

Develop of a proof of concept for the utilization of data for an optimal and controllable indoor environment at the DTU Library Living Lab

Jørgen Waarsøe Falch
s103801

DTU



Kongens Lyngby 2016

Technical University of Denmark
Department of Civil Engineering
Brovej, building 118,
2800 Kongens Lyngby, Denmark
Phone +45 4525 1700
byg@byg.dtu.dk
www.byg.dtu.dk

Summary (English)

The library at DTU has a vision of improving the indoor climate by making the library more intelligent and by turning the entire library into a living lab. Researchers and students should be able to use the living lab to conduct experiments that can lead to a improved indoor climate in the library. The first phase in turning the library into a living lab involves removing all the lamps and replacing them with LEDs. While replacing the lamps the library will also install sensors that can be used as a part of the living lab. The DTU library is now looking for ideas to how the new lighting system and sensors can be an integrated part of the living lab.

This thesis is about developing a software system, which can help the living lab improve to the indoor climate, by taking advantage of the new LED lamps in the library. The system developed consists of an automation system and a smartphone app. The automation system can be configured by users of the living lab users to take input from sensors and adjust the lighting accordingly. The smartphone app is used by the library guests to adjust the lighting decided by the automation system.

The tests made showed that the software is working as intended and that it can be a useful tool for conducting experiments. However the initial tests show that an automated lighting system can be very distracting. Therefore it is important that users can override the automation system.

Summary (Danish)

Biblioteket på DTU har en vision om at forbedre indeklimaet, ved at gøre biblioteket mere "intelligent" og transformere hele biblioteket til et living lab. Forskere og studerende skal kunne bruge living lab til at foretage eksperimenter, der eventuelt ville kunne føre til et forbedret indeklima. Den første fase vil forvandle biblioteket til et living lab og består i at erstatte de eksisterende lamper med LED belysning. I samme proces bliver der også installeret en række sensorer, som skal indgå som en del af living lab. DTU søger nu ideer til hvordan sensorer og LED belysning kan blive en integreret del af living lab.

Dette kandidat projekt handler om udviklingen af et software system til brug i det nye living lab. Det udviklede system kan hjælpe living lab med at foretage eksperimenter, ved at udnytte de muligheder LED belysning giver. Systemet består af et automationssystem og en android applikation. Automationssystemet kan konfigureres af brugerne af living lab-brugerne til at justere lyset på baggrund af input fra forskellige sensorer. Android applikationen bruges af bibliotekets gæster til at justere den automatiske lyssætning. De test der er blevet foretaget viser at den udviklede software virker efter hensigten, og kan være et nyttigt redskab til at foretage eksperimenter. En test viste dog også at automationssystemet kan være særdels forstyrrende. Det er derfor vigtigt at brugerne kan tilsidesætte automationssystemet ved manuel betjening.

Preface

This thesis was prepared at DTU Byg in fulfilment of the requirements for acquiring a M.Sc. in Engineering under guidance of Alfred Heller. This project accounts for a total of 30 ECTS points.

The thesis deals with developing a system, that can help researchers and students at DTU to conduct experiments regarding the effects of dynamic lighting on the indoor climate.

The code for the controller application and the web server can be found on github:

Controller app: <https://github.com/lakerolis/LightMeter.git>

Web server: https://github.com/lakerolis/Dynamic_lighting.git

The thesis consists of the following chapters:

Chapter 1: Introduction

Chapter 2: Background

Chapter 3: Proposal for System Requirements

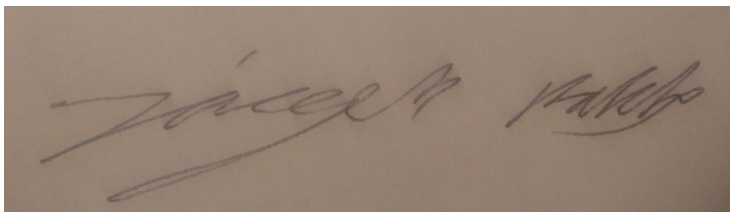
Chapter 4: Design of the system

Chapter 5: Implementation

Chapter 6: Testing

Chapter 7: Discussion and Conclusion

Lyngby, 21-December-2016

A rectangular image showing a handwritten signature in dark ink on a light-colored background. The signature is written in a cursive style and appears to read "Jørgen Waarsøe Falch".

Jørgen Waarsøe Falch

Contents

Summary (English)	i
Summary (Danish)	iii
Preface	v
1 Introduction	1
2 Background	3
2.1 DTU Case and Motivation	3
2.1.1 Smart library Pilot project	4
2.1.2 Living Lab	6
2.2 Project Description	7
2.2.1 Problem Formulation	7
2.3 Dynamic Lighting	8
2.3.1 Benefits of Dynamic lighting	9
2.4 Existing Technologies	10
2.4.1 Phillips Hue	10
2.4.2 Lightify Home	10
2.4.3 Lightify Pro	11
2.4.4 Samsung SmartThings	12
2.4.5 ITTT	13
2.4.6 Summary of Existing technologies	13
3 Proposal for System Requirements	15
3.1 Users	16
3.2 Sequence diagrams	18
3.2.1 User Story Mapping	19

4	Design of the system	23
4.1	Web Server	24
4.1.1	MVC	24
4.1.2	Communication	27
4.1.3	Automation System	28
4.2	Controller Application	29
4.3	Database	30
4.4	Security	31
4.5	Sensor Application	32
5	Implementation And Limitations	35
5.1	Hardware setup	36
5.2	Back-end	37
5.2.1	Web server	37
5.2.2	Automation system	40
5.2.3	Database	43
5.3	Front-end	45
5.3.1	Controller App	45
5.3.2	Sensor App	47
6	Testing	49
6.1	Black-box testing	49
6.2	User interface	51
6.3	Testing the system in a work situation	52
6.3.1	Test 1 - Testing the automation system	53
6.3.2	Test 2 - Testing the controller app	55
7	Discussion and Conclusion	57
7.0.1	Future work	58
A	System Sequence Diagrams	61
B	Administration Web Page	65
B.1	Index	66
B.2	Rules	67
B.3	Sensors	68
B.4	Aactions	69
B.5	Conditions	71
	Bibliography	73

Introduction

The European Intelligent Building Group (EIBG) defines an intelligent building as a building that “creates an environment which maximizes the effectiveness of the building’s occupants while at the same time enabling efficient management of resources with minimum life-time costs of hardware and facilities”[Pel]. The market for Intelligent buildings or some times referred to as smart buildings (smart homes if it is a residence), has been increasing rapidly and will continue to do so over the next decade[Mar][mem16]. This can be explained by a number of reasons. A report created by Memoori[mem16] mention a significantly decline in the cost of sensors and bandwidth. At the moment there is a trend for businesses to be green[You10] and making a building intelligent can greatly reduce the CO2 emissions[Sin07]. Being green is not the only motivation, there is also money to be saved on operating costs by making a building more intelligent[Fer][Ked]. The ability to automatically control the heating, ventilation and air conditioning (HVAC) and the lighting, based on people presence, can reduce the energy consumption. An intelligent building can increase the comfort level of those using it or decrease it if it is done wrong[SW01]. An intelligent building is not only about automation but also about control. Depending on the tasks the users are performing they require different settings like heating or lighting. By letting the users being able to control these settings and thereby personalising the building, the building will to a greater extend meet the requirements of the users. Now DTU library wants to be part of the trend and will in the coming years turn the the old library into a "Smart Li-

brary". "Smart library" is a term used by DTU library to describe the project of making the library intelligent. The library hopes that an intelligent library can improve the indoor climate. As part of "smart library" the DTU library wants to create a so called living lab. The goal is that researchers and students can use the living lab to conduct experiments. Lighting is already automated in many public buildings. With the use of PIR-sensors, short for passive infrared sensor, the lighting is turned on if movement is detected and it turns off if no movement has been detected in a given period of time. Compact fluorescent lamps (CFL) get a reduced life span when turned on and off frequently[doE]. LED on the other hand does not suffer such loss[doE]. Adjusting the brightness can help to improve the indoor climate. However dimming the light can also have a negative effect on the lamps' life span unless using LED's. This makes it possible to make changes to the lighting more frequently and thereby save energy, without shortening the life span of the lamps.

This thesis is about creating a platform that can control the lighting at the library, and help the living lab to conduct experiments on how lighting effects the indoor climate at the library. First there will be a review of existing technologies and possibilities regarding the use of dynamic lighting. A prototype will be created that can be used by the living lab to conduct lighting experiments and possibly improve the indoor climate at the library.

Background

This chapter will describe the DTU case and the motivation for this thesis. The chapter contains a description of the planned living lab and the changes that is going to be made to the library. The chapter also contains the problem definition and the limitations of the thesis.

2.1 DTU Case and Motivation

When the library at DTU was built, it was built for housing books. Since then all the books have been moved to the basement and the main activity in the library is studying. The problem is that the lighting fixtures have never been replaced. Currently the lighting in the library is made for housing books and not for people studying. During the year 2017 DTU Campus Service (CAS) will replace all the lamps in the library. All the 620 lamps will be replaced by LED, and CAS expect significant savings on energy consumption[6]. With the new light fittings it will be possible to adjust the color temperature and intensity of the light. Replacing the lamps is the first step of the project "smart library"[dtub]. As of November 2016 There is no automation related to the lighting at the library. The lighting is controlled manually by lighting switches. Lars Binau from the office for Innovation and Sector Services at the DTU library, believes that by making

the library smart and by personalising it to its users, it will increase the indoor climate level and thereby possibly the productivity of its users. DTU library also wish to make a so called "living lab". A place both students and professors can use for creating experiments. Living lab will be covered in more detail in the coming section "Living Lab". The basic idea is to utilize the change of the light fixtures to include sensors and communication abilities. These sensors can be used to gather data about the amount of people in the library and their movement patterns, and the amount of light present. The data gathered can be used to increase the understanding of how people use the library, and as part of the living lab. The amount of sensors that can be installed depends on the foundlings granted by the building owner, CAS, a sub organization of the university. CAS is mostly concerned about building maintenance. For the operators LED lighting gives cost savings, and make the changes economical viable. Some of the savings made can be used for investments in sensors for comfort.

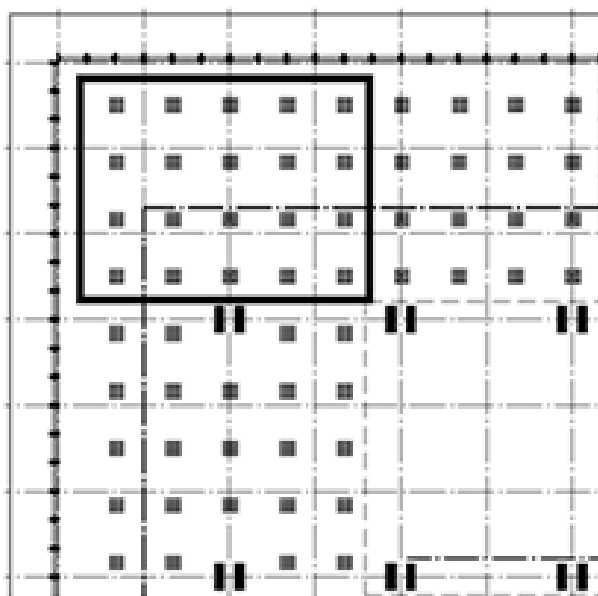
2.1.1 Smart library Pilot project

Before all the lamps are going to be replaced and sensors to be installed in the fittings, a pilot project has been made. 20 lamps was replaced with LED's and different sensors will be installed including a CO2-sensor, a sound level meter, a moisture meter and a lux meter. The goal with the pilot project is to test the quality of the chosen equipment and to decide the amount sensors needed.

2.1.1.1 Pilot project area

The northwestern part of the library's 1st floor will function as the pilot project area. The idea is to test the solution developed in this thesis in the pilot project area. The pilot project area is for testing and it undergoes changes frequently. The following description is from the initial setup[dtua].

Figure 2.1 illustrate the pilot project areas initial setup. The area consists of 20 fixtures (4x5), marked with a grey square. There will be one PIR-sensor and one Lux meter in the area. The windows are marked with a dotted line. If proven necessary more sensors can be added. The LED lamps used in the area are created by the company iBond. The light temperature of the lamps can be changed from 2700K to 5700K. The lamps cant change color, but only the color temperature can be altered. DALI is short for "Digital Addressable Lighting Interface" and is an open standard used for lighting control. DALI enables equipment from different manufactures to be connected. A lamp manufactured



Figur 1. 5x4 Armaturer

Figure 2.1: Pilot Project Area, Reprinted from DTU library, Retrieved November 10, 2016, <http://www.bibliotek.dtu.dk>

by one company can connect with a dimmer from another company, if both the dimmer and lamp support the DALI standard. The lamps installed support DALI and can be controlled with a lighting system called "Lightify pro". Lightify is created by OSRAM and will be explained in more detail in the "Existing Technologies" section.

2.1.1.2 Pilot Project Update

During the pilot project carried out by the library, it was concluded that the tested LED lamps from iBond did not live up to the expectations. It was also found that replacing the ceiling would give a lot more freedom to where the lamps and sensors can be placed. Instead of using the existing holes in the ceiling for the new lamps, rails will be installed where lamps and sensors can be added. This gives more freedom to how many and to where the lamps can be placed. The initial set of lamps to be installed were 620. In the new setup there won't be a fixed number of lamps since new lamps can be added or removed as needed.

It is not decided yet which lamps will be used. However it has been decided that, unlike the lamps from iBond, the new lamps need to be able to change not only the temperature and brightness but also the color. It is not a requirement but the library would like Li-Fi in the new lamps. Li-Fi is short for "Light Fidelity" and is a wireless communication network similar to Wi-Fi. Unlike Wi-Fi which uses radio waves, Li-Fi uses light waves from LEDs to transfer data[Haa]. iBeacons will be added in the pilot area to test if it has influence on the Wi-Fi signal strength. If no significant decrease in the signal strength is detected, iBeacons will be installed throughout the library. iBeacon is a protocol created by Apple. Devices using the iBeacon protocol, also called beacons, have unique ID and transmits bluetooth signals obtainable by other bluetooth devices like smartphones[App]. By having a grid of beacons, smartphones are capable of knowing its exact location inside a building. If added to the library, the beacons could be a part of a lighting system.

The initial results from the pilot project show that "Smart library" is still in an early stage. Everything created to be part of "Smart library" needs to be flexible so it can adapt to changes made.

2.1.2 Living Lab

A Living lab is a research and innovation concept where researchers use real-life settings as their lab. Living labs integrate research and innovation by placing their subjects in the center of the experiment[Alm11]. The research and innovation are thereby made in co-creation with their subjects, while they live their everyday life. Living labs will often operate in a demarcated area. In the case described in this thesis the living lab is the library at DTU. At the library a distinction is made between two different types of living labs.

- An area in the library will be closed and used as a classroom. In this room it will be possible to add new sensors and experiment with different lighting systems, ventilation systems, and other products related to an intelligent building. This room will have little or no restrictions. It can be used for learning and research or for situations requiring specific settings.
- The rest of the library will also function as a living lab. First priority in the open area is a good indoor climate. The vision is to let the users interact and have an impact on the indoor climate of the building. Unlike the closed area there will some restrictions to what extend the users can interact with the building in the open area. The restrictions is there to avoid disturbing other users of the library. All usage data will be collected so they can be analysed if needed. People not using the smart facilities of

the library will also be part of the living lab as passive actors. Data about temperature, air quality, whereabouts and so on will be collected.

Companies will also be able to test their products or make experiments using the library as a living lab. DTU Library wants to be a place with good indoor climate by making the library smart. The hope is to attract students and researchers to use the living lab to find ways to improve the indoor climate.

2.2 Project Description

DTU library wants to improve the indoor climate, by making the library smarter. The library is looking for ideas to how the new lighting system can be part of the Living lab and help to increase the indoor climate. This thesis will focus on the lighting aspect of the "Smart Library" and how the indoor climate can be improved by developing a software platform, that takes advantage of the sensors and the LED lamps installed in the pilot project. The thesis will look at previous studies to understand in what ways lighting can be used to improve the indoor climate. Existing technologies related to intelligent lighting systems will be analysed to determine if they can be used in the library. This thesis will not analyse what exact lighting settings that will be the best in a given situation or what sensors to be used. **This thesis is about creating a solution that helps the library and the Living Lab to increase the indoor climate.** Reducing the energy consumption may be an additional benefit, but this remains a secondary objective.

2.2.1 Problem Formulation

This section outlines the overall goal of this thesis.

The goal is to create a solution to help the library and a living lab to increase the indoor climate by using the newly installed LED lamps and sensors at the library. The solution will be a software system that can connect the lamps at the library with different types of sensors. Sensors could include lux meters, sound level meter and PIR sensors, but the system needs to be able to handle all kinds of sensors. The system is a prototype so it is important that the system has a good architecture, so future developers are able to continue the development. The overall goals for the software system to be developed are:

- Help the Living Lab to conduct research

- Being able to control the lighting to help the library to improve the indoor climate
- A transparent software architecture, which can be further developed by others
- Saving energy (not first priority)

Improving the indoor climate and helping the users of the Living lab has first priority. Saving energy is a bonus but not if it compromises indoor climate or the Living lab users. Improving the indoor comfort will require looking at studies made on correlations between lighting and indoor climate. Saving energy will require an automated lighting system, able of dimming or turning off the lighting in well lit or empty areas. Living Lab users need to be able to control the automated lighting system, and access data to analyse different lighting settings. It is very important that the software is flexible so students and researchers can add new sensors or lighting systems if needed. The first phase of "smart library" is only a small corner of the library. The system needs to be scalable so it can handle the rest of the library later on. In this thesis a transparent architecture means a system design using well known software architectural patterns. This makes it easy for other developers to continue the development.

2.3 Dynamic Lighting

This section looks at the possibilities for using dynamic lighting to meet the requirements of the system to be developed. Artificial lighting is typical constant in both colour temperature and luminance[dKS10], where natural daylight changes throughout the day. In this thesis dynamic lighting will refer to artificial lighting that changes colour temperature and luminance during the day. Dynamic lighting changes according to pre-set parameters(automation) or by human interaction.

As stated in the problem formulation topic for this thesis is to create a it-platform, which can be used for improvement of the indoor climate in general and the lighting in particular at the DTU library. A detailed analysis of dynamic lighting is therefore outside the scope of this thesis. A brief introduction to dynamic lighting is however relevant as dynamic lighting will be the primary application area for the platform developed in this project. Therefore this section includes an overview of the benefits of dynamic lighting and technologies available for supporting dynamic lighting.

2.3.1 Benefits of Dynamic lighting

Some studies show that the indoor climate of an office space can increase the productivity level of the office workers[LM97][Fis00][Jen][RKA]. A study shows the productivity level can be increased as much as up to 15%[LM97]. However some studies found a big uncertainty of the magnitude of the increase in productivity level by improving the indoor climate[Fis00]. The indoor climate can also have an effect of the overall health of the office workers[Fis00][Woj]. For private companies an increase in productivity means an increase in revenue. This is not the case at the DTU library, but increasing the productivity and the health of its users certainly is. A big part of improving the indoor climate is to improve the lighting in a building[Mor]. Even though some studies indicate that lighting has an impact on productivity, it is uncertain how much dynamic lighting can increase productivity, if any at all. The US Department of Energy has estimated that businesses will benefit financially from dynamic lighting[Hag]. No major studies confirms this, and they don't estimate the gains in productivity. The uncertainty of the impacts of dynamic lighting makes this project even more important. One of the reasons for developing a system for controlling the light at library, is to be able to use the library as a real-world lab. The lab can then be used to conduct further studies on the impact dynamic lighting has on the indoor climate. There are several studies showing that people find lighting they can control themselves more satisfying than typical artificial office lights[dKS10][Log][Log13]. Besides making the users more satisfied in general, dynamic lighting can also have several other benefits. Seasonal affective disorder (SAD) or more commonly known as winter depression, is extreme sensitivity towards seasonal changes[OOB14][MD]. Though not much is known about SAD, researchers agree, that people suffering with SAD are very sensitive to light or the lack of it[MD]. Studies indicate that people with SAD feels better after exposure to bright light. However exposure to bright light is not enough. It is also a matter of getting the light at the right time of the day, the morning being the most important[MD]. Studies have shown that people with sub-syndromal seasonal affective disorder (SSAD) benefits from bright light at their workplace. The light has a positive effect on mood, energy, alertness and productivity[AKBH01]. So giving the users of the DTU library the option to be exposed to bright light, in the morning and afternoon, might increase the live quality for people sensitive to seasonal changes.

The library is used for numerous of activities: reading papers of laptop/paper, brainstorming, performing experiments, taking a nap etc. Different activities requires different lighting. Studies shows that creativity is effected by the surrounding light. Dim lighting has shown to improve creativity performance[SW13]. Darkness makes people feel free from constrains and a willingness to take more risks[SW13]. Normally if someone wants to relax or take a nap they prefer darkness or at least dim lighting. If you are reading from paper you might want

brighter light compared to reading of a screen. Since the library is used for many different activities, finding a lighting setting suited for everything is impossible. This means user controlled lighting in the library can offer a lot of value.

2.4 Existing Technologies

This section presents current dynamic lighting solutions and how they work. At the end of the section the solutions will be discussed and compared to the system requirements of the DTU library.

2.4.1 Phillips Hue

Phillips hue is a lighting system that target the private households. The core system consists of a bridge, light bulbs and a smartphone application. The system can be expanded with dimmers and switches. Philips are currently working on adding a PIR sensor that can be added to the system[Huec]. When the bridge is plugged in the system is up and running, and can be controlled with a smartphone. The bridge itself only establishes a connection between the bulbs and the router. The official app serves as the brain of the system, but it has a lot of limitations. The app can not be paired with third-party sensors, dimmers or buttons. The system has an API which allow developers to create third-party applications to control the light. Looking at the subreddit for Phillips Hue points towards an active community[huea]. More than 14 new threads were added within one day (2016-10-20)[huea]. The bridge is limited to 50 connected devices. In 2015 Phillips banned third-party light bulbs from their system[Hueb]. Phillips later reversed that decision, but you cant be certain Phillips bridge will continue to support third-party light bulbs. It is not possible to set up multiple users. As long as you are connected to the wifi you can control all the light bulbs with no restrictions.

2.4.2 Lightify Home

Lightify Home is created by Osram and is similar to Phillips Hue. As seen on figure 2.2 the setup is the same as Phillips hue and as mentioned before they are even compatible with each other, as of December 2016. Like the Phillips Hue adding third party devices, not including bulbs, to the Osram gateway(equivalent to the Hue bridge) is not possible[Lig]. Lightify home bulbs

is compatible with SmartThings. It could be an issue in larger setups that no more than 5 users can use the Lightify simultaneously. Lightify also comes with an API, however all apps using the API will have to access the OSRAM's Lightify Cloud service.

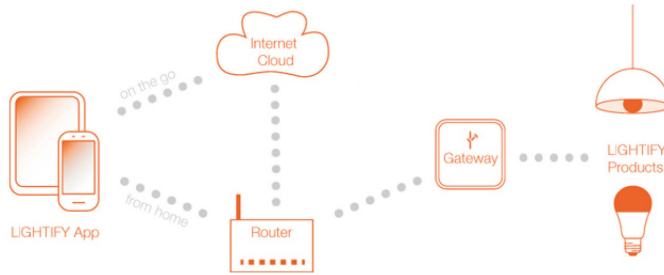


Figure 2.2: Lightify Home Setup. Reprinted from Osram, Retrieved November 16, 2016, <http://www.osram.dk>

2.4.3 Lightify Pro

Lightify Pro is a lighting system made for small to medium sized offices and can control up to 100 devices. Lightify Pro is used in the DTU library as a showcase for DTU Smart Library. All lamps can be made Lightify Pro compatible by installing the DALI controlling device LIGHTIFY Pro DSE[pro]. Light and motion detection sensors are compatible, but installing other kinds of sensors is not possible. Unlike both Phillips Hue and Lightify Home systems, there are two kinds of users in the system. Figure 2.3 shows the setup including 2 types of users/apps. The commissioning app is used to setup the sensors and the lamps and to divide the lamps into groups/rooms. The control app is used by the users to control the strength and colors of the lamps. There is an API available and unlike Lightify Home using the Lightify cloud is not required. As you could expect the components of Lightify Pro is expensive compared to that of Lightify Home.

2.4.3.1 Lightify pro testing

While testing the available API some problems were experienced. Connecting to the gateway and getting a response didn't take long. However every call

requiring a user's credentials failed with the following response:

```
"errorCode":5001,"errorMessage":"Invalidcredentials"
```

Contacting the customer support was not a success. The community is very small and did not have a solution either. The available API documentation could be out of date, or a simple reset of the gateway could fix the problem, but since the gateway was in use this was not an option.

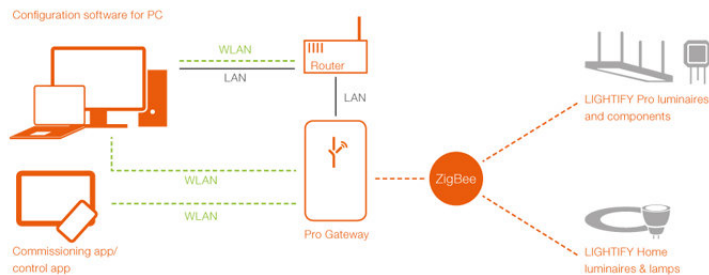


Figure 2.3: Lightify Pro Setup. Reprinted from Osram, Retrieved November 16, 2016, <http://www.osram.dk>

2.4.4 Samsung SmartThings

SmartThings is primarily made for use in private residences. SmartThings is made for controlling and connecting all smart products in a residence including lighting. The system's ecosystem consists of a hub connected to the home-router, a smartphone app and "Things". "Things" are home-network based devices compatible with the hub. Currently there exist 50 different "Things" compatible with the SmartThings hub, including lighting systems and alarm systems and motion sensors. Both Phillips hue and Lightify light bulbs are supported by Samsung SmartThings. The hub makes it possible to control "Things" through a wifi network. Instead of having different PIR-sensors to control both security and lighting systems, SmartThings can connect the different systems to the same sensor.

Samsung has created an API so giving developers the option to create third-party apps. "SmartRules" can be mentioned as a third-party app. "SmartRules" is a rule engine using smartThings to automate different processes. An example could be turning on the light in the bedroom when the alarm clock goes on.

SmartThings supports devices using both ZigBee and Z-Wave as communication standards. Not all ZigBee or Z-wave devices are compatible. The device needs to be partner with SmartThings to work. Currently 100 "Things" can be connected to the hub at the same time. SmartThings can have multiple users but only one user type exists and adding restrictions to users is not an option.

Security issues with SmartThings have been reported. Researches from Michigan university where able to unlock a door, controlled by the SmartThings hub, without permission[FJP16].

2.4.5 ITTT

ITTT can link automate tasks by linking services together. It is comparable with the rule engine "SmartRules" created for Samsung SmartThings rule engine. ITTT is short for "if this then that", is an online service letting users create applets. An ITTT applet is a condition (the "if this" part) paired with an action (the "then thart" part). When conditions are triggered the paired action is performed. Conditions are triggered based on input from services such a facebook, time, or calender. Actions use services like SMS, phone calls and Phillips hue, just to name a few.



Figure 2.4: ITTT applet

Figure 2.4 illustrates an example of an applet. 15 min before a google calender event is about to start Phillips hue light bulbs will turn red. ITTT can be used in cooperation with Samsung SmartThings to automate and control devices connected to Samsung SmartThings. ITTT is limited to services that ITTT has approved. ITTT can not be used offline and all services has to be online as well.

2.4.6 Summary of Existing technologies

Only the most relevant lighting solutions has been described. Phillips Hue is interesting because of its popularity and for being the best rated system for residences[Tor]. OSRAM is relevant since it is currently used at the library, and for having a pro version. Companies like Belkin and GE (General Electric) have their own products. These products are similar to both Lightify home and Hue. They are built for residence and consists of a hub connected to bulbs with ZigBee. None of these solutions fulfills all the requirements the library has as a standalone solution. None of the residence solutions can handle multiple user

types and don't scale well due to a limitation on 50 connected devices. Lightify pro is designed for larger lighting setups as the one at the library. However like the systems for residences it is not possible to use third party sensors and usage data is not stored. SmartThings makes it possible to connect multiple different smart systems and let them work together through a rule-based system. Combining SmartThings with Lightify Pro could solve both the automation and customization requirements for the lighting at the library. Multiple users have reported that 100 "Things" is not enough for their home setup[thi]. If 100 things is not enough for certain households. its questionable if its enough for the library. The lag of multiple user types and being limited to 100 things, is not viable for controlling the lighting at the library. The approach of combining multiple sensors and smart systems with a rule-based system is what the library is looking for.

CHAPTER 3

Proposal for System Requirements

In this chapter we try to define the requirements to an IT lighting control system that would match the demands for the library case. A proposal for the requirements will be presented in this chapter. A Use-case diagram will be used to illustrate what actions the different user types can perform. To gain a deeper understanding of the flow of the system, the most important use cases will be elaborated as a "system sequence diagram". A system sequence diagram is a detailed illustration of the system flow of a use case. The software system has 3 overall goals.

- Help Living Lab to conduct research
- Give DTU library full control over the lighting at the library
- A transparent software architecture, which can be further developed by others

As of November 2016 all lighting is controlled by a few lighting switches. The software system will give the library full control over the lighting system. Administrators at the library will be able to control the lighting through an automation system. The automation system can control what conditions must

exist for the light to turn on/off or dim. An example could be that the library wants the light to turn off in a section, if enough daylight is present. For this to be possible the software system needs to be able to communicate with other systems and devices, like a light-meter and the lighting system in the library.

The library guests must be able to adjust the lighting. As discovered in the background chapter some studies indicate that people are more satisfied with lighting they can control. If changes are made by the automation system that the guest finds disturbing, he or she shall have the possibility to reverse the changes made. The library guests will be able to customize the lighting with a controller app installed on a smartphone or a tablet.

The living lab users can also benefit from giving the library guests the possibility to control the lighting. The users of living lab will be able to see which lighting settings the guests prefer and compare them with the conditions in the library, at a given time. The living lab users will also be able to configure the automation system, to create experiments. An example of an experiment involving the automation system, could be to have a section of the library imitating the daylight outside. Assuming people only make changes to the lighting if they are not satisfied, it will be possible to see what kind of lighting the library guests prefer, in a given situation.

The software system is a proposal for a possible software solution and not a finished product. This makes a solid software architecture important, if future developers are to continue the work. A solid architecture makes it easier for developers to read the code and continue the work.

3.1 Users

There will be 3 different user types in the system:

- Administrator
- Living Lab user (researcher)
- Guest

The use cases used in this thesis is built around Craig Larman's definition found in the book "Applying UML and Patterns"[Lar]. A use case diagram illustrates the names of the actors and use cases, and the relation between them. Figure 3.1 shows the 3 different user types in the system and their use cases. The

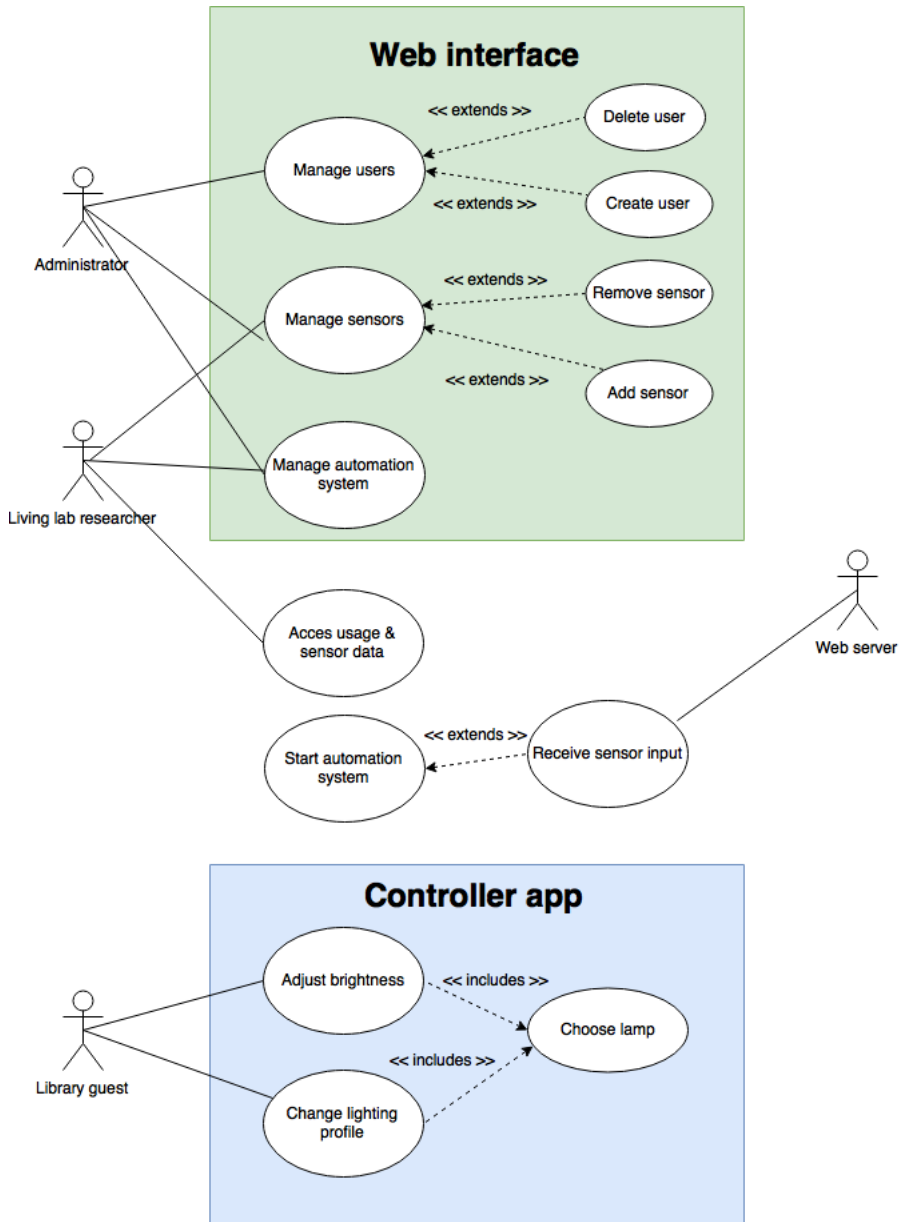


Figure 3.1: Use case diagram

administrator is the library management. The administrator is responsible for setting up the automated lighting system. They decide how the lighting at the

library should respond to sensor inputs. The indoor climate and the energy consumption of the lighting is indirectly managed by the administrators, as they control the default lighting settings. Administrators are also in charge of creating users of the type "Researcher". Administrators access the system through a web-interface where a password is required. There can only be one Administrator. Researcher is a user type who is part of the Living lab. There can be multiple Researchers in the system. Researchers can adjust the automated lighting system, just like the administrator, but with a number of restrictions. This is done through the web interface, where login credentials are needed. Researchers can also access all usage data stored in the system with an API. Guest is the user type of all visitors with access to the WiFi at the library. Guests can adjust the lighting settings on individual lamps/groups connected to the system. Adjustments by guests is done with a smartphone application. Guests can either adjust the brightness level or choose between predefined profiles. When a guest makes adjustments to a lamp, the automated lighting settings will always be overwritten. No login credentials is required but guests will have to be logged on to the library's WiFi, to make use of the samrtphone application. The web server needs to be able to receive sensor inputs, and start the automation system if the sensor exists in the system. The automation system will be explained in greater detail in the chapter "Design of the system".

3.2 Sequence diagrams

System sequence diagrams (ssd) is used to elaborate an use case. A ssd does not show how the software handles different opertions, but it shows the flow between the actors and the software system. Leaving out technical details makes a ssd very useful to get an understanding and an overview of the flow of between the system and the actors. The two most essential use cases for this system has been elaborated as sequence diagrams.

The system sequence diagram shown on figure 3.2, shows the flow of a library guest adjusting the lighting using the controller app. It can be seen that the controller app is not communicating directly with the lighting system. The usage data is an important part of living lab and by having the web server handling the requests from the user, it is possible to log all the changes the user makes. After the user input is saved, the web server sends a request to the lighting system to adjust the brightness. When the lighting has been adjusted the controller app is updated so it shows the current lighting settings.

Figure 3.3 shows how the web server handles a sensor input and when a sensor detects a change, that change is send to the web server. The web server checks

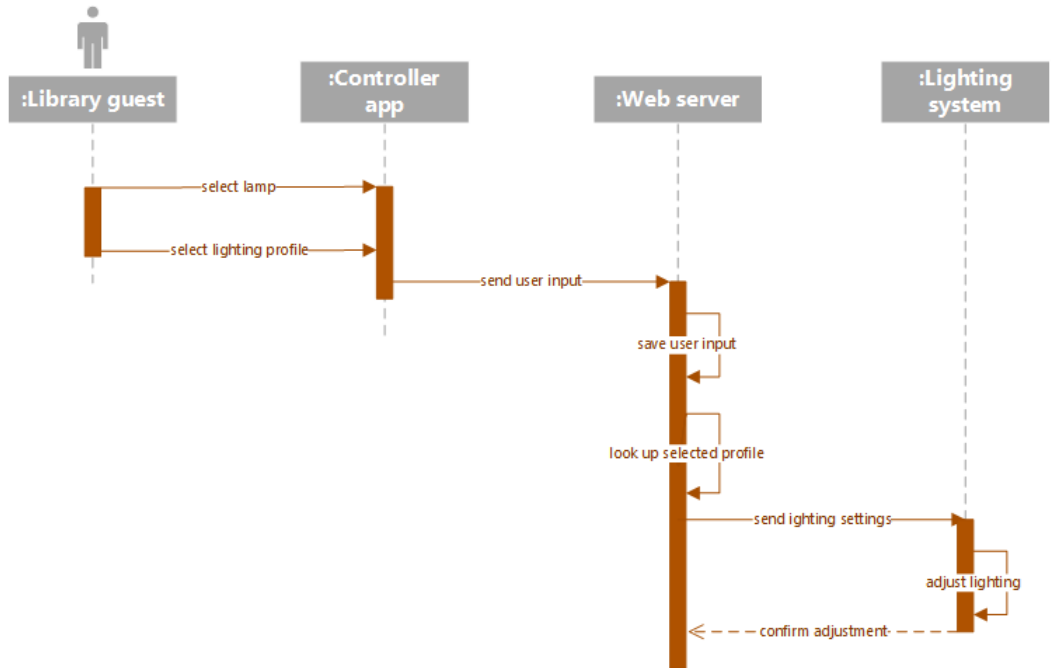


Figure 3.2: Sequence diagram of a user adjusting the lighting

the system to see, if the sensor exists in the system. If the sensor is part of the system, the sensor input is stored on the web server, and the web server starts the automation system. The web server then tells the lighting system to change, depending on how the administration has set up the automation system.

3.2.1 User Story Mapping

User story mapping is a method where all interactions between the product and the users are written down and sorted. The end result is an overview of the complete usage flow of the product. It can be used by the developers to design the system, and to get a non technical overview of the systems functions. The user story map seen on figure 3.4 uses the definition found in the book "User Story Mapping" by Jeff Patton. A user story map contain no technical detail about the product and tells the story of the product from a user's point of view. Story mapping is a flexible tool that can change over time after research or testing. The rule system seen on the user story map were altered after researching how ITTT and Samsung smartThings works (see existing technologies). The tool is

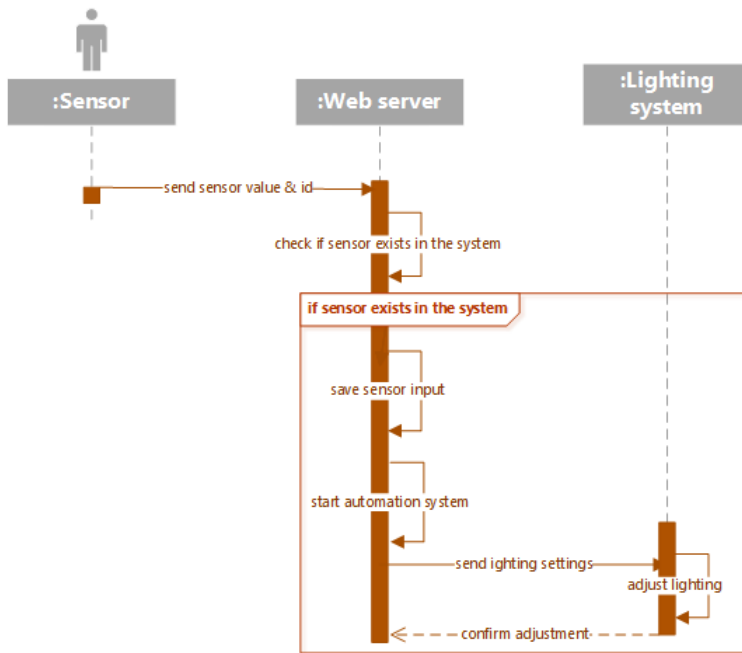


Figure 3.3: Sequence diagram of the automation system

often used in agile and lean software developing environment. The user story map can be used to prioritize the working order of the product to be developed. The stories is sometimes sliced into a MVP (minimal viable product). The user story map seen on figure 3.4 is the latest edition used in this thesis. The map is two dimensional where the horizontal axis represents time and the vertical axis represents priority. The horizontal axis represents the flow seen from the user's point of view, while the vertical axis represents task priority. In software development priority often represent a sprint or a release version of the software.

As seen on figure 3.4 the map contains 3 different labels. Each label has its own meaning.

- The blue labels is referred to as user activities. Activities describes what the user wants to do with the product.
- The yellow labels are called user tasks and describes what the user needs to do to archive the different activities. Tasks can be seen as high level user stories.

- The white labels are referred to as user stories. The user tasks might be doable in multiple ways and is shown as user stories. User stories can be seen as a more detailed user task.

In this thesis the first priority is letting the administrators control the automated lighting system and letting the guest manual adjust the lighting. Retrieving data through a user interface and managing accounts is very important for the final product, but is not prioritized for a proof of concept. The stories marked with green is done while do grey is scheduled to be done.

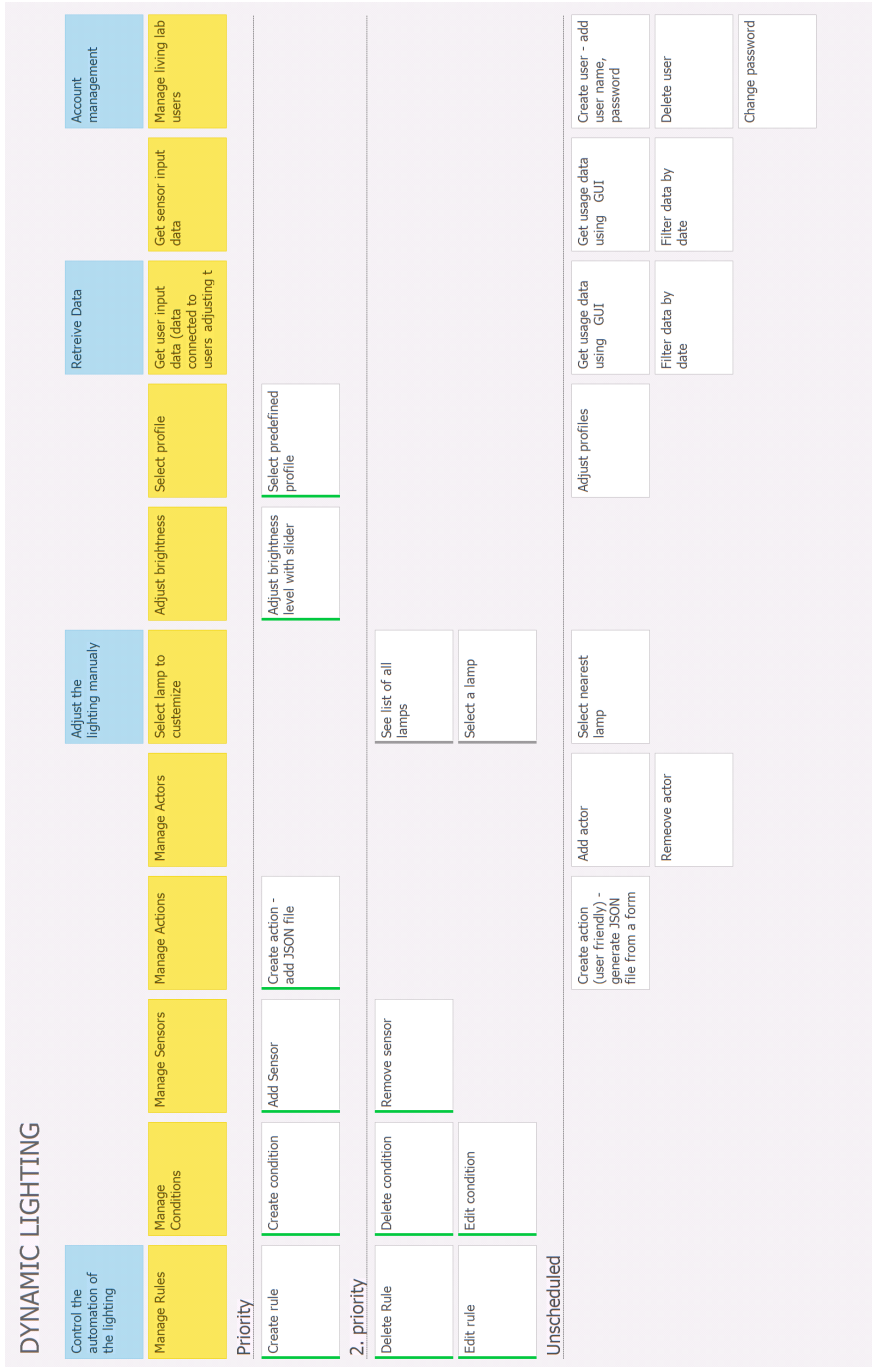


Figure 3.4: User story map

Design of the system

This chapter deals with the overall design of the system. This chapter will not only explain how the system is built, the reasons for the different design choices will also be discussed. The system basically consists of a web server connected to a database and a controller app. All back-end code will be written on the web server. The controller app will be pure front-end, and work as a tool for the user to communicate with the web server. All logic will be handled by the web server. The back-end, the front-end and the database design is described in this chapter. A use case diagram and an explanation of each use case can be found in the "Proposal for System Requirements" chapter. The use cases discussed are the basis for the design of the system. As seen on the use case diagram in figure 3.4, there are 4 actors in the system. The web server, the administrator, living lab user and library guests.

- The System(Web server)
 - Receive sensor inputs
 - Store Usage data
 - Control the lighting automatically
- Administrator
 - Manage Living Lab users

- Control the automation process of the system
 - Manage sensors in the system
- Living lab user
 - Control the automation process of the system
 - Manage sensors in the system
 - Access Usage data
- Guest
 - Adjust the brightness
 - Change lighting profile

The list outlines the overall features the different actors requires from the system.

4.1 Web Server

The web server is written in the language Ruby and is using the framework "Ruby on Rails". Ruby on Rails was chosen mainly because its a MVC-framework. A MVC-framework helps structuring the code by dividing it into three parts, which makes it easier for developers to maintain and develop the system. MVC will be explained in detail in the coming section. Rails encourages the use of the well known software engineering pattern and paradigm CoC and DRY. CoC is short for Convention over Configuration and is a big part of the philosophy behind Rails. The purpose of CoC is to minimize the number of decisions the developer has to make without losing flexibility. An example of such a decision is data table naming convention. If a class on the web server is called "Sensor" the corresponding table name in the database would be "sensors" by default. CoC saves time spend on developing and makes it easy for other developers to understand and maintain the code. DRY is short for "Don't Repeat Yourself" and is strategy reducing repetition and thereby complexity of a software system. MVC encourages the developer to write DRY code and is an important part of rails. The Automation system and control application and sensor application, will be covered later in the chapter.

4.1.1 MVC

To understand the architecture of the web server it is necessary to understand MVC as this is the structure used in Ruby on Rails. MVC is short for "Model

View Controller". MVC is a software architecture paradigm where information is processed in a certain way. MVC is used in systems implementing a user interface. When using MVC the code is divided into three interconnected parts: Model, View and Controller. Figure 4.1 shows the parts of the MVC design pattern and illustrates how they are connected.

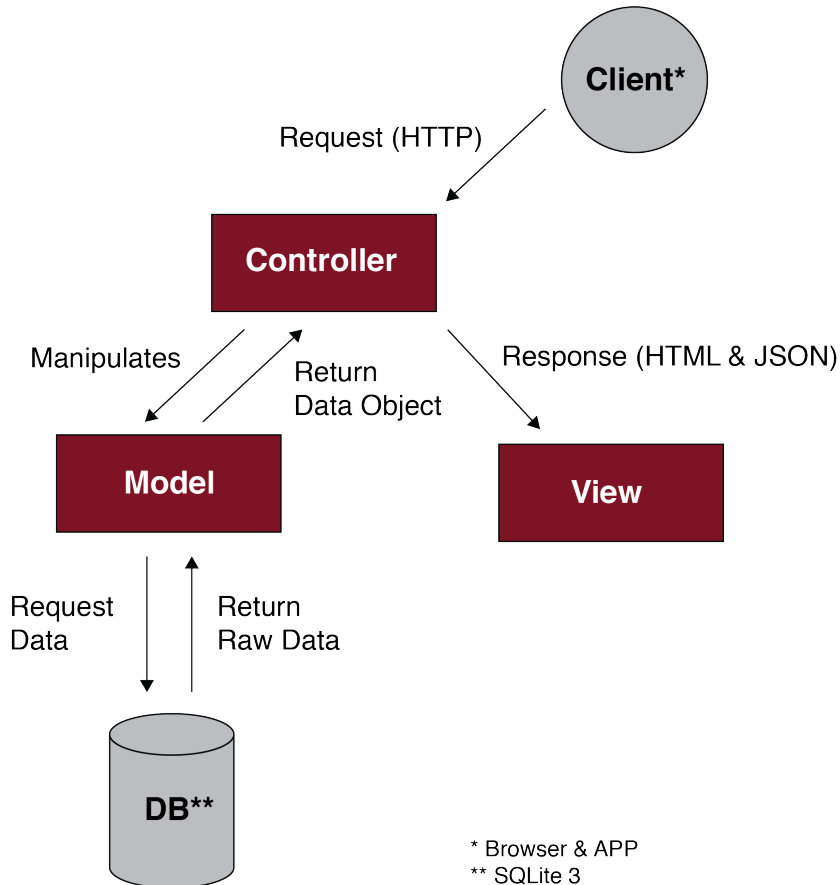


Figure 4.1: MVC Architecture

- **Model:** All CRUD operations and data handling in general is written in the model. CRUD is an acronym for create, read, update and delete which are the four major functions in a relational database. The model takes care of adding and retrieving data from the database and maintains the relationship between objects and the database. In Ruby on Rails the

model is also referred to as "ActiveRecord". The model processes the raw data from the database and pass it on to the controller as an object.

- View: The view is the only part of the system the user will ever see and consists of HTML, CSS and Javascript. If the view is send to a browser the browser would render the view as a web page. The view uses data from the model accessed through the controller. Unlike the Model the communication with the controller is one-way. The view only listens to the controller. From a users point of view it looks like he interacts with the view, but in reality all interaction is with the controller. In Ruby on Rails the view is referred to as "ActiveRecord". When interacting with the web server through the smartphone app the view is not relevant, as the app has its own interface.
- The controller sends commands to the model in order to edit or retrieve data from the database. Requests made by the client are processed by the controller. On the web server developed in the thesis the requests send to the controller will consists of HTML-requests. The controller controls what data from the model the view can access. In Ruby on Rails the controller is referred to as "ActionController".

When the web server follows a solid and a strict structure, it makes it easier to maintain. When developing software avoiding repetition of the code is desirable. A solid structure helps preventing repetition. Repetition in software is to be avoided. The more repetition in code the more code needs to be edited when a change is made. In a MVC framework changing the view doesn't require a change in the model and vice versa. By having the MVC structure it is easier for new people to work on the system and the structure helps keeping repetition to a minimum.



Figure 4.2: Basic Web Architecture

Figure 4.2 illustrates an example of a web application not following the MVC structure. Everything is handled at the same place. A structure like this requires that the developer knows everything about the whole system. When using MVC as an architecture this is not an issue for the developer. Front-end developers

with no experience with back-end development, can change the view without dealing with the model or the controller.

4.1.2 Communication

The web server will include an API, short for "Application Programming Interface". An API lets one piece of software speak to another through web services. The API will be used to receive data inputs from sensors and as communication between the web server and apps to control the lighting. A RESTful web service, sometimes referred to simply as REST, is short for "Representational state transfer". An API built as a RESTful web service uses HTTP-requests as communication. The data send between the web services is written in the JSON format. JSON is a language independent data format used for transmitting data objects consisting of attribute-value pairs. JSON is language independent making it a good choice for transmitting data between independent systems. XML is another example of such a format. JSON is chosen in this system because its more lightweight than XML, and it is supported by the lighting systems looked at in this thesis.

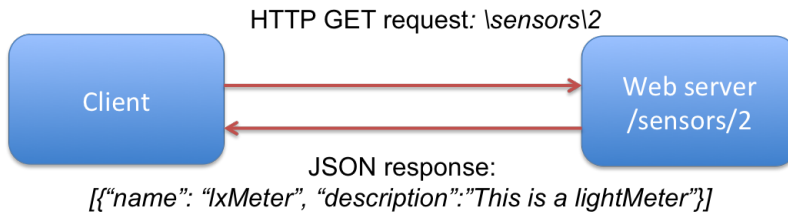


Figure 4.3: Example of a GET request send to a RESTfull web service

Figure 4.3 illustrates a client requesting data from the web server about a sensor with the ID 2. The web server responses by returning a JSON-file with information about the sensor.

The communication between the web server and the lighting system is also handled with the use of RESTfull web services. All the systems looked upon in the "existing technologies" section supports communication through RESTfull web services.

4.1.3 Automation System

The automation system needs to be able to link different sensors and systems together. The automation system should be able to change the lighting system based on inputs from sensors or services not directly supported by the lighting system. ITTT, described in "existing technologies, is able to communicate with different sensors and services independent of each other. The automation system is inspired by ITTT in the way it links different systems together. A system like ITTT can be referred to as a "production rule system". The automation process will be controlled by "rules". In this system the rules will be setup and managed by the Living Lab users, if access is granted and by the administrator. A rule consists of a number of preconditions, from now on referred to as conditions, and an action. If the conditions are all met the rule is triggered and the action will be executed. Figure 4.4 illustrates process of evaluating a rule.

When a rule is evaluated all conditions are checked as seen figure 4.4. If a condition is false, the evaluation stops. If all conditions are true the rule is triggered and the action will be executed.

A condition consists of a trigger value and an operator. Depending on the input value from a sensor the condition is either true or false. Equation 4.1 shows the simple equation that determines the outcome of the condition.

$$condition = (sensorValue <, >, = triggervalue) \quad (4.1)$$

Figure 4.5 illustrates the flow for how the system reacts to a sensor input. When the system receives a sensor input, all the rules with a relation to the specific sensor will be examined. If a rule fulfills a set of conditions an action will be executed. When executed the system will send a POST-request to the actor related to the action.

A concrete example has been made to explain how the rule system works. In this case we want the artificial lighting to change according to the amount of lux measured by a lux meter. The lux meter will have the identifier `sensor1`. First a rule is created. To make the light dim if `sensor1` detects more than 300 lux, the following condition is added to the rule: `if (value of sensor1) > 300`. where ">" is the operator of the condition and 300 is the trigger value. If condition is true the rule will trigger and the action will execute. Therefore an action needs to be added to the rule. An action needs to be written as a JSON file.

If `sensor1` sends a value to the system greater than 300 the rule created will be triggered and the action executed. Executing the action will in this case mean

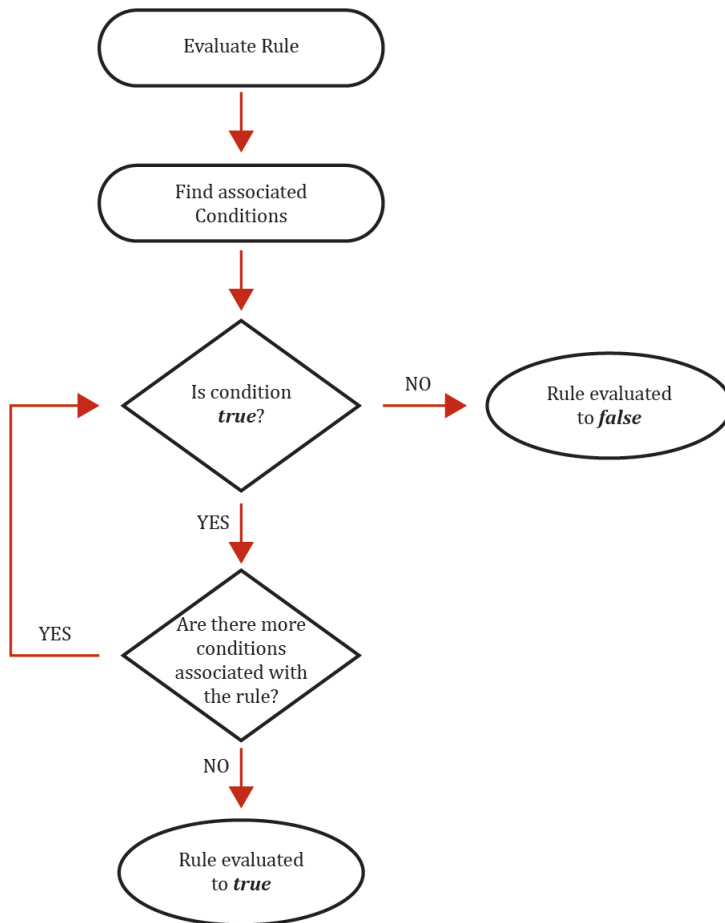


Figure 4.4: Rule evaluation

sending the JSON file the lighting system, and the brightness of the Phillips hue bulb will be set to 50.

4.2 Controller Application

The controller application allows the users to override the lighting settings on a single lamp or a group of lamps, depending on the library setup. The Controller app will be able to select a lamp and adjust the brightness of the lamp. 4 prede-

defined profiles will let the user switch between 3 lighting profiles and a "default" profile. The "default" profile will let the lamp be controlled by the automation system described earlier in this chapter. The other profiles is situational lighting settings, and is defined by the living lab users. The following are examples of situational profiles:

- Relax - Turn down the brightness and make the temperature warmer.
- All Nighter - Turn up the brightness and make temperature cooler.
- Off - Turn of the lamp.

The application is only front-end and all logic is handled by the web server. All actions performed using the app will send a HTML-request to the web server. The HTML-request contains information of the selected lamp and the brightness level and the selected profile. The web server will handle the HTML-request by saving the action performed in the database and send the action as a JSON file to the lighting system. The Interface of the controller application change according to the state of the selected lamp, by requesting the current state of the lamp from the web server.

4.3 Database

This section describes the database that fullfils the requirements for the system to be developed.

In figure 4.6 a database model diagram can be seen. The diagram shows what tables are required and how the tables are related to each other.

The automation system involves the following tables:

- sensors
- rules
- conditions
- actors
- actions

The rules table is what binds everything together. Besides the more obvious columns a rule has a "state" and the column "triggered_at". The idea of "state" is that you can activate and deactivate a rule. "triggered_at" is a timestamp of when the rule were triggered last. This is used by the web server to control how often a rule can be triggered. A rule is related to multiple actions. If a rule is triggered all the related actions will be executed. An "actor" in this thesis is a third party software system that can receive input and react on the input. In thesis an actor will be a lighting system like Phillips hue or Osram Lightify. In theory an actor could also be a system controlling the blinds or even the windows themselves. The idea is for rule to be able to control multiple lighting systems or even other type of actors. The "aaction" table has an attribute called "config" and contains a JSON string. The string contains the lighting settings that will be sent to the actor, if the related rule is triggered. A rule can have multiple conditions. For a rule to trigger, all conditions have to be true. The "condition" table has an "operator_value", an "operator" and is related to a "sensor". The "operator" is a string in the database but the web server can only handle the 3 different operators: "<", ">", "=" . The "operator_value" is a fixed integer set by an administrator. The "sensor" table has a column called "value". "value" is updated every time a sensor sends an input to the web server. The web server uses the "operator_value", "operator" and "sensor_value" determines if a condition is true or false, see the section "Automation system". It can be seen that "users" is related to "user_types". A user can be of the type "administrator" or "researcher" (living lab user). There will be 3 different user types by only 2 needs to be managed by the database. The user type "guest" is not managed by the web server but by DTU. Everyone with access to the library WiFi will be a user of the type "guest". The last table is called "lighting_profile". This table is going to contain the lighting configurations of the different lighting profiles found on the controller app. By having the lighting profiles stored in the database as opposed to the controller app, living lab users can configure the lighting profiles without updating the app.

4.4 Security

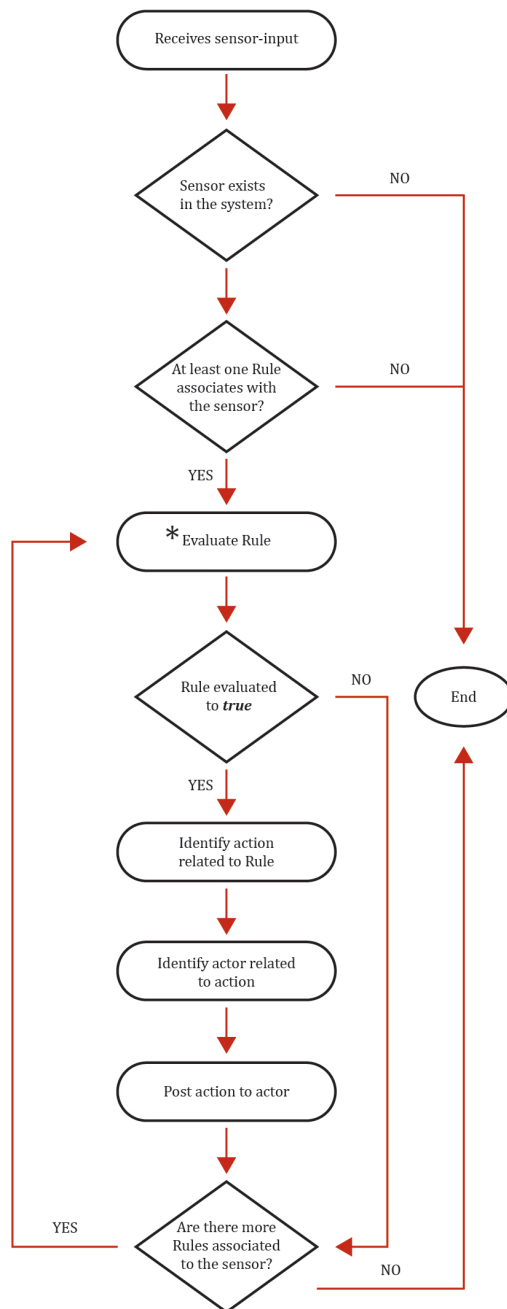
Security is out of the scope for this thesis and has not been prioritised. Only when logging in as an administrator or a living lab user, you will need login credentials. The login credentials will be hardcoded, meaning it wont be possible to change the password.

The guests of the library which are the users of the controller app will not require to login. Not everyone can use the app, and its not possible to sit at home and change the lighting at the library. To use the controller app, it is required to be

connected to the library wifi. The library therefore indirectly controls who can use the controller app to change the lighting settings, by having control over who can access the wifi.

4.5 Sensor Application

It was hoped that the light-sensors currently installed in the library could be used for testing. However the sensors were not connected to an online server. Getting the data from the sensors would require to manually extract the data. Doing it manually is not a viable solution. The whole point is to let the DTU library control the lighting automatically. It was therefore decided to use the light-meter in a smartphone as a light-sensor. Besides using the developed sensor app to test the automation system, it can also be used to show future developers, how to create a sensor application and connect it to the automation system. Using smartphones as sensors is meant as a temporary solution, until the library has developed a method for retrieving sensor data automatically.



* See Figure 4.4: Rule evaluation

Figure 4.5: Rule System

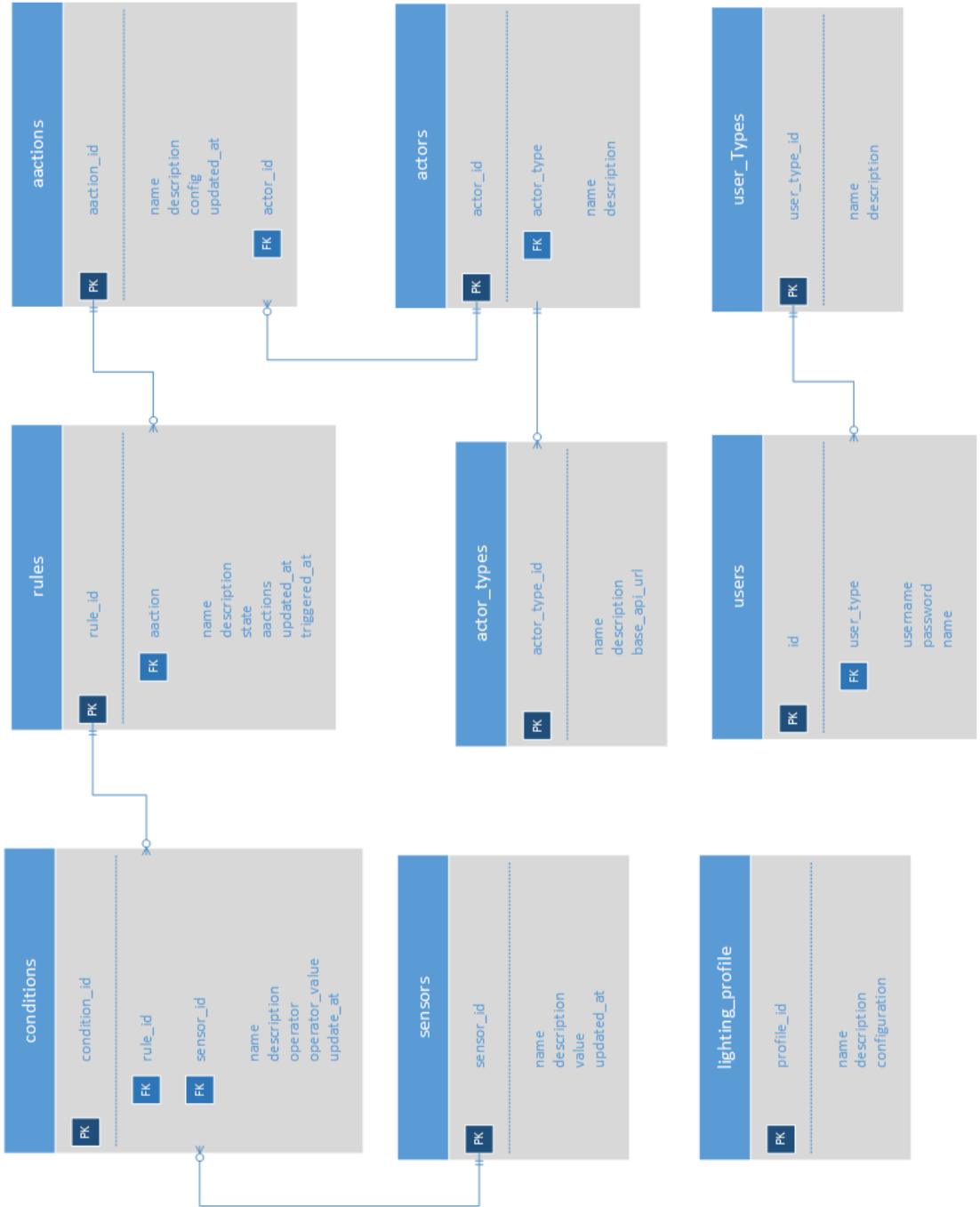


Figure 4.6: ER diagram of the database

Implementation And Limitations

In this chapter the Implementation of dynamic lighting system is described. Both the systems hardware and software is explained in detail. Not everything described in the Design chapter has been implemented yet and some things are implemented a bit differently. These differences between design and implementation are due to time constraints and will be altered in the future to match the initial design. The differences are minor and don't effect the systems features. The goal is that future developers will continue the work on the platform. For this to happen, it is important that the platform is build around well known software standards. When implementing the web server and apps. well known standards have been used as much as possible.

The code for the controller application and the web server can be found on github:

- **Controller app:** <https://github.com/lakerolis/LightMeter.git>
- **Web server:** https://github.com/lakerolis/Dynamic_lighting.git

5.1 Hardware setup

The following is a description and an illustration of the hardware needed for controlling the lighting with the developed system. The illustration also shows the relation between the components in the system. The setup described here might differ from the final setup at the library. The library has not yet decided what lighting system that will be used at the library, but it won't be Phillips Hue. In this thesis Phillips Hue will be used as the lighting system. Hue has a well documented API and an active community making it ideal for the prototype.

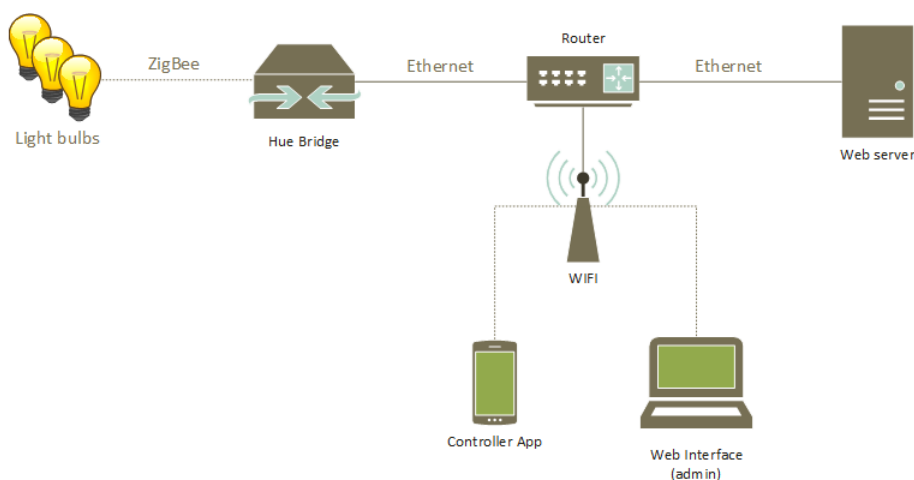


Figure 5.1: Hardware Setup

Looking at figure 5.1 the hardware components relation can be seen. In the prototype setup the bridge and the light bulbs used are made by Phillips. The bridge and the bulbs can be replaced with other lighting systems controllable by an open API, with very little effort. The system runs on a web server connected to a local router over WiFi. In the final setup the web server would be connected to the DTU network and not a separate router. The database is running on the same server as the system. If needed the database can be moved to another server. The connection between the controller app and the administration page to the web server goes through the WiFi network from the router. To use the controller app or the administration page, it is required to be connected to the same network as the web server.

5.2 Back-end

The back-end of the system consist of a web server that handles the automated lighting system and all inputs from the controller app and sensor apps. This section describes how the web server has been implemented following the MVC-architecture. The section also describes the implementation of the automation system that is running on the web server. At the end of this section the implementation of the database is illustrated with a database diagram and each table in the database is explained.

5.2.1 Web server

The web server is written in ruby using "RubyMine" as IDE. All communication with the web server is handled by RESTfull web services. As described in more detail in the design chapter the web server is written in Ruby on Rails and is following the MVC design pattern. The implementation of the model, view and controller is described below.

5.2.1.1 Model

In Rails models is called ActiveRecord. As mentioned in the design chapter CoC (convention over configuration) is the philosophy behind Ruby on Rails. This results in very little configuration is needed, if the default configuration of Ruby on Rails is sufficient. The models in rails consist of a "migration" file and a "model" file.

```
1 class CreateSensors < ActiveRecord::Migration
2   def change
3     create_table :sensors do |t|
4       t.text :name
5       t.text :description
6
7       t.timestamps null: false
8     end
9   end
10 end
```

Figure 5.2: Sensor migration file

Figure 5.2 shows a migration file of the type "sensor". The migration file consists of all code related to making changes to the database. All the attributes of the sensor can be seen as a part of the migration file. If you were to change the an

attribute of the sensor it should be done in this file. The model file is where you would manipulate data from the database, without changing the data in the database. Validation of data input has not been implemented yet, but it should be in the future. Data validation is very important to avoid invalid data in the database. Data validation needs to be handled in the model file.

All the models have been generated in the terminal using the command *"rails generate model"*. Using the terminal will automatically generate the corresponding migration file as well.

5.2.1.2 View

In rails the views is also called "Action View". An Action View can be written in many different ways. In the thesis all the Action Views consists of "erb" files. An erb file is a mixture of HTML code and Ruby code. All code between "`<% %>`" is ruby code. Ruby will render the erb file as a HTML file so a browser can interpret the view.

```

1 <div class ="sensors">
2   <table class="defaultTable">
3     <tr>
4       <th>Name</th>
5       <th>Description</th>
6       <th></th>
7     </tr>
8     <% @sensors.each do |sensor| %>
9       <tr>
10        <td><%=sensor.name%>(<%=sensor.id%>)</td>
11        <td><%=sensor.description%></td>
12        <td><%= link_to 'Delete', 'sensors/delete/'+sensor.id.to_s, data: { confirm: 'Are you sure?' }
13        ,class: 'link_delete' %></td>
14      <%end%>
15    </tr>
16  </table>
17 </div>
18 <%= link_to 'New sensor', 'sensors/new', class: 'link_new'%>

```

Figure 5.3: erb file of sensor index Action View

Figure 5.3 shows the erb file view, responsible for displaying all sensors in the system. Ruby converts the erb file to the HTML file seen on figure 5.4.

All the views has the same navigation menu seen on figure 5.5. To follow the DRY (Dont Repeat Yourself) principal mentioned in the design chapter, all views share the same navigation menu. This has been achieved by creating an erb file only containing the navigation menu. In the beginning of each view the erb file, containing the navigation menu, is loaded. This results in only having

```

1 <div class ="sensors">
2   <table class="defaultTable">
3     <tr>
4       <th>Name</th>
5       <th>Description</th>
6     </th></th>
7   </tr>
8     <tr>
9       <td>s1(1)</td>
10      <td></td>
11      <td> <a data-confirm="Are you sure?" class="link_delete" href="sensors/delete/1">Delete</a></td>
12    </tr>
13      <td>s3(3)</td>
14      <td></td>
15      <td> <a data-confirm="Are you sure?" class="link_delete" href="sensors/delete/3">Delete</a></td>
16    </tr>
17      <td>testSensor(4)</td>
18      <td>vi tester med denne sensor</td>
19      <td> <a data-confirm="Are you sure?" class="link_delete" href="sensors/delete/4">Delete</a></td>
20    </tr>
21      <td>lux-måler(6)</td>
22      <td>måler lux</td>
23      <td> <a data-confirm="Are you sure?" class="link_delete" href="sensors/delete/6">Delete</a></td>
24    </tr>
25      <td>appInputs(1337)</td>
26      <td></td>
27      <td> <a data-confirm="Are you sure?" class="link_delete" href="sensors/delete/1337">Delete</a></td>
28    </tr>
29  </table>
30 </div>
31 <a class="link_new" href="sensors/new">New sensor</a>

```

Figure 5.4: HTML of sensor index Acion View

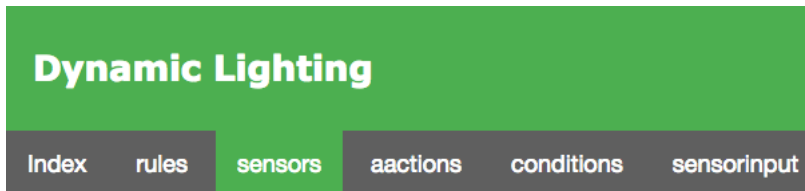


Figure 5.5: Navigation menu

to change one erb file, and not all erb files, when the design of the navigation menu is changed.

CSS also referred to as Cascading Style Sheets is used for styling the views. The actions of pressing a button or navigation with menu is handled by JavaScript using the framework jQuery. jQuery was chosen because its the default JavaScript framework in Ruby on Rails.

5.2.1.3 Controller

In Ruby on Rails called "routes.rb". The routes file decides which controller receives which request. This thesis uses mostly RESTful routing which is the

current standard for routing in Rails[rub].

```
resources :rules
```

The line of code seen above shows an example of RESTful routing in the thesis. RESTful routing is easy to read for developers and only requires 1 line of code. This single line of code creates 7 different routes in the application. Ruby on Rails will by default link the routes to the controller called `"rules_controller.rb"`

HTTP verb	URL	controller	method	used for
GET	/rules	Rules	index	display a list of all rules
GET	/rules/new	Rules	new	return an HTML form for creating a new rule
POST	/rules	Rules	create	create a new rule
GET	/rules/#	Rules	show	display a specific rule
GET	/rules/#/edit	Rules	edit	return an HTML form for editing a rule
PUT	/rules/#	Rules	update	update a specific rule
DELETE	/rules/#	Rules	destroy	delete a specific rule

Figure 5.6: The 7 routes created by RESTful routing. "#" represents a number

Figure 5.6 shows all the 7 different routes created. The figure shows the different HTTP-requests and what method in the rules controller they are linked to. All these routes can also be created manually without using RESTful routing. It is not recommended to create the routes manually, this might confuse other developers. A few routes have been created manually in the thesis, but this is to be changed in the future.

The "rules" controller can be seen on figure 5.7. If the web server receives a GET-request with the following relative URL `"/rules/new"`, the routing file will pass the request to the rules controller. The rules controller will then execute the method "new".

5.2.2 Automation system

The automation system is following the flow covered in the design chapter. The automation system is handled by the controller called "sensor_input_controller". This controller handle all inputs send from a sensor. The web server also sees

```

1  class RulesController < ApplicationController
2    def index
3      @rules = Rule.all
4    end
5
6    def new
7      @rule = Rule.new
8      @actions = Aaction.all
9    end
10
11   def create
12     @rule = Rule.new(actor_params)
13
14     if @rule.save
15       redirect_to '/rules'
16     else
17       render 'new'
18     end
19   end

```

Figure 5.7: Part of the "rules" controller

the controller app as a sensor, so all input from the controller app is being routed to the "sensor_input_controller".

```

1  def receiveSensorInput
2    value = params[:value]
3    sensor = Sensor.find(params[:id])
4    sensor.update_attribute(:value, value)
5
6    if(sensor.id == 1337)
7      start_app_action(value)
8      logger.info "action from app: "+value
9    else
10     start_condition_checker
11   end
12   ...

```

Figure 5.8: "receiveSensorInput" method

Figure 5.8 is the method that handles POST-requests from sensors. It can be seen that the sensor with the id of "1337" is handled differently than other sensors. This is to distinguish between a sensor and the controller app. When a sensor is added to the web server the ActiveRecord (the model) handles everything and delegates the sensor with a sensor_id. The controller app has been added manually and given the sensor_id "1337". If the controller app sends an input, it will overrule the automation system and change the lighting settings. If the input doesn't come from the controller, the app will start the method "start_condition_checker".

In figure 5.9 part of the condition checker can be seen, the whole controller can be found in the Appendix. In the design chapter it was mentioned that only the

```

trigger = false
rules_in_play.each do |r|
  rule_id = r.first.rule_id.to_s
  r.each do |c|
    operator = c.operator
    operator_value = c.operator_value.to_i
    sensor_name = c.sensor.name
    sensor_value = c.sensor.value.to_i
    if operator == '>' && sensor_value > operator_value
      trigger = true
    elsif operator == '<' && sensor_value < operator_value
      trigger = true
    elsif operator == '=' && sensor_value == operator_value
      trigger = true
    else
      trigger = false
      break
    end
  end
end

```

Figure 5.9: "start_condition_checker" method

rule related to the sensor, for a given input, would be examined. It has been implemented so all rules will be examined when receiving a sensor input. Figure 5.9 show how each rule is evaluated. For every rule each condition belonging to that rule will be examined 1 by 1. If a condition is "true" the next condition will be evaluated. If there are no conditions left to evaluate and the "trigger" is true the associated Action will be executed as seen on figure 5.10. On the other hand if a condition is "false", the "trigger" will be set to false and the next rule will be evaluated.

```

Action.all.each do |a|
  if triggered_rules.include? a.rule.id.to_s
    logger.info 'trigger name: '+a.actor.name+' config:'+a.config

    # uri = URI('http://192.168.0.15/api/ZeydomqLv4VhSIBga8Est4ksSNTTcjwjcpTRX9GA/lights/1/state') #home
    uri = URI('http://192.168.1.102/api/ZeydomqLv4VhSIBga8Est4ksSNTTcjwjcpTRX9GA/lights/3/state')
    req = Net::HTTP::Put.new(uri, 'Content-Type' => 'application/json')
    # req.body = {on: false}.to_json
    req.body = a.config
    res = Net::HTTP.start(uri.hostname, uri.port) do |http|
      http.request(req)
    end
  end
end

```

Figure 5.10: Code responsible for executing an action

When an Action is executed the JSON file related to the rule, will be send to the lighting system as a PUT-request.

5.2.3 Database

The database is described in the design chapter. Ruby on Rails is a framework built around the CoC philosophy. This means that Ruby on Rails provides default settings for almost everything including the database. The default database in Ruby on Rails is "SQLite 3" and requires no additional configuration. SQLite 3 is used as the database for the web server. It can be argued that more people know how to use MySQL than SQLite, but when working with Ruby on Rails the developer barely notices what database is used. This is because of the ORM used in Ruby on Rails called "ActiveRecords". An ORM is short for "Object Relational Map" and helps the developer to interact with the database. The developer don't manually have to make SQL calls to the database. Using ActiveRecord means that the developer don't have to know SQLite 3 to interact with the database.

$$SELECT * FROM sensor \tag{5.1}$$
$$Sensor.all \tag{5.2}$$

Equation (5.1) shows a SQL statement fetching all sensors from the database, while equation (5.2) shows how ActiveRecord is used to fetch all in Ruby on Rails. As seen in equation (5.2) ActiveRecord handles all database interaction with ruby as ruby-objects. Without ActiveRecord converting the data from the database to an object, it would have to be done manually.

The database diagram seen on figure 5.11 represents the database implemented on the web server. All the 4 tables are related to the automation system. It can be seen that it is smaller than the one found in figure 4.6 in the design chapter. The implemented database does not contain the following tables:

- users
- user_types
- actors
- actors_types
- lighting_profiles

The users and user_types tables are to be added in the future. The relevance of the actor table depends on whether or not multiple lighting systems are needed.

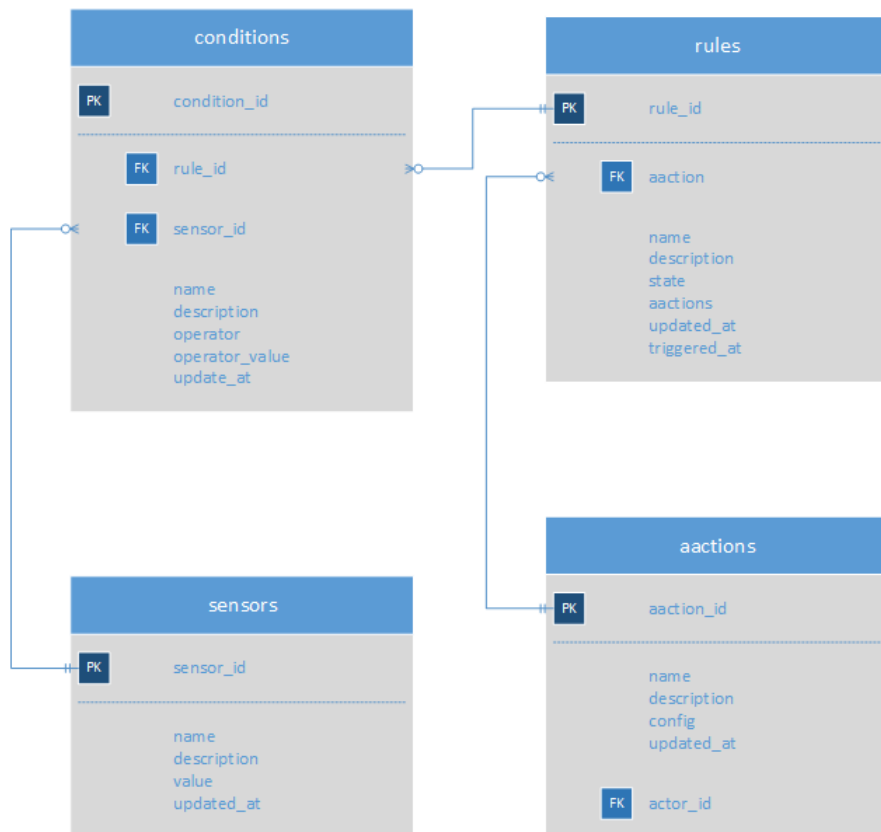


Figure 5.11: Web server database

As proof of concept these tables are not required, but having multiple users is important so the library can manage who has access to the data and to make changes to the system.

It can be seen that "state" is part of the rules tables, but this only a part of the database and has not yet been implemented on the web server. The attribute "actor_id" can be seen as a column in the aactions table. Since the actors table is not used by the web server yet the "actor_id" serves no purpose at the moment. This also means that all aactions is sent to the Phillips hue lighting system.

5.3 Front-end

Besides the web server the system consists of two different interfaces, an administration page and a controller app. The administration page is used to control the lighting automation system. The controller app is used to manually change the automated lighting settings from a smartphone. Any input from the smartphone app will override the automated lighting settings. A sensor app has also been created. The sensor app has been created to test the automation system and to showcase how a sensor can communicate with the web server and the automation system. The controller app and the sensor app are part of the front-end section, because the apps just send data to the web server and are not supposed to handle any logic. Both apps however does have some logic which will be described in the coming sections.

5.3.1 Controller App

The controller app is used to personalise the lighting settings by overriding the automation system. The app is written in Java using the official Android IDE called "Android Studio". In the design chapter it is mentioned that the controller app should communicate with the lighting system through the web server. In the current version of the app the communication is only one way. The controller app sends data to the web server but no data is returned to the app. This results in the controller app not knowing the state of the lighting system. The controller app is therefore not able to show the current selected profile or the brightness level of the lighting system.

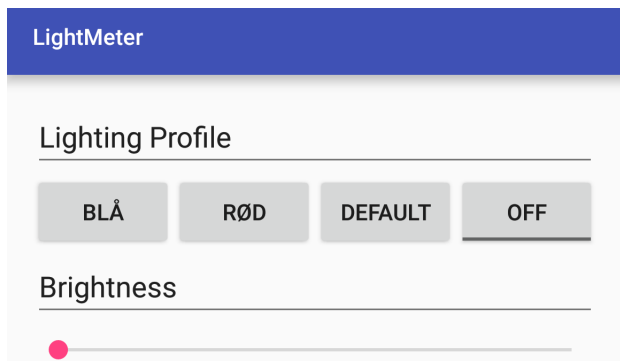


Figure 5.12: Screenshot of the controller app

Figure 5.12 shows the current version of the controller app. In figure 5.12 it can

be seen that there are 4 profiles.

```

1 | onOff.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
2 |     @Override
3 |     public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
4 |         if(!isChecked){
5 |             onOff.setText("off");
6 |             sendRawPostRequest("{\"on\":true}", ipText.getText().toString());
7 |         }else{
8 |             onOff.setText("on");
9 |             sendRawPostRequest("{\"on\":false}", ipText.getText().toString());
10 |        }
11 |    }
12 | });
13 |
14 | resetButton.setOnClickListener(new View.OnClickListener() {
15 |     @Override
16 |     public void onClick(View v) {
17 |         sendRawPostRequest("{\"on\":true,\"xy\": [0.4448,0.406],\"bri\":254,\"sat\":121,\"hue\":15342}", ipText.getText().toString());
18 |     }
19 | });
20 |
21 | redButton.setOnClickListener(new View.OnClickListener() {
22 |     @Override
23 |     public void onClick(View v) {
24 |         sendRawPostRequest("{\"on\":true,\"xy\": [0.3689,0.3719],\"bri\":254,\"sat\":53,\"hue\":33016}", ipText.getText().toString());
25 |     }
26 | });
27 |
28 | blueButton.setOnClickListener(new View.OnClickListener() {
29 |     @Override
30 |     public void onClick(View v) {
31 |         sendRawPostRequest("{\"on\":true,\"xy\": [0.5017,0.4152],\"bri\":144,\"sat\":200,\"hue\":13524}", ipText.getText().toString());
32 |     }
33 | });

```

Figure 5.13: Hardcoded profiles

Figure 5.13 shows the code of the 4 lighting profiles. It can be seen that all the profiles have been coded into the web server and can only be modified by a developer. When the web server can return data to the controller app, the profiles should be stored in the database connected to the web server. If stored in the database the administrators or Living Lab users can change the profiles.

The lighting profiles is written as a JSON file and send to the web server when selected by the user. The web server then uses the Phillips hue API and sends the lighting settings from the profile to the Phillips hue bridge.

```

{"on":true,"xy": [0.4448,0.406], "bri":254, "sat":121, "hue":15342}

```

Figure 5.14: JSON file readable by the Hue API

Figure 5.14 is the part of the code seen on figure 5.13 that uses the Phillips Hue API.

The app is written in java using the official Andriod IDE called "Android Studio".

5.3.2 Sensor App

The web server needs to be able to receive and respond to sensor inputs from various sensors. The sensor app is made for testing the platform and for proof of concept. The idea is that users of the Living Lab can create their own sensors and integrate it with the rule system found on the web server.

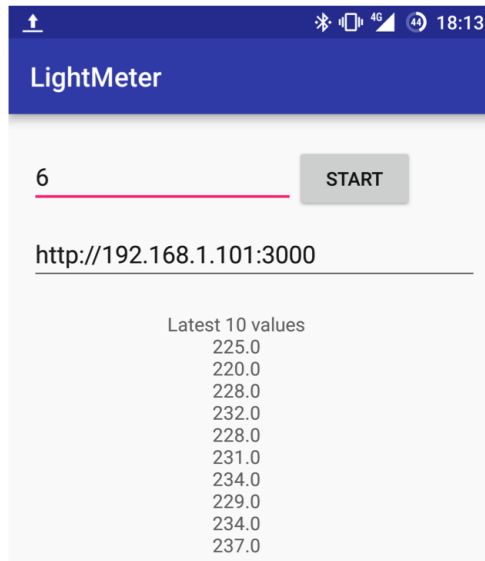


Figure 5.15: Light-meter app



Figure 5.16: The light meter on a sony smartphone

Figure 5.15 shows a screenshot of the developed sensor app. The app measures lux using the light-meter found on the front of most smartphones, see figure 5.16. The accuracy of the lux measured by the app is depending on the quality

of the light-meter installed in the phone.

The picture above is a screenshot of the sensor app. There is a numberField, a textField, a button and a textView. The number in the numberField represents the sensor-id found on the web server. If a sensor with the id has not been created on the web server the web server will ignore the input from the sensor. The textField contains an ip-address. The ip-address controls what server the sensor inputs are send to. This part would in most cases be hardcoded and not visible in the app. The reason for having a field for the ip-address is that it gives the option to change it. This is convenient because the web server is not running on a server with a static ip-address, so the ip-address on the web sever changes. If the system was to be implemented at the library, it should have a static ip-address. The textView displays the latest 10 sensor inputs.

The app is designed so it records every time there is a change in lux. If the sensor were to send a request for every sensor input recorded it would put a lot of pressure on the server. The server used during this thesis is a macBook pro from 2009. The server is able to handle the request made by one sensor, but if more sensors were added, the server might go down. To avoid too many request going to the server, the app only sends data every 5 seconds. This has been implemented by creating a "TimerTask"-thread running on the app.

As mentioned before all logic is supposed to be handled by the web server, but the controller does a bit of data manipulation. The app is designed so it records every time there is a change in lux. If something blocked the light-meter the lux measured by the sensor would drop. This would however not be a true representation of the light in the area. To avoid the app to send an input measured while blocked, the average of the 10 latest sensor recordings is send to the web server. Sending the HTML-requests is done using a android library called "Volley". "Volley" is a HTTP-library that makes networking easier and faster without having to write tons of code[Goo]. To use the build in light-meter a class called "LightMeter" has been created. The "LightMeter" class implements a "SensorEventListener" and uses the sensor of type "TYPE_LIGHT".

CHAPTER 6

Testing

Developing this system has been an agile process, so testing the software has been an on-going process. Black-box testing also known as functional testing[bla], has been the primary testing method. The UI (user-interface) has not been prioritized in the thesis, so none of the results from the test has been implemented. The automation system in combination with the controller app and sensor app has also been tested.

6.1 Black-box testing

The automation system is the most complex part of the system. To make it easy to test how the system reacts to a sensor input, a method to mimic a sensor input has been implemented.

The screenshot shows a web interface titled "Dynamic Lighting" with a green header. Below the header is a navigation bar with tabs for "Index", "rules", "sensors", "aactions", "conditions", and "sensorinput". The "sensorinput" tab is active. Below the navigation bar, the heading "create sensor input" is displayed. There are two input fields: "sensor id" and "sensor value". Below these fields is a "Post" button.

Figure 6.1: Web interface of sensor test function

Figure 6.1 shows how the interface of the sensor input test function. Pressing "Post" will send the sensor id and sensor value to the web server. Since this function is only made for testing, all feedback will be presented in the server log as seen on figure 6.2.

```

RULE TRIGGERED!
Rule ID: 10
  Action Load (0.3ms) SELECT "aactions".* FROM "aactions"
  Rule Load (0.1ms) SELECT "rules".* FROM "rules" WHERE "rules"."id" = ? LIMIT 1 [["id", 8]]
  Rule Load (0.1ms) SELECT "rules".* FROM "rules" WHERE "rules"."id" = ? LIMIT 1 [["id", 9]]
  Rule Load (0.4ms) SELECT "rules".* FROM "rules" WHERE "rules"."id" = ? LIMIT 1 [["id", 10]]
Action perfomed: Turn on light
Action config: {"on":true,"bri":100}
Redirected to http://192.168.1.101:3000/sensorinput
Completed 302 Found in 30ms (ActiveRecord: 5.1ms)

```

Figure 6.2: Output from web server, showing a rule being triggered after receiving a sensor input

The output seen on figure 6.2 is showing the log file of a rule being triggered after receiving a sensor input. After connecting the web server to the Phillips hue bridge, it was possible to get more than just a log file as feedback. With the bridge added it was possible to see the lighting change and use that as feedback as well. If the rule is triggered the feedback will be seen as a change in the lighting depending on the action related to the triggered rule. While testing the automation system with the Phillips hue, a flaw in the system was discovered. If the web server receives sensor inputs rapidly after each other, the light could start blinking. The solution only allows a rule to be triggered every two minutes. This solution is not ideal and should only be temporary. The problem is that in some occasions 2 minutes might be too long. Imagine the

system automatically turned the light off. Immediately after a person enters the room, and the lighting wont turn back on because a rule cant trigger more than once every 2 minutes. A more flexible solution must be created in the future. It was also discovered that a change of the lighting settings, can be very distracting if the change is to drastic. It should be prioritized to make a smooth transition between lighting settings, to avoid the distraction.

6.2 User interface

The usability of both the controller app and the web interface has been tested. The usability has not been prioritized, so only a simple user test was made. The test consisted of giving the user a list of tasks and observe the user while trying to accomplish each task.

As seen on figure 5.12 in the implementation chapter, the controller app is very simple and only consists of a slider and 4 profile buttons. The subject (user) were tasked to set the brightness to 100% and to change the profile. Both tasks were accomplished with no problems.

Using the web interface was very difficult and only very simple tasks was completed. The subjects were to complete the following tasks:

- Add a new sensor called "test sensor"
- Delete the sensor called "test sensor"
- Edit the name of the action called "sluk lys"
- Create a rule called "test rule", that turns on the light when the lux measured by the sensor "lx-meter" is below 50 lx

Navigating between the pages and adding, editing and deleting object operations was accomplished by all the subjects. None of the subjects understood how the sensor, rule, action and conditions are related.

The controller app is fairly simple and the subjects found it very easy to use. Unless new features are added the design doesn't need to be changed for the sake of usability. When it comes to setting up a rule in the system, no one was able to do so. In the future setting up rules would need to be redesigned.

6.3 Testing the system in a work situation

Besides the on-going software tests and the usability tests, the effect of the automation system and the controller app has been tested. The tests are performed in a living room used for both relaxing and for working. The subject works as an architect and works with a computer for both reading and drawing. Besides using the computer the subject also uses plain paper when creating design sketches. In the background chapter it was found that different types of work require different lighting settings. The subject both use the area for relaxing and for working with paper and with computer. The optimal test environment would be closer to the one found at the library and it would involve more people.

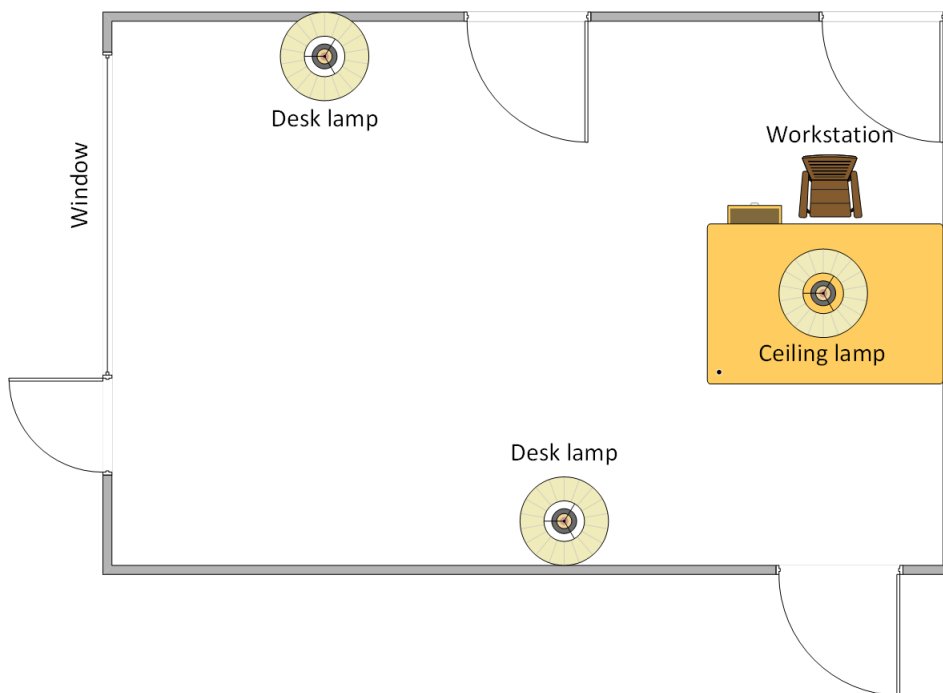


Figure 6.3: Illustration of the living room used for the testing

Figure 6.3 is an illustration of the area where the test was performed. The workstation is furthest away from the windows with a ceiling lamp. The ceiling lamp and the 2 other lamps in the room are using a Phillips Hue light bulb. If the automation system or the subject changes the lighting it affects all the lamps in the room. Two different tests were performed under these conditions:

- Both automation system and controller app in use
- Only controller app in use

6.3.1 Test 1 - Testing the automation system

For testing the automation system the implemented light-meter app was used. The sensor was placed near the window measuring the incoming light and was not affected by the artificial lighting coming from the lamps. The system was setup so it would decrease the brightness of the lamps if there were an increase in the natural lighting. If on the other hand the natural lighting were declining the lamps would increase in brightness. The subject was able to change the lighting using the controller app during the test. The goal with the test was to see how the subject would react on changes made by the automation system. The server recorded all inputs from the sensor app and controller app in a log file. It can also be seen when a rule is triggered and changes are made to the lighting settings.

```

Started POST "/sensorinput" for 192.168.1.102 at 2016-12-10 11:12:10 +0100
Cannot render console from 192.168.1.102! Allowed networks: 127.0.0.1, ::1, 127.0.0.0/127.255.255.255
Processing by SensorInputController#receiveSensorInput as HTML
Parameters: {"id"=>"1338", "value"=>"283.0", "sensor_input"=>{"id"=>"1338"}}
Sensor Load (0.2ms) SELECT "sensors".* FROM "sensors" WHERE "sensors"."id" = ? LIMIT 1 [{"id", 1338}]
(0.1ms) begin transaction
SQL (0.4ms) UPDATE "sensors" SET "value" = ?, "updated_at" = ? WHERE "sensors"."id" = ? [{"value", "283.0",
["updated_at", "2016-12-10 10:12:10.421687"], ["id", 1338]}]
(2.6ms) commit transaction
SENSOR INPUT RECEIVED
sensor name(id): Light-meter App(1338)
sensor value: 283.0
...
RULE TRIGGERED!
Rule ID: 10
Aaction Load (0.3ms) SELECT "aactions".* FROM "aactions"
Rule Load (0.1ms) SELECT "rules".* FROM "rules" WHERE "rules"."id" = ? LIMIT 1 [{"id", 8}]
Rule Load (0.1ms) SELECT "rules".* FROM "rules" WHERE "rules"."id" = ? LIMIT 1 [{"id", 9}]
Rule Load (0.4ms) SELECT "rules".* FROM "rules" WHERE "rules"."id" = ? LIMIT 1 [{"id", 10}]
Action performed: Turn on light
Action config: {"on":true,"bri":100}
Redirected to http://192.168.1.101:3000/sensorinput
Completed 302 Found in 30ms (ActiveRecord: 5.1ms)

```

Figure 6.4: A Section of the log file for test 1, showing a rule triggered by a sensor input

Figure 6.4 is a part of the log-file recorded by the web server. It can be seen that the web server receives an input from the "Light-meter App" with the value of 283 lx. It can be seen that the rule with Id 10 is triggered and an action called "Turn in light" is executed. This is action sets the brightness of the lamps to a 100. The maximum brightness of the Phillips hue bulbs used is 254.

In figure 6.5 it can be seen that the web server receives an input from the controller app. The input increases the brightness to 229. It can be seen that

```

Started POST "/sensorinput" for 192.168.1.102 at 2016-12-10 11:12:18 +0100
Cannot render console from 192.168.1.102! Allowed networks: 127.0.0.1, ::1, 127.0.0.0/127.255.255.255
Processing by SensorInputController#receiveSensorInput as HTML
Parameters: {"id"=>"1337", "value"=>{"bri":229}, "sensor_input"=>{"id"=>"1337"}}
Sensor Load (0.2ms) SELECT "sensors".* FROM "sensors" WHERE "sensors"."id" = ? LIMIT 1 [{"id", "1337"}]
(0.1ms) begin transaction
SQL (1.2ms) UPDATE "sensors" SET "value" = ?, "updated_at" = ? WHERE "sensors"."id" = ? [{"value", "{"bri":229}"}, {"updated_at", "2016-12-10 10:12:18.408264"}, {"id", "1337"}]
(2.5ms) commit transaction
Controller app action: {"bri":229}
ACTION FROM APP
sensor value: {"bri":229}

```

Figure 6.5: A Section of the log file for test 1, showing an input from the controller app

the request from the controller app is made 8 seconds after the web server dimmed the light to a brightness-value of 100. The subject dimmed the lighting multiple times right after the lighting was dimmed.

Event	Time	Configuration
Rule 10 triggered	11:12:10	"bri":100
Action from app	11:12:18	"bri":229
Rule 10 triggered	11:14:15	"bri":100
Action from app	11:14:27	"bri":230
Rule 10 triggered	11:16:20	"bri":100
Action from app	11:16:29	"bri":235
Rule 10 triggered	11:18:25	"bri":100

Figure 6.6: A table showing all events changing the lighting settings

Table 6.6 above shows how many times a rule was triggered and how many times the subject used the controller app to change the lighting. It can be seen how the subject, in all cases except 1, increased the brightness shortly after the web server dimmed the lighting. It is worth to notice that the subject was not present in the room the last time the lighting was dimmed. After the last change made by the automation system the test ran 5 minutes.

The automation system works as it should. The sensor app sends a value to the web server and if a rule is triggered the lighting is changed accordingly to the action related to the rule. The controller app overrides the rule system and correctly changes the lighting. If the subject was present in the room while the lighting was dimmed, the subject used the controller app to increase the brightness every time. This could indicate that drastically dimming the lighting

is not ideal if a person is in the room.

6.3.2 Test 2 - Testing the controller app

A separate test for the controller app was made. In this test the automation system was turned off. The lighting settings could only be changed by the subject. As mentioned the subject works both with a laptop and with drawing on paper. In the background chapter it was found that different types of work require different lighting settings. The goal was to see if the subject would use the controller app while working and change the lighting depending on the type of work. In the log-file it can be seen how multiple changes are made in the beginning of the test. These changes were made by the subject to test the different lighting profile in the app. The test lasted 1 hour and 5 minutes. During the test the controller app was only used once, when the subject stopped working.

```
Started POST "/sensorinput" for 192.168.1.102 at 2016-12-10 13:23:20 +0100
Cannot render console from 192.168.1.102! Allowed networks: 127.0.0.1, ::1, 127.0.0.0/127.255.255.255
Processing by SensorInputController#receiveSensorInput as HTML
Parameters: {"id"=>"1337", "value"=>{"\on\":true,\xy\":[0.5017,0.4152],\bri\":144,\sat\":200,\hue\":13524}"},
 "sensor_input"=>{"id"=>"1337"}}
Sensor Load (4.9ms) SELECT "sensors".* FROM "sensors" WHERE "sensors"."id" = ? LIMIT 1 [{"id", 1337}]
(0.6ms) begin transaction
SQL (0.6ms) UPDATE "sensors" SET "value" = ?, "updated_at" = ? WHERE "sensors"."id" = ?
[["value", {"\on\":true,\xy\":[0.5017,0.4152],\bri\":144,\sat\":200,\hue\":13524}"],
 ["updated_at", "2016-12-10 12:23:20.610576"], ["id", 1337]]
(7.0ms) commit transaction
Controller app action: {"on":true,"xy":[0.5017,0.4152],"bri":144,"sat":200,"hue":13524}
ACTION FROM APP
```

Figure 6.7: A section of the log file from test 2, showing the subject changing lighting profile

Figure 6.7 shows the only time the subject used the controller app. The subject used the controller app to set the lighting profile to "relax". this mode has a low brightness and is warmer than the default lighting. The subject pointed out that "relax" profile did not do much at the given time since the natural lighting out shined the one from the bulbs while in "relax" mode. It could be interesting to look at the possibility to have "dynamic lighting profiles", so the profiles depend on the season and the time of the day.

The controller app worked as intended, though it was used very little. The subject didn't change the lighting setting when switching between working on the laptop and drawing on paper. This could be because the lighting was fine for both conditions, it could be "laziness" or the subject simple forgot it was possible.

Discussion and Conclusion

The aim of this thesis is to create a prototype platform that can enable the DTU library to improve the indoor climate by facilitating the living lab in conducting experiments. The platform will not increase the indoor climate by itself, but it can help other researchers to do so. The platform lets researchers set up an automation system that can control the lighting in the library. The platform provides researchers with data from the environment, which can help them to decide how the lighting should be in any given situation. In this thesis a light-meter app was created to demonstrate how the lighting can be controlled based on the Lux measured.

The web server developed is built in the Ruby framework Ruby on Rails using the MVC-architecture. The aim has been to create code that is easy for most developers to read. The architecture makes it easy for front-end developers to improve the design. A user test of the platform has been made. The test showed that the user interface used for handling the automation system is confusing for users. It is important to improve the interface, so others than the developer can manage the system.

The tests made cannot conclude whether or not an automation system can improve the indoor climate. However the automation system can, in combination with the controller app, help researchers to see how users react towards certain lighting settings. For example the tests indicated that dimming the light while

working can be annoying for users. Analysis of the log files showed that the test person increased the light every time she noticed it was dimmed. It is suggested to carry out more extensive user test in order to analyze whether users will increase the lighting if the dimming of light happens gradually over time and not immediately.

The automation system created is working and can be overruled by the controller app. There are a few limitations that should be handled in the future implementations of the platform. A temporary solution was created to prevent the lighting from changing more than once every 2 minutes. A more flexible solution should be created in order to give the researchers more control over how often the automation system can make changes in the lighting. The tests made also points to a flaw in the web server. The automation system should not be able to make changes in the lighting, shortly after a user has adjusted the lighting from the controller app.

At the moment researchers can only get information from the system by reading the log file on the web server. This is not a very user friendly solution. If researchers don't know what to look for in the log file, it will take them long time to make sense out of the data. The log file from the first test that lasted 11 minutes was more 5000 lines. Analyzing a log file from a whole day or maybe a week would take a very long time. Therefore the data should be stored in a database so it becomes manageable.

This thesis has proved that it is possible to develop a system that can automatically change the lighting based on sensor input, and at the same time let users override these changes. This is proved by developing a prototype of such a system. It is however recommended to make a thorough test of the impact of a lighting automation system on the indoor climate, before implementing the system in the whole library. It is also recommended to test each rule added to the automation system in a closed area before released in the library. The tests made as part of this thesis show that an automated lighting system can be disturbing for users, if not properly designed.

7.0.1 Future work

If the system developed is to be used in the DTU library more work is needed. Having multiple user types on the web server should be first priority. In the current state either you have full access or you do not have access at all. All users with access can adjust the lighting system, or possibly delete the configurations made by the administration.

One of the goals with the developed system is that it can be used to conduct experiments and collect data. However the data is currently not easily accessible. Before the collected data can be analyzed by people without a degree in computer science, the data need to be stored in a database and not in a log file. When the data is stored in a database you can start to think about how to export and possibly visualize data, to help researchers and students to analyze the data collected.

The user interfaces of the controller app and the web interface have not been prioritized in this thesis. If further developments in this area are considered, the web interface should be prioritized, since the user test indicated that this interface was found to be very confusing.

When it comes to let library guests control the lighting, the most important future improvement will be that users should be able to select which lamp they will adjust. Currently the controller app adjusts all the lamps, when changes are made by use of the app. It is not viable that a library guest can control all lamps in the entire library with the press of a single button, so this should be addressed in the future. The library is currently testing the possibility of installing beacons around the library. As described in the section "Pilot Project Area", a grid of beacons can be used to determine users' location inside a building. If the beacons are installed it could be interesting to look at possibilities to use the beacons as a part of the developed system. Knowing the indoor position of a user opens up for the possibility of letting the controller app automatically adjust the lamp closest to the user.

If beacons are installed in the library, it would also be interesting to look at the possibility to implement an intelligent lighting system. This could be done by finding patterns in the lighting preferences for each guest visiting the library and automatically adjust the lighting in the area where the guest is seated according to their personal preferences observed by the system. This would require the library guests to create a user profile, so the web server can identify each library guest.

APPENDIX A

System Sequence Diagrams

This appendix contains the system sequence diagrams used in the thesis in a larger version.

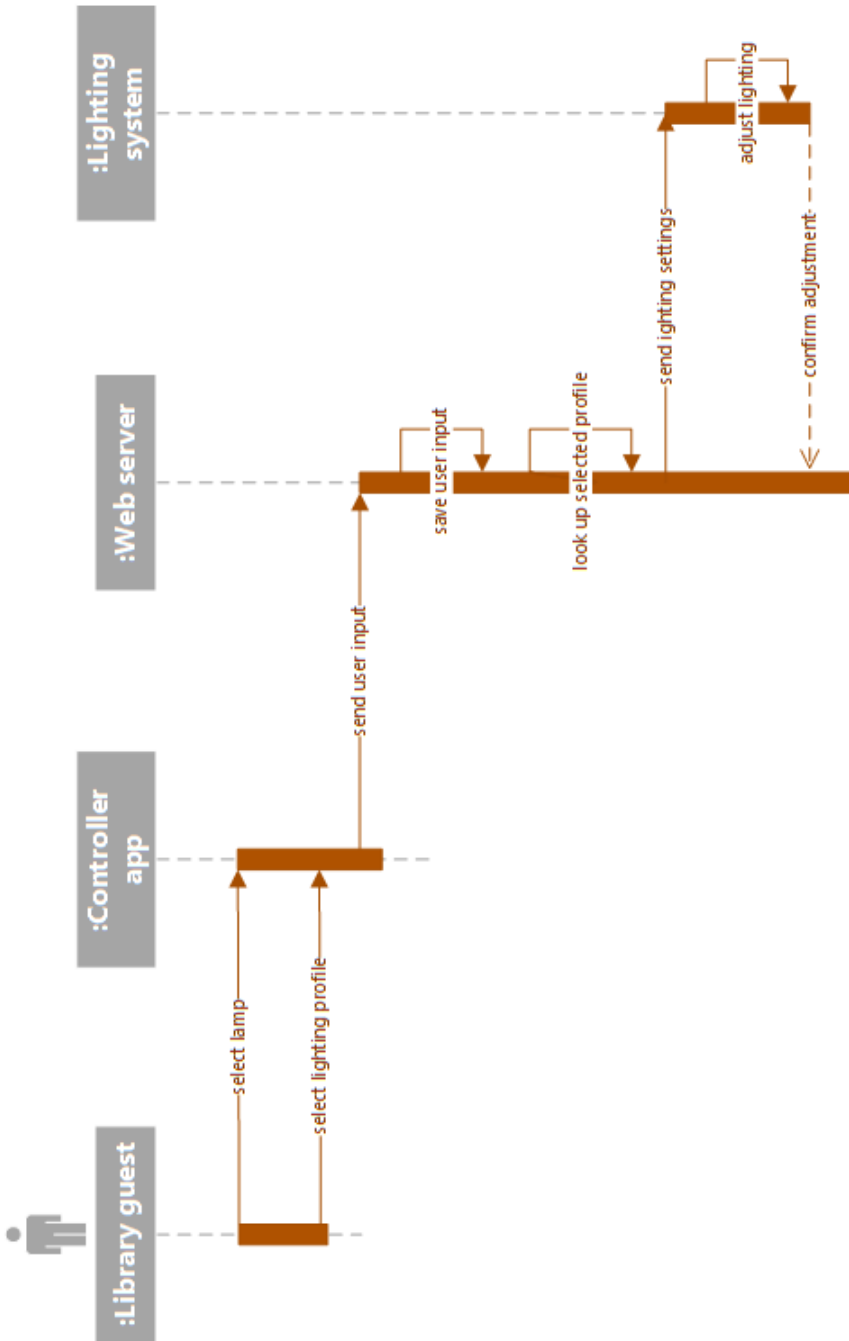


Figure A.1: Sequence diagram of a user adjusting the lighting

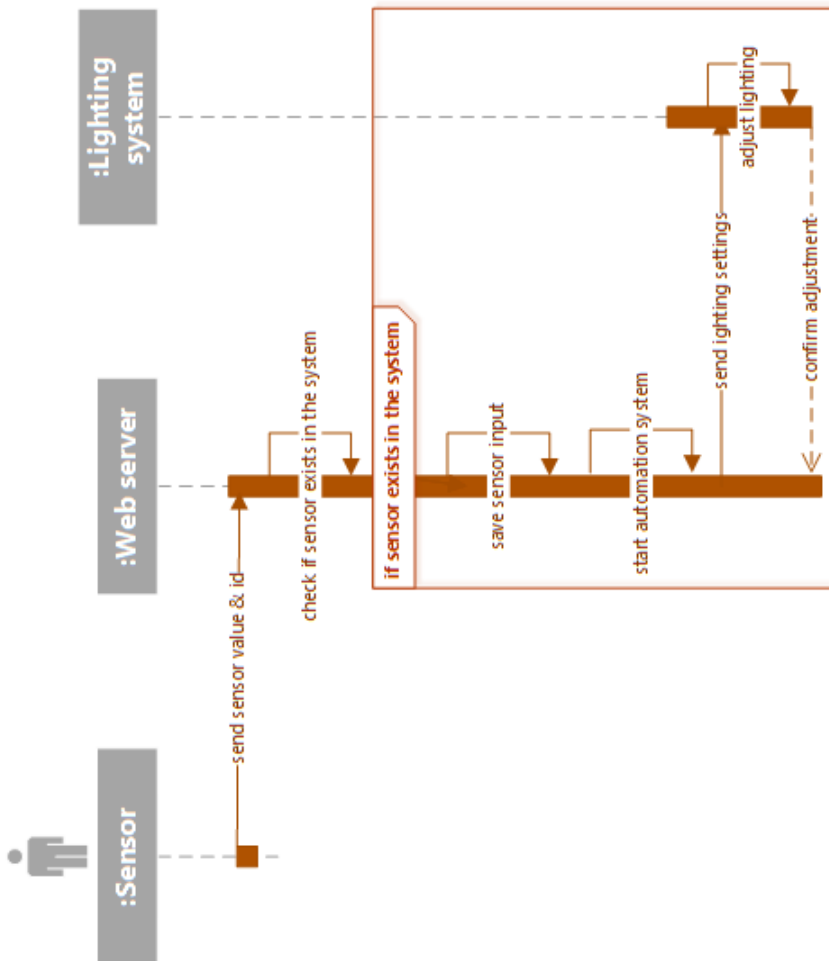


Figure A.2: Sequence diagram of the automation system

APPENDIX B

Administration Web Page

This appendix contains pictures of the administration page used to setup the automation system.

B.1 Index

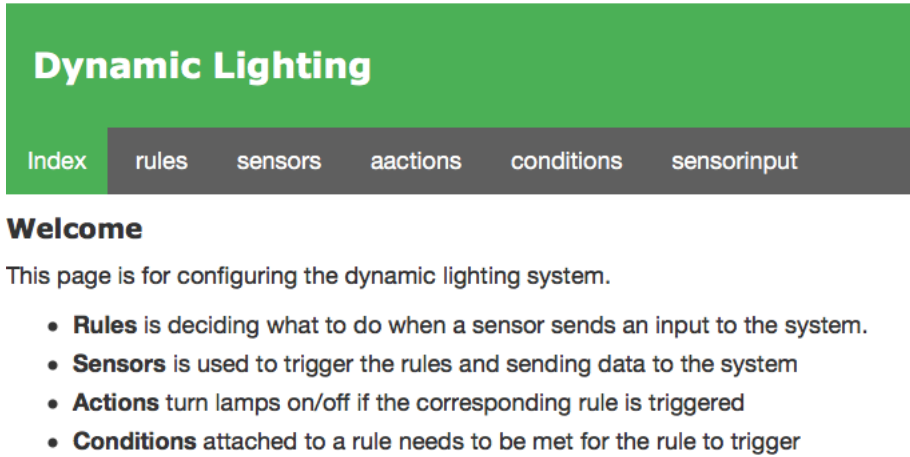


Figure B.1: Index page

B.2 Rules

Name	Description	State	Actions	Conditions	
Turn of light	Turn off light based on sensor input	1	Turn off light	Bright lighting	Delete
Increase brightness		1	Increase brightness	dark	Delete
Turn on light		1	Turn on light	turn on light turn on light	Delete
increase to 200					Delete

New rule

Figure B.2: Rule administration page

Name

Description

State

Conditiontype

Create

Figure B.3: Create rule

B.3 Sensors

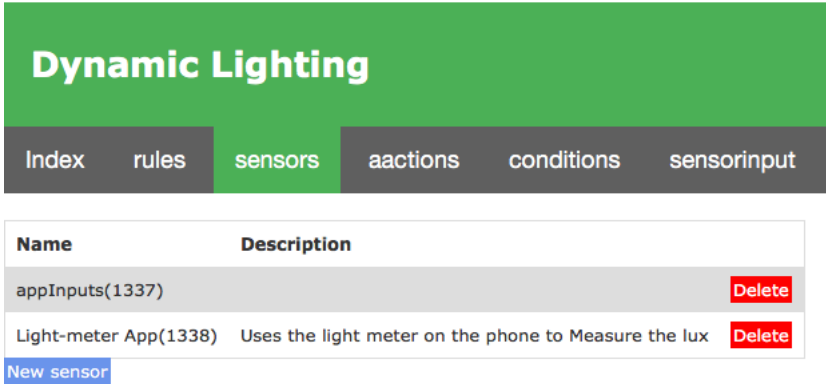


Figure B.4: Sensor administration page

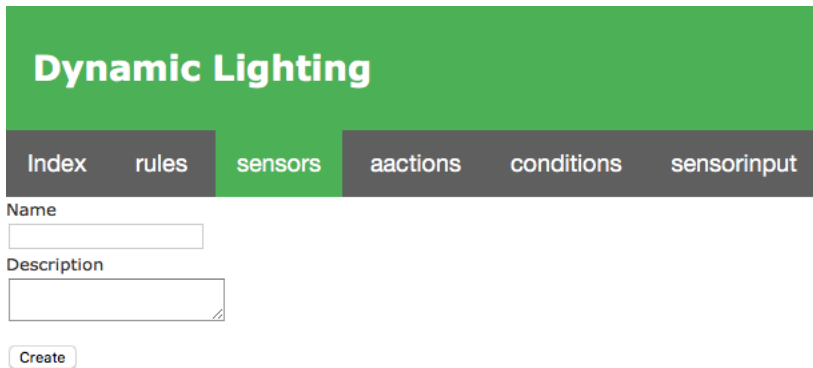


Figure B.5: Create sensor page

B.4 Aactions

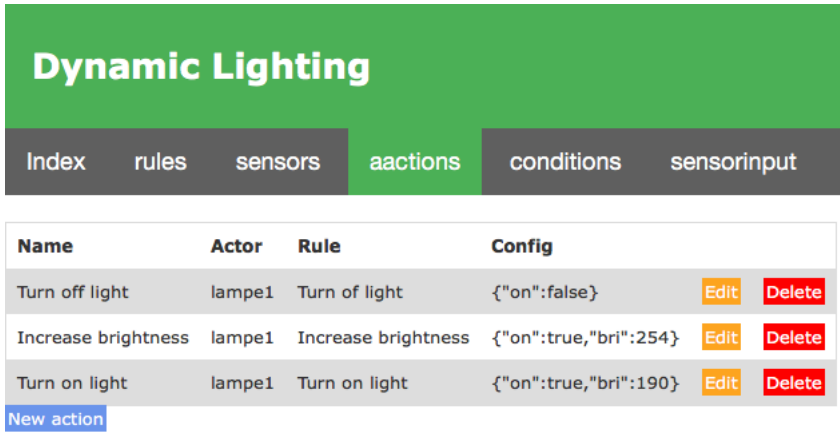


Figure B.6: Aactions administration page

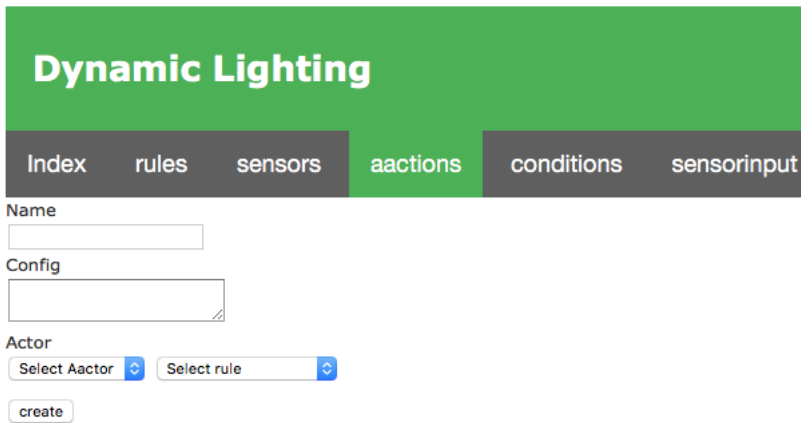


Figure B.7: Create action page

Dynamic Lighting

Index rules sensors **actions** conditions sensorinput

Name
Increase brightness

Config: simple advanced

on off

Actor
lampe1

Rule
Increase brightness

Update Aaction

Figure B.8: Edit actions page

B.5 Conditions

Name	Sensor	Rule	Condition		
Bright lighting	Light-meter App(1338)	Turn of light	x > 400	Edit	Delete
dark	Light-meter App(1338)	Increase brightness	x < 150	Edit	Delete
turn on light	Light-meter App(1338)	Turn on light	x < 250	Edit	Delete
turn on light	Light-meter App(1338)	Turn on light	x > 100	Edit	Delete

[New conditions](#)

Figure B.9: Conditions administration page

Name

Description

Operator

Operator value

Select Sensor ⌵

Select rule ⌵

Create

Figure B.10: Create condition page

Dynamic Lighting

Index rules sensors aactions conditions sensorinput

Name
Bright lighting

Description

Operator
>

Operator value
400

Light-meter App

Turn of light

Update Condition

Figure B.11: Edit condition page

Dynamic Lighting

Index rules sensors aactions conditions sensorinput

Name				
Bright lighting				Edit Delete
dark				Edit Delete
turn on light				Edit Delete
turn on light	Light-meter App(1338)	Turn on light	x > 100	Edit Delete

New conditions

Fra "http://localhost:3000":
Are you sure?

Annuler OK

Figure B.12: Delete condition dialog box

Bibliography

- [AKBH01] D H Avery, D Kizer, M a Bolte, and C Hellekson. Bright light therapy of subsyndromal seasonal affective disorder in the workplace: morning vs. afternoon exposure. *Acta psychiatrica Scandinavica*, 103:267–274, 2001.
- [Alm11] Wareham J. Almirall, E. Living labs: Arbiters of mid- and ground-level innovation. *technology analysis and strategic management*, 2011. Accessed: 2016-11-22.
- [App] Apple. About ibeacon on your iphone, ipad, and ipod touch. <https://support.apple.com/en-gb/HT202880>.
- [bla] Black-box testin. http://www.webopedia.com/TERM/B/Black_Box_Testing.html.
- [dKS10] Y. de Kort and K. Smolders. Effects of dynamic lighting on office workers: First results of a field study with monthly alternating settings. *Lighting Research and Technology*, 42:345–360, 2010.
- [doE] USA department of Energy. When to turn off your lights. <http://energy.gov/energysaver/when-turn-your-lights>. Accessed: 2016-10-18.
- [dtua] Dtu pilot project. http://www.smartcampus.dtu.dk/smartlibrary/pilot_project. Accessed: 2016-10-26.
- [dtub] Dtu smart library. <http://www.smartcampus.dtu.dk/Smartlibrary>. Accessed: 2016-10-18.

- [Fer] David Ferris. Can big data make a dumb building smart? <http://www.eenews.net/stories/1060032541>. Accessed: 2016-10-18.
- [Fis00] William J Fisk. HEALTH AND PRODUCTIVITY GAINS FROM BETTER INDOOR ENVIRONMENTS AND THEIR RELATIONSHIP WITH BUILDING ENERGY EFFICIENCY 1. 2000.
- [FJP16] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security Analysis of Emerging Smart Home Applications. pages 1–19, 2016.
- [Goo] Google. Transmitting network data using volley. <https://developer.android.com/training/volley/index.html>.
- [Haa] Harald Haas. Wireless data from every light bulb. http://www.ted.com/talks/harald_haas_wireless_data_from_every_light_bulb.
- [Hag] Barry Hagglund. Software-based control will personalize lighting and deliver broad benefits. <http://www.ledsmagazine.com/articles/print/volume-9/issue-6/features/software-based-control-will-personalize-lighting-and-deliver-broad-benefits-magazine.html>. Accessed: 2016-10-19.
- [huea] Active subreddit for phillips hue. <https://www.reddit.com/r/Hue/>. Accessed: 2016-10-20.
- [Hueb] Philips banned third-party bulbs. http://www.ledinside.com/news/2015/12/can_philips_lead_the_smart_lighting_industry_developments_with_hue_bridge. Accessed: 2016-10-18.
- [Huec] Philips hue motion sensor. <http://www2.meethue.com/en-us/motion-sensor>. Accessed: 2016-10-18.
- [Jen] Andrew Jensen. How office lighting affects productivity.
- [Ked] Kedington. Smart building management technologies easily justify the cost. <http://www.kedington.ie/kedington-intelligent-buildings/affordable-smart-building-management-technologies-easily-justify-the-cost/>. Accessed: 2016-11-22.
- [Lar] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*.
- [Lig] wigwag lightify review. <http://blog.wigwag.com/osram-lightify-led-recessed-dimmable-review/>. Accessed: 2016-10-20.

- [LM97] Carol Lomonaco and Dennis Miller. Comfort and control in the workplace. *ASHRAE Journal*, 39(9):50–56, 1997.
- [Log] Asta Logadóttir. A case study on occupant controlled lighting in offices. Proceedings of the 28th Session of the CIE, Manchester, United Kingdom, 28 June – 4 July 2015.
- [Log13] Iversen A. Markvart J. Corell D. D. Thorseth A. Dam-Hansen C. Logadóttir, Á. Comparison of user satisfaction with four different lighting concepts. *Publication Cie*. 2013.
- [Mar] Marketsandmarkets.com. Intelligent building automation technologies market by system services, it technologies industry verticals - global advancements, worldwide forecast analysis(2014 - 2019).
- [MD] MPH Martin Downs. Winter darkness, season depression. <http://www.webmd.com/depression/features/seasonal-affective-disorder#1>. Accessed: 2016-10-19.
- [mem16] memoori. How do smart buildings make a building green?, 2016. Accessed: 2016-11-22.
- [Mor] Jennie Morton. The dark side of poor lighting. <http://www.buildings.com/article-details/articleid/17149/title/the-dark-side-of-poor-lighting/viewall/true.aspx>.
- [OOB14] Halszka Oginska and Katarzyna Oginska-Bruchal. Chronotype and personality factors of predisposition to seasonal affective disorder. *Chronobiology international*, 31(4):523–31, 2014.
- [Pel] Matias Peluffo. Defining today’s intelligent building, 2015. <http://www.commscope.com/Blog/Defining-Todays-Intelligent-Building/>. Accessed: 2016-11-22.
- [pro] Osram lightify pro dse. https://www.osram.com/osram_com/tools-and-services/tools/lightify-smart-connected-light/lightify-pro-intelligent%2c-connected-light-for-professional-applications/lightify-pro-products/lightify-pro-dse/index.jsp. Accessed: 2016-10-20.
- [RKA] Ph.d. Center for indeklima og energi Rune Korsholm Andersen.
- [rub] rubyonrails.org. Rails routing from the outside in. <http://guides.rubyonrails.org/v2.3.11/routing.html>.
- [Sin07] Jim Sinopoli. The internet of things in smart commercial buildings, 2007. Accessed: 2016-11-22.

- [SW01] M.J. Skelly and M.a. Wilkinson. The evolution of interactive facades: improving automated blind control., 2001.
- [SW13] Anna Steidle and Lioba Werth. Freedom from constraints: Darkness and dim illumination promote creativity. *Journal of Environmental Psychology*, 35:67–80, 2013.
- [thi] Smartthings limitations. <https://community.smartthings.com/t/100-device-limit-issue/41900>. Accessed: 2016-11-2.
- [Tor] Timothy Torres. The best smart light bulbs of 2016. <http://uk.pcmag.com/digital-home/41566/guide/the-best-smart-light-bulbs-of-2016>. Accessed: 2016-11-22.
- [Woj] Emily Wojcik. American psychological association - better lighting, better work. <http://www.apa.org/gradpsych/2012/03/odd-jobs.aspx>. Accessed: 2016-10-19.
- [You10] Ernst Young. Global trend 2: increasing focus on resource efficiency and climate change. <http://www.ey.com/gl/en/issues/business-environment/business-redefined---global-trend-2--increasing-focus-on-resource-efficiency-and-climate-change>, 2010. Accessed: 2016-11-22.