**Washing Machine System**

**Architecture and Database**

Documentation of a project for the purpose of the course BIE-SI1.

Authors: Alejandro Chirivella Ciruelos, Emily Hannon, Alexander Muryshkin, Andrey Olos, Noam Yehuda, Lawrence Zawahri

**Contents**

1. System Architecture 3

1.1 Server 3

1.1.1 Docker 3

1.1.1.1 PostgreSQL 3

1.1.1.1.1 SUZ Schema 3

1.1.2 Tomcat 4

1.1.2.1 Java 15 (Spring Framework) 4

1.1.2.1.1 SUZ Web Application 4

1.2 User's PC 4

1.2.1 Internet Browser 5

2. Architecture 5

3. Logical Architecture 6

3.1 Package Diagram 6

3.2 Controller 6

3.3 Service 7

3.4 Dto 7

3.5 Repository 7

3.6 Entity 7

4. Database Model 8

4.1 «table» BUILDING 9

4.2 «table» KEY 10

4.3 «table» RESERVATION 10

4.4 «table» ROOM 10

4.5 «table» ROOM\_HISTORY 10

4.6 «table» STUDENT 11

4.7 «table» WASHING\_MACHINE 11

4.8 «table» SWAP\_REQUEST 11

4.9 «table» COMPLAINT\_ABOUT\_STUDENT 12

4.10 «table» PENALTIES 12

4.11 «table» RECEPTIONIST 12

4.12 «table» COMPLAINT\_ABOUT\_WM 13

# System Architecture

Graphical user interface, application

Description automatically generated

Figure 1 - System Architecture

## Server

The server will be running on a regular PC and it hosts the web application and the local SUZ database. Also, it is connected to the user's PCs and to the dormitory database server by using TCP/IP.

### Docker

Docker uses OS-level virtualization to provide software in packages called containers. Docker gives us the ability to transfer the database in case of a crush or other possible issues.

#### PostgreSQL

PostgreSQL is an advanced open-source relational database system which is based on SQL. It is stored inside a Docker container.

##### SUZ Schema

SUZ Schema is the executable database code derived from a database schema. It communicates with the web application through SQL.

### Tomcat

Tomcat provides the server with a HTTP web environment in which the Java code will run. It will handle the HTTPS connections with the user's browsers.

#### Java 15 (Spring Framework)

Java 15 is the version of the Java API will be running inside the Tomcat Server.

##### SUZ Web Application

This is the Java code that will make up the web application. The code will run inside of the Java 15 API and the Spring framework. It will communicate with the SUZ database by using SQL queries through the Spring framework, so we do not have to code the queries manually.

## User's PC

The web application will be accessed on the user's PC through a web browser.

### Internet Browser

The browser will connect to the server through browsers such as Mozilla Firefox, Google Chrome, Edge or Safari by using an HTTPS connection.

# Architecture

This chapter describes the architecture of the web application SUZ dorm application.

The web application for students and administration to reserve and manage schedules of dorm appliances will be implemented as a Java application based on the following the technologies:

* SpringBoot framework
* Tomcat
* Gradle
* Docker
* Postgres
* HTML, CSS, JavaScript
* Java 15+
* Mockito

The web application will run on the server, where the application makes request to get information. The server hosts the database where all the data is stored and can run in all environments and be available for most standard web browsers (some issues arise of firefox, in unix), on a computer or smart phone.

The application will only need around 1 GB of memory, but will have up to 2.5 GBs allocated for it for future development. The database with its contents will have currently a maximum of 25 GB for the ability to store as much information as might be needed without the probability of it crashing due to amount of memory.

Using of the SpringBoot framework will allow us to generate out database in our Postgres database and access any needed information from the tables.

All this is located in the docker environment which allows us to easily place our server anywhere.

The current server prototype is built with the ability to expand in an agile development style depending on our user’s needs, and all parts have unit tests that must be run successfully for the server to operate.

# Logical Architecture

Diagram

Description automatically generated

Figure 2 – Package Diagram

## Package Diagram

The package diagram shows us how the different java files are located in our application. It is used to keep separated classes with similar functionalities, so it is easier to know which the purpose of each class is depending on the package it is in. We have five different packages: controller, service, dto, repository and entity. If we look at this as a three-layered structure, we should put the controller in the Presentation Layer (where HTTP requests are handled), the service and the dto in the Business Layer (where the logic of our application is) and, the repository and the entity in the Data Layer (where the connection with the database is managed). The upper layers always have dependencies of the lower layers: the Presentation layer imports the classes from the Business Layer, and the Business Layer imports the classes from the Data Layer. Also, the repository imports the entity, and the service imports the dto.

## Controller

In this package we have the classes than handle the HTTP request. They check if the requests are well-formed and decide what logic of the Business Layer use and what to return to the client. They also determine which URLs and which HTTP methods to use to interact with the server.

## Service

In this package we have the classes that have the logic of our application. They determine the methods of how to create, delete, update or find an object without directly connecting to the database. They use the methods of the Data Layer in their own methods.

## Dto

In this package we have the classes that determine the objects we send and receive when we communicate with the user. For each, object there might be (not always) two different dtos, one for the creation of the object (which is sent by the user to the server) and one for sending this object back to the user (which is sent by the server to the user). We might have two dtos for each class because we usually receive different information from the user that the one we sent back to the user.

## Repository

In this package we have the classes that have the methods to interact directly with the database. They are used as a layer of abstraction for the upper layers, so they can retrieve information from the database without having to code manually an SQL Query. In our application, we don’t have to code these methods because the Spring Framework does it automatically for us. We just have to create the classes and then Spring does the rest.

## Entity

In this package we have the classes that have the objects that represent each table of our database. These objects act as an abstraction layer of the different rows of each table of the database, so if you modify one of the attributes of an instance of these objects you modify the value in that certain row of the database.

# Database Model

In this part of the project, we will specify the database schema which the application will be connected to

Graphical user interface, application, Word

Description automatically generated

Figure 3 - Database Model

## Postgres



Figure 4 - Database Model

The database model contains the description of all the entities, with their corresponding attributes. As well as the relations between different entities. As said before, this database has been implemented using

## 4.1.1 «table» BUILDING

Building table stores all buildings of university

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| building\_id | integer | True | Unique primary key of building. |
| name | varchar(50) | False | The name of the building, if exists. |
| address | varchar(100) | True | The address of the building. |
| number | integer | True | The street number of the building. |
| location | integer | True | The dorm location the building belongs to. |

## «table» KEY

Key table stores data of keys for washing machine

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| key\_id | integer | True | Unique primary key of building. |
| student\_id | integer | False | The ID of the student that the key was last given to if not yet returned. |
| return\_status | boolean | True | This status represents if the key is free or is currently help by a student. |
| washing\_machine\_id | integer | True | The unique ID of the washing that the key gives access to. |

## «table» RESERVATION

Reservation tables stores reservations of students

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| reservation\_id | integer | True | The unique primary key identifier of the reservation. |
| created\_date | date | True | The day and time the reservation was created. |
| start\_time | timestamp | True | The day and time the reservation is supposed to start. |
| end\_time | timestamp | True | The day and time the reservation is supposed to end. |
| student\_id | integer | True | The unique identifier of the student who made the reservation. |
| washing\_machine\_id | integer | True | The unique identifier of the machine the reservation was made for. |
| swap\_req\_id | integer | False | The unique identifier of the swap request that is for the reservation, if one exists. |

## «table» ROOM

Room table stores data of rooms in building.

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| room\_id | integer | True | The unique primary key identifier of the room. |
| number | integer | True | The number of the room in the building. |
| building\_id | integer | True | The unique identifier of the building the room is in. |

## «table» ROOM\_HISTORY

Room\_history table stores student stays dates in the room

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| student\_id | integer | True | The unique primary key identifier of the student who was in the room. |
| room\_id | integer | True | The unique primary key identifier of the room. |
| start\_date | date | True | The day the student began their stay in the given room. |
| end\_date | date | True | The day the student ended their stay in the given room. |

## «table» STUDENT

Student table stores all data of students

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| student\_id | integer | True | The unique primary key identifier of the student. |
| current\_occupent | boolean | True | If the student is a current occupant of the dorms. |
| penelty\_points | integer | True | The amount of penalty points the student holds. |
| first\_name | varchar(50) | True | The first name of the student. |
| last\_name | varchar(50) | True | The surname of the student. |

## «table» WASHING\_MACHINE

Washing\_machine table stores all data about washing machine

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| washing\_machine\_id | integer | True | The unique primary key identifier of the washing machine. |
| status | boolean | True | If the washing machine is currently in service. |
| floor\_number | integer | True | The floor number the washing machine is located in. |
| building\_id | integer | True | The unique primary key identifier of the building the machine is located in. |

## «table» SWAP\_REQUST

Swap\_request table stores all the information needed about a student requesting a reservation that is already take, otherwise known as a ‘swap request’.

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| swap\_req\_id | integer | True | The unique primary key identifier of the swap request. |
| time\_req | date | True | The time the swap request was created. |
| reason | varchar(100) | False | The reason the student wants a swap request, or a note the requesting student can leave if they want to. |
| status | boolean | True | The status of if the request was responded to. |
| student\_id | integer | True | The unique if of the student requesting the ID. |

## «table» COMPLAINT\_ABOUT\_STUDENT

Complaint about student table holds all the needed info of the complaint a student makes about the previous student who used the machine before them.

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| complaint\_student\_id | integer | True | The unique primary key identifier of the complaint. |
| created\_time | date | True | The time the complaint was created. |
| description | varchar(100) | True | The description of the complaint that the student is complaining about. |
| status | boolean | True | The status of if the request was responded to. |
| closed\_time | date | False | The time the complaint was responded to by the receptionist, if responded. |
| receptionist\_id | integer | False | The unique identifier of the receptionist that responded to the complaint. |
| student\_id | integer | True | The unique identifier of the student the complaint is about. |

## «table» PENALTIES

Penalties table holds the info about complaint that were found as valid by the receptionist.

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| penelties\_id | integer | True | The unique primary key identifier of the penelty. |
| points\_before | integer | True | The amount of points the student held before the complaint was approved. |
| start\_date | date | True | The day the penalty started. |
| complaint\_student\_id | integer | True | The unique primary key identifier of the complaint the penalty came from. |

## «table» RECEPTIONIST

The table Receptionist helps the information of the receptionist user.

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| receptionist\_id | integer | True | The unique primary key identifier of the receptionist. |
| first\_name | varchar(50) | True | The first name of the receptionist. |
| last\_name | varchar(50) | True | The surname of the receptionist. |
| building\_id | integer | True | The unique primary key identifier of the building the receptionist belongs to. |

## «table» COMPLAINT\_ABOUT\_WM

Complaint about wm table holds all the needed info of the complaint a student makes about the machine they were using.

| **Column name** | **Data type** | **Not null** | **Description** |
| --- | --- | --- | --- |
| complaint\_wm\_id | integer | True | The unique primary key identifier of the complaint. |
| created\_time | date | True | The time the complaint was created. |
| description | varchar(100) | True | The description of the complaint that the student is complaining about. |
| status | boolean | True | The status of if the request was responded to. |
| closed\_time | date | False | The time the complaint was responded to by the receptionist, if responded. |
| receptionist\_id | integer | False | The unique identifier of the receptionist that responded to the complaint. |
| student\_id | integer | True | The unique identifier of the student that made the complaint. |
| reservation\_id | integer | True | The unique identifier of the reservation that the complaint came from. |