



MINI SOFTWARE TESTING PROJECT



---

## Router management application

---

*Student : CHADEN BEN AMMAR*

December 4, 2023

# Contents

<b>List of Tables</b>	<b>3</b>
<b>List of Figures</b>	<b>4</b>
<b>1 Requirement Specification and Analysis.</b>	<b>8</b>
1.1 Introduction . . . . .	8
1.2 Requirement Specification . . . . .	8
1.2.1 Functional requirements . . . . .	8
1.2.2 Non-functional requirements . . . . .	8
1.3 Analysis of Requirements . . . . .	9
1.3.1 Use Case Diagram . . . . .	9
1.4 Working Environment . . . . .	9
1.4.1 Hardware Environment . . . . .	9
1.4.2 Technological Choices . . . . .	10
1.4.3 Software Environment . . . . .	11
1.5 Architecture . . . . .	12
1.6 Conclusion . . . . .	12
<b>2 Testing with Junit</b>	<b>13</b>
2.1 Introduction . . . . .	13
2.2 Testing . . . . .	13
2.2.1 Importance of Testing . . . . .	13
2.2.2 Testing Levels . . . . .	13
2.2.3 Unit Testing . . . . .	14
2.2.4 Comparison between Unit testing frameworks . . . . .	14
2.3 Junit . . . . .	14
2.3.1 JUnit Versions Comparison . . . . .	14
2.3.2 The Architecture of Junit . . . . .	15
2.3.3 JUnit Annotations . . . . .	15
2.3.4 How to setup the junit on vscode . . . . .	15
2.4 Conclusion . . . . .	17
<b>3 Implementation of the application</b>	<b>18</b>
3.1 Introduction . . . . .	18
3.2 CPE . . . . .	18
3.2.1 What is CPE management server . . . . .	18
3.2.2 Relation of the application with the CPE . . . . .	18

3.3	Interfaces of the application . . . . .	19
3.3.1	Get All routers . . . . .	19
3.3.2	Add router . . . . .	19
3.3.3	Get Router By serial number router . . . . .	20
3.3.4	Update router . . . . .	20
3.3.5	Delete router . . . . .	21
3.4	Conclusion . . . . .	21
<b>4</b>	<b>Testing the application</b>	<b>22</b>
4.1	Introduction . . . . .	22
4.2	Unit testing of the application . . . . .	22
4.2.1	USE case : Add router . . . . .	22
4.2.2	Use Case: Get Router . . . . .	24
4.2.3	Use Case: Update Router . . . . .	27
4.2.4	Use Case: Delete Router . . . . .	29
4.2.5	Use Case: Reboot Router . . . . .	31
4.2.6	Use Case: Reset Router . . . . .	33
4.2.7	Use Case: Connectivity . . . . .	35
4.3	Testing Allure Report . . . . .	37
4.4	Conclusion . . . . .	38
	<b>Bibliography</b>	<b>40</b>

# List of Tables

2.1	Language xUnit Frameworks . . . . .	14
2.2	Comparison of JUnit Versions . . . . .	14
2.3	JUnit Annotations . . . . .	15
4.1	Add router . . . . .	22
4.2	Get Router . . . . .	25
4.3	Update Router . . . . .	27
4.4	Delete Router . . . . .	29
4.5	Reboot Router . . . . .	32
4.6	Reset Router . . . . .	34
4.7	Connectivity . . . . .	36

# List of Figures

1.1	Use Case Diagram . . . . .	9
1.2	JavaFx logo . . . . .	10
1.3	Java logo . . . . .	10
1.4	MySQL logo . . . . .	11
1.5	Junit logo . . . . .	11
1.6	Vscode logo . . . . .	11
1.7	Allure Report logo . . . . .	11
1.8	Architecture of the application . . . . .	12
1.9	Layered Architecture of the application . . . . .	12
2.1	Extension pack for java . . . . .	16
2.2	Write the test code . . . . .	16
2.3	Activate the test . . . . .	16
2.4	Test the solution . . . . .	17
3.1	Router management app main interface . . . . .	19
3.2	Add router interface . . . . .	19
3.3	Show Details interface . . . . .	20
3.4	Update router interface . . . . .	20
3.5	Delete router interface . . . . .	21
4.1	Add router . . . . .	23
4.2	Add two routers with the same serial number . . . . .	23
4.3	Add router with a missing field . . . . .	23
4.4	Add router with null serial number . . . . .	24
4.5	Add router with wrong data type . . . . .	24
4.6	Get router by serial number . . . . .	25
4.7	Get router with null serial number . . . . .	25
4.8	Get router with null serial number . . . . .	26
4.9	Get router with null serial number . . . . .	26
4.10	Get router with wrong data type . . . . .	26
4.11	Update router . . . . .	27
4.12	Update non-existing router . . . . .	28
4.13	Update router with null data . . . . .	28
4.14	Update router with empty data . . . . .	28
4.15	Update router with wrong data type . . . . .	29
4.16	Delete existing router . . . . .	30

4.17 Delete non-existing router . . . . .	30
4.18 Delete router with null serial number . . . . .	30
4.19 Delete router with empty serial number . . . . .	31
4.20 Delete router with wrong serial number type . . . . .	31
4.21 Reboot existing router . . . . .	32
4.22 Reboot non-existing router . . . . .	32
4.23 Reboot router with null IP address . . . . .	33
4.24 Reboot router with invalid IP address . . . . .	33
4.25 Reset existing router . . . . .	34
4.26 Reset non-existing router . . . . .	34
4.27 Reset router with null IP address . . . . .	35
4.28 Reset router with invalid IP address . . . . .	35
4.29 Check connectivity of an existing router . . . . .	36
4.30 Check connectivity of a non-existing router . . . . .	36
4.31 Check connectivity of a router with null IP address . . . . .	37
4.32 Check connectivity of a router with wrong IP address type . . . . .	37
4.33 ALLURE REPORT Overview . . . . .	38
4.34 ALLURE REPORT Graphs . . . . .	38
4.35 ALLURE REPORT Suites . . . . .	38

# Abbreviations list

*CRUD*: Create, Read, Update, Delete

*CPE*: Customer Premises Equipment

*HCI*: Human-Computer Interface

*NFR* : non-functional requirements

# Introduction

In this mini-class project, we're crafting a Router Management application that not only encompasses router CRUD operations but also introduces us to the intricacies of the Customer Premises Equipment (CPE) management server that offers a remote Configuration and monitoring.

The Router Management App goes beyond being just a regular program, it serves as a gateway into the world of unit testing. Our primary objective in this project is not only to ensure that every part of the app works seamlessly but also to spotlight its capabilities in managing routers and CPE effectively for real-life situations.

As we progress, extensive testing using JUnit will be conducted, treating JUnit as a powerful tool to verify the robustness of our router management solution. However, before delving into testing, it's crucial to understand the fundamental operations of the application. This includes the comprehensive CRUD (Create, Read, Update, Delete) functionalities for routers and the strategic management of Customer Premises Equipment (CPE).

Furthermore, the application excels in remote configuration, offering administrators of internet service providers a streamlined way to manage routers and CPE from a distance. This project isn't solely about routers, it's a hands-on experience exploring JUnit testing in detail.

This report is organized into four chapters. The initial chapter covers Requirement Specification and Analysis, followed by a dedicated section on Testing with JUnit. The third chapter delves into the Implementation of the application, and the report concludes with the fourth chapter focusing on Testing the application.

# Chapter 1

# Requirement Specification and Analysis.

## 1.1 Introduction

This chapter introduces the first step in the implementation of our project. It is dedicated to specifying and Analysing the functional and non-functional requirements of the router management application. Furthermore, it describes the working environment (hardware and software) and the adopted architectures.

## 1.2 Requirement Specification

### 1.2.1 Functional requirements

The functional requirements are the features that the future system must achieve in order to satisfy the user the functionality that the user can do:

- **Add Router:** This feature allows a user to include a new router into the system.
- **Update Router:** A user can modify the information or settings of an existing router.
- **Delete Router:** This feature enables the deletion of a router from the system.
- **Get All Routers:** A user can retrieve a list of all routers currently registered in the system.
- **Get Router by Serial Number:** This functionality allows a user to find and retrieve specific router details using its serial number.
- **Reboot Router:** A user can initiate a restart of a router through this feature.
- **Reset Router:** This allows a user to restore a router to its default settings.
- **Connect Router:** A user can establish a connection to a router using this feature.

### 1.2.2 Non-functional requirements

The non-functional requirements (NFR) describe the requirements that are not directly related to the functionalities of the application but have an impact on the quality.

Performance: Speed, reliability, and the robustness of the service, coupled with good traceability, and, most importantly, without causing any application blockages.

Usability: The application must provide a clear, ergonomic, and intuitive interface that facilitates the use of various functionalities.

## 1.3 Analysis of Requirements

Analysis of Requirements In this section, we will present the various functionalities that our application must provide, and we will describe them using use case diagrams.

### 1.3.1 Use Case Diagram

Use case diagrams model the behavior of a system and enable the capturing of system requirements.

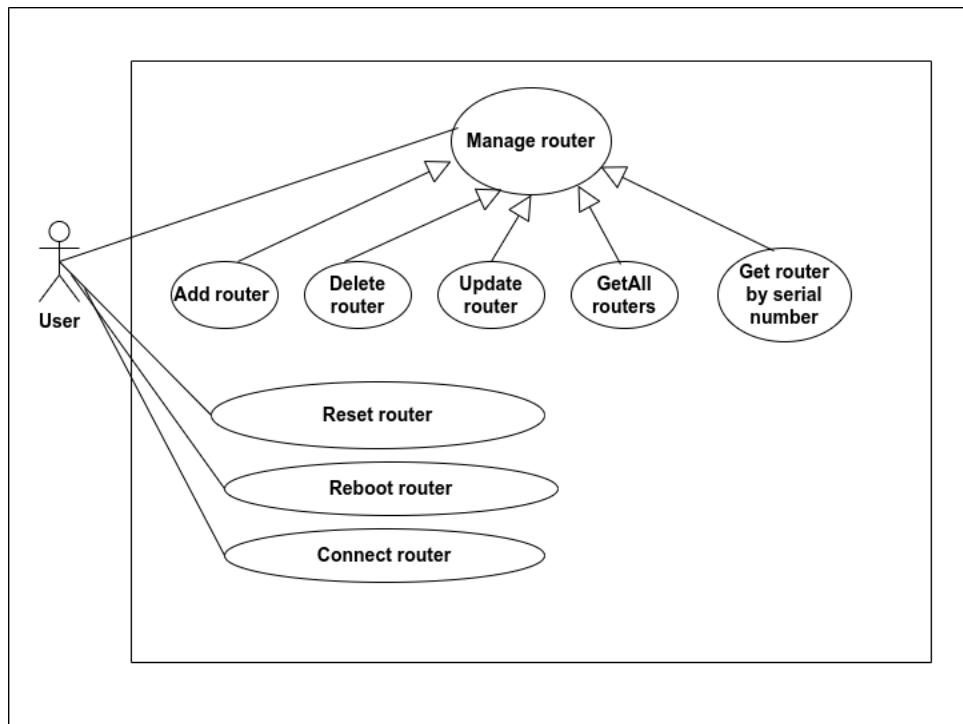


Figure 1.1: Use Case Diagram

PS: We are not talking about the end user here, we are actually referring to the administrator of the internet service provider.

## 1.4 Working Environment

In this section, we will provide a comprehensive description of the hardware environment, software environment, and all the tools used.

### 1.4.1 Hardware Environment

The work was carried out on laptops equipped with the specifications detailed in the table.

Processor	Intel CORE i5 1.8 GHz
Manufacturer	LENOVO
Hard Drive	1 TB
RAM	8 GB
Operating System	UBUNTU 22.04

#### 1.4.2 Technological Choices

In the upcoming section, we will present the technologies used for front-end, the Back-End (server-side), the database and the testing environment that we have selected for the development of our application.

##### Front-end:

Regarding the HCI of the application, we are using the JavaFx.

JavaFx [6] is a versatile Java-based platform for creating interactive user interfaces. Developed by Oracle, it simplifies desktop and enterprise application development with a modern programming environment. JavaFX boasts a robust scene graph, multimedia support, and a variety of UI controls, making it easy to design visually appealing applications. Its seamless integration with Java enhances development efficiency, making it a popular choice for building cross-platform applications with dynamic and user-friendly interfaces.



Figure 1.2: JavaFx logo

##### Server:

Regarding the server, we are using the Java [5] environment. Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.

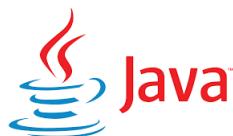


Figure 1.3: Java logo

##### Database:

MySQL [4] is an open-source relational database system using SQL for data management. It's known for speed, reliability, and ease of use, making it popular for web applications.



Figure 1.4: MySql logo

#### Testing:

JUnit [3] is a widely used open-source testing framework for Java that provides support for unit testing. It allows developers to write and run tests to verify that individual units or parts of their code are working correctly. JUnit follows the principles of test-driven development (TDD) and simplifies the process of automating the testing of Java applications. With JUnit, developers can easily create test cases, execute them, and assert expected outcomes, helping ensure the correctness and reliability of their code.



Figure 1.5: Junit logo

#### 1.4.3 Software Environment

To accomplish our work, we used the UBUNTU environment, along with the software systems. In the following, we provide a brief overview of these tools.

**VScode** [2] : this tool features a highly productive code editor that can be combined with programming language services, providing the power of an integrated development environment (IDE) and the speed of a text editor. It is used to create our application.



Figure 1.6: Vscode logo

**Allure Report** [8] : is an open-source framework for visualizing test results in formats compatible with testing frameworks like JUnit. It offers detailed and interactive reports with rich visualizations, simplifying result analysis. With a user-friendly interface and seamless integration, Allure Report enhances transparency in testing processes, providing valuable insights for informed decision-making.



Figure 1.7: Allure Report logo

## 1.5 Architecture

In the development of our application, we strategically employed a Layered Architecture to enhance its overall structure and maintainability. This architectural paradigm organizes the code into distinct layers, each serving a specific purpose. The data layer is dedicated to managing database-related operations, ensuring efficient storage and retrieval of information. The data access layer (persistence) handles the data access logic, providing a bridge between the data layer and the rest of the application. Then, the model layer encompasses the core application models, defining the structure of the data. Lastly, the HCI representing the interfaces of the application like it is represented in the following figure:

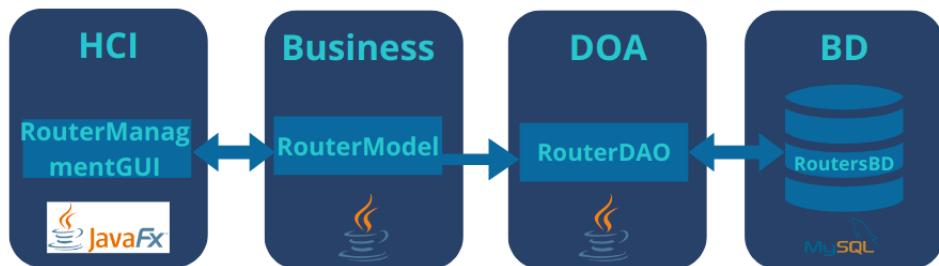


Figure 1.8: Architecture of the application

This separation of concerns allows for modular development, making it easier to manage and scale the application. Changes in one layer have minimal impact on others, promoting a flexible and adaptable structure. The Layered Architecture facilitates collaboration among developers, streamlines debugging and maintenance, and ultimately contributes to the efficiency and sustainability of the application over its life-cycle.

The following picture represents a screenshot of the project proving the use of a Layered architecture.

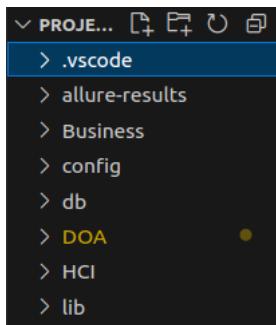


Figure 1.9: Layered Architecture of the application

## 1.6 Conclusion

This chapter introduces an initial phase of the project aimed at gaining a deeper understanding of the system. We have presented the requirement specifications, along with the working environment and the utilized architecture.

# Chapter 2

## Testing with Junit

### 2.1 Introduction

This chapter introduces the essentials of testing in software development, focusing on the utility of testing and the definition of JUnit, a key Java testing framework. It provides concise guidance on configuring and leveraging JUnit to ensure the reliability of our software. The chapter aims to equip readers with practical insights into effective testing within the project's scope.

### 2.2 Testing

#### 2.2.1 Importance of Testing

Testing is a critical element in software development, ensuring the quality and reliability of an application. It helps detect and rectify defects early, validates adherence to requirements, and contributes to code integrity. Ultimately, testing is essential for delivering a reliable and user-friendly software experience.

#### 2.2.2 Testing Levels

Test levels [1] are groups of testing activities that are organized and managed together. Each test level is an instance of the testing process, carried out in relation to the software at a specific development level, from individual units or components to complete systems.

- **Component Testing:** Component testing (also known as unit testing or module testing) focuses on testing components that can be tested separately.

- **Integration Testing:**

Integration testing focuses on the interactions between components or systems.

- **Systeme Testing:**

System testing focuses on the behavior and capabilities of an entire system or product, often considering end-to-end tasks that the system can perform and the non-functional behaviors it exhibits during the execution of these tasks.

- **Acceptance Testing:** Acceptance tests, like system tests, typically focus on the behavior and capabilities of an entire system or product.

### 2.2.3 Unit Testing

Component testing [1] (also known as unit testing or module testing) focuses on components that can be tested independently. The objectives of component testing are as follows:

- Reduce risk
- Verify if the functional and non-functional behaviors of the component align with their design and specifications
- Strengthen confidence in the quality of the component
- Identify defects in the component
- Prevent defects from progressing to higher levels of testing

### 2.2.4 Comparison between Unit testing frameworks

To write unit tests, we have access to frameworks that streamline the testing process. We only need to write the test classes, and the framework takes care of discovering, executing, and providing results or detected errors. The following table illustrates associations between certain languages and their respective xUnit-type frameworks:

Language	xUnit Framework
C	CUnit
Java	JUnit
MATLAB	mlUnit
PHP	PHPUnit

Table 2.1: Language xUnit Frameworks

## 2.3 Junit

JUnit is a mature framework designed to facilitate the writing and execution of automated tests.

### 2.3.1 JUnit Versions Comparison

The table below offers a concise overview of various JUnit versions, showcasing their release dates and key features. This quick reference allows developers to make informed decisions based on the evolving capabilities of each version, ensuring effective test automation for their projects.

Version	Release Date	Key Features
JUnit 3	2002	TestSuite, TestCase, Assert methods
JUnit 4	2006	Annotations, Parameterized Tests
JUnit 5	2017	Jupiter API, Extension Model

Table 2.2: Comparison of JUnit Versions

### 2.3.2 The Architecture of Junit

JUnit's fifth version [7] comprises three sub-projects:

- **JUnit Platform:** Offers an API allowing tools to discover and execute tests. It establishes an interface between JUnit and clients wishing to run tests (such as IDEs or build tools).
- **JUnit Jupiter:** Provides an API based on annotations for writing JUnit 5 unit tests and a TestEngine for their execution.
- **JUnit Vintage:** Introduces a TestEngine for running JUnit 3 and 4 tests, ensuring backward compatibility.
- The objective of this architecture is to separate the responsibilities of testing, execution, and extensions. It also aims to facilitate the integration of other testing frameworks into JUnit.

### 2.3.3 JUnit Annotations

JUnit provides annotations [7] to enhance the structure and behavior of test classes. The following table presents key JUnit annotations and their purposes.

Annotation	Description
@Test	Indicates that the method is a test method.
@BeforeEach	Denotes that the annotated method should be executed before each test method.
@AfterEach	Denotes that the annotated method should be executed after each test method.
@AfterAll	Indicates that the annotated method should be executed after all test methods in the class.
@BeforeAll	Indicates that the annotated method should be executed before all test methods in the class.

Table 2.3: JUnit Annotations

### 2.3.4 How to setup the junit on vscode

In this part we are going to walk through the process of setting up junit.

1. Install the plugin Extension pack for java

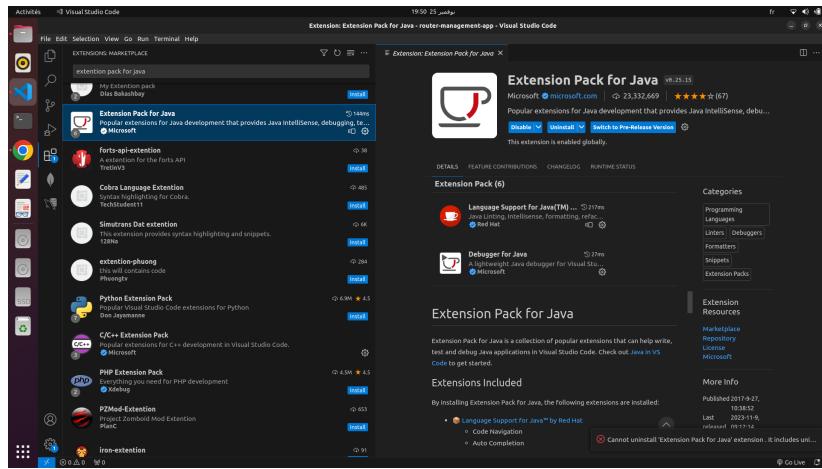


Figure 2.1: Extension pack for java

## 2. Write the test code

```

Calculator.java
=====
public class Calculatrice {
    public int additionner(int nombre1, int nombre2) {
        return nombre1 + nombre2;
    }
}

CalculatorTest.java
=====
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatriceTest {
    @Test
    public void testAdditionner() {
        // Arrange
        int nombre1 = 3;
        int nombre2 = 5;

        // Act
        int resultat = Calculatrice.additionner(nombre1, nombre2);

        // Assert
        assertEquals(8, resultat);
    }
}

```

Figure 2.2: Write the test code

## 3. Activate the test

Figure 2.3: Activate the test

#### 4. Test the solution

```
CalculatriceTest.java - June - Visual Studio Code
File Edit Selection View Go Run Terminal Help
TESTING
TEST EXPLORER
Filter (e.g. text, exclude, step)
V/I june 4ms
june 4ms
() <Default Package> 4ms
CalculatriceTest 4ms
TestAdditionner 1ms
public class CalculatriceTest {
    @Test
    public void testAdditionner() {
        // Arrange
        int nombre1 = 5;
        int nombre2 = 5;
        // Act
        int resultat = Calculatrice.additionner(nombre1, nombre2);
        // Assert
        assertEquals(expected:10, resultat);
    }
}
PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS
% TE S TC 1 , V2
% TS TTR E E 1 , testAdditionner( C alc ulatric eTest ) , false, 1 , false, -1 , testA d st ) ,
% TE S TS 1 , testAdditionner( C alc ulatric eTest )
% TE S TE 1 , testAdditionner( C alc ulatric eTest )
TEST RUN TIME 2 8
< TEST EXPLORER
✓ Test run at 11/25/2023, 8:16:39 AM
○ TestAdditionner()
○ Test run at 11/25/2023, 8:16:39 AM
○ Test run at 11/25/2023, 8:02:09 PM
○ Test run at 11/25/2023, 8:01:59 PM
○ Test run at 11/25/2023, 8:00:49 PM
Ln 16 Col 24 Spaces: 4 UTF-8 LF (.) Java Go Live Prettier
```

Figure 2.4: Test the solution

## 2.4 Conclusion

In this chapter we discovered what is testing and why it's important also the different type of test the junit environment and how to set it up.

# Chapter 3

# Implementation of the application

## 3.1 Introduction

This section, provide a brief overview of what CPE is and its significance in the context of your application. Also, we are going to discuss the implementation of our application by displaying the different interfaces.

## 3.2 CPE

### 3.2.1 What is CPE management server

A CPE management server plays a crucial role in managing and maintaining various Customer Premises Equipment (CPE) devices, typically found in homes, offices, or other user locations. Here some examples:

- **Remote Configuration:** It allows administrators or service providers to remotely configure settings on CPE devices. This includes tasks such as modifying network parameters, updating firmware, adjusting security settings, or changing operational features without physically accessing the devices.
- **Monitoring and Diagnostics:** The server enables continuous monitoring of CPE devices. It gathers data on device health, performance metrics, connectivity status, and other important information. This data is used for diagnostics, identifying issues, and proactively addressing potential problems.

### 3.2.2 Relation of the application with the CPE

Our application establishes a robust relationship with Customer Premises Equipment (CPE) through specialized functionalities, including remote reboot, reset, and connect. These features serve as pillars for Remote Configuration and Monitoring, enabling users to efficiently manage CPE configurations and conduct remote diagnostics. The remote reboot ensures seamless router operation, while the reset function provides a reliable means of restoring routers to default settings. The connect feature facilitates real-time monitoring and diagnostics of CPE health, contributing to a more responsive and efficient network infrastructure.

### 3.3 Interfaces of the application

#### 3.3.1 Get All routers

The "Get All Routers" interface is a streamlined gateway in our application, offering a quick overview of all routers. Simplifying access to essential details.

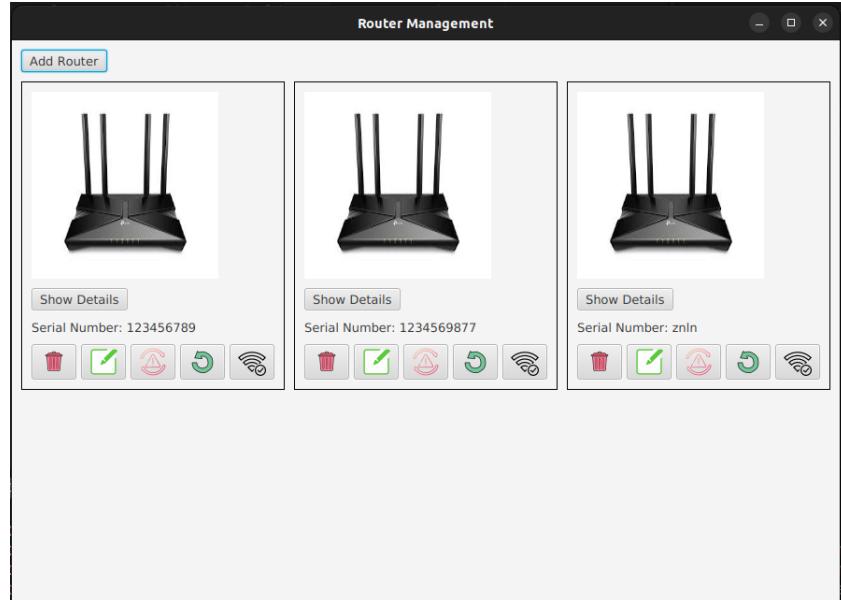


Figure 3.1: Router management app main interface

#### 3.3.2 Add router

The "Add Router" interface seamlessly integrates with the main application, providing users with a straightforward process to input essential details for new routers.

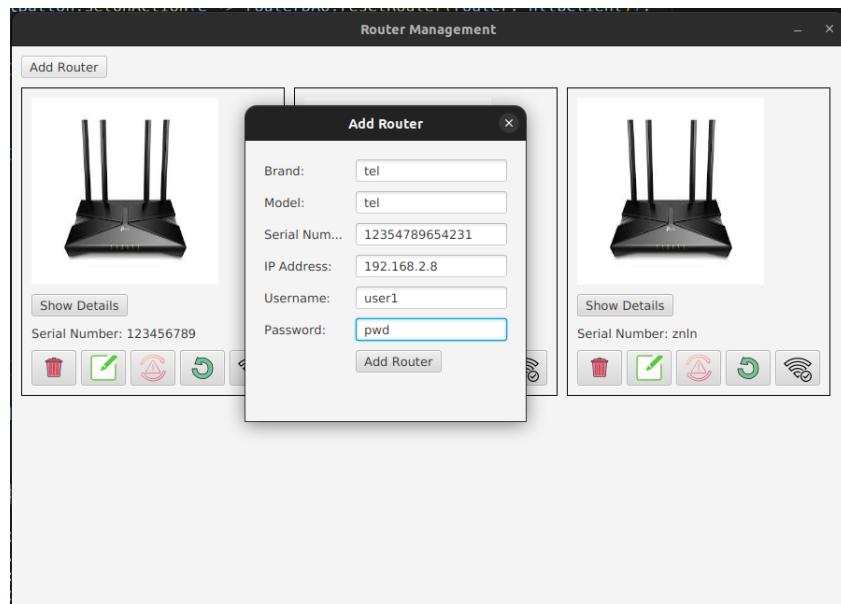


Figure 3.2: Add router interface

### 3.3.3 Get Router By serial number router

The "Show Details" button triggers the Get Router by Serial Number interface, where users can effortlessly access comprehensive information about a specific router.

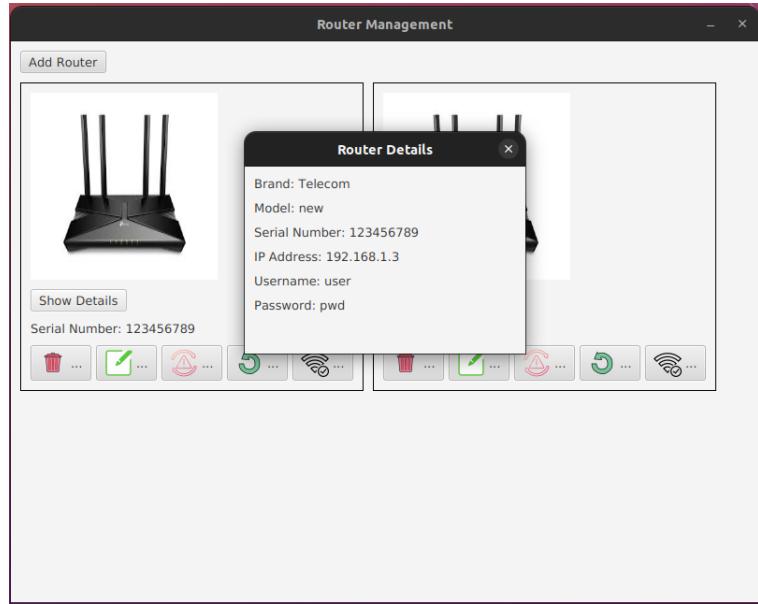


Figure 3.3: Show Details interface

### 3.3.4 Update router

The "Update Router" interface, accessed via a dedicated button, streamlines the modification of router configurations within our application.

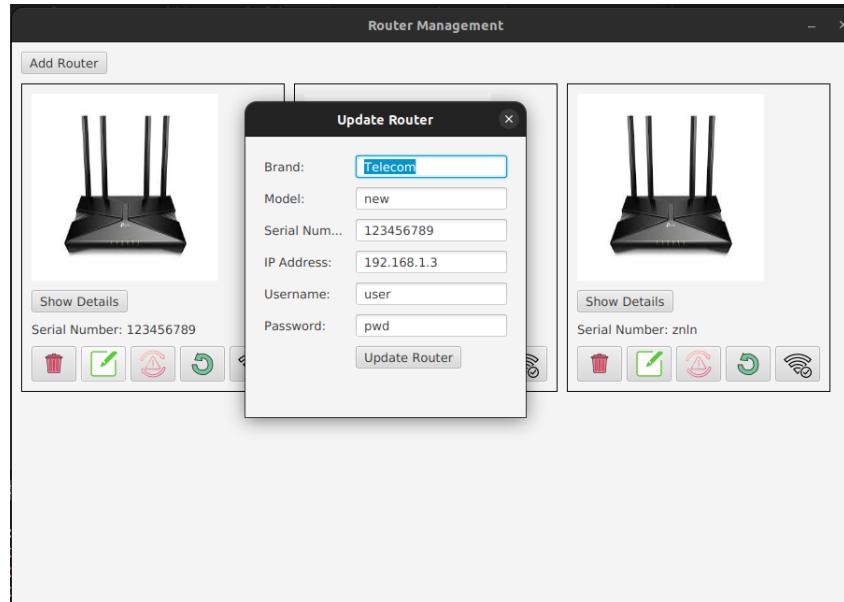


Figure 3.4: Update router interface

### 3.3.5 Delete router

Upon clicking the "Delete" button, users are prompted with a confirmation screen, offering two options: "OK" or "Cancel." This deliberate approach ensures a secure process when removing a router from the network. Users can confidently proceed with the deletion by selecting "OK" or opt to cancel the action by choosing "Cancel." This simple yet effective confirmation mechanism adds an additional layer of user control, preventing unintentional deletions and contributing to the overall safety and reliability of our network management system.

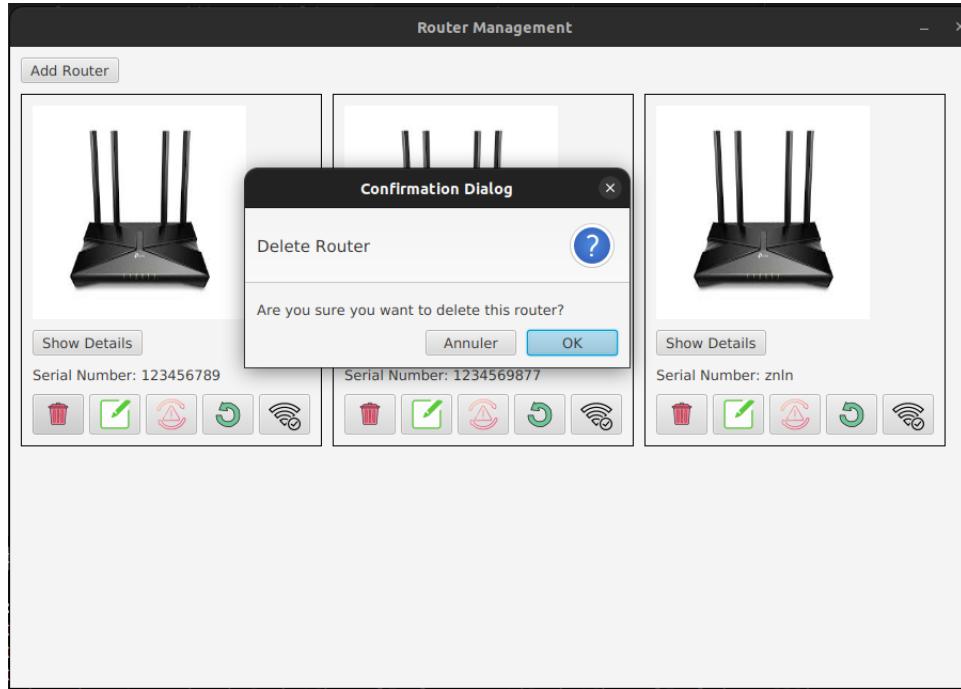


Figure 3.5: Delete router interface

## 3.4 Conclusion

In this chapter we have seen the different interfaces of the application. And in the next chapter will discuss the different test cases.

# Chapter 4

## Testing the application

### 4.1 Introduction

In this chapter we are going to be testing the different use cases of our application. And offering a test report.

### 4.2 Unit testing of the application

#### 4.2.1 USE case : Add router

In the table 4.1, we outline various test cases designed to evaluate the functionality of adding a new router. Each row represents a distinct test scenario, detailing the description of the test case, the provided test data, the observed result, and the expected outcome.

N	Description of the test cases	Test data	Result	Expected result
1	Add router	(“1”, “Model”, “192.168.1.2”, “brand”, “user”, “pwd”)	valid	valid
2	Add two routers with the same serial number	(“1”, “Model”, “192.168.1.2”, “brand”, “user”, “pwd”) (“1”, “Model”, “192.168.1.2”, “brand”, “user”, “pwd”)	invalid	invalid
3	Add router with a missing field	(“1”, “192.168.1.2”, “brand”, “user”, “pwd”)	invalid	invalid
4	Add router with null serial number	(Null, “192.168.1.2”, “brand”, “user”, “pwd”)	invalid	invalid
5	Add router with wrong data type	(1, “192.168.1.2”, “brand”, “user”, “pwd”)	Error	Error

Table 4.1: Add router

## 1. Add router

RouterDAOTest.java (Visual Studio Code)

```

private void dropTestTable() throws SQLException {
    try (Connection connection = DatabaseConnectionTest.getConnection();
         Statement statement = connection.createStatement()) {
        statement.execute("DROP TABLE routers");
    }
}

@Text
void testAddRouter() {
    RouterModel router = new RouterModel();
    router.setBrand("testBrand");
    router.setSerialNumber("testSerialNumber");
    router.setIpAddress("192.168.1.100");
    router.setUsername("testUser");
    router.setPassword("testPassword");
    router.setPwD("testPwD");

    RouterModel retrievedRouter = routerDAO.addRouter(router);
    assertEquals("testBrand", retrievedRouter.getBrand());
    assertEquals("testSerialNumber", retrievedRouter.getSerialNumber());
}

```

TEST RESULTS

- ✓ Test run at 11/28/2023, 10:20:49 PM
- testAddRouter()
- Test run at 11/28/2023, 10:19:09 PM
- Test run at 11/28/2023, 10:18:40 PM
- Test run at 11/28/2023, 10:18:35 PM
- Test run at 11/28/2023, 9:55:02 PM

RUNTIME1139

Figure 4.1: Add router

## 2. Add two routers with the same serial number

invalidTest.java (Visual Studio Code)

```

private void testAddRouterWithDuplicateSerialNumber() {
    RouterModel router1 = new RouterModel();
    router1.setBrand("testBrand1");
    router1.setSerialNumber("testSerialNumber1");
    router1.setIpAddress("192.168.1.100");
    router1.setUsername("testUser1");
    router1.setPassword("testPassword1");

    RouterModel router2 = new RouterModel();
    router2.setBrand("testBrand2");
    router2.setSerialNumber("testSerialNumber"); // Same serial number as router1
    router2.setIpAddress("192.168.1.101");
    router2.setUsername("testUser2");
    router2.setPassword("testPassword2");

    // Add the first router (should be successful)
    routerDAO.addRouter(router1);

    // Try to add the second router with the same serial number
    assertThrows(SQLException.class, () -> routerDAO.addRouter(router2)); org.junit.Assert.assertSame("org.junit.Assert.assertSame", "org.junit.Assert.assertSame");
}

```

TEST RESULTS

- ✓ Test run at 11/28/2023, 8:26:19 PM
- testAddRouterWithDuplicateSerialNumber()
- Test run at 11/28/2023, 11:15:54 PM

RUNTIME1993

Figure 4.2: Add two routers with the same serial number

## 3. Add router with a missing field

invalidTest.java (Visual Studio Code)

```

private void testAddRouterWithMissingField() {
    RouterModel router = new RouterModel();
    router.setBrand("testBrand");
    router.setSerialNumber("testSerialNumber");
    router.setIpAddress("192.168.1.100");
    router.setUsername("usernameNull"); // Missing required field

    // Try to add the router with a missing required field
    assertThrows(SQLException.class, () -> routerDAO.addRouter(router)); org.junit.Assert.assertSame("org.junit.Assert.assertSame", "org.junit.Assert.assertSame");
}

```

TEST RESULTS

- ✓ Test run at 11/28/2023, 8:27:57 PM
- testAddRouterWithMissingField()
- Test run at 11/28/2023, 6:26:19 PM
- Test run at 11/28/2023, 11:15:54 PM

RUNTIME945

Figure 4.3: Add router with a missing field

#### 4. Add router with null serial number

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like RouterModel.java, RouterDAOTest.java, and InvalidTest.java.
- Code Editor:** Displays the `InvalidTest.java` file containing Java test code. The code attempts to add a router with a missing serial number and asserts that a `SQLException` is thrown.
- Terminal:** Shows the command `mvn test` being run.
- Output:** Shows the test results with one failure: `java.lang.AssertionError: expected [null] but found null`.
- Test Explorer:** Shows no test results yet.

Figure 4.4: Add router with null serial number

#### 5. Add router with wrong data type

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like RouterModel.java, RouterDAOTest.java, and InvalidTest.java.
- Code Editor:** Displays the `InvalidTest.java` file containing Java test code. The code attempts to add a router with a serial number set to an integer instead of a string and asserts that a `SQLException` is thrown.
- Terminal:** Shows the command `mvn test` being run.
- Output:** Shows the test results with one failure: `java.lang.AssertionError: expected [null] but found null`.
- Test Explorer:** Shows no test results yet.

Figure 4.5: Add router with wrong data type

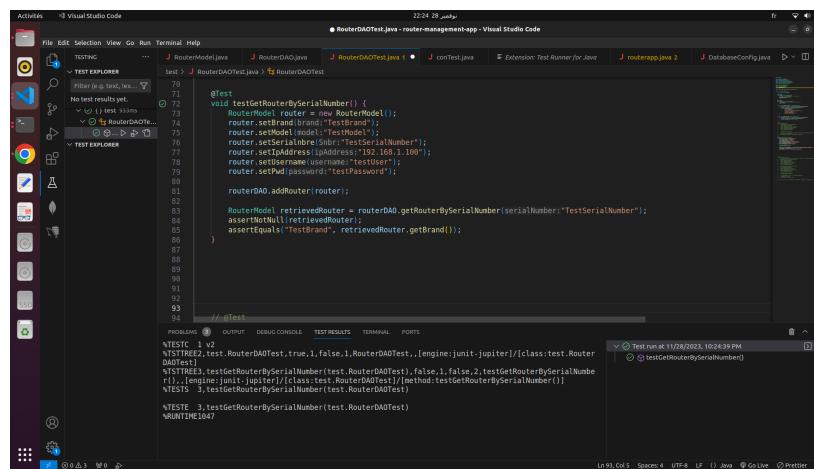
#### 4.2.2 Use Case: Get Router

In the following table (Table 4.2), we present various test cases designed to assess the functionality of retrieving router information. Each test scenario, outlined in a separate row, includes a detailed description, the corresponding test data, the observed result, and the expected outcome.

N	Description of the test cases	Test data	Result	Expected result
1	Get router by serial number	("1")	valid	Retrieve router information
2	Get router with non-existing serial number	("999")	invalid	Router not found
3	Get router with null serial number	(Null)	invalid	Invalid input
4	Get router with empty serial number	("")	invalid	Invalid input
5	Get router with wrong data type	(1l)	Error	Data type mismatch

Table 4.2: Get Router

### 1. Get router by serial number



```

    @Test
    void testGetRouterBySerialNumber() {
        RouterModel router = new RouterModel();
        router.setBrand("TestBrand");
        router.setModel("TestModel");
        router.setIpAddress("192.168.1.100");
        router.setUsername("testUser");
        router.setPassword("testPassword");

        routerDAO.addRouter(router);

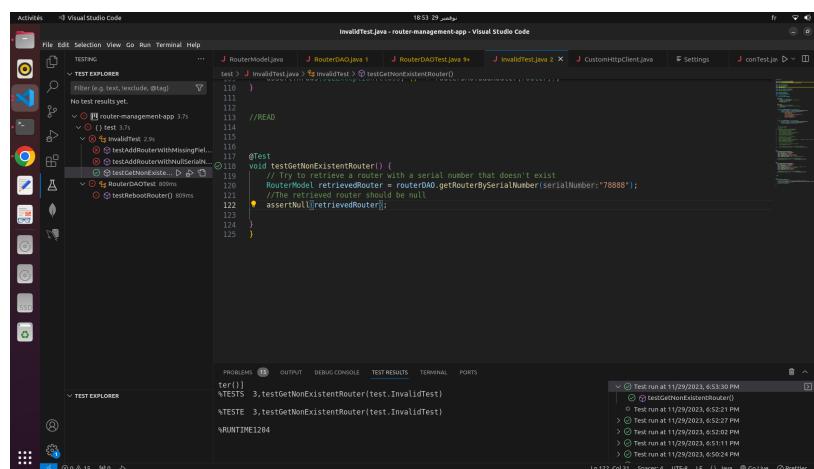
        RouterModel retrievedRouter = routerDAO.getRouterBySerialNumber(serialNumber:"TestSerialNumber");
        assertNotBeNull(retrievedRouter);
        assertEquals("TestBrand", retrievedRouter.getBrand());
    }

    // @Test
    void testGetNonexistentRouter() {
        RouterModel retrievedRouter = routerDAO.getRouterBySerialNumber(serialNumber:"78888");
        assertNull(retrievedRouter);
    }
}

```

Figure 4.6: Get router by serial number

### 2. Get router with non-existing serial number



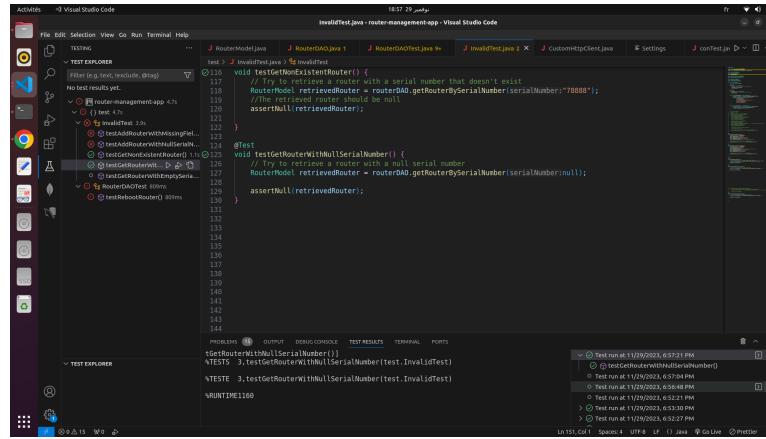
```

    @Test
    void testGetNonexistentRouter() {
        RouterModel retrievedRouter = routerDAO.getRouterBySerialNumber(serialNumber:"78888");
        assertNull(retrievedRouter);
    }
}

```

Figure 4.7: Get router with null serial number

### 3. Get router with null serial number

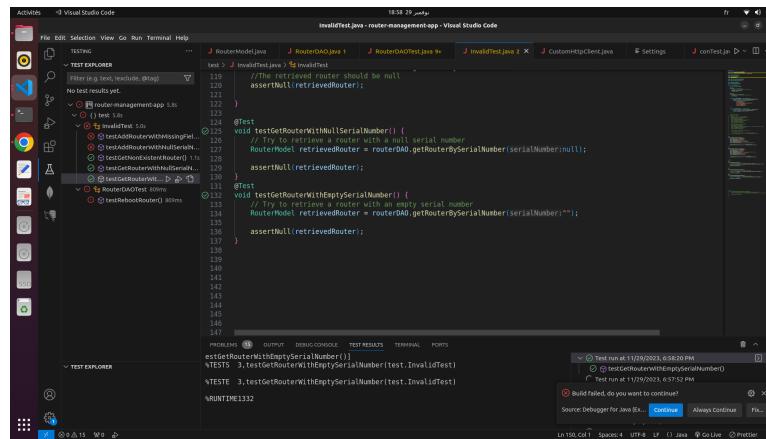


The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure for "router-management-app".
- Code Editor:** Displays the file `invalidTest.java` containing Java test code. The code includes methods like `testGetRouterWithNullSerialNumber()` and `testGetRouterWithEmptySerialNumber()`.
- Test Explorer:** Shows no test results yet.
- Output:** Shows the command `mvn test` running, with output indicating successful test runs.
- Terminal:** Shows the command `mvn test` being run.

Figure 4.8: Get router with null serial number

### 4. Get router with empty serial number

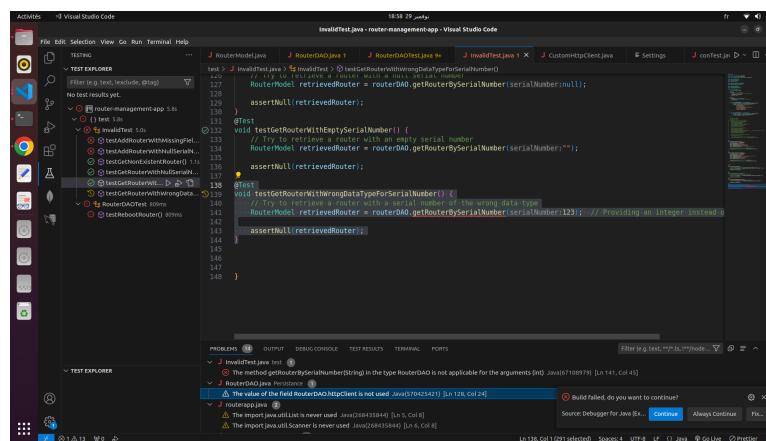


The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure for "router-management-app".
- Code Editor:** Displays the file `invalidTest.java` containing Java test code. The code includes methods like `testGetRouterWithEmptySerialNumber()`.
- Test Explorer:** Shows no test results yet.
- Output:** Shows the command `mvn test` running, with output indicating successful test runs.
- Terminal:** Shows the command `mvn test` being run.

Figure 4.9: Get router with empty serial number

### 5. Get router with wrong data type



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure for "router-management-app".
- Code Editor:** Displays the file `invalidTest.java` containing Java test code. The code includes methods like `testGetRouterWithWrongDataTypeForSerialNumber()`.
- Test Explorer:** Shows no test results yet.
- Output:** Shows the command `mvn test` running, with output indicating successful test runs.
- Terminal:** Shows the command `mvn test` being run.

Figure 4.10: Get router with wrong data type

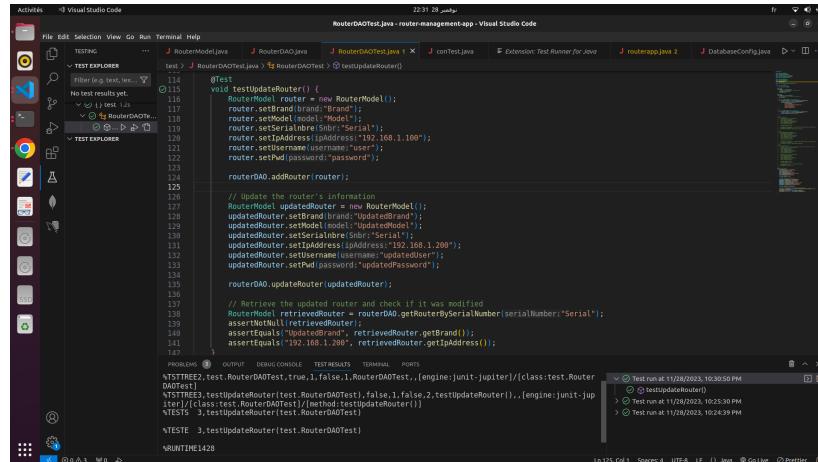
### 4.2.3 Use Case: Update Router

In the following table (Table 4.3), we present various test cases designed to assess the functionality of updating router information. Each test scenario, outlined in a separate row, includes a detailed description, the corresponding test data, the observed result, and the expected outcome.

N	Description of the test cases	Test data	Result	Expected result
1	Update router information successfully	(Router data)	Valid	Router information updated successfully
2	Update non-existing router	(Router data)	Invalid	Router not found
3	Update router with null data	(Null)	Invalid	Invalid input
4	Update router with empty data	("")	Invalid	Invalid input
5	Update router with wrong data type	(Wrong data type)	Error	Data type mismatch

Table 4.3: Update Router

#### 1. Update router



The screenshot shows a Visual Studio Code interface with several tabs open. The active tab is 'RouterDAOTest.java' which contains Java test code for updating a router. The code uses JUnit annotations like @Test and @Before. It creates a new RouterModel object, sets its properties (brand, model, serial number, IP address, user name, and password), adds it to a RouterDAO object, and then updates the router's IP address. After the update, it retrieves the router again and checks if its IP address has been modified. The test passes. Below the code editor, the terminal shows the command 'mvn clean test' and the output of the test run, which includes the test name 'testUpdateRouter' and its status as successful.

```

    Activities -> Visual Studio Code
    File Edit Selection View Go Run Terminal Help
    TESTING ... J RouterModel.java J RouterDAO.java J RouterDAOTest.java X J context.java Extension: Test Runner for Java J routesapp.java 2 J DatabaseConfig.java
    No test results yet.
    < J RouterDAOTest.java > J RouterDAOTest > J testUpdateRouter()
    115     @Test
    116     void testUpdateRouter() {
    117         RouterModel router = new RouterModel();
    118         router.setBrand("Model");
    119         router.setModel("Model");
    120         router.setSerialNumber("Serial");
    121         router.setIpAddress("192.168.1.100");
    122         router.setUsername("User");
    123         router.setPassword("password");
    124
    125         routerDAO.addRouter(router);
    126
    127         // Update the router's information
    128         RouterModel updatedRouter = new RouterModel();
    129         updatedRouter.setBrand("UpdatedBrand");
    130         updatedRouter.setModel("UpdatedModel");
    131         updatedRouter.setSerialNumber("UpdatedSerial");
    132         updatedRouter.setIpAddress("192.168.1.200");
    133         updatedRouter.setUsername("updatedUser");
    134         updatedRouter.setPassword("updatedPassword");
    135
    136         routerDAO.updateRouter(updatedRouter);
    137
    138         // Retrieve the updated router and check if it was modified
    139         RouterModel retrievedRouter = routerDAO.getRouterBySerialNumber("Serial");
    140         assertNotEquals(null, retrievedRouter);
    141         assertEquals("UpdatedBrand", retrievedRouter.getBrand());
    142         assertEquals("192.168.1.200", retrievedRouter.getIpAddress());
    143     }
    144 }

    M3T3R22.test.RouterDAOTest, true, 1, false, 1, RouterDAOTest., {engine:junit-jupiter}/[class:test.RouterDAOTest]
    M3T3R3,testUpdateRouter(test.RouterDAOTest, false, 1, false, 2, testUpdateRouter(), {engine:junit-jupiter}, RouterDAOTest)
    M3T3R5,testUpdateRouter(test.RoutingDAOTest)
    M3T3S 3,testUpdateRouter(test.RoutingDAOTest)

    TESTE 3,testUpdateRouter(test.RoutingDAOTest)

```

Figure 4.11: Update router

## 2. Update non-existing router

```

    void testUpdateNonExistRouter() {
        if (routerDAO.getRouterBySerialNumber("RVM") == null) {
            RouterModel updatedRouter = new RouterModel();
            updatedRouter.setBrand("UpdatedBrand");
            updatedRouter.setModel("UpdatedModel");
            updatedRouter.setSerialNumber("RVM");
            updatedRouter.setIpAddress("192.168.1.100");
            updatedRouter.setPassword("updatedPassword");
            updatedRouter.setHttpPort("8080");
        }
    }

    @Test
    void testUpdateNonExistRouter() {
        RouterModel retrievedRouter = routerDAO.getRouterBySerialNumber("RVM");
        assertEquals(null, retrievedRouter);
    }
}

```

Figure 4.12: Update non-existing router

## 3. Update router with null data

```

    void testUpdateRouterWithNullSerialNumber() {
        RouterModel router = new RouterModel();
        router.setSerialNumber(null);
        router.setIpAddress("192.168.1.100");
        router.setUsername("user");
        router.setPassword("password");
        router.setHttpPort("8080");
        router.setBrand("UpdatedBrand");
        router.setModel("UpdatedModel");
        router.setSerialNumber("RVM");
        router.setIpAddress("192.168.1.100");
        router.setUsername("user");
        router.setPassword("updatedPassword");
        routerDAO.updateRouter(router);
    }

    @Test
    void testUpdateRouterWithNullSerialNumber() {
        RouterModel originalRouter = routerDAO.getRouterBySerialNumber("Serial");
        assertEquals("user", originalRouter.getUsername());
        assertEquals("password", originalRouter.getPassword());
        assertEquals("UpdatedBrand", originalRouter.getBrand());
        assertEquals("UpdatedModel", originalRouter.getModel());
        assertEquals("RVM", originalRouter.getSerialNumber());
        assertEquals("192.168.1.100", originalRouter.getIpAddress());
        assertEquals("8080", originalRouter.getHttpPort());
    }
}

```

Figure 4.13: Update router with null data

## 4. Update router with empty data

```

    void testUpdateRouterWithEmptySerialNumber() {
        RouterModel updatedRouter = new RouterModel();
        updatedRouter.setSerialNumber("");
        updatedRouter.setIpAddress("192.168.1.100");
        updatedRouter.setHttpPort("8080");
        updatedRouter.setBrand("UpdatedBrand");
        updatedRouter.setModel("UpdatedModel");
        updatedRouter.setSerialNumber("RVM");
        updatedRouter.setIpAddress("192.168.1.100");
        updatedRouter.setUsername("user");
        updatedRouter.setPassword("password");
        routerDAO.updateRouter(updatedRouter);
    }

    @Test
    void testUpdateRouterWithEmptySerialNumber() {
        RouterModel originalRouter = routerDAO.getRouterBySerialNumber("Serial");
        assertEquals("user", originalRouter.getUsername());
        assertEquals("password", originalRouter.getPassword());
        assertEquals("UpdatedBrand", originalRouter.getBrand());
        assertEquals("UpdatedModel", originalRouter.getModel());
        assertEquals("RVM", originalRouter.getSerialNumber());
        assertEquals("192.168.1.100", originalRouter.getIpAddress());
        assertEquals("8080", originalRouter.getHttpPort());
    }
}

```

Figure 4.14: Update router with empty data

## 5. Update router with wrong data type

```

    test > J InvalidTest.java > J RouterModel.java > J RouterDAO.java > J RouterDAOTest.java > J testUpdateRouterWithInvalidDataType()
    230     }
    231     }
    232     @Test
    233     void testUpdateRouterWithInvalidDataType() {
    234         RouterModel router = new RouterModel();
    235         router.setBrand("Brand");
    236         router.setModel("Model");
    237         router.setSerialnbr("Snbr:Serial");
    238         router.setIpAddress("ipaddress:192.168.1.108");
    239         router.setPw("password:user");
    240         router.setPwd("password");
    241         routerDAO.addRouter(router);
    242     }
    243     RouterModel updatedRouter = new RouterModel();
    244     updatedRouter.setBrand("UpdatedBrand");
    245     updatedRouter.setModel("UpdatedModel");
    246     updatedRouter.setSerialnbr("Snbr:1122");
    247     updatedRouter.setIpAddress("ipaddress:192.168.1.109");
    248     updatedRouter.setPw("password:User");
    249     updatedRouter.setPwd("password:User");
    250     routerDAO.updateRouter(updatedRouter);
    251 }

    Meter.RouterModel a router with an invalid IP address
    RouterModel updatedRouter = new RouterModel();
    updatedRouter.setBrand("UpdatedBrand");
    updatedRouter.setModel("UpdatedModel");
    updatedRouter.setSerialnbr("Snbr:1122");
    updatedRouter.setIpAddress("ipaddress:192.168.1.109");
    updatedRouter.setPw("password:User");
    updatedRouter.setPwd("password:User");
    routerDAO.updateRouter(updatedRouter);

    PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS Filter (e.g. text: "maven")
    J InvalidTest.java test
    J RouterDAOTest.java test
    J RouterDAO.java Persistence
    △ The value of the field RouterDAO.httpClient is not used [java:1704154241] [Ln 128, Col 24]
    J RouterDAOTest.java test
    J RouterDAO.java Persistence
    △ The import java.util.List is never used [jav:268435844] [Ln 5, Col 8]
    △ The import java.util.Scanner is never used [jav:268435844] [Ln 6, Col 8]
    J RouterDAOTest.java test
    J RouterDAO.java Persistence
    △ The import java.net.HttpURLConnection is never used [jav:268435844] [Ln 15, Col 8]
    △ The import java.net.HttpURLConnection is never used [jav:268435844] [Ln 11, Col 8]
    △ The import java.net.HttpURLConnection is never used [jav:268435844] [Ln 15, Col 8]
    △ The import java.net.HttpURLConnection is never used [jav:268435844] [Ln 17, Col 8]
    △ The import java.util.List is never used [jav:268435844] [Ln 17, Col 8]

    Build failed, do you want to continue? Continue Always Continue Fix...
    Source: Debugger for Java (x) Lint Java Go Live Prettier
    Ln 128, Col 1 Spaces: 4 UTF-8 (L) Java Go Live Prettier
  
```

Figure 4.15: Update router with wrong data type

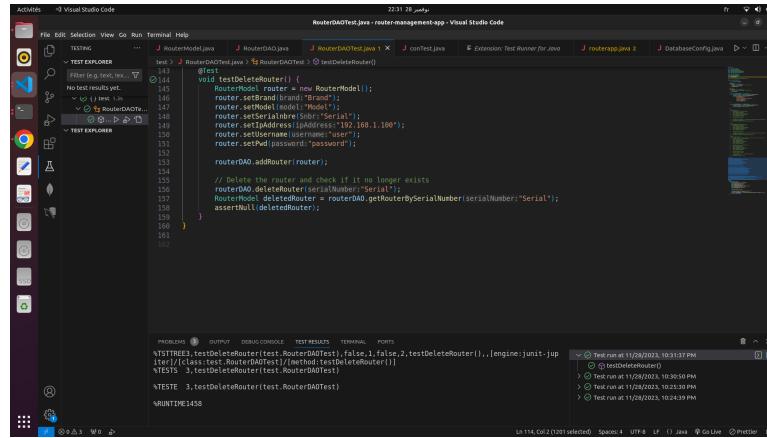
### 4.2.4 Use Case: Delete Router

In the following table (Table 4.4), we present various test cases designed to assess the functionality of deleting router information. Each test scenario, outlined in a separate row, includes a detailed description, the corresponding test data, the observed result, and the expected outcome.

N	Description of the test cases	Test data	Result	Expected result
1	Delete existing router	("1")	Valid	Router deleted successfully
2	Delete non-existing router	("999")	Invalid	Router not found
3	Delete router with null serial number	(Null)	Invalid	Invalid input
4	Delete router with empty serial number	("")	Invalid	Invalid input
5	Delete router with wrong serial number type	(Wrong data type)	Error	Data type mismatch

Table 4.4: Delete Router

## 1. Delete existing router



The screenshot shows the Visual Studio Code interface with the file `RouterDAOTest.java` open. The code contains a test method `testDeleteRouter` that creates a new router model, adds it to a DAO, and then attempts to delete it. It checks if the router is deleted by verifying its serial number.

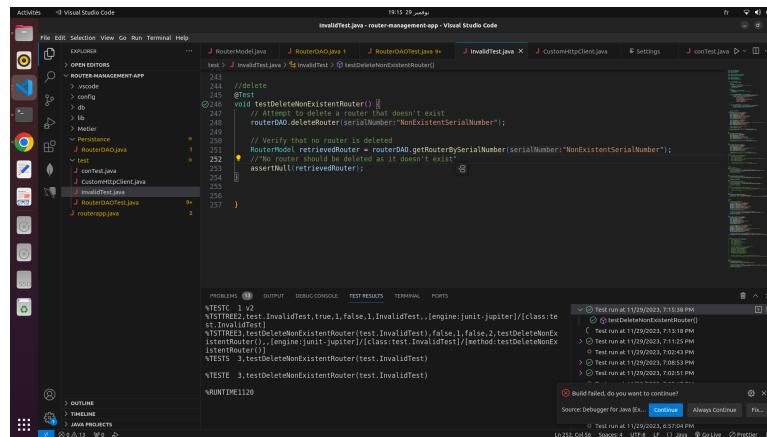
```
    void testDeleteRouter() {
        RouterModel router = new RouterModel();
        router.setBrand("Brand");
        router.setModel("Model");
        router.setSerialNumber("Serial");
        router.setIpAddress("ipAddress");
        router.setUsername("username");
        router.setPassword("password");

        routerDAO.addRouter(router);

        // Delete the router and check if it no longer exists
        routerDAO.deleteRouter(serialNumber);
        RouterModel retrievedRouter = routerDAO.getRouterBySerialNumber(serialNumber);
        assertNull(retrievedRouter);
    }
}
```

Figure 4.16: Delete existing router

## 2. Delete non-existing router

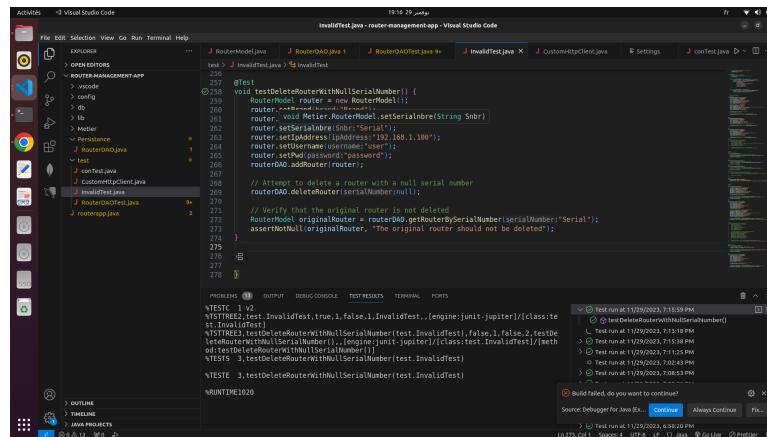


The screenshot shows the Visual Studio Code interface with the file `invalidTest.java` open. The code contains a test method `testDeleteNonexistentRouter` that attempts to delete a router with a non-existent serial number. It then checks if the router still exists.

```
    void testDeleteNonexistentRouter() {
        // Attempt to delete a router that doesn't exist
        routerDAO.deleteRouter(serialNumber);
    }
}
```

Figure 4.17: Delete non-existing router

## 3. Delete router with null serial number



The screenshot shows the Visual Studio Code interface with the file `invalidTest.java` open. The code contains a test method `testDeleteRouterWithNullSerialNumber` that attempts to delete a router with a null serial number. It then checks if the router still exists.

```
    void testDeleteRouterWithNullSerialNumber() {
        RouterModel router = new RouterModel();
        router.setBrand("Brand");
        router.setModel("Model");
        router.setSerialNumber(null);
        router.setIpAddress("ipAddress");
        router.setUsername("username");
        router.setPassword("password");

        routerDAO.addRouter(router);

        // Attempt to delete a router with a null serial number
        routerDAO.deleteRouter(null);

        // Verify that the original router is not deleted
        RouterModel originalRouter = routerDAO.getRouterBySerialNumber(serialNumber);
        assertNull(originalRouter, "The original router should not be deleted");
    }
}
```

Figure 4.18: Delete router with null serial number

#### 4. Delete with empty serial number

```

    test > J InvalidTest.java J RouterModel.java J RouterDAO.java J InvalidTest.java J RouterDAOTest.java J RouterTest.java J CustomHttpClient.java J context.java J RouterApp.java J RouterModelTest.java J RouterApp.java

    test 1 RouterModelTest.java void testDeleteRouterWithEmptySerialNumber() {
        RouterModel router = RouterModel.getRouter();
        router.setSerialNumber("serial");
        router.setBrand("Brand");
        router.setModel("Model");
        router.setIpAddress("192.168.1.100");
        router.setIpAddress("192.168.1.100");
        router.setUsername("user");
        router.setPassword("password");
        routerDAO.addRouter(router);
    }

    test 2 // Attempt to delete a router with an invalid serial number
    routerDAO.deleteRouter("serial");

    test 3 Verify that the original router is not deleted
    RouterModel originalRouter = RouterDAO.getRouterBySerialNumber("serial");
    assertNotNull(originalRouter, "The original router should not be deleted");
}

```

Figure 4.19: Delete router with empty serial number

#### 5. Delete router with wrong serial number type

```

    test > J InvalidTest.java J RouterModel.java J RouterDAO.java J InvalidTest.java J RouterDAOTest.java J RouterTest.java J CustomHttpClient.java J context.java J RouterApp.java J RouterModelTest.java J RouterApp.java

    test 1 RouterModelTest.java void testDeleteRouterWithInvalidSerialNumberType() {
        RouterModel router = new RouterModel();
        router.setBrand("Brand");
        router.setModel("Model");
        router.setIpAddress("192.168.1.100");
        router.setIpAddress("192.168.1.100");
        router.setUsername("user");
        router.setPassword("password");
        routerDAO.addRouter(router);
    }

    test 2 // Attempt to delete a router with an invalid serial number
    routerDAO.deleteRouter((byte[]) null);

    test 3 Verify that the original router is not deleted
    RouterModel originalRouter = RouterDAO.getRouterBySerialNumber("serial");
    assertNotNull(originalRouter, "The original router should not be deleted");
}

```

Figure 4.20: Delete router with wrong serial number type

#### 4.2.5 Use Case: Reboot Router

In the table (Table 4.5), various test cases are presented to evaluate the functionality of rebooting a router. Each test scenario, outlined in a separate row, includes a detailed description, the corresponding test data, the observed result, and the expected outcome.

##### 1. Reboot existing router

N	Description of the test cases	Test data	Result	Expected result
1	Reboot existing router	("192.168.1.2")	valid	Router successfully rebooted
2	Reboot non-existing router	("192.168.1.5")	invalid	Router not found
3	Reboot router with null IP address	(Null)	invalid	Invalid input
4	Reboot router with invalid IP address	(555)	error	Data type mismatch

Table 4.5: Reboot Router

```

    void testRebootRouter() throws IOException, InterruptedException {
        // Set up and start the embedded HTTP server
        int port = 9999;
        HttpServer httpServer = HttpServer.create(new InetSocketAddress(port), 0);
        httpServer.createContext("/reboot", new RebootHandler());
        Executors.newSingleThreadExecutor().submit(() -> httpServer.start());
        // Create RouterDAO with the actual HttpClient
        RouterDAO routerDAO = new RouterDAO();
        // Create a RouterModel for testing
        RouterModel router = new RouterModel();
        router.setIpAddress("http://localhost:" + port);
        // Perform the reboot operation
        routerDAO.rebootRouter(router);
        // Stop the embedded HTTP server
        httpServer.stop(0);
        // Handler to simulate the /reboot endpoint
        class RebootHandler implements HttpHandler {
            @Override
            public void handle(HttpExchange exchange) throws IOException {
                ...
            }
        }
    }
}

```

Figure 4.21: Reboot existing router

## 2. Reboot non-existing router

```

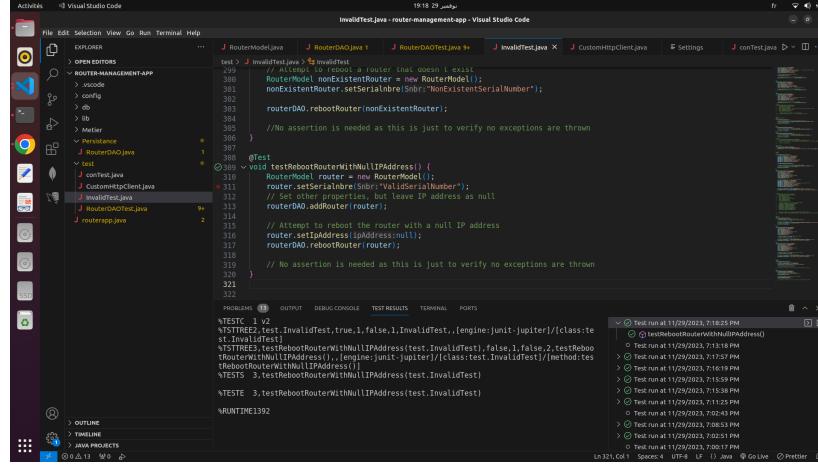
    void testDeleteRouterWithInvalidSerialNumber() {
        RouterDAO routerDAO = new RouterDAO();
        RouterModel router = routerDAO.getRouter("Serial");
        routerDAO.deleteRouter(router);
        RouterModel originalRouter = routerDAO.getRouter("Serial");
        assertEquals(originalRouter, router);
    }
}

void testRebootNonExistentRouter() {
    RouterDAO routerDAO = new RouterDAO();
    RouterModel nonExistentRouter = routerDAO.getRouter("NonexistentSerialNumber");
    routerDAO.rebootRouter(nonExistentRouter);
}

```

Figure 4.22: Reboot non-existing router

### 3. Reboot router with null IP address

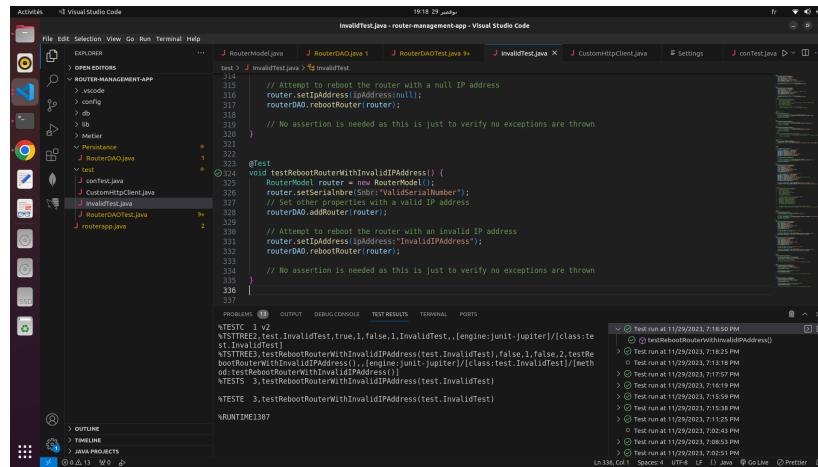


The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "ROUTER-MANAGEMENT-APP" with files like RouterModel.java, RouterDAO.java, RouterDAOTest.java, InvalidTest.java, Context.java, CustomHttpClient.java, and RouterApp.java.
- Code Editor:** Displays the Java code for the InvalidTest.java file. The code includes a test method named "void testRebootRouterWithNullIPAddress()". It creates a new RouterModel object, sets its serial number, and then calls the rebootRouter method on a RouterDAO object.
- Terminal:** Shows the command "invalidtest" being run, which outputs several test results for different test cases.
- Output:** Shows the results of the test runs, indicating success for most cases and failures for the ones involving null IP addresses.

Figure 4.23: Reboot router with null IP address

### 4. Reboot router with invalid IP address



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "ROUTER-MANAGEMENT-APP" with files like RouterModel.java, RouterDAO.java, RouterDAOTest.java, InvalidTest.java, Context.java, CustomHttpClient.java, and RouterApp.java.
- Code Editor:** Displays the Java code for the InvalidTest.java file. The code includes a test method named "void testRebootRouterWithInvalidIPAddress()". It creates a new RouterModel object, sets its serial number, and then calls the rebootRouter method on a RouterDAO object.
- Terminal:** Shows the command "invalidtest" being run, which outputs several test results for different test cases.
- Output:** Shows the results of the test runs, indicating success for most cases and failures for the ones involving invalid IP addresses.

Figure 4.24: Reboot router with invalid IP address

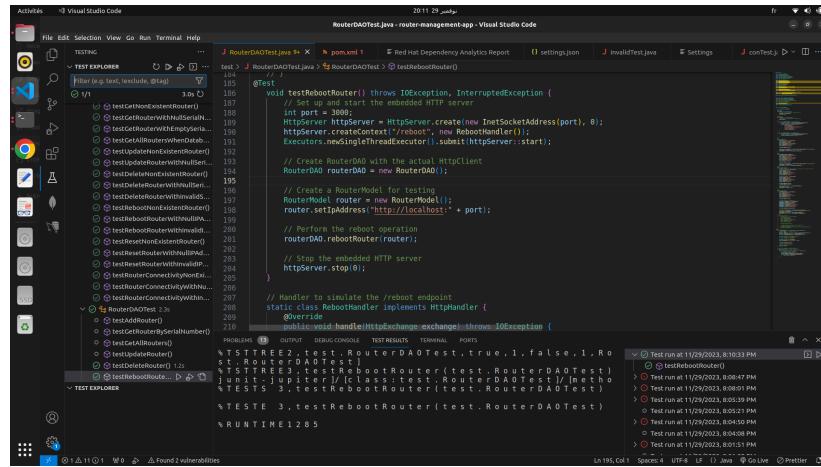
#### 4.2.6 Use Case: Reset Router

In the table (Table 4.6), various test cases are outlined to assess the functionality of resetting a router. Each test scenario, represented in a separate row, includes a detailed description, the corresponding test data, the observed result, and the expected outcome.

N	Description of the test cases	Test data	Result	Expected result
1	Reset existing router	("192.168.1.2")	valid	Router successfully reset
2	Reset non-existing router	("192.168.1.5")	invalid	Router not found
3	Reset router with null IP address	(Null)	invalid	Invalid input
4	Reset router with invalid IP address	(2314)	Error	Data type mismatch

Table 4.6: Reset Router

## 1. Reset existing router



```

    public void testResetExistingRouter() throws IOException, InterruptedException {
        // Set up and start the embedded HTTP server
        int port = 3000;
        HttpServer httpServer = httpServer.create(new InetSocketAddress(port), 0);
        httpServer.createContext("/reboot", new RebootHandler());
        Executors.newSingleThreadExecutor().submit(httpServer::start);

        // Create RouterDAO with the actual HttpClient
        RouterDAO routerDAO = new RouterDAO();

        // Create a RouterModel for testing
        RouterModel router = new RouterModel();
        router.setIpAddress("http://localhost:" + port);

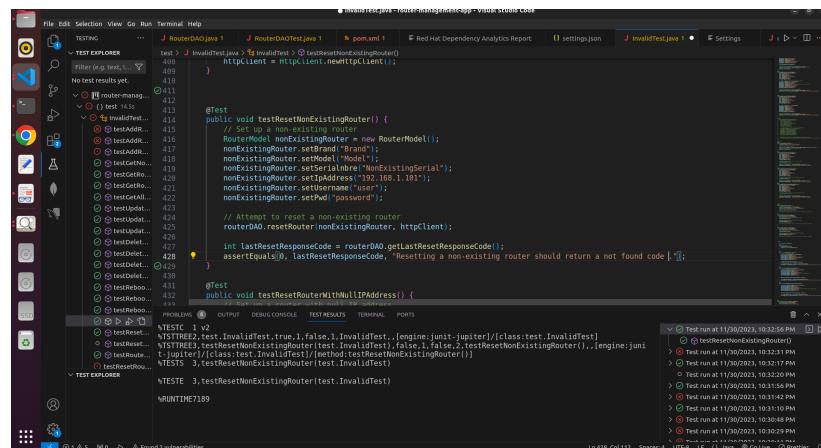
        // Perform the reboot operation
        routerDAO.rebootRouter(router);

        // Stop the embedded HTTP server
        httpServer.stop(0);
    }
}

```

Figure 4.25: Reset existing router

## 2. Reset non-existing router



```

    public void testResetNonExistingRouter() {
        // Set up a non-existing router
        RouterModel router = new RouterModel();
        nonExistingRouter.setBrand("Brand");
        nonExistingRouter.setModel("Model");
        nonExistingRouter.setSerial("Serial");
        nonExistingRouter.setIpAddress("192.168.1.100");
        nonExistingRouter.setUsername("User");
        nonExistingRouter.setPassword("Pass");

        // Attempt to reset a non-existing router
        routerDAO.resetRouter(nonExistingRouter, httpclient);

        int lastResetResponseCode = routerDAO.getLastResetResponseCode();
        assertEquals(404, lastResetResponseCode, "Resetting a non-existing router should return a not found code.");
    }
}

```

Figure 4.26: Reset non-existing router

### 3. Reset router with null IP address

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure for 'ROUTERMANAGEMENTAPP' containing 'RouterManagementApp.java', 'J\_RouterDAOTest.java', 'pom.xml', 'settings.json', and 'InvalidTest.java'. A file 'invalidtest.java' is open in the editor.
- Editor:** The code for 'invalidtest.java' is displayed, specifically the 'testResetRouterWithNullIpAddress' method. It includes assertions for a bad request code and a specific error message.
- Terminal:** Shows multiple test runs for 'InvalidTest' starting from 11/29/2023 at 10:32:17 PM.
- Output:** Displays the results of the test runs.
- Problems:** Shows one unresolved compilation problem related to 'setSerialNumber'.
- Test Explorer:** Shows the test cases: '3.testResetRouterWithNullIpAddress(test.InvalidTest)'.
- Status Bar:** Shows 'In 446 Col 122 Spaces: 4' and other standard status bar items.

Figure 4.27: Reset router with null IP address

### 4. Reset router with invalid IP address

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure for 'ROUTERMANAGEMENTAPP' containing 'RouterManagementApp.java', 'J\_RouterDAOTest.java', 'pom.xml', 'settings.json', and 'InvalidTest.java'. A file 'invalidtest.java' is open in the editor.
- Editor:** The code for 'invalidtest.java' is displayed, specifically the 'testAddRouterWithInvalidDataType' method. It includes assertions for a bad request code and a specific error message.
- Terminal:** Shows multiple test runs for 'InvalidTest' starting from 11/29/2023 at 10:32:17 PM.
- Output:** Displays the results of the test runs.
- Problems:** Shows one unresolved compilation problem related to 'setSerialNumber'.
- Test Explorer:** Shows the test cases: '3.testAddRouterWithInvalidDataType(test.InvalidTest)'.
- Status Bar:** Shows 'In 123 Col 108 selected Spaces: 4' and other standard status bar items.

Figure 4.28: Reset router with invalid IP address

#### 4.2.7 Use Case: Connectivity

In the table (Table 4.7), various test cases are outlined to evaluate the connectivity functionality of the router management application. Each test scenario, represented in a separate row, includes a detailed description, the corresponding test data, the observed result, and the expected outcome.

N	Description of the test cases	Test data	Result	Expected result
1	Check connectivity of an existing router	("192.168.2.3")	valid	Connectivity test successful
2	Check connectivity of a non-existing router	("192.168.2.5")	invalid	Router not found
3	Check connectivity of a router with null IP address	(Null)	invalid	Invalid input
4	Check connectivity of a router with wrong IP address type	(1222)	Error	Data type mismatch

Table 4.7: Connectivity

## 1. Check connectivity of an existing router

```

RouterDAO.java
public class RouterDAO {
    // Implementation details
}

RouterDAOTest.java
public class RouterDAOTest {
    @Test
    void testRouterConnectivity() {
        RouterDAO routerDAO = new RouterDAO();
        RouterModel router = new RouterModel();
        router.setIpAddress("192.168.1.1");
        routerDAO.addRouter(router);

        // Perform the router connectivity test
        routerDAO.routerConnectivity(router.getSerialNumber());

        // Capture the console output
        String consoleOutput = output.toString();

        // Check if the console output contains the expected message
        assertFalse(consoleOutput.contains("Router with serial number TestSerialNumber is not reachable."));
    }
}

```

Figure 4.29: Check connectivity of an existing router

## 2. Check connectivity of a non-existing router

```

InvalidTest.java
public class InvalidTest {
    @Test
    void testLastResetResponseCode() {
        int lastResetResponseCode = routerDAO.getLastResetResponseCode();
        assertEquals(500, lastResetResponseCode, "Resetting a router with wrong data type should return an internal server error code");
    }
}

RouterDAOTest.java
public class RouterDAOTest {
    @Test
    void testRouterConnectivityWithNonExistingRouter() {
        RouterModel router = new RouterModel();
        router.setSerialNumber("NonExistingRouter");
        router.setIpAddress("192.168.1.2");
        routerDAO.addRouter(router);

        // Attempt to check the connectivity of the router with an invalid IP address
        router.setIpAddress("192.168.2.1");
        routerDAO.routerConnectivity("NonExistingRouter");
    }
}

RouterDAO.java
public class RouterDAO {
    // Implementation details
}

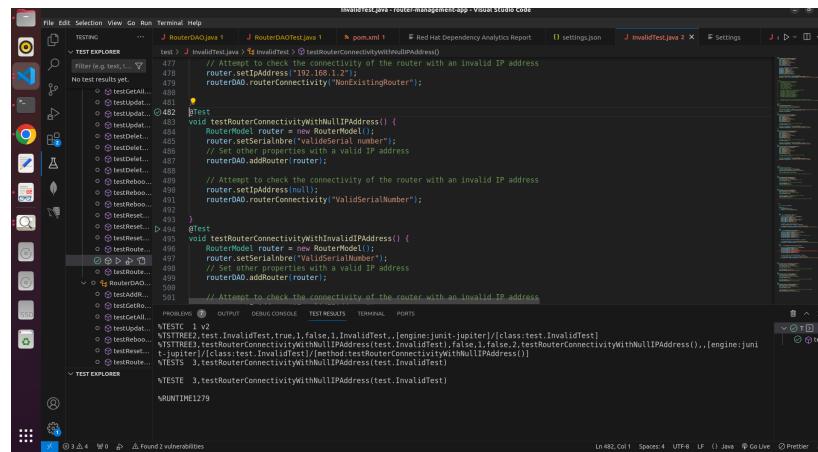
RouterDAOTest.java
public class RouterDAOTest {
    @Test
    void testRouterConnectivityWithInvalidIPAddress() {
        RouterModel router = new RouterModel();
        router.setSerialNumber("ValidSerialNumber");
        router.setIpAddress("192.168.1.1");
        routerDAO.addRouter(router);

        // Set other properties with a valid IP address
        router.setIpAddress("192.168.1.2");
        routerDAO.routerConnectivity("ValidRouter");
    }
}

```

Figure 4.30: Check connectivity of a non-existing router

### 3. Check connectivity of a router with null IP address



The screenshot shows the Visual Studio Code interface with the following details:

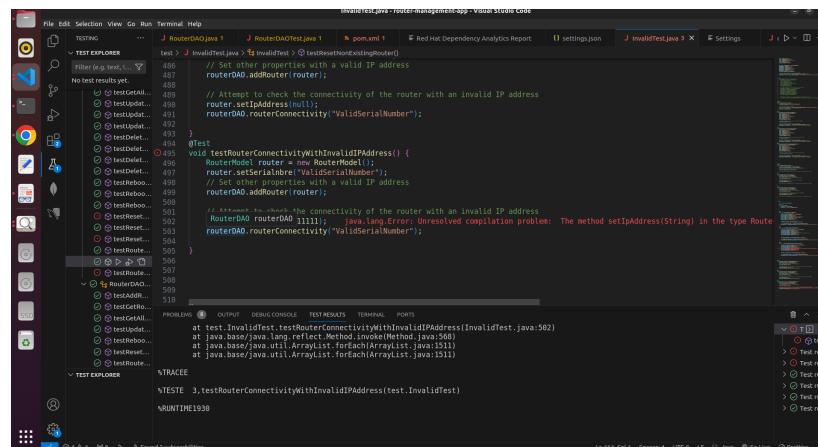
- File Explorer:** Shows the project structure with files like RouterDAO.java, RouterDAOTest.java, pom.xml, settings.json, and InvalidTest.java.
- Code Editor:** Displays Java code for testing router connectivity. A specific test method is highlighted:

```
    void testRouterConnectivityWithNullIPAddress() {
        // Attempt to check the connectivity of the router with an invalid IP address
        router.setIpAddress("192.168.1.2");
        routerDAO.routerConnectivity("NonExistingRouter");
    }
```
- Terminal:** Shows the command "InvalidTest.java" being run.
- Output:** Shows the test results:

```
TESTS: 1
InvalidTest.java > $ invalidTest
No test results yet.
```
- Problems:** Shows one error: "Unresolved compilation problem: The method setAddress(String) in the type RouteDAO is not visible".
- Test Explorer:** Shows the test RouterDAOTest has passed.

Figure 4.31: Check connectivity of a router with null IP address

### 4. Check connectivity of a router with wrong IP address type



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like RouterDAO.java, RouterDAOTest.java, pom.xml, settings.json, and InvalidTest.java.
- Code Editor:** Displays Java code for testing router connectivity. A specific test method is highlighted:

```
    void testRouterConnectivityWithInvalidIPaddress() {
        // Set other properties with a valid IP address
        router.setIpAddress(null);
        routerDAO.routerConnectivity("ValidSerialNumber");
    }
```
- Terminal:** Shows the command "InvalidTest.java" being run.
- Output:** Shows the test results:

```
TESTS: 1
InvalidTest.java > $ invalidTest
No test results yet.
```
- Problems:** Shows one error: "Unresolved compilation problem: The method setAddress(String) in the type RouteDAO is not visible".
- Test Explorer:** Shows the test RouterDAOTest has passed.

Figure 4.32: Check connectivity of a router with wrong IP address type

## 4.3 Testing Allure Report

Let's delve into the insights provided by the Testing Allure Report, offering a comprehensive view of our test results with rich visualizations and user-friendly analytics.

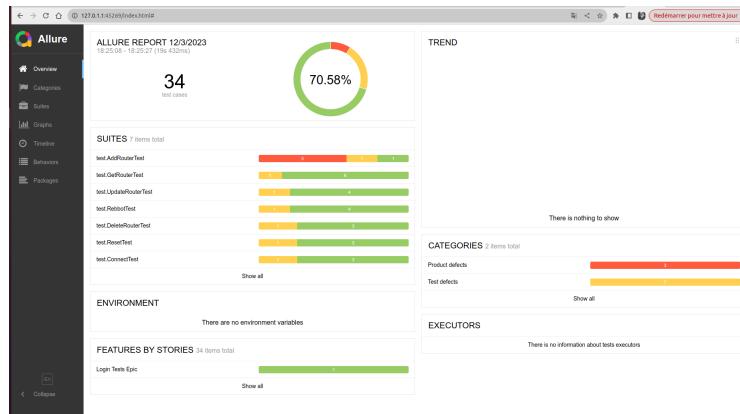


Figure 4.33: ALLURE REPORT Overview

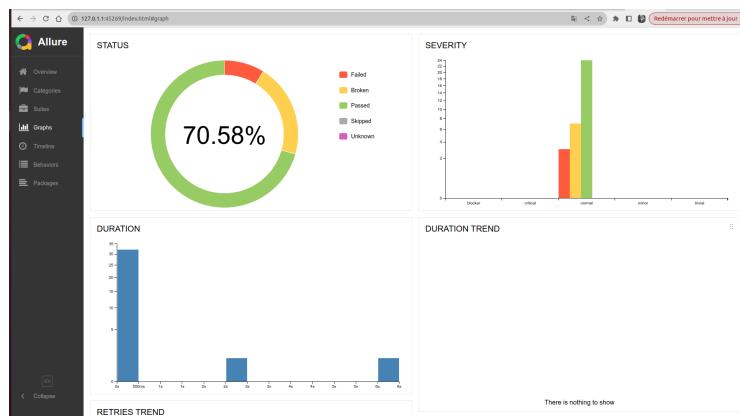


Figure 4.34: ALLURE REPORT Graphs

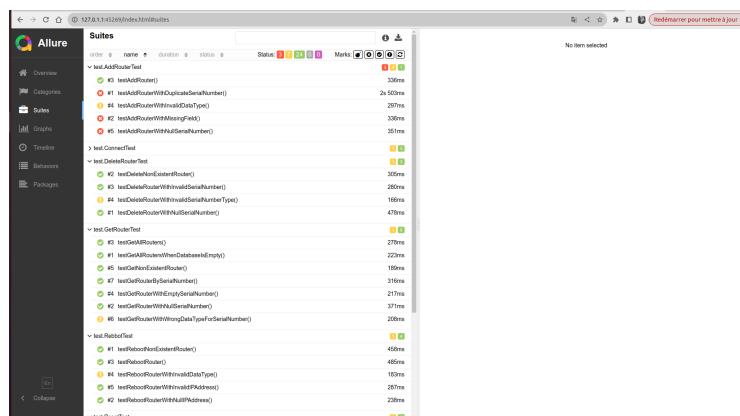


Figure 4.35: ALLURE REPORT Suites

## 4.4 Conclusion

In conclusion, this chapter delved into a comprehensive exploration of various use cases for our application. Through rigorous testing, we gained valuable insights into the strengths and limitations of our system across different scenarios.

# Conclusion

In summary, our mini project on managing routers and using the Router Management App was a great way to learn about routers and testing with JUnit. It's not just a regular computer program; it's like a special tool that helps us make sure everything in the app works really well. Our main goal is to check that every part of the app is strong and reliable for real-life situations.

We did a lot of testing using JUnit, which is a powerful tool to make sure our router management solution works the way it should. We're not only looking at regular router stuff in this project but also paying attention to something called Customer Premises Equipment (CPE). It's a crucial part of how the app works, and our tests make sure it works well too.

So, this project isn't just about routers; it's like a hands-on journey where we explore JUnit testing in detail. Our main aim is to make sure Customer Premises Equipment works perfectly. By doing this, we're learning the skills to create strong and trustworthy apps for managing networks.

# Bibliography

- [1] *ISTQB Syllabus*. 20/11/2023. [https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB\\_CTFL\\_Syllabus-v4.0.pdf](https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB_CTFL_Syllabus-v4.0.pdf)
- [2] Microsoft. *Visual Studio Code*. 20/11/2023. <https://code.visualstudio.com>
- [3] *Junit*. 25/11/2023. <https://junit.org/>
- [4] Oracle. *Mysql*. 25/11/2023. <https://www.mysql.com/>
- [5] Oracle. *Java Programming Language*. 25/11/2023. <https://www.oracle.com/java/>
- [6] *JavaFx*. 29/11/2023. <https://openjfx.io/>
- [7] *JUnit information*. 23/11/2023. <https://www.jmdoudoux.fr/java/dej/chap-junit.htm>
- [8] *Allure report*. 3/12/2023. <https://allurereport.org/>