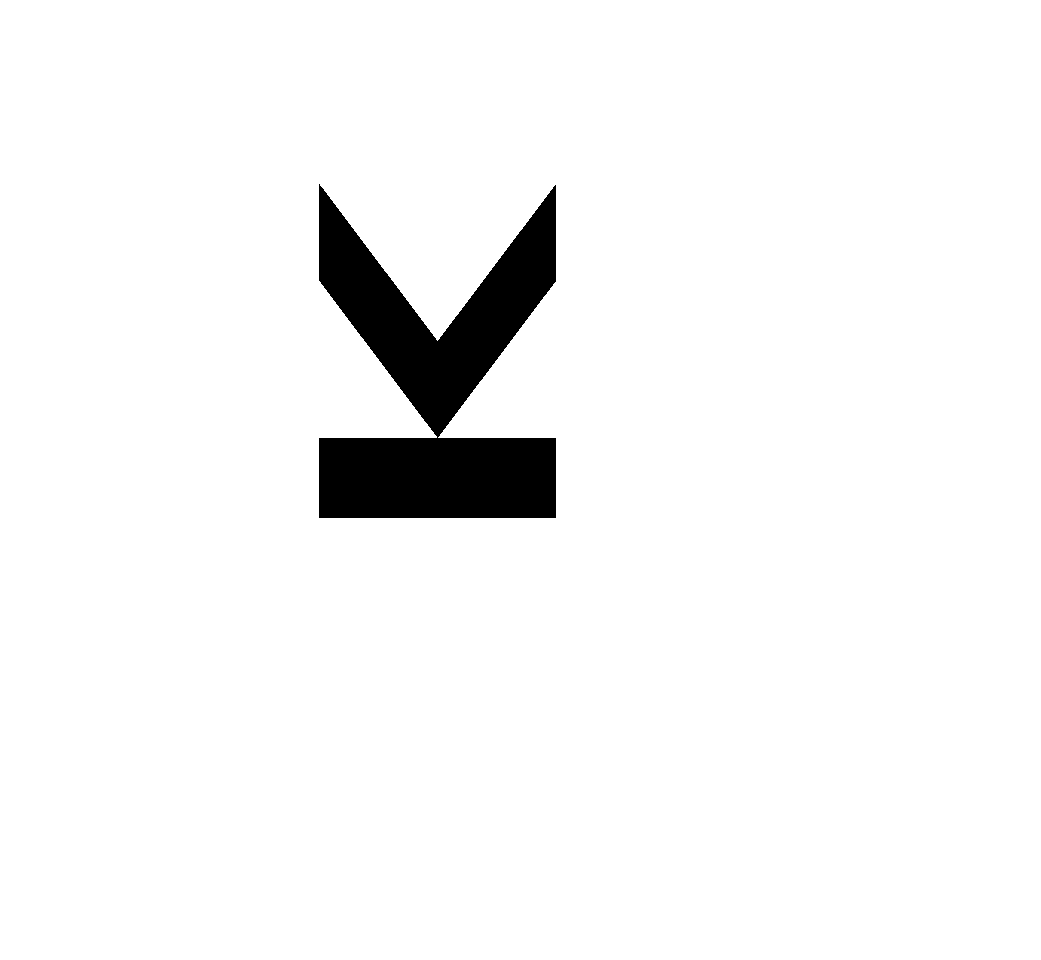
06/2022



Project  
documentation

Smart Room Application

Team 1

Florian Dobretsberger, Matthias Herzog, Petra Körper,

Markus Mühleder

**Table of Contents**

[1. Introduction 3](#_Toc107991208)

[2. Implemented Requirements 3](#_Toc107991209)

[3. Overview of the System from the User´s Point of View 5](#_Toc107991210)

[4. Overview of the System from the Developer´s Point of View 7](#_Toc107991211)

[4.1. Design 7](#_Toc107991212)

[4.1.1. Overview of the System 7](#_Toc107991213)

[4.1.2. Important Design Decisions 8](#_Toc107991214)

[4.2. Implementation 9](#_Toc107991215)

[4.3. Code Quality 12](#_Toc107991216)

[4.4. Testing 13](#_Toc107991217)

[5. Installation guide 14](#_Toc107991218)

**Version history**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Creator | Changes |
| 1.1 | 15.06.2022 | Dobretsberger | added Req-Matrix |
| 1.2 | 16.02.2022 | Dobretsberger | overview User-Int. |
| 1.3 | 17.06.2022 | Dobretsberger | added Reports |
| 1.4 | 01.07.2022 | Dobretsberger | added graphics |
| 1.5 | 04.07.2022 | Dobretsberger | Updated Req-Matrix |
| 1.6 | 04.07.2022 | Herzog, Körper | Requirements |
| 1.7 | 04.07.2022 | Dobretsberger | Added Test-Descriptions |
| 1.8 | 05.07.2022 | Dobretsberger | Review |

Table 1: Version history

# Introduction

This document describes the most important parts of the smart room application in terms of design and the implementation of the requirements. Furthermore, it should help to understand the relationship between the used technologies and interfaces.

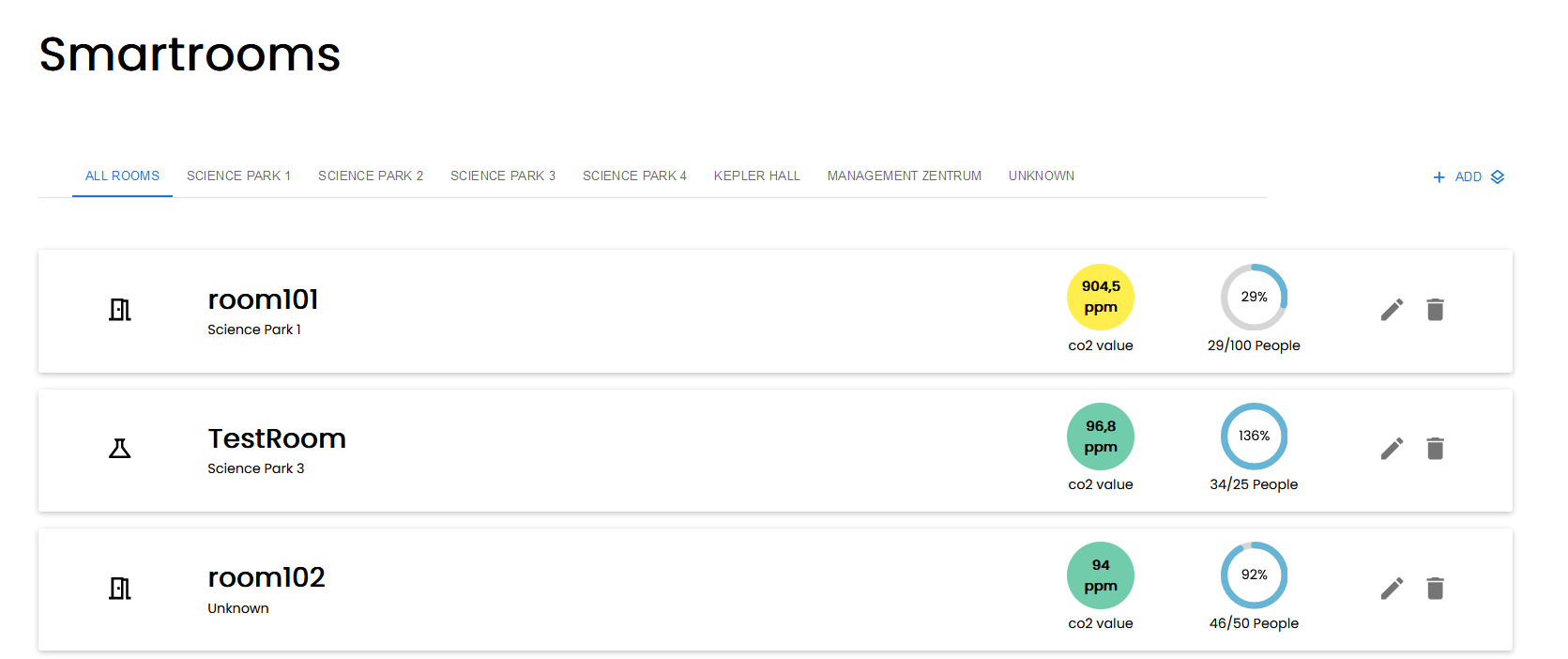
# Implemented Requirements

All requirements have been implemented successfully within three sprints.

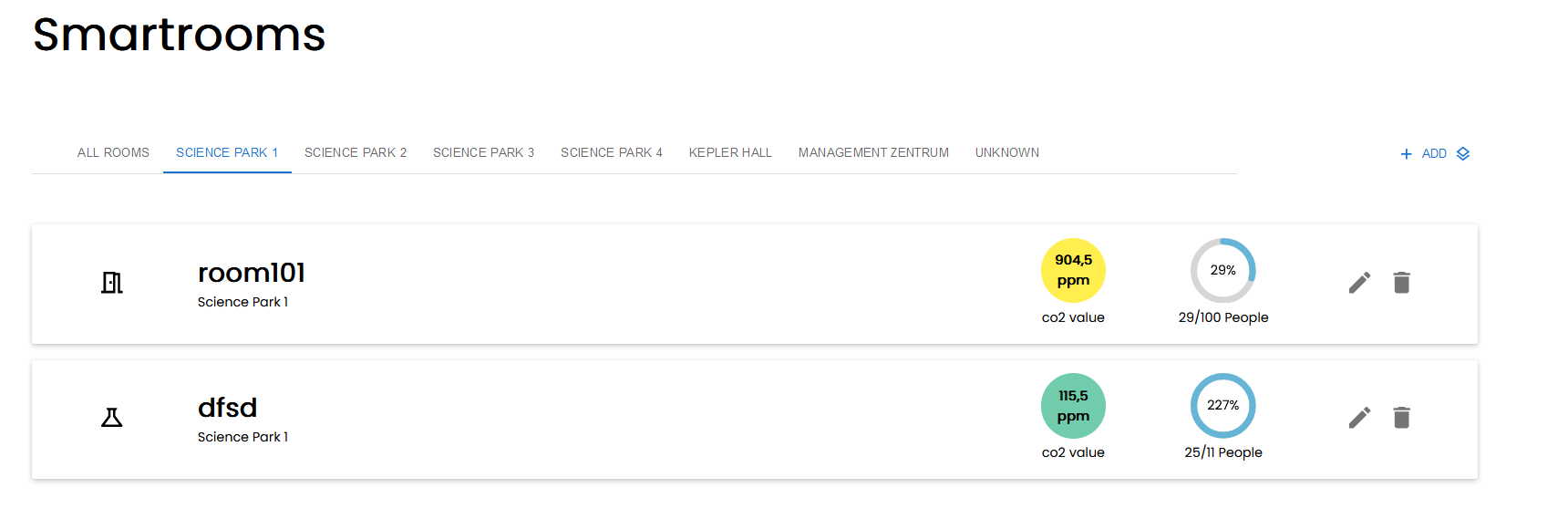
First of all, we have to point out that some prerequisites in terms of preparation where necessary to meet all requirements. These tasks were documented in zen-hub. https://app.zenhub.com/workspaces/se-praktikum-62345aeefc8e8c00125ad915/board

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Requirement | Responsible | Effort in SP |
| 1 | Basic: Import data from CSV to an Entity-Relationship database. | Markus Mühleder  Florian Dobretsberger | BE #37 (21 SP) |
| 2 | Basic: Create/Update Rooms (id, size, available doors, windows, lights and fans). | ALL | FE #23 (13 SP)  BE #49 (5 SP)  BE #36 (8 SP)  FE #47 (5 SP)  FE #46 (5 SP) |
| 3 | Basic: Update and remove rooms. | ALL | FE #25 (3 SP)  BE #49 (5 SP)  FE #45 (3 SP) |
| 4 | Basic: Visualize available rooms. | Petra Körper  Matthias Herzog | FE #24 (21 SP)  FE #31 (3 SP)  FE #22 (5 SP)  FE #43 (5 SP) |
| 5 | Basic: Visualize static information for each room (id, size, available doors, windows, lights and fans). | Petra Körper  Matthias Herzog | FE #32 (2 SP)  FE #26 (8 SP) |
| 6 | Basic: Develop a line chart that shows real-time data regarding light/fan/window/door. | Petra Körper  Matthias Herzog | FE #28 (13 SP) |
| 7 | Basic: Develop a line chart that shows the co2/temperature values and the number of people for each room over time. | Petra Körper  Matthias Herzog | FE #29(3 SP)  FE #30(5 SP) |
| 8 | Basic: Save rooms structure (rooms + static information) in a .csv file. | Markus Mühleder  Florian Dobretsberger | BE #37 (21 SP) |
| 9 | Basic: Automatically add random values of co2/temperature/number of people for a specific room. | Markus Mühleder | BE #41 (13 SP) |
| 10 | Basic: Live update of visualizations for co2, temperature, and lights/ventilators/windows/doors status for each room. | Petra Körper  Matthias Herzog | FE #48 (13 SP) |
| 12 | Remote Control: Allow to lock/unlock doors via the user interface. | Petra Körper  Matthias Herzog | FE #27 (13 SP) |
| 13 | Remote Control: Allow to turn on/off lights via the user interface | Petra Körper  Matthias Herzog | FE #27 (13 SP) |
| 14 | Remote Control: Allow to open/close windows via the user interface. | Petra Körper  Matthias Herzog | FE #27 (13 SP) |
| 15 | Remote Control: Allow to turn on/off fans via the user interface. | Petra Körper  Matthias Herzog | FE #27 (13 SP) |
| 16 | Security: Send alarm if temperature is above 70 degrees celsius. | Markus Mühleder  Florian Dobretsberger | BE #59 (5 SP) |
| 17 | Security: Unlock all doors if temperature is above 70 degrees celsius. | Markus Mühleder  Florian Dobretsberger | BE #59 (5 SP) |
| 18 | Energy Saving: Turn lights on if there are people in the room. | Markus Mühleder  Florian Dobretsberger | BE #57 (5 SP) |
| 19 | Energy Saving: Lights should be turned off if the room is empty. | Markus Mühleder  Florian Dobretsberger | BE #57 (5 SP) |
| 20 | Energy Saving: Turn off running devices if the room is empty. | Markus Mühleder  Florian Dobretsberger | BE #57 (5 SP) |
| 21 | Air Quality: Open window + activate fan if co2 values are > 1000 parts per million (ppm). | Markus Mühleder  Florian Dobretsberger | BE #58 (5 SP) |
| 22 | Air Quality: Change room color in user interface based on co2 values.  - green if c2o values are < 800 ppm  - yellow if co2 values are between 800 and 1000 ppm  - red if co2 values are above 1000 ppm | Petra Körper  Matthias Herzog | FE #24 (21 SP)  FE #26 (8 SP) |

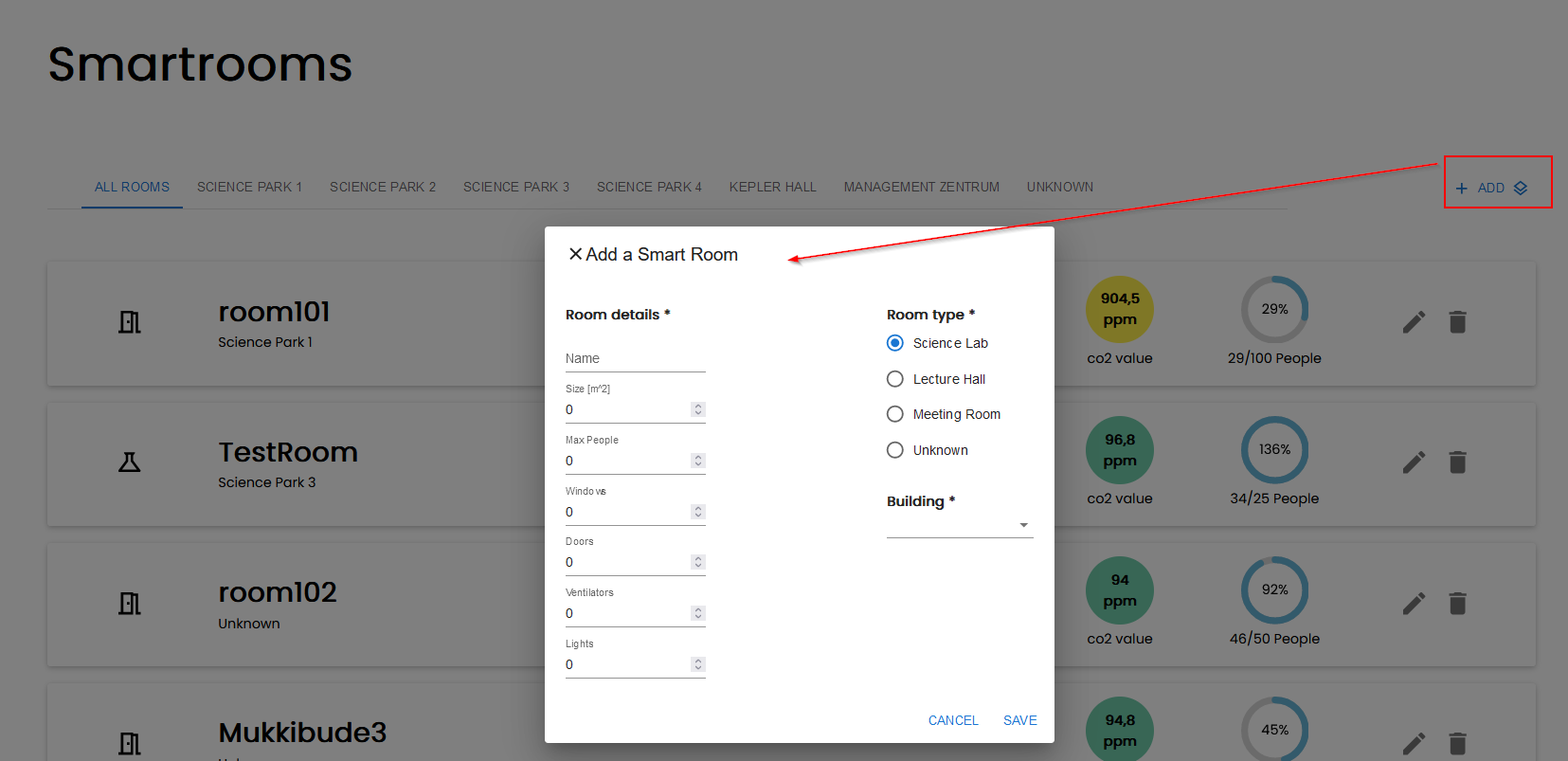
# Overview of the System from the User´s Point of View

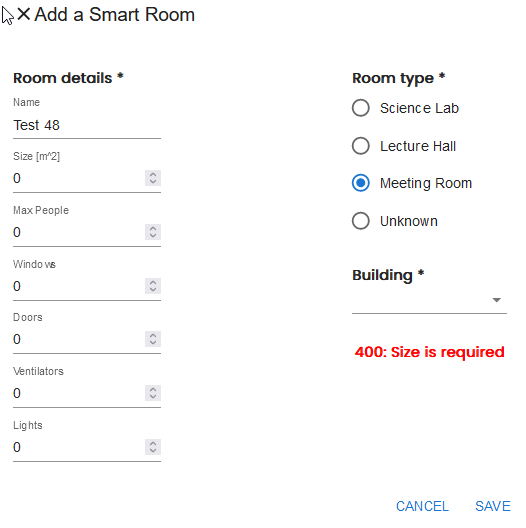
The Mainpage initially displays all rooms, which are available within the system. The user is able to switch between the buildings via the menu. Furthermore, the current co2 value and the capacity-state regarding the appropriate room are being displayed automatically.

Rooms filtered by building:

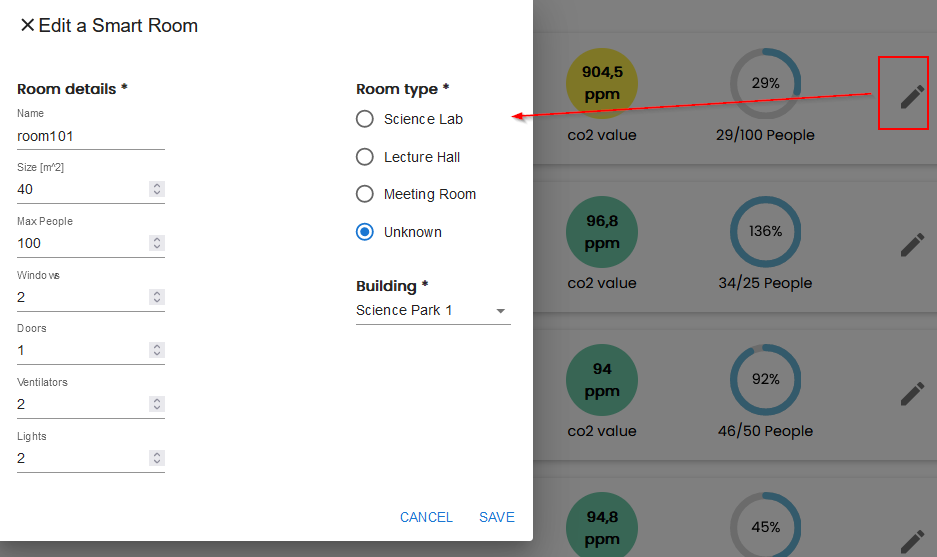


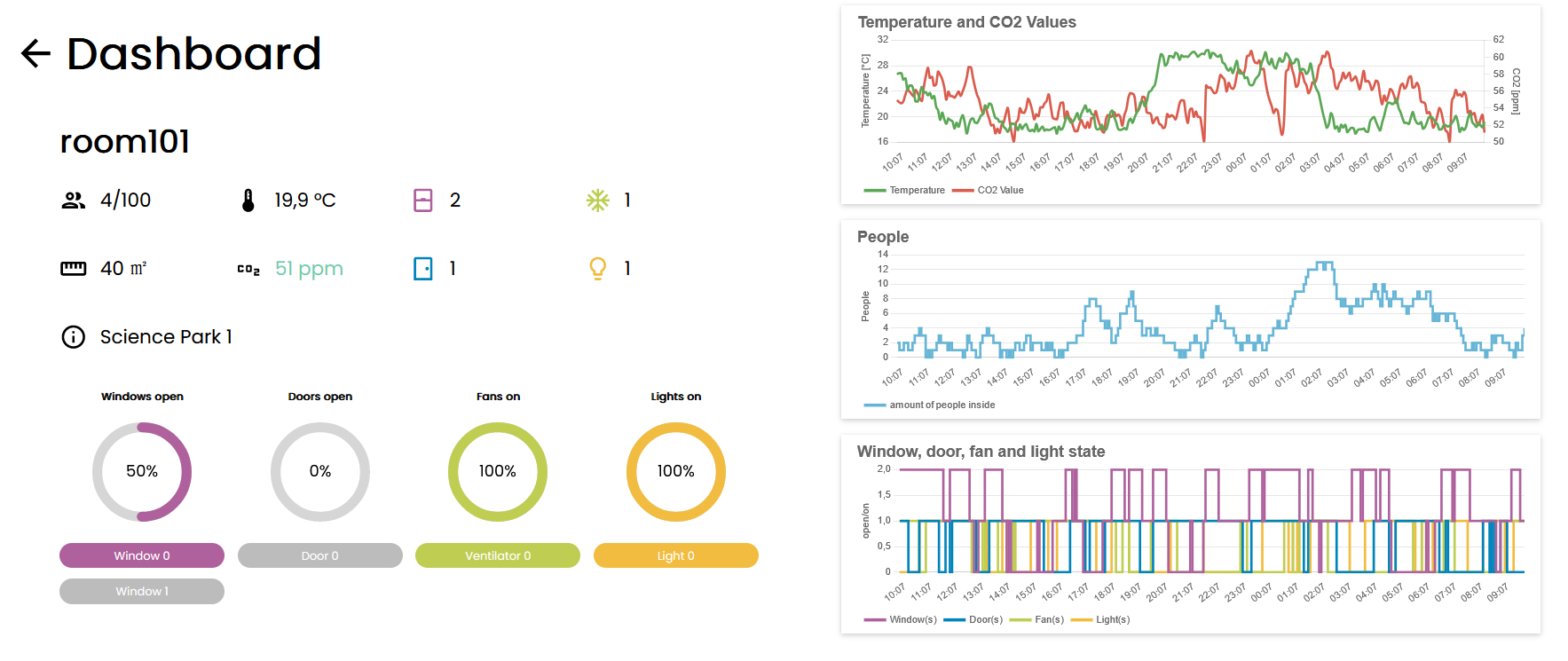
The User is able to create a new room via the ADD Button. A dialog appears and the user needs to fill in all necessary details regarding the room.





In case of wrong input the user gets a notification in terms of the related error. The user is able to edit an existing room via the edit-button.





By clicking on a room-section on the overview the user gets a detailed page of the predefined properties. Moreover, the user can control the state of windows, doors, fans and light by clicking on the appropriate button.

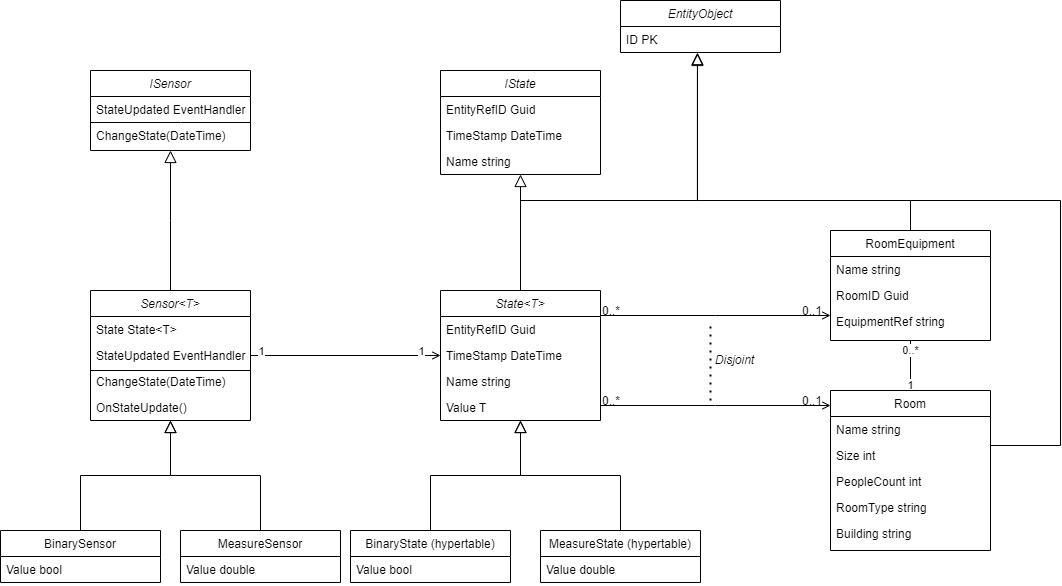
# Overview of the System from the Developer´s Point of View

## Design

### Overview of the System

*UML Diagram with explanations*

*Design patterns used (e.g. Model-View-Controller)*



### Important Design Decisions

*Decision: React Framework*

*Reason: well maintained (because facebook is the main contributor), experience within the team*

*Considered Alternatives: Angular, Vue.js*

*Assumptions: lightweight and freedom to chose fitting dependencies, faster development through experience in contrast to other frameworks.*

*Effect: it worked as expected*

*Decision: Material UI*

*Reason: up to date with the design, preconfigured components*

*Considered Alternatives: create the components by hand*

*Assumptions: easy to use and time saved for the other parts of the implementation*

*Effect: it worked as expected*

*Decision: .Net 6*

*Reason: pre-existing knowledge in C#, .NET 6 and needed frameworks for the project*

*Assumptions: saving time by using frameworks we know*

*Effect: we had time to focus on software architecture, tools, and project management*

*Decision: Micro Service Architecture*

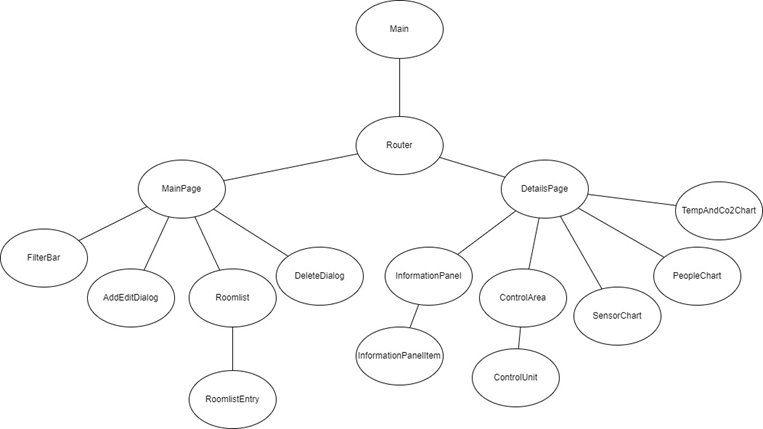
*Reason: for a more efficient use of resources, an increase of availability and a reduction of dependencies between functionalities and project members*

*Assumptions: to get the maximum out of free cloud resources and make parallel working less complex*

*Effect: CI/CD got more complex, parallel working was more easy, fast deployment of smaller parts for the UI, better visibility of the progress*

## Implementation

**Frontend Component-Tree:**



The implementation is based on the concept of a master detail page because the requirements stated the need for a list-based overview of the rooms. This page is represented through the main page which gives a basic overview of the rooms, every room links to the according details page. The details page contains the dashboard for the selected room and displays more specific information than the main page. Through this separation every page is clean and gives a better overview of the data.

Ein Bild, das Text enthält.

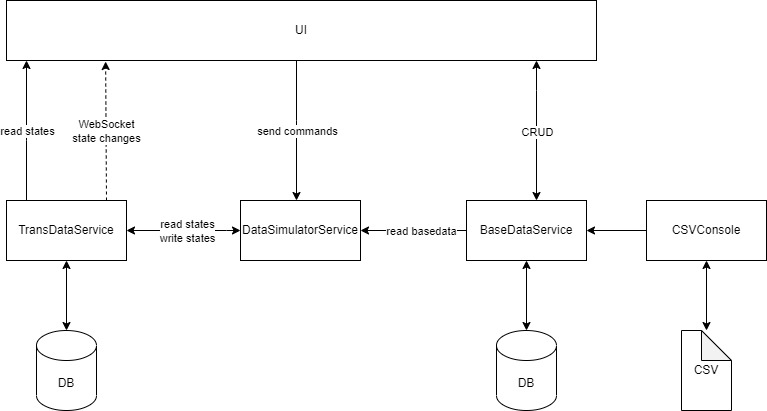
Automatisch generierte BeschreibungComponents which are used in the master and the details page are extracted, to avoid code duplication. Furthermore, functions which are used more than once are also extracted into a utils folder, so they are accessible for every page. The communication with the backend is structured into two different services: room and state; which allows the use of the singelton pattern. Through the singelton pattern there are no inconsistencies.

Ein Bild, das Text, Screenshot, computer enthält.

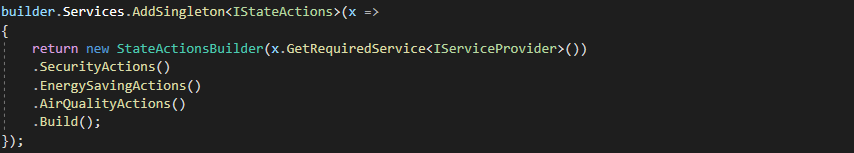
Automatisch generierte BeschreibungWith the help of props it is possible to customize one component as a template so it can be reused in different scenarios like the room-list-item, the control-unit and the information-panel-item. Props are like parameters in a function, they change the output.

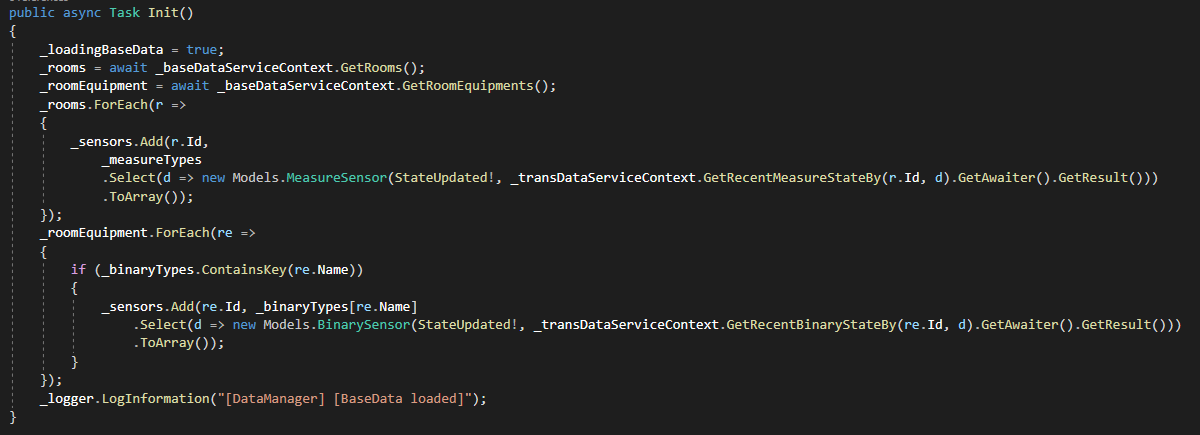
One might ask the question why this was not used with the charts as well. On the one hand there where restrictions from the backend, like what functions needed to be called, and on the other hand, due to the library which was used (Chart.js) and how it is designed, it would have resulted in too many props. Therefore, it is necessary to know when to generalize and when not to.

**Service-Architecture:**

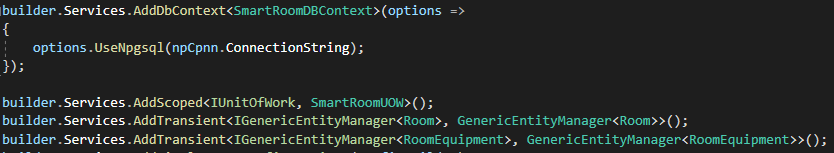


The illustration above shows the data flow between our services. We divided our backend into 4 services.

* The **TransDataService** handles all sensor data which we call trans data. The Service provides a read and write interface, aggregated queries included. Also a web socket is implemented with SignalR, so the frontend can subscribe to the data changes and alarms. For the different mechanisms like air quality, we have implemented a builder pattern for a clean decapsulation. As the following snippet show the builder was easy to use in the main program: 
* The DataSimulatorService simulates real sensors and actors. The services supply interfaces to trigger actors and to check the status of the simulator. For the data generation the simulator reads at the start the latest trans data and base data and creates sensor instances. These instances generate data random between 0.01 and 1 minute. If the data gets updated, the service sends the new data to the TransDataService. The Following code snippet shows the method which creates the instances of the sensors.



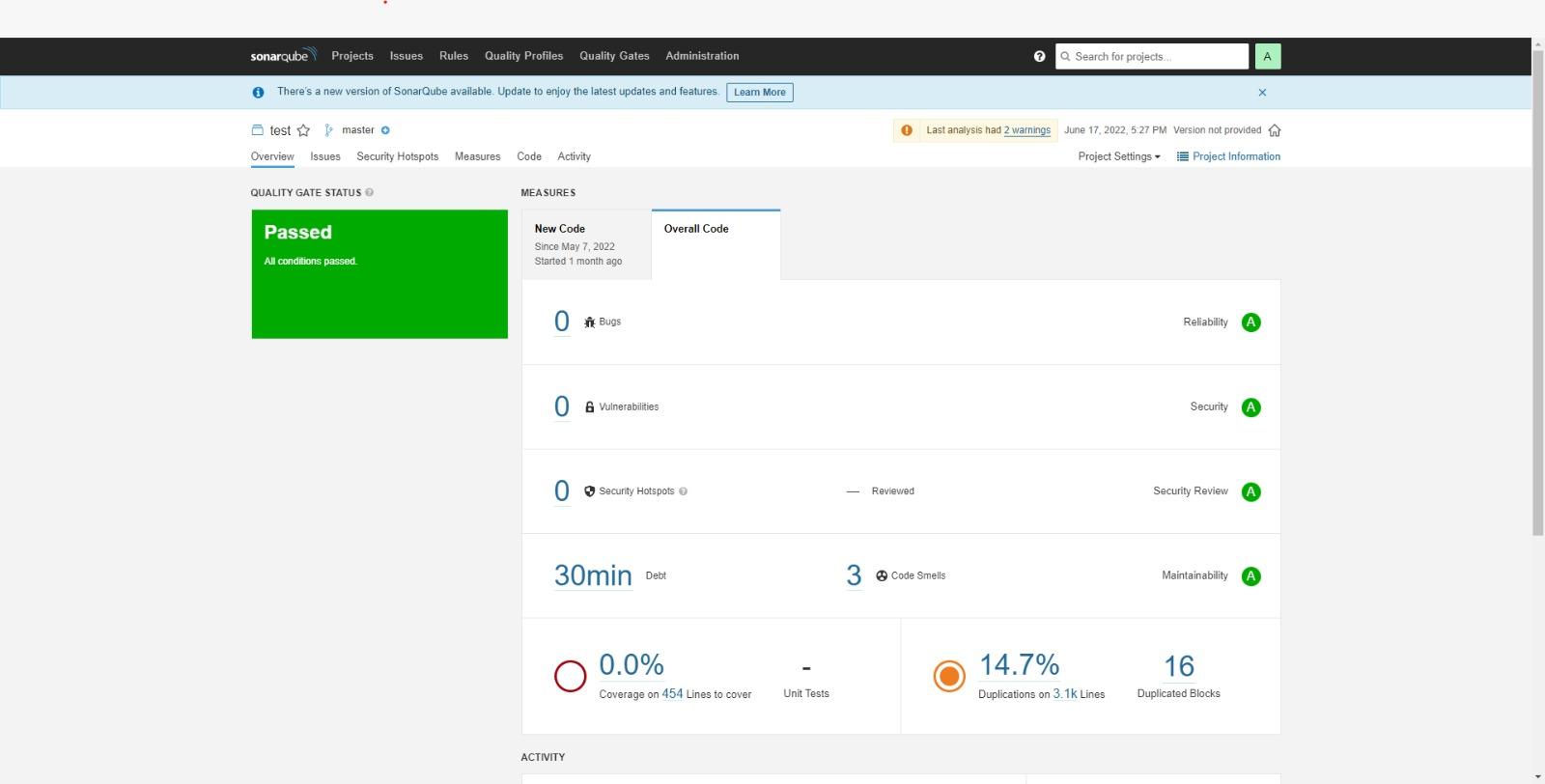
* The BaseDataServcie provides the CRUD operations for static data (rooms, room equipment’s, …). This service uses a small Postgres SQL instance. The Following code snippet shows the usage of our generic CRUD pattern in the main program.



* The CSVConsole is a console application which im- and exports the static data.

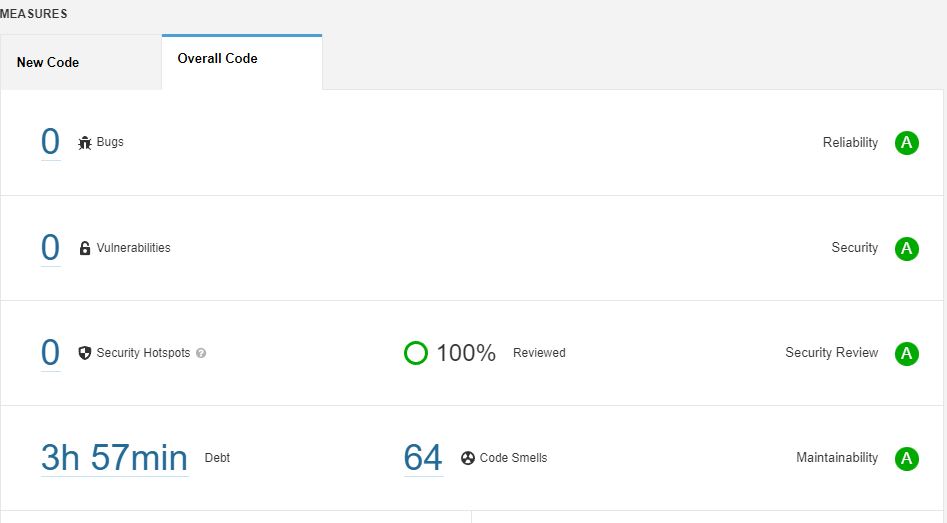
## Code Quality

**Code Quality Frontend**



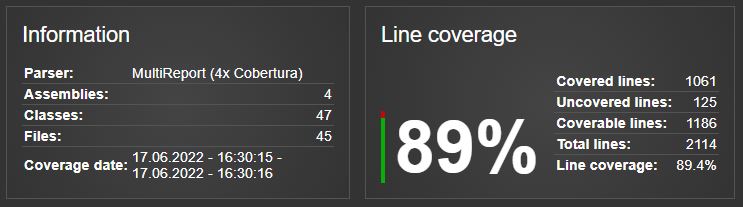
Due to the inner workings of Sonarqube there is a high amount of code duplication. This comes from the import statements and the HTML elements which are used more than once. Additionally, there were a few code smells which have been fixed easily. As an example, “use different variable names for different scopes”. This is not mandatory in JavaScript but it is considered bad practice.

**Code Quality Backend**



## Testing

*Overview of created JUnit tests (it can be described a few selected tests)*



Example #1:

|  |  |
| --- | --- |
| Test Case ID | Ctor\_ValidFileName\_GetFileName() |
| Designed by | Florian Dobretsberger |
| Execute on | 24.06.2022 |
| Carried out by | Florian Dobretsberger |
| Tested Requirement | Check if constructur GenericCSVWriter<Object> initializies a new csv-file |
| Requirement | Creating a new csv-file |
| Test steps | var writer = new GenericCSVWriter<Object>(new List<Object>(), "Test.csv"); |
| Test data | Test.csv |
| Expected result | Filename exists: Assert.Equal("Test.csv", writer.FileName); |
| Post condition | CSV file with example data |
| Status | Test passed |
| Comments | - |

Example #2:

|  |  |
| --- | --- |
| Test Case ID | Room\_PropertyNames\_Exist(string name) |
| Designed by | Markus Mühleder |
| Execute on | 08.06.2022 |
| Carried out by | Florian Dobretsberger |
| Tested Requirement | Check, if room properties are existing and names have not changed. |
| Requirement | Visualize static information for each room (id, size, available doors, windows, lights and fans) |
| Test data | “” |
| Test steps | [Theory]  [InlineData("Name")]  [InlineData("PeopleCount")]  [InlineData("Size")]  [InlineData("RoomType")]  [InlineData("Building")]  [InlineData("RoomEquipment")]  [InlineData("Id")]  public void Room\_PropertyNames\_Exist(string name)  {  Assert.NotNull(typeof(Room).GetProperties().First(p => p.Name.Equals(name)));  } |
| Expected result | All properties are not null. |
| Post condition | Unchanged |
| Status | Test passed |
| Comments | Important to visualize changes at the web interface |

Example #3:

|  |  |
| --- | --- |
| Test Case ID | Decrypt\_ValidEncString\_ExpDecryptedString() |
| Designed by | Markus Mühleder |
| Execute on | 26.05.2022 |
| Carried out by | Florian Dobretsberger |
| Tested Requirement | Test encryption and decryption |
| Requirement | Correct decyption of encypted data. |
| Test steps | public void Decrypt\_ValidEncString\_ExpDecryptedString()  {  var cypher = new Aes256Cipher(\_key);  var encrString = "XrOYnGAPkoTh4lB5zRdAAMWOEwZMgqD6kq7tXdI9JB5NhkL9khk/O6klzgBLLs9h";    Assert.Equal(\_decrString, cypher.Decrypt(encrString));  } |
| Test data | "XrOYnGAPkoTh4lB5zRdAAMWOEwZMgqD6kq7tXdI9JB5NhkL9khk/O6klzgBLLs9h" |
| Expected result | Correct decyption. |
| Post condition | String has changed to the excepted decrypted string |
| Status | Test passed |
| Comments | - |

# Installation guide

<https://github.com/jku-win-se/teaching.ss22.prse.digitaltwin.team1/tree/main/Frontend/smart-home-ui>

<https://github.com/jku-win-se/teaching.ss22.prse.digitaltwin.team1/tree/main/Backend>