

# UNIBZ Roomfinder

## Finding Available Room to Study

Project Documentation by:

Danilo Fink

[dfink91@gmail.com](mailto:dfink91@gmail.com)

Luca Pellegrini

[luca9294@gmail.com](mailto:luca9294@gmail.com)

Faheem Shahid

[faheemshahid99@gmail.com](mailto:faheemshahid99@gmail.com)

Nuzula Muscha

[nuzula.muscha@gmail.com](mailto:nuzula.muscha@gmail.com)

Anjan Karmakar

[anjan8@gmail.com](mailto:anjan8@gmail.com)

Gabriele Minneci

[gabriele.minneci@gmail.com](mailto:gabriele.minneci@gmail.com)

Heider Jeffer

[hheider.jeffer@gmail.com](mailto:hheider.jeffer@gmail.com)

Srishti Gaihre

[Srishti.kristi12@gmail.com](mailto:Srishti.kristi12@gmail.com)

Document Status: Final Documentation

Date(s) : December 18<sup>th</sup>, 2017

Version : 2.0

Free University of Bozen-Bolzano

Faculty of Computer Science

Dominikanerplatz 3 - piazza Domenicani, 3

Italy - 39100, Bozen-Bolzano



# Table of contents

<b>Theoretical Fundament of the Project “UNIBZ Roomfinder”</b>	<b>4</b>
The Software Life Cycle Revisited	4
People Management and Team Organization	4
Software Quality	4
Cost Estimation	5
Estimation Procedure	5
Project Planning and Control	5
Requirement Engineering	5
Modelling	6
Software Architecture	6
Software Design	6
Software Testing	7
Software Maintenance	7
Software Tools	7
User Interface Design	8
Software Reusability	8
<b>Project Documentation and results</b>	<b>9</b>
The Software Life Cycle Revisited	9
People Management and Team Organization	9
Mintzberg’s Coordination Mechanism: Divisionalized Form	9
Team Organisation: Agile Team	9
Software Quality	10
Cost Estimation	10
EARLY STAGE: Early Effort and Cost Estimation	10
Cost Determination	10
KLOC Estimation based on Object modules	10
Effort Estimation based on KLOC value and Complexity	11
Early Stage Conclusions:	11
FINAL STAGE: Final Effort and Cost Estimation	12
Cost Determination	12
KLOC Count Criterion	12
Final KLOC Count and Effort Estimation	13
Final Stage Conclusions	14
Project Planning and Control	14
Analysis of the characteristics of our project	14
The Design Problem	14
Requirement Engineering	16
Requirements	16
Modelling	17
1. Entity-relationship Diagram	17
2. Finite State Machine	17
	2

3. Data Flow Diagram	18
4. Sequence Diagram	19
5. Use case Diagram	19
6. Object Relationship and Hierarchy	20
Software Architecture	21
Software Design	22
Software Testing	23
Test Plan	24
Software Maintenance	25
Software Tools	25
Management and cooperation	25
Development	26
Deployment	26
User Interface Design	27
Presentation Specification	27
Dialog specification	29
Users Task: View All the freerooms	29
User Goal: View the availability of one room	30
Software Complexity	31

# Theoretical Fundament of the Project “UNIBZ Roomfinder”

In this section we document which theoretical principles we applied to our project. Most of the discussed principles are described in the book “Software Engineering: Principles and Practice” by Hans Vliet. In the later section “[Project Documentation and results](#)” we will discuss how we used them and show the results.

## The Software Life Cycle Revisited

In our project we decided to use agile methods to develop the application. We used agile model because it has much more flexibility with respect to traditional methods like the waterfall model. The agile process allows us to break down each of task into prioritized requirements and delivering each of task within interactive cycle. Each iteration was reviewed and assessed in a meeting every week among the project team. The feedback and comment gained from the meeting are used to determine the next step in project development.

Our way of working is quite near to the Scrum Model. In each meeting we had a working version of the program with the requirements, chosen in the last meeting, implemented.

In the meeting we commented the work of others and prioritized the requirements to be implemented in the next sprint. We used Trello in order to manage the Backlog Board and have an updated view of the status of the tasks.

## People Management and Team Organization

After the first weeks and meetings, we tried to fit our coordination and organization into some of the discussed practices from the book. To find out how to manage the different people in our team we first looked into Mintzberg's Coordination Mechanisms. Our groups organization was carried out quite natural, but we still tried to fit it to one team organisation techniques discussed in the book.

## Software Quality

During our initial discussions, we inherently agreed upon five quality factors that our project must satisfy. The factors being:

- Correctness
- Reliability
- Efficiency
- Usability
- Portability

With Correctness, we wanted to ensure that our application satisfies its specifications and fulfills the basic objectives of the user which is to find available rooms in the university.

With Reliability, we wanted to ensure that our application is reliable whereby it performs its intended function as and when needed with a high degree of precision.

With Efficiency, we wanted to ensure that the computing resources and code required to perform the basic function must not be overwhelming for the device to handle, and it should return the results within a few seconds.

With Usability, we wanted to ensure that our application was easy to use, and the effort required to learn, operate and prepare the input, is minimal.

And with Portability, we wanted to ensure that our application can run on devices with different hardware/software configuration environment and resources. That our device should be cross-platform.

To maintain the software quality, we depend on a simple version of the Personal Software Process technique.

## Cost Estimation

Software development costs are usually difficult to estimate reliably in the early phases of its production. And even though there are a number of cost estimation formulas, we know that blind application of any of the formulae from existing models will not yield a good estimate.

In our case we have a project with angular which is a relatively recent technology, therefore it is hard to find extensive accounts reliable data with a significant sample size.

Consequently, we have chosen to determine the cost of our project by a combination of expert-based technique and simplified cocomo model.

## Estimation Procedure

First we determine the estimate of lines of code required in terms of Object points, and make a schedule of the tentative time required to complete the project based on inputs provided by the team of expert-developers. We can then calculate the estimate for the man-months required, and thereby derive the price for the entire project based on the man-month price-value evaluation. Later we compare it with the actual time take, the KLOC, and the costs involved to verify and consolidate our estimates.

The initial and final estimation procedures and their comparison are discussed later.

## Project Planning and Control

To get a better viewpoint at our project we analyzed its characteristics (certainties of product, process and resources) to be able to match it with one of the four archetypal control situations (Realization, Allocation, Design, Experimental). After this we prepared a Work Breakdown Structure to find possible activities during our project. We estimated the costs of the different activities and assigned prerequisites to each of them. We also made a Gantt Chart, but we didn't find it fitting our development style after seeing how we were switching between activities.

## Requirement Engineering

We discussed the functional and nonfunctional requirements especially in the earlier iterations of the project.

We made at first general requirements, after having created the first scratch of the User Interface we improved them as they are now.

We focused more on the product functionalities, defining mainly the requirements needed for the project development and the most necessary for letting it work.

Since we have decided the requirements by ourselves, they are general so that we had more options for the design and implementation phases.

We manually verified and validated the requirements during the weekly meetings.

## Modelling

To have clear understanding about our requirement, we have developed modeling diagram such as Entity Relation Diagram, Sequence Diagram, Data Flow Diagram and Finite State Machine. The diagrams explain how the system should behave in relation to other functions and/or classes.

These modelling techniques are often used in Software Process Management, Requirements Engineering and Design Engineering, where models can vary from basic sketch of system to classified behaviour of systems. Most common notations that are used in making models are Box and Line sketches, and UML diagrams. These Box and Line sketches, and UML diagrams are helpful in communication between the user and the requirements engineer and between the requirements engineer and the developer.

In Unified Modeling Language(UML) boxes may denote a part of system or a single component of system, and Lines may denote the relation between the components. By using UML notations we also get to know the flow the system and communication between the different parts of the systems. Essentially, modelling techniques provide a design framework to the developers to enable them to understand and deconstruct major systems as different subsystems.

## Software Architecture

In software architecture, we design the architecture of how the application will retrieve information about room and its availabilities between client and server side.

We use our proxy server to get the list of all the lectures (and the lecture rooms) of a particular day from UNIBZ API, while we get the information about the list of all the rooms, from In-Memory API. Our services takes the data from both the APIs and compute the free rooms. The filters are applied to the obtained data and displayed in the User Interface.

The architecture and the description is described in detail in the later section.

## Software Design

We have used the Agile approach also for the software design. We have designed our product through different levels of abstraction: at the beginning we focused on the core components and the general modules of the software, without designing all the specific objects and relations inside them. In the successive iterations we have modelled the different modules we were using in more details.

The design of the system is dependent by the structure of the framework we have used. The modularity we have created has been useful for assigning the implementation tasks in the

team, but the more we advanced in the implementation, the more a change could affect the functionalities of other modules. As a consequence more coupling is added, we will see examples in the documentation and results section.

We have tried to put as much information hiding as we could in the various components, so that the program results with less coupling and more reusability. The angular framework gave us guidelines to relate some type of objects instead of others, for instance the services have complete access to the specific components they support, but not to the others.

Based on the fact that our product is not big, we didn't focus too much on its complexity. We have tried to make a good design following the main best practices.

## Software Testing

As already said, our project is based on the Angular framework. Angular provides essentially two main tools for testing *Karma* and *Jasmine*.

We used *Karma* to test the services.

Karma is a JavaScript command line tool that can be used to spawn a web server which loads your application's source code and executes your tests. You can configure Karma to run against a number of browsers, which is useful for being confident that your application works on all browsers you need to support. Karma is executed on the command line and will display the results of your tests on the command line once they have run in the browser. Karma is a NodeJS application, and should be installed through npm/yarn.

We used *Jasmine* to test the angular components.

Jasmine is a behavior driven development framework for JavaScript that has become the most popular choice for testing Angular applications. Jasmine provides functions to help with structuring your tests and also making assertions. As your tests grow, keeping them well structured and documented is vital, and Jasmine helps achieve this.

## Software Maintenance

For the software maintenance part we did not have plan yet, because the system is not accessible to all unibz students yet. Consequently, we are not able to measure the performance and accessibility of the system. Thus, further maintenance was thus not taken into consideration.

However, maintenance of the system is absolutely essential once the application comes into widespread operation, in order to make sure the system functions and apis are working as expected. Furthermore, the user feedback may give us valuable insight on the working of the system, and their feedback can help evolve the system sustainably over time.

## Software Tools

At the beginning we have chosen the tools to use for the different task according to how they fit with them and our previous experience. The mandatory tools for using the specific technology we have chosen had to be learnt by the entire team: we spent a significant amount of time in the first phase in learning and getting used to our technology stack.

For the specific tasks, each of us chose the best tools we already knew. We shared previous personal experiences with the other team members for common problems encountered in our learning curve. Since the project is small, the main tools are used to support the development phases, yet we have also used some tools for the team cooperation.

## User Interface Design

After the requirements were defined, we started with specifying the User Interface in details. We did brainstorming and came up with a mockup UI using Paint as our preliminary User Interface, which is then refined multiple times throughout the project implementation when we find any possible improvements.

For specifying the user actions, we used User Action Notation technique which is described in this section in details. For evaluation of the user interface, we used the Heuristics evaluation techniques where we used the principles proposed by Nielsen, as given in the book. Here are some of the principles we used to evaluate the usability of our user interface.

1. **Match between system and the real world:** We checked if our UI is understandable by our user.
2. **User control and freedom:** Our project provides the user with the freedom to user to have their own filters.
3. **Consistency and standards:** We evaluated the consistency of our project and assured that our project follows the standards. For example, we use the asterisk to specify the field is required.
4. **Recognition rather than recall:** We evaluated the project and assured that the user does not have to remember so we provide the user with drop down menus.
5. **Aesthetic and minimalist design:** We used the aesthetic and minimalist design. We evaluated and assured that our project's UI does not overload user with unnecessary information.

## Software Reusability

We plan to use existing API from university that sends information such as: Building, Room, Time and Date into our application. We also use angular.io as our framework to build our application. We see that this API is compatible with the angular framework and it serves many useful data that can be used in our application. Therefore, we are reusing the existing unibz API for university lectures. Other than that the angular framework allows us to reuse some of the predefined features.



# Project Documentation and results

In this section we want to show and discuss the results we achieved in our project.

## The Software Life Cycle Revisited

At beginning of the project, we had to choose which software methodology best suited our project needs. Do we want to be AGILE or do we want to follow a more traditional approach? Being a young team and foreseeing high probabilities to have frequent changes in the requirements, we opted for an AGILE approach.

Being AGILE, we did not spend a lot of time doing a very detailed requirements analysis and elicitation at the beginning, but we selected only some basic features to implement for the first sprint, being aware that the requirements will change and will be more detailed going on with the project.

Being the team composed of 8 members, we also decided to work in *pair-programming*. Being not a real project, we did not have users/clients that gave us a feedback: at each meeting we had to simulate the users giving opinion/feedback about the work done by the other pairs.

Have we been really AGILE? For sure. At each meeting, we had a working improved version of the application and each pair said what it has been done and the problems encountered. In the second part of the meeting, we discussed about the tasks for next iteration.

## People Management and Team Organization

To be able to find the right mechanisms and organisations, our team first needed to get to know each other a bit, to find out which way we wanted to go.

### Mintzberg's Coordination Mechanism: Divisionalized Form

Looking at the different coordination mechanisms we found that a sort of divisionalized form was the one we were actually already carrying out. The tasks that were assigned could be pursued in total autonomy and in weekly meetings the group checked if the goals have been reached in a sufficient manner. For us this was a good way of working, since all of us came from different backgrounds and had different experiences.

Once the requirements and the architecture was set, we tried to split the group into subgroups with different responsibilities. We had people being responsible for the backend, a second group was responsible for the frontend and design and the third subgroup took to testing. The weekly meetings have been held to check whether a group run into problems and what the next steps would be. These technique allowed us to verify and validate activities, to discover errors or assess the quality of the code.

### Team Organisation: Agile Team

As stated before we assigned most tasks of the project to mainly 3 teams carrying out a specified role. In a normal case we had 2 people working on each team, and 2 people being jumpers meaning they would help where needed. But also each team had the possibility to ask for help from other teams, resulting in a reassignment of a person of one team to

another. We clearly didn't want to propose any hierarchical organisation because we wanted everyone to feel on the same level and have equal opportunities at the beginning of the project. Throughout the project another one of our goals was always to find the right people for the tasks that were due.

## Software Quality

The quality factors we included in our project, namely: Correctness, Reliability, Efficiency, Usability, and Portability were validated at every incremental stage of the project by checking out how many requirements were implemented at the end of the every sprint and whether or not they support the quality factors. In cases, where the quality factors were not supported, the task was set for the new sprint and the quality demands were met.

The three steps in the process:

> Data Collection > Discussion and Analysis > Constant Improvement

## Cost Estimation

As discussed earlier, most major algorithmic cost-estimation models are based on extensive data about past projects. But since we lack enough quantitative data about past projects which use the same technology stack as we do, we cannot possibly rely solely on a predefined model.

Therefore, we have chosen to determine the cost for our project based on a combination of expert-based techniques and simplified cocomo.

### **EARLY STAGE:** *Early Effort and Cost Estimation*

#### Cost Determination

First we determine the estimate of lines of code required in terms of Object points, and make a schedule of the tentative time required to complete the project based on inputs provided by the team of expert-developers. We can then calculate the estimate for the man-months required, and thereby derive the price for the entire project based on the man-month price-value evaluation.

#### KLOC Estimation based on Object modules

MODULE	KLOC	COMPLEXITY	TEAM EXPERIENCE	TEAM CAPABILITY
1. Search functionality a. filtering	0.1	medium	low	medium
2. API communication	0.5	medium	low	low

3. UI	0.5	low	low	high
4. Testing	0.2	medium	low	medium

Objects	Complexity Weight		
	Low	Medium	High
Modules	1	2	3

Effort Estimation based on KLOC value and Complexity

ACTIVITY	DURATION	CONSTRAINTS/ PREREQUISITES
1. Requirements engineering	7 days	-
2. Architecture of the software	8 days	1
3. Design the user interface	7 days	2
4. Testing plan	3 days	3
5. Development	30 days	4 (Deadline: 1st December)
6. Testing a. Part A b. Final Testing	20 days in total 15 days 5 days	3 and 4 3 4
7. Deployment	4 days	5 and 6

Early Stage Conclusions:

Our estimation for the completion of the project stands at 79 days in total for a team of 8 members, working on an average of 1 hour daily per person.

By this estimate our project would need a total of:

79 days x 1 hours/member x 8 members = 632 hours

In our case, we consider 1 man-month to be equal to 160 work hours. (Based on a 40 hour work-week schedule, with 4 weeks of work every month, 40 hrs/week \* 4 weeks = 160 hrs)  
Taking an average rate of €15/hr for each work hour, each man-month calculates to €2400.

Since each man-month is equal to 160 hours.

The number of man-months needed for the project would be  $632/160 = 3.95$  man-months

The cost of 3.95 man-months would be  $= 3.95 \times 2400 \text{ €} = \mathbf{9480 \text{ €}}$

### **FINAL STAGE:** *Final Effort and Cost Estimation*

#### Cost Determination

Here we check the actual lines of code required in terms of Object points and compare it with our early estimation, and make a count of the actual time required to complete the project based on real effort put forth by the team of developers. Also, we then calculate the man-months involved, and thereby derive the price for the entire project.

#### KLOC Count Criterion

In our cost and effort estimation, we use lines of code to determine the time required to complete a function or object point. However, not all lines of code in the project files are counted in the final cost estimation to be fair to the client. And to make this clear, here's the count criterion tally which lists and shows the inclusion/exclusion of code by type.

	INCLUDES	EXCLUDES
<b>Statement Type</b>		
	Executable statements	Compiler directives
	Non-executable statements	Comments
	Declarations	Banners
	Inline code comments	Non-blank spacers
		Line breaks
<b>How is it produced</b>		
	Programmed	Auto-generated
	Converted w/ translators	Removed/Deleted
	Reused w/out change	
	Modified	
<b>Origin</b>		
	Custom reuse library	Unmodified language library
	Adaptations from prior work	Unmodified utility library
		COTS
		GFS

#### Final KLOC Count and Effort Estimation

MODULE	KLOC	COMPLEXITY	TEAM EXPERIENCE	TEAM CAPABILITY
1. Search functionality a. filtering	0.4	medium	medium	medium
2. API communication	0.5	high	low	low

3. UI	0.5	medium	medium	high
4. Testing	0.2	medium	low	medium

### Final Stage Conclusions

Therefore, the total lines of code is noted and calculated to be = 1.6 KLOC

The number of actual hours required to complete the project = 78 days x 1.1 hours/member  
x 8 members = 686 hours

As discussed, we consider 1 man-month to be equal to 160 work hours. (Based on a 40 hour work-week schedule, with 4 weeks of work every month, 40 hrs/week \* 4 weeks = 160 hrs)  
Taking an average rate of €15/hr for each work hour, each man-month calculates to €2400.

So, for our calculations, we shall equate 1 man-month to be equal to €2400.

Now, the number of man-months =  $686/160 = 4.29$  man-months

Total cost =  $4.29 \times 2400 \text{ €} = \mathbf{10296 \text{ €}}$

The final cost differed from the initial estimate, increased by 816 € (8.6%) from initial estimate.

## Project Planning and Control

### Analysis of the characteristics of our project

Product certainty	High	When looking at the idea, the requirements and the systems goals felt clear.
Process certainty	Low	We decided to use tools and technologies that most of us have never dealt with, to expand our horizons
Resource certainty	Low	We didn't know each other and everyone's capabilities.

This analysis leads to a specific archetypal control situation: The Design Problem

### The Design Problem

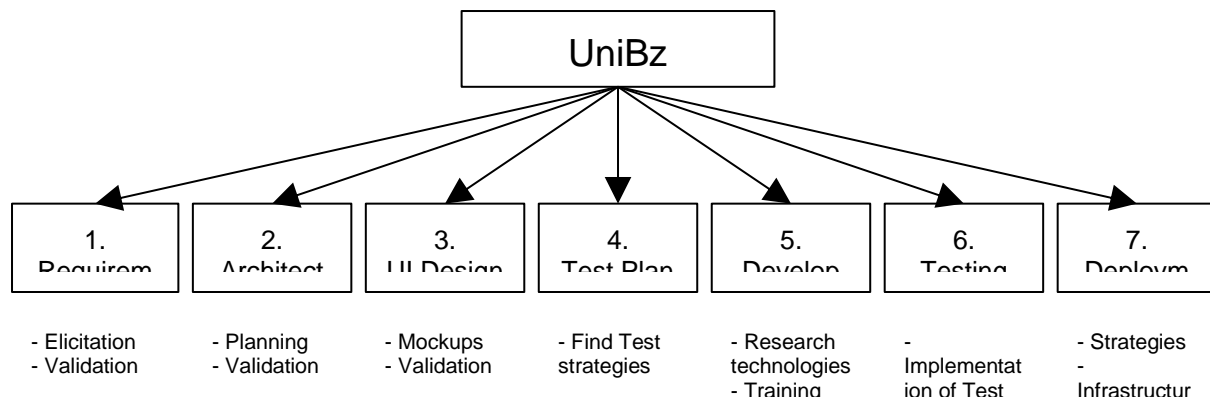
1. Goals in control: Control of the process
2. Coordination mechanism: Standardization of the process
3. Development Strategy: Incremental
4. Cost estimation: Estimations

We were looking for a good way to update each other with the help of tools and occasional meetings. But tools have been found cumbersome to be the only foundation of communication, that was when we started scheduling fixed meetings on Wednesdays, where we held a short meeting for recap of the week before, discussions about adjustments

and problems and finally to assign tasks for the next week. From that moment on the project management tools (PMT) were replaced by our weekly meetings as the main tool for communication. PMTs were used to support the meetings, we could write down the decisions, assign and track the work with them.

These weekly meetings allowed us to follow an incremental development approach, where we have been able to change our requirements, architectures, the allocation of work every week.

Cost estimation was especially hard for us, but we tried to predict cost in days for every major task that we identified in a Work Breakdown Structure (WBS).

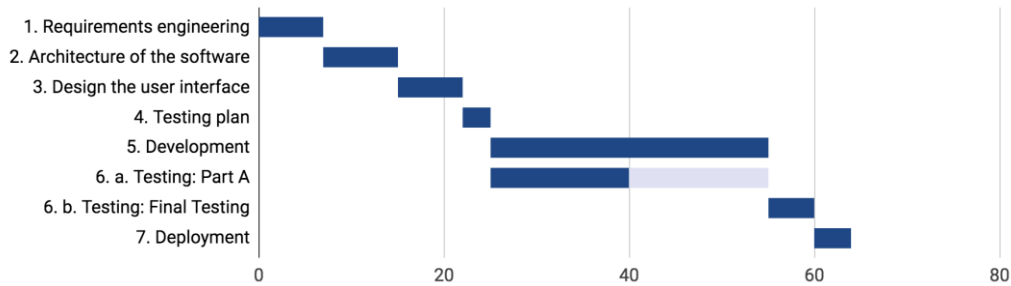


Work Breakdown Structure of the Project "UniBz Roomfinder"

Activity	Duration	Prerequisites
1. Requirements engineering a. Elicitation b. Validation	7	-
2. Architecture a. Planning b. Validation	8	1, 5.a, 5.b
3. UI Design a. Mockups b. Validation	7	1
4. Test Plan a. Find test strategies	3	5.a, 5.b
5. Development a. Research technologies b. Training c. Implementation	30	1 1 1, 2
6. Testing a. Implementation of test plan	20	4
7. Deployment a. Strategies b. Infrastructure c. Test deployment	4	5.a, 5.b 5.a, 5.b 5.c

Table with the durations and the prerequisites of activities

After we first tried to use a Gantt-Chart for the planning of our activities, we decided that it does not fit our incremental development strategy. We found ourselves jumping back and forth between activities or doing them at the same time. Simple examples are: we had to rewrite some requirements or revisit the architecture, after we found out about some new functional requirements. The Gantt-Chart was too inflexible for our project needs and was therefore discarded.



An early version of a Gantt-Chart, that we discarded shortly after.

## Requirement Engineering

At the beginning we defined general requirements, during the early iterations we put more details on them. The final requirements have been defined after the third iteration.

### Requirements

Main goal: Show available rooms at a given point in time. (rooms are lecture halls in the University)

1. Show the period of time a room is free and where it is.
2. Show the rooms that are available during a specified time range:
  - main ranges
    - morning
    - afternoon
    - evening
    - entire day
  - chosen by the user
3. Show availability of a specific room in a specific day
4. Show capacity of a room (*optional*).
5. Filter rooms by building. Room names (example: E311) consist of the the letter of the building (example: E), the floor (example: 3 for 3rd floor) and the room number (example: 11)
6. The application should run on multiple platforms: mobile (iOS, Android) or desktop (windows, macOS, linux)

Non functional requirements:

- The system should have reliability to be accessed by users concurrently without any downtime. e.g. 100 user accessing at the same time (We didn't complete this requirement because of the lack of users)



- Usability: design a simple graphical user interface to use the application. It has to be mobile friendly and it should follow at least 3 Nielsen principles. (See the UI Design Chapter)

## Modelling

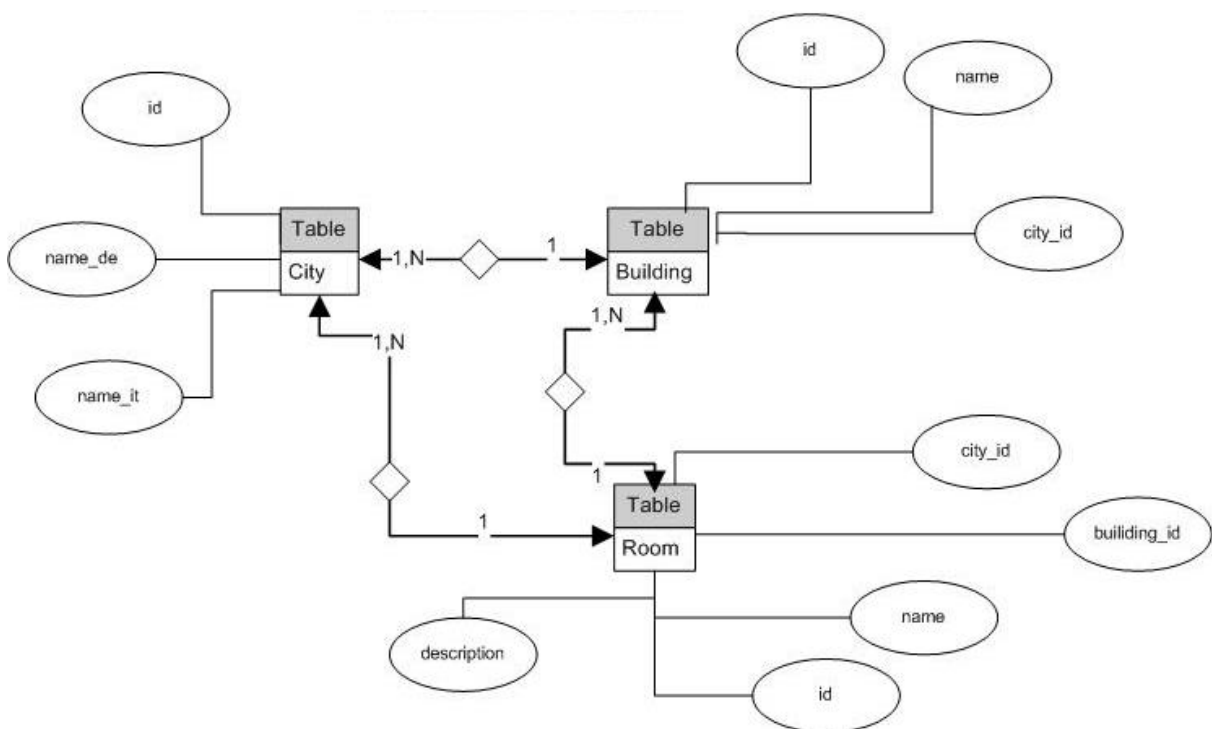
### 1. Entity-relationship Diagram

Entity relationship is a modelling technique in which we can extract the possible entities of a system and its attributes. And after defining entities we make relationships between them.

For the UNIBZ Roomfinder we have entities with their attributes like:

- City (id, name\_de, name\_it)
- Building (id, name, city\_id)
- Room (id, city\_id, building\_id, name, description)

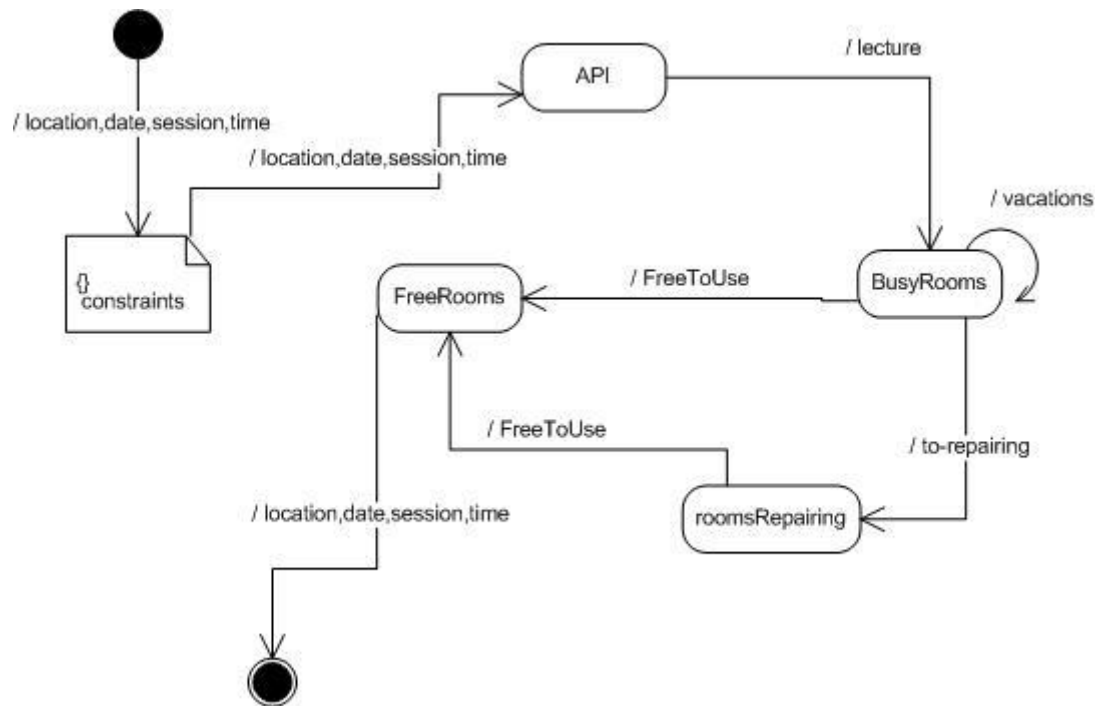
The following ER diagram shows the relationships between these entities with its cardinality between entities.



Entity-Relationship diagram

### 2. Finite State Machine

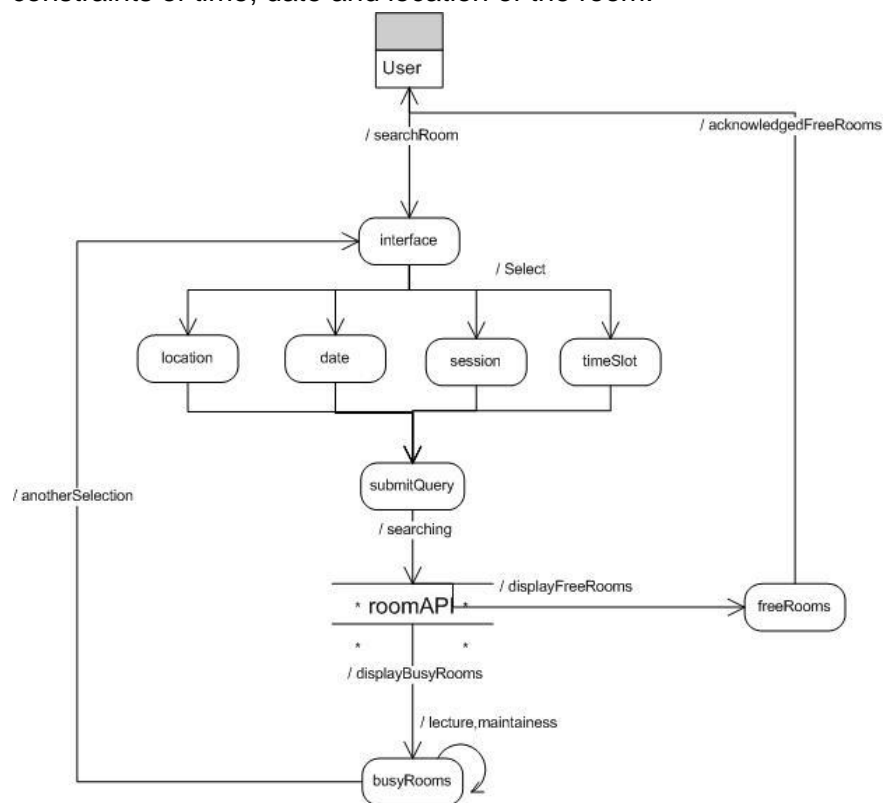
In modelling a Finite State Machine (FSM) describes the states of the system and the transitions between states. For UNIBZ Roomfinder we have different states of a room. In the following FSM diagram we can see that a room is in RoomBusy state during lecture or when room is in maintenance. And the state of the room will change to FreeRooms as soon as there is no lecture and it is ready for students to use it.



Finite State Machine for rooms of UNIBZ Roomfinder

### 3. Data Flow Diagram

A Data Flow Diagram describes a system as a set of process and shows the flow of data that links these processes. The following Data Flow Diagram shows that a user can find free rooms from UNIBZ Roomfinder through an interface. Users can search a free room by entering the constraints of time, date and location of the room.

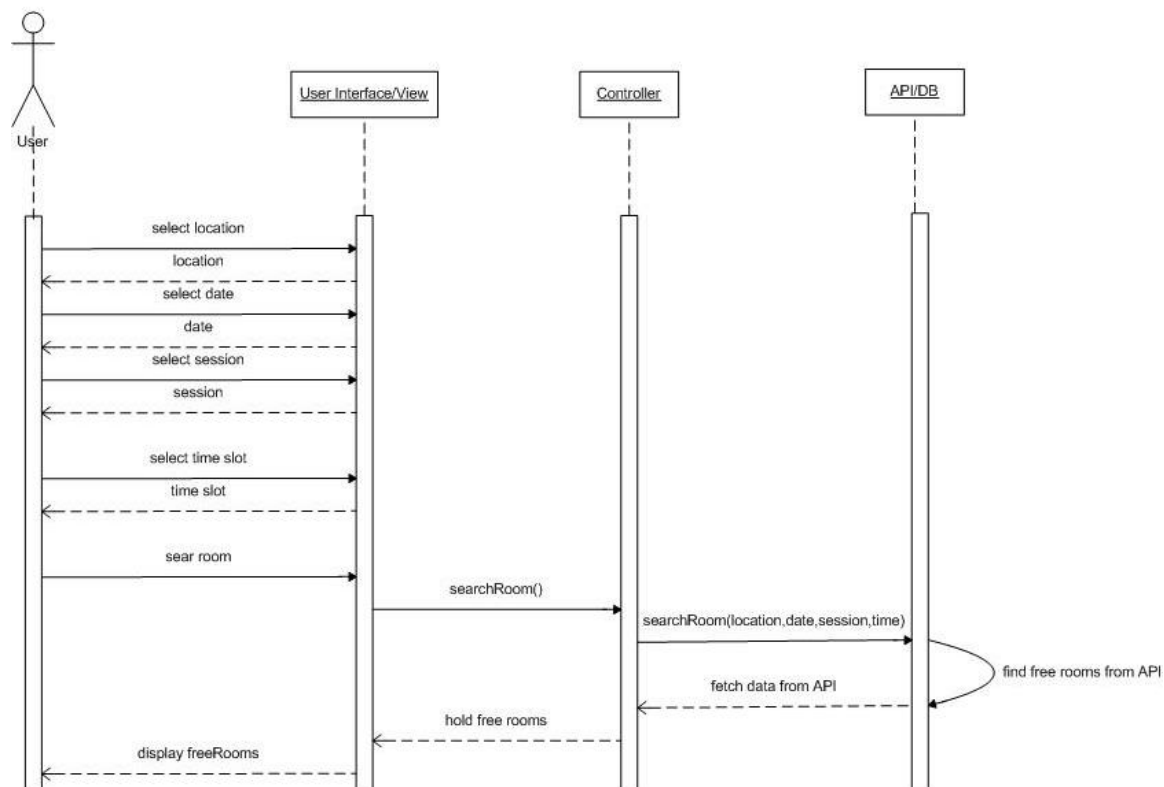


Data Flow Diagram

#### 4. Sequence Diagram

In a system objects send messages to each other to communicate. To complete a specific task, messages are exchanged between objects and have to follow a specific sequence to make sense. A sequence diagram not only describes the communication between objects but also the sequence of the messages between objects. The following Sequence Diagram of UNIBZ RoomFinder shows the sequence of messages between the user, the interface and the services/apis after every message.

Messages are shown as labeled arcs from one object to another. The arrangement of messages in vertical indicates order sequence of messages and their interactions between a user, an interface of the system. An object that handles operations i.e. controller and the API. The first message comes from an outside, unknown source.



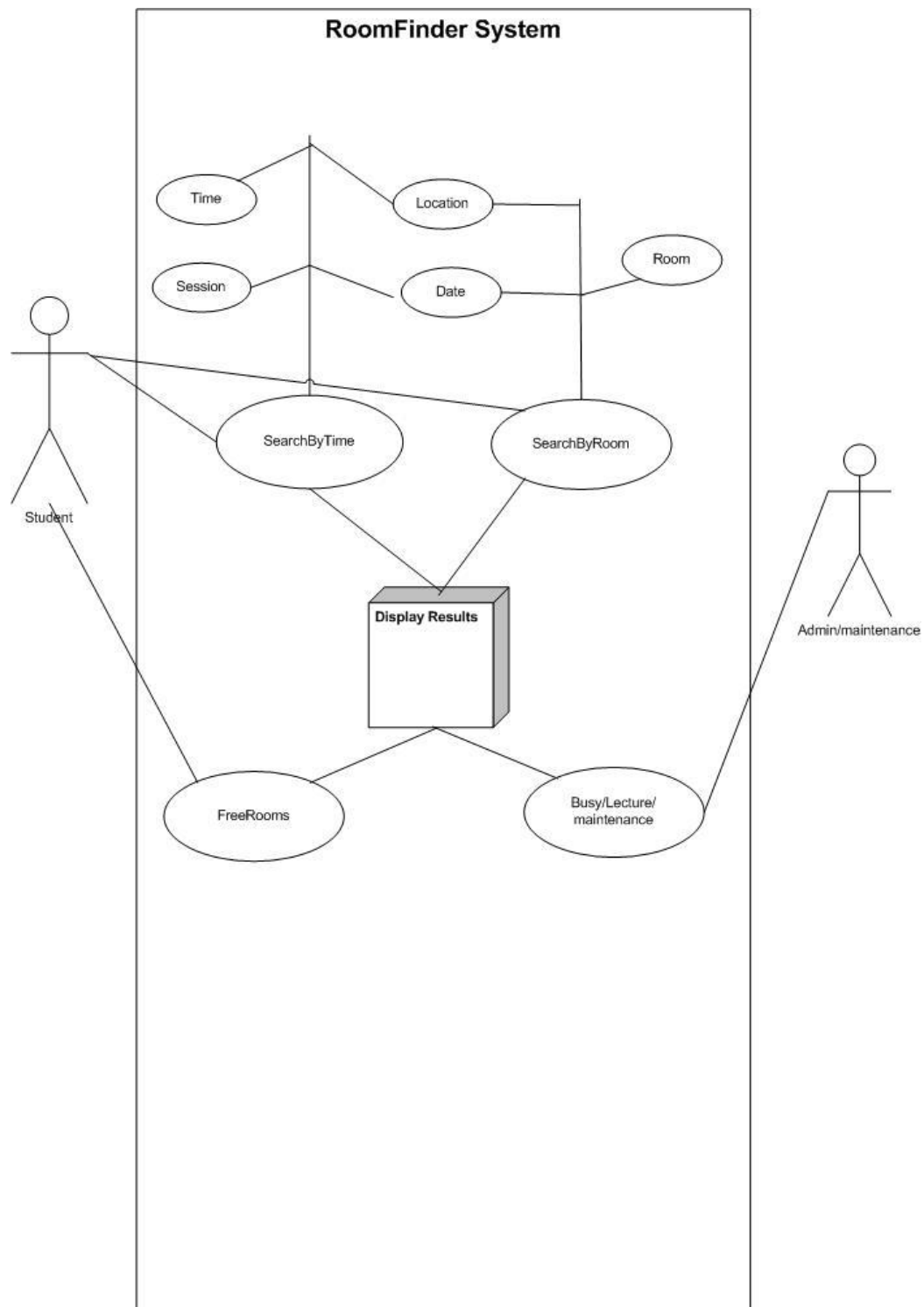
Sequence diagram

#### 5. Use case Diagram

Use cases are scenario or stories that describe how a specific task will execute. The use case diagram is an overview of different use cases. Use cases are enclosed in a rectangle that shows the boundary of the system. An actor that directly interacts with use cases is positioned outside of such rectangle.

In UNIBZ Roomfinder the main actor that interacts with system is a student. To find free rooms in the university through our application a student has two options, that generate two use cases:

- Search by time
- Search by room

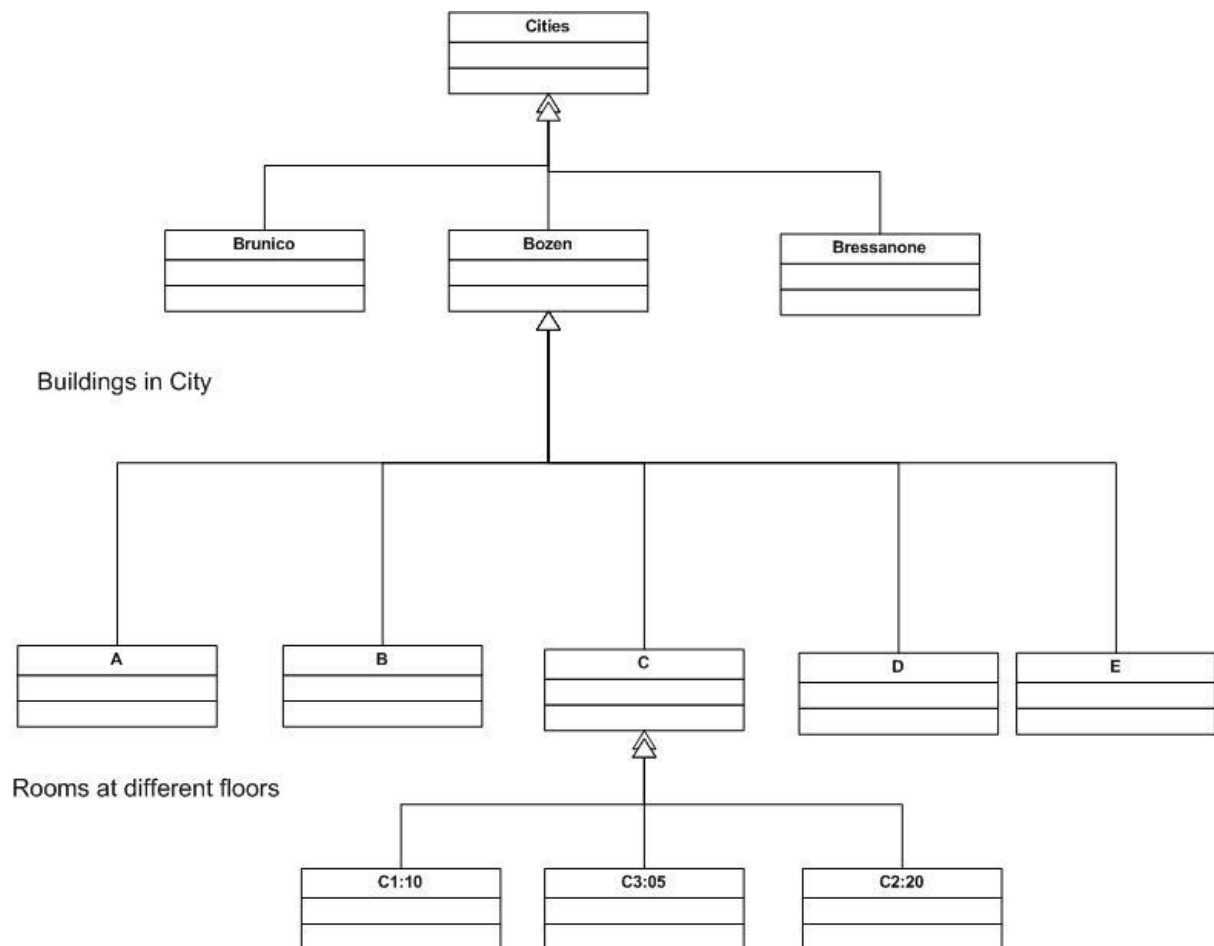


Use Case Diagram

## 6. Object Relationship and Hierarchy

Generally we are not talking about the individual objects. Our purpose is to identify the main objects of UNIBZ Roomfinder with their attributes and then relate these objects to each other. The relationships between objects of our system can be expressed in a Classification Hierarchical Structure.

The following figure shows how rooms at different floors are generalized into a specific building (C) and how buildings are generalized into a specific city (Bozen) which is the top level of this hierarchy.



Object Relationship and Hierarchy Diagram

## Software Architecture

Our application basically uses the two services that provide data about the timetable and rooms.

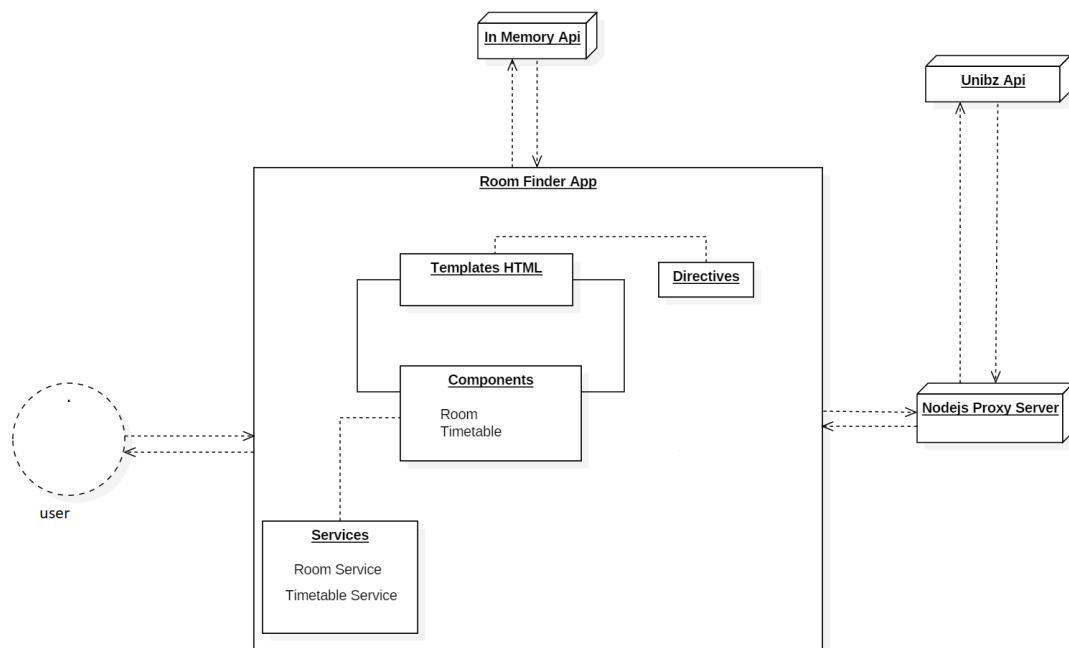
The timetable service was designed to deliver an array of lectures for a specified time range and city. At first we were trying to contact the UNIBZ-Server directly, but it didn't work from the web application because it was using AJAX-requests. Such requests fall under the so called Cross-Origin-Request policy that servers need to allow specifically. Since we were not having access to the UNIBZ-Server to change its settings, we had to change our architecture and add a proxy server as intermediate between our application and the timetable-api from the university. We implemented the proxy server with a framework called express.js. Our applications timetable service could now send requests to our proxy server, that manipulates them and forwards them to the timetable-api. The response from the api is again forwarded to our web application.

The room service can be used to retrieve data about all possible rooms from a so called In-Memory-API. It has a static list of all the rooms along with the building and city information.

Additionally the room services can take an array of lectures and a time range as an input to calculate the rooms that are not occupied during the given time range.

The information is then manipulated by the Home Component that provides the main interface to the user.

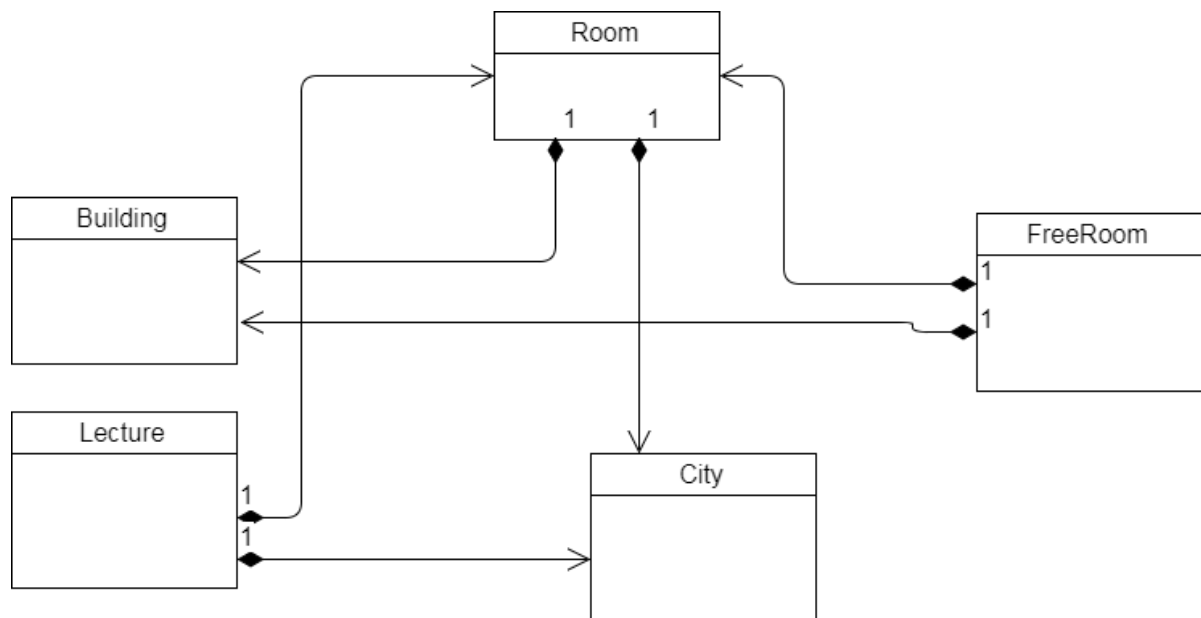
The diagram below represents an abstraction of the architecture of our system.



Software Architecture of Room Finder App

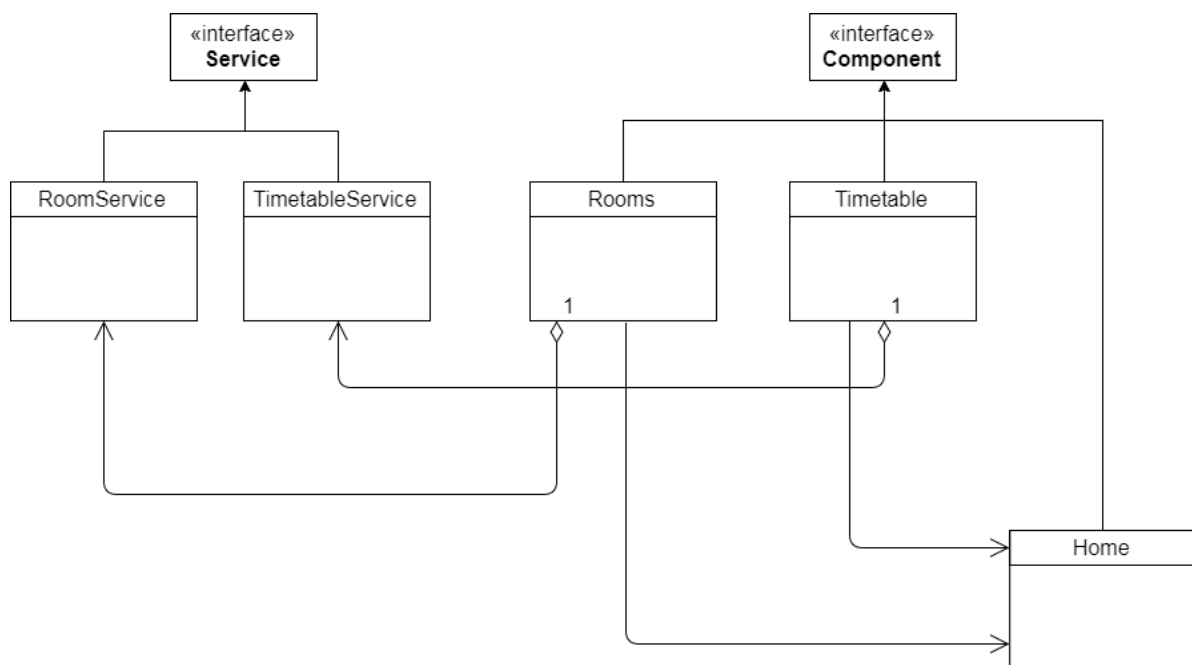
## Software Design

As we can see in the Modelling chapter, the data structure is made according to the information we want to show to the user and the ones that are given by the University API. The classes store the information and support the components and services based on the Angular Architecture.



UML Class Diagram of the Data Classes (Model)

The Home Component creates and manages the html page. Rooms and Timetable use their respective service to make the operations that are requested by the Home component.



UML Class Diagram of the Services and Components (Controller and View)



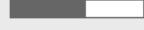
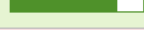

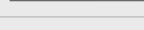
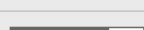
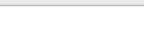
## Software Testing

Testing proved to be a hard task to tackle. The testing framework around Angular and Typescript was new to all of us and therefore required intensive research and training to have some useful tests running. Throughout the project we used Jasmine Testing Framework and Karma Test Runner, which are two tools that come with Angular. A third tool is Protractor that can be used to write end-to-end (e2e) tests.

The goal at the beginning was to reach more than 90% branch coverage, but that goal was not met on 18<sup>th</sup> december 2017.

### All files

58.72% Statements 256/436 61.54% Branches 96/156 45.54% Functions 51/112 57.33% Lines 223/389

File		Statements		Branches		Functions		Lines	
src		100%	16/16	100%	0/0	100%	1/1	100%	16/16
src/app		88%	22/25	69.23%	9/13	77.78%	7/9	85%	17/20
src/app/classes		57.5%	23/40	37.5%	3/8	40%	4/10	55.56%	20/36
src/app/help		80%	8/10	75%	6/8	33.33%	1/3	75%	6/8
src/app/home		20.49%	25/122	54.55%	18/33	4%	1/25	20.54%	23/112
src/app/room		63.16%	12/19	75%	12/16	20%	1/5	62.5%	10/16
src/app/rooms		58.82%	10/17	76.92%	10/13	20%	1/5	61.54%	8/13
src/app/services		74.87%	140/187	58.46%	38/65	64.81%	35/54	73.21%	123/168

Coverage report

### Test Plan

Our Test Plan was to first look into how to test services, as our first priority was to make sure that the data that we use in the web application is correct.

Our application essentially calls two services: one running locally, that provides data about the rooms, buildings and cities, another one provided by the university in order to get data about the timetable. At the beginning, we tried to make simple calls directly the services without success. To unit test our services, that usually run asynchronously, we had to provide mock-responses of the web-apis, they were using themselves. This ensures that while running tests on our “units” we don’t run into problems that are caused by components that we cannot influence, e.g. the UNIBZ Server.

Second priority was given to testing our components and their behaviour. Testing the different components with Jasmine has not been an easy task for our group, that has never tested Angular components. We were able to test the correctness of different web-page fields (values corresponded to given test values), but so far we have not figured out the right way to simulate user actions and validate the results.

The third part in testing would have been e2e tests, but we were not yet able to implement any of them, as we run out of time.



# Karma v1.7.1 - connected

Chrome 63.0.3239 (Mac OS X 10.13.1) is idle

Jasmine 2.6.4



17 specs, 0 failures

```
1st tests
  true is true
AppComponent
HelpComponent
MockBackend HeroService Example
  getHeroes() should query current service url
  getHeroes() should return some heroes
  getHeroes() while server is down
HomeComponent
RoomComponent
RoomsComponent
MockBackend RoomService
  getRooms() should return 139 rooms of the unibz
  getCities() should return the 3 Cities of unibz
  getBuildingsInCity(1) should return all builds of the Unibz Bolzano (6)
  getRoomsInCity(3) should return all rooms of the Unibz Brunico (11)
  getCity(1) should return City Object of Bolzano
  getBuilding(1) should return Building Object A of Bolzano
  getRoom(1) should return Room Object E312 of Bolzano
  calcFreeRooms at 6th December should return some results
InMemoryRoomService
MockBackend TimetableService
  getLecturesTest() should query current service url
  getLecturesTest() should return some testdata - fakeAsync
  getLecturesTest() should return some testdata
  getLecturesTest() while server is down
  getLecturesInCityForDay(city: number, day: Date) for the 6th should return something
```

A screenshot of a report by Karma

## Software Maintenance

Since the system is currently built new and the number of user who access are not so many. the performance of system still can be managed by our server, so far we do not have any maintenance activity yet. In future, if the system is eventually accessible to all students of UNIBZ, we will review the system performance and upgrade any function needed.

## Software Tools

The tools we have used are the same for the common tasks, instead for the individual ones (especially the development) each of us chose the tools that best fit their own workflow.

## Management and cooperation

Google Drive stores the documentation, we chose it because it has real-time changes, comments and chat.

Trello was used for task management and work assignment. It has been used a lot at the beginning, but then was a simple repository for remembering which task was assigned to whom: we decided and updated the tasks mainly during the meetings.



## Development

The editor tool is the one that differs more:



For the source control we have used Bitbucket



For the testing phase we have used Karma, a specific tool for unit testing, and Jasmine: a testing tool for the correctness of the integration between components and services. They are already integrated in the angular framework. This part has been one of the more difficult to learn, because no one had already used the Angular framework before and we didn't have experience in testing.



## Deployment

The deployment part is done with Amazon Web Service at this [URL](#)

The different architectural components are launched on different ports:

University API: 3000

Our API: 80

Web application: 4200



We have created a script for the deployment on AWS using the linux bash:

1. pull the latest commit on the master branch from the repository
2. launch our API
3. launch the web application

## User Interface Design

After defining the requirements we start with specifying the User Interface in details. Following subsections gives the detail information about the procedure and the refinement that we did when designing the User Interface for our project.

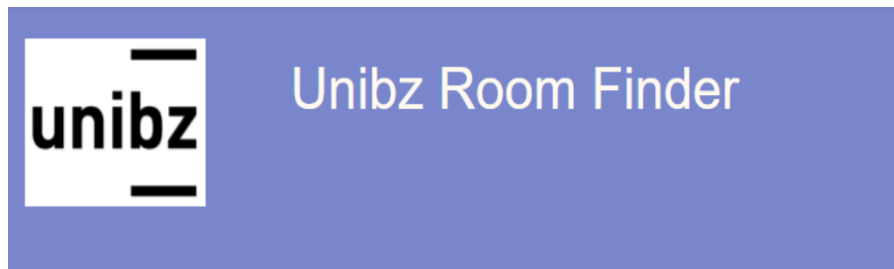
### Presentation Specification

Initially, we came up with a mockup UI using Paint as our preliminary User Interface which we refined later into our second prototype which was developed using HTML, CSS. We discussed the possible improvement from the prototype and agreed upon the changes, which lead to the final prototype development where we used angular along with HTML, CSS.

Following are illustrations of the prototypes that we have developed. The pictures show the gradual improvement and refinement from one prototype to the other.

DESIGN #1		
	<div>Date: <input type="text" value="(Today)"/></div> <div>Session: <input type="text"/></div> <div>From: <input type="text" value="HH:MM"/></div> <div>To: <input type="text" value="HH:MM"/></div> <div><input type="button" value="Find"/></div> <hr/> <div>Search by Specific Room</div> <div><input type="text" value="DAY"/></div> <div><input type="text" value="ROOM"/></div> <div><input type="button" value="GO"/></div>	<div>Goes to Design #2</div> <div>Goes to Design #3</div>

Prototype 1



Date

Session

From

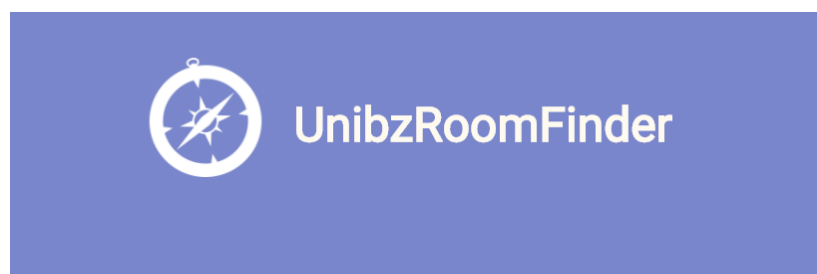
To

**Search by Specific Room**

Date

Room No.

Prototype 3



Location

Date

Session

From

To

Prototype 3

## Dialog specification

For specifying the dialogs and the components that we have used in our UI design, we used User Action Notation. The following table represents the UAN.

### Users Task: View All the freerooms

Action Number	Users Action	Interface Feedback	Interface State
1	<b>Click</b> on the <b>dropdown</b> menu to select the <b>location (Optional)</b>	<b>Show</b> all available <b>location options</b>	
2	<b>Select</b> the <b>location</b> from the available options (Not needed if action 1 is not performed )		Call action to allow user to select the building name for the selected location.
4	<b>Select</b> the <b>Building Name</b> from the options. (Not needed if action 1,2 and 3 are not performed)		
5	<b>Click</b> on the <b>datepicker</b> (optional)	Show the calendar	
6	<b>Select</b> the <b>date</b> from the calendar (not needed if action 5 is not performed)	Show the selected date in the date Field	
7	<b>Click</b> on the dropdown to select the <b>session</b> (Optional)	Show all the session options	
8	<b>Select</b> one of the <b>session</b> options (Optional)	Show the selected session in the session field and auto-populate the From and To time based on the selected session)	
9	<b>Select/ Type</b> the <b>“From time”</b> (optional,has a default value, not needed if action 8 is performed)	Show the typed or selected From time in the From field	
10	<b>Select / Type</b> <b>“To time ”</b> (optional,has a default value, not needed if action 8 is performed)	Show the typed or selected To Time in the To field	

10	<b>Click Submit</b> button (required)		Call a method to get the data from the database
----	---------------------------------------	--	---

User Goal: View the availability of one room

Action Number	User Action	Interface Feedback	Interface State
1	<b>Click</b> on the <b>location</b> drop down menu. (Optional)	Show all the location options that are available.	
2	<b>Select</b> a <b>location</b> (not needed if action 1 is not performed)	Change the option available for the Room dropdown box	
3	<b>Click</b> on the <b>datepicker</b> (optional, has a default value)	Show the calendar	
4	<b>Select</b> the <b>date</b> from the <b>calendar</b> (not needed if action 3 is not performed)	Show the selected date in the date Field	
5	<b>Click</b> on the <b>Room</b> drop down menu (required)	Show all the available Room.	
6	<b>Select</b> the <b>room</b> drop down menu(required)		
7	<b>Click</b> on the <b>submit</b> button (required)		Call method that takes all the input and finds all the time when the room is free

## Software Complexity

Now we can say yes our project is satisfied the high quality because the code with less complexity will contains less errors , easier and faster to test easier to understand and easier to maintain.

*“Design is not just what it looks like and feels like. Design is how it works.”*  
*Steve Jobs*

In this project we used Halstead metrics for better understanding how our project work. This paper will show you how we measured the software quality of our project in this way we will have the advantage to improve our project and optimize it in a very low cost with minimum time (IBM Knowledge Center - Halstead Metrics <https://goo.gl/wRfT3L> ).

### Why Halstead Complexity Metrics

- To obtain high quality software with low cost of testing
- The halstead metrics should be measured as early as possible in coding so the developer can adapt his code when recommended values are exceeded and the complexity metrics are used to locate complex code.
- It would be capable to measure the bugs, Finally...
- Above all it is a free software tools can be compiled with node.js

The complexity metrics that we used (Halstead metrics Node test, see at: [complexity-report](#) ) will show us the following:

Parameter	Meaning
n1	Number of unique operators
n2	Number of unique operands
N1	Number of total occurrence of operators
N2	Number of total occurrence of operands

When we select source file to view its complexity details in Metric Viewer, the following result is seen in Metric Report:

Metric	Meaning	Mathematical Representation
n	Vocabulary	$n_1 + n_2$
N	Size	$N_1 + N_2$
V	Volume	Length * Log2 Vocabulary
D	Difficulty	$(n_1/2) * (N_1/n_2)$
E	Efforts	Difficulty * Volume
B	Errors	Volume / 3000
T	Testing time	Time = Efforts / S, where S=18 seconds.

### Operators and Operands

Operators and operands are defined by their relationship to each other – in general an operator carries out an action and an operand participates in such an action. A simple example of an operator is something that carries out an operation using zero or more operands.

An operand is something that may participate in an interaction with zero or more operators.

### Halstead complexity - Tool installation

We already have node.js for our project. Then, for a project-based install:

```
npm install complexity-report
```

### Halstead complexity - Usage

```
cr [options] <path>
```

The tool will recursively read files from any directories that it encounters automatically.

### Halstead complexity - Command-line options:

```
-D, --maxhd <halstead difficulty>  specify the per-function Halstead difficulty threshold
-V, --maxhv <halstead volume>       specify the per-function Halstead volume threshold
-E, --maxhe <halstead effort>       specify the per-function Halstead effort threshold
```

### Halstead complexity - How it works

Once  $n_1$  and  $n_2$  are defined precisely we can simply compute 'Program (Code) Vocabulary' and program code length as we said before in this project we will use the Node test to



define  $n_1$  and  $n_2$  of or application as a results we will have:

$$\text{Vocabulary} = n_1 + n_2$$

While the Code Length is:

$$N = N_1 + N_2$$

Then we can measure estimated length of our project by using the following equation:

$$N = n_1 \log_2 n_2 + n_1 \log_2 n_2$$

After that we have to measure the minimum number of bits needed to code the program we use the following:

$$V = \text{Program Volume} + N \log_2 n$$

We have to measure the potential program volume for our project is:

$$V^* = (2 + n_2 * \log_2 (2+n_2))$$

The Program volume  $V$  and potential program volume  $V^*$  is used to explain the concept level of our project as following:

$$L = \text{Program level} = V^*/V$$

## Conclusion

Complexity is the quality of consisting of many interrelated parts. When software consists of many interrelated parts, it becomes more difficult to reason about.

Software that is difficult to reason about is a more fertile breeding ground for bugs than software that is simple.

## Recommendation

Halstead complexity that we used to measure our project is highly recommended by the MIT ([escomplex/complexity-report](https://escomplex.com/complexity-report)).