**Group 24 Report**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arthur Trützschler von Falkenstein |  | Ivan Nestorov |  | Nada Mouman |
| Radu Gaghi  Yoon Hwan Jeong |  | Rounan Li |  | Silviu Marii |

# Introduction

At TU Delft we have a large campus, so the problem that was presented to us was to find a way to help lecturers and students alike to find free rooms across the entirety of the university grounds so they can give lectures, study or work in groups in spaces which suit their needs.

This was the main problem we had to address, but we also provide other functions besides reserving rooms. Users can rent bicycles from buildings in the campus which have them, and also order food from restaurants on campus based on their reservation for a room.

We designed an application called ***#insert name#*** to address this problem. It allows the user to browse through all the available rooms, find the rooms which suits their needs best and reserve it for a period of time. This can also be done in advance to make sure the room will not be reserved by someone else.

We approached the problem by splitting the problem in three parts: the GUI, the functionality and the database design. We decided on how the user interface was going to look; we designed a schema for the database and also decided on the most important features our application should have. We then split the group according to these three parts and started working in parallel.

**Product**

The first page you see when opening the app is the login screen. Here you input your credentials and are taken to the main page. On the right there is a sliding menu which gives quick access to the functions our app provides. There are three main buttons on this page, one for reserving a room, one for ordering food, and on the right there is a sliding menu which gives quick access to the functions our app provides.

The first button takes you to the reservation page. Here on the right you can see a table displaying all the rooms, the building they’re in and the room size. On the left the facilities available in the selected room are displayed. (insert something about filtering here). A room can be selected and you are taken to the date picker page which allows you to select a date, a start time and an end time for your presentation in fifteen-minute intervals.

The second button …..will write when it’s added

The third button takes you to the bike rental screen. Where you can see a list of all the buildings and the number of bikes that are available. If you select a building and rent a bike, a bike will unlock and you will be taken to another page where you can drop off the bike at any building on campus. If you have rented two bikes in a day, attempting to rent another one will result in receiving a message telling you that you have reached the limit.

On the slide menu you can access “My Reservations Page” where you can see the reservations you have made and you can also cancel any of them.

The admin page is only available to the admin, and it is also the On the admin page you can add, edit or delete rooms or building in the database. You can also cancel any reservation or delete user accounts. Admins are also the ones which authorize users, and they can do that too.

**Architecture**

Our application consists of two (arguably 3) main parts; the client-side and the server-side which also includes the database.

The first component of the server-side is the entity. These classes define the tables and relations in our database. Spring generates these tables and relations based on the attribute of each class and the respective annotations directly in the linked database. For example, the entity Room has an attribute “building” and the annotation “One-To-Many". This indicates that a building contains many rooms and Spring will set up a foreign key between these two tables in the database. Each entity has a corresponding repository and controller. Repositories encapsulate storage and retrieval for the before-mentioned entities. They are interfaces in which custom queries can be defined. Repositories contain many useful methods for finding instances in the database, but in order to have more complex queries we had to write some custom methods. Using the relations set up in the entities, Spring helps you generate queries for an entity using any attribute it has, including foreign keys. We also wrote native queries in SQL for adding tuples to the database. Controllers contain the business logic of our application. They contain many endpoints which serve different purposes such as retrieving data but also deleting or changing data from the database through the methods of the linked repository. Each method in the controller is mapped to a path on the server. For example, the filter method in the RoomController class has 4 request parameters. These can be included or not in the path depending on how they are set up and they are used in the query. Users can select rooms that have a certain size, a specific facility that they need, search for a room in a specific building, all, some or none of these conditions (*consider moving this phrase to product section*). To sum up, the entities define a template, the repositories hold instances of these entities and the controllers allow performing CRUD operations on the instances.

Write about client-side…

The first component of the client-side is the views. Each one of them contains only the “start” method which is used for closing the already existing stage and starting a new one with a new scene. The views are triggered by the user interactions. The extra Callback method and the unique lines of code in DatePickerView add the calendar on the GUI interface and also it loads the restrictions right away so that the user can view the available dates for a reservation. The parent of each scene is the content of a file of type FXML (all FXML files contain a main container -the parent- in which all other nodes/components reside). Each FXML file has a Controller connected to it, offering the so needed functionality through specifying the command onAction = “#methodToBeCalledInControllerClass”. Furthermore, the Controllers can influence the aspect of the view by declaring the node/container with the annotation @FXML. Also, the Controllers use entities for an easier store of the data received on request from the server-side after the deserialization process. Most Controllers have implemented the initialize() method that performs any necessary post-processing action. As an example, the initialize() in RoomListSceneController injects the data requested in the FXML fields after they are initialized and populated and thus avoiding unwanted errors and exceptions. A class worth mentioning is the DateTimePicker. It implements the DatePicker library adding a nicer format for the date, several listeners and ways for retrieving the data inserted by a user. Since the available time slots for a new reservation are not stored on the database, the DatePicker class retrieves the already existing reservations, opening and closing hours of the building in which the selected room is located, in order to generate the available ones. The communication between client-side and server-side is made in the ServerCommunication class, returning a String of type JSON.

Arthur has talk about UserManager, Ivan might add something about the CSS, style, etc.

**Responsible CS**

Yoon will add security things

**Alternative products**

Write about library reserving function from TU Delft

# REFERENCES (leave like this so we have a template)

1. Patricia S. Abril and Robert Plant, 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan, 2007), 36-44.

DOI[: https://doi.org/1](https://doi.org/)0.1145/1188913.1188915.

1. Sten Andler. 1979. Predicate path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*

*(POPL '79)*. ACM Press, New York, NY, 226-236.

DOI:https://doi.org/10.1145/567752.567774

1. Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago.

DOI:https://doi.org/10.1007/3-540-09237-4.

1. David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.).

Wiley, New York, NY..

