Relatório VVS – Assignment 2

Guilherme Sousa – 58170

1. Introduction

The second assignment was an interesting project where we used HtmlUnit to test an application that ran on a server in Wildfly, we also used DbSetup given by the teacher to populate the database, to later do tests on it. Finally, Mockito was also used, and we saw that it wasn't possible to mock.

Before using HtmlUnit, DbSetup and Mockito, it was asked that we searched for bugs, in my case, all bugs were found by simply "playing" with the web application.

The report will be divided by this order:

- **Introduction**
- **Bugs found.**
- **Testing with *HtmlUnit*.**
- **Testing with *DbSetup*.**
- **Usage of *Mockito*.**

2. Bugs found

As it was asked in the assignment, I found three bugs, one of them related to character encoding and the other two related to the usage of VAT.

Bug list:

- **"Client Info" doesn't support different characters (ex: Chinese characters)**
- **When we Insert a new sale with an invalid VAT (ex: 1) an error is shown but the sale is added anyways**
- **"Show Customer Sale's Delivery" accepts VATs that do not exist**

Before showing each bug, here are my computer configurations, which was used to test the application:

CPU: AMD Ryzen 5 7600X
GPU: XFX Radeon RX 6700 Speedster SWFT309 10GB GDDR6
SSD: SSD M.2 2280 Western Digital WD Blue SN570 1TB TLC NVMe PCIe Gen 3.0x4
RAM: Kingston Fury Beast RGB 32GB (2x16GB) DDR5-5200MHz 1R CL36
Motherboard: ATX MSI Pro B650-P WiFi
Windows 11 Pro
Java 8 JDK

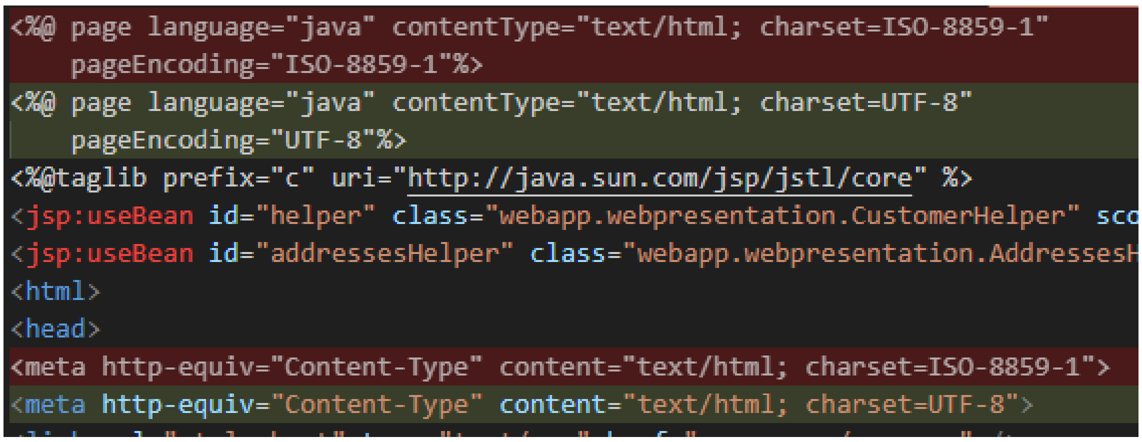
Bug 1 – “Client Info” doesn't support different characters

Bug Context:

- **Location**: CustomerInfo.jsp
- **Trigger**: Adding a new customer with special characters

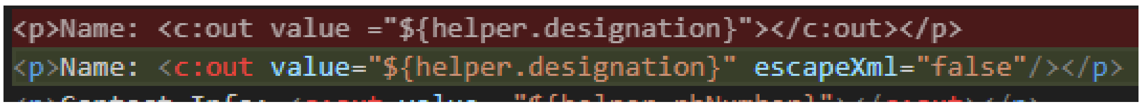
Bug Resolution:

Bug in the code:



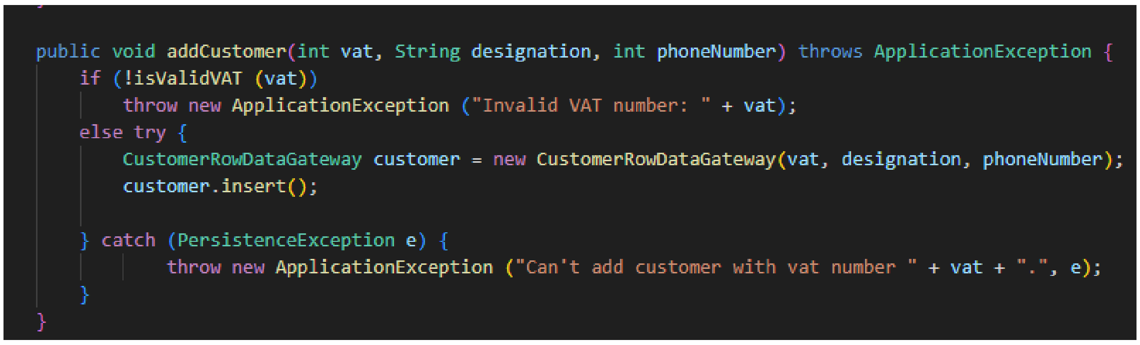
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<jsp:useBean id="helper" class="webapp.webpresentation.CustomerHelper" sco
<jsp:useBean id="addressesHelper" class="webapp.webpresentation.AddressesH
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

To make sure the special characters appeared correctly, this change was also necessary, also in CustomerInfo.jsp:



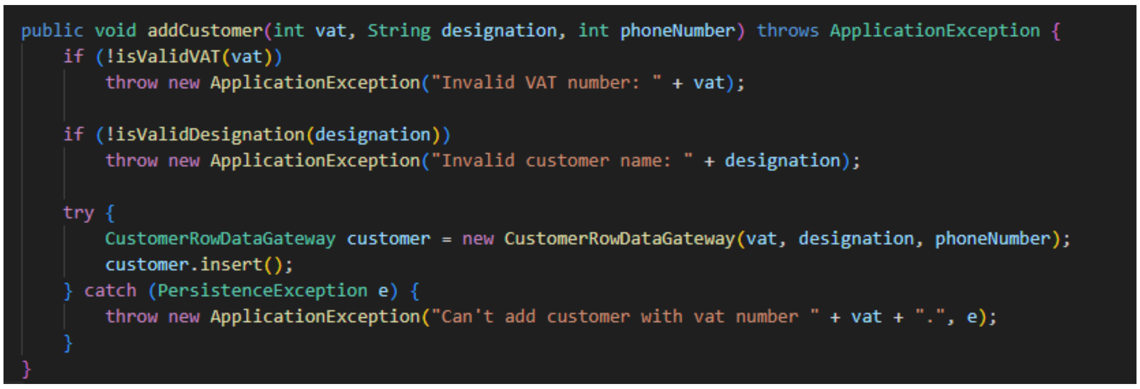
```
<p>Name: <c:out value ="${helper.designation}"></c:out></p>
<p>Name: <c:out value="${helper.designation}" escapeXml="false"/></p>
```

But this change made it so XSS attacks could happen so to make sure that wouldn’t happen I also changed the addCustomer function in the CustomerService from this:

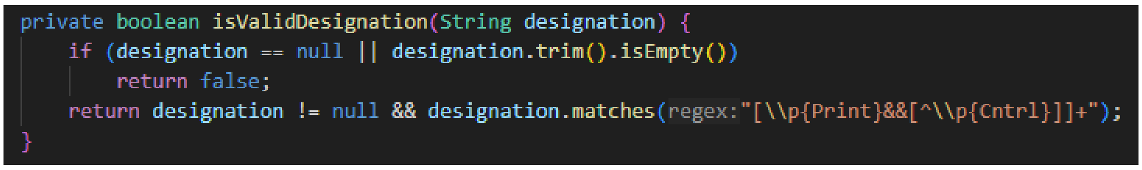


```
public void addCustomer(int vat, String designation, int phoneNumber) throws ApplicationException {
    if (!isValidVAT (vat))
        throw new ApplicationException ("Invalid VAT number: " + vat);
    else try {
        CustomerRowDataGateway customer = new CustomerRowDataGateway(vat, designation, phoneNumber);
        customer.insert();

    } catch (PersistenceException e) {
            throw new ApplicationException ("Can't add customer with vat number " + vat + ".", e);
    }
}
```

To this:



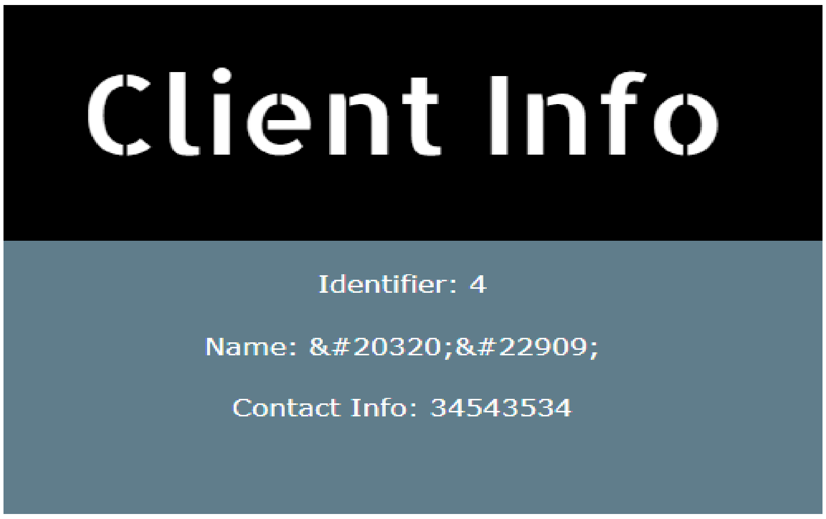
```
public void addCustomer(int vat, String designation, int phoneNumber) throws ApplicationException {
    if (!isValidVAT(vat))
        throw new ApplicationException("Invalid VAT number: " + vat);

    if (!isValidDesignation(designation))
        throw new ApplicationException("Invalid customer name: " + designation);

    try {
        CustomerRowDataGateway customer = new CustomerRowDataGateway(vat, designation, phoneNumber);
        customer.insert();
    } catch (PersistenceException e) {
        throw new ApplicationException("Can't add customer with vat number " + vat + ".", e);
    }
}
```



```
private boolean isValidDesignation(String designation) {
    if (designation == null || designation.trim().isEmpty())
        return false;
    return designation != null && designation.matches(regex:"[\\p{Print}&&[^\\p{Cntrl}]]+");
}
```

After testing with these changes the special characters were showing as they should (in my case I tested with Chinese characters)

It went from this:



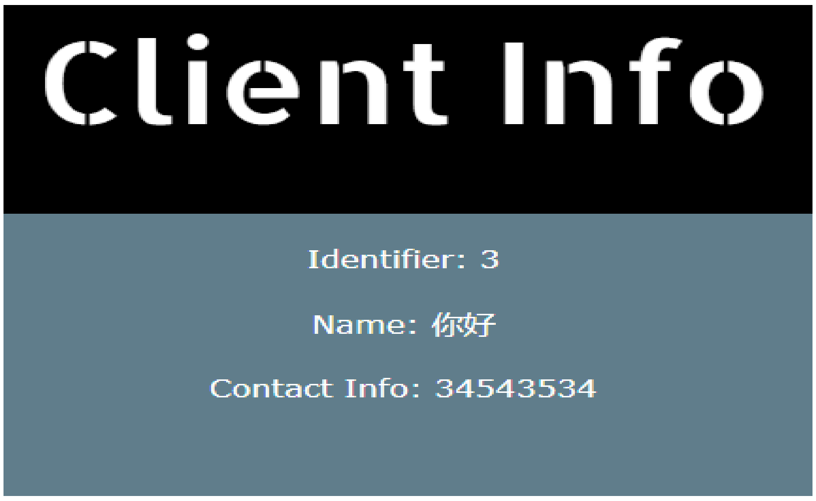
Client Info

Identifier: 4

Name: &#20320;&#22909;

Contact Info: 34543534

To this:



Client Info

Identifier: 3

Name: 你好

Contact Info: 34543534

After fixing this error I found a similar one with the addresses but It seemed harder to fix and I didn't want to lose more time with it because I already had my three bugs selected.

Reachability:

- **Flow Path:**
    - User Input → AddCustomerPageController → CustomerService → CustomerInfo.jsp

- **Detailed Flow Analysis:**

  Entry Point: **AddCustomerPageController**

    - Receives customer data from form
    - Processes VAT, phone, and designation

  Service Layer: **CustomerService**

    - Validates customer data
    - Creates new customer record
    - Retrieves customer information

  View Layer: **CustomerInfo.jsp**

    - Displays customer information
    - Bug Location: Character encoding issue in display

Infection:

- **Bug Source**:
    - Initial Point: CustomerInfo.jsp
    - Root Cause: ISO-8859-1 encoding instead of UTF-8
    - Affected Component: Character display in customer information

- **Infection Path:**
    - Database → Service Layer → Helper → JSP View

- **Data Corruption Points:**
  1. **Input Stage**
      - Customer name with special characters

    - Contact information with special characters
1. **Processing Stage**
    - CustomerHelper processes customer data
    - Data transformation between layers
1. **Display Stage**
    - JSP rendering of customer information
    - Form field values

- **Affected Components:**
    - Customer Information Display

Propagation:

- **Initial Propagation Point:**
    - Source: CustomerInfo.jsp with ISO-8859-1 encoding
    - Trigger: Displaying customer data with special characters

- **Propagation Path:**
    - Database → Service Layer → Helper → JSP View → User Interface

- **Data Flow Propagation:**
    - Database Layer
        - Customer data stored with special characters
        - Contact information with special characters
    - Service Layer
        - CustomerService retrieves data
        - Data passed to CustomerHelper
    - Helper Layer
        - CustomerHelper processes customer data
        - Data prepared for JSP display
    - View Layer
        - JSP renders data with incorrect encoding
        - Special characters displayed incorrectly
        - User sees corrupted data

- **Propagation Effects**
  - Customer Information:

```
○  <p>Name: <c:out value ="${helper.designation}"></c:out></p>
```

    Special characters in names displayed incorrectly:
    Example: "你好" might display as " &amp;#20320;&amp;#22909”

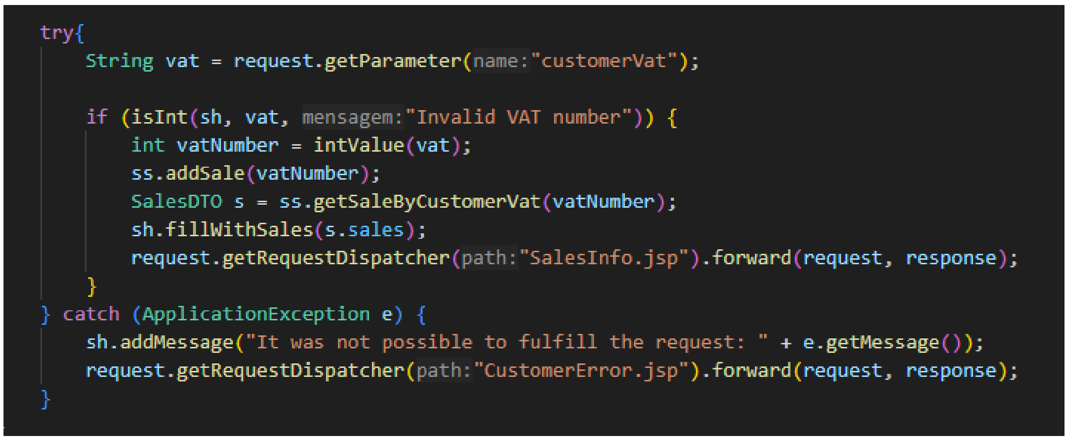
Bug 2 – When we Insert a new sale with an invalid VAT an error is shown but the sale is added anyways

Bug Context:

- **Location**: AddSalePageController.java
- **Trigger**: Adding a new sale with an invalid VAT (must be an int)

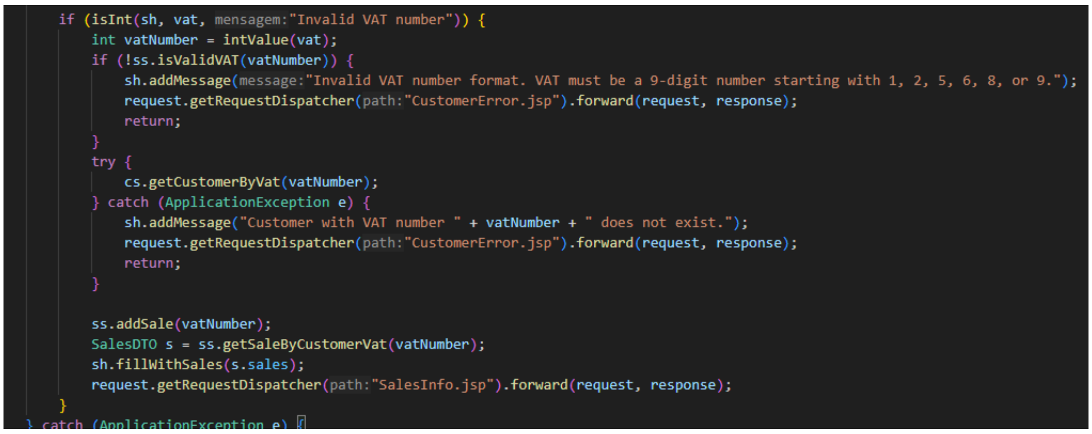
Bug Resolution:

Bug in the code:



```
try{
    String vat = request.getParameter(name:"customerVat");

    if (isInt(sh, vat, mensagem:"Invalid VAT number")) {
        int vatNumber = intValue(vat);
        ss.addSale(vatNumber);
        SalesDTO s = ss.getSaleByCustomerVat(vatNumber);
        sh.fillWithSales(s.sales);
        request.getRequestDispatcher(path:"SalesInfo.jsp").forward(request, response);
    }
} catch (ApplicationException e) {
    sh.addMessage("It was not possible to fulfill the request: " + e.getMessage());
    request.getRequestDispatcher(path:"CustomerError.jsp").forward(request, response);
}
```

To fix this bug we need to check that the VAT has the correct format, and the customer exists (the customer check would be enough, but this gives better errors for the user to understand what’s wrong:



```
        if (isInt(sh, vat, mensagem:"Invalid VAT number")) {
            int vatNumber = intValue(vat);
            if (!ss.isValidVAT(vatNumber)) {
                sh.addMessage(message:"Invalid VAT number format. VAT must be a 9-digit number starting with 1, 2, 5, 6, 8, or 9.");
                request.getRequestDispatcher(path:"CustomerError.jsp").forward(request, response);
                return;
            }
            try {
                cs.getCustomerByVat(vatNumber);
            } catch (ApplicationException e) {
                sh.addMessage("Customer with VAT number " + vatNumber + " does not exist.");
                request.getRequestDispatcher(path:"CustomerError.jsp").forward(request, response);
                return;
            }

            ss.addSale(vatNumber);
            SalesDTO s = ss.getSaleByCustomerVat(vatNumber);
            sh.fillWithSales(s.sales);
            request.getRequestDispatcher(path:"SalesInfo.jsp").forward(request, response);
        }
    } catch (ApplicationException e) {
```

The function isValidVAT comes from SaleService and was originally private so I changed it to public so it could be used in other classes.

Reachability:

- **Flow Path:**
  - User Input → AddSalePageController → SaleService → SalesInfo.jsp

- **Detailed Flow Analysis:**

  Entry Point: **AddSalePageController**

  - Receives VAT number from form
  - Performs initial integer validation
  - Calls SaleService for VAT format validation
  - Initiates sale creation process

  Service Layer: **SaleService**

  - Validates VAT number format
  - Creates a new sale record
  - Retrieves sale information for display
  - Bug Location: Insufficient validation before sale creation

  View Layer: **SalesInfo.jsp**

  - Displays sale information
  - Shows error messages if validation fails
  - Bug Location: Sale appears in view even with invalid VAT

Infection:

- **Bug Source**:
  - Initial Point: AddSalePageControllerRoot
  - Cause: ISO-8859-1 encoding instead of UTF-8
  - Root Cause: Insufficient VAT validation before sale creation
  - Affected Component: Sale creation and display process

- **Infection Path:**

User Input → AddSalePageController → SaleService → SaleRowDataGateway → Database → SalesInfo.jsp

- **Data Corruption Points:**
  1. **Input Stage**
     - Invalid VAT number format
     - Non-existent customer VAT
     - Missing customer validation
     - Missing VAT format validation
  2. **Processing Stage**
     - SaleService processes sale creation
     - SaleRowDataGateway inserts invalid sale
     - No transaction rollback on validation failure
  3. **Display Stage**
     - SalesInfo.jsp shows invalid sale
     - Error message displayed but sale still created
     - Inconsistent state between error and created sale

- **Affected Components:**
  - Sale Creation Process
  - Sale Display System
  - Error Handling Mechanism
  - Database Integrity

Propagation:

- **Initial Propagation Point:**
  - Source: AddSalePageController with insufficient validation
  - Trigger: Attempting to create a sale with invalid VAT number

- **Propagation Path:**
  User Input → AddSalePageController → SaleService → SaleRowDataGateway → Database → SalesInfo.jsp → User Interface

- **Data Flow Propagation:**
  - Controller Layer

        - AddSalePageController receives VAT input
        - Only performs basic integer validation
        - Proceeds with sale creation despite invalid VAT
    - Service Layer
        - SaleService processes sale creation
        - VAT validation occurs but sale still created
        - No customer existence verification
    - Data Layer
        - SaleRowDataGateway inserts invalid sale
        - Database accepts sale with invalid VAT
        - No foreign key constraint enforcement
    - View Layer
        - SalesInfo.jsp displays invalid sale
        - Error message shown but sale still visible
        - Inconsistent state displayed to user

- **Propagation Effects**
    - Sale Creation:

```
if (isInt(sh, vat, "Invalid VAT number")) {
```

      Only checks if the VAT is an int, ignores its format and if there is an user with that VAT.
    - Database State:
        - Invalid sales stored in database
        - Orphaned sales without valid customers
        - Inconsistent data relationships
    - User Interface:
        - Error message displayed
        - Invalid sale still appears in list
        - Confusing user experience

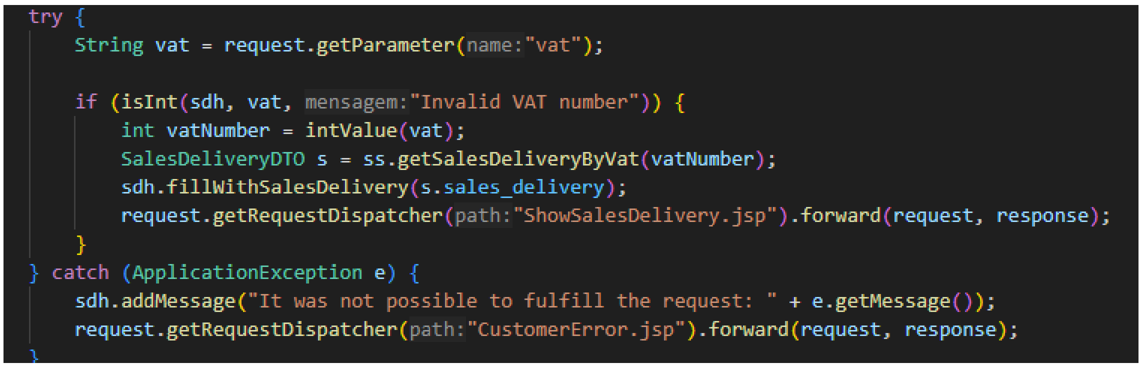
Bug 3 – In "Show Customer Sale's Delivery" it is possible to put a VAT that does not exist, no error occurs, and Sale Info appears empty

Bug Context:

- **Location**: GetSaleDeliveryPageController.java
- **Trigger**: Showing customer's sale delivery with an invalid VAT (must be an int)

Bug Resolution:

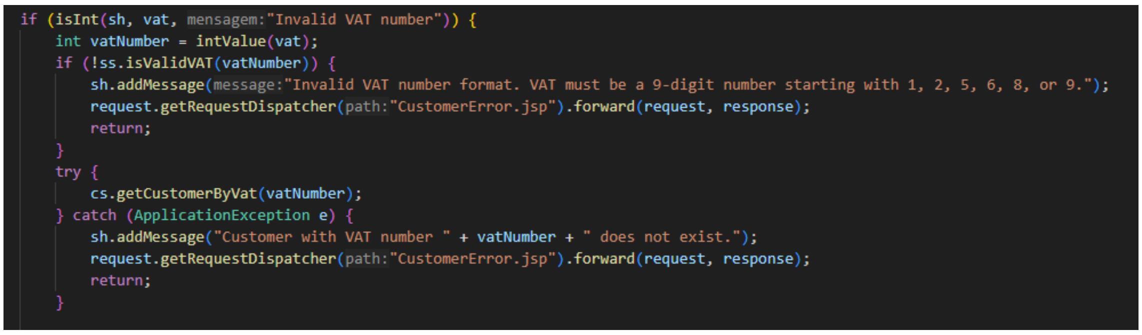
Bug in the code:



```java
try {
    String vat = request.getParameter(name:"vat");

    if (isInt(sdh, vat, mensagem:"Invalid VAT number")) {
        int vatNumber = intValue(vat);
        SalesDeliveryDTO s = ss.getSalesDeliveryByVat(vatNumber);
        sdh.fillWithSalesDelivery(s.sales_delivery);
        request.getRequestDispatcher(path:"ShowSalesDelivery.jsp").forward(request, response);
    }
} catch (ApplicationException e) {
    sdh.addMessage("It was not possible to fulfill the request: " + e.getMessage());
    request.getRequestDispatcher(path:"CustomerError.jsp").forward(request, response);
}
```

It's basically the same bug as bug number two, the code is simply checking if the VAT is an int or not and it ignores the format of the VAT and if there is a customer with that VAT or not.

Corrected version of the code:



```java
if (isInt(sh, vat, mensagem:"Invalid VAT number")) {
    int vatNumber = intValue(vat);
    if (!ss.isValidVAT(vatNumber)) {
        sh.addMessage(message:"Invalid VAT number format. VAT must be a 9-digit number starting with 1, 2, 5, 6, 8, or 9.");
        request.getRequestDispatcher(path:"CustomerError.jsp").forward(request, response);
        return;
    }
    try {
        cs.getCustomerByVat(vatNumber);
    } catch (ApplicationException e) {
        sh.addMessage("Customer with VAT number " + vatNumber + " does not exist.");
        request.getRequestDispatcher(path:"CustomerError.jsp").forward(request, response);
        return;
    }
```

It was also necessary to make the isValidVAT a public method in SaleService, same as bug number two.

Reachability:

- **Flow Path:**
    - User Input → GetSaleDeliveryPageController → SaleService → ShowSalesDelivery.jsp

- **Detailed Flow Analysis:**

  Entry Point: **GetSaleDeliveryPageController**

    - Receives VAT number from form
    - Performs initial integer validation
    - Calls SaleService for VAT format validation
    - Bug Location: Only validates if VAT is an integer, missing customer existence check

  Service Layer: **SaleService**

    - Validates VAT number format
    - Retrieves sales delivery information by VAT
    - Bug Location: No validation of customer existence before retrieving sales delivery
    - Returns empty sales delivery data for non-existent customers

- **View Layer**: **ShowSalesDelivery.jsp**
    - Displays sales delivery information
    - Shows empty results for non-existent customers
    - Bug Location: No error message displayed for invalid/non-existent VAT numbers

Infection:

- **Bug Source**:
    - Initial Point: GetSaleDeliveryPageController
    - Root Cause: Insufficient VAT validation before retrieving sales delivery information
    - Affected Component: Sales delivery display process

- **Infection Path:**

  User Input → GetSaleDeliveryPageController → SaleService → SaleDeliveryRowDataGateway → Database → ShowSalesDelivery.jsp

- **Data Corruption Points:**
  1. **Input Stage**
     - Invalid VAT number format
     - Non-existent customer VAT
     - Missing customer validation
     - Missing VAT format validation
  2. **Processing Stage**
     - SaleService processes sales delivery retrieval
     - SaleRowDataGateway queries for non-existent customer
     - No validation before database query
     - Empty result set returned instead of error
  3. **Display Stage**
     - ShowSalesDelivery.jsp displays empty results
     - No error message for invalid/non-existent VAT
     - Inconsistent user feedback

- **Affected Components:**
  - Sales Delivery Retrieval Process
  - Sale Delivery Display System
  - Error Handling Mechanism
  - User Feedback System
  - Data Validation System

Propagation:

- **Initial Propagation Point:**
  - Source: GetSaleDeliveryPageController with insufficient validation
  - Trigger: Attempting to show sales delivery with invalid/non-existent VAT number

- **Propagation Path:**

  User Input → GetSaleDeliveryPageController → SaleService → SaleDeliveryRowDataGateway → Database → ShowSalesDelivery.jsp → User Interface

- **Data Flow Propagation:**
  - Controller Layer

        - GetSaleDeliveryPageController receives VAT input
        - Only performs basic integer validation
        - Proceeds with sale delivery retrieval despite invalid VAT
    - Service Layer
        - SaleService processes sale delivery retrieval
        - No VAT format validation
        - No customer existence verification
    - Data Layer
        - SaleDeliveryRowDataGateway queries for non-existent customer
        - Database returns empty result set
        - No validation before query execution
    - View Layer
        - ShowSalesDelivery.jsp displays empty res
        - No error message for invalid/non-existent VAT
          Misleading empty display instead of error

- **Propagation Effects**
    - Sales Delivery Retrieval:
    -
      ```
      if (isInt(sh, vat, "Invalid VAT number")) {
      ```

      Only checks if the VAT is an int, ignores its format and if there is an user with that VAT.
    - Database State:
        - Empty result sets for invalid VATs
        - No validation of customer existence
        - Inconsistent data relationships
    - User Interface:
        - Empty display shown
        - No error message for invalid VAT
        - Confusing user experience
        - Misleading feedback

3. Testing with HtmlUnit

Address Addition Test

Objective: Verify address management functionality
Test: Added two new addresses for an existing customer
Result: PASS
Verification:

- Address table correctly displayed both new addresses
- Total row count increased by exactly two
- All address details were properly stored and displayed

Customer Creation Test

Objective: Validate customer registration system
Test: Created two new customers and verified listing
Result: PASS
Verification:

- Both customers appeared in "List All Customers" view
- All customer information (VAT, name, phone) was correctly displayed
- Data integrity maintained across customer records

Sale Status Test – Open

Objective: Verify initial sale status
Test: Created a new sale for a customer
Result: PASS
Verification:

- Sale appeared in customer's sales list
- Status correctly showed as "Open"
- Sale details properly associated with customer

Sale Status Test – Closed

Objective: Verify sale status update functionality
Test: Closed an existing sale
Result: PASS
Verification:

- Sale status changed from "Open" to "Closed"
- Status update persisted in database
- Change reflected immediately in customer's sales list

End-to-End Delivery Test

Objective: Validate complete sales delivery workflow
Test: Created customer → sale → delivery → verification
Result: PASS
Verification:

- All intermediate pages displayed correct information
- Customer creation successful
- Sale properly associated with customer
- Delivery correctly linked to sale
- Final delivery view showed all expected information

4. Testing with DbSetup

VAT Uniqueness Test

Objective: Verify VAT number uniqueness constraint
Test: Attempted to add customer with existing VAT
Result: PASS
Verification:

- System correctly rejected duplicate VAT
- Appropriate error message displayed
- Database integrity maintained

Customer Update Test

Objective: Validate customer information update
Test: Modified customer contact information
Result: PASS
Verification:

- Changes successfully saved to database
- Updated information correctly displayed
- Data persistence confirmed

Customer Deletion Test – Empty List

Objective: Verify complete customer removal
Test: Deleted all customers
Result: PASS
Verification:

- Customer list became empty
- No residual customer data remained
- System state properly reset

Customer Re-creation Test

Objective: Validate customer re-registration
Test: Deleted and re-added same customer
Result: PASS
Verification:

- Customer successfully re-added
- No exceptions occurred
- All customer data properly restored

Customer Deletion – Sales Cleanup

Objective: Verify cascading deletion
Test: Deleted customer with associated sales
Result: FAIL
Verification:

- Customer successfully removed
- All associated sales were not deleted
- Database referential integrity was not maintained

Fix:

- Add ON DELETE CASCADE to the foreign key constraints in the database schema for the SALE, ADDRESS, and SALEDELIVERY tables where they reference CUSTOMER.All associated sales were not deleted (taken from backlog from my colleague Gabriel Henriques as he was the one who found the bug)
- Change the order of the DROP TABLE making sure DROP TABLE CUSTOMER comes last, otherwise it won't be able to delete the tables due to foreign key dependencies
- After these changes the associated sales were deleted and database referential integrity was maintained

Sale Addition Test

Objective: Verify sale creation tracking
Test: Added new sale to system
Result: PASS
Verification:

- Total sales count increased by one
- New sale properly recorded
- Count accuracy maintained

Extra tests:

Sale Status Test

Objective: Validate sale status management
Test: Created and closed a sale
Result: PASS
Verification:

- Sale status properly tracked
- Status changes correctly persisted
- Status history maintained

Sale Delivery Test

Objective: Verify sale delivery functionality
Test: Created sale delivery for existing sale
Result: PASS
Verification:

- Delivery properly associated with sale
- Delivery information correctly stored
- Customer-sale-delivery relationship maintained

Sale Delivery Validation Test

Objective: Verify sale delivery constraints
Test: Attempted to create delivery for non-existent sale
Result: PASS
Verification:

- System correctly rejected invalid delivery
- Appropriate error handling
- Data integrity maintained

Sale Delivery Association Test

Objective: Validate sale-delivery relationship
Test: Created delivery and verified customer association

Result: PASS
Verification:

- Delivery correctly linked to sale
- Customer association maintained
- All relationships properly established

5. Mockito

Is it possible to mock some SUT business layer's modules, namely services, in order to remove dependencies and test these modules separately?

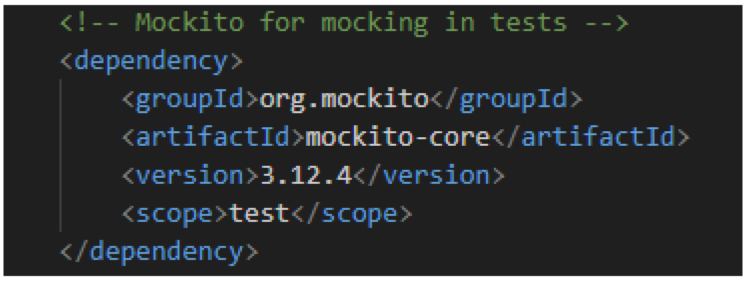
The answer to this question is no, there are a few reasons for that:

1. Services are implemented as enums (e.g., CustomerService.INSTANCE)
2. They have direct database dependencies through RowDataGateway classes
3. There's no interface abstraction or dependency injection

To make mocking possible, these things would need to be added/removed:

1. Create interfaces for services
2. Remove enum pattern
3. Implement dependency injection
4. Abstract database access through repositories

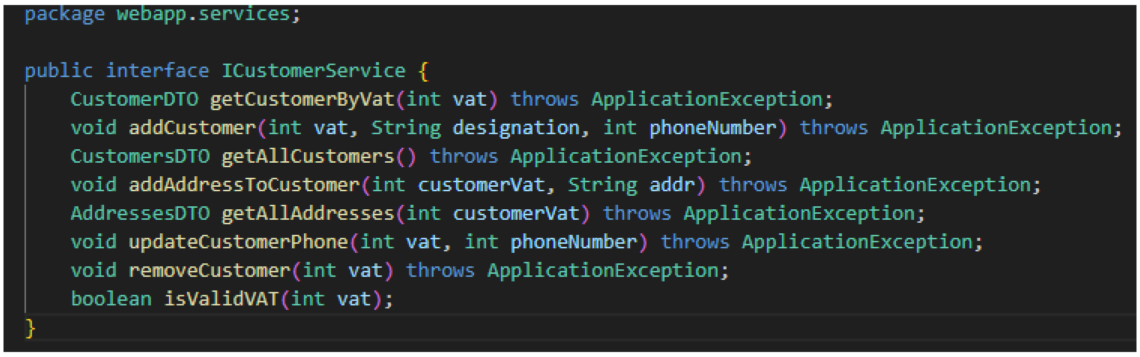
Before changing anything in the code with Mockito, we need to add the dependency in pom.xml:



```
<!-- Mockito for mocking in tests -->
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>3.12.4</version>
    <scope>test</scope>
</dependency>
```

So now to refactor as I said above we would need to do create interfaces for services, in my case, I'll do it for the CustomerService, then remove the enum patterns from CustomerService (it will be commented on the file because with these changes the other tests do not work), after that we implement dependency injection in AddCustomerPageController (basically remove the INSTANCEs and add a the constructors, and to finish it off we create a test for the controller.

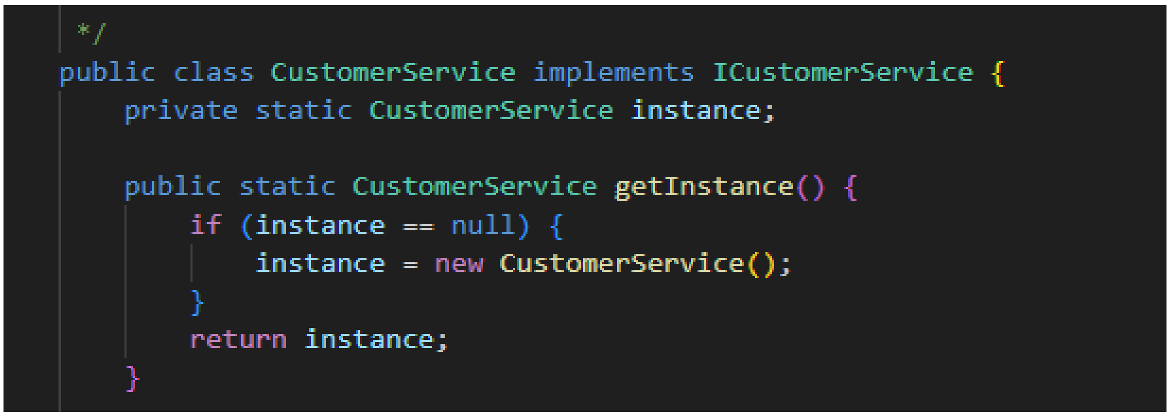
Heres the interface for the CustomerService:



```java
package webapp.services;

public interface ICustomerService {
    CustomerDTO getCustomerByVat(int vat) throws ApplicationException;
    void addCustomer(int vat, String designation, int phoneNumber) throws ApplicationException;
    CustomersDTO getAllCustomers() throws ApplicationException;
    void addAddressToCustomer(int customerVat, String addr) throws ApplicationException;
    AddressesDTO getAllAddresses(int customerVat) throws ApplicationException;
    void updateCustomerPhone(int vat, int phoneNumber) throws ApplicationException;
    void removeCustomer(int vat) throws ApplicationException;
    boolean isValidVAT(int vat);
}
```

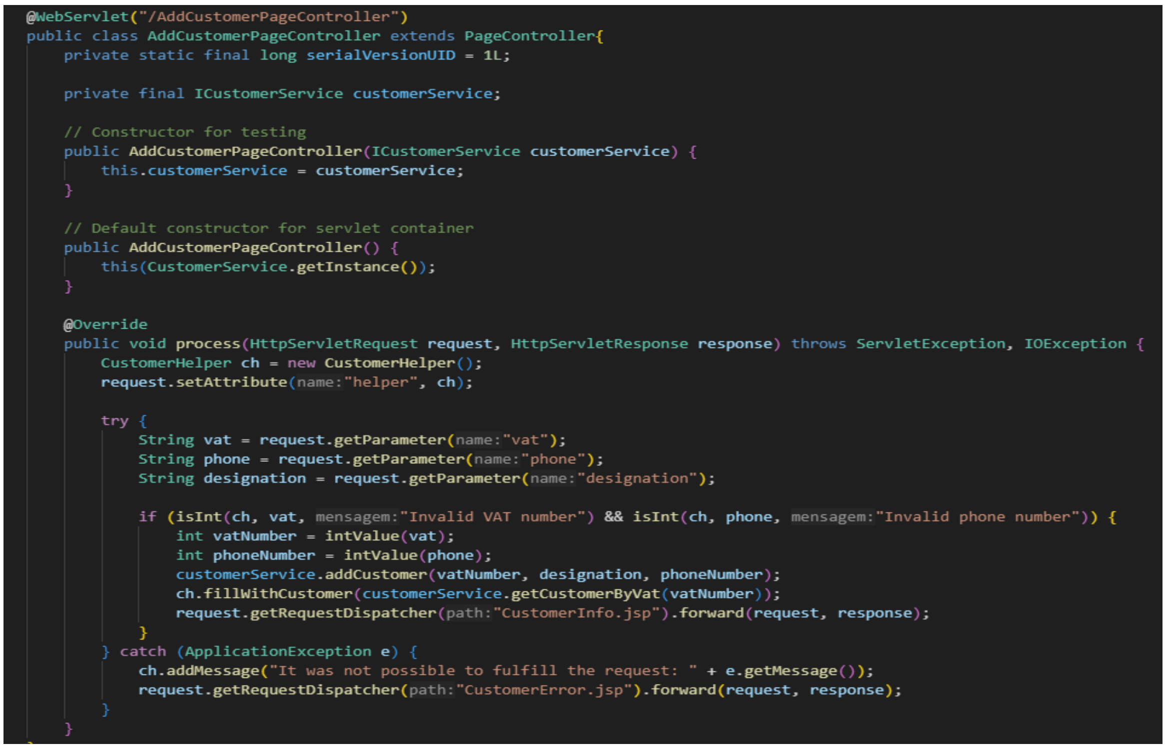
It has all the methods of the CustomerService, as it should.

Then in the CustomerService class we implement the interface and remove the Enum to replace it with a getInstance() method.



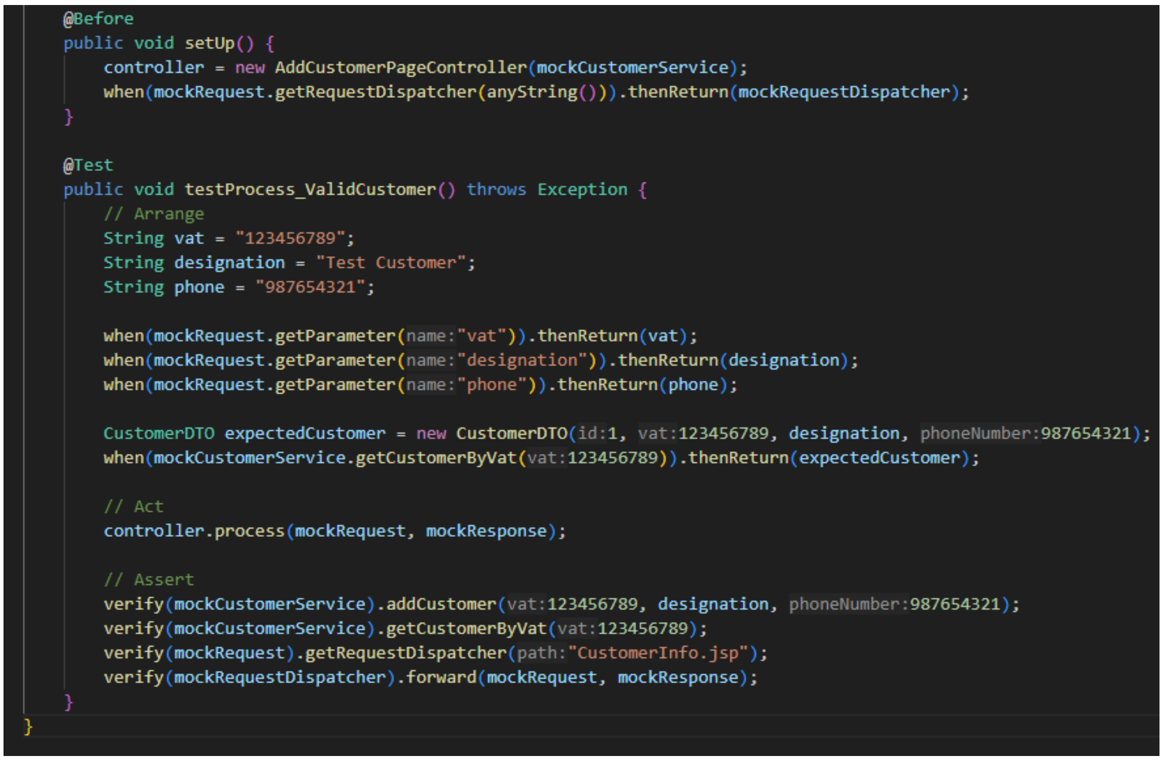
```java
 */
public class CustomerService implements ICustomerService {
    private static CustomerService instance;

    public static CustomerService getInstance() {
        if (instance == null) {
            instance = new CustomerService();
        }
        return instance;
    }
```

Now for the addCustomerPageController as I said before, we remove the CustomerService.INSTANCE and add two constructors, one for mocking with the interface and the default constructor.



```java
@WebServlet("/AddCustomerPageController")
public class AddCustomerPageController extends PageController{
    private static final long serialVersionUID = 1L;

    private final ICustomerService customerService;

    // Constructor for testing
    public AddCustomerPageController(ICustomerService customerService) {
        this.customerService = customerService;
    }

    // Default constructor for servlet container
    public AddCustomerPageController() {
        this(CustomerService.getInstance());
    }

    @Override
    public void process(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        CustomerHelper ch = new CustomerHelper();
        request.setAttribute(name:"helper", ch);

        try {
            String vat = request.getParameter(name:"vat");
            String phone = request.getParameter(name:"phone");
            String designation = request.getParameter(name:"designation");

            if (isInt(ch, vat, mensagem:"Invalid VAT number") && isInt(ch, phone, mensagem:"Invalid phone number")) {
                int vatNumber = intValue(vat);
                int phoneNumber = intValue(phone);
                customerService.addCustomer(vatNumber, designation, phoneNumber);
                ch.fillWithCustomer(customerService.getCustomerByVat(vatNumber));
                request.getRequestDispatcher(path:"CustomerInfo.jsp").forward(request, response);
            }
        } catch (ApplicationException e) {
            ch.addMessage("It was not possible to fulfill the request: " + e.getMessage());
            request.getRequestDispatcher(path:"CustomerError.jsp").forward(request, response);
        }
    }
```

To finish Mockito, we have the test, I decided to test a valid customer because we use a controller that only requires one service to make it easier for the creation of interfaces and such.



```java
    @Before
    public void setUp() {
        controller = new AddCustomerPageController(mockCustomerService);
        when(mockRequest.getRequestDispatcher(anyString())).thenReturn(mockRequestDispatcher);
    }

    @Test
    public void testProcess_ValidCustomer() throws Exception {
        // Arrange
        String vat = "123456789";
        String designation = "Test Customer";
        String phone = "987654321";

        when(mockRequest.getParameter(name:"vat")).thenReturn(vat);
        when(mockRequest.getParameter(name:"designation")).thenReturn(designation);
        when(mockRequest.getParameter(name:"phone")).thenReturn(phone);

        CustomerDTO expectedCustomer = new CustomerDTO(id:1, vat:123456789, designation, phoneNumber:987654321);
        when(mockCustomerService.getCustomerByVat(vat:123456789)).thenReturn(expectedCustomer);

        // Act
        controller.process(mockRequest, mockResponse);

        // Assert
        verify(mockCustomerService).addCustomer(vat:123456789, designation, phoneNumber:987654321);
        verify(mockCustomerService).getCustomerByVat(vat:123456789);
        verify(mockRequest).getRequestDispatcher(path:"CustomerInfo.jsp");
        verify(mockRequestDispatcher).forward(mockRequest, mockResponse);
    }
}
```

I did not test if these were done correctly because the rest of the project had several errors due to the changes made. I kept the changes done commented so the original code is still working, otherwise the HtmlUnit and DbSetup tests could fail.

6. Conclusion

To conclude this report, this project was very fun too but more importantly it gives us experience on different ways to test applications with tools like HtmlUnit and DbSetup. With these tools we managed to find a lot of bugs and ensured that everything else was consistent. We also saw that we can’t directly use Mockito in this application due to various limitations in the code.