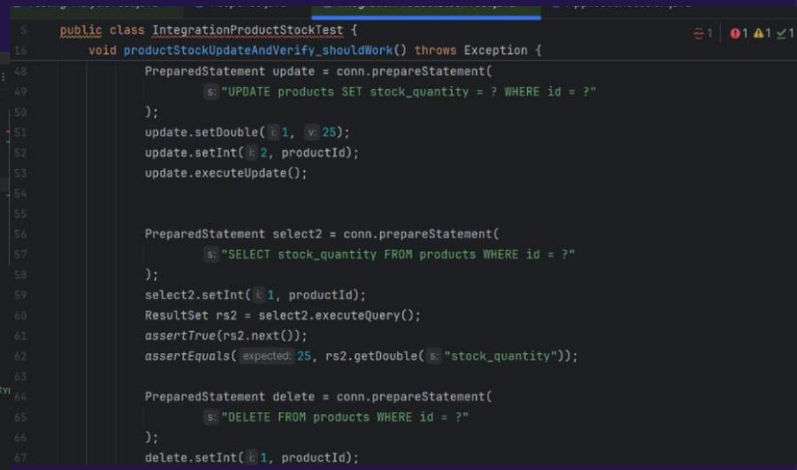
Test 1: Product Stock Update & Verification

- Insert product into database
- Retrieve product and verify initial stock
- Update stock quantity and re-verify
- Delete product after test (cleanup)
- Confirms correct interaction between **Product module and database layer**

Test 2: Supplier Delete Constraint (Parent–Child)

Insert supplier and linked product
Attempt supplier deletion (blocked by FK constraint)
Delete dependent product
Successfully delete supplier
Validates **referential integrity enforcement**



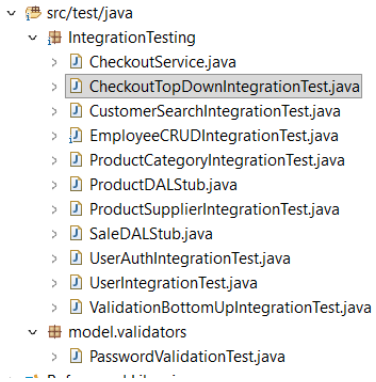


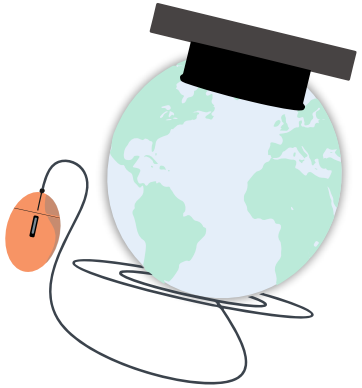
Integration Testing

: Ester Shumeli

Key scenarios tested:
-User creation, retrieval, and authentication
-Checkout logic: stock validation, total calculation, sale persistence
-Customer phone number validation across validator layers
-Product–Category and Product–Supplier foreign key enforcement
-Customer and Employee full CRUD workflows (insert → search → update → delete)

Designed and executed multiple integration tests covering core POS subsystems
Tests were implemented using JUnit with both real database integration and stubs, depending on the strategy





Integration strategies applied:
- **Decomposition-Based Integration** (User insertion & authentication)
- **Top-Down Integration** (Checkout process using stubs)
- **Bottom-Up Integration** (Validation logic, DAL workflows)
- **Sandwich (Hybrid) Integration** (Product–Category relationship)
- **Big Bang Integration** (Employee CRUD subsystem)

What was verified ☑

- ✓ Correct interaction between **application logic, DAL, and MySQL database**
- ✓ Enforcement of **foreign key constraints** and referential integrity
- ✓ Proper error handling for invalid input and constraint violations
- ✓ Safe cleanup and restoration of database state after each test

System Testing

Performed by: Abdulaziz Bezan , Evisa Nela

Evisa Nela

• Performed **end-to-end system testing** of the POS system

• Verified **user login authentication** across UI, controller, DAL, and database

• Tested **supplier deletion constraints**, ensuring foreign key enforcement

• Confirmed correct system behavior and data integrity

• **All system tests passed**

Abdulaziz Bezan

- Performed **end-to-end system testing** of all POS modules

- Found **critical failures** in UI-database integration and error handling

-Identified **missing input validation** causing crashes with invalid data

-Test coverage analysis revealed untested error paths (60% path coverage)

-Provided **specific fixes** for validation and exception handling

-Led to a system FAILURE, lack of error handling makes it less efficient.

THANK YOU ☺