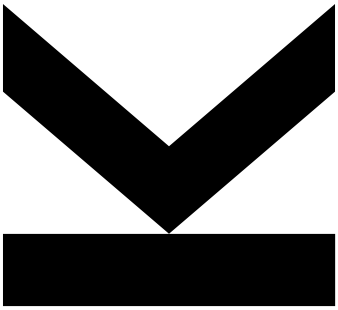
JOHANNES KEPLER
UNIVERSITÄT LINZ

24.01.2023

PROJECT DOCUMENTATION



Smart Room Application

Team 3

Florian Offenberger, Jakob Hubauer,
Fabio Leidwein, Lukas Nömeyer

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
jku.at
DVR 0093696

JOHANNES KEPLER UNIVERSITÄT LINZ

Table of Contents

Version history

Version	Date	Creator	Changes
1.0	02.02.2023	Jakob Hubauer	Creating the document, introduction
1.1	07.02.2023	Jakob Hubauer	Management Perspective, User Perspective
1.2	07.02.2023	Jakob Hubauer	Installation guide
1.3	09.02.2023	Jakob Hubauer	Design decisions 1-3
1.4	10.02.2023	Lukas Nömeyer	General small rework and adding Management Perspective
1.5	10.02.2023	Florian Offenberger	Added Information about the Implementation

Table 1: Version history

Introduction

This document is about the software project "Smart Room", which was realized in the course "Praktikum Software Engineering". An application was developed, with which a user can administer rooms and different components of a room. That means that you can add or delete rooms and change them. In a room you can add or delete components (door, window, fan, light). These components can be switched on and off or opened and closed. In addition, you can import or export components into a CSV file. This file will be saved directly on the desktop.
On line-charts information of a room is displayed, which represents the temporal course of the room temperature, the air quality, as well as the number of persons in the room.

Management Perspective

Which of the requirements did you implement? Which teammate was responsible? and how many hours he/she expend implementing this? If any requirement has not been implemented, then, the team should provide explanations for each requirement.

Nr	*Requirements*	*Team-mate*	*expenditure of time in h*
1	Basic: Import data from CSV to an Entity-Relationship database.	*Jakob,* *Lukas*	*3* *1*
2	Basic: Create/Update Rooms (id, size, available doors, windows, lights and fans)	*Fabio,* *Florian,* *Lukas*	*50* *60* *23*
3	Basic: Update and remove rooms	*Fabio,* *Florian,* *Lukas*	*15* *20* *10*
4	Basic: Visualize available rooms	*Jakob*	*4*
5	Basic: Visualize static information for each room (id, size, available doors, windows, lights and fans).	*Jakob*	*4*
6	Basic: Develop a line chart that shows real-time data regarding light/fan/window/door.	*Not implemented*	*because of time reasons*
7	Basic: Develop a line chart that shows the co2/temperature values and the number of people for each room over time.	*Lukas,* *Jakob*	*2* *3*
8	Basic: Save rooms structure (rooms + static information) in a .csv file.	*Jakob*	*2*
9	Basic: Automatically add random values of co2/temperature/number of people for a specific room.	*Lukas*	*5* *Not fully implemented. Data is generated and stored in the frontend due to late implementation on server side*

10	Basic: Live update of visualizations for co2, temperature, and lights/ventilators/windows/doors status for each room.	*Lukas*	*10*
12	Remote Control: Allow to lock/unlock doors via the user interface.	*Lukas* *Florian*	*4* *2*
13	Remote Control: Allow to turn on/off lights via the user interface	*Lukas* *Florian*	*4* *10*
14	Remote Control: Allow to open/close windows via the user interface.	*Lukas* *Florian*	*4* *2*
15	Remote Control: Allow to turn on/off fans via the user interface.	*Lukas* *Florian*	*4* *2*
16	Security: Send alarm if temperature is above 70 degrees celsius.	*Jakob*	*2*
17	Security: Unlock all doors if temperature is above 70 degrees celsius.	*Jakob*	*3*
18	Energy Saving: Turn lights on if there are people in the room.	*Jakob*	*3*
19	Energy Saving: Lights should be turned off if the room is empty.	*Jakob*	*3*
20	Energy Saving: Turn off running devices if the room is empty.	*Jakob*	*3*
21	Air Quality: Open window + activate fan if co2 values are > 1000 parts per million (ppm).	*Jakob* *Florian* *Fabio*	*2* *5* *2*
22	Air Quality: Change room color in user interface based on co2 values.	*Jakob* *Florian*	*2* *5*

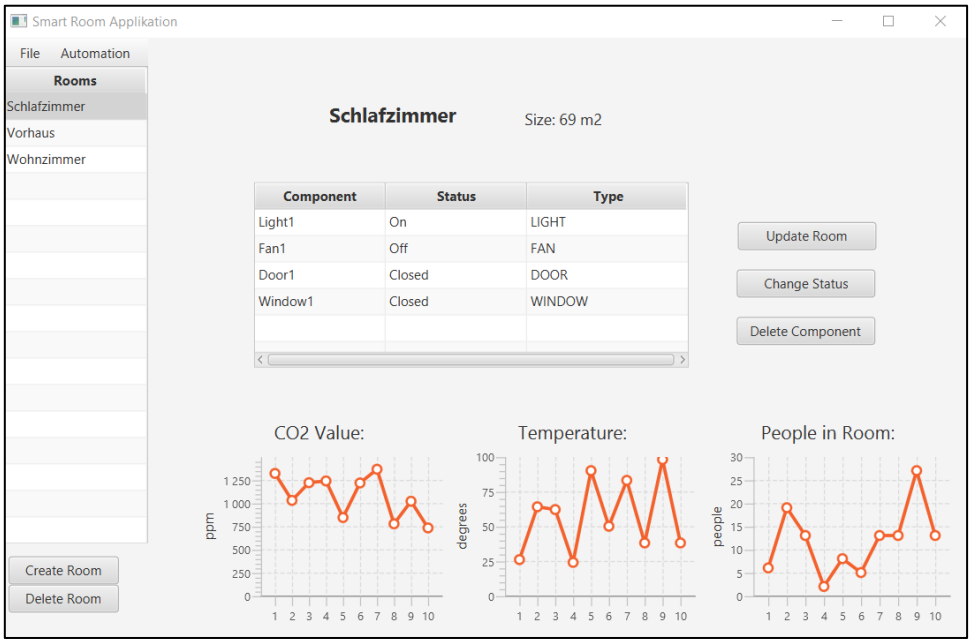
Genauere Zeitaufzeichnungen sind unter /docs/Clockify zu finden.

User Perspective

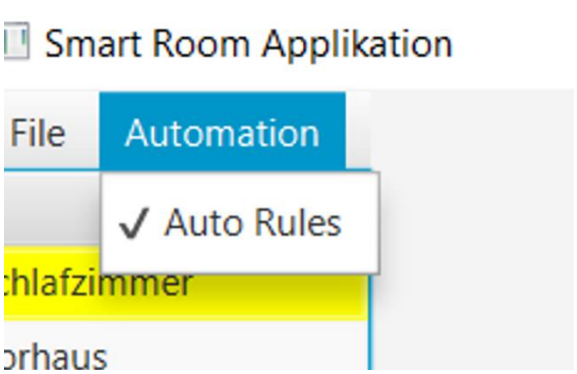
How were the requirements implemented in the user interface? (Screenshots of the user interface and description of the functionality based on scenarios) How the users should use the system?

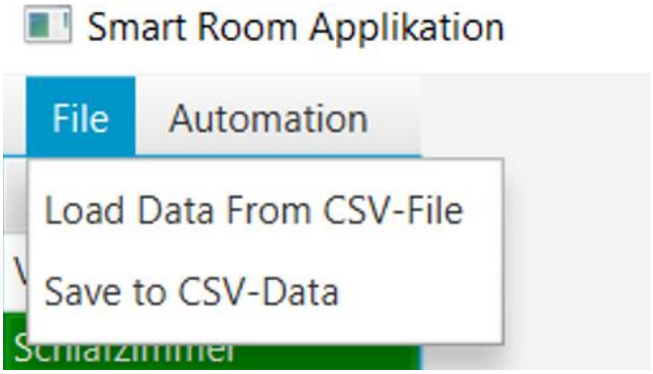
Start Scene

StartScene: When the program is opened, the following start scene appears.
The name of the room and its size is displayed as a heading.
In the table on the left, you can see all the rooms that are loaded from the database and are also stored there. Below this table there are two buttons. By clicking on the "Delete Room" button you can delete the room marked above. By clicking on the "Create Room" button, you can access a new scene, which is described in a separate paragraph.
In the other table you can see all the components that are in the selected room and are also loaded from the database. In the column "Component" you can see the name of the component, in the column "Status" you can see the current status of the component, where "true" means on or open and "false" means off or closed. In the column "Type" the type of the component is shown as additional information to avoid confusion. By clicking on the line and thus selecting a respective component and then clicking on the "Delete Component" button, the respective component can be deleted; by clicking on the "Change Status" button, the status of the component can be changed. By clicking the "Update Room" button, you can access a new scene, which is described in a separate paragraph.
The three line-charts show the CO2 value, the temperature and the number of people in the room. Every 10 seconds new random values are loaded. If the auto-rules are activated, the components are automatically switched on, off, opened or closed. And the color of the room in the table on the left changes according to the co2 value in the room.



By clicking on "File" in the upper left corner, the following menu bar opens, with which you can either load components and the corresponding rooms from a CSV file or save them to a CSV file. The location of the file is by default the desktop and the name is "components.csv".
By clicking on “Automation” a menu bar drops down in which you can enable and disable the auto rules.



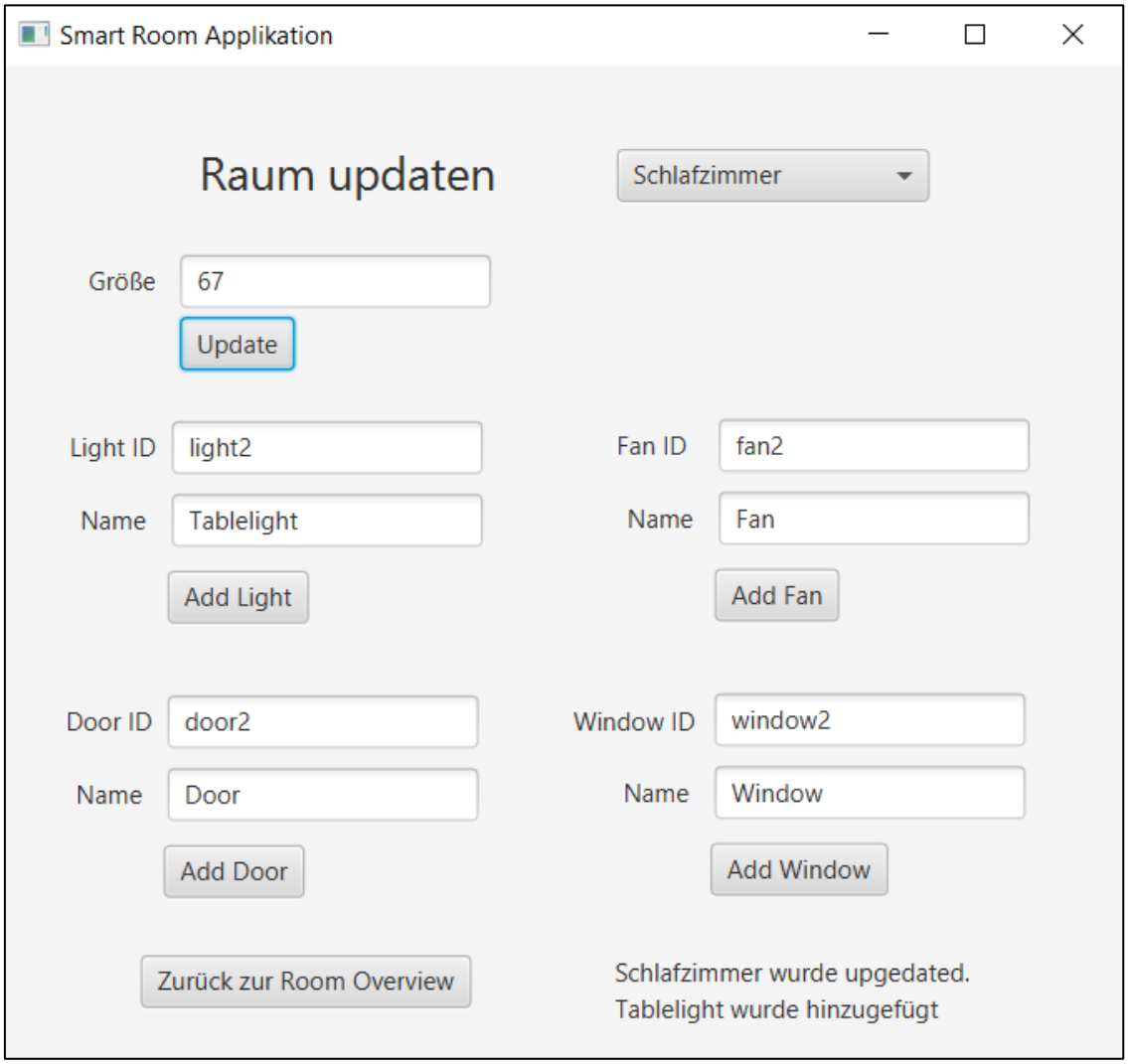


Create Room Scene

By clicking the "Create Room" button on the Start Scene you will get to the following scene. Here you can enter the Room ID and the size of the room in square meters, where the Room ID corresponds to the name of the room.

Update Room Scene

By clicking on the "Update Room" button you get to the Scene, where you can add components to a room selected by a ComboBox, as well as change the size of the room.

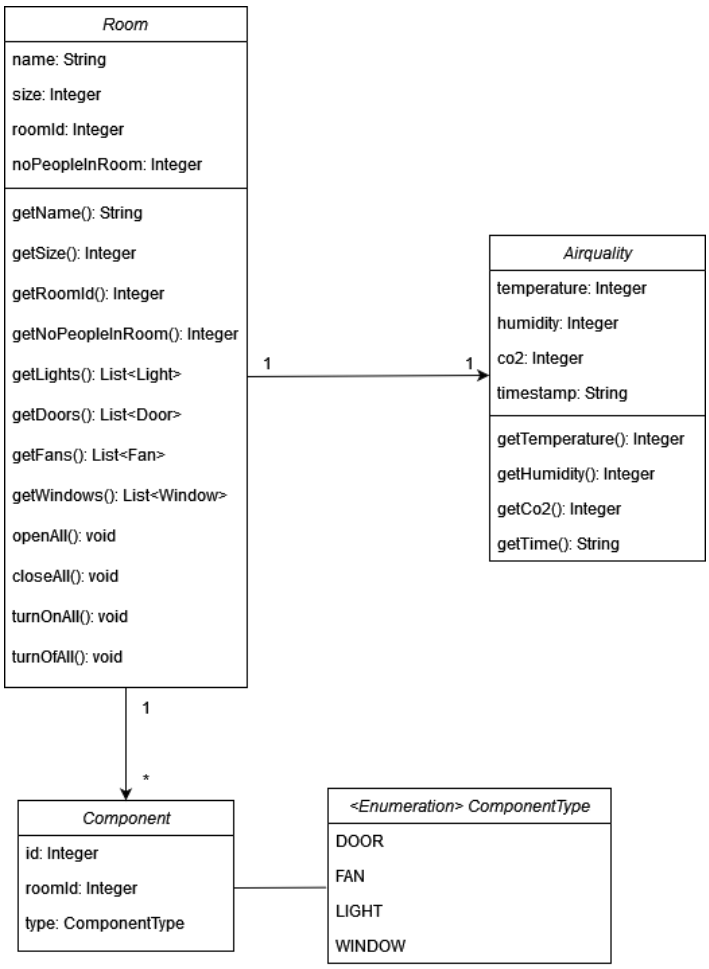


Developer Perspective

Design

Overview of the System

UML Diagram



Important Design Decisions

Description of the 3-5 most important design decisions in the following scheme

Decision1: *JavaFX for the graphical user interface*
Reason: Because the entire visual design of the application can be specified by stylesheets, so that design and functional code can be completely separated.
Considered Alternatives: Java Swing
Assumptions: Build a simple user interface where the user has nearly everything on one page.
Effect: Create only one controller class because we almost all functions are on one page.

Decision2: *JDBC for database connection*
Reason: Because JDBC provides a unified interface to databases from different vendors and is specifically designed for relational databases.
Considered Alternatives: Hibernate
Assumptions: JDBC enables to create queries and update data to a relational database using SQL.
Effect: We already learned SQL in previous courses. That's why the implementation with SQL was easier for us than with HQL.

Decision3: *Fans, doors, windows, lights are "components" on the client's side.*
Reason: For the visualization of the room components in a table we designed a unified "components" class. That means that every component has the properties of a room/fan/door/window plus a field for status and the component type.
Assumptions: This allowed us to display and manage all components of a room on one table.
Effect: Clear visualization of all components of a room with name, status, type.

Implementation

Maven Project Structure
A Maven project is a project management and build tool in Java. Maven projects typically follow a standard directory structure and are organized into modules. Our structure of a Maven project with a parent project skeleton, a server and client look like this:

Parent Project Skeleton

```
|
+-- server
|    |
|    +-- pom.xml
|    +-- src
|         |
|         +-- main
|         |    |
|         |    +-- java
|         |    +-- resources
|         |
|         +-- test
|              |
|              +-- java
|              +-- resources
|
+-- client
|    |
|    +-- pom.xml
|    +-- src
|         |
|         +-- main
|         |    |
|         |    +-- java
|         |    +-- resources
|         |
|         +-- test
|              |
|              +-- java
|              +-- resources
+-- pom.xml
```

In our structure, the parent project skeleton serves as the top-level project that builds all the modules together. The pom.xml file at the top-level project defines the parent project, its dependencies, and the modules that it builds.

The server and client directories each contain their own pom.xml file that defines their respective dependencies and build information. The src directory contains the source code for each module, organized into main and test directories. The main directory contains the production code, while the test directory contains the test code.

The java directories within each module contain the Java source code, while the resources directories contain resources such as configuration files, database scripts, etc.

With this structure, we can manage and build the different parts of our application as separate modules, making it easier to maintain and upgrade. Additionally, the use of Maven as a build tool

provides a standard way to manage dependencies and build our project, making it easier for all group members to work on the project.

Important Pieces of Code:

Example: Add a Room

This process describes the steps involved in adding a room from the client to the server and then to the database.

1. On the client side, a new room object is created with the room id, room size, and measurement unit. This room object is then serialized into a JSON string using the **objectMapper.writeValueAsString** method.
2. The client then makes a **POST** request to the server at the **/Rooms** endpoint, sending the serialized room object as the request body and setting the **Content-Type** header to **application/json**.
3. On the server side, the **addRoom** method is triggered by the **POST** request to the **/Rooms** endpoint. This method takes the request body and creates a new **Room_Object** instance with the same data.
4. The server then inserts the room data into the database using the **db.add_room** method, passing in the connection, table name, room id, room size, and measurement unit as arguments.
5. In the **add_room** method, a JDBC **Statement** object is created and used to execute an SQL **insert** statement, which adds the new room data to the **room** table.
6. The server then returns a response to the client, containing the added room object, with a status code of **200 OK**.
7. On the client side, the response from the server is received and deserialized into a **Room_Object** instance using the **objectMapper.readValue** method.

This process shows how the client can interact with the server to add data to the database, and how the server can use JDBC to execute database operations.

Here you can see the different code snippets.

Client:

```
@Override
public Room_Object addRoom(String room_id, double room_size, String measurement_unit) {

    Room_Object room = new Room_Object(room_id, room_size, measurement_unit);
    String body = "";
    try {
        body = objectMapper.writeValueAsString(room);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
    HttpRequest request =
            HttpRequest.newBuilder().uri(URI.create(startURI + "/Rooms")).
                    header("Content-Type", "application/json").
                   POST(HttpRequest.BodyPublishers.ofString(body)).build();

    try {
        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
        Room_Object room_object = objectMapper.readValue(response.body(), new TypeReference<Room_Object>(){});
        return room_object;//or true;
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
    }

    return null;
}
```

Server:

```
@PostMapping("/Rooms")
public ResponseEntity<Room_Object> addRoom(@RequestBody Room_Object room) {
    Room_Object room_object = new Room_Object(room.getRoom_id(), room.getRoom_size(), room.getMeasurement_unit());
    db.add_room(c, "room", room.getRoom_id(), room.getRoom_size(), room.getMeasurement_unit());
    return ResponseEntity.ok(room);
}
```

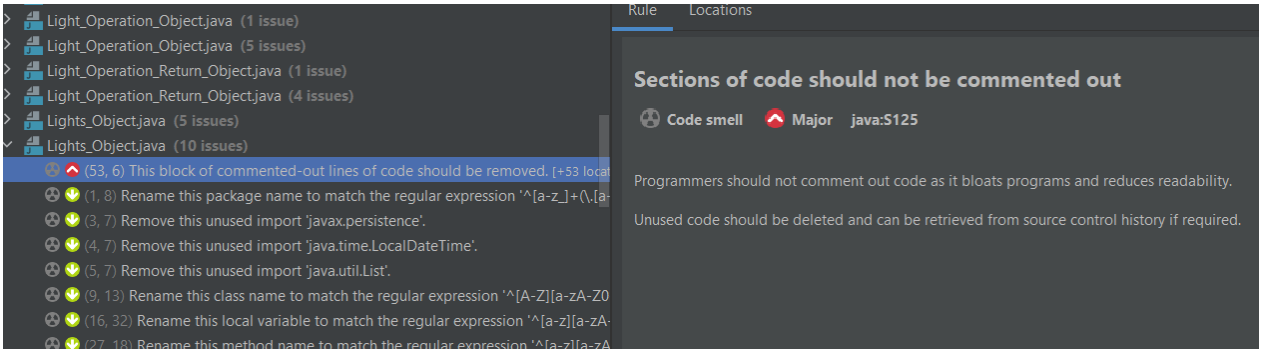
Save in Database with JDBC:

```
public void add_room(Connection c, String table_name, String roomid, double size, String unit){
    Statement statement;
    try {
        String query = String.format("insert into %s(roomid, size, unit) values ('%s','%s','%s');", table_name, roomid, size, unit);
        statement = c.createStatement();
        statement.executeUpdate(query);
        System.out.println("Row Inserted");
    } catch (Exception e){
        System.out.println(e);
    }
}
```

Code Quality

Description of the use of PMD, description of the findings and which of them have been fixed.

Our code quality tool was Sonarlint. SonarLint is an extension for Integrated Development Environments (IDEs) such as IntelliJ, which is our IDE.
SonarLint analyzes the source code and checks for potential issues, such as bugs, security vulnerabilities, and code smells, and provides immediate feedback to the developer, including suggestions for how to fix the issue.
SonarLint automatically detects all code quality errors and we fixed the serious ones. Most of the errors that are still about are commented code, parameters or imports that we do not need.



Testing

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

Test 1

```
@Test
public void testAddRoom_checkIfRoomIsAdded() throws SQLException {
    // Arrange
    Room_Object expectedRoom = new Room_Object("testroom", 100.0, "m2");
    Connection c = mock(Connection.class);
    RESTController restController = new RESTController();
    Statement statement = mock(Statement.class);

    when(c.createStatement()).thenReturn(statement);

    // Act
    ResponseEntity<Room_Object> actualResponse =
restController.addRoom(expectedRoom);

    // Assert
    assertEquals(expectedRoom, actualResponse.getBody());
    assertEquals(HttpStatus.OK, actualResponse.getStatusCode());
}
```

Description of the 3 selected tests according to the following pattern:

Test Case ID	testAddRoom_checkIfRoomIsAdded
Designed by	Fabio
Execute on	AddRoom method
Carried out by	Fabio
Tested Requirement	If add a room is successfully
Requirement	Room_id must be unique
Test steps	Create mock connection, then addRoom, then check if same as expectedRoom and statusCode = OK
Test data	"testroom", 100.0, "m2"
Expected result	Room is successfully added
Post condition	Room is successfully added
Status	200 OK
Comments	

Test 2

```
@Test
public void testDeleteLight_returnsOk() {
    // Arrange
    PostgreSQLJDBC postgreSQLJDBC = mock(PostgreSQLJDBC.class);
    when(postgreSQLJDBC.deleteLightById(any(Connection.class), anyString(),
anyString())).thenReturn(true);
    RESTController restController = new RESTController();
    restController.postgreSQLJDBC = postgreSQLJDBC;
    String room_id = "testroom";
    String light_id = "testlight";

    // Act
    ResponseEntity<String> actualResponse =
restController.deleteLight(room_id, light_id);

    // Assert
    assertEquals(HttpStatus.OK, actualResponse.getStatusCode());
}
```

Test Case ID	testDeleteLight_returnsOk
Designed by	Fabio
Execute on	DeleteLight method
Carried out by	Fabio
Tested Requirement	If delete a light is successfully
Requirement	Light_id must exist

Test steps	Create mock connection, then deleteLight, then check if StatusCOde = OK
Test data	"testroom", "testlight”
Expected result	Light is successfully deleted
Post condition	Light is successfully deleted
Status	200 OK
Comments	

Test 3

```
@Test
public void testGetAllWindows_CheckIfReturnTypeIsCorrect() {
    // Arrange
    RESTController restController = new RESTController();
    PostgreSQLJDBC postgreSQLJDBC = mock(PostgreSQLJDBC.class);
    restController.postgreSQLJDBC = postgreSQLJDBC;
    String room_id = "room_id";
    List<Window_Object> expectedWindows = new ArrayList<>();
    when(postgreSQLJDBC.getWindows(any(),
eq(room_id))).thenReturn(expectedWindows);

    // Act
    ResponseEntity<List<Window_Object>> actualResponse =
restController.getWindows(room_id);

    // Assert
    assertEquals(expectedWindows, actualResponse.getBody());
    assertEquals(HttpStatus.OK, actualResponse.getStatusCode());
}
```

Test Case ID	testGetAllWindows_CheckIfReturnTypeIsCorrect
Designed by	Fabio
Execute on	GetWIndows method
Carried out by	Fabio
Tested Requirement	If windows are correctly returned from room_id
Requirement	Room_id must be unique
Test steps	Create mock connection, then getWindow, then check if same as expectedWindows and StatusCode = OK
Test data	"room_id”
Expected result	Window is successfully returned
Post condition	Window is successfully returned
Status	200 OK
Comments	

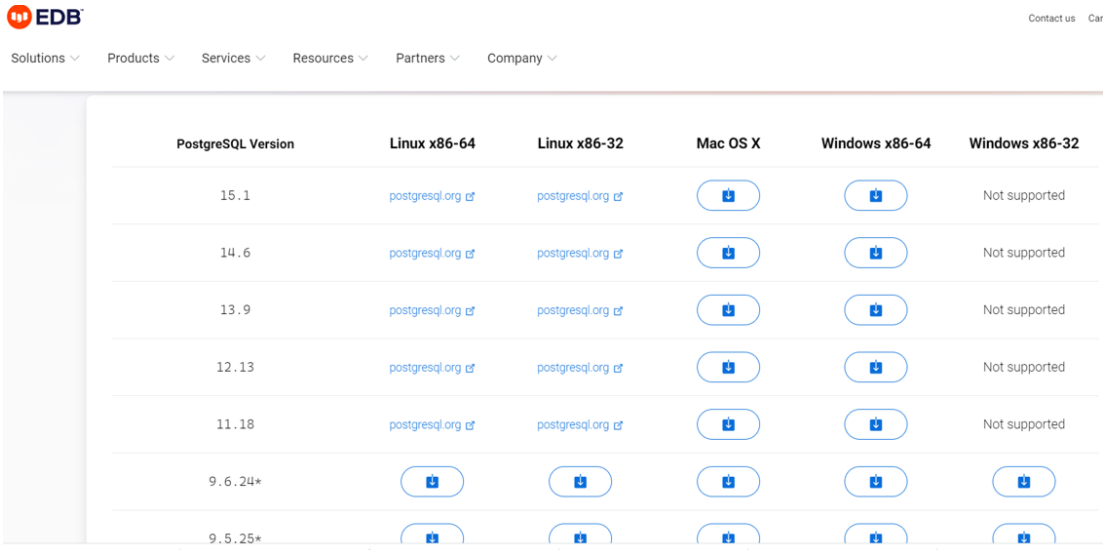
Installation guide

Description of how to install and start the system: Step by step
jku-win-se/teaching.ws22.prse.smartroom.Team3 (github.com)

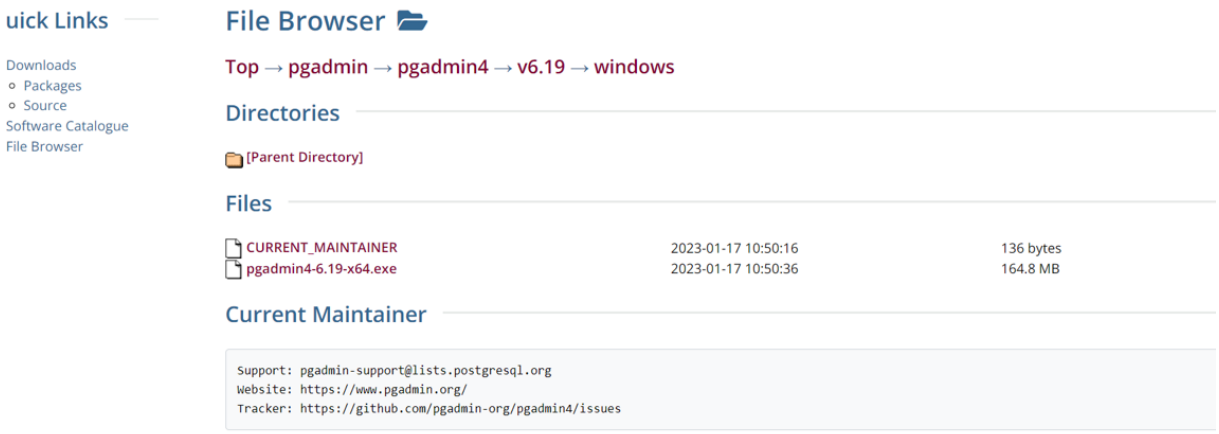
Installation-guide: SmartRoom Gruppe 3

Step by step guide:

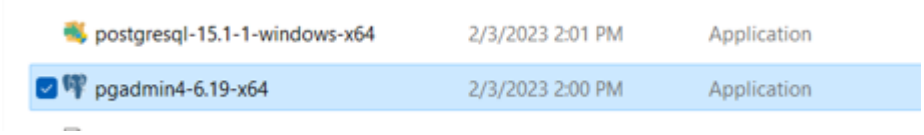
1) **Download postgresql from https://www.enterprisedb.com/downloads/postgres-postgresql-downloads**



2) **Install pgadmin 4 from https://www.pgadmin.org/download/**



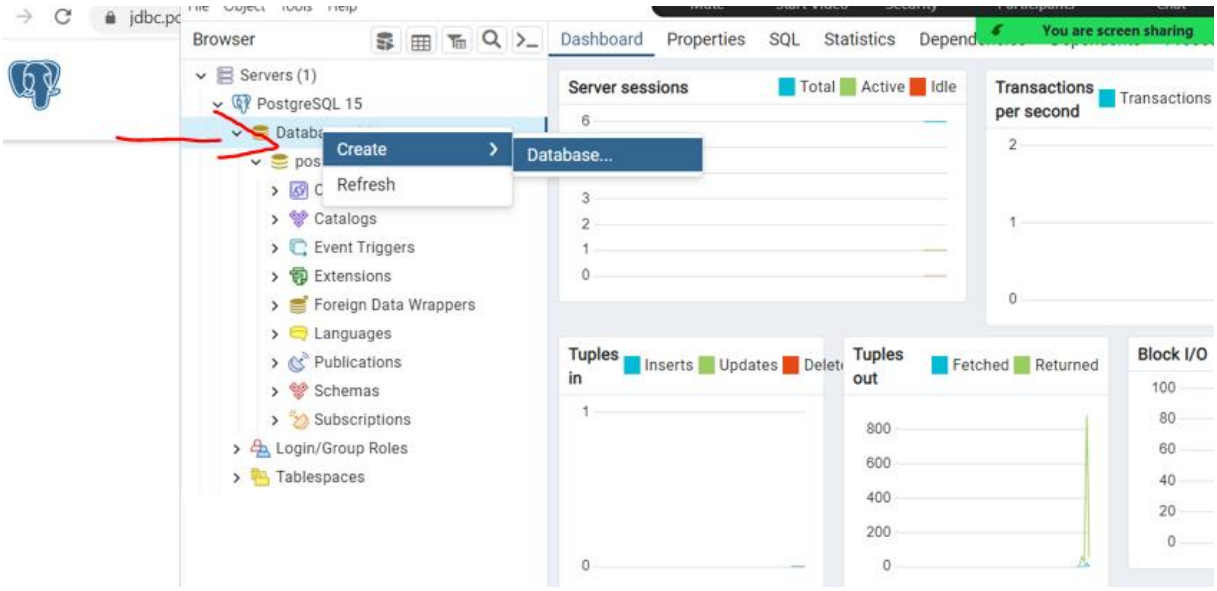
3) **After that you should have these two applications in your download folder**



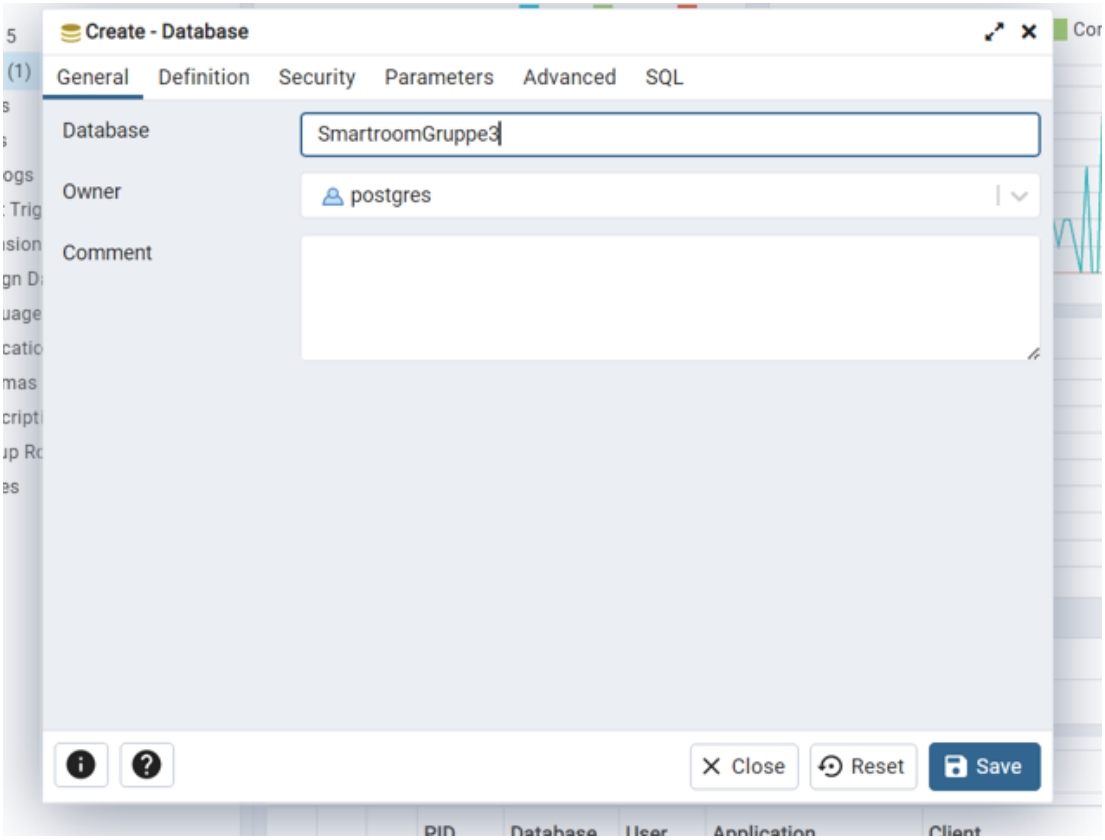
4) **Install both**

5) Open pgadmin4 and set master password to “smartroom3”

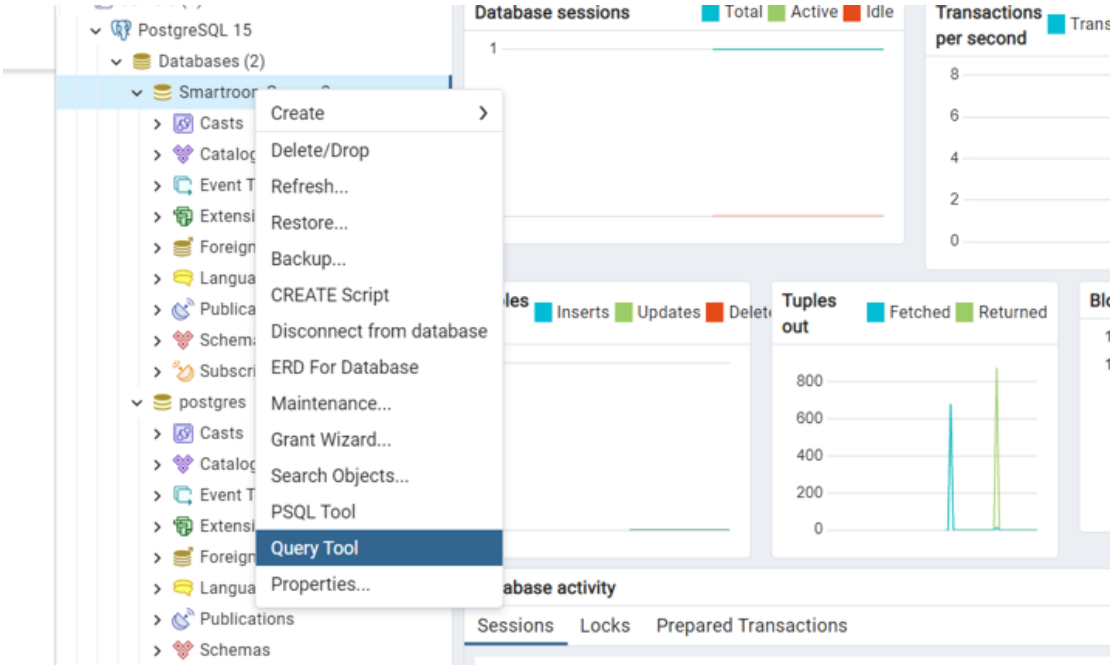
6) Create database with right click on database > create > database…



7) Name the database “SmartroomGruppe3”



8) Initialize the database with right click on SmartroomGgruppe3 > Query Tool with following Code:



*/*Bei Erstellung der Datenbank:*
user: postgres
password: smartroom3
Datenbankname: SmartroomGruppe3
Groß/Kleinschreibung wichtig/*

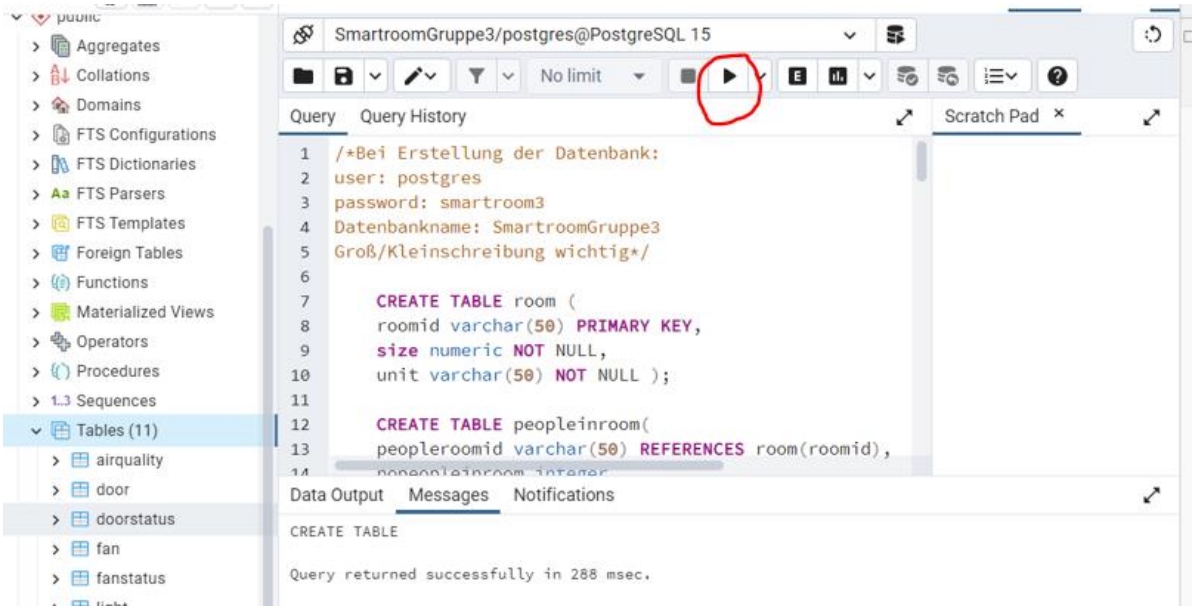
```
CREATE TABLE airquality (
    deviceid character varying(50) NOT NULL,
    fanid character varying(50) NOT NULL,
    roomid character varying(50) NOT NULL,
    co2 integer,
    co2unit character varying(50),
    humidity integer,
    humidityunit character varying(50),
    temperature integer,
    temperatureunit character varying(50),
    airqualitytimestamp timestamp without time zone NOT NULL
);

CREATE TABLE door (
    doorid character varying(50) NOT NULL,
    roomid character varying(50),
    doorname character varying(50)
);

CREATE TABLE doorstatus (
    doorid character varying(50) NOT NULL,
    doorisopen boolean,
```

```
    doortimestamp timestamp without time zone NOT NULL
);


CREATE TABLE fan (
    fanid character varying(50) NOT NULL,
    roomid character varying(50),
    fanname character varying(50)
);

CREATE TABLE fanstatus (
    fanid character varying(50) NOT NULL,
    fanison boolean,
    fantimestamp timestamp without time zone NOT NULL
);

CREATE TABLE light (
    lightid character varying(50) NOT NULL,
    roomid character varying(50),
    lightname character varying(50)
);

CREATE TABLE lightstatus (
    lightid character varying(50) NOT NULL,
    lightison boolean,
    lighttimestamp timestamp without time zone NOT NULL,
    hex character varying(50),
    brightness integer
);

CREATE TABLE peopleinroom (
    peopleroomid character varying(50) NOT NULL,
    nopeopleinroom integer,
    peopletimestamp timestamp without time zone NOT NULL
);

CREATE TABLE room (
    roomid character varying(50) NOT NULL,
    size numeric NOT NULL,
    unit character varying(50) NOT NULL
);

CREATE TABLE windows (
    roomid character varying(50),
    windowid character varying(50) NOT NULL,
    windowname character varying(50)
);
```

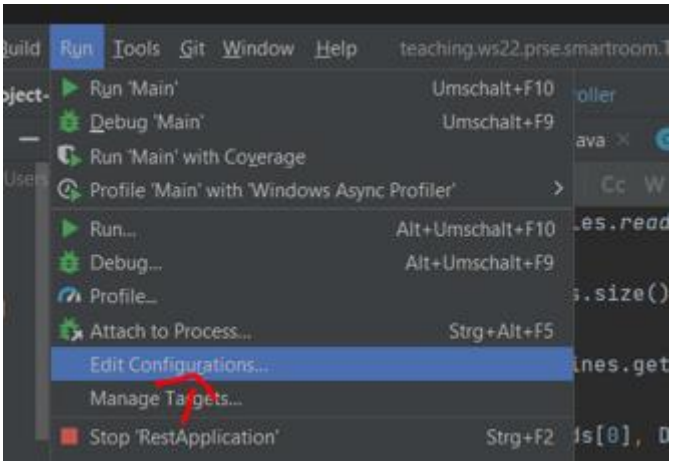
```
CREATE TABLE windowstatus (
    windowid character varying(50) NOT NULL,
    windowisopen boolean,
    windowtimestamp timestamp without time zone NOT NULL
);

ALTER TABLE ONLY airquality
    ADD CONSTRAINT airquality_pkey PRIMARY KEY (fanid, airqualitytimestamp, roomid);

ALTER TABLE ONLY door
    ADD CONSTRAINT door_pkey PRIMARY KEY (doorid);

ALTER TABLE ONLY doorstatus
    ADD CONSTRAINT doorstatus_pkey PRIMARY KEY (doorid, doortimestamp);

ALTER TABLE ONLY fan
    ADD CONSTRAINT fan_pkey PRIMARY KEY (fanid);

ALTER TABLE ONLY fanstatus
    ADD CONSTRAINT fanstatus_pkey PRIMARY KEY (fanid, fantimestamp);

ALTER TABLE ONLY light
    ADD CONSTRAINT light_pkey PRIMARY KEY (lightid);

ALTER TABLE ONLY lightstatus
    ADD CONSTRAINT lightstatus_pkey PRIMARY KEY (lightid, lighttimestamp);

ALTER TABLE ONLY peopleinroom
    ADD CONSTRAINT peopleinroom_pkey PRIMARY KEY (peopleroomid, peopletimestamp);

ALTER TABLE ONLY room
    ADD CONSTRAINT room_pkey PRIMARY KEY (roomid);

ALTER TABLE ONLY windows
    ADD CONSTRAINT windows_pkey PRIMARY KEY (windowid);

ALTER TABLE ONLY windowstatus
    ADD CONSTRAINT windowstatus_pkey PRIMARY KEY (windowid, windowtimestamp);

ALTER TABLE ONLY airquality
    ADD CONSTRAINT airquality_fanid_fkey FOREIGN KEY (fanid) REFERENCES fan(fanid);

ALTER TABLE ONLY airquality
    ADD CONSTRAINT airquality_roomid_fkey FOREIGN KEY (roomid) REFERENCES room(roomid);

ALTER TABLE ONLY doorstatus
```

```
    ADD CONSTRAINT doorstatus_doorid_fkey FOREIGN KEY (doorid) REFERENCES door(doorid) ON
DELETE CASCADE NOT VALID;

ALTER TABLE ONLY fanstatus
    ADD CONSTRAINT fanstatus_fanid_fkey FOREIGN KEY (fanid) REFERENCES fan(fanid) ON DELETE
CASCADE NOT VALID;

ALTER TABLE ONLY door
    ADD CONSTRAINT fk_door_room FOREIGN KEY (roomid) REFERENCES room(roomid) ON DELETE
CASCADE NOT VALID;

ALTER TABLE ONLY fan
    ADD CONSTRAINT fk_fan_room FOREIGN KEY (roomid) REFERENCES room(roomid) ON DELETE
CASCADE NOT VALID;

ALTER TABLE ONLY light
    ADD CONSTRAINT fk_light_room FOREIGN KEY (roomid) REFERENCES room(roomid) ON DELETE
CASCADE NOT VALID;

ALTER TABLE ONLY windows
    ADD CONSTRAINT fk_windows_room FOREIGN KEY (roomid) REFERENCES room(roomid) ON
DELETE CASCADE NOT VALID;

ALTER TABLE ONLY lightstatus
    ADD CONSTRAINT lightstatus_lightid_fkey FOREIGN KEY (lightid) REFERENCES light(lightid) ON
DELETE CASCADE;

ALTER TABLE ONLY peopleinroom
    ADD CONSTRAINT peopleinroom_peopleroomid_fkey FOREIGN KEY (peopleroomid) REFERENCES
room(roomid);

ALTER TABLE ONLY windowstatus
    ADD CONSTRAINT windowstatus_windowid_fkey FOREIGN KEY (windowid) REFERENCES
windows(windowid) ON DELETE CASCADE NOT VALID;
```

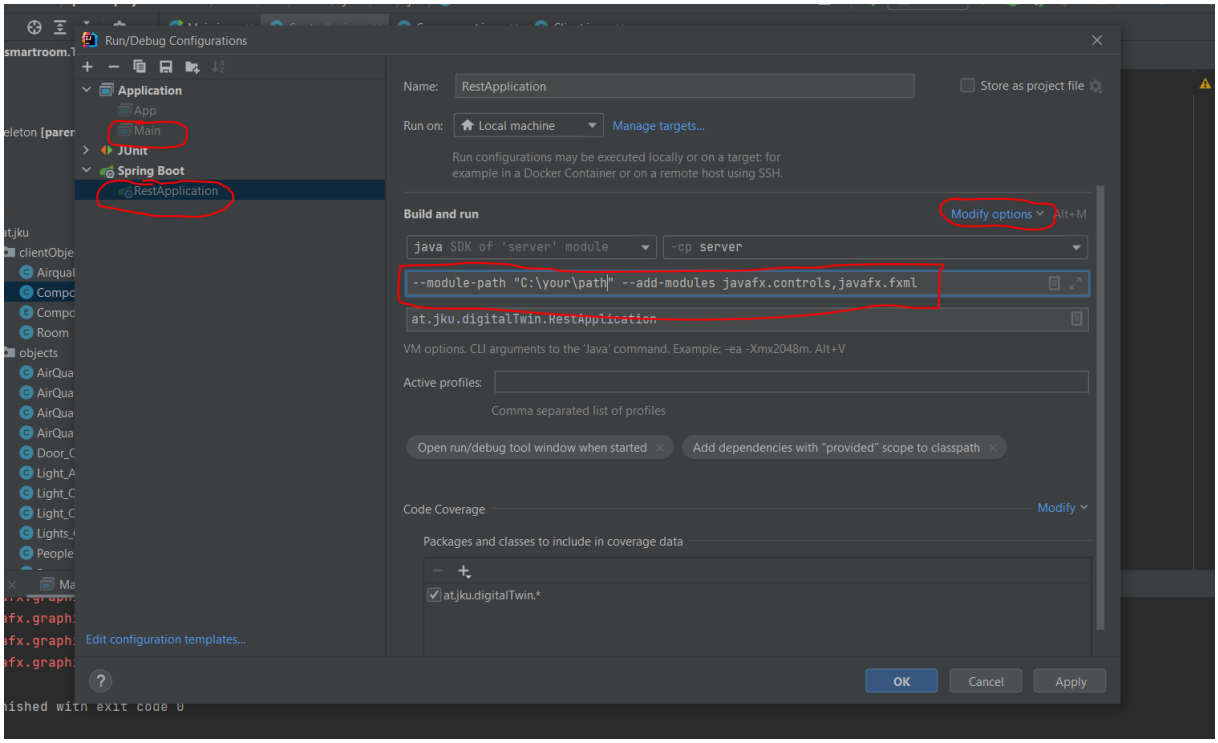
9) Run the code with a click on the “play button”



10) Start IntelliJ and open > Edit Configurations…



11) Set VM-Options for Main and RestApplication



You may have to click on “Modify optins” > Add VM options
Instead of your\path write the folder path of your “lib” folder with following files:
Download this files from GitHub!


javafx-graphics-17-win.jar


javafx-fxml-17-win.jar


javafx-controls-17-win.jar


javafx-base-17-win.jar

12) Finally you can run the RestApplication and after that the Main Application