UNIVERSITÀ
DEGLI STUDI
FIRENZE

**Scuola di Informatica**

Corso di Laurea Magistrale in Informatica

# Applying Modern Testing Techniques to Java Desktop Applications: A Case Study on Hostel Management using Maven,Docker, Jacoco and SonorCloud with Github


**Name:** Syed Bilal Hassan
**Matricula No:** 7127510
**Email Id:** syed.hassan@edu.unifi.it

**Professor Name:** Lorenzo Bertini

**Email ID:** (lorenzo.bettini@edu.unifi.it)


## Course: Automated Software Testing

**LINK TO THE BOOK:** http://leanpub.com/tdd-buildautomation-ci

Anno Accademico 2024/2025

# DEDICATION

**With love and gratitude, I dedicate this work to**

**my dearest wife, Sartaj,**

**and my cherished son, Jibran**

Dedica: scrivi qui la dedica

# TABLE OF CONTENTS

# Introduction:

This project implements a desktop application to manage room assignments in a student hostel. The system is designed to manage multiple apartments, each containing seven rooms.

The main objectives of the project are:

- **Track room availability:** The application monitors which rooms are free and which are occupied.
- **Prevent double bookings:** Rooms cannot be assigned to more than one tenant simultaneously.
- **Provide a GUI:** Users interact with the system through a Swing-based graphical interface.
- **Demonstrate software testing techniques:** The project applies **unit testing, integration testing, end-to-end (E2E) testing, mutation testing**, and **code coverage analysis** to ensure a high level of reliability.

The system is built in **Java 17**, using **Maven** for project management and dependency handling. **Docker and Test containers** are used to provide reproducible testing environments, particularly for the MongoDB database used in integration and end-to-end tests.

This project serves as a practical application of the techniques covered in the Automated Software Testing course, showcasing test-first development, automated verification, and robust system design.

## Applied Techniques and Frameworks :

• **Unit Testing (JUnit 4)**: Ensures correct behavior of individual methods in Room and Apartment.

• **Mocking (Mockito 5)**: Simulates dependencies in tests to isolate the unit under test.

• **GUI Testing (AssertJ Swing)**: Automates interaction with Swing components and verifies user interface behavior.

• **Test containers (Docker)**: Provides ephemeral MongoDB instances for integration tests without requiring local installation.

• **JACOCO:** Measures code coverage.

• **PIT (Mutation Testing)**: Ensures tests detect faults effectively, focusing on critical classes Room and Apartment.

• **SonarCloud:** Measures code coverage, Code Smells , and remove any technical debt and vulnerabilities and shows the results in the form of badge in the Readme file of the Github

## Design and Implementation Choices:

- **Architecture:** Simple MVC approach separating data models (Room, Apartment) from GUI views.
- **Database:** MongoDB is used for persistence; Test containers ensures consistent integration test environments.
- **GUI:** Swing components were chosen for desktop interactivity. Careful attention was given to avoiding screen-resolution dependencies and thread-related issues.
- **Docker:** All database dependencies are containerized, ensuring reproducible builds for tests and CI/CD.
- **Testing Strategy:**
  - Unit tests for core logic (Room, Apartment).
  - Integration tests for database interactions.
  - E2E tests for GUI workflows.

# Testing Strategy:

Testing is divided into three levels, each covering a different aspect of the application:

## 1. Unit Testing

- Focused on individual classes: Room, Apartment.
- Used JUnit 4 and Mockito to test methods such as:
  - assignTenant(), vacate(), isAvailable()
  - Counting available rooms, checking if an apartment is full or empty
- Verified exception handling for double assignment attempts.

## 2. Integration Testing

- Verified the interaction between controllers and the database using MongoDB Test containers.
- RoomMongoRepositoryTestContainersIT tests included:
  - Finding rooms by room number
  - Saving new rooms
  - Vacating rooms
  - Handling null or empty tenants
- MongoDB container automatically starts and is cleaned after each test for reproducibility.

## 3. End-to-End (E2E) Testing

- Verified GUI interactions using AssertJ Swing:
  - Adding tenants via the GUI
  - Selecting rooms in the JList and vacating them
  - Checking that error messages are displayed correctly
- GUI tests interact with a live MongoDB Test container, simulating real-world usage.
- Ensures that the full flow of the application works correctly.

# Challenges and Solutions:

While developing the Hostel Management application, I ran into a few practical challenges, which I had to solve to make the system reliable and meet the course requirements.

## Testing of toString() Method:

Some methods, like the toString() method in the Room class created for the workflow,but, SonarCloud marked them as uncovered, which prevented the Quality Gate from passing. Even though testing toString() isn't usually necessary, I decided to write simple unit tests for it. This not only increased the code coverage but also allowed the project to pass SonarCloud's Quality Gate.Testing is divided into three levels, each covering a different aspect of the application:

## Integrating GUI, controller, and database for testing:

It was quite chalengin to integrate the GUI and and the Testing the SWINGAPP.Altough i spent a lot of time of time but it was worthwhile as i learnt number of new things and concepts.Testing the full workflow was tricky because the application uses Swing for the GUI, controllers for business logic, and MongoDB for persistence. To handle this, I used AssertJ Swing for GUI tests and Testcontainers to run a temporary MongoDB instance for integration and end-to-end tests. I also added a feature to automatically create rooms during GUI tests, which simplified setup. This way, I could test the whole application as a user would interact with it.

## Configuration of SonarCloud setup:

One of another most difficult part was to introduce and implement the SonarCloud.As we know the SonarCloud is quite help full in testing techniques but I got a lot of problems while Implementing it.This first and the foremost was the implementing the Exclusions in pom.xml i tried multiple stratigies in the properties section to exclude some methods but all in vain.Even I tried the Java ID to exlude and make the coverage pass but it was not quite fruitful and it affects my whole workflow structure so i remove it and as told earliar above i created testing Method for toString() after that my configuration of sonarcloud was seuccesfule

## Instructions to Build and Run the Project:

The project is fully automated and can be executed on any machine where the following tools are already installed:

- Java 17.0.15 (the project uses Java 17.15)
- Maven
- Docker and Docker Compose

No manual setup of MongoDB or external services is required because all database tests run using Test containers, which automatically starts a temporary MongoDB container during the build.

I would recommend to use Java17.0.15 because some times when running from scratch from a fresh machine it may cause error, when during the proof of concept of the project I came to know that it works perfectly with the 17.0.15.

## Steps to Run the Project

## Clone the Project

git clone https://github.com/SyedBilal61/TDDhostel_management.git

cd TDDhostel_management

## Verify Environment

- Check Java version(must jdk 17.0.15)

  java -version

- Check Maven version:

  mvn -version (apache-maven-3.9.11)

- Check Docker:

  docker --version

  docker info

## Build and Run Tests

mvn clean verify

## The command will:

- Compile the code
- Run unit, integration, and end-to-end tests
- Automatically start and stop a MongoDB container for integration/E2E tests
- Generate code coverage reports using Jacoco
- Upload coverage and metrics to SonarCloud (if configured)

## View Test Reports

- Surefire reports (unit & integration tests): target/surefire-reports
- Jacoco coverage report: target/site/jacoco/index.html

## Run the Application

- Start the Swing GUI (optional, for local testing)

Moreover, If you want to run the app manually and want to check the the UI frame work of the Application, you can open the project after cloning and in the folder src/main/java there is package called com.hostel.app.swing there is file called HostelSwingApp.java.

This file is the entry point for the our hostel_management Application and it does the following operations:

**Command-line options**: Allows you to configure MongoDB host, port, database name, and collection via command-line arguments. Default values are set to localhost:27017 and the rooms collection in the hostel database.

**MongoDB Connection**: Establishes a connection to MongoDB using MongoClient. This is normally handled automatically by Testcontainers during testing, but here it allows you to run the app manually with a real or local MongoDB instance.

**GUI Setup**:

- Initializes the RoomSwingView (the Swing GUI).
- Connects the controller to the view.

So to run this and check this manually the UI frame you must have to run the mongodb container using docker with the following command

docker run -d --name my-mongodb -p 27017:27017 mongo

and after few moments you go on the src/main/java and in the package com.hostel.app.swing and you fill find the file HostelSwingApp.java ,just Right click on the file and **RUN AS > Java application** where you will find the UI frame of the application so that you can perform and check all the CRUD operations

This will start the full Swing GUI, connected to the specified MongoDB, allowing you to interact with apartments, rooms, and tenants as a user would. It essentially simulates the real application environment, which is also what your teacher would see when testing the project.

# CI/CD, Coverage, and Code Quality

The project is hosted on GitHub and fully integrated with Continuous Integration (CI) using GitHub Actions and Maven. Each push to the repository triggers the build workflow, which automatically:

- Compiles the project
- Runs unit, integration, and end-to-end (E2E) tests
- Starts a temporary MongoDB container via Testcontainers for integration/E2E tests
- Generates code coverage reports using Jacoco
- Sends metrics to SonarCloud for quality analysis

## Code Coverage:

Code Coverage is tracked in two ways:

- Coveralls: Displays the percentage of code covered by automated tests. The coverage badge in the README shows the real-time status for the latest build
- SonarCloud: Also measures code coverage, along with other quality metrics.The colour of the badge is by default grey

## SonarCloud Metrics

SonarCloud provides a comprehensive view of code quality, including:

- Quality Gate: Indicates whether the project passes SonarCloud's quality standards. Passing the Quality Gate ensures high reliability and maintainability

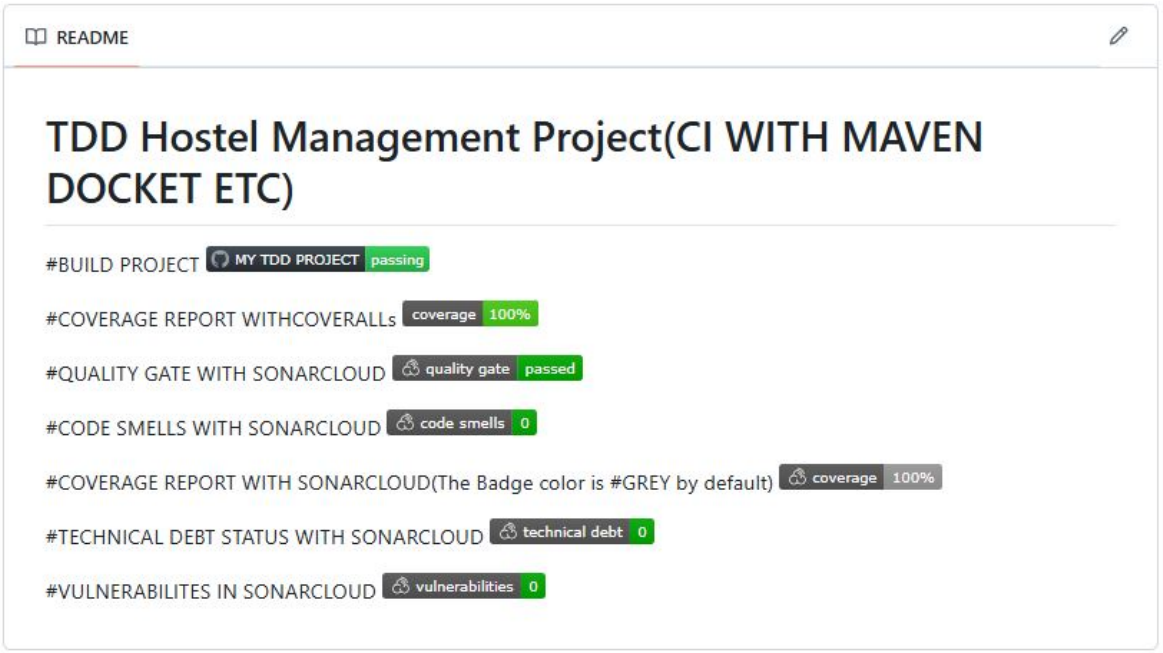
- Code Smells: Highlights potential issues in the code that might not cause immediate errors but could lead to maintainability problems.

- Technical Debt: Estimates the effort required to fix all detected code issues. Low technical debt means a cleaner, more maintainable codebase.

- Vulnerabilities: Detects potential security issues in the code. The goal is to have zero vulnerabilities, which improves software security.

These badges in the README make it easy for anyone  to instantly see the project's **build status, test coverage, and code quality** at a glance. They demonstrate that the project not only works correctly but also adheres to professional software engineering standards

Here is pictorial Glimpse of the that metrices from my project:



## Conclusion:

This project demonstrates the practical application of modern software testing techniques to a Java desktop application for managing hostel rooms. By implementing unit tests, integration tests, end-to-end (E2E) tests, and mutation testing, the system ensures high reliability and robustness.

Using Testcontainers and Docker, the project eliminates the need for manual database setup, providing a fully reproducible testing environment. SonarCloud and Coveralls integration highlights code quality, test coverage, and maintainability, ensuring that potential code smells, vulnerabilities, and technical debt are minimized

# << THANK YOU >>
# <<THE END>>