

Smart Room Application

Team 2

Nuray Seker, Elma Buljina, Abir Sikder, Stefan Pilgerstorfer

Project  
documentation

Content

[Introduction 3](#_Toc126962042)

[Management Perspective 4](#_Toc126962043)

[Abir:. 4](#_Toc126962044)

[Elma: 5](#_Toc126962045)

[Stefan: 6](#_Toc126962046)

[Nuray: 7](#_Toc126962047)

[User Perspective 9](#_Toc126962048)

[Developer Perspective 12](#_Toc126962049)

[Design 12](#_Toc126962050)

[Overview of the System 12](#_Toc126962051)

[Important Design Decisions 15](#_Toc126962052)

[Implementation 15](#_Toc126962053)

[Code Quality 16](#_Toc126962054)

[Testing 17](#_Toc126962055)

[Installation Guide 18](#_Toc126962056)

Version history

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Creator | Changes |
| 0.1 | 2023-02-09 | Abir Sikder | Initial |
| 1.0 | 2023-02-10 | All | End Delivery |
|  |  |  |  |
|  |  |  |  |

Table 1: Version history

# Introduction

"Smart Room" is an advanced application designed to serve as a digital representation of a physical room. It enables manual control of various room components, including doors, windows, lighting, and fans. Equipped with sensors, the application not only detects the number of people present in the room, the CO2 level, and room temperature, but it also automatically adjusts the room's facilities based on specific conditions, such as the presence of people. The app doubles as a display and graphic visualizer of crucial data, including rooms, room facilities, temperatures, CO2 levels, and number of people. Room information can be added manually or via a CSV file and is stored in a database that the application can access.

With "Smart Room," room management and control have never been easier.

This app offers an opportunity to optimize and improve the comfort of rooms. It provides a complete overview of all relevant room data and allows for real-time monitoring and control. The integration of sensors and automatic adjustments based on specific conditions ensure a comfortable environment without the need for constant attention. Monitoring and managing a room is simple because to the data's visual display, and adding data manually or via a CSV file enables for customization to suit different needs. You have a complete solution for maintaining and keeping an eye on your rooms with "Smart Room."

This documentation provides a comprehensive overview of the development process of a system from various perspectives including Management, User, and Developer.

The Management Perspective section focuses on the requirements that were implemented and provides information on which team member was responsible and the number of hours they spent on implementation. The section also includes explanations for any requirements that have not been implemented.

In the part titled "User Perspective," which includes screenshots of the user interface and descriptions of the functionality based on various scenarios, it is explained how the requirements were translated into the user interface. The part also outlines the proper system usage for users.

The Developer Perspective section is divided into four sections: Design, Implementation, Code Quality, and Testing. The Design section provides an overview of the system, including a UML diagram with explanations, and the most important design decisions that were made during the development process. The section includes information on the decision, reason, considered alternatives, assumptions, and effect.

The implementation part includes details on the project structure, dependencies, libraries used, and a few carefully chosen lines of code. It also describes the key elements of the implementation. The Code Quality section includes information on the use of PMD and a description of the findings, as well as which findings have been fixed.

The Testing section provides an overview of the JUnit tests that were created and includes a description of three selected tests, including the test case ID, designer, execution information, requirements, test steps, test data, expected result, post-condition, status, and comments.

Finally, the Installation Guide section provides instructions on how to install and start the system, including a link to the GitHub repository.

# Management Perspective

## Abir:

* API: 60h
  + AirQuality
  + TempertureSensor
  + HumiditySensor
  + Co2Sensor
  + Ventilator
  + Window
  + LightSource
* API Client: 30h
  + Window
  + Humidity
  + Ventilator
  + Temperature
  + LightSource
* Automation Rules: 5h
  + Turn lights on/off depending on number of people in room
  + Unlock all doors if temperature is above 70 degrees
* Unit Tests: 10h
  + TemperatureSensor
  + TemperatureSensorRecords
  + HumiditySensor
  + HumiditySensorRecords
* UML: 2h
* ER: 5h
* Use Case: 5h
  + 16-22
* JavaDoc: 5h
* Projectdocumentation: 5h
  + Management Perspective
  + Code Quality
  + Overview of the System
  + Testing

## Elma:

* API: 6h
  + putAirQuality
  + API extension
* GUI: 51h
  + Edit Room (Release 2)
  + Create Room (Release 2)
  + ImportController
  + RoomTemperatureChart
  + NumberOfPeopleChart
  + Co2Chart

* Unit Tests: 28h
  + Co2Sensor
  + Ventilator
  + VentilatorRecords
  + PeopleInRoom
* ER: 3h (vor Gruppenwechsel)
* Projectdocumentation: 2,5h
  + Introduction
  + Management Perspective

## Stefan:

* Project Structure: 20h
  + Database Design (ER & UML)
* Database Design and Creation: 30h
  + AWS Creation
  + Firewall Rules
  + Database Testing
  + Database Model and Relations
* API: 30h
  + RestController
  + Door
  + Room
  + PeopleInRoom
* Automation Rules: 5h
  + Open window + activate fan if co2 values are > 1000 parts per million (ppm)
  + Automation rule: Turn off running devices if the room is empty.
* Simulator (Random Values): 10h
* Unit Tests: 5h
  + Co2SensorRecords
  + LightSource
* Use Case: 5h
  + Use Case 9-15
* JavaDoc: 10h
* Code Quality 10h
* Project documentation: 3h
  + Management Perspective
  + Implementation
  + Installation Guide

## Nuray:

API: 5h

Door API: 5h

GUI: 120h

* Homepage: 30h
  + Remote Control
  + Static Information
  + Co2/Temp/Number of People
* NewRoomController: 20h
  + New Room
  + Add devices to new Room
* EditRoomController: 40h
  + Name
  + Size
  + Devices – Add, change values, delete
* AllRoomsController: 30h
  + Delete
  + Export function

**Not (completely) implemented:**

* Not all planned tests were realizable.
* Values of diagrams are missing.
* All found bugs are fixed, but most likely there are still some left.

**Total Hours (with Team Meetings and lecture hours):**

**2022:**



Figure 1: Clockify 2022

**2023:**



Figure 2: Clockify 2023

**Sum:**

Abir: 62+69 = 131h  
Elma: 65+88 = 153h  
Nuray: 69+61 = 130h  
Stefan: 86+42 = 128h

# User Perspective

The basic user interface shows the selected room. Powerable and Openable devices can be turned on or off, opened or closed by using the sliders. The actual state in form of parameters like room temperature, co2level, number of people and room size are displayed in the right.

In addition, the number of enhancements of the room is displayed at the right bottom. (rooms, windows, …)

With the Pencil button besides “Welcome”, the Room can be edited. Selecting change name, allows a personalisation of the name displayed in the GUI.

With a click on refresh, all server data is reloaded and displayed in the GUI.

Graphical user interface, table

Description automatically generated

Figure : Homepage

If the user clicks on new room, a new room can be created with a variable amount of doors, windows, fans and light sources. Those are automatically created with default values.  
A room name can also be specified.

Graphical user interface, application

Description automatically generated

Figure 4: Create new room

Selecting all rooms from the left navigation takes the user to an overview of all rooms. Rooms can be deleted or focusses, which takes the user to the initial application main window (Figure 1), with a different room selection and parameters. With a click on Export, a csv file can be exported.

Graphical user interface, application

Description automatically generated

Figure 5: All rooms

A picture containing icon

Description automatically generated

Figure : Automation Rule: Color based on Co2-Values

This part of the GUI gets coloured based on the actual values. (RED, GREEN, BLUE, as described in requirements)

# Developer Perspective

## Design

### Overview of the System

Our project is modelled in three layers, the client, server, and simulator.

**Client-UML:**

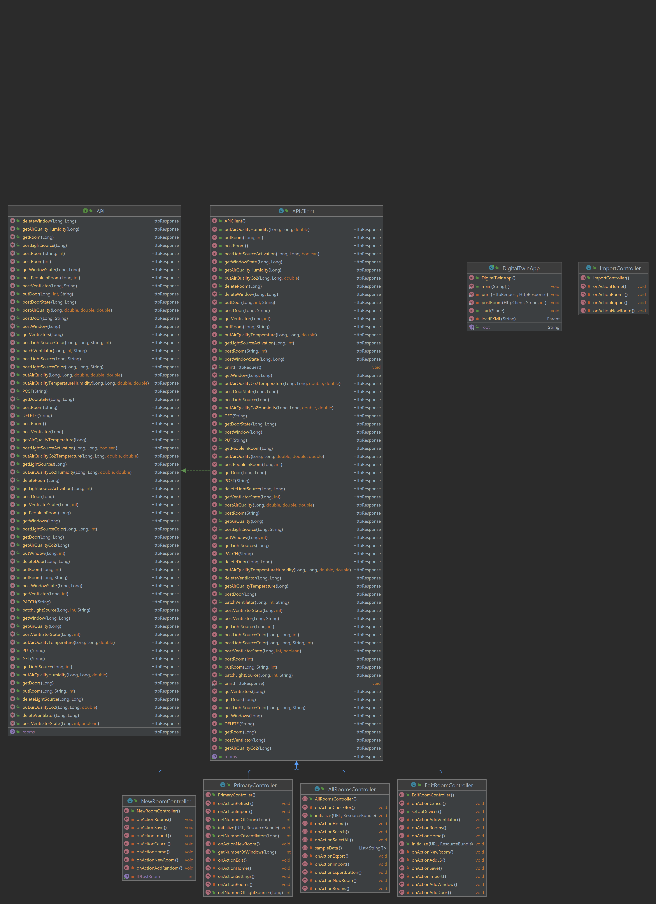


Figure 7: Client UML

The HTTP Responses are implemented in the API Client, which are based on the CRUD commands implemented in the RestController. Additionally, four Controllers are connected with the API Client. the PrimaryController, AllRoomsController, EditController and the NewRoomController.

**Server-UML:**

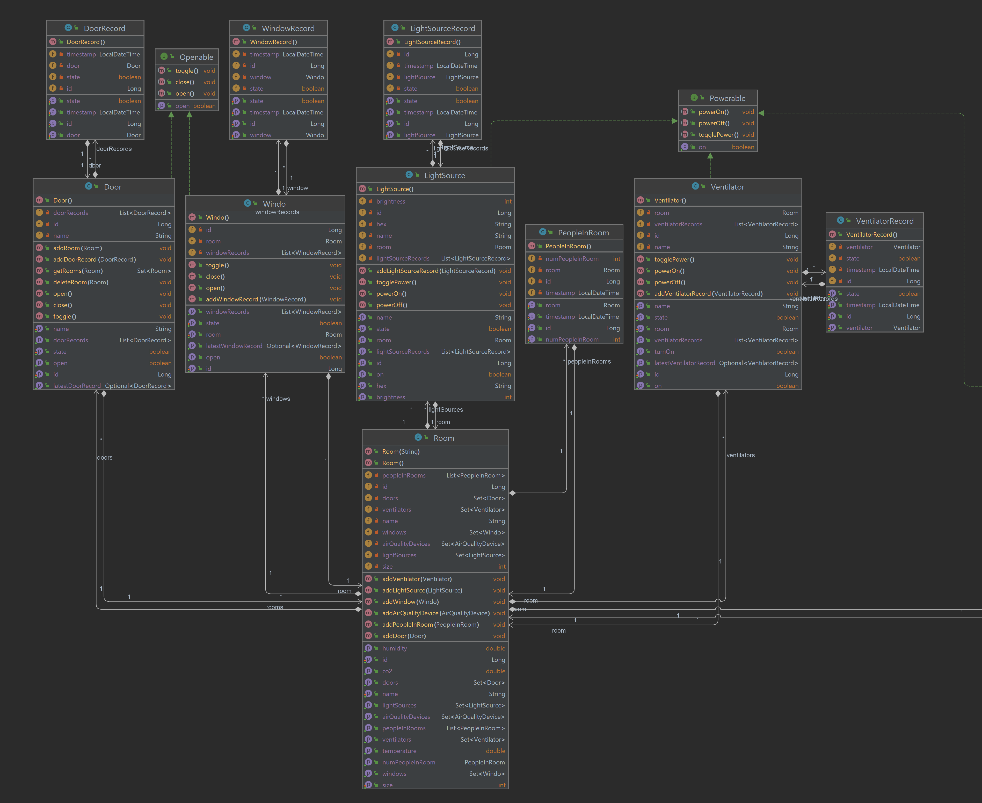


Figure 8: Section of the Server-UML Diagram (Room)

In order to program the API interfaces, a few classes were required first. In this part of the Server-UML Diagram, you can see the relationships for the class „Room“. A room can have multiple doors, windows, lights, and fans. Doors and windows are connected via the "Openable" interface, which handles methods for opening and closing these doors and windows. Additionally, lights, fans, and the AirQualityDevice (see Figure 8) are connected via the "Powerable" interface, which is similar to the "Openable" interface and requires methods for turning the lights, fans, and AirQualityDevice on and off.

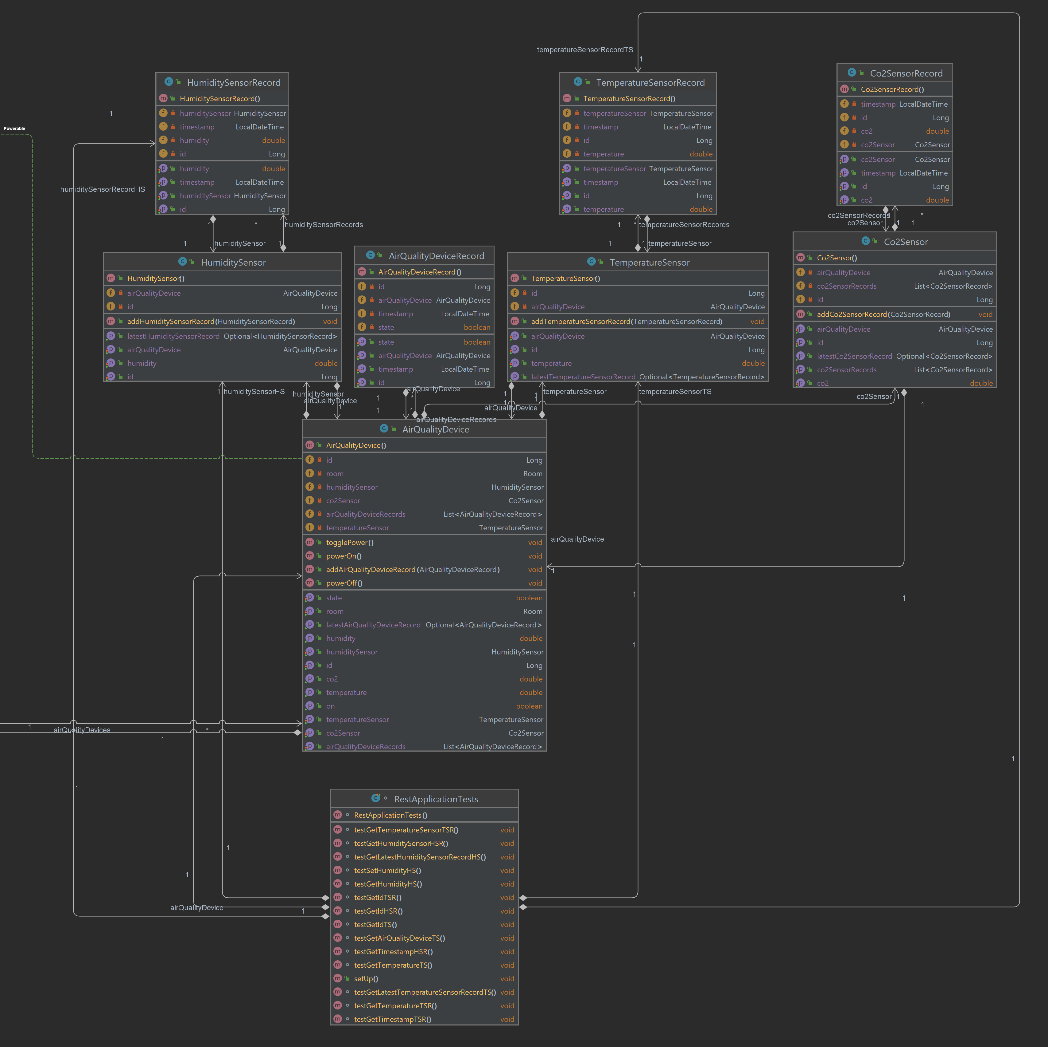


Figure 9: Section of the Server-UML Diagram (AirQualityDevice)

In this part of the Server-UML Diagram you can see the AirQualityDevice, which can have multiple individual sensors, including temperature, humidity, and a CO2 sensor. Unit tests for the server are implemented in the "RestApplicationTests" class.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 10: Section of the Server-UML Diagram (RestController)

The API interfaces that are also used in the client are managed in the "RestController" class.

CRUD commands for the individual classes have been implemented here.

**Simulator-UML:**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 11: Simulator-UML

The HttpResponses are implemented in the API Client, which are based on the CRUD commands implemented in the RestController. Unit tests for the simulator are implemented in the "SimulatorTests" class.

Design patterns used: Model-View-Controller

# Important Design Decisions

Decision: Database MySQL

Reason: good implementation in JAVA, Maven and Hibernate Framework

Considered Alternatives: PostgreSQL

Assumptions: AWS Platform, Integration Complexity

Effect: good portability to other DBMS, free licence open source

--------------------------------------------------------------------------------------------------------------------------------

Decision: Hibernate Framework

Reason: automated CRUD operations

Considered Alternatives: JDBC

Assumptions: automated generation of DB Scheme when relations are properly defined

Effect: faster migration to different hosting/DBMS

--------------------------------------------------------------------------------------------------------------------------------

Decision: Spring Framework

Reason: automated JSON generation

Considered Alternatives: manual Data transfer processing

Assumptions: easily extensible

Effect: more initial workload but when configured easily extended

# Implementation

Since the requirement was a REST API we used the java spring, hibernate, JSON, MySQL, JUnit, Mockito and some more frameworks/plugins. See the generated (parent) pom.xml or the corresponding pom.xml of the modules for more details. We structured our project in four modules:

* Client: GUI
* Server: JSON REST API
* Simulator: Used mostly for testing
* Hasher: password generation for sprint security, we set rules for all allowed, but a security version with authentication for the rest API is easily implemented

Client and Simulator are treated as clients. We implemented an interface class and an API client class, which together can be used to implement new clients to communicate with the server in an easy way. Just extend the API Client class and all communication methods can be used with minimal parameters.

Text

Description automatically generatedAs database we decided to use a web hosted version of MySQL. We use Amazon AWS free version to work on the same data. If another DBMS should be used, this is easily possible, by just changing the database server in resources/application.properties of the server.

Figure 12: Database Properties

If a new database (server) is selected, the complete data structure is automatically created, and the application can be run.

The Simulator is used for real data testing and can test the underlying database with matching return values. In addition, testing is done by the Rest Application Tests where data is mocked and the functionality of the java classes is tested.  
For detailed implementation of the data structure please refer to the UML diagrams.

# Code Quality

We used SonarLint and corrected hundreds of code quality issues, which were mostly best practice tips. The range was from simplification tasks to remove side effects, …

Text

Description automatically generatedThere are no more code issues residing besides three code complexity issues which were structurally not correctable.

Figure 13: SonarLint (RestController)

The last listed one was enforced by the given API.

# Testing

Overview of created JUnit tests:

* See code (via GitHub)

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungWe created two parts of Tests, the first using Mock Data to check JAVA classes. The second part covers the check of the server and its returning values.

Figure 14: Unit Tests with Mock Data

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 15: Code Coverage

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 16: API-Server and Database Test (sample)

# Installation Guide

Standard Installation for developers by cloning the GitHub repository and use IntelliJ or eclipse:

* start server/DigitalTwinServer
* start client/DigitalTwinApp
* if you want to use the simulator, start simulator/Simulator
* for spring security you can use the hasher/HasherApp to generate a password  
  The permissions in server/security/SecurityConfiguration have to be adapted

Link to GitHub source code (repository):

Text

Description automatically generatedhttps://github.com/jku-win-se/teaching.ws22.prse.smartroom.Team2

Figure 17: Maven install

A screenshot of a computer screen

Description automatically generated with medium confidenceAfter installing the application under the corresponding module folders/target the executable jar files are created.

Figure 18: created executable .jar file

These can be executed by using the following java command:

Text

Description automatically generated

Figure 19: execute server jar

Text

Description automatically generated

Figure 20: server running

Same procedure for client/simulator/hasher:

Text

Description automatically generated

Figure 21: execute client jar

Graphical user interface

Description automatically generated

Figure 22: Client and Server running

A picture containing graphical user interface

Description automatically generatedSimulator:

Figure 23: Simulator

Text

Description automatically generatedHasher (for optional spring security):

Figure 24: Hasher