Browser based room booking system

Project:	Bachelor project
University:	ITU 2011
Advisor:	Peter Sestoft

Kristian Klarskov Marquardsen	kkma@itu.dk	220387-1611
Frederik A. Wordenskjold Noerregaard	fawn@itu.dk	140987-1727

Abstract

Vi har i dette projekt arbejdet med udviklingen af et interface, samt en delvis implementering af et lokaleplanlægningssystem til IT Universitetet i København.
Vi har inden udviklingen undersøgt de eksisterende systemer tilgængelige på universitetet, samt kravene for et sådan system, og konkluderet at der på nuværende tidspunkt enten ikke er systemer der er tilstrækkelige, eller slet ikke findes.
Vi har derfor baseret projektet på udvikling af et delvist implementeret system specifikt til ITU, der tilbyder reservation af lokaler, samt lokale planlægning. Den domæne specifikke viden vi har tilegnet os gennem projektet, har vi brugt i vores udvikling af programmet. Vi har lagt fokus på udvikling af brugergrænsefladen, da vi mener dette er en meget central del for at et sådan system kan benyttes med succes. En brugervenlig grænseflade kan gøre det nemt for brugere af universitetet at få et overblik over de pågældende rum, og dermed nemt reservere et lokale at arbejde i. Samtidig kan brugergrænsefladen hjælpe de ansvarlige med planlægning af lokaler for et semester, og yde dem støtte i denne process.
Vi har baseret projektet og udviklingen af brugergrænsefladen på en iterativ process, hvorfor vi meget tidligt i processen har udviklet forslag til design, som vi efterfølgende har testet, vha. tænke højt tests. Dette har ført til flere iterationer af de samme skærmbilleder, og det endelige design er også blevet testet for at sikre at målgruppen ikke ville have problemer med at bruge applikationen.
For at vise et eksempel på en faktisk fungerende applikation, har vi foruden interfacet, udviklet en webbaseret prototype af programmet. Vi har implementeret rum reservationsdelen i prototypen, så det er muligt at reservere et rum vha. 3 forskellige metoder, alt afhængig af behov.
Vi har ved efterfølgende tests kunne konkludere at vores brugergrænseflade har givet en øget støtte til lokaleplanlægningen, der har givet de ansvarlige et større overblik. Vores applikation har samtidig tilføjet et system, der gør det nemmere for brugere af universitetet at planlægge deres tid på universitetet, da de gennem programmet kan få information om hvornår et specifikt eller tilfældigt rum er ledigt, og dermed være sikre på at have et rum at arbejde i ved at reservere det.

Contents

Chapter 1

Background

1.1 Introduction

In this project we have designed an application which fullfils two tasks concerning the use of rooms at the IT University (*ITU*). We have developed a user friendly interface for managing the meeting and classrooms. This has been split into two distinct parts. The ad-hoc handling of meeting and classrooms (*day to day booking*), and an interface which helps the staff in assigning rooms to courses at the start of a semester (*semester planning*).

Currently, no application keeps track of the day to day bookings, and it is instead handled manually by pen and paper.
Our day to day booking interface will provide a larger degree of flexibility when finding a place to work, conduct meetings etc., as it provides the possibility to book rooms in advance and get an overview of the currently available rooms.

Additionally, the current semester planning is inefficient, decentralized and very time consuming. This process is also done manually, though with some support from an application (Tabulex [3]) normally used in elementary schools. We want to construct a system that centralizes the semester planning, and aids the persons responsible as much as possible, by making it easier for the staff to make a decision based on the information and suggestions provided by the application.

The program has been implemented as a web based solution, to make it globally accessible. We have chosen a web framework named Lift, written in Scala, which provides us with some needed functionality out of the box.

This allows us to put our main focus on creating a solid user interface. Due to this choice of focus, some parts of the system have not been implemented.

Appendix A.7 presents a glossary explaining the various terms and definitions used throughout the report.

1.2 Domain

This sections contains information about the actual domain in question; the IT University in Copenhagen, and the various resources and people relevant to our application.
ITU has 3 different room types, which is managed by two different departments.

- **Meeting rooms**
  Used for small to medium sized meetings, student group work and exam preparation.
- **Class rooms**
  Mainly used for lectures, exercises and examinations. Also used for events and meetings, if none of the meeting rooms are available or large enough. Class rooms also include the auditoriums.
- **Offices**
  Used by the professors, staff members and ph.d. students.

As of writing, room management is split between the Study Administration (SA) and the Facility Management (FM). Within the study administration, there is another division of labour which all in all adds to the confusion of handling the limited rooms efficiently.

1.2.1 Facilities Management

FM manages daily adhoc planning, event planning and a subset of the rooms; the office rooms and the current day to day booking: When a person wishes to reserve or use a meeting room, the FM employee will refer to a binder with a paper-overview of the current status of each single room.

This approach presents several problems:

- It is not possible for the students to book a room in advance. Employees can book rooms in advance for conferences, faculty meethings etc., but the students can only book a room on the current day.
- It is not possible to get an overview of the rooms currently available. This is a problem when you have specific requirements like the need for specific equipment, or when you need a room with a specific capacity. FM can answer some of these inqueries, but they rely on experience as they do not have this information available anywhere.

Additionally, only FM can tell you if and which rooms are currently unbooked, which at times can be a time consuming process during busy periods.

1.2.2 Study Administration

The SA manages the class rooms and the auditoriums, thus also in charge of allocating rooms for lectures and examinations.
In regards to room management, the Study Administrations biggest challenge is the planning of a semester. First step is to collect data from the course database to see which courses will be held in the coming semester. When a list of courses has been compiled, a system called Tabulex is used to create a draft schedule to avoid overlaps of teachers and courses. On the computers at ITU, it takes Tabulex up to 8 hours to generate this data. The output from Tabulex is finally used to manually create the room assignments for a standard week of the semester.
Note that a semester contains several different classes and study programmes. The SA uses a term called tracks to divide the various study programmes into sections. A track is one specific study programme at one specific semester, e.g. *Software Development 3. Semester*. Each track has a weekly schedule, which is created manually in the final process of the semester planning.

1.2.3 Current systems

Excel sheets and Word documents

The Study Administration uses an Excel sheet to get an overview of the availability of the different rooms managed by them. Each room has been assigned to a tab containing a weekly overview. This works well for getting a general overview of what the different rooms are used for, but it makes it very difficult to e.g. find an available room, as one would have to browse through all the tabs one by one, until a suitable room is found. If one has requirements for specific equipment, it is nessesary to refer to a seperate paper (a Word document), listing the equipment available for each room.

Tabulex

Tabulex is the program used by the part of the Study Administration that allocates rooms for courses. In Tabulex, a user defines the relationship between classes, rooms, teachers and courses. Binding 1 of each together, you have an entity representing a class attending a specific course in a specific room held by a specific teacher. When all entities have been defined, Tabulex arranges

them in a weekly schema, in such a way that no resources of different entities overlap, e.g. making sure no teachers teach two course simoustanely and no rooms are used for two courses at the same time.

Tabulex is good at allocating general schemas, but it lack a lot of functionality relevant for ITU. When two courses overlap each other, which might happen if the distribution of resources makes it impossible for the algorithm to make a schema, Tabulex does not suggest any alternative solutions, or at least provide some information about the problematic entities in question. This makes it difficult, and very time consuming to come up with a proper alternative solution, as the user has to make a manual correction, which often results in another conflict.
Tabulex also does not keep track of equipment in each room. This is possible to solve through Tabulex though, as you can assign custom rules for the program to respect. As this is just simple rules, it might be hard to understand why these rules were created in the first place, if you were not the person responsible. This makes it hard for more than one person to use the application.
Finally, Tabulex does not keep track of the capacity of each rooms, which makes it hard to do a proper allocation, as you cannot put smaller classes in small rooms, and vice versa.

1.3 Scope

Since we want to focus on the creation of a solid interface that solves the above problems, the implementation has been a secondary priority. We have divided the project into two main tasks, which together constitute to the main body of such an application. The success of the project depends on how well the interfaces for these two tasks have been designed.
In the following, we will describe the two main tasks and how they relate to each other. In general, we will work with all types of rooms except the offices, as they are handled differently. It would not make any sense to book an office, as they are assigned specifically to staff and students.

1.3.1 Day to day booking

Our program should provide an interface to support booking of rooms (excluding offices) at the IT University. Users of this function might have very different requirements. Students might just need a room to work, while an employee might book a specific room in advance for a meeting. Others might

need specific equipment to be available. Because of the many requirements, the program should not only provide the user with options, but also suggest solutions.
Since most people using our application will be students looking for a room without any specific requirements, the simple task of finding any available room should be easy accessible and quick to use.

The day to day booking should thus contain:

- A general overview of the available auditoriums, meeting- and class rooms to book.
- An interface that quickly books a room, using a minimum of options. (**Quick booking**)
- An interface for booking a room, selected by clicking on that specific room. (**Simple booking**)
- An interface that presents the user with several options, such as the need for specific equipment, room capacity, time of day, etc. (**Advanced booking**)

The application should:

- Suggest *a single room* to the user, when using the quickbook function. The room suggested, should be the *optimal* room, in terms of efficient room allocation. This means that the application should suggest the smallest room.
- Suggest a list of rooms that fulfil the maximum number of requirements, given as input by the user through the selective booking function. If all requirements can be fulfilled, only the optimal rooms (see above) should be presented to the user.

To exemplify a working application, we will implement some parts of the day to day booking interface.

1.3.2 Semester planning

In addition to the day to day booking, we will design the user interface for a semester planning system. Planning a semester is in the case of ITU not just a matter of assignment. Each semester have more courses and day/time combinations than there is rooms, so the essence of it,is providing the tools

to prioritize in an informed manner. At the point when a semester needs to be planned, most courses only have a start-week and an end-week, along with the number of enrolled students. Some courses can, eg. due to external teachers with busy schedules, have narrowed down their options, and we must of course including support for these scenarios. Worth noting, and part of our considerations in the design, is that there will only be a few people using this part of the system, and these people will be easily accessible for training.

The semester planning should contain:

- An interface to manage courses.
- An interface to manage tracks.
- An interface to handle any problems with planning that cannot be resolved automatically.
- An interface to view/review the current semester's allocation.

The application should:

- Apply a constraint so that no teachers have to be attending two courses at the same or any overlapping time.
- Apply a constraint so that no courses are held in the same room at the same, or any overlapping time.
- Attempt to not put courses in meeting rooms, unless all class rooms are unavailable in all possible scenarios.
- Provide alternatives to assignments and allow for wishes/preferences.
- Provide suggestions for resolutions if no full semester plan is possible.

We will design the user interface for the semester planning, but not implement or add the required functionality. Doing the full scale implementation is not something we are able to do because of time constraints, while also not being crucial to prove our point of data presentation and usability.

1.3.3 Relations

Even though the two tasks present very different functionality, the relationship between them needs to be defined.

The semester planning should have precedence at all times. This means that any day to day bookings interfering with the semester planning will be deleted. Because of this, it should also not be possible to book a room during the use of the semester planning interface.

In practice there might be exceptions, that we cannot predict. ITU might have a very important meeting that should not be deleted, or overwritten by the semester planning algorithm. These exceptions will be taken into consideration when implementing the application.

A day to day booking cannot be made, if it overlaps with a scheduled event (like a lecture) created through the event planning interface.

Not all users should have access to the semester planning. Only users defined as *administrators* have access to this part of the system.

1.3.4 Functionality out of scope

Some functionality are not relevant for the two main tasks in our scope. The following list presents components that will not be implemented, and will have little impact on the user interface.

- An administration panel.
  Each administrator of the system, should have access to *all* bookings, and should be able to change/delete bookings.
- A communication protocol for sending and receiving messages.
  This is nessesary if e.g. a booking is deleted by an administrator through the semester planning interface.
- Some kind of restrictions to the day to day booking.
  It should e.g. not be possible for a user to book all available rooms at the same time.

Chapter 2

Analysis

2.1 General vs. domain specific solution

We have chosen to implement the application solely for ITU, to include support for domain specific constraints and situations. The semester planning part of our application is not a trivial problem to solve. It fact, it is a so called NP-problem, which is a problem that is unsolvable within a reasonable period of time. This is due to the many types of constraints, which can vary a lot for each domain. While we concentrate on the UI and not the algorithms that should actually solve the problem, we still need to provide the data nessesary for the algorithm to function.
By making it a domain specific problem, we can limit the number of constraints to reduce the complexity. This makes it possible for us to include only the relevant user interface elements, and exclude the elements that *might* have been relevant for other domains. This reduces clutter of the interface, and makes it possible for us to include support for the needed domain specific elements.

2.2 Existing solutions

Existing solutions can be divided into two categories:

- **Room management software**
  This kind of software gives a good visualization of available rooms on a current day. All programs have taken a general approach by listing each room on the x-axis, time on the y-axis and boxes representing a reservation with information inside it. Figure 2.1 shows an example of such an application.

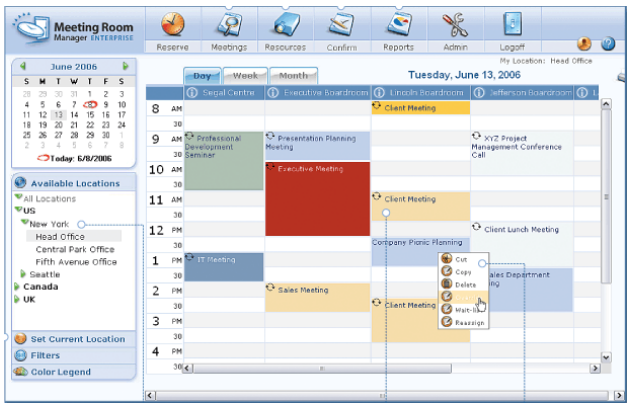

Figure 2.1: Meeting Room Manager by NetSimplicity [9]

While having nothing to do with semester planning, this representation works as a room booking system if you only have a few rooms. On a 800x600 resolution monitor, 5 rooms is enough to make horizontal scroll nessesary, making it hard to get an good overview of the current occupation of the rooms.

- **Schema based planning software**
  These applications are developed for primary schools and universities. They show weekdays on the the x-axis, time on the y-axis and a box in the schema represents a booked time slot. This gives a good weekly overview, but you cannot get an overview of how a specific room is used. More importantly, ITU needs another abstraction level, the tracks, representing one semester for a particular study programme, which is not possible to represent in any of the applications. This would make it hard to gather information from a particular track, as the related

courses would be scattered throughout the overview. Figure 2.2 shows an example of a schema planning application.

Figure 2.2: EMS Campus by DEA [4]

In general, all of the applications take a general approach. This makes the user interface less user friendly, as important application specific variables cannot be taken into account, like a map of the specific building or domain specific naming conventions.

2.3 Target audience

Our target audience can be divided into two groups:

- Student body at ITU
- Employees at ITU

Student body

The student body covers ages 18 - 40+ and with very different backgrounds. Based on the nature of the university, it is fair to assume that most students have at least basic familiarity with IT and computer use in general.

Employees

The computer proficiency of the employees can range from expert to novice, and no clear-cut distinctions can be made. We have, however, observed that the majority of the people our system would affect are using tools with some similarities, and are therefore assuming their proficiency to be above novice users.

Note that even though our day to day booking interface will mostly be used by students, we can make no assumptions that the target audience for this part of the application *only* contains students. We need to make sure the day to day booking can be used by a very wide range of people, including also the employees.
No one outside of ITU (not being registrered in the userbase) should be able to use the application, as it should not be possible for random companies to book and occupy rooms without permission.

2.4 Requirements

Since the ITU does not currently have a full system, we have put up initial requirements for such a system. As described in the target audience 2.3, the system will be used by a wide variety of people, and there will be virtually no way to train them. We have met this problem by writing up all the tasks our system must support, along with quality requirements describing success criterias for our application. [8]

2.4.1 Non-functional requirements

Preparing for the creation of mockups following various usability tests, it is nessesary to set up a reference point to what we actually want to measure. This helps us to keep the following mockups and usability tests focused.
Usability quickly becomes very subjective, if you do not attempt to define it in more precise terms. We have used Soren Lausens five usability requirements [8], as a measurement of the usability of our application.

In a perfect world, one might think that all of the 5 usability requirements would be met. However, this is rarely desired, as it does not emulate a clear focus of what should be prioritised in the application. Here follows a rundown of the usability factors and how they relate to our project. Note that the **fit for use** requirement has been left out. This is defined as:

Can the system support the tasks that the user has in real life?

We do not think this is an actual usability requirement, as it is merely concerned with the functionality of the program.

- **Task efficiency**
  Since the task of finding a room often will be done on the fly, on the way to school, or right before a meeting takes place, it is important

that the application is responsive and gives the user a proper response to a booking query. It should give the user a good overview to support his decision, and suggest alternatives if his query cannot be met. In general, we have failed if the user finds it more effective to book a room the old fashioned way, by asking the Facility Management.

- **Understandability**
  Interviewing the Study Administration suggested that the staff responsible for doing the semester planning had a large need to double-check that the important tasks performed by the system were done correctly. A.4. In general, they had very little confidence in the system. Therefore, to avoid resistance towards change, it is important that the system clearly communicates it's functionality to the user, during normal use, and expecially in error prone situations to avoid confusion.

- **Ease of learning**
  The set of people using our system is large and will change rapidly. It is therefore important that the system can be used "out of the box". If not, people will simply not use it, as it takes to much of their already limited time to get familiar with. Also, there is a close relation between ease of learning, and the effectiveness of the support the system offers. If the user does not understand how to use the system, they will not be able to benefit from the support layer.

- **Subjective satisfaction**
  Some might argue that the subjective satisfaction is not very relevant in our case, because we're developing a tool for daily support and not a program that in any way should be amusing or entertaining.
  However experts believe that users are more willing to accept the program and trust its efficiency if they find it visually appealing [10]. Apple is a good example of using design and pleasing aesthetics as major selling points.

2.4.2 Functional requirements

To establish a foundation for the requirements of our system, we have considered the functionality our system must contain. These functional requirements only contains high level goals, and not *how* they must be met, as we will adjust the way they take place based on an evaluation of usability factors, tests and the domain analysis.

The system must...

1. distinguish users from each other
2. store data and information persistently
3. support the user in performing the tasks specified in section 2.5

In regard to the first item, all students and faculty at the ITU already have unique logins, and it would be natural to reuse those for our system. Also, it should not be possible for non-authorized users to use our system. Being a webbased system, we want the data to be stored persistantly and centrally, so different computers at different times can be used to retrieve the same data. Naturally, the system has no use without actually fulfilling the tasks it has been designed to, hence the listing as a functional requirement.

Note that we have few functional requirements compared to some systems. This stems from the fact that the system, from an end-user perspective, is pretty simple. Our vision is simply to enforce this perspective, and provide the *expected* functionality, without hindrance of high complexity and feature creep. [14]

2.5 Tasks

The following tasks are the actions one should be able to be perform within our system. Note that track and teacher have been combined to one task, as they are both just used as constraints, and will be edited in the same way.

1. Day-to-day booking

T1.1 Book room. May or may not be a specific room or a specific time/day

T1.2 Check availability of room. May or may not be today.

T1.3 Cancel booking. May or may not be before the booked timeslot.

T1.4 Extend booking. May or may not be before the booked timeslot.

2. Semester planning

T2.1 Create course. May or may not have full information available at the point of creation.

T2.2 Create track/teacher.

T2.3 Edit course.

T2.4 Edit track/teacher.

T2.5 Plan semester. A full solution may or may not be possible

A full example task including subtasks can be found below, while the full list including subtasks covering all scenarios can be found in appendix A.2.

T1.1:	Book room
Start:	User opens system
End:	User have gotten a room
Frequency:	Average of 10 bookings pr. day
Difficult when:	No rooms are available

Subtasks	**Example solution**
1. Find room	
1a. No specific room needed	System suggests any available room
1b. User has specific needs	System allows to search by parameters
1c. No available rooms	System allows to search at future dates
2. Book room	
2a. User does not specify booking duration	System suggests a standard time duration
2b. User specifies time interval	
3. Confirm booking	
3a. User confirms	
3b. User declines	System cancels the booking

2.5.1 Quality Requirements

Based on the task list above is the quality requirements we wish to meet in able to consider the level of usability acceptable.

1. Without instructions, 90% of novice users must be able to complete T1.1 to T1.4 without variants.
2. Without instructions, 80% of novice users must be able to complete T1.1 to T1.4 including variants.
3. With a maximum of 30 minutes instructions, 90% of administration personel must be able to complete T2.1 to T2.5.
4. Without instructions, 50% administration personel must be able to complete T2.1 to T2.5.

2.6 Domain specific challenges

Throughout the project, we encountered some very domain specific challenges, that ultimately should be answered by ITU. They are nevertheless important to our application and will be highlighted in this section. We have discussed these issues with a member of the study administration, and our default choices are based on that, but it is something that ITU as an institution must decide on policies for.

2.6.1 Approval

It is very plausible that FM wishes to assume full control of the rooms. In practice, this would mean that users could make requests for rooms, and FM would then manually have to approve or decline every request. We find it a very reasonable wish, but with a caveat: If FM is not constantly active, it would not be a big improvement over the current way of handling rooms, but simply a new look to an old way. In our prototype, we have assumed that FM would be willing to accept the users making bookings themselves, and simply grant them the power to delete bookings at will.

2.6.2 Limiting bookings

To prevent abuse, such as a student booking the whole school for a month straight, it is important to have some limitations based on user levels. For example, limiting a student to have one or two active bookings, and possibly more for staff and faculty. This would of course already be solved if the aforementioned approval system was agreed upon.

2.6.3 Overriding bookings

We mention in section 1.3.3 that semester planning should take precedency over day to day booking, and that it might not always be the case in reality. This is a policy that ITU internally have to sort out, and we will have to adjust the application according to their final decision. By default, we will however allow day to day bookings to be overwritten, as it is our belief that a good room allocation is more important than a single meeting/appointment.

Another concern regarding prioritizing bookings, are the cases of extending bookings. If a users wishes to extend a booking for a room they currently occupy, and another booking for the same room has been made already, it could be considered if the current occupant should be allowed to extend if

a replacement room can be found automatically for the secondary booking. By default however, extending is only possible if the room is not booked.

2.6.4 Occupation of rooms without booking

If a user books a room that according to our system is available, but in reality is occupied, ownership of the room can become a point of argument. One would think that the day to day booking functionality should be a convenient tool to use if you find it useful, but not required.
This might not be possible in reality though, because of situations like the before mentioned. We therefore assume that rooms should be booked before you have the right to use them. This is a very important point, as this will force some people to use the program, ever though they do not want to. The program should thus be very easy and quick to use, so the users dont feel they waste time booking a room, when they could have just avoided it.

Chapter 3

User interface design

3.1 Method

We have chosen methods and processes that enable us to achieve results in a consistent and structured manner. In this section, we will describe the methods we have applied, and explain why they were chosen over similar alternatives.

3.1.1 Iterative design

When designing a user interface with the intention of reaching a high level of usability, a systematic approach is key. The process we have used, can be roughly divided into 4 steps, where the last two step should be repeated until the Quality Requirements, listed in 2.5.1, have been met. [8]

The steps are:

1. Analysis of users and tasks
2. Construct prototype
3. Perform usability tests
4. Amend prototype

The semester planning and day to day booking have been described seperately, due to the differences between them. We will provide a detailed walkthrough of the process for an element in each part, and the full history of mockups can be found in appendix A.6.

3.1.2 Usability testing

With usability being our primary focus, choosing a proper method of testing is key. A number of different variations of user-based testing exist: [8]

- **Observe only** - Let the user work out the tasks on his own, and simply observe his progression
- **Think aloud** - Let the users explain their thoughts and decision as they (try to) complete the tasks
- **Cooperation** - Have two users solve the tasks together, and encourage them to discuss what is going on

Among the most widely used technique for testing user interfaces in development stages are the think aloud method. [7] As one can imagine, not all test subjects find it natural to explain their thoughts as they go, but with interesting questions from the test evaluator, most users get familiar with the concept very fast. We have chosen the think aloud test for two primary reasons. First, because even in the worst cases where a user does not adapt to explaining why and what they do, you can still get the same, if not better, results than with the *observe only* test. Secondly, the *cooperation* test has the potential danger of concealing usability problems. In the case where test subjects *t1* and *t2* are cooperating on a task, a problem t1 would have encountered alone, will not be discovered due to t2 solving it beforehand, and vice versa. This results in potentially only finding problems which both users would have encountered on their own.

Surprisingly, we have not found any description of this problem by the established experts, and can therefore not verify our theory, but have nonetheless decided to avoid this version of usability testing, based on this.

Choosing which, and how many, test subjects is needed for a test is a heavily debated subject in the usability community. Most tend to agree that 5 or below is more than enough for each round. The controversy arise when figuring out how representative the test subjects most be for your target audience. Soren Lauesen opens the corresponding chapter with: [8]

> *"A usability test must of course be made with users that correspond to the real users we expect in practice."*

However, Steve Krug has a different view: [7]

> *"The importance of recruiting representative users is overrated. [...]"*

"The best-kept secret of usability testing is the extent to which it doesn't much matter who you test."

What we can take away from their different arguments for and against, is that testing with representative users is *good*, but not vital when it simply comes to eliminating problems. We have chosen to do both. The system is intended to be used by everyone at ITU, which will also include newly started students and faculty. However, for the semester planning we value the knowledge held by representative users to be essential, and have therefore this limited test subjects to that. Another point of agreement among the experts, is that testing early and often will *always* trump testing with many users at a later point. [7, 8, 11]

The table in figure 3.1 shows how many users have been tested for each part of the system for each round. As seen, we've opted to only test one user for the semester planning, which is simply due to the lack of people with the required knowledge, and the very narrow target audience.

For the actual tests, we have used the tasks listed in section 2.5 to construct the questions, and ensure that we reach the critical parts of the system. The questions and scenarios presented to test subjects have to fullest extent possible been formulated so they provide no implicit help, and use as few domain specific words as possible. [8]

	Round 1	Round 2	Round 3	Total
Day to day	2	2	2	6
Semester	1	1	n/a	2

Figure 3.1: Overview of test users per round

	Round 1		Round 2		Round 3	
	User 1	User 2	User 3	User 4	User 5	User 6
	ITU student	Tech savvy outsider	ITU student	Average outsider	Average outsider	Novice outsider
Quick booking	*Not in test*	*Not in test*	*Not in test*	*Not in test*	Pass	Pass
Simple booking	Task failure	Task failure	Minor problem	Pass	Pass	Medium problem
Advanced booking	Pass	Medium problem	Pass	Pass	Pass	Pass
Booking on different day	Minor problem	Pass	Pass	Annoying	Pass	Pass
Check availability	Pass	Pass	Medium problem	Task failure	Pass	Minor problem
Cancel booking	*Not in test*	*Not in test*	Annoying	Pass	Pass	Pass

Figure 3.2: Test results for the day to day booking

Figure 3.2 provides a full overview of our findings from the usability testing of the day to day system, which we will use as a reference in the day to day booking chapters. We have used the problem classification defined by

Soren Lauesen [8].
What stands out here is that user 6 encountered two problems, one of which resulted in him not completing the assigned task in a satisfactory manner. Naturally, this is not something we could wish for in the last round of testing, but it did provide insight as to how users with very little technical knowledge interpret our system.

The full test documentation can be found in appendix A.3.

3.1.3 Gestalt laws

In usability theory, *gestalt laws* (or gestalt principles) are often used to manipulate the users perception of elements. Listed here is a brief overview of the laws we have utilized in the design. For extensive walkthrough of each law, refer to Soren Lauesen [8].

- **Law of proximity**
  Pieces that are close togehter are perceived as belonging together.
- **Law of closure**
  The area inside a closed line is perceived as a shape.
- **Law of good continuation**
  Pieces on a smooth line are perceived as belonging together.
- **Law of similarity**
  Things that look alike are perceived as belonging together.
- **Law of parallel movement**
  Things that move in parallel are perceived as belonging together.

It is vital to be aware of these principles when designing a user interface, as unintended gestalts may be confusing for users, and in severe cases cause complete task failures. [8]

3.2 Day to day booking

As described in previous chapters, the day to day booking should be usable without any instructions. We have tried to imitate familiar designs, and use common design conventions. As Steve Krug puts it [7]:

> *"All conventions start life as somebody's bright idea. If the idea works will enough, other sites imitate it and eventually enough people have seen it in enough places that it needs no explanation."*

The function that went through most interations was the *simple booking* function. This function will thus be the focus of each iterations in the day to day booking chapters. We will go through the process of creating and testing each mockup related to this function.

3.2.1 Wireframe

The front page serves as a portal to the rest of the system, and thus it should be easy to understand. To ensure that we had the vital navigation functions in place before beginning the mockups, we constructed a wireframe [5]. The wireframe, as seen on figure 3.3, has the following elements:

Figure 3.3: Wireframe for the frontpage

- **1. Logo**
  The logo for the application, which by convention should be located in the top left corner and link to the frontpage. [7]
- **2. Toolbar**
  A navigation toolbar. Should be used to login and logout, and for accessing the personal user account overview.

- **3. Calendar**
  An object used to navigate to a specific day in the future (or past).
- **4. Overview**
  The main overview, which should present a map of the university.
- **5. Function panel**
  A panel containing buttons used to enter the booking interface, or the semester planning.

We will refer to the different wireframe sections as (wire:section). For example, the advanced booking button is referred to as (wire:5.2).

3.2.2 First round

Mockup

The first thing we wanted to test was the booking dialogue, as this was a major part of the user interaction. When the user selected a room on the map, the dialogue seen in figure 3.4 should appear in the function panel (wire:5). We wanted to give the user the impression that they never left the homepage, and to make sure no-one got lost in subscreens and menus.

Test

In our first usability test, the users were to complete the following tasks:

1. Find a random room
2. Find a random room tomorrow
3. Find a specific room
4. Various variations of tasks involving finding rooms with specific requirements

What may come as a surprise, is that many users at some point or the other, tried to complete the simple tasks by using the advanced search button (wire:5.2) instead of e.g. using the map to select a room (wire:4).
Additionally, our *another day* button was often ignored.

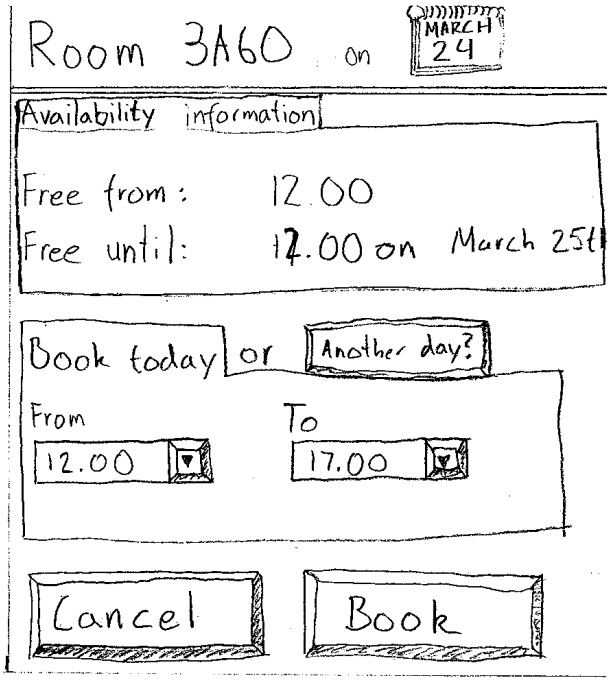

Figure 3.4: First draft of the booking panel

3.2.3 Second round

Mockup

Second time around, we tried making it as basic as possible to reduce the clutter, to make the users realize that this screen was for very simple tasks. We simply presented the user with a dialogue for selecting how many hours from now the room should be booked.

Test

These changes made all the test users pass the simple booking task, and it made them realize that the advanced search button(wire:5.2) should be used for tasks including specific room requirements. It did present another problem though: Now, no test users were able to figure out the availability of a room in the near future, as there was no button indicating this functionality when you had selected a room. The calendar (wire:3) was actually intended for this purpose, but this was not used by any of the users.

This round of testing was probably the one with the biggest impact on

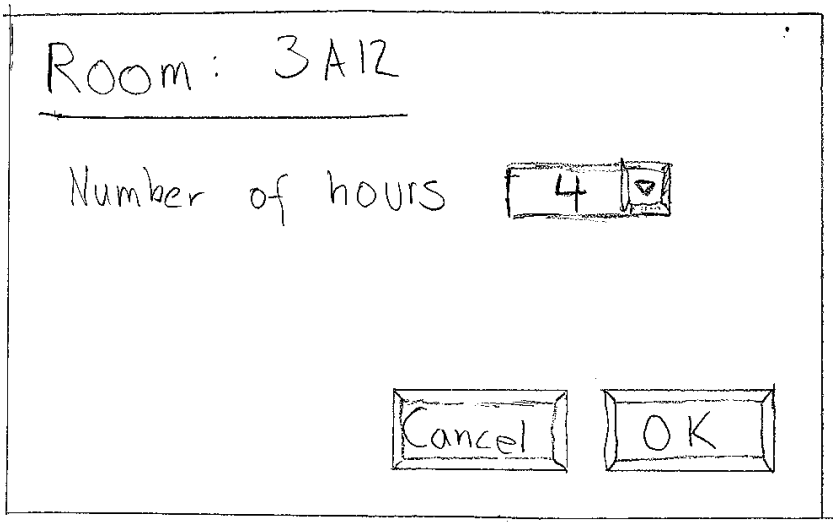

Figure 3.5: Second attempt at the simple booking. This time as a window overlaying the overview (wire:4)

the final design. As shown on the test results from figure 3.2, *quick booking* was not thought of yet in the first round. Even though it created another problem of not providing the needed functional for getting a weekly overview, we discovered general satisfaction with the simplicity of the overlaying panel seen in figure 3.5. We wanted to keep this simplicity, while still making sure users could find a week overview of each room, which led to a very different third iteration.

3.2.4 Third round

Mockup

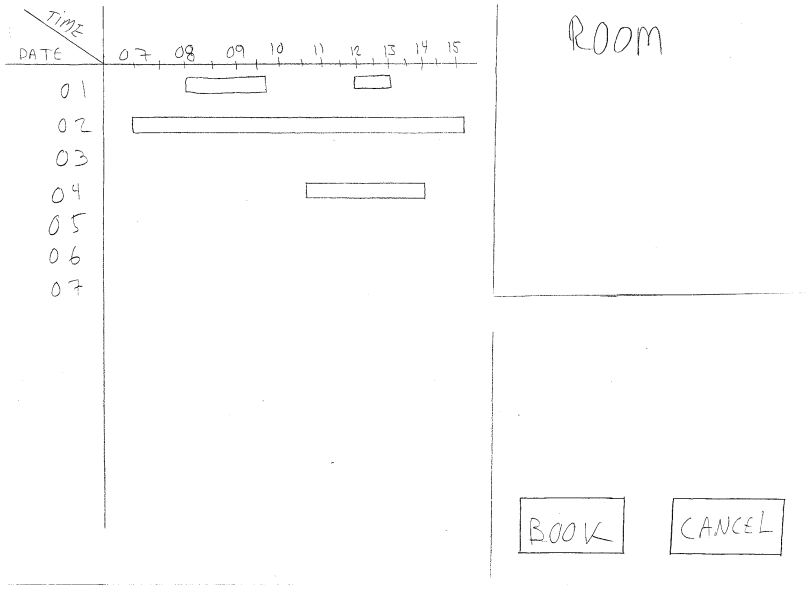

Figure 3.6: The mockup screen for a week overview on a specific room.

In previous mockups, both users clicked on the specific room in an attempt to find the weekly overview. In this iteration, we changed the mockup to do exactly this. When the users clicked on a room on the map (wire:4), the screen would change to the view in figure 3.6.

As shown, we have decided to invert the x/y-axes as opposed to normal day-time overviews in eg. Outlook or Google Calendar. This is done because of the standard timeline being from left to right, and not top to bottom. We have simply added 7 timelines on top of each, as we thought it more intuitive when trying to get an overview. Our assumptions proved correct from every single test, as not a single user had a problem understanding the view immediately.

Test

All the test subjects used this overview effectively, when trying to book a room for a day not being the current. While the users passed the simple

room booking tests in the second iteration, we did not test this again, but changed the function slightly due to new information gained through previous tests. The 'find me a room' button (wire:5.1) did not present the users with an option to choose the duration of the booking anymore, but simply booked the room for 4 hours. The user would then have the opportunity to extend the booking if necessary. Also, the room was now booked immediatly as the user clicked the button, to prevent problems with concurrent bookings, as discussed later in section 4.6. Finally, we changed the label from 'Find me a room', to 'Quick Booking', to better reflect the actual functionality of the button.
Figure 3.9 in the proposed design section shows the quick booking dialogue as presented to the user.

3.2.5 Final round

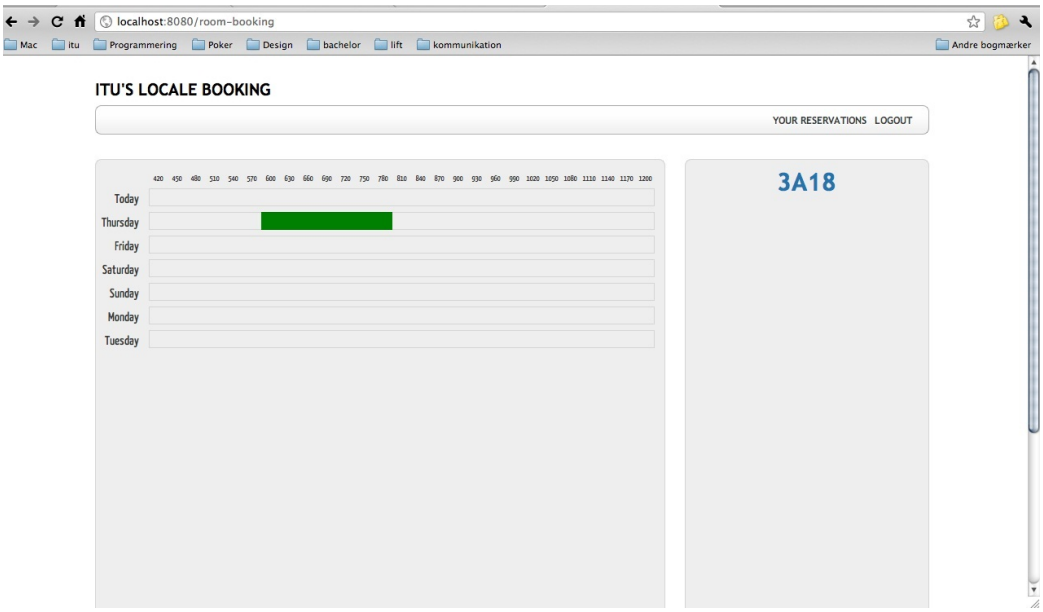

Figure 3.7: The finished screen for a week overview on a specific room, also used by Simple booking.

The final design to book a room, as shown on figure 3.7, grants an availability overview of 7 days, starting with the current day. Through our tests, we noted test subjects trying to drag the mouse to select, and others clicking for a start time followed by a click on end time, so we have chosen to allow for both. To further assist the user in figuring out how it works, we have added highlighting of both the day and time when mouse is moved across the matrix.

The test results in figure 3.2 show a medium problem for simple booking in third round, which was caused by him not discovering that the map was clickable, but he managed to succed by muddling through.

We have not conducted extensive tests for the final round, but simply performed sanity checks with previous test subjects, due to the reasonably small amount of changes between the rounds.

3.2.6 Proposed design

The process described above have naturally been repeated for each part of the system, resulting in our proposed design for the final interface. Following is a description of all the screens that have been implemented, and their respective functions.

Front page

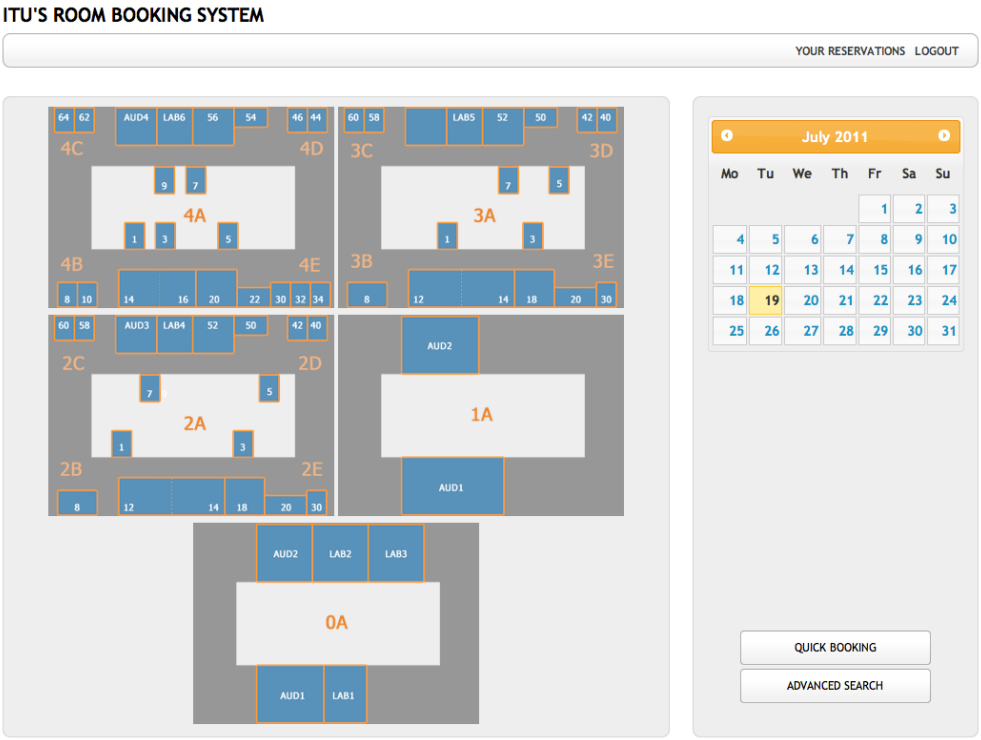

Figure 3.8: The finished frontpage design

The frontpage shown on figure 3.8 adheres a lot to the wireframe described above. We have made a couple of important decisions with the design. First off, we have decided to resize the rooms on the overview so the scale of the rooms compared to each other, is not the same scale as in reality. We have of course kept the location, and the approximate scale to adjecent rooms, but all the very small rooms have been enlarged to allow for easier selection. Secondly, all offices, bathrooms, closets and kitchens have been left out. This is done to reduce clutter, and emphasize the rooms we are concerned with.

Quick Booking

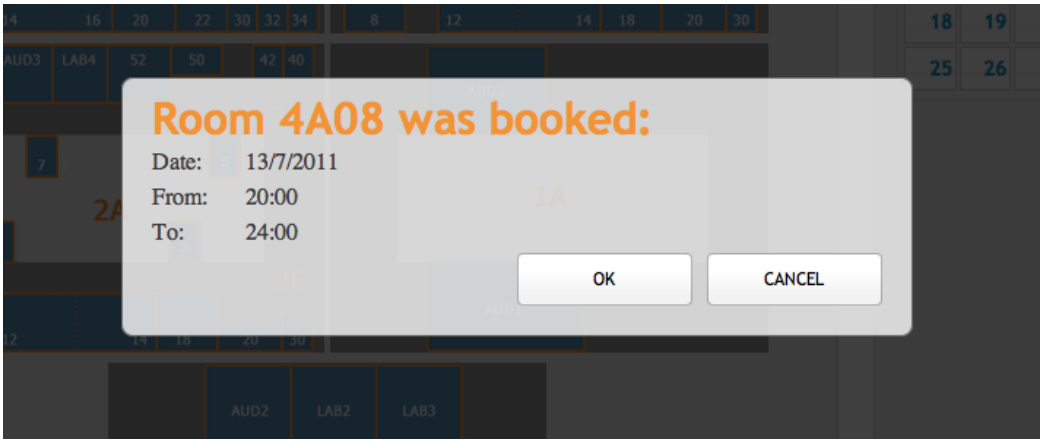

Figure 3.9: The overlay when quick booking is pressed

Shown on figure 3.9 is the overlay appearing when the 'Quick Booking'-button is pressed. It simply finds the smallest available room and books it for a default 4-hour span. The technicalities behind this function is described further in section 4.7.3.

Simple Booking

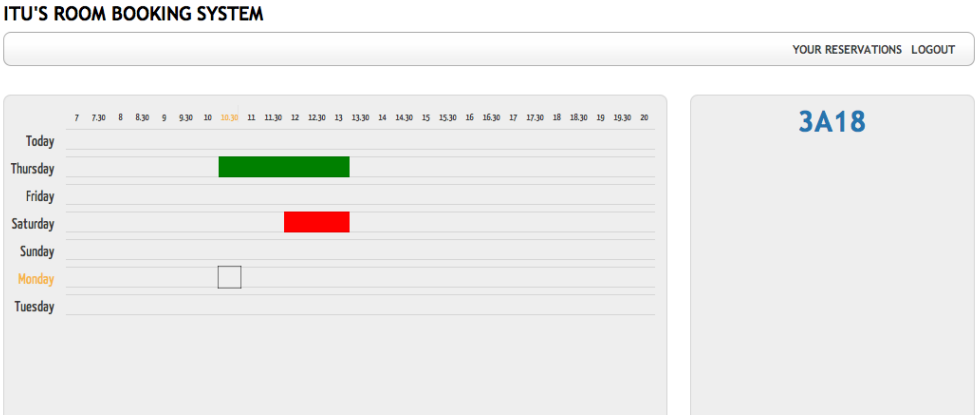

Figure 3.10: The overview and simple booking screen used when a specific room is clicked on the map

When a specific room on the map overview is clicked, the screen displayed on figure 3.10 will be presented. The specifics of this part of the system have been discussed in the example of the design process in section 3.2.4. It is worth noting that it only shows a 7-day overview, starting with today. We have considered to allow for scrolling back and forth in weeks, to allow the curious users to check bookings further in the future than a week. We decided to not implement it in the prototype, as it would be an unnecesarry distraction for almost all situations this view would be used for, and the calendar already provides this functionality. Our tests revealed that users tend to utilize 'Advanced Search' for everything that was not immediately easy, and we concluded that if a booking needed to be made far in the future, that would be the function they used.

Advanced Booking

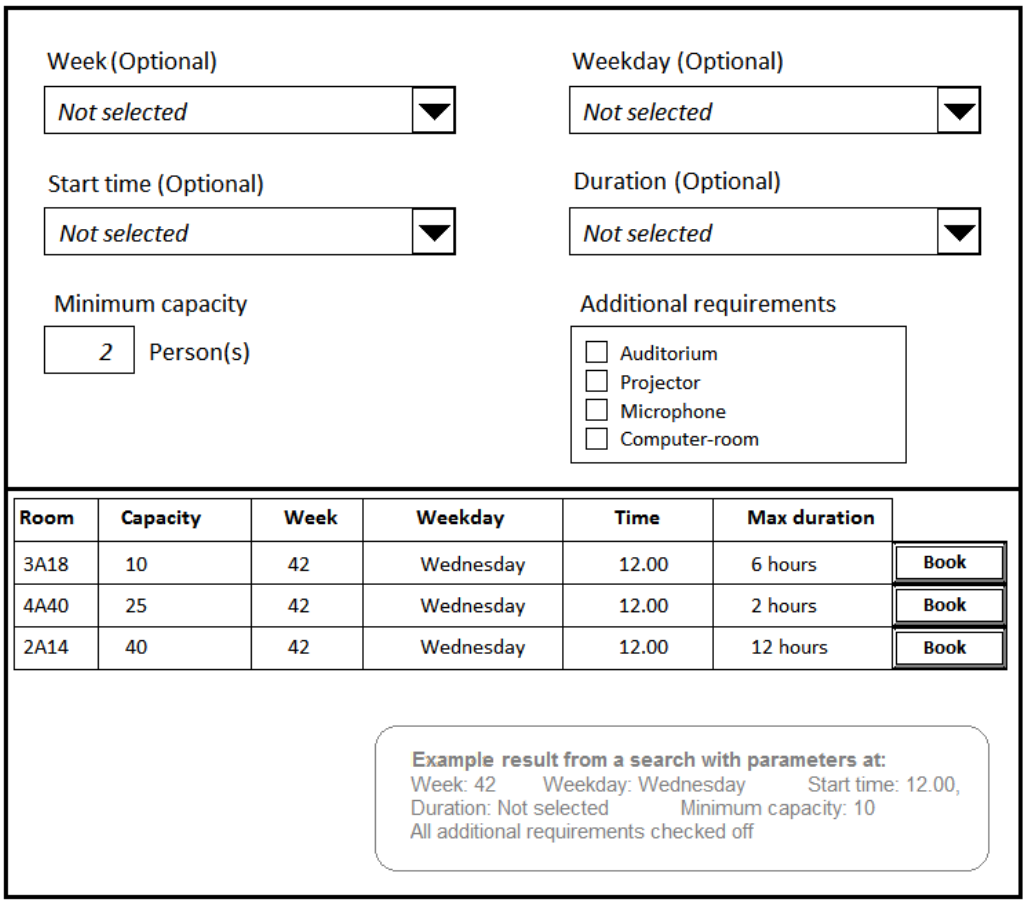

Figure 3.11: The design of the not-implemented advanced booking.

The mockup on figure 3.11 shows the concept for our advanced booking screen. We have not made a search button, but instead allowed the result section to update dynamically for each parameter entered. Note that the search result pane is filled with results from an example search, and does not correspond to the displayed default values in the form above.

The narrow design allows it to only overlay the map, and thus leave the calendar still visible. If a date is clicked on the calendar, the week number and day is filled into the search form. This functionality is not advertised by text, as it is not necesarry to succesfully use the system, and the text should be either very long, or run the risk of being confusing rather than helpful. [7]

Your Reservations

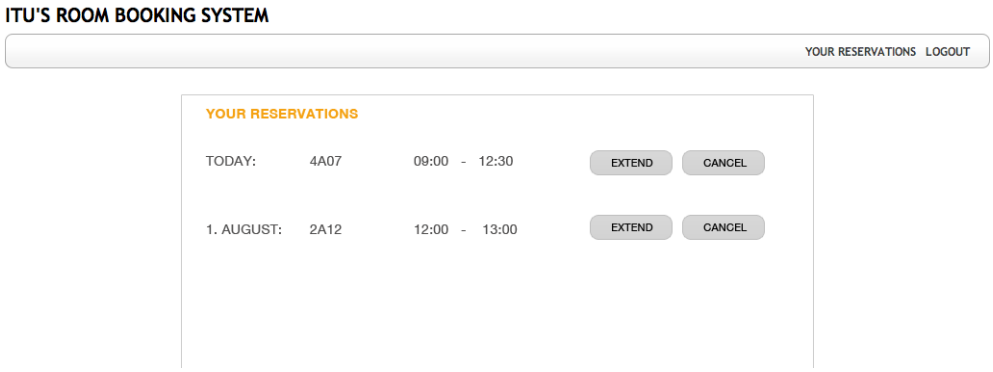

Figure 3.12: A users overview of ongoing/upcoming bookings

The personal reservation screen, reached by the link in the top menu, is displayed on figure 3.12. We have decided to omit past bookings from the view, as they would serve no real purpose other than increasing clutter for very active users.

Review of quality requirements

The quality requirements listed in section 2.5.1 state that:

- Without instructions, 90% of novice users must be able to complete T1.1 to T1.4 without variants.
- Without instructions, 80% of novice users must be able to complete T1.1 to T1.4 including variants.

Due to time constraints, we had to make the choice between many rounds of initial testing, and extensive test of final round. With the advice from aforementioned usability experts, we decided to focus out efforts on testing early and often, as opposed to late and extensive.

With the collected feedback from all rounds of testing, many of which have been on small iterations of the same screens, we estimate that these requirements will indeed be met by our target audience.

3.3 Semester planning

The key challenge for designing the semester planning user interface, is to find an optimal way to represent the data, and to provide a good overview

to support the user in the planning phase. This includes suggesting alternative solutions to problems, and provide enough views to help the user solve problems that have to be sorted out manually.

3.3.1 Wireframe

With our agile approach to designing the system, we have focused on getting testable mockups done as soon as possible. For most of the screens in the semester planning, this means drawing on a whiteboard where we can quickly add, remove and edit the screens until we have a satisfactory mockup, based on the data collected from interviews and analysis. However, for the screens which contains a lot of information, or requires a lot of user input, we have utilized the wireframe again to try and organize the data more systematic.

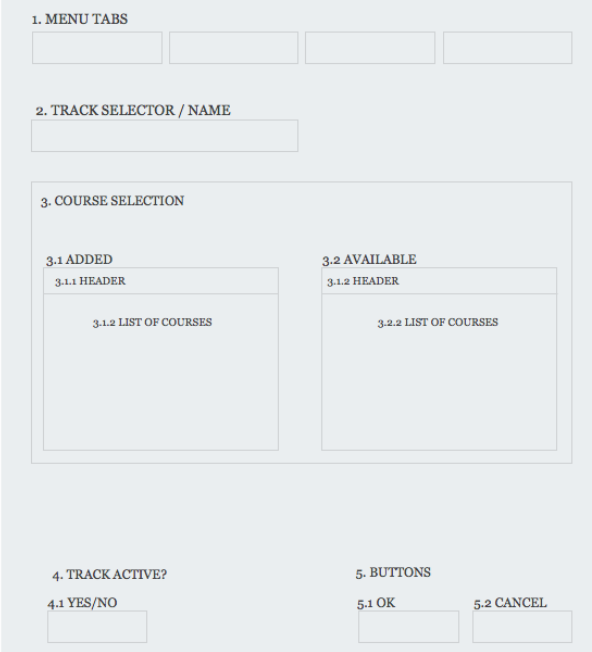

Figure 3.13: Wireframe for the track management

Our first whiteboard mockups had track assignment done via the course management, where the user would add the tracks to a course as it was created. We quickly realized that by decentralizing the track management and handling it on a course to course basis, an important overview would be lost.

This lead us to take all things track related away from the course management, and add it to a wireframe, shown on figure 3.13. With a single view

for all things track-related, confusion is less likely as users will not need to browse different screens, to gather all related information. [8].

The same view have been utilized for the teacher management, both out of convenience, and to reduce the amount of learning needed to succesfully use the system.

3.3.2 Design phase

Limiting the test

Our tests for the semester planning have not been as numourous as for the day to day booking. They have however been a lot more extensive in terms of time, tasks and discussions. We interviewed the member of Study Administration who currently handles the semester planning, and tested our mockups with her aswell.

The advantages of having *the* end-user available through the process are significant. It allows us to potentially leap over several iterations, as we can make the initial mockups a lot more detailed. Additionally, we can draw constant feedback and inspiration from a user with extensive domain-specific knowledge. It is important to note that there are potential pitfalls connected with this aswell. By narrowing down our test users, we run the risk of hiding potential issues that our specific test subject solves with tacit knowledge.

Our justification for limiting the testing to our expert end-user is based on several things. First, due to the system being custom made for the ITU, the value of testing with a broader audience diminish due to lack of domain knowledge held by the actual end-user(s). Secondly, the system is being developed under time constraints, and with the diminishing value in mind, we deemed the trade-off to be reasonable.

Data views

On figure 3.14 is the first effort for the result screen. The idea was to conceal data which was not immidiately critical to the assignment, thus leaving more room for relevant data. When this was put to the test, it was revealed that even though it was designed based on an interview, it was insufficient to our test subject; she wanted to see how each track was assigned. Moving along, the list view remained in place, though expanded with additional sortable columns, but a new view was added to assist in the track overview.

Figure 3.15 shows an example schedule for a track. This view is explained in section 3.3.3.

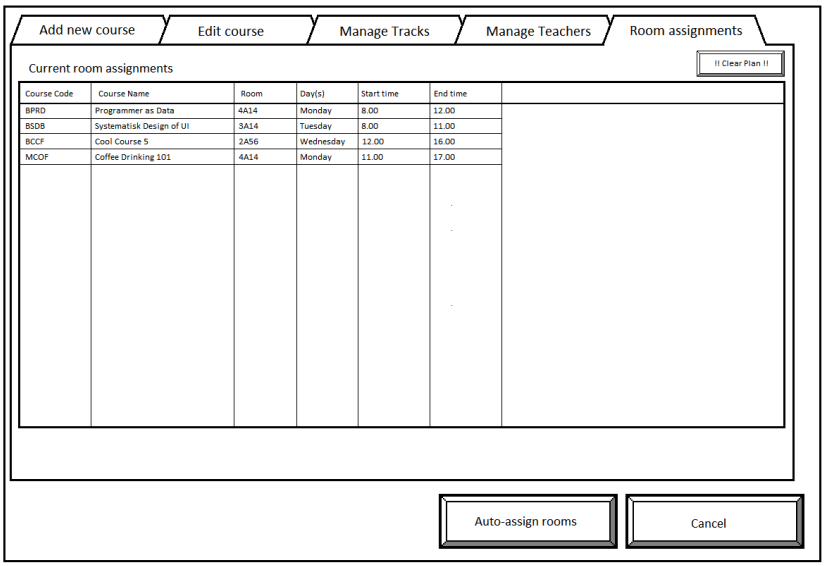

Figure 3.14: First version of the result screen from a succesful semester plan (with only 4 courses)

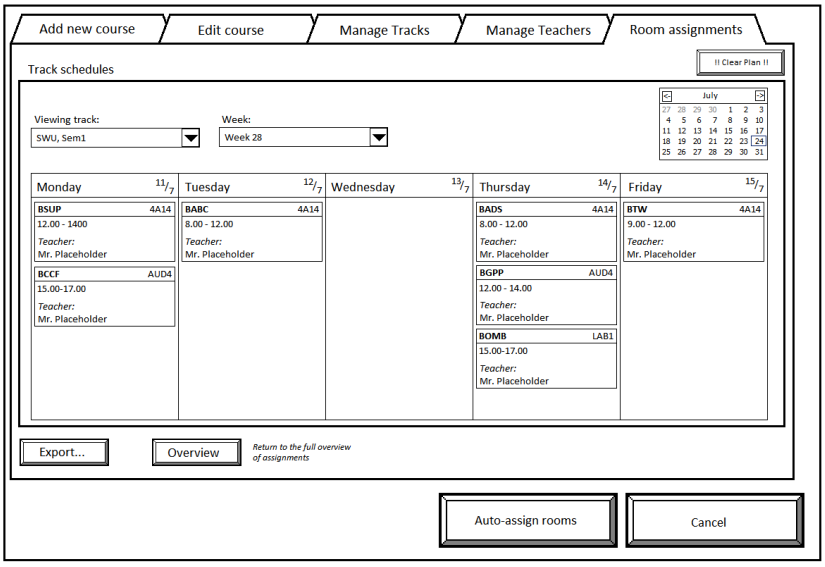

Figure 3.15: The schedule overview added in second iteration

3.3.3 Proposed design

Following is a description of all the screens needed to complete a semester plan.

Add course

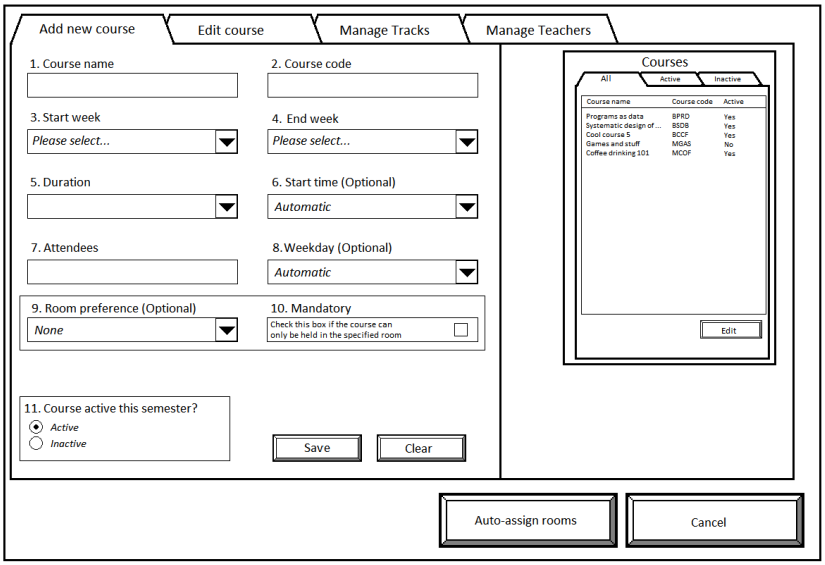

Figure 3.16: Tab for adding a new course to the system

The first screen users of the semester planning will encounter is the 'Add course' shown on figure 3.16. It consists mainly of basic data input, but with a few key points to note:

- **Numbering of input fields**
  Different users have different habits when it comes to reading input forms. To avoid confusion, we simply numbered the fields so they are filled in a desired order.

- **Use of Gestalt Laws**
  Our tests showed a bit of confusion regarding item 10, so we put a box around it in accordance to the *Gestalt Law of Closure*. The same was applied for item 11.

- **Optional and Automatic**
  Although items 6, 8 and 9 were labeled as 'Optional' in the first test,

it was not clear to our test subject that they would be filled out automatically if left untouched. This was solved by having the same info twice: Keeping 'optional' in the label, and renaming the default value to 'Automatic'.

- **Explanatory text**
  Generally we have tried to avoid explanatory texts in the system as a whole, simply to reduce the noise level [7]. However, to further alleviate the confusion caused around item 10, we have put a text inside a box next to the relevant checkbox.

- **Active/Inactive**
  Item 11 is used on both the course management screens and the track management screen. As the courses at ITU usually run in either spring or winter each year, it is possible to reduce the workload from semester to semester by allowing courses and tracks to be put to inactive when a semester ends, instead of having to recreate the same courses every 6 months.

- **Course list**
  This tabbed list exist to allow the user to keep track of which courses have already been added. Our mockup does not make it clear, but it is of course sortable. The 'Edit' button on the list will open the course in the 'Edit course' tab.

- **Save and Clear**
  These buttons should be self-explanatory, but important to note that 'Save' will also clear the input fields to make ready for next course, and add the course to the course list on the right hand side.

Edit course

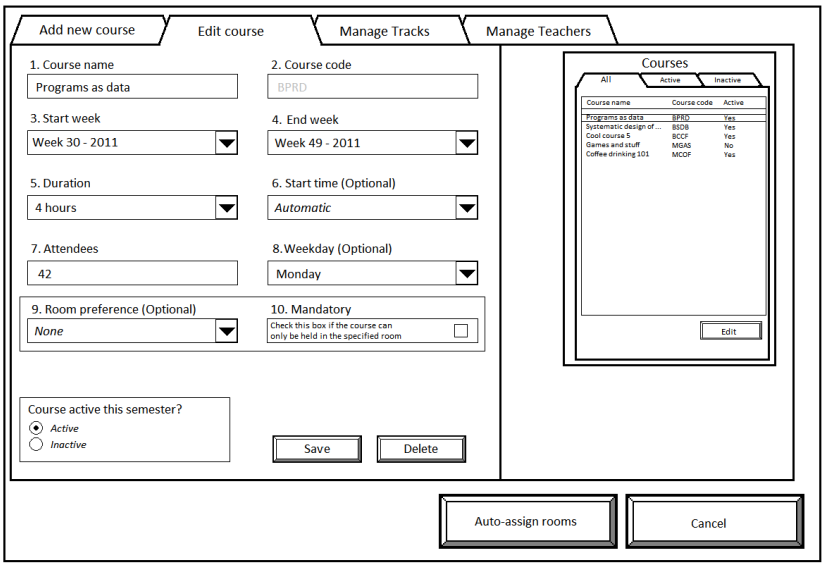

Figure 3.17: Tab for editing and removing a course from the system

The screen on figure 3.17 looks very much like the 'Add course' screen, but was made as a seperate tab to emphasize that an existing course was open. Additionally, it allows us to block input when a course is currently assigned in a full semester plan. As shown on the mockup, the course code is colored gray, indicating that it is not editable, as course codes are unique and should not be edited once created. In case the course code changes, a new course will have to be added.

Manage tracks

Figure 3.18 shows the track management. After courses have been added, the rest of the progression through the system is simply setting up constraints for the automated assignment, first of which is the tracks. Points of interest include:

- **Active/Inactive and buttons**
  These two elements are the same as on previous screens, allowing the user to only have to 'learn' it once.

- **Track courses**
  We decided to make a list with all the courses currently in the system

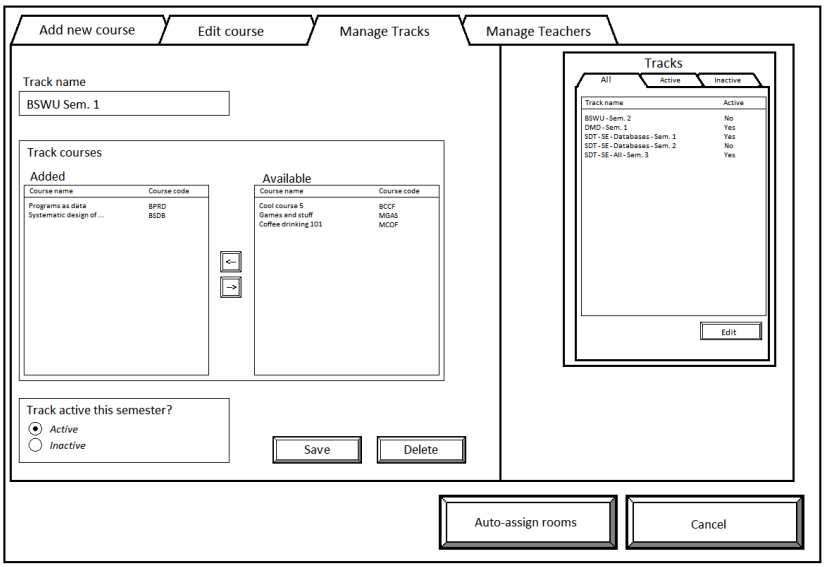

Figure 3.18: Tab for creating, editing and removing tracks from the system

and another list which can be filled with selections from it. This eliminates the need for searching and typing. Not visible in the mockup, is that the lists are sortable by column.

- **Track list**
  The course list from previous screens have been substituted for a similar list of tracks with the familiar 'edit' button.

- **Room preference** The room preference list can be used to add a constraint about which exact room the course should be assigned. This is relevant if a course requires e.g. an auditorium.

Manage teachers

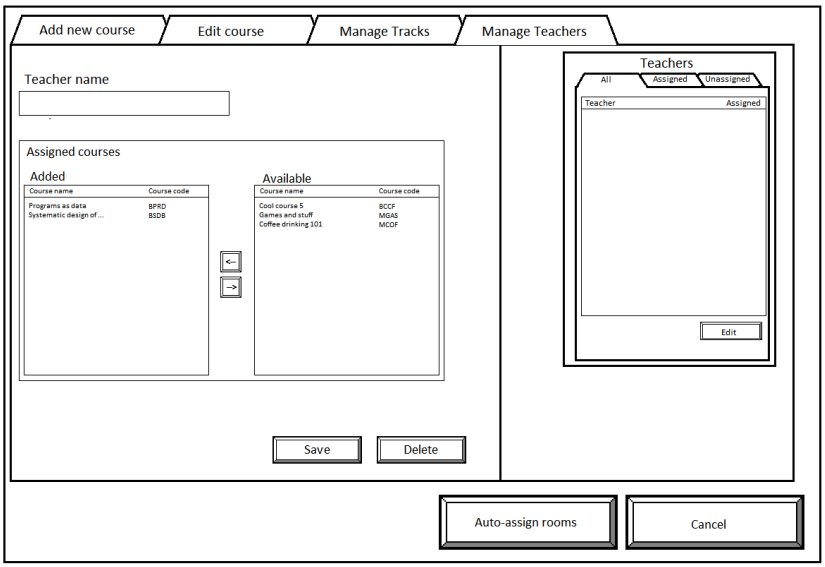

Figure 3.19: Tab for creating, editing and removing teachers from the system

Figure 3.19 shows the tab for managing teachers. As it's main function is, like tracks, to provide constraints for the planning, it looks very similar to track management. Only thing worth making note of, is the removal of the 'Active' selection, and the corresponding change to 'Assigned' in the right-hand list view. The 'active'-selection have been removed due to it being redundant. If a teacher is inactive, he will simply not be assigned to any courses. The list view will show a teacher as 'Assigned' if they have at least one active course associated with them.

Room assignment - Clash resolution

When the huge 'Auto-assign' button in the bottom of all the tabs is pressed and a full assignment is not possible, the clash resolution screen on figure 3.20 will appear. A clash happens when one or more constraints are in the way of assigning rooms to all courses. The constraints can be from tracks, teachers or course specific entries. The system will try to asses how the problem can be solved by loosening up constraints, affecting the smallest number of courses at a time, and sort them in that order. In the cases where a solution cannot be suggested, the user will be informed. The changes have to be done manually by the user. This is out of our scope, so the specific algorithm design will not be discussed further.

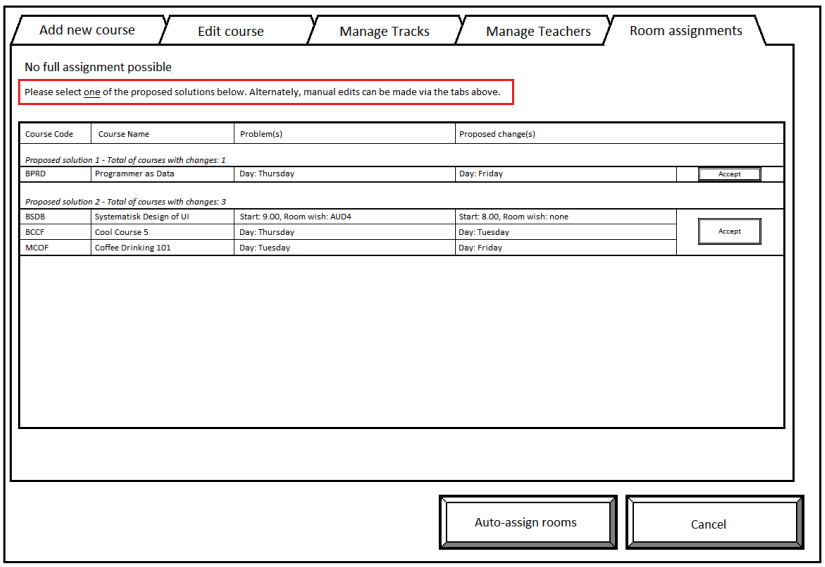

Figure 3.20: The screen appearing when a full assignment is not possible due to constraints.

The example clash on the mockup suggests a problem was found on a thursday, and provides two example solutions. From our test and discussion with the study administration employee, we learned that what she really desired was a specific answer to "what the problem was". This is an expected reaction, but in reality a highly complex problem. The 'problem' is that an assignment was not possible. What exactly causes the problem, is not one singular thing, but a combination of *everything* and our approach to simplifying it is therefore to suggest the solution that requires the least amount of changes.

If a proposed solution is not selected, but a manuel edit is performed instead, the auto-assignment will have to run again. Once a solution is selected, or an assignment have been made without clashes, the room assignment tab will remain visible.

Completed room assignment

Figure 3.21 shows the evolved version of the view discussed in section 3.3.2. The changes made can be summarized as:

- **Columns added**
  The columns 'Teacher' and 'Track' have been added, and all columns have been emphasized as sortable. In the cases where a course belongs

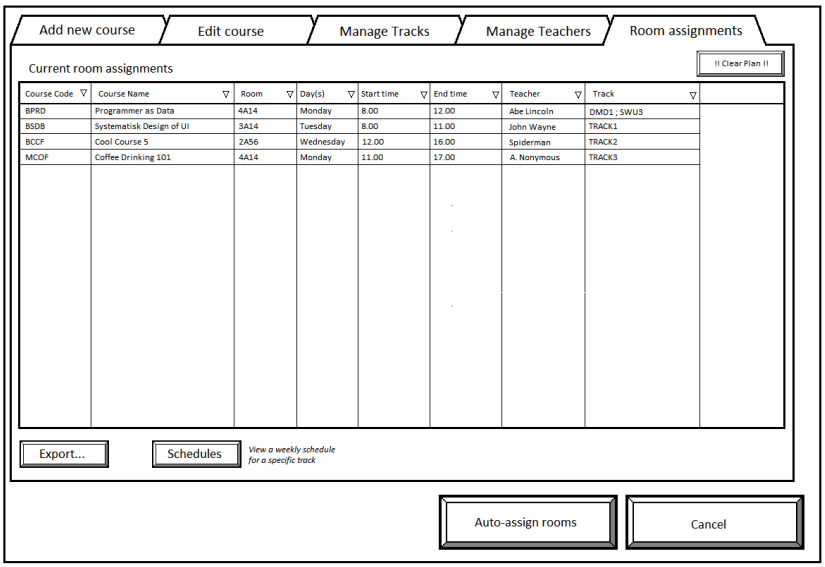

Figure 3.21: The screen appearing when a full assignment have been made

to several tracks and/or teachers they will simply be duplicated when sorting by those two options, as it is the easiest way to provide the overviews required. In general, duplicated data is to be avoided, but in this specific case it provides some flexibility by allowing grouping of entries, no matter how the constraints are put.

- **Export**
  The option to export the plan can be valuable for data manipulation at later stages. Because our data is presented in a table like format, it would be easy to convert the data to common file formats, e.g. XML og CSV.

- **Schedules**
  This button opens the schedule overview described in section 3.3.2 and in the following section. An explanatory text have again been added, to assist the user in discovering the view on first visit.

- **Clearing the plan**
  Placed in the top right corner is the button to clear the plan. Naturally, warnings and confirmations will be served on click.

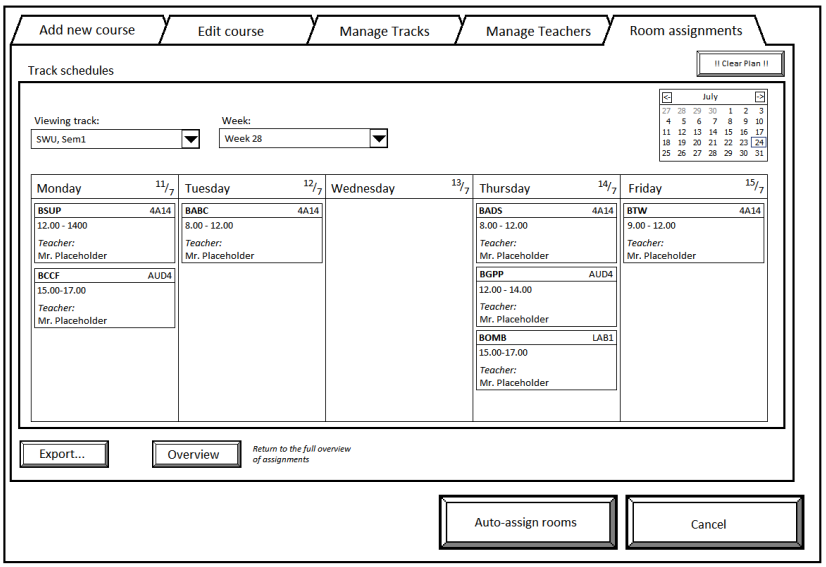

Figure 3.22: The schedules overview for tracks.

Completed room assignment - Schedules

The view on figure 3.22 was, as mentioned, added as direct result of our tests with the study administration representative. The different parts are:

- **Dropdown menus**
  The two dropdown menus control the area below. The 'Viewing track' lists all the active tracks in the plan, and the 'Week' lists the weeks of the year. A year selector has been omitted, as the week menu will simply contain weeks for e.g. a year ahead and default to the current week.

- **Calendar**
  A small calendar have also been added in approximately the same position as in the day to day booking part of the system, which will update the week-dropdown to the selected week. Current date will be highlighted as shown on the mockup.

- **Schedule view**
  The main area of this screen contains the weekly schedule for the selected track and week. Note that the boxes do not scale in height with time, as that would make short courses unreasonably small, or long courses very large. We also opted to omit the full course names, as the unique course codes are usually sufficient for our target audience.

- **Overview button**
  A click on this button will simply return to the previus list view of full assignments.

Review of quality requirements

The quality requirements listed in section 2.5.1 state that:

- With a maximum of 30 minutes instructions, 90% of administration personel must be able to complete T2.1 to T2.5.
- Without instructions, 50% administration personel must be able to complete T2.1 to T2.5.

For the first requirement, we can with certainty say that it is met, due to having designed the system with a lot of input from the representative from Study Administration that will be the end-user.

The second requirement have not been tested, as we could not produce realistic test subjects who had no knowledge of our system, but as with the first point, we feel that their involvement will prove to greatly reduce the learning curve of a fully implemented system.

Chapter 4

Implementation

The implemention of the application can roughly be divided into two parts: A server, being the layer between the user and the database, and a client presenting the data to the user. This chapter will describe the requirements for each part, the technical challenges of the project and how they were met.

4.1 Choice of platform

We decided to create a web based solution instead of a desktop application, mostly based on the nature of the project but also on our own interests. The web based solution does not require a client to be installed on the users system, which makes it instantly accessible. The webbased solution also makes the program available on any device with an internet connection; from a laptop, a smartphone or a dedicated monitor at ITU.

This is of course a tradeoff, as we have much more experience with developing desktop applications, which would probably be much faster and cause fewer errors to debug. However, we do think it was a natural choice to put the application on the internet, because of the large number of people who will use it every day.

4.2 Language

Since we mainly focus on the UI, we wanted to use a framework for the development process, giving us some functionality "out of the box". This would reduce the time spent on the technial implementation, and give us more time to focus on the UI.

We have chosen to use *Scala* [12] as the implementation language; a multi-paradigm language running on the JVM. Scala is a superset of java, featuring some functional programming constructs, like pattern matching and immutable types. While it is possible to use the java libraries, Scala provides many immutable alternatives to make concurrent programming easier.

A large motivation for choosing Scala, was the rich functionality provided by the Scala based web framework *Lift* [13]. Lift is designed as a framework for developing web applications, with a high focus on security and performance. Lift provides us with the following functionality relevant to our project:

- **Sitemap**
  A sitemap is used to represent the structure of a website. Each page in the structure can be specified to require a certain level of authorization to be accessible.

- **Sessions**
  Lift support handling of multiple users with sessions. A session is a cookie, with some information about the user associated with it.

- **Xml bindings**
  Lift uses xml as reference points for dynamic code injection. The static parts of the application are built using normal HTML. XML are used as placeholders, or bindings, for the data that are supposed to change during a session. This makes it possible to completely seperate the development of the presentation layer (the client), and the data layer (the server).

Lift is thus giving us a lot of the functionality we need to make our application work, but it also poses a large challenge, as Lift requires decent experience with Scala, which we do not yet have.

4.3 Development environment

During development, we have used a tool called Simple Build Tool(SBT) [6] to compile and test our application. SBT comes with a local web server (called Jetty), and a Lift project template, seperating the client and server. Appendix A.5 describes the basic structure of such a project.

SBT support automatic compiling, so when a file included in the project is changed, Jetty will automatically compile the changed file(s) and restart the webserver. Appendix A.1 contains a setup and user manual for the application.

4.4 Technical requirements

This section describes the technical requirements for the server and the client. General requirements have been described in section 2.4.2, and the actual implementation will be discussed in section 4.7.

4.4.1 Server

- **General authorization**
  The server should not allow access to the system unless the user is logged in. The sitemap functionality in Lift should be used to request authorization and present the appropriate pages.

- **User levels**
  In our system, we need two different user roles; one for staff members which should have access to the semester planning application(*super users*), and one for all other users (*basic users*). Specific user roles are assigned an integer indicating the users *level*. Super users are level 2, while basic users are level 1. The server should use the sitemap to only present the semester planning to super users, thus users with a level of 2.

- **Support of multiple users**
  Lift Sessions should be used to distinguish users from each other. The Session associated with a user is not a placeholder for the actual user *object*, but merely a unique identifier. The Session cookie should only contain the unique ID of the user, fetched from the database.

4.4.2 Client

- **Clickable map with server callback**
  The client should present a clickable map of the rooms at ITU. When the user selects a room, the view should change and let the user book the selected room for the current day. This requires that the client sends data to the server, which is usually done within a HTML *form*-element using a submit button. However, as we are dealing with a clickable

image, this is not an option. Lift does not support interactions with images, but does support callbacks from javascript functions. We can thus use a HTML clickable map [15], and attach a javascript callback through Lift, enabling us to call a function when the user selects a room.

- **Interactive Calendar**
  The users should be able to select a different day than the current, giving them the option to book a room in advance. JQuery-UI [2] offers a datepicker widget [1], which supports this functionality.

- **Custom time and day picker**
  The day/time-picker for our weekly overview is a custom-made widget, as it requires functionality not provided by any standard javascript widgets. Since Lift support callbacks through javascript, it is possible to create such a widget by using this functionality together with simple HTML attributes as date and time identifiers.

- **XML Exporter**
  It should be possible to export the data created during semester planning. Since the data will be used in multiple ways (for printing, digital archiving, etc.) the output should be in XML for maximum application support.
  The exporter has to be manually created, e.g. by using a Java XML generator.

4.5 Database design

From the requirements in section 2.4 and the technical requirements above, we derived the database design seen in figure 4.1. While parts of the design are trivial, here follows the ideas behind some of the more non-triviel elements.

4.5.1 Course and planning table

The course tabel contains information about a specific course. It should be possible to create a course without immediately adding it to the semester planning, which is why the two tables seem to contain some of the same data. The *room_req* field is a *room request* field. We learned from our interviews (appendix A.4) that teachers often have specific room requests, which this field reflects. This might be a wish, or an actual request which should

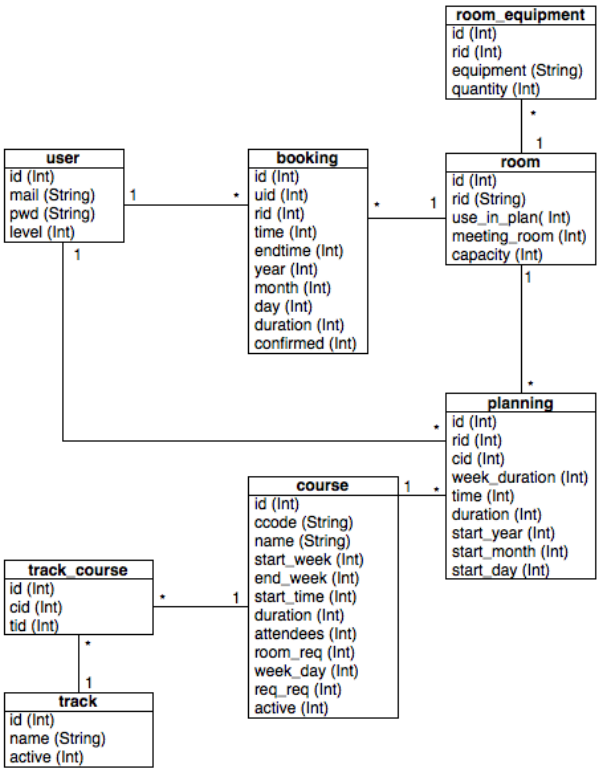

Figure 4.1: Database design

be respected by the application, if e.g. a course requires equipment only available in one room. In that case, the *req_req* field is set to 1, indicating that this is a constraint and not a wish.

4.5.2 Day and time modelling

We have chosen not to use the built in date field type in MySQL, because we use javascript to exchange data between the client and the server. Javascript does not understand this data type by default, and while it might be possible to write a function that creates an object representing the same type, we will still have to deal with string splitting and concatenation. This is very inefficient and takes up more space compared to using simple integers.
We have therefore used integers to represent all kinds of date and time data. An example is the *booking* table, which contains information about a specific booking. *time* and *endtime* represents the start time and end time of the booking respectively, by the number of minutes after midnight. If a booking

starts 10.30 in the morning, the start time field will thus be 630. This way of representing the time has a couple of advantages:

- Simple calculations can be used to find the duration of a booking, and the actual time the fields represent. We do not need to work with strings, which makes the code clearner and more efficient.
- For the same reason, user validation is quick and easy to perform.
- We can still execute specific date and time sql queries by using basic relationel operators, which makes it more efficient to e.g. lookup bookings after a specific time.

4.6 Concurrent room selection

While we expect a lot of users to actively using our application at the same time, the problem with concurrent room selections has to be adressed. While the actual implementation has been left out because of our focus for the project, we have had a lot of considerations about how this can be solved.

Scenario and possible solutions

These solutions are based on the simple scenario where users **X** and **Y** are searching for rooms at the same time. They could obviously be extrapolated to cases where even more users are interacting. We assume that **X** was the first user to click the search button. Online users should be tracked, to make the logging possible.

Hide bookings from certain users

Make all rooms that fits **X**, unavailable for **Y** while **X** is deciding on which room to use.	
Pros	**Cons**
Easy to implement	If all rooms fit the search of **X**, no rooms will be available for **Y**. If there are 10 rooms in total, and 10 users search for 10 random rooms, no rooms will ever be available for the next user.

Dynamic refresh

The screen which shows available rooms for user **Y** should update when user **X** selects a room for booking.	
Pros	**Cons**
High usability and responsiveness	Increased number of requests to the server.
More flexible; allows several users to have many choices at once.	If user **X** changes his mind during a booking, user **Y** misses a valid opportunity to book the room.
Ultimately, this solution is a tradeoff between the probability of concurrent selections occuring and server-request rate. If the server is set to refresh each second, this might reduce the probability of errors, but the server might not scale very well.	

User awareness

Information about whether or not other users are looking at the same room will be presented for the user, to make him/her aware that other people might be interested in the room.	
Pros	**Cons**
It does not come as a sudden surprise if the room is booked before them.	Increased number of requests to the server.
Basically this is not a different solution than **dynamic refresh**, but just a way of minimizing the server requests, making it perform better, but increases the probability of concurrent selections.	

Of all these solutions, the most effective would be **dynamic refresh**. This however, is also generating the most server requests, but since the number of seconds between each refresh can be varied, this seems like a flexible solution as well. It could be made even better, by making each client sending a signal to the server when a booking was made, so the server could notify the clients.

4.7 Actual implementation

4.7.1 User authorization

User authorization has been implemented using the built-in Sitemap feature of Lift. In a Boot file, used to initialize the program, a sitemap object is created:

```
val siteMap = SiteMap(
    Menu("Index") / "index" >> AuthRequired >> Hidden,
    Menu("Advanced") / "advanced" >> Hidden,
    Menu("RoomBooking") / "room-booking" >> AuthRequired >> Hidden,
    Menu("Login") / "login" >> LoginAuth >> Hidden,
    Menu("Your Reservations") / "reservations" >> AuthRequired,
    Menu("Logout") / "logout" >> EarlyResponse(() => {
        Session.loggedIn.apply(0)
        Full(RedirectResponse("login"))
    })
)
// Tell Lift to use our sitemap
LiftRules.setSiteMap(siteMap)
```

Each line represents one page, which is added to the static menu simply by using its constructor. The name after the slash is the actual name of the HTML page. A page can be assigned several functions which should be called before each page is evaluated and shown. In our sitemap, all pages should call the *AuthRequired* function, which redirects the user to the login page if not logged in. Several pages calls the *Hidden* function, which is a built in function in Lift. Hidden pages are not shown in the menu, and cannot be accessed directly.
The logout menu item uses an anonymous inner function, to sign out the user, by setting the session cookie to 0 (see section 4.7.2), and redirecting to the login screen.

4.7.2 User sessions

Handling of multiple users in Lift is done by using the built in *SessionVar* class:

```
object Session{
    object loggedIn extends SessionVar[Int](0)
    object uid extends SessionVar[Int](-1)
    object currentRoom extends SessionVar[String]("-1")
}
```

While our *Session* object is static, a cookie for each SessionVar is stored on the users computer when the object is created, thus making it possible for the users to share the global Session object, while still having individual

data stored in the SessionVar object.
As an example, take the uid object. This extends a SessionVar of type Int, with a default value of -1. When the user signs in, this value is changed from the default value to the specific id of the user, fetched from the database.

Setting and retrieving the values can be done in constant time:

```
// In another class, do something with the user id
room.book(Session.uid.is)

// ... Change it, e.g. when the user signs out
Session.uid.apply(-1)
```

This is a convient way of always having access to the unique id of the user.

4.7.3 Quick booking

When the user click the *Quick Booking* button on the frontpage (see section 3.2.6), the application searches the database for the smallest, available room.

JQuery is used to attach a click-listener to the button:

```
jQuery('#quickfind').click(function(){
    liftAjax.lift_ajaxHandler('""" + funcName2 + """='+uid, null, null, null);
});
```

funcName2 Queries the database for an available room, and returns the room id if any is found:

```
val func2 = () => {
    var roomid = Database.findRoomNow
    if(roomid != -1){
        var booking = Database.bookRoom(rid = roomid)
        ...
    }
    else{
       // No rooms
       ...
    }
}
```

The *bookRoom* function of the *Database* object takes a minimum of one parameter. The parameters not called, are filled with default options assigned by the predefined method signature:

```
def bookRoom(uid : Int = Session.uid.is, rid : Int,
             from : Int = now, to: Int = now + 240){ ... }
```

Here, the user id (uid) is retrieved from the *Session* object, the beginning time of the booking is set to the current time, and the booking is by default set to last 4 hours.
The function books the room, and notifies the user.

If the user for some reason changes his mind and wants to cancel his booking, the cancel button can be used. This will delete the booking that was just created.
It might seem strange that we first book the room, and then cancels it, instead of just booking the room when the user accepts it. However, by booking the room immediately, we avoid concurrency issues which could occur if user **Y** books the same room as user **X** is about to accept.

4.7.4 Simple Booking

The entry to simple booking interface is the map on the left hand side of the application. As mentioned in section 4.4.2, the map should be clickable. The image of the map has been created from scratch in Photoshop. We used the map of the floors of ITU located in the elevators, to draw the rooms and their approximate size. The finished image was then sliced into 1 image for each floor. The images are simply put on top of each other in the HTML structure, and an attribute referencing the correct HTML map has been added:

```
<img src="img/floor4.png" usemap="floor4map" />
<img src="img/floor3.png" usemap="floor3map" />
<img src="img/floor2.png" usemap="floor2map" />
<img src="img/floor1.png" usemap="floor1map" />
<img src="img/floor0.png" usemap="floor0map" />
```

A map element contains x number of area elements, which represents a clickable area on the map. The following map represents the clickable areas of the first floor:

```
<map name="floor1map">
<area shape="rect" coords="68,2,147,59" id="aud2" />
<area shape="rect" coords="68,149,173,207" id="aud1" />
</map>
```

The coordinates had to be inserted manually, by using a digital ruler to measure the pixel values of the corners. A Javascript listener has been added to all area elements. When an area element is clicked, the Javascript function fetches the id attribute and uses this to redirect the user to a weekly overview of the correct room.

The Simple Booking time selector function had to be implemented from scratch. We wanted to make a function that allowed the user to select a time range, on a specific day of the week. (See section 3.2.6 for screenshots of this functionality).
We decided to only use Javascript for this, by including all relevant information in the HTML-tags, and only use Lift to call a function to book the room in the selected time span. Each day is divided into blocks from 7 in the morning to 21 in the evening. It is possible to select even and half hour blocks:

```
for(var j = 7; j < 21; j++){
    t = j * 60;
    t30 = t + 30;
    html = html + '<div class="timeBlock block'+t+'"></div>';
    if(j != 20) {
        html = html + '<div class="timeBlock block'+t30+'"></div>';
    }
}
```

Block420 is thus the block representing 420 minutes after midnight = 7 in the morning. JQuery is used to store the time values when a user clicks on a timeblock. JQuery can react on all elements with the class timeblock, and use the *this* selector to reach just the element the user selected:

```
$("#theGrid li").not(".topLabel").find(".timeBlock").click(function(){
    beginClick = (!beginClick);
    var timeSelector = (($(this).attr('class')).split(" "))[1];
    var timeLabel = $("#theGrid ul li.topLabel").find("."+timeSelector);
});
```

The *timeLabel* variable are set to the node in the header which has the same *timeSelector* class as the block the user selected. The actual time is shown in human readable form in the inner html of the header, making it easy to retrieve the actual time the user selected through this node, by calling the *html()* function of the *timeLabel* variable.

Chapter 5

Evaluation

In this chapter, we will evaluate some aspects of the project, to reflect on the actual success of the choices we made.

5.1 Impact

The following is an attempt to estimate what impact a fully implemented version of our system would have for the relevant stakeholders.

5.1.1 Launch

Before a fully implemented system could be launched, policies and decisions should be made for the issues raised in section 2.6. With any kind of changes to current procedures in an institution of this size, there will always be a transistion period where things are not being used properly, not being used at all. We hope that the intuitive and user friendly approach we have taken, will help ease this particular part of the launch.

5.1.2 Facilities Management

As previously stated, FM handles the day to day booking manually, by manning a desk and browsing/updating a binder. With our system it will depend on whether or not they choose the approval approach mentioned in section 2.6.1. In case of using approval, their work would be faster, but in the same approximate amount. If they instead choose to follow our recommendations and allow users to make their own bookings, in limited number, and have the power to delete them, their time spent managing rooms could be greatly reduced.

5.1.3 Study Administration

This is the single greatest benefactor of a succesful implementation, as the current semester planning is highly time consuming, and quite troublesome. With our system, the need for Tabulex would be gone, along with the manual work to fill the gaps that Tabulex cannot do automatically. Through our discussion with the representative from the Study Administration, she mentioned that getting the overview of whether a not a track was fully assigned was the real problem. Our proposed design suggest a track-based approach with easy overview on how the schedule and assignments for a single track looks. The automated suggestions for clash-handling would also enable them to make educated decisions on what courses need to be moved and which rooms to be reassigned. Naturally, our system would not be able to solve all the problems encountered in such a complex assignment, but *any* suggestion is more than what they have currently.

5.1.4 Students, faculty and staff

For the daily visitors to ITU, finding a meeting room or workspace would be greatly simplified, as the need to wait in line at the information desk would be eliminated. Additionally, rooms can be reserved from home, so no one would have the dissappointment of showing up, only to discover that no room is available.

5.2 Expansions and improvements

The following is a list of features we would have liked to be a part of the systsm, if we had had the time. Some parts of our application could also be improved, these are mentioned here as well.

- **Responsiveness**
  The application does not use any ajax or JQuery to improve the user experience. Ajax could have been used to avoid refreshing the page when e.g. selecting a room to book from the map. JQuery could have been used to create fluid animations, like sliding effects, for a better user experience.

- **Timespan dragging**
  When you select when you want to book a room, you select a timespan by first clicking on the starting time of your booking, and then the endtime. One of our test users tried to drag from the start- to end

time, which obviously did not work, because we had not implemented it. We mention earlier that we will allow for both, but only the click solution have been implemented in the prototype.

- **Map overlays to show availability**
  While we have covered the need for finding a specific room, as well as just any available room, it is not possible for the users to get an overview of the current room availability, which might be practical for some users. We would have liked this to be implemented as an overlay on the frontpage map, so a room colored with a red background meant that the room was currently in use.

- **Integration with the student base**
  With the proper integration, the system could utilize all the information on people currently in the student base, and use this database as the main user base for the system. People present in the student base, would automatically have access to our application, using the same credentials.

- **Mobile version**
  The ability to access the system from a mobile device would allow users to book while in transit for example. Note that many devices already have decent support for normal webpages.

- **Office management**
  Anoter possible expansion specific to ITU, is the ability to manage the offices we have excluded from our scope. In theory, it could function a lot like the semester planning, but would obviously be less complex, due to the fact that offices are booked for a longer period, several people can fit into them and there are currently enough space for all the people requiring it.

- **Better error handling**
  While we have testet major error handling in our application, like when a clash happens during semester planning, we have not tested error messages for minor errors. This would have to be implemented in a finished product, and the error messages would have to be tested, to make sure they were understandable for the user.

5.3 Reflection

5.3.1 Lift

We chose Lift as our framework because of the rich features it contained, which could serve our project well. Before the project, we had no experience with Lift or Scala, but Lift seemed like a good choice for our application, and while we have lots of experience with java, we thought that Lift would be easy to get used to.
However, we quickly found out that Lift used Scala to an extent which was very hard for us, as new to the language, to comprehend. Lift uses the functional paradigms extensively, and combined with the lack of documentation, it was very difficult for us to figure out what the various classes was for, and how they should be used.
We did however manage to use the some of the features of Lift that we intended, but the problems we had with Lift are reflected in the final result of our product. We did not have time to implement as much of the application as we wanted, and we do not use the full potential of Lift in some of our classes.

When looking back, Lift was still the right choice for the application, but probably not the right choice for us personally. We would have been able to implement a larger part of the system if we used something we were familiar with, like PHP, from the beginning.

5.3.2 Choice of project

This type of project has been very different to us, as none of our previous projects have been focusing on interfaces. We have only had one single course about designing user interfaces, on our first semester.
This project was a nice change of pace, and something we enjoyed and learned a lot from. During the project, we learned how much of a difference a well designed user interface actually makes, when doing the usability tests.
While it has been a good experience, it has also been a challenge. Becuase of our rather limited knowledge and experience with GUI building, we basically started from scratch. We had to study a lot of literature to be able to base our choices on other than personal taste and intuition. Before we finally decided on the first mockup, which we afterwards tested and changed, we spent a lot of time on initial mockups written on whiteboards, which later were discarded because they simply did not work the next day we revisited them. This process was repeated several times before the first initial mockup

of the frontpage, and a couple of times in between the iterations, after the actual usabilty tests.
With all initial and tested mockups included, we've done about 15 mockups for our relative simple application.

Chapter 6

Conclusion

In this project, we have developed a graphical user interface and a simple prototype, for a room management system for use at the IT University of Copenhagen.

We have divided the project into two main areas; a room booking system which can be used by all members of ITU, and a semester planning system, which should be used by the persons currently in charge of room allocation for the courses of a semester.

Our focus has been on the graphical development, because we think a good interface is of high importance in a system where overview is key. We have used the domain knowledge gathered throughout the project in our graphical user interface and prototype, to make the application as intuitive as possible to use for the target audience.

We have been using a iterative approach, which resulted in testable mock-ups early in the process. These mockups have been tested and redone through several rounds, with the help of several test subjects performing think aloud tests. We have tested the final version of our interface, and the results were generally satisfying for all purposes.

We have developed a working prototype, which demonstrates one way to book rooms using our system, and the general final concept of the user interface. The application has been implemented as a web-based solution, to make it easily accessible from anywhere. We have used the multi-paradigm programming language Scala in the development, and while it did give us some problems, it provided us with some important functionality for our project.

We have finally suggested the steps required to put our system to use, and discussed the impact our system would have on the domain it was built for.

Bibliography

[1] Datapicker/calendar widget reference. http://jqueryui.com/demos/datepicker/.

[2] Jqueryui home. http://jqueryui.com/.

[3] Tabulex ApS. Home of tabulex. http://www.tabulex.dk/.

[4] DEA. Ems campus home. http://www.dea.com/ProductsAndServices/Campus/Default.aspx.

[5] Jesse James Garrett. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. New Riders Publishing, 2010.

[6] Mark Harrak. Simple build tool for scala. https://github.com/harrah/xsbt/wiki.

[7] Steve Krug. *Don't make me think*. Pearson Education Ltd., 2005.

[8] Soren Lauesen. *User Interface Design; A software Engineering Perspective*. Pearson Education Ltd., Essex, 2005.

[9] NetSimplicity. Meeting room manager home. http://www.netsimplicity.com/.

[10] Jakob Nielsen. Alert box january 21, 2001: Usability metrics. http://www.useit.com/alertbox/20010121.html.

[11] Jakob Nielsen. Alert box march 19, 2000: Why you only need to test with 5 users. http://www.useit.com/alertbox/20000319.html.

[12] Martin Odersky. Scala language. http://www.scala-lang.org/.

[13] David Pollak. Lift framework. http://liftweb.net/.

[14] Joshua Porter. *Designing for the Social Web*. New Riders Publishing, 2008.

[15] W3C. Html map and area elements. http://www.w3.org/TR/html40/struct/objects.html#h-13.6.1.

Appendix A

Appendices

A.1 Setup and user guide

We have developed the application in a local environment, making it possible to run the program on any computer with the right setup. The server has been tested running on a Mac with OS X Snow Leopard / Lion and Windows 7 Pro 32bit. The client has been tested in Chrome version 11 and 12, but there should be no complications using any Webkit based browser.

A.1.1 Installation

The following should be installed on the system, and be present on the class path.

- Java version 1.5 or later.
- Scala 2.8.1 or later.
- Simple Build Tool (SBT) 0.7.4 or later.
- MySQL version 5.1.44 or later.

A.1.2 Setup

- Extract the content of the .zip archive **itu_room_booking.zip** to a new directory (here called **root**).
- Open CMD on Windows, or Terminal on Mac and cd to **root**.

- Start your MySQL server if it is not already running. We have used XAMPP for Mac, which comes with a built in interface to start and stop the server.

- Type **mysql –u username –p password <database_setup.sql** where *username* is your MySQL username ('root' by default) and *password* is your MySQL password.
  This will create and setup the database. Note that you can skip the '–p password' part if you do not use a password.

- Open the file **root/src/main/scala/bootstrap/liftweb/Boot.scala** and edit this line:

  Full(DriverManager.getConnection('jdbc:mysql://localhost/ituplan','root',''))

  to this:

  Full(DriverManager.getConnection('jdbc:mysql://localhost/ituplan','mysql_username', 'mysql_password'))

  where *mysql_username* and *mysql_password* is the same username and password as entered before. If you do not use a password, just use empty quotes ('').

A.1.3 Running the application

- If you are not there already, cd to the **root** directory.

- Type **sbt** and press enter.
  If SBT is present on the class path, the environment will load.

- Type **update**.
  SBT will locate and download all dependencies for the project, including the Lift framework.

- Type **~jetty-run**.
  This will compile the application, and start the built in Jetty-server. Note that a bug might result in the Java virtual machine running out of memory. In that case, start sbt and jetty again.

- Open your browser, and go to **localhost:8080**.

- We have created a test user where the username is **a** and the password is **a** as well.
  Use this to login to the application.

The QuickBooking function is fully implemented in the prototype. Also, the sitemap and session functionality gained through lift is working. The simple booking functionality is not connected with the server, but the day and time selector has been implemented.

A.2 Tasks

Simple task list

1. Day-to-day booking

T1.1 Book room. May or may not be a specific room or a specific time/day

T1.2 Check availability of room. May or may not be today.

T1.3 Cancel booking. May or may not be before the booked timeslot.

T1.4 Extend booking. May or may not be before the booked timeslot.

2. Semester planning

T2.1 Create course. May or may not have full information available at the point of creation.

T2.2 Create track.

T2.3 Edit course.

T2.4 Edit track.

T2.5 Plan semester. A full solution may or may not be possible

Detailed task list

T1.1:	Book room
Start:	User opens system
End:	User have gotten a room
Frequency:	Average of 10 bookings pr. day
Difficult when:	No rooms are available

Subtasks	**Example solution**
1. Find room	
1a. No specific room needed	System suggests any available room
1b. User has specific needs	System allows to search by parameters
1c. No available rooms	System allows to search at future dates
2. Book room	
2a. User does not specify booking duration	System suggests a standard time duration
2b. User specifies time interval	
3. Confirm booking	
3a. User confirms	
3b. User declines	System cancels the booking

T1.2:	Check availability of room
Start:	User selects a room
End:	User has gotten the required information
Frequency:	Average of 15 lookups pr. day
Difficult when:	Checking past current week

Subtasks	**Example solution**
1. Select room	
2. View availability	System shows availability for current week
2a. For day in current week	
2b. For day past current week	System allows browsing by week

T1.3:	Cancel booking
Start:	User has an unwanted booking
End:	Booking is cancelled
Frequency:	Average of 0.5 times pr. day
Difficult when:	User has many bookings
Subtasks	**Example solution**
1. Select booking 1a. User has one booking 1b. User has several bookings	 System shows a sortable list
2. Cancel booking	
3. Confirm cancellation 3a. User accepts 3b. User declines	 System cancels booking System does not cancel booking

T1.4:	Extend booking
Start:	User needs more time with a room
End:	Booking is extended
Frequency:	Average of 2 times pr. day
Difficult when:	Room is already booked
Subtasks	**Example solution**
1. Select booking 1a. User has one booking 1b. User has several bookings	 System shows a sortable list
2. Check availability 2a. Room is available 2b. Room is unavailable	 System suggests similar room
3. Extend booking 3a. User does not specify booking duration 3b. User specifies time interval	 System suggests a standard time duration
4. Confirm extension 4a. User confirms 4b. User declines	

Note that tasks for track/teachers are essentially the same, and examples have been given for track only.

T2.1:	Create course
Start:	Course data is available
End:	Course have been created
Frequency:	Average of 100 courses pr. semester
Difficult when:	-
Subtasks	**Example solution**
1. Create course	
2. Fill data 2a. All data is available 2b. Not all data is available	 System allows to partially fill in data
3. Submit course	

T2.2:	Create track/teacher
Start:	Courses have been created
End:	Track has been created
Frequency:	Average of 15 pr. semester
Difficult when:	-
Subtasks	**Example solution**
1. Create track	
2. Assign course(s) to track	
3. Save track	

T2.3:	Edit course
Start:	Updated course data is available
End:	Course has been udpated
Frequency:	Average of 10 courses pr. semester
Difficult when:	-
Subtasks	**Example solution**
1. Find course 1a. The unique ID of the course is known 1b. The unique ID is unknown	 System sorts by ID System allows sorting by long name
2. Update data 2a. Remove data/contraints 2b. Add data/contraints	
3. Save changes	

T2.4:	Edit track/teacher
Start:	Courses needs to be added/removed from track
End:	Track has been updated
Frequency:	Average of 5 pr. semester
Difficult when:	-

Subtasks	**Example solution**
1. Find track	System shows list of long names
2. Update track 2a. Add course(s) 2b. Remove course(s)	
3. Save track	

T2.5:	Plan semester
Start:	Courses and tracks have been created
End:	All active courses have a room assigned
Frequency:	Average of once pr. semester
Difficult when:	No solution is possible due to constraints

Subtasks	**Example solution**
1. Start the planning	
2. View result 2a. No problems occur 2b. Clash(es) occur	System suggests changes to courses that would allow for a solution
3. Tweak result 3a. Alternatives were provided 3b. No alternatives	Manual tweaking of courses necessarry
4. Confirm track schedules	System provides a track-by-track overview
5. Approve semester plan	

A.3 Usability Tests

A.3.1 Day to day booking

Round 1

The following scenarios were presented to the test subjects:

1. You and your group have met, and intend to conduct group-work without being disturbed. (Task: T1.1)

2. Your group has finished work today, and decided to reconvene tomorrow. You have been assigned to arrange for a place to work. (Task: T1.1)

3. Your group is done with the assignment and wish to practice your powerpoint presentation using a projector in the near future. (Task: T1.1)

4. While working in (room which was just found) you have been interrupted by another group asking when the room is next available. (Task: T1.2)

5. Add-on: They refuse the first time you suggested as they have gone home by then/don't work on that day/have vacation that week. (Task: T1.2)

Test subject 1, ITU student1:

1. Presses 'Find me a specific room'
   Skims through, looking for an easier option
   Fills out search options partly, then realizes it's too complicated and clicks cancel
   Hesitantly presses 'Find me a room'
   Picks start and end time
   Presses book
   Result: Task failure due to overcomplication - Button labels are unclear

2. Presses 'Find me a specific room'
   Spots calendar just after clicking, clicks cancel and selects tomorrow on the calendar
   Clicks 'find me a room'
   fills in and accepts

Result: Minor problem - drawn towards 'specific' instead of using calendar when other dates are involved

3. Confidently presses 'find me a specific room'
   Checks off 'projector' and todays date, leaves rest default
   Result: Pass - no remarks.

4. Initially puzzled. Asks: "Can I click the map?", clicks before evaluator confirms
   Clicks the room he was granted from search
   Uses finger to trace timeline from left to right
   Provides multiple options: Later today and tomorrow (with specific time intervals)
   Result: Pass - Note: doubt whether map was clickable.

5. Add-on excluded as test subject already provided multiple answers to 4.

Test subject 2, tech savvy outsider:

1. Skims interface and hesitates. Clicks 'find me a room'
   Fills out start and end time
   Doesn't notice header showing which room was chosen
   clicks a random room on map
   Realizes mistake and clicks cancel
   Clicks 'find me a room', fills out again and presses book
   Result: Task failure due to overcomplication - Didn't notice that a room was assigned - Header hidden by clutter

2. Presses 'find me a room'
   Clicks 'Another day' and selects tomorrow
   Fills in time and presses book
   Result: Pass - note: chose 'another day' instead of calendar, both are valid

3. Clicks 'find me a room'
   Fills in time and accepts
   Evaluator informs that the room does not have the required tools (projector)
   Clicks 'specific', checks off projector, searches and books.
   Result: Medium problem - Simplifies problem leading to potential task failure, but lowered severity to medium as question formulation might have been the problem, and task was solved flawlessly once problem was understood

4. Clicks the room on map
   Informs that room is available from 13.00 today
   Result: Pass - no remarks.

5. Evaluator declines 13.00 today and requests next week
   Clicks calendar
   Informs that monday 9.00 is free
   Result: Pass - no remarks

Round 2

The following scenarios were presented to the test subjects:

1. You and your group have met, and intend to conduct group-work without being disturbed. (Task: T1.1)

2. Your group has finished work today, and decided to reconvene tomorrow. You have been assigned to arrange for a place to work. (Task: T1.1)

3. Your group is done with the assignment and wish to practice your powerpoint presentation using a projector in the near future. (Task: T1.1)

4. While working in (room which was just found) you have been interrupted by another group asking when the room is next available. (Task: T1.2)

5. Add-on: They refuse the first time you suggested as they have gone home by then/don't work on that day/have vacation that week. (Task: T1.2)

6. Group member calls in sick, and work tomorrow is cancelled. (Task: T1.3, T1.4)

7. You have been tasked with gathering all the first-year students for a surprise announcement in Auditorium 1. Date is not set. (T1.1)

Test subject 3, ITU student2:

1. Clicks a meeting room on map
   Fills in time and books
   Result: Minor problem - Solves task by using knowledge of ITU

2. Clicks calendar tomorrow
   Clicks find room and accepts
   Result: Pass - no remarks

3. Clicks specific
   Checks off projector and books
   Result: Pass - no remarks

4. Clicks room on map
   Dialogue does not offer a way to view tomorrow, subject is puzzled
   Cancels, and begins reading, realisizes calendar can be used
   clicks calendar, clicks room, provides answer
   Result: Medium problem - functionality unintuitive/lacking

5. Browses days by calendar
   Result: Pass

6. Starts reading interface
   Does not notice top panel until almost giving up
   Selects booking, cancels booking
   Result: Annoying - Did not spot until having read through whole interface

7. Clicks auditorium 1
   Begins browsing with calendar and decides on next tuesday
   Result: Pass - no remarks

Test subject 4, "average" outsider1:

1. Clicks 'find room'
   Accepts
   Result: Pass - no remarks

2. Clicks 'find room'
   Overlay covers calendar, can't click
   clicks cancels
   (note: forgets about calendar it seems)
   Click 'find specific'
   Fills in date and time for tomorrow
   Result: Annoying - hits dead end with 'find room' + calendar combination

3. Clicks 'specific'
   Fills in, checks 'projector'
   Result: Pass

4. Clicks 'specific'
   Cancels
   Clicks 'find room'
   Cancels
   Starts reading the interface
   Clicks room
   Cancel
   Gives up
   Result: Task failure - Subject not able to complete assigned task (calendar + room click combo would have solved)

5. Add-on skipped due to task failure

6. Instantly starts checking top menu
   Clicks reservations
   cancels booking
   Result: Pass - subject is used to convention about menu bar

7. Clicks aud1
   clicks calendar
   Result: pass - 'date' must have been keyword that made him think of the calendar again.

Round 3

The following scenarios were presented to the test subjects:

1. You and your group have met, and intend to conduct group-work without being disturbed. (Task: T1.1)

2. Your group has finished work today, and decided to reconvene tomorrow. You have been assigned to arrange for a place to work. (Task: T1.1)

3. Your group is done with the assignment and wish to practice your powerpoint presentation using a projector in the near future. (Task: T1.1)

4. While working in (room which was just found) you have been interrupted by another group asking when the room is next available. (Task: T1.2)

5. Add-on: They refuse the first time you suggested as they have gone home by then/don't work on that day/have vacation that week. (Task: T1.2)

6. Group member calls in sick, and work tomorrow is cancelled. (Task: T1.3, T1.4 by association)

7. You have been tasked with gathering all the first-year students for a surprise announcement in Auditorium 1. Date is not set. (T1.1)

Test subject 5, "average" outsider2:

1. Presses 'instant booking'
   Accepts offered room
   Result: Pass - no remarks

2. Clicks tomorrow on calendar
   Presses 'instant booking' and accepts
   Result: Pass - seems to have grasped how 'instant booking' works right from the get go

3. Clicks 'advanced'
   Fills in, and selects projector
   Result: Pass - no remarks

4. Clicks assigned room on map
   Uses week overview to provide options
   Result: Pass - test subject comments that the week overview is very nice

5. Clicks reservations in top menu
   Selects bookign and cancels it
   Result: Pass

6. Click aud1
   Suggests day next week
   Add-on: Evaluator declines and says that this month won't work
   Browses a month forward on calendar and clicks a random day
   Suggest new day/time
   Result: Pass + add-on - Test subject performed extrordinarily well.

Test subject 6, novice outsider

1. Presses 'instant booking' Accepts
   Result: Pass - no remarks

2. Hesitates, clicks calendar tomorrow
   clicks instant and accepts
   Result: Pass - hesitation for a few seconds, dismissed as unfamiliarity with test format

3. Clicks 'advanced'
   Fills out everything, selects projector
   selects and books
   Result: Pass - subject may not have noticed that most things can be left out

4. Clicks 'reservations'
   Informs when his booking ends
   Evaluator declines - today is no good
   Returns, click 'instant'
   quickly realisizes mistake, cancels
   clicks room on map
   provides answer from overview
   Result: Minor problem - Used reservation screen initially

5. Clicks 'reservations'
   cancels booking
   Result: Pass - no remarks

6. Clicks advanced
   fills out, checks off 'auditorium'
   finds aud1 and books
   Evaluator informs that today is no good
   Subject is puzzled - clicks 'instant', cancels
   Clicks calendar then 'instant', cancels
   Remembers map, clicks aud1
   suggests new date
   Result: Medium problem - Didn't realise map was usable for this, although it had been clicked before. Solved by muddling through

A.3.2 Semester planning

Both rounds of testing for semester planning was done directly following eachother with design changes being made on the fly while test subject wass interviewed about other parts of the system.

Round 1: Semester planning

Purpose of test

The purpose of this test is to find out whether our UI fills the following criteria:

1. Contains enough functionality to support the full task of assigning rooms to courses for a semester

2. Is designed so that the target-user will be able to fulfill the task with little or no instructions

More specifically, the points we need to address is:

3. Is our attempt at 'clash resolution' satisfactory?

4. Is the 'track management' efficient enough for the problem at hand?

5. Will the target-user protest against the manual entering of courses etc, even though there is possibilty for re-use over the course of years?

Tasks to perform

The following tasks are what we need the user to perform:

The annotations in parenthesis after are the task from the comprehensive task list they refer to.

———- Initiation tasks ———-

I1. Add a course (T2.1)

I2. Add a course that cannot be moved by auto-assignment (T2.1)

I3. Remove the restriction from the course added by I1/I2 (T2.3)

I4. De-active a course that is already created (T2.3)

I5. Create a new track (T2.2)

I6. Edit a track (add a course and remove another one eg.) (T2.4)

———- Main tasks ———

M1. Plan a semester (no clash happens) (T2.5)

M2. Plan a semester (clash happens) (T2.5)

M2.1. Resolve the clash automatically (T2.5)

M2.2. Resolve the clash manually (T2.5)

———- Rare task ———

R1. Clear the room assignment to make next semester ready

Test course

I1
I2
I4
I5
I6
M2
M2.1
M2.2
I3
M1
R1

Lead-ins

I1 - Add a course

We already added a few courses to the system, but forgot one. Please add BTES

[TEST]—[BTES]
[Week 30]—[Week 49]
[4 hours]—[ ]
[42]—[ ]
[ ]—[ ]

Correct solution:
- 'Add new course' tab
- fill in
- 'Save'

I2 - Add a course that cannot be moved by auto-assignment

Another course has been forgot, this one needs to be in AUD4 for some wierd reason

[AUD]—[BAUD]
[Week 30]—[Week 49]
[4 hours]—[ ]
[42]—[ ]
[AUD4]—[X]

Correct solution:
- 'Add new course' tab
- fill in
- 'Save'

I4 - De-active a course that is already created

Oops, turns out that BTES didn't get enough students this semester and have been put on hold this semester. Lets not delete it, but make sure it daasn't get a room.

Correct solution:

- 'Edit course' tab
- Click BTES
- Click Edit
- Check 'Inactive box'
- 'Save'

I5 - Create a new track

BPRD and BSDB are on the same track, and should therefore not be planned at the same time. Lets add the track.

Correct solution:

- 'Manage tracks' tab
- fill in name
- Select courses using the arrows
- 'Save'

I6 - Edit a track (add a course and remove another one eg.)

DMD has taken a new direction and now needs more programming. Thus, BPRD have been added to the first semester. Fix it!

Correct solution:

- 'Manage tracks' tab
- Click 'DMD - sem 1' in the list
- Click 'Edit
- Select courses using the arrows
- 'Save'

M2 - Plan a semester (clash happens)

Alright, we're all set. Lets assign the rooms!

Correct solution:

- Any tab
- Click 'Auto-assign'
- BAM CLASH

Depending on initial reaction, go to either m2.1 or m2.2

M2.1 - Resolve the clash automatically

Fix it!

Correct solution:

- Accept one of the solutions

M2.2 - Resolve the clash manually

Fix it!

Correct solution:

- Switch tab to edit course

- Edit shit until it works
- try again

I3 - Remove the restriction from the course added by I2/I1

The course BAUD have lost it's privileges and can no longer claim AUD4

Correct solution:
- 'Edit course' tab
- Click BAUD
- Click Edit
- Check 'required off' / Clear room box
- 'Save'

M1 - Plan a semester (no clash happens)

We've fixed all the stuff that could clash, lets do it!

Correct solution:
- 'Auto-assign'
- ???
- Profit

R1 - Clear the room assignment to make next semester ready

This semester was a blast, lets try again.

Correct solution:
- 'Room assignment' tab
- Click the danger button
- accept the insane warning (possible text input)

RESULTS OF USABILITY TEST

USER Malene de Bruin

Studieadministration

Malene is head of semester planning. At the start of each semester, she plans and coordinates the rooms for the various courses, using tabulex and a homemade spreadsheet.

She is thus the exact person that will be using the semester planning part of our application.

———- Initiation tasks ———-

I1. Add a course

Skriver kursus, trykker paa start week. Vaelger 30. Er i tvivl om antal uger (?). Vaelger duration. "NÅÅÅÅH, gaar altid ned af og ikke hen af.

"Starttidspunkt, ville jeg umiddelbart lade staa blankt. Ville det vaere tydeligt at systemet vaelger det for mig. Tror jeg ville vaelge klokken 8, uden at vide hvorfor!".

"Weekdays, not selected. Der er ikke noget room-request. Ville proeve at klikke paa room required, da der er en checkbox". Forstaar meningen.

I2. Add a course that cannot be moved by auto-assignment

Skriver navnet og kode, vaelger det samme som foer. "Jeg er meget i tvivl om optional, og hvad der sker naar jeg ikke vaelger det". Vaelger required og gemmer.

I3. Remove the restriction from the course added by T2/T1

Trykker paa edit course. Vaelger kurset fra listen, trykker edit. Fjerner required - save.

I4. De-active a course that is already created

Trykker paa edit courses. "Ahhh, jeg skal fremkalde BTES". Vaelger i listen, og trykker edit. Vaelger inactive, save.

I5. Create a new track

Klikker paa manage track. Flytter dem til added, og skriver et navn. Trykker paa save.

I6. Edit a track (add a course and remove another one eg.)

Markerer track, trykker paa edit. Markere og flytter, save.

———- Main tasks ———

M1. Plan a semester (no clash happens)

M2. Plan a semester (clash happens)

"Det jeg ikke kan se er hvad problemet er. Ellers ret god loesning at den foreslaar alternativer. Hvad saa hvis man ikke accepterer?. Jeg vil gerne selv goere det manuelt, men det kan jeg ikke. AHHHH, det er da klart". (Efter Bird har vist hende firkanten med tekst).

M2.1. Resolve the clash automatically

M2.2. Resolve the clash manually

———- Rare task ———

R1. Clear the room assignment to make next semester ready

"Ville have det daarligt med at slette alle kurserne een for een. Ville hellere goere dem inaktive; ville foeles bedre hvis de ligger der, og ikke bliver slettet".

Finder clear plan efter Bird har uddybet. Finder knappen. "Det er meget rarere at knappen er der, clearer og ikke sletter".

A.4 Study Administration interviews

A.4.1 Interview during testing

"Kan man sortere, kan man printe, kan det blive til en plan. Det ville være nice at kunne en masse ting herfra. Vi printer tit rum og til os selv, men også til andre. Sorterer næsten altid efter linie -¿ tracks. Sorterer til efter udbyder linie".

"Også helt oplagt at sortere på lokale, så man kan se om der er huller".

- Der skal være mulighed for efterfølgende at ændre ting. Evt. få et "uge view", og kunne flytte lidt rundt som man har lyst til.

- Malene ville gerne kunne se de forskellige clashes visuelt. "For mig er det noget med noget visuelt, så man kan se problemet".

- Understøttelse af at kurser ikke følger lige efter hinanden.

- Man skal kunne godkende clashes. Dette skal vise en liste over kurser der bliver påvirket af dette.

- Automatic i stedet for optional.

A.4.2 Interview 16-03-11

Møde med Malene de Bruin fra studieadministrationen.

Undervisnerne booker ikke selv lokaler til undervisning, men kan afgive ønsker til studieadministrationen, der står for planlægningen.

System låst (så elever ikke kan booke) når der laves lokaleplanlægning.

Ingen begrænsninger på hvad elever må booke (undtagen måske auditorie 1 og 2), eller hvor mange rum de må booke af gangen, men alle bookinger skal godkendes.

3 instanser: FM står for "event" planlægning. Skal koordineres med studieadministrationen. Studieadministrationen står for skemaplanlægning i tabulex. "John Arne" står for at overføre data fra tabulex til excel-ark, som giver et centralt overblik over lokaleallokering. Tabulex er decentraliseret; det ligger kun på én computer. John Arne står også for ad-hoc lokale-booking og eksamensplanlægning.

Lige nu er der ikke særlig god lokaleudnyttelse: Mange små rum, der ikke er store nok til undervisning. Man burde kunne slå mange af dem sammen. (fysisk). PC rum står tit tomme.

Vigtigt at vide hvor mange der er plads til i et lokale, og hvad der er til rådighed.

Tabulex kan tage højde for hvornår undervisningen skal foregå osv. Virker til at fungere som et godt system til overblik over kursus-aktivitet, men det er ikke et lokaleplanlægningssytem.

De største fordele ved et nyt system:

- Distribuerede booking. Fedt hvis alle kan benytte et centralt system.

- Sikkert system. Lige nu er der mange fejlkilder. Største (og værste) er dobbeltbookinger. Da de 3 instanser ikke bruger samme system, er risikoen betydende.

Ønsker:

- Mulighed for at ændre på lokale-opsætningen. "Nu er disse to (tilstødende lokaler) ét samlet, så der er plads til flere mennesker" -¿ bedre lokale-udnyttelse.

- Integration med kursusbasen.

Integration af alle 3 instanser. (Løser formentlig sig selv, evt. ved bruger-roller).

Hvad sker der hvis x booker lokale n klokken 14, mens y sidder og arbejde i det? Hvem har retten til lokalet?

A.5 Code structure

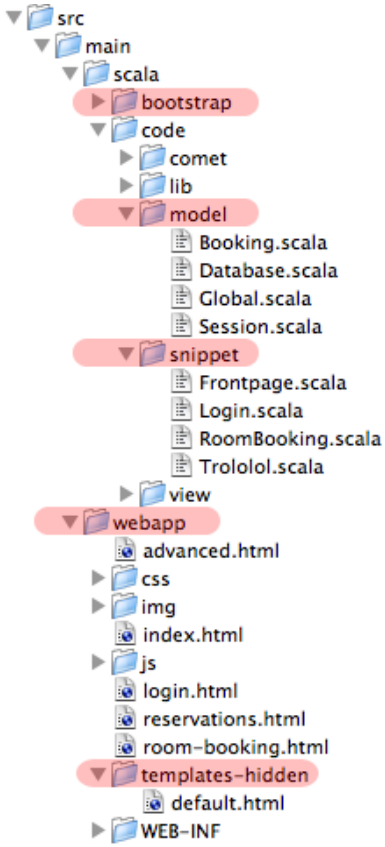

Figure A.1: The structure of a SBT Lift project

Figure A.1 shows the basic structure of a Lift project, using the template provided by SBT. The presentation layer is located in the webapp directory, containing the markup, graphics and design of the user interface. The templates-hidden folder contains a template file, which content is wrapped around all other pages. This is basically a header file, which should code (like menus) that should be repeated on each page.

The scala directory contains the server implementation. The bootstrap folder contains logic used to initialize the application, while the model and snippet folders contain the different entities of the application, and the xml bindings.

A.6 Mockups

A.6.1 Room booking

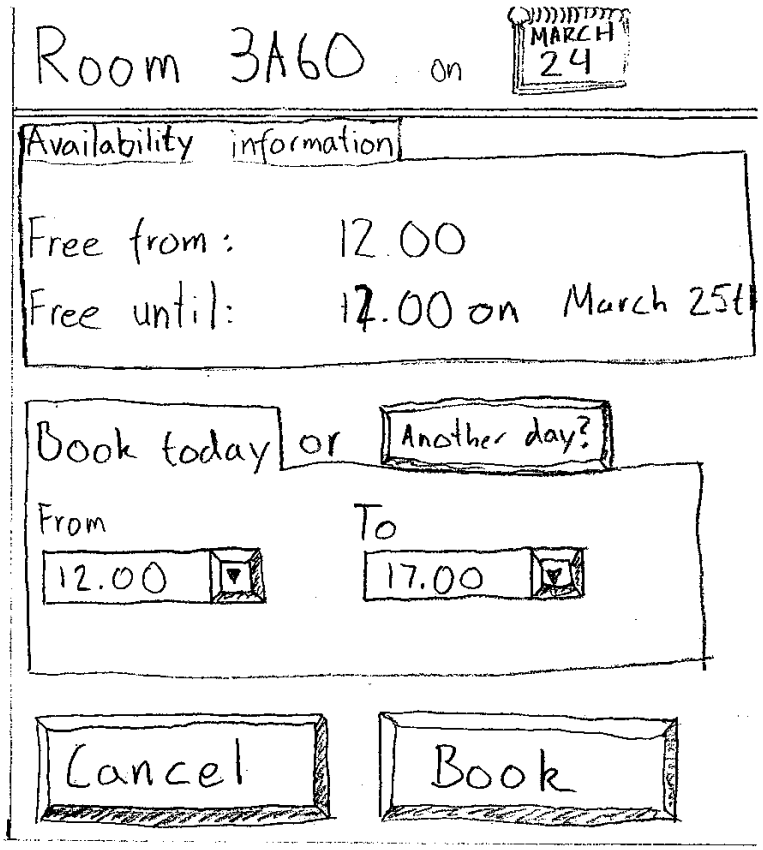

Figure A.2: First mockup for simple booking

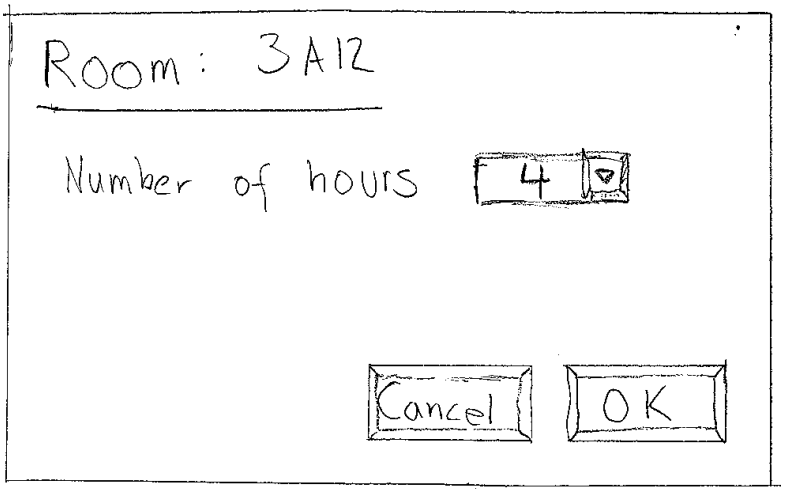

Figure A.3: Second mockup for simple booking

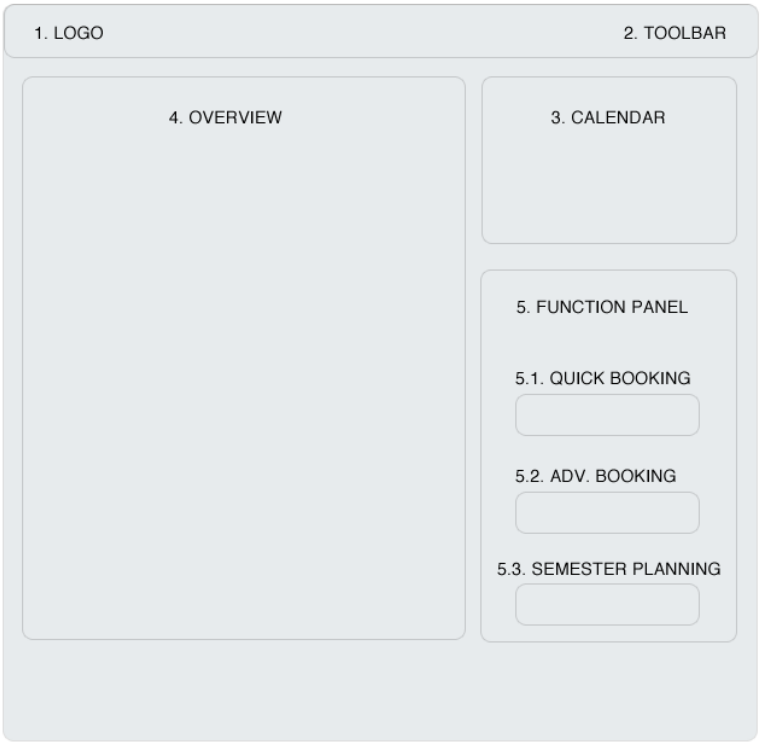

Figure A.4: Wireframe of frontpage

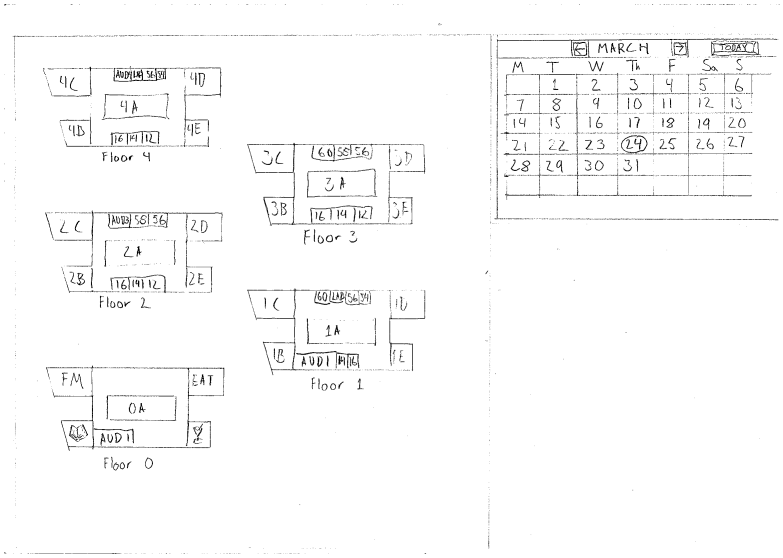

Figure A.5: Mockup of frontpage (map overview)

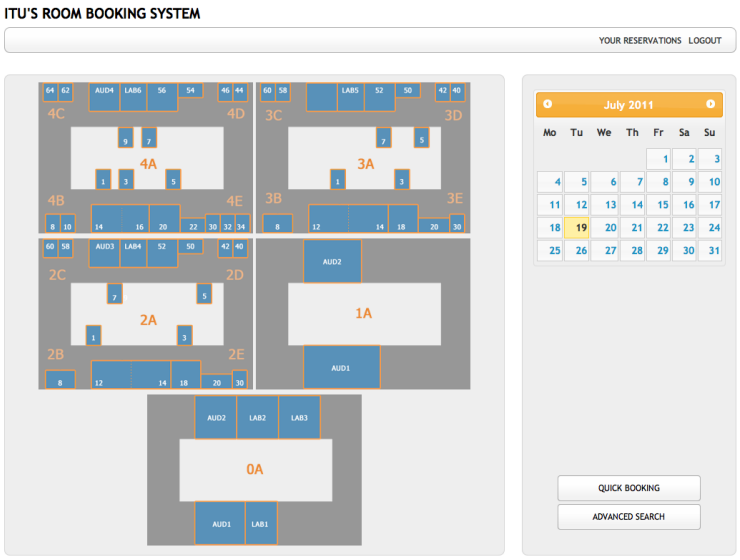

Figure A.6: Final implemented version of the frontpage

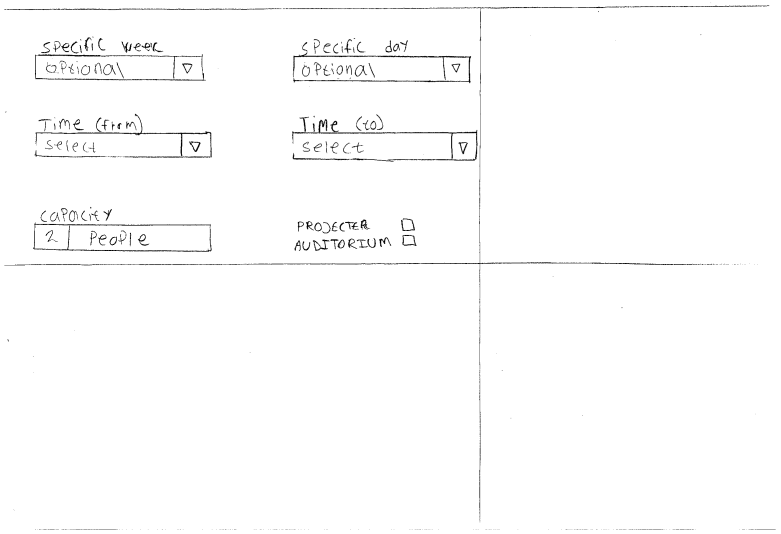

Figure A.7: Sliced mockup of advanced booking, with placeholders for overlays

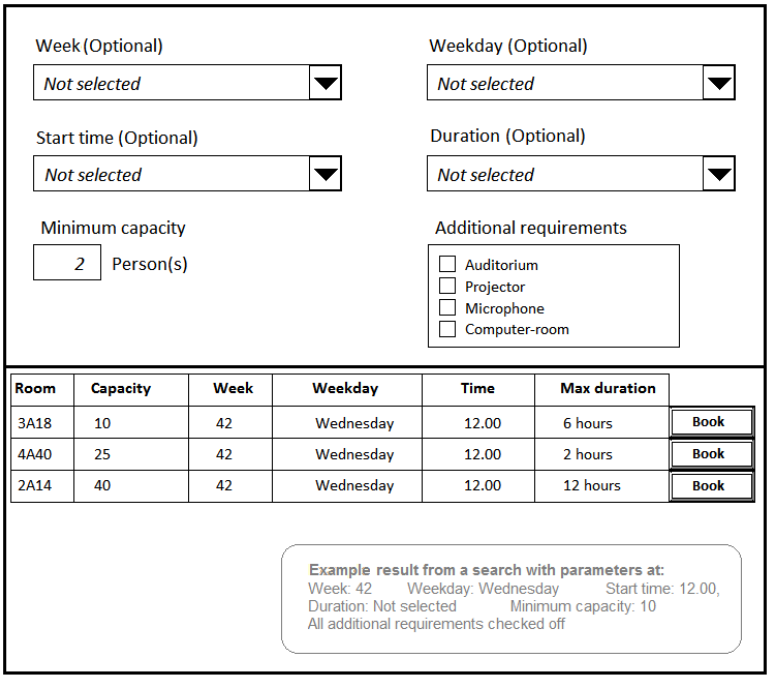

Figure A.8: Mockup of search result for advanced booking

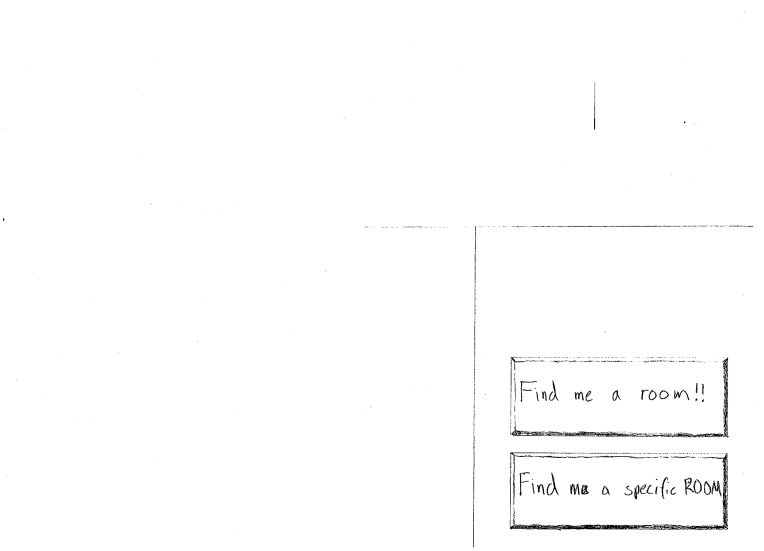

Figure A.9: Buttons for test of first mockup

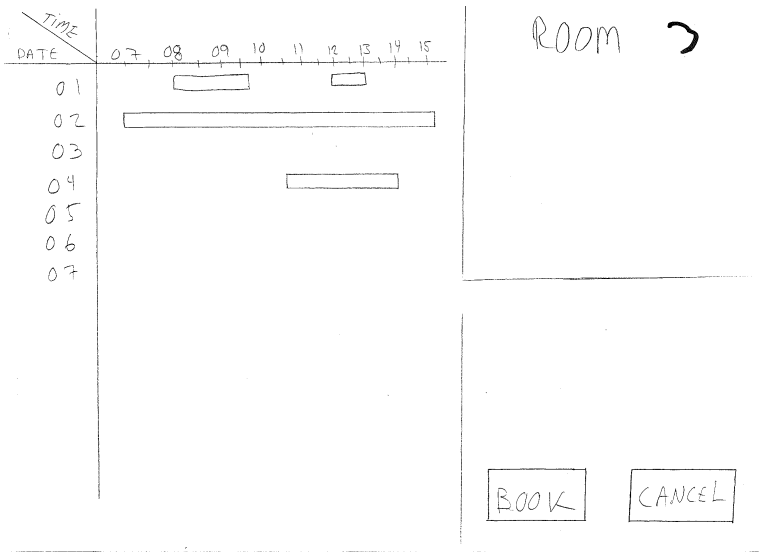

Figure A.10: Mockup of the weekly booking overview of a specific room

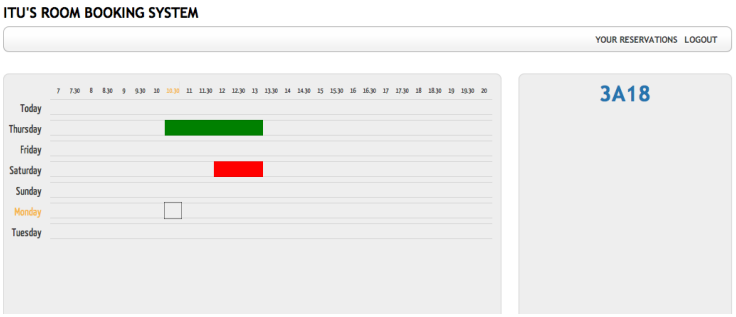

Figure A.11: Final design of the weekly booking overview

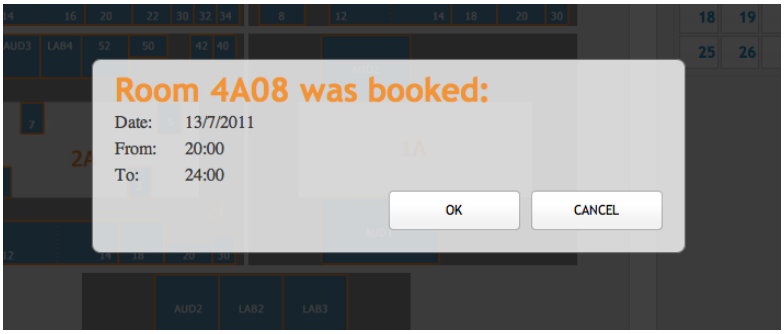

Figure A.12: Quick Booking confirmation dialog

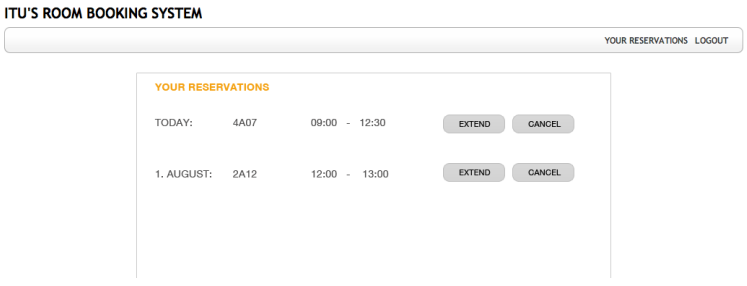

Figure A.13: Personal Reservations interface

A.6.2 Semester planning

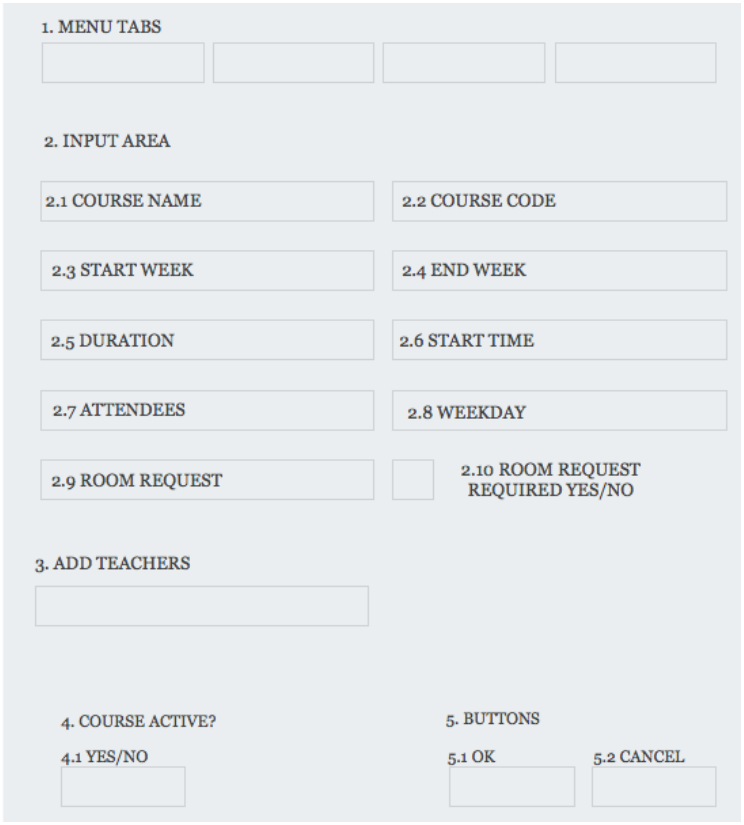

Figure A.14: Wireframe for screen to add a course to the semester plan

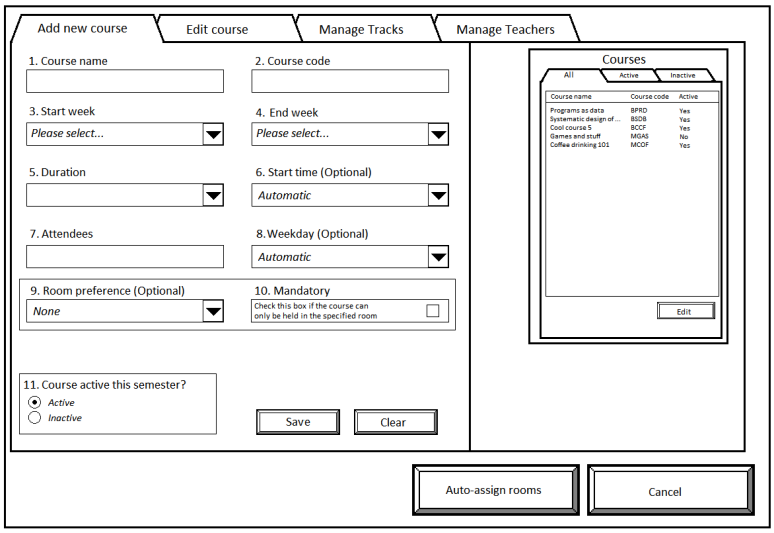

Figure A.15: Second mockup for screen to add a course to the semester plan

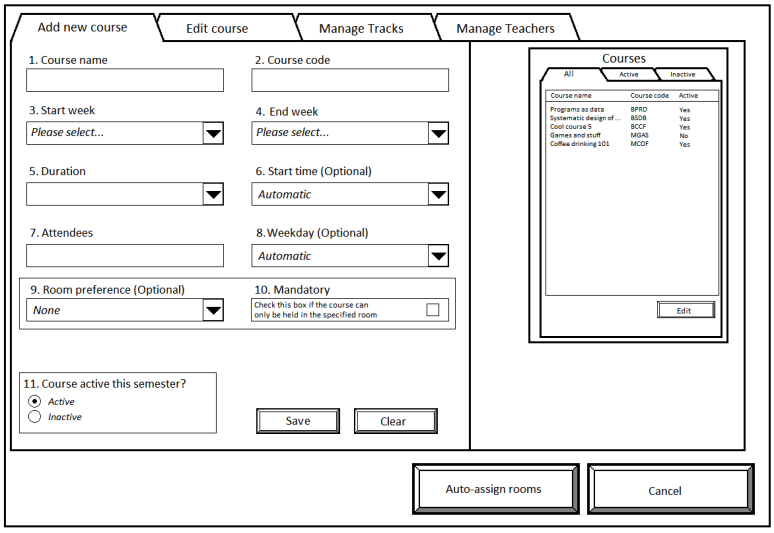

Figure A.16: Second mockup for screen to add a course to the semester plan

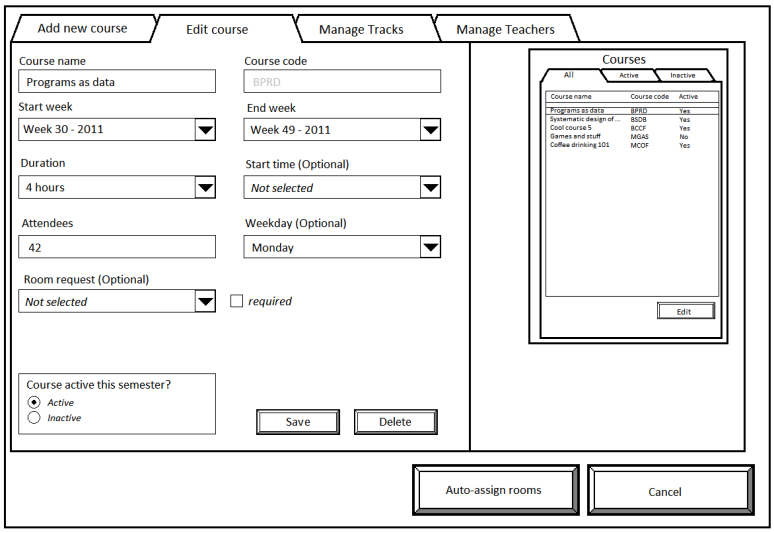

Figure A.17: First mockup for screen to edit a course to the semester plan

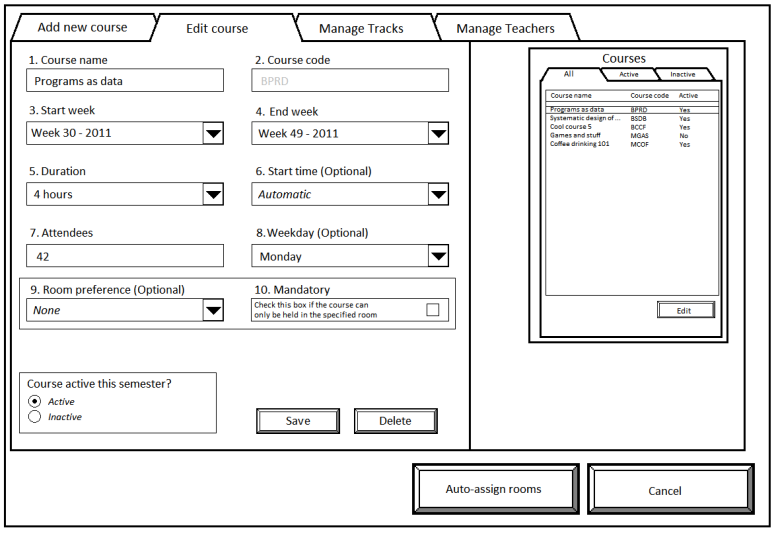

Figure A.18: Second mockup for screen to edit a course to the semester plan

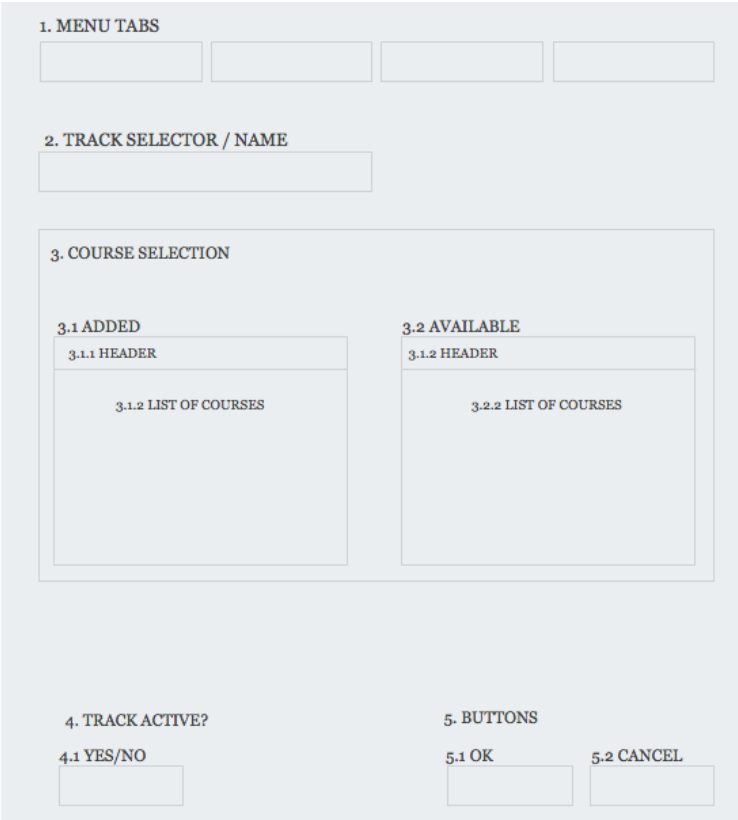

Figure A.19: Wireframe for screen to assign courses to tracks. (manage tracks)

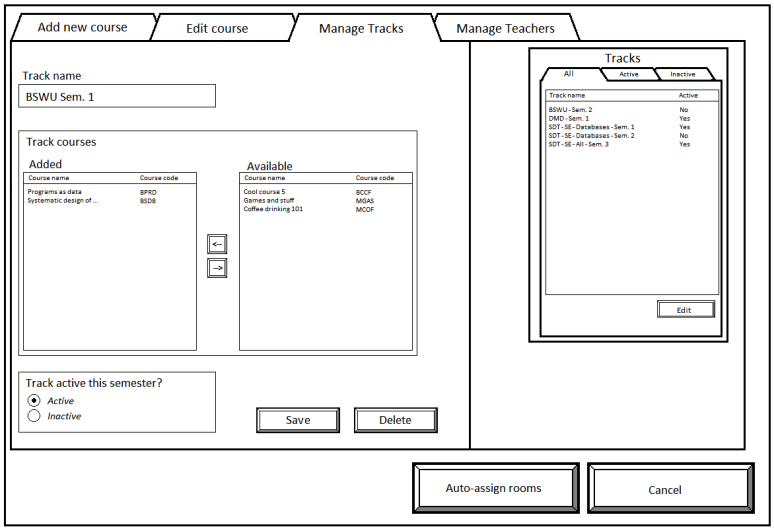

Figure A.20: First mockup for screen to assign courses to tracks. (manage tracks)

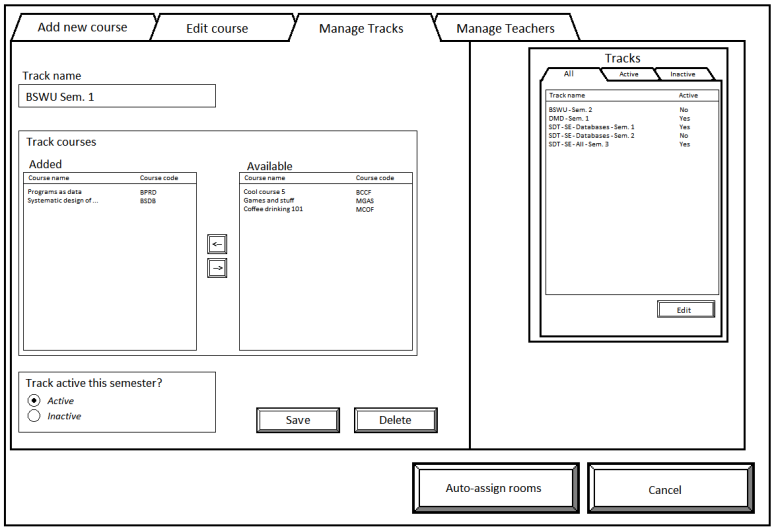

Figure A.21: Second mockup for screen to assign courses to tracks. (manage tracks)

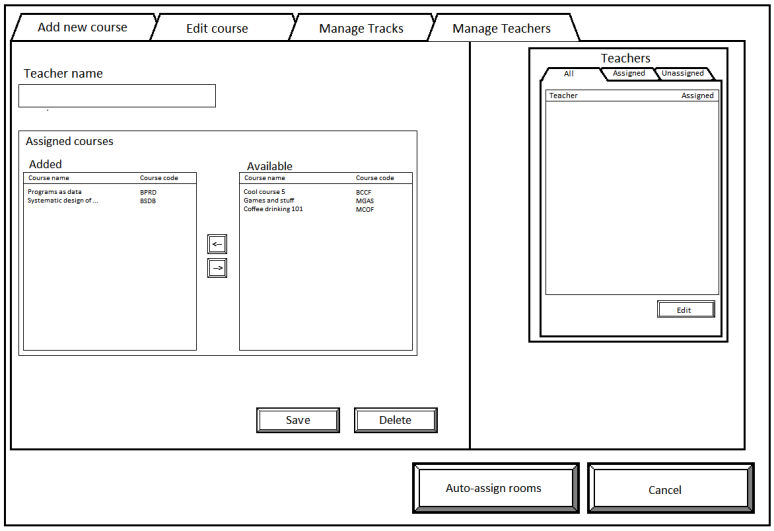

Figure A.22: First mockup for screen to assign teachers to courses. (manage teachers)

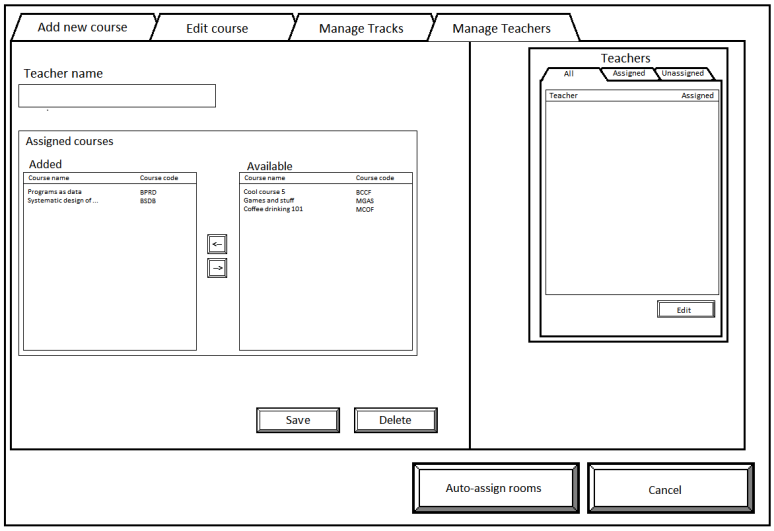

Figure A.23: Second mockup for screen to assign teachers to courses. (manage teachers)

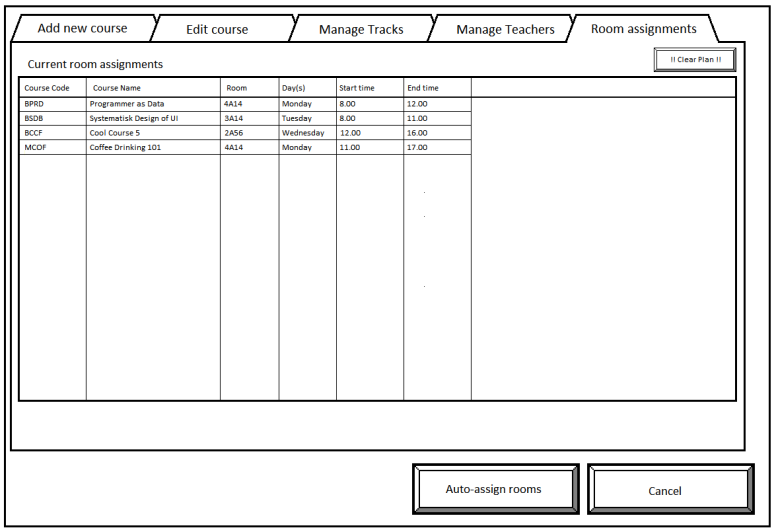

Figure A.24: First mockup for the screen showing the finished semester plan

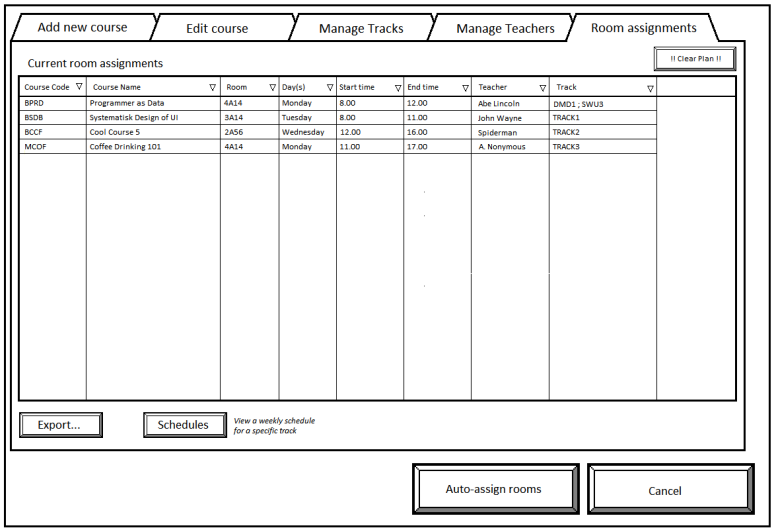

Figure A.25: Second mockup for the screen showing the finished semester plan

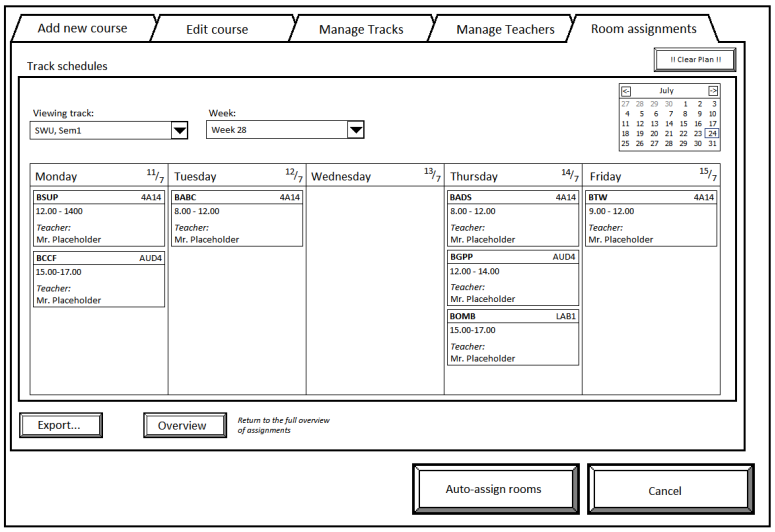

Figure A.26: Mockup for the weekly schedule overview

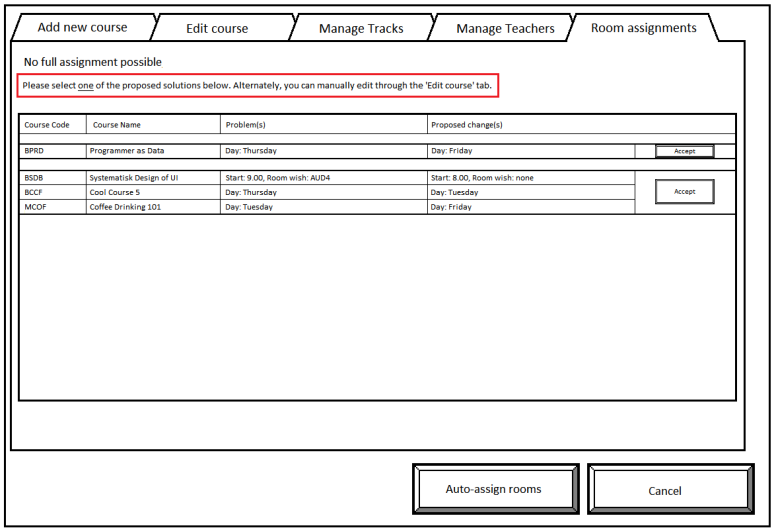

Figure A.27: First mockup for the screen showing the finished semester plan, where some complications are present

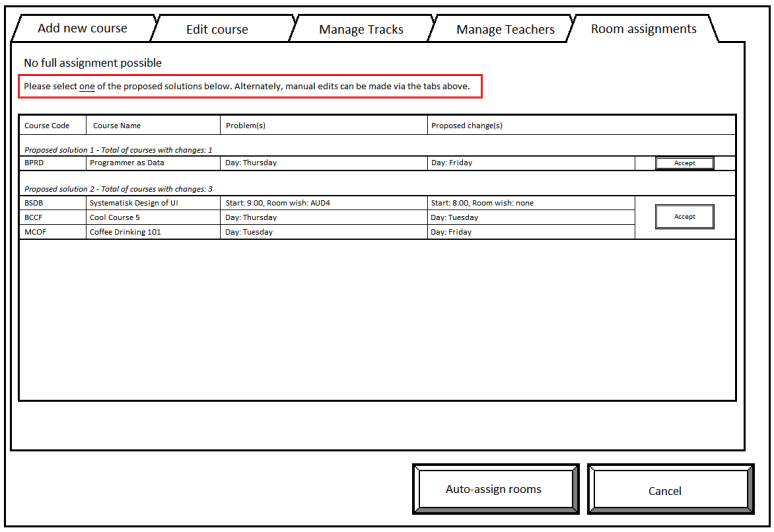

Figure A.28: Second mockup for the screen showing the finished semester plan, where some complications are present

A.6.3 Whiteboard sketches

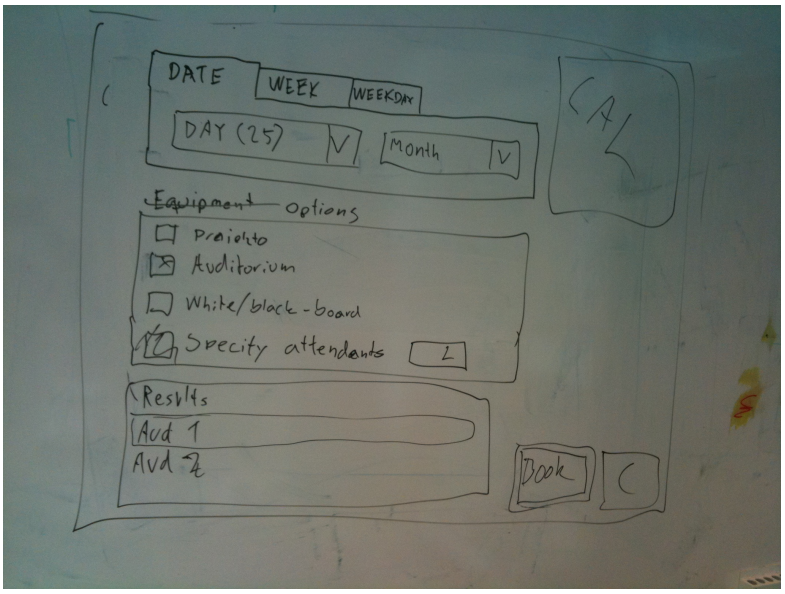

Figure A.29: Simple booking sketch

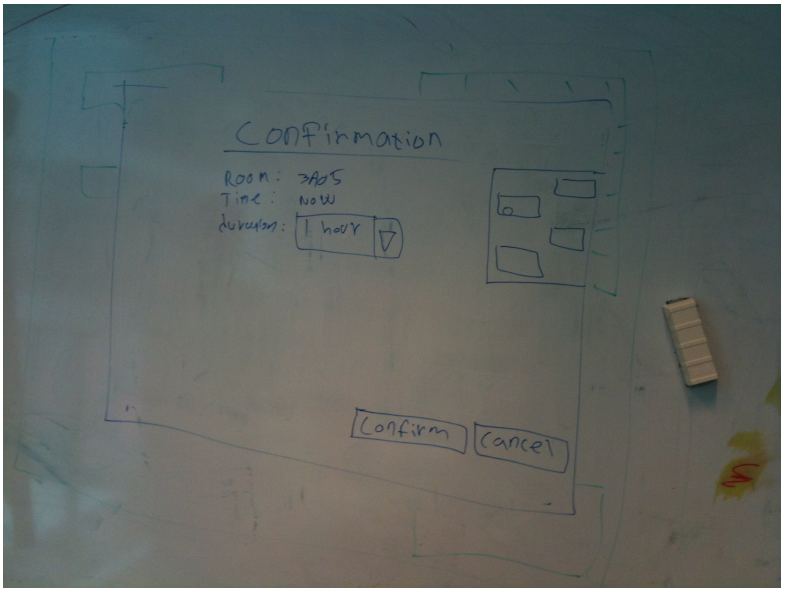

Figure A.30: Early confirmation sketch for simple booking

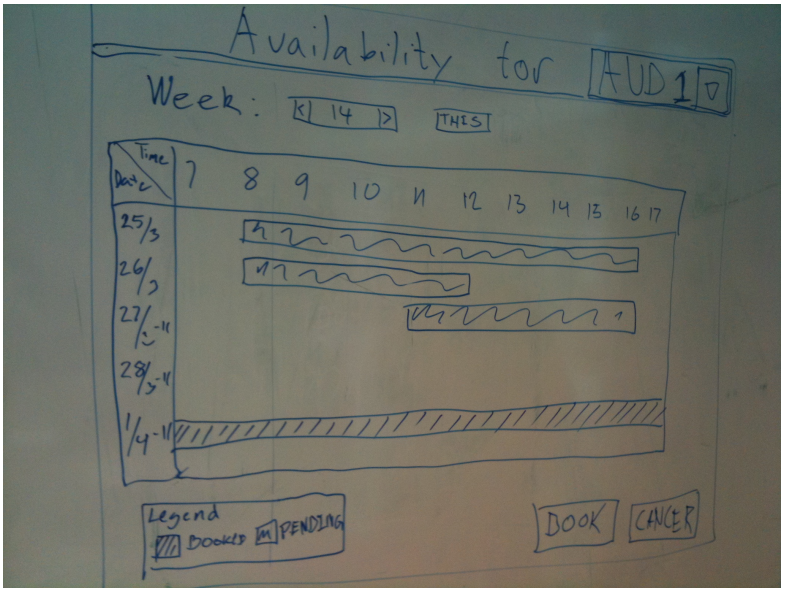

Figure A.31: Sketch for availability grid for a specific room

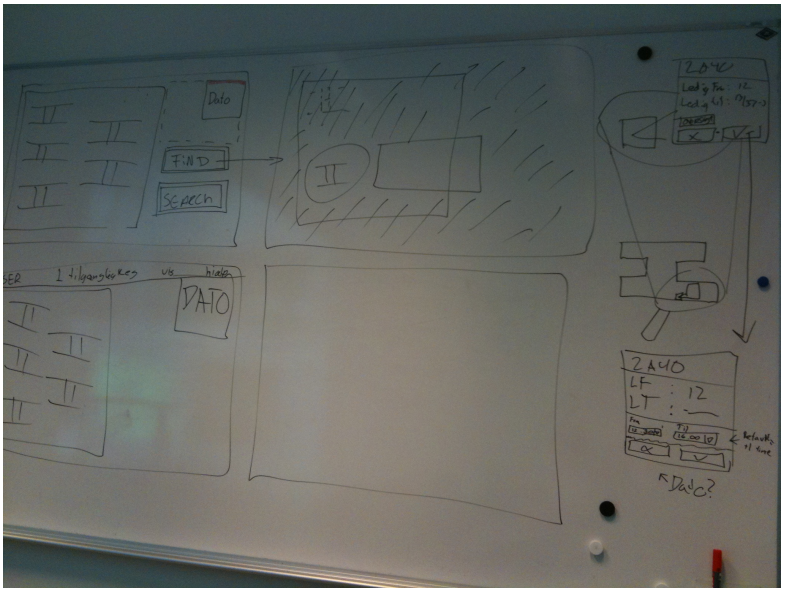

Figure A.32: Concept sketch

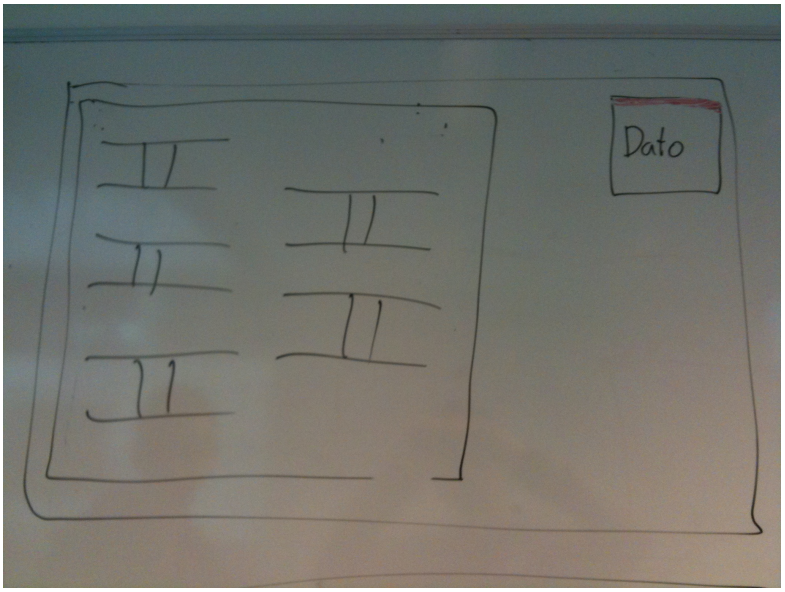

Figure A.33: First concept sketch of map overview

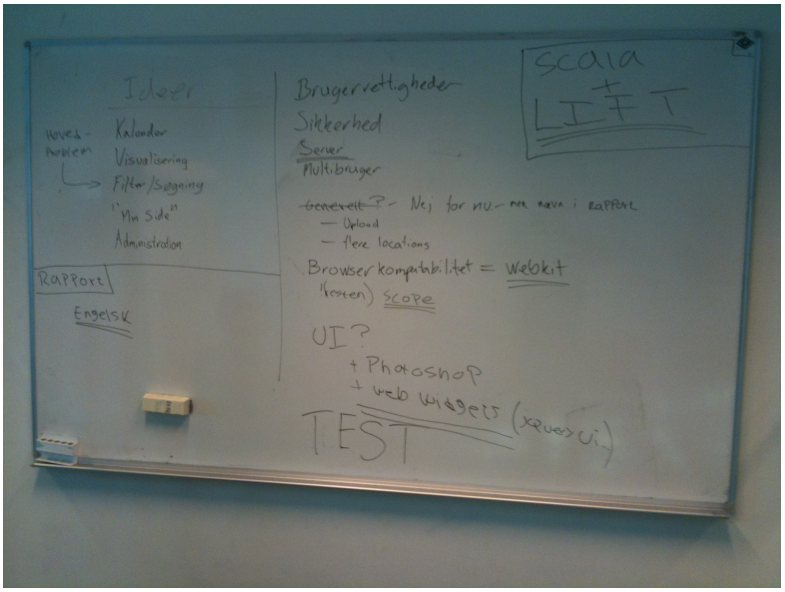

Figure A.34: Brainstorm sketch

A.7 Glossary

- **Advanced booking**
  The process of using our advanced booking interface to book a room, specifying requirements which the room should fulfill.
- **Booking**
  The entity that represents a room reserved by a user, at a specific point in time.
- **Clash**
  When one or more constraints conflict, making it impossible for the semester planning algorithm to create a full semester plan.
- **Constraint**
  A rule that always should be satisfied by the semester planning algorithm. A constraint can be built into the algorithm, or be passed as an input. An example of a built constraint, is the rule that no courses should be held in the same room at the same time.
- **Client**
  An application or system that accesses a remote service on another computer. In our case, the clients are the browsers used by our users, rendering the html pages of our web application, and showing the information gathered from the server.
- **Day to day booking**
  The part of our application and interface concerned with the booking functionality, where members of ITU can reserve a room for a period of 4 hours.
- **Evaluator**
  The person executing a think aloud test with a test user.
- **Gestalts**
  Design laws, describing how we perceive relationships between geometric elements.
- **Javascript**
  A scripting language used to manipulate the client side of our application.
- **Lift**
  An Open Source web application for Lift, used for porting a Scala implemention to an actual working web application.

- **Mockup**
  A quickly created layout and interface of a single part of an application, used to quickly test a proposed design.

- **Prototype**
  A partly implemented application, used to test functionality and/or design.

- **Quick booking**
  The simplest way of booking a room with our application. Day to day booking is the process of clicking the "Quick Booking" button, which books any room at ITU, which is available at the current time for a period of 4 hours.

- **SBT**
  Simple Build Tool. Used to build and compile a Lift application. SBT comes with a local web server which can be used to run a fully functional local copy of a lift application.

- **Scala**
  The server-side programming language used through out the project. Scala is based on the Java Virtual Machine, making it object oriented, but with certain features taken from functional programming.

- **Semester planning**
  The part of our application concerned with the process of creating a plan for a semester; creating courses and assigning rooms to them.

- **Server**
  The server provides the service called by the client. In our application, the server handles the bookings and users of our application.

- **Session**
  One session starts when a user succesfully signs in to our application. It ends when they log out, or close the browser.

- **Simple booking**
  Covers the process of booking a room using the map on the frontpage, then selecting a day of the week and a timespan in which the room should be reserved.

- **Tacit knowledge**
  Knowledge that is difficult to transfer to another person by means of

writing or verbal communication. In our project, the persons residing in the domain have tacit knowledge, which we should take into consideration when designing the interfaces.

- **Test subject/user**
  The person that performs the tests in a think aloud test, controlled by an evaluator.

- **Think aloud test**
  A method of usability testing. Involves test users performing specific tasks, while explaining what and why they do what they do.

- **Usability / User-friendly**
  Usability is the ease of use and learnability of human made objects. Objects with high usability are easy to use, easy to learn, intuitive and user friendly.

- **User**
  In our application, a user is someone cabable of using our applications. This means all persons registrered in the userbase of ITU.

- **User experience**
  A good user experience for a user, means that the user liked a specific product or service. User experience is subjective, and there is not necessarily a connection between user experience and usability.

- **Wireframe**
  A drawn mockup, with focus on the data that the particular interface should include. Used to evaluate a design idea with actual data.