Mini software testing project

Router management application

Student : CHADEN BEN AMMAR

December 4, 2023

Contents

List of Tables

List of Figures

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.

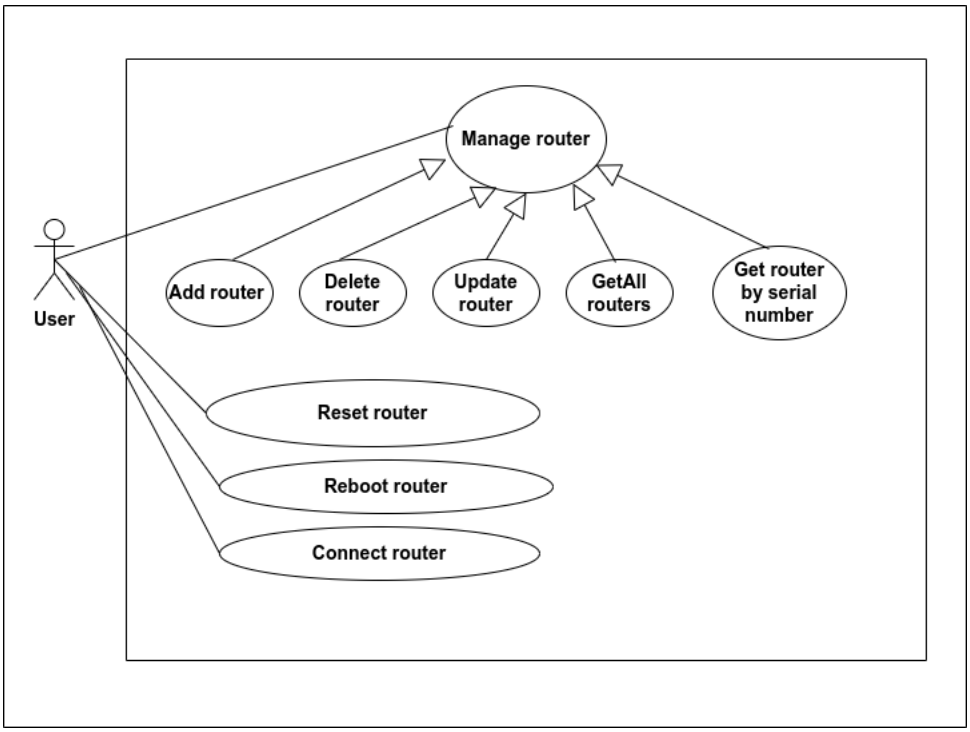

Figure 1.1: Use Case Diagram

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:

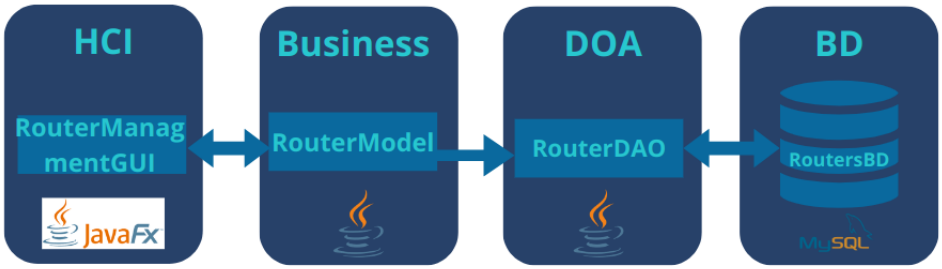

Figure 1.8: Architecture of the application

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.

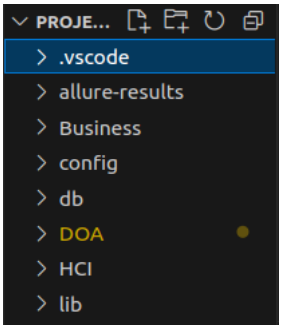

Figure 1.9: Layered Architecture of the application

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

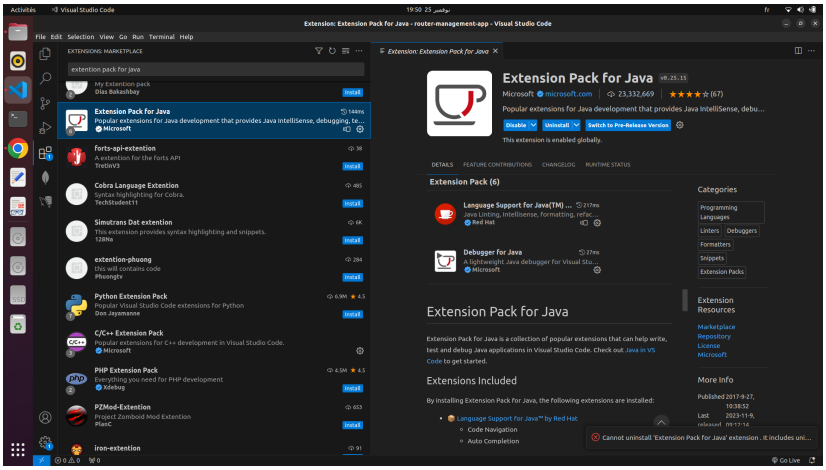

Figure 2.1: Extension pack for java

2. Write the test code

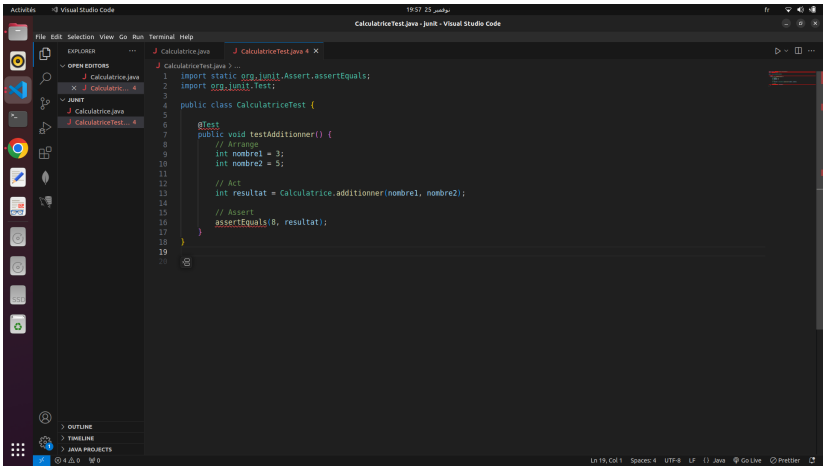

Figure 2.2: Write the test code

3. Activate the test

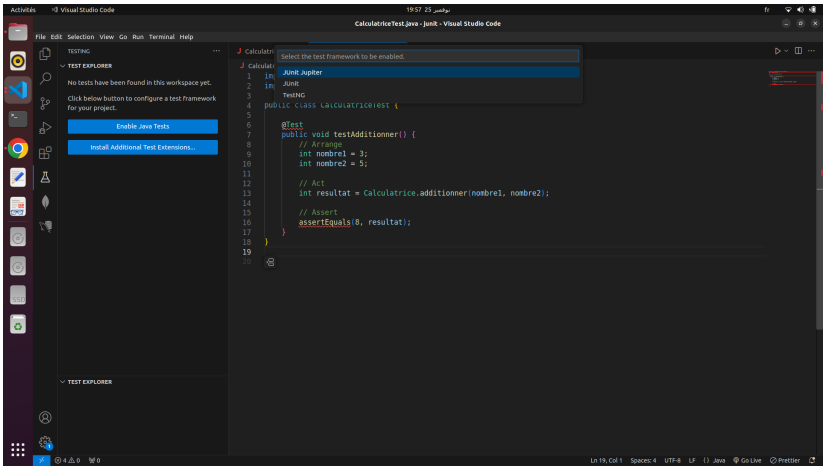

Figure 2.3: Activate the test

4. Test the solution

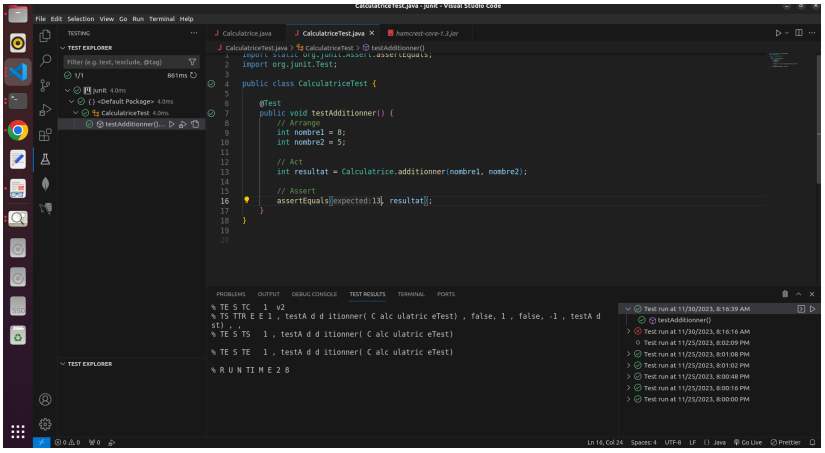

Figure 2.4: Test the solution

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 exapmles:

- **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.

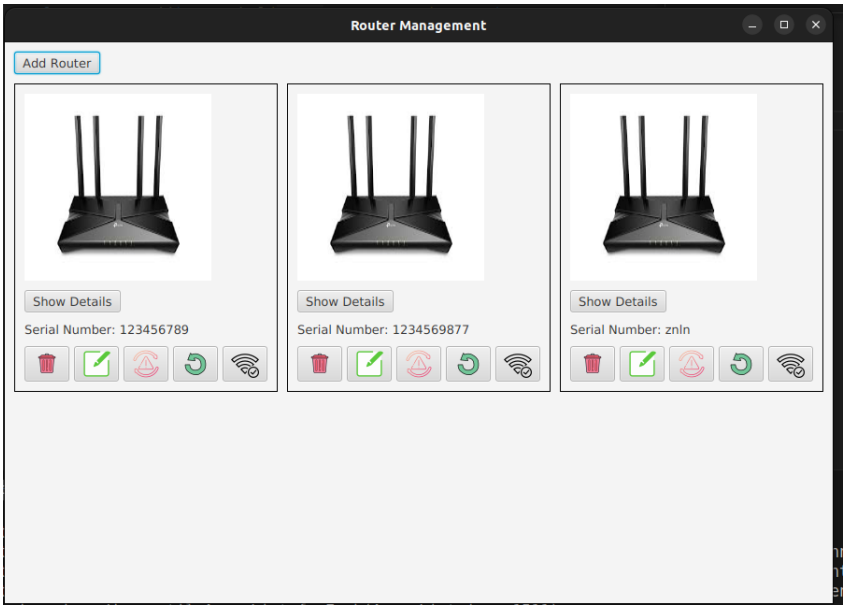

Figure 3.1: Router management app main interface

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.

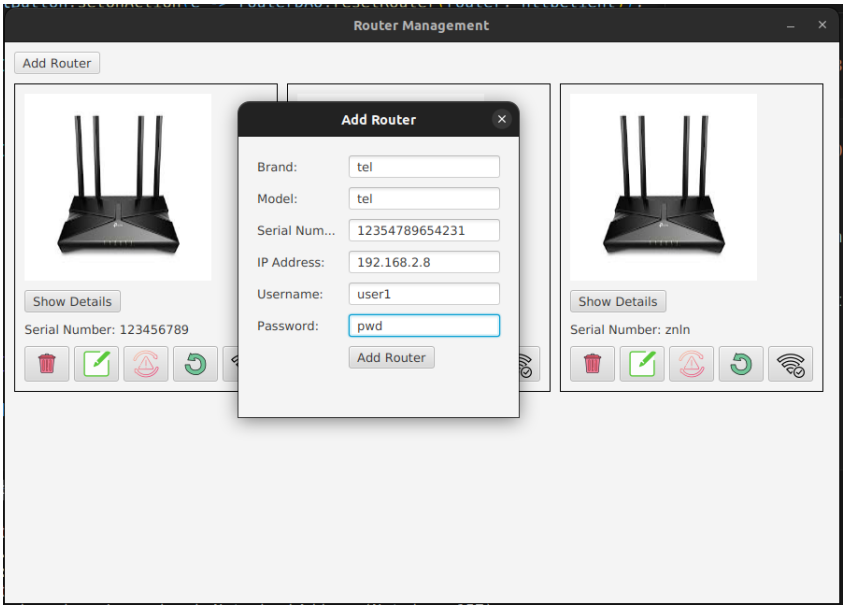

Figure 3.2: Add router interface

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.

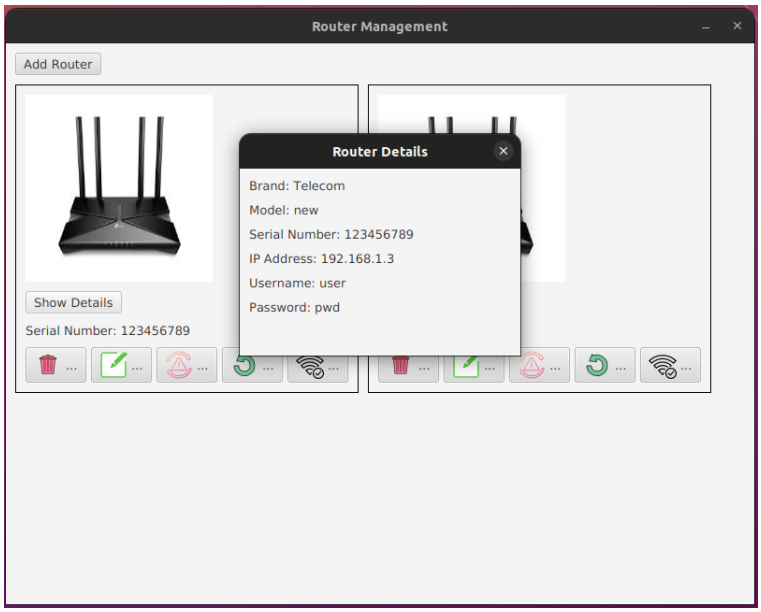

Figure 3.3: Show Details interface

3.3.4 Update router

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

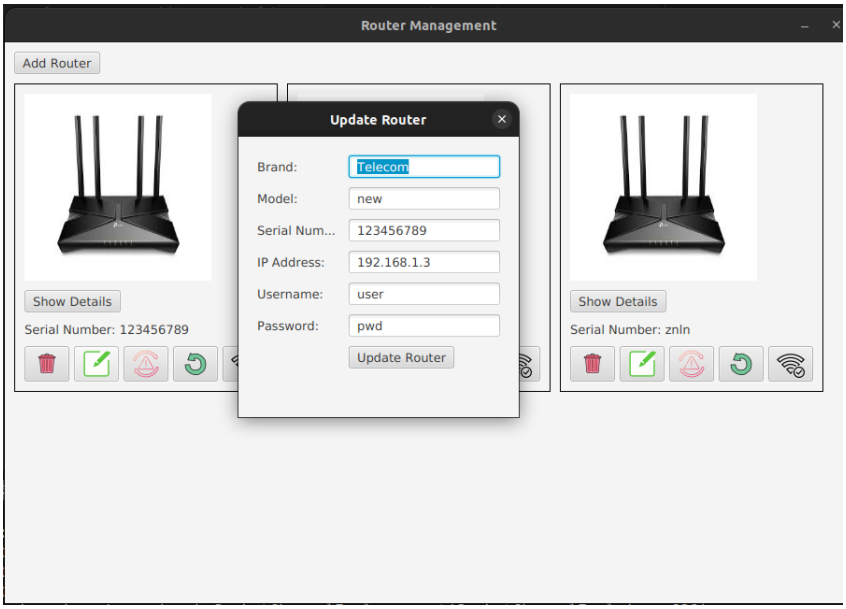

Figure 3.4: Update router interface

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.

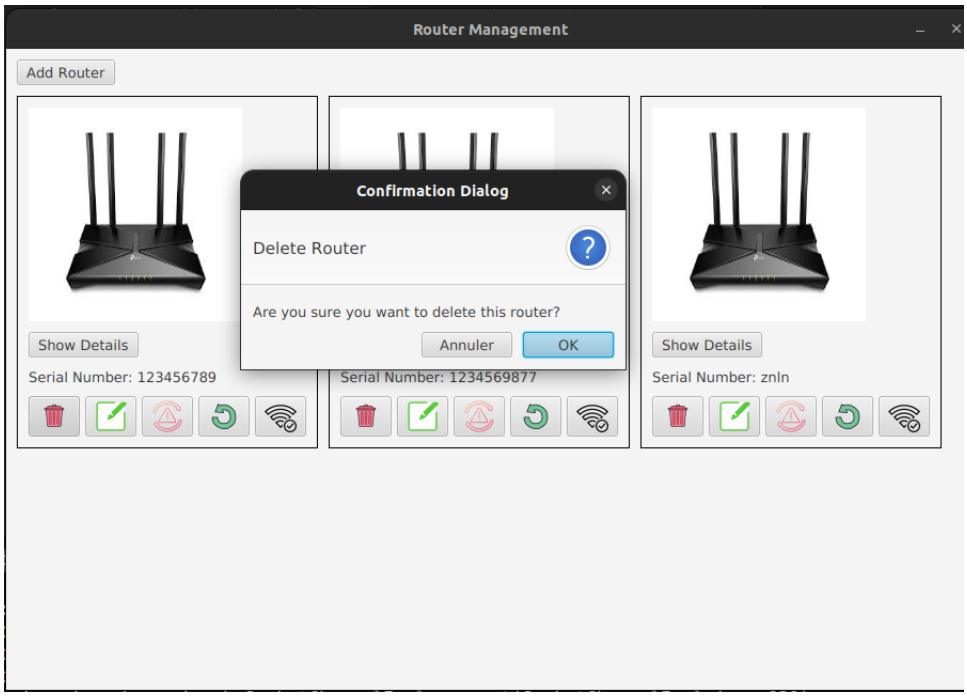

Figure 3.5: Delete router interface

3.4 Conclusion

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

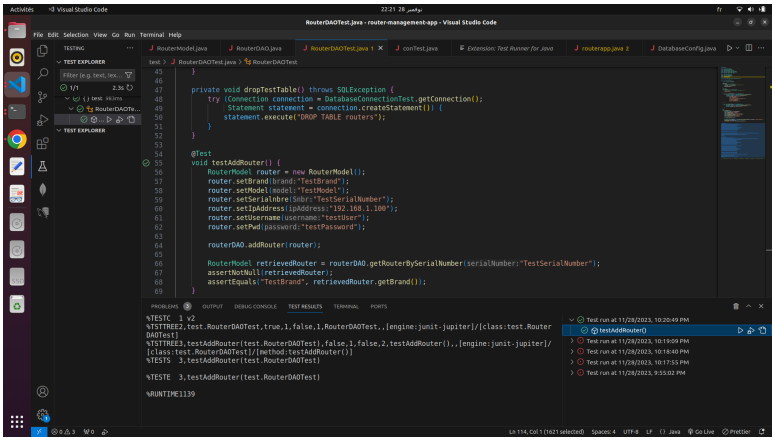

Figure 4.1: Add router

2. Add two routers with the same serial number

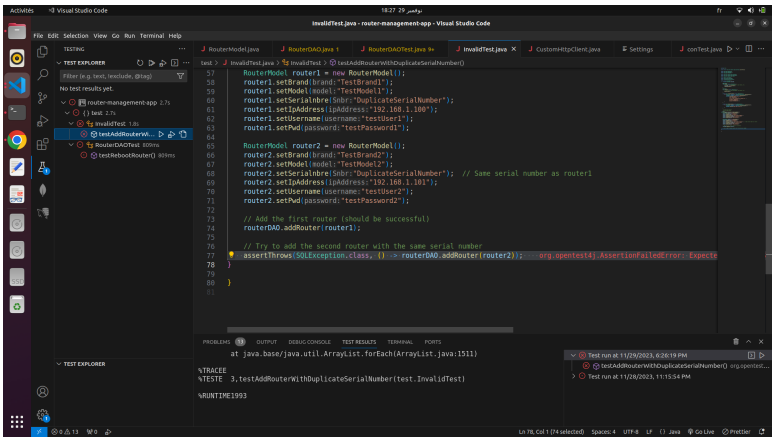

Figure 4.2: Add two routers with the same serial number

3. Add router with a missing field

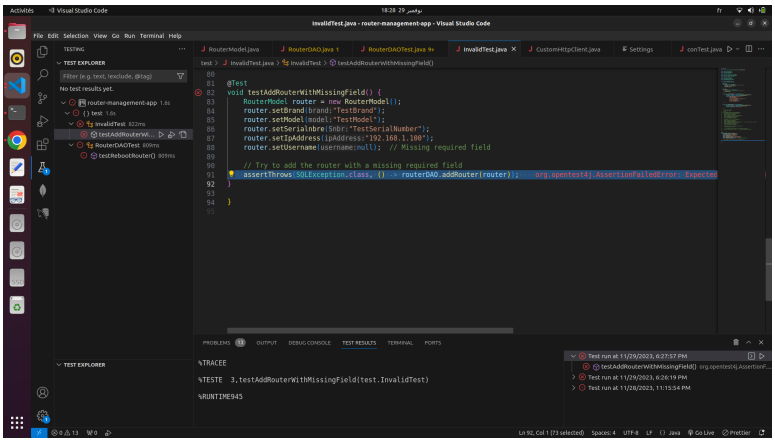

Figure 4.3: Add router with a missing field

4. Add router with null serial number

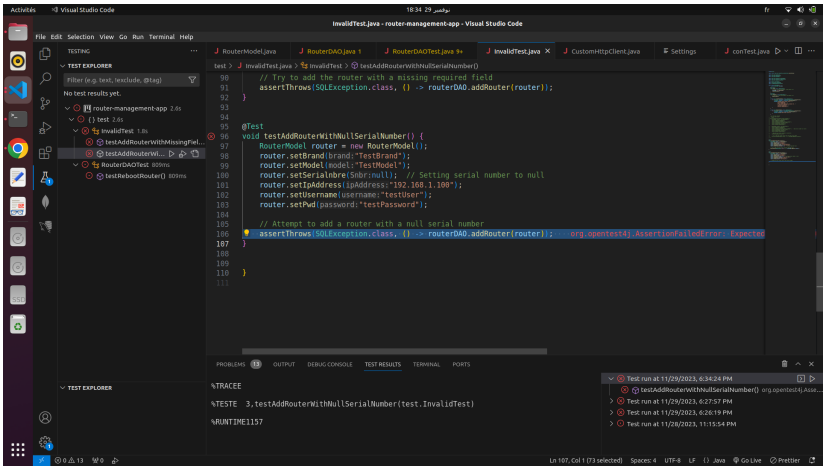

Figure 4.4: Add router with null serial number

5. Add router with wrong data type

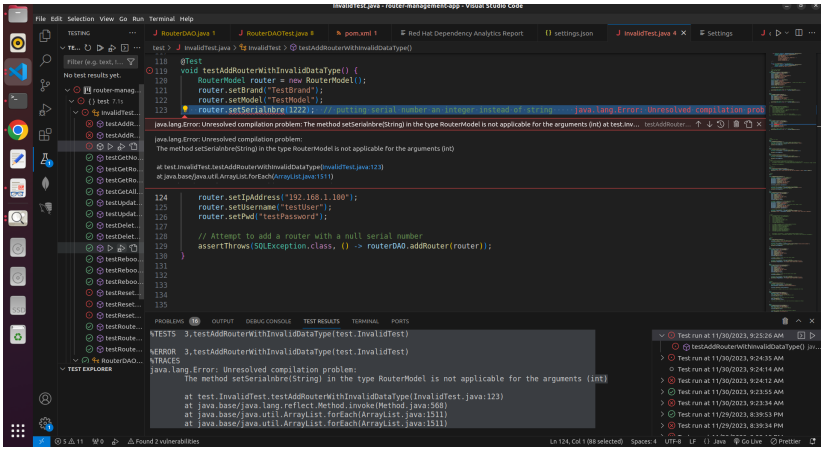

Figure 4.5: Add router with wrong data type

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

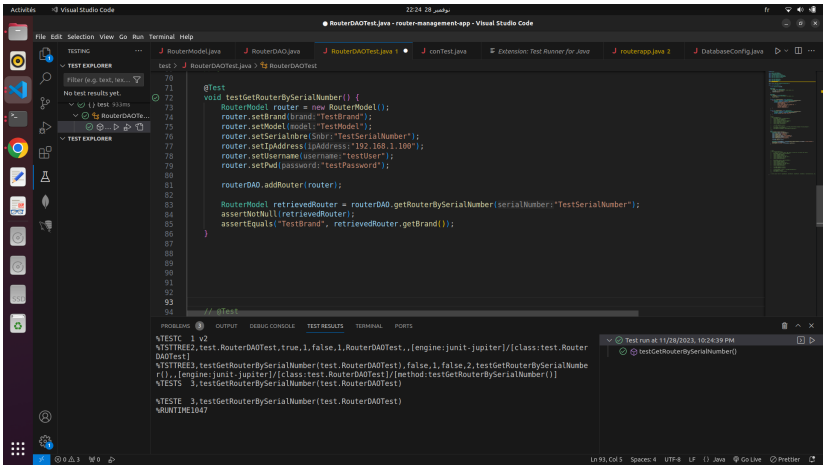

Figure 4.6: Get router by serial number

2. Get router with non-existing serial number

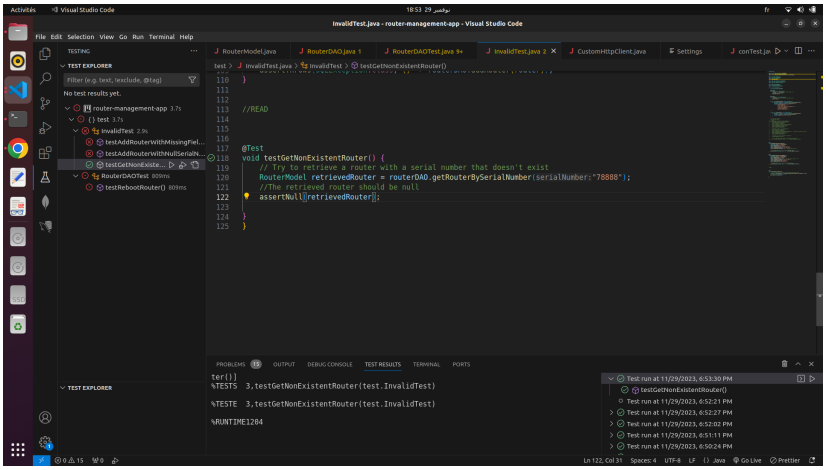

Figure 4.7: Get router with null serial number

3. Get router with null serial number

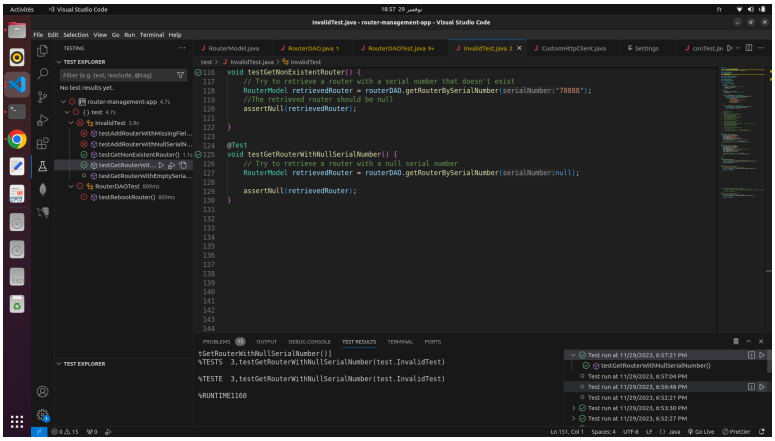

Figure 4.8: Get router with null serial number

4. Get router with empty serial number

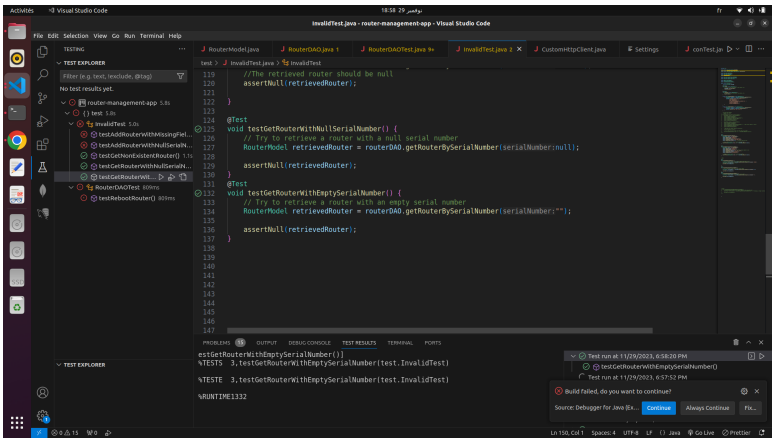

Figure 4.9: Get router with null serial number

5. Get router with wrong data type

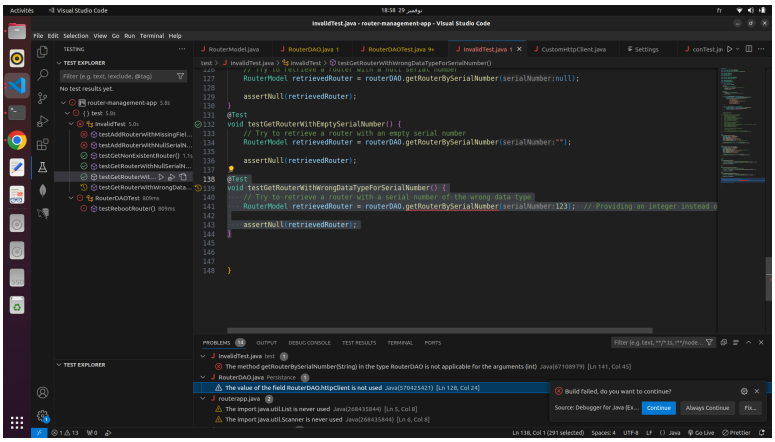

Figure 4.10: Get router with wrong data type

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

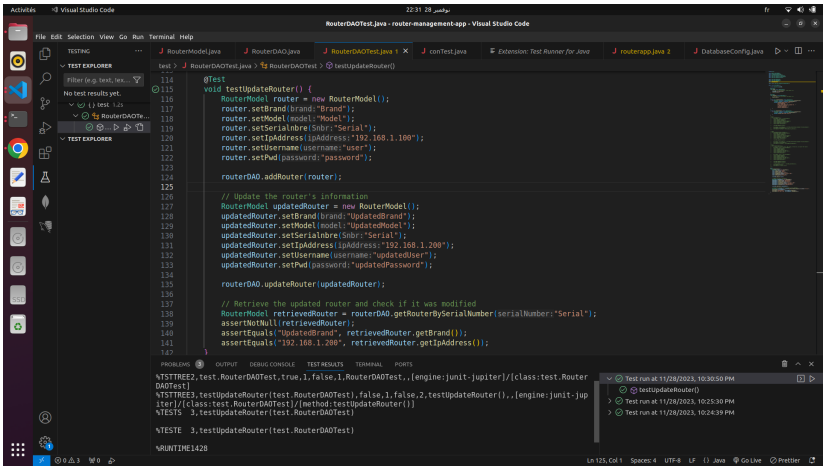

Figure 4.11: Update router

2. Update non-existing router

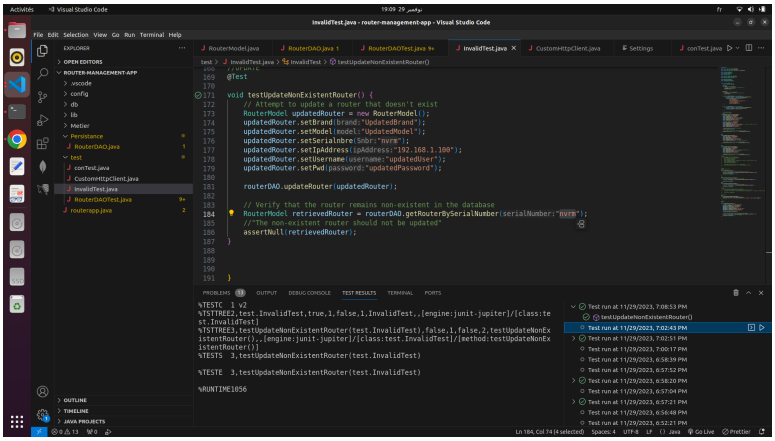

Figure 4.12: Update non-existing router

3. Update router with null data

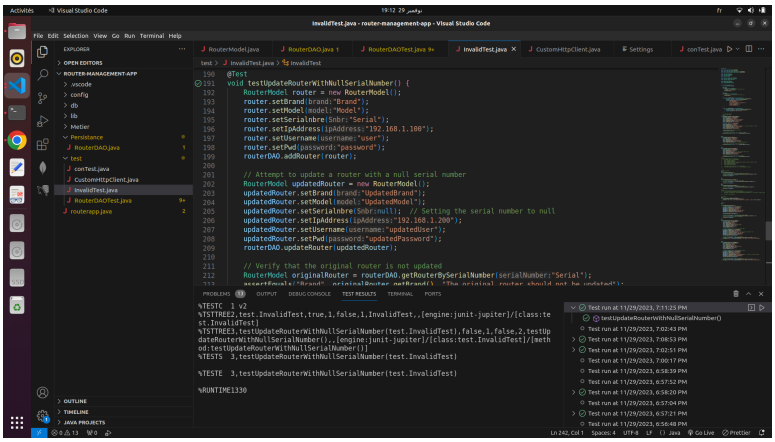

Figure 4.13: Update router with null data

4. Update router with empty data

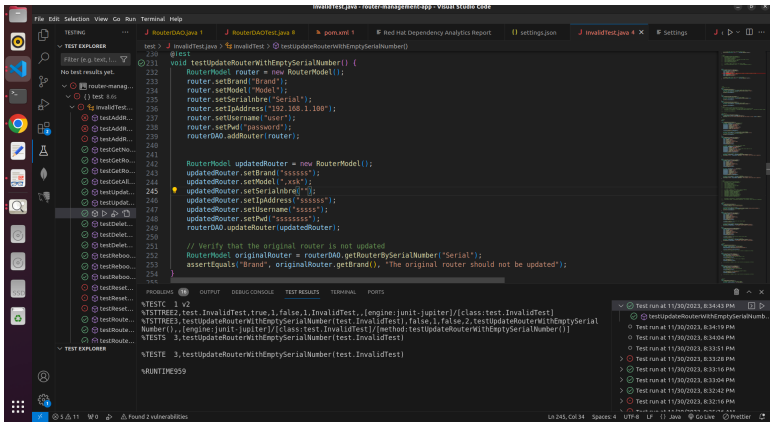

Figure 4.14: Update router with empty data

5. Update router with wrong data type

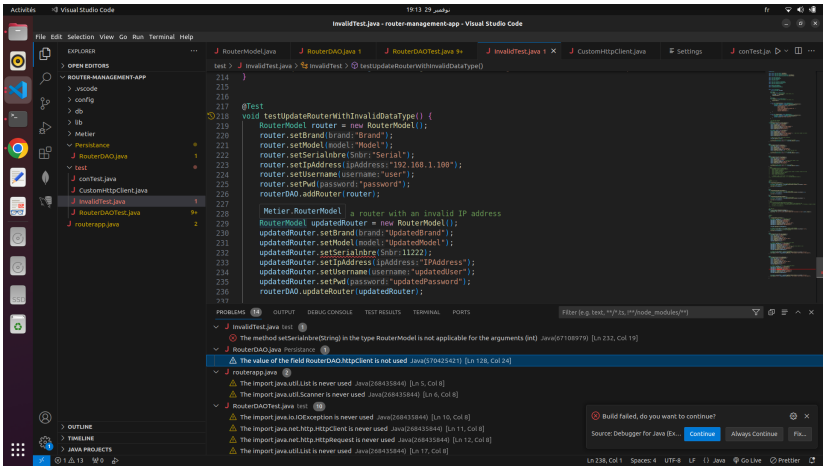

Figure 4.15: Update router with wrong data type

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

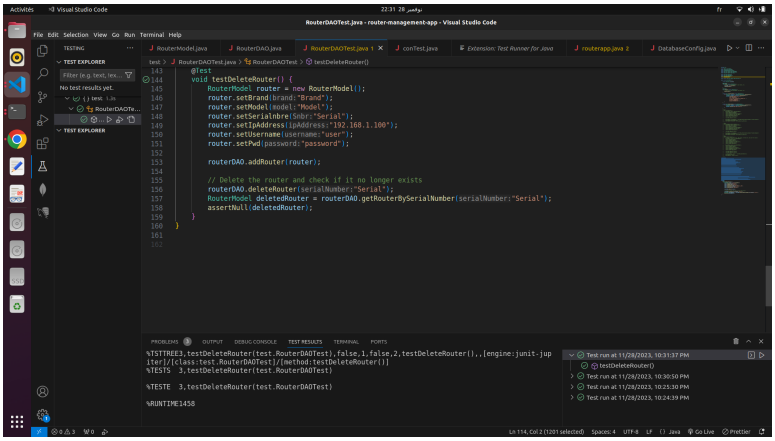

Figure 4.16: Delete existing router

2. Delete non-existing router

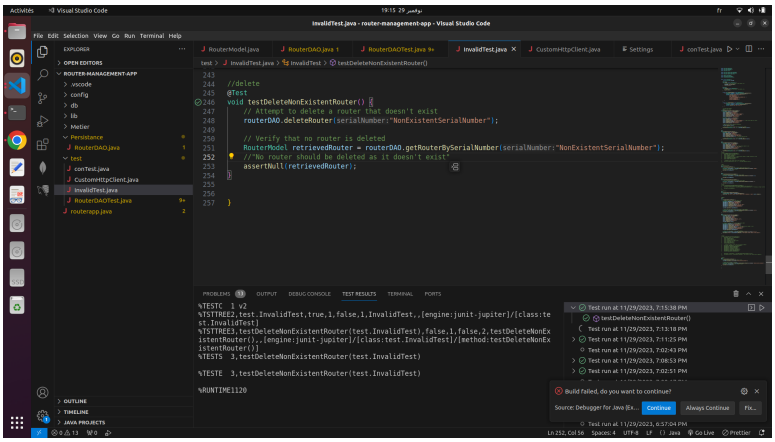

Figure 4.17: Delete non-existing router

3. Delete router with null serial number

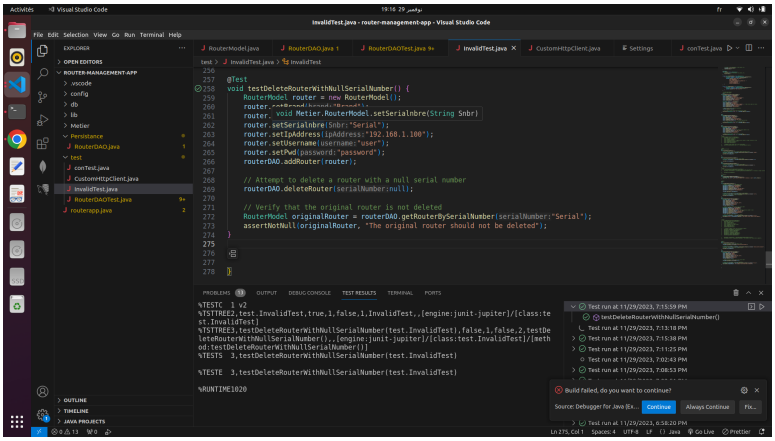

Figure 4.18: Delete router with null serial number

4. Delete with empty serial number

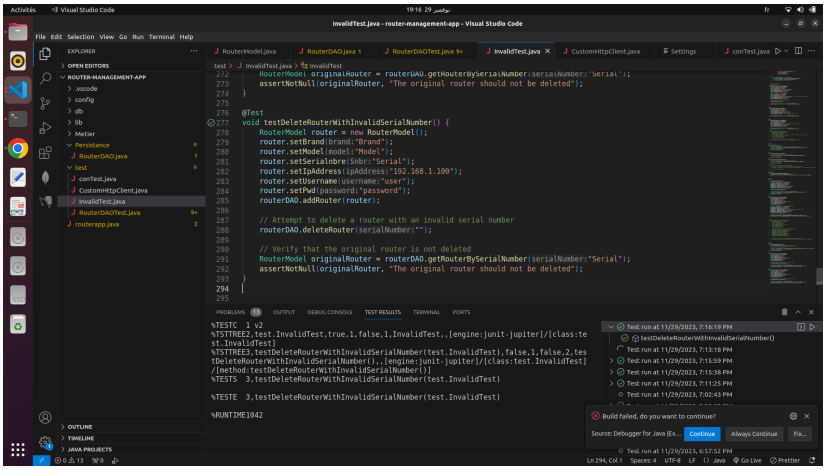

Figure 4.19: Delete router with empty serial number

5. Delete router with wrong serial number type

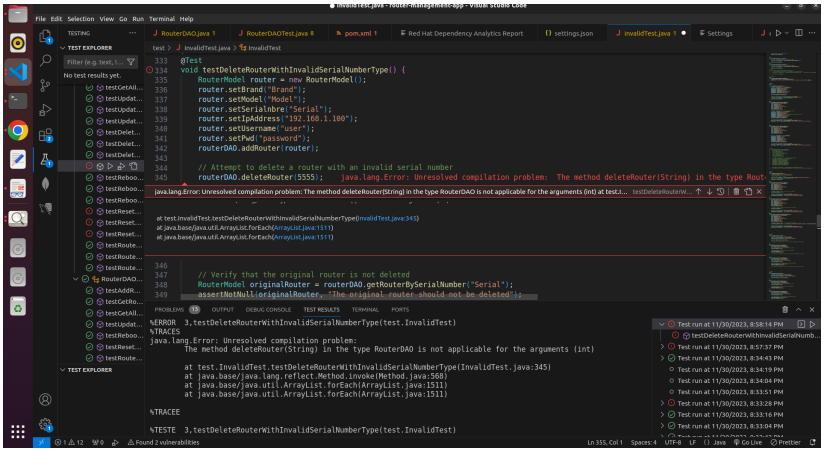

Figure 4.20: Delete router with wrong serial number type

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

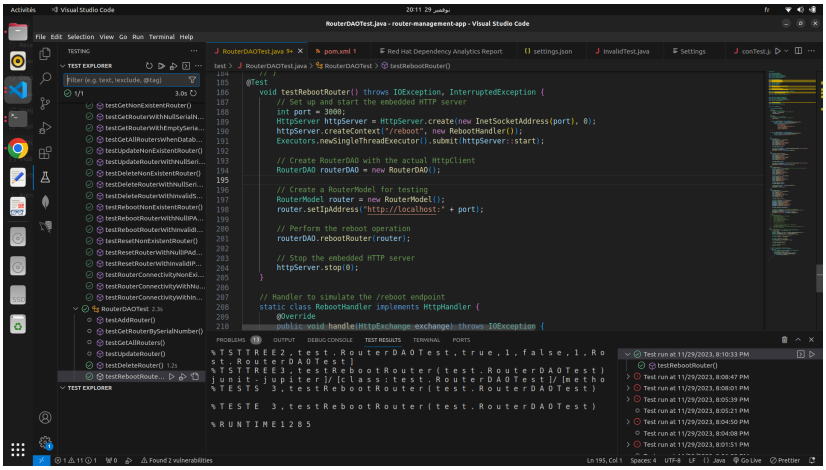

Figure 4.21: Reboot existing router

2. Reboot non-existing router

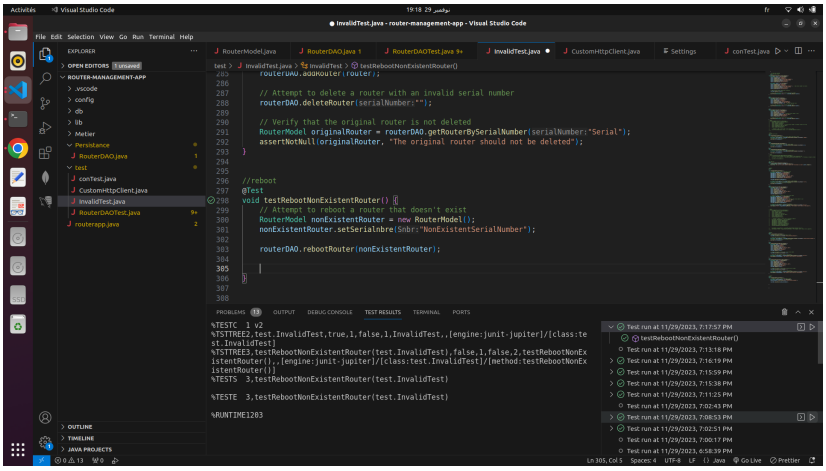

Figure 4.22: Reboot non-existing router

3. Reboot router with null IP address

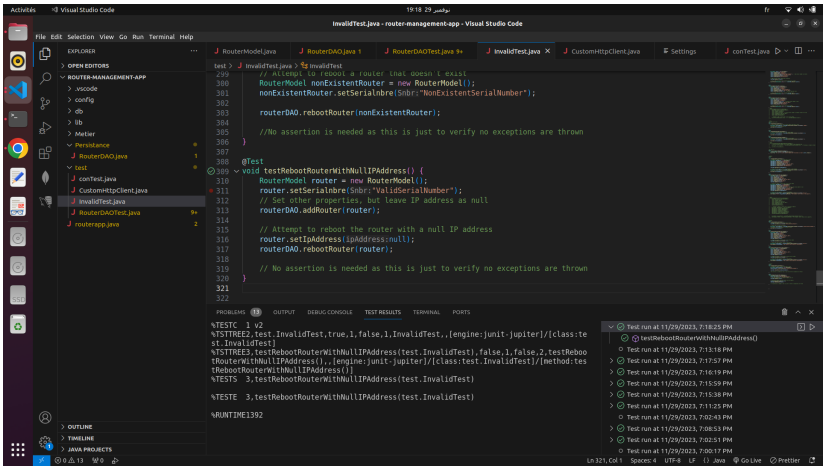

Figure 4.23: Reboot router with null IP address

4. Reboot router with invalid IP address

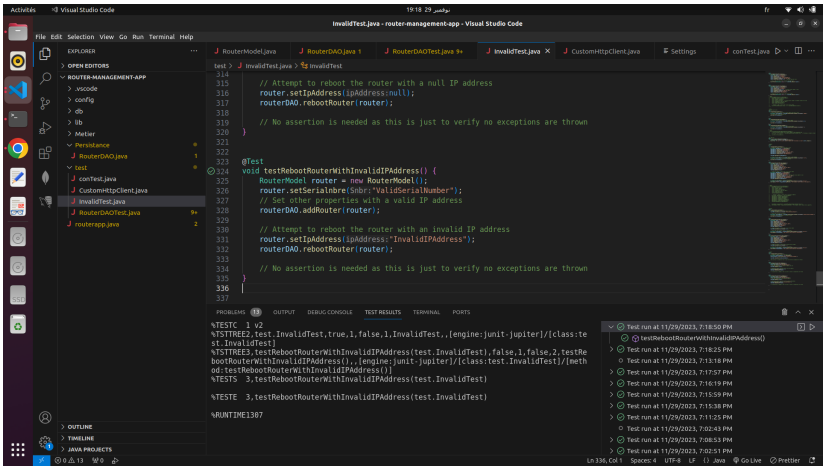

Figure 4.24: Reboot router with invalid IP address

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

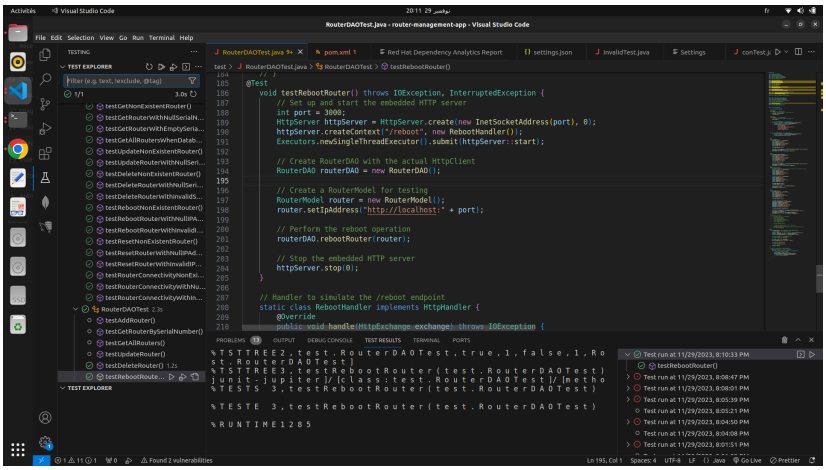

Figure 4.25: Reset existing router

2. Reset non-existing router

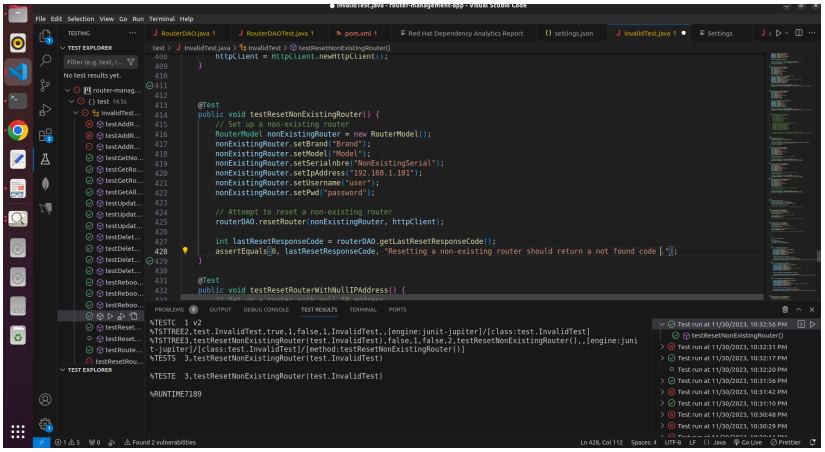

Figure 4.26: Reset non-existing router

3. Reset router with null IP address

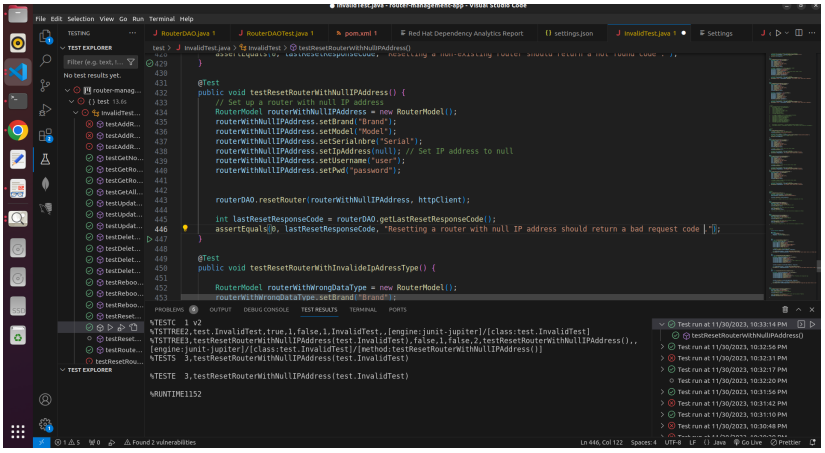

Figure 4.27: Reset router with null IP address

4. Reset router with invalid IP address

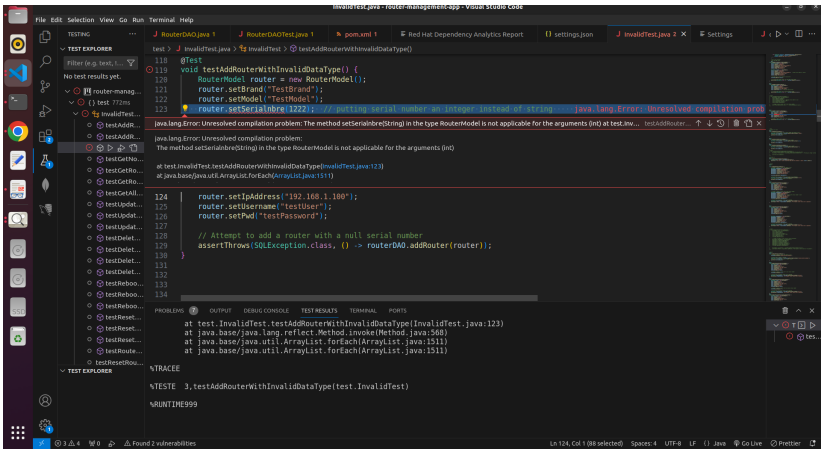

Figure 4.28: Reset router with invalid IP address

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

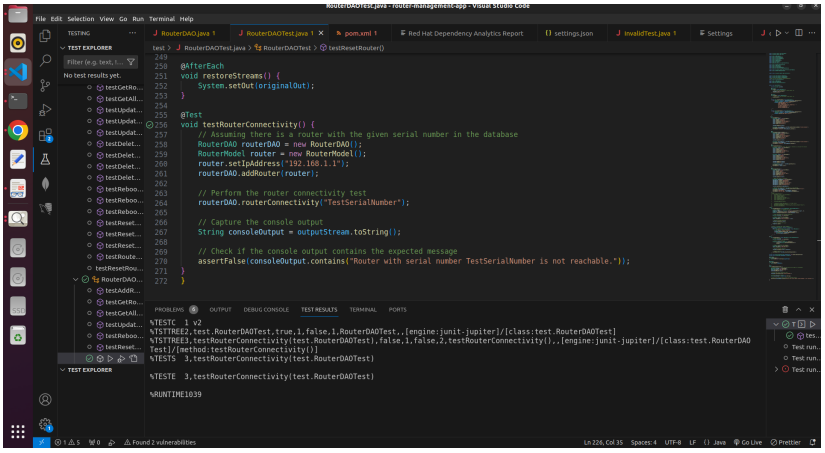

Figure 4.29: Check connectivity of an existing router

2. Check connectivity of a non-existing router

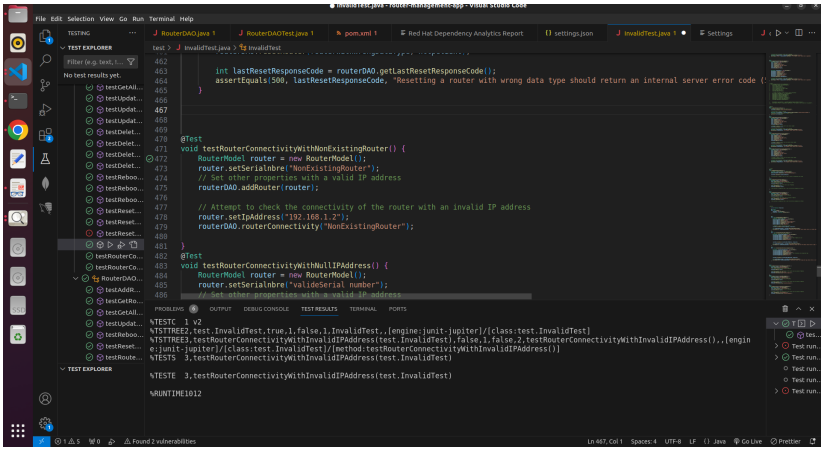

Figure 4.30: Check connectivity of a non-existing router

3. Check connectivity of a router with null IP address

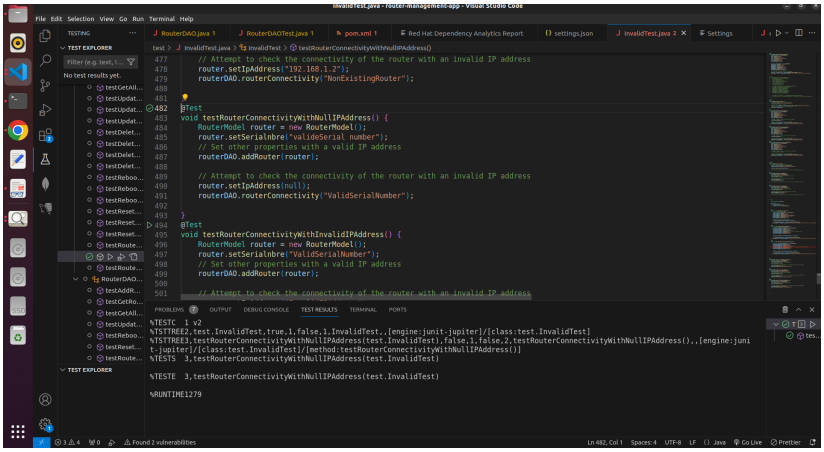

Figure 4.31: Check connectivity of a router with null IP address

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

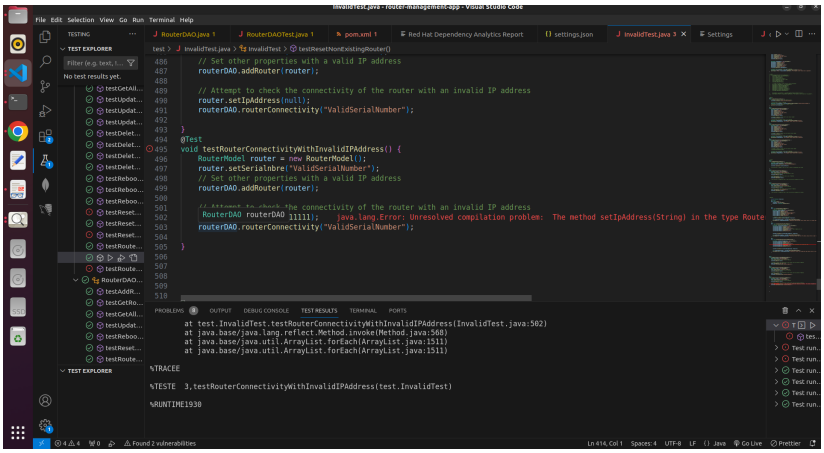

Figure 4.32: Check connectivity of a router with wrong IP address type

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.

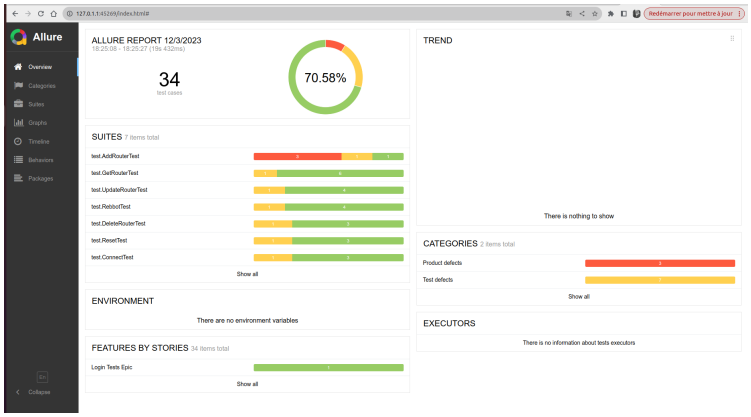

Figure 4.33: ALLURE REPORT Overview

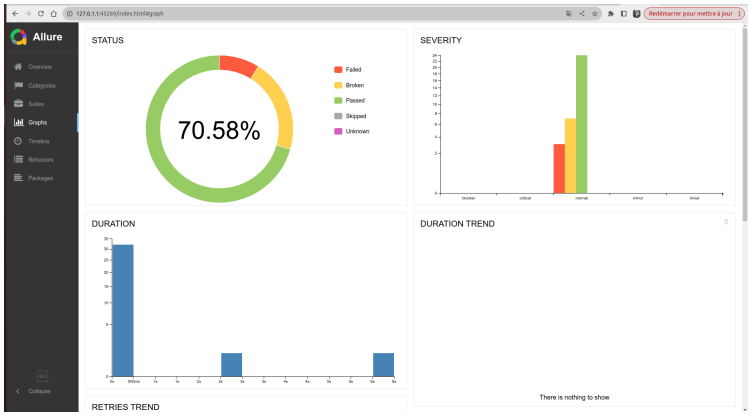

Figure 4.34: ALLURE REPORT Graphs

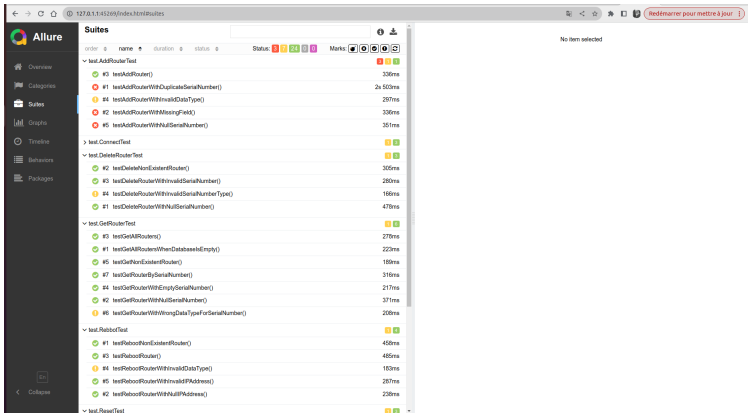

Figure 4.35: ALLURE REPORT Suites

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

[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/