Student Dormitory Distribution

- P2-Project -

Project Report
A218b

Aalborg University
Department of Computer Science

Google Drive, GitHub, Bootstrap, Discord, Trello

Department of Computer Science
Aalborg University
www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Student Dormitory Distribution

Theme:
xxx

Project Period:
Spring Semester 2022

Project Group:
A218b

Participant(s):
Banafsheh Nourizadehbarabi
Thomas Bjeldbak Madsen
Tobias Christian Hansen

Supervisor(s):
Salahuddin Abdul Rahman

Copies: 1

Page Numbers: 85

Date of Completion:
May 25, 2022

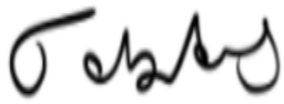
Abstract:

Lack of student dormitory distribution system complicates finding accommodation as a student. Even if they find a place to live, other factors as student environment can have an impact on the students mental health and their academic performance. This report has been made because of the P2-project at Aalborg University. How can a software solution be made for students, domestic and international, to help them find the most compatible roommate based on personal preferences? The group researched the impact of matching with the most compatible roommate, and how an implemented platform can be beneficial for students. It can be concluded that the development of a software solution matching students based on preferences creates a sufficient solution to the problem presented in the problem statement.

Contents

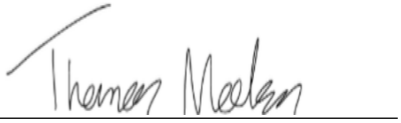
Preface

Aalborg University, May 25, 2022



Tobias Christian Hansen
<tcha21@student.aau.dk>

Banafsheh Nourizadehbarabi
<bnouri19@student.aau.dk>



Thomas Bjeldbak Madsen
<tbma21@student.aau.dk>

Chapter 1

Introduction

Whether it be in an international or merely danish sense, the degree to which students are choosing to go university are steadily increasing, which is nothing new.
It is often seen that university students resort to live with one or multiple roommates, which for some may not create the ideal environment for learning and self growth. However, the methods of searching and ultimately finding a roommate has not ever seen much change.
This may result in unnecessary frustration caused by not being able to access what you know is available, further incentivizing the idea behind a method of matching roommates.
The report examines the problem that there exists no service providing matching of roommates based on preferences and in turn presents a possible solution in Denmark. This is achieved with an algorithm comparing a potential dormitory applicant to existing residents from different dormitories.

Chapter 2

Problem Analysis

To have a general understanding about the problem, it is necessary to investigate the causes and effects of the problem from different aspects. Doing this will help find a more reliable and also optimized solution.

2.1 Problem definition

Finding accommodation is one of the first things that students think about, after deciding to study at a university. If a person decides to live far from their families or to study in a city or a country, that is different from where they used to live, this is then of course especially prevalent.
Traditional on-campus housing does not exist in Denmark as it does in places such as the US [14], but students have other options, such as renting a room, sharing an apartment or house, student dorms or halls of residences called "Kollegier" [14].
Cost of living for international students in Denmark has been calculated to be around 800-1200 Euro/month[31]. This amount is even higher in the capital city. Students need to use almost one-third of this amount for their accommodation, because the costs of accommodation are between 400-670 Euro/month and even more in the bigger cities. Considering the rather high costs of living in Denmark and knowing the fact that the demand for student housings are much higher than the available options [6], increases the competition for gaining access to dorms, since these are the cheaper options making them more popular among students, despite the students not always knowing who or what they are walking into. Exactly this uncertainty of not knowing who you are going to spend a considerable amount of time with, is why we feel the need for a solution such as our own.
Current examples of methods for finding a suitable place to live in Denmark are searching in the related housing portals and looking in local Facebook groups. In the industry of student housing, scams are common place and therefore students must be more careful during the rental process[17]. Alongside this, the process

of applying for residency is quite time consuming and sometimes costly. The current available portals only give the students a chance to find the proper housing according to their budget and the size of the housings. However, these are not the only factors that meet a students needs. Characteristics of the people whom students are going to live with, like roommates and neighbors, their personality, gender, nationality, field of study etc. can change a student's quality of life for the better or worse. Every student has different preferences, so having a chance to live around people that match their personal preferences can make a students study experience much better[9]. As will be seen later in the report, students can gain significant advantages in different aspects of life based on the people they surround themselves with on a regular basis.

2.2 Initial problem

In these modern times, higher education has in a broader scope increased over the years incrementally as well as drastically, both overseas and in Denmark [30], [13]. This makes it more important than ever to have any kind of system set up to properly sort a large amount of people applying for the same types of positions, and even placing them together where it could be beneficial to do so. Some services already make an attempt at this, using quizzes to find people to house with, but these are generally limited in nature and typically cannot be found in Denmark.
Incorporating such things as personality tests and other personalized testing to test for compatibility in students is not a concept that is very well explored in the danish space, with self checkout-like services such as AKU-Aalborg or at least similar services for their particular area which is what most people use to find a place to stay during their university tenure.
As such, while there are many options available to a student studying at a university level in terms of finding such places, you are not given much of a jump-start in terms of finding a compatible roommate or if you are studying abroad and have just come to Denmark, with most if not all of the options available requiring user interaction that needs to happen in a language that the individual student may not be proficient in. The fact that such things like multiple survival guides for navigating the do's and dont's of finding proper accommodation [21], [17] exist, describes part of a problem that has its place in the current sphere. By making it easier for individuals to find an appropriate roommate based on their individual needs and thus accommodation that provides the environment where the individual can best achieve their goals, while also being simple and easy to understand, will drastically reduce the process that an individual will need to go through, thus letting up on resources both on their part and their would-be governmental help should they get stuck in the process.
This is not to mention the potential social and academic benefits that may ensue

from these students being paired with just the partner that matches them as well as possible. While the benefits of such pairings have not been readily studied and researched, you could assume that such a pairing more often than not would be beneficial to the individual union, as clashes of personality and its underlying complexities would be minimized by grouping together the most compatible individuals [29].

2.3 State of the Art

A State-of-the-art analysis refers to how much a certain field/area of interest has been developed; in our case it is accommodation and housing with focus on how you best find a good pairing of roommates. A State-of-the-art analysis is developed with the intention of clearly defining the problem at hand by taking a look at the existing solutions, hereby justifying the relevance and importance of the problem and present examples of solutions given that are similar. A State-of-the-art analysis also helps broaden our outlook beyond just our own project.

The existing solutions looked at more in depth below are AKU-Aalborg, Ungdomsboliger and Studee, all of which are aimed at younger people and/or students, with varying degrees of specifying search and preferences. We also decided to investigate international applications and websites that have to do with accommodation and housing, such as ROOMIE, RoomieMatch and ROOMATERS. These will however not be looked at more in depth. After studying the relevant solutions, the main focus points of every solution are set up and listed below.

- Which kind of roommate they want to live with; a transfer student, current upperclassmen or a first year student,
- If they want to live with someone from their own major,
- What are their study habits,
- When they normally go to bed,
- If they want to be roommate with someone who smokes,
- If they want to be roommate with someone who is from another country,
- The type of music students prefer,
- How often they have visitors,
- What they do in their free times,
- Average time they study weekly,

- Methods of dealing with disagreements,
- Students attitudes towards the personal belongings,
- And if they have ever been convicted a felony.

Additionally it is also noted that some services allow an applicant to choose their roommates directly with entering their full name, if they have any pre-selected roommates [11]. An example of this could be The Louisiana State University (a public land-grant research university in Baton Rouge, Louisiana). The system at LSU works by assigning a PAWS ID to each student, which can then be used to specify your roommate. The portal of LSU also has another option that suggests the user several choices based on students profile information such as age, classification and gender. After the students have found their roommates, they will be redirected to select the type of rooms and beds from the existing options [27].
The services looked at until now, have all been accessible through website, but as apps grow evermore prevalent, relevant apps are also looked at, such as; ROOMIE, ROOMSTER, RoomEasy, and SpareRoom. All of these apps provide students the option to find roommates based on the city they are going to study in [1].
BUNKUP, CIRCLE, CITRU, RoomieMatch and DIGGZ are other options to find accommodations, where some priorities like gender, sexual orientation, lifestyle choices and pet-friendliness are also included [1].
Compared to the other cases, ROOMATERS comprises more factors and preferences (like interests and hobbies), and it also uses a personality test to find the best matching roommate [1].

2.3.1 AKU-Aalborg

The association AKU-Aalborg is to the best of their ability trying to give as many options to the consumer as possible. This is done by playing the role of being a middleman or as a hub to provide a better overview. The service itself, works by making an application with mundane information like nationality, preferred moving date and whether or not you have kids. To complete the application, the applicant must select their own top priority [4]. Since AKU-Aalborg works as a hub in this aspect, selection of accommodation, is done by specifying the area, you would like to live in, and from here the rest of the searching is done manually.

When looking at a service like AKU-Aalborg as a hub to find accommodation, it is worth to remember that AKU-Aalborg merely deals with the municipality of Aalborg.

2.3.2 Ungdomsboliger

When looking beyond accommodation in just Aalborg to a more country-wide service, one would be able to stumble upon Ungdomsboliger. Since Ungdomsboliger covers a bigger area, it will have a higher success rate, in giving a satisfying result, if preference span a large distance. Ungdomsboliger is reminiscent of AKU-Aalborg in that, it serves as a hub to redirect consumers to a supplier of accommodation, corresponding to their preferences [35]. Setting it apart from AKU-Aalborg is therefore based on where the consumer is able to apply for their housing.

2.3.3 Studee

As previously mentioned, a service like AKU-Aalborg takes a relatively small number of preferences into account when generating what housing to present to the consumer. A service like "Studee", raises the number of preferences considered and uses those preferences to provide the user with different educational programs/studies. Unlike the before mentioned websites, applications and such, "Studee" is not a service provider for finding accommodation but rather a service provider for students to find different educational programs in a couple of dozen countries around the world. While "Studee" in itself is not related to the problem at hand, its concept and structure could provide inspiration for the final program, as it is a well-built program, that does what we want ours to do just by tackling a different subject matter.

At this part of the State of the Art analysis, it can be gathered that increasing the amount of potential precise results, seems like an appropriate goal to strive for in creating a similar product.

2.3.4 How do we differ from State of the Art

Despite looking at the previous 3 examples of services designed to make getting accommodation easier as a student, none of these have been specifically designed by or for the school. By schools being in charge of their own methods of assembling groupings of appropriate students, they would of course themselves be able to choose what aspects of every student they deem relevant. Two of the schools putting in effort to best assign housing is "Wells College" and "Hilbert College".

Differing more greatly from "AKU-Aalborg", "Ungdomsboliger" and "Studee", "Wells" and "Hilbert College" give the ability to better take personal preferences such as personality type, how much you are going to be on campus, or whether or not you would like to live with someone of the same major [12][11].

How exactly the different preferences is used in tandem with an algorithm can be seen in the section "Scope of Project".

2.4 Questionnaire

The project is based on a questionnaire, which asks about different preferences like gender, age range, preferred bedtime, etc. These questions are going to be used in the algorithm to specify the roommates compatibility. Collected data from the questionnaire will lead students to have a sorted list of the most compatible roommate options.

2.4.1 Ensuring truthful answers

There are many reasons of why some people are not answering a questionnaire truthfully. They might want to improve their own feelings of self-worth by exaggerating different aspects like income or job title and therefore accomplish their feeling of self-worth. Not only do they feel the need to appear successful, they also tend to follow what is generally seen as correct in society, and do not want to feel like they are on the outside with their different opinion on something [23].
Some people may become defensive when it comes to certain sensitive topics such as religion, sexual relations or orientations, drugs, race-relations, etc. They may be ashamed that they feel this way about the topic and do not want others to know they feel like that [23]. If the participant does not trust that their answers will be anonymous or confidential then they might choose a dishonest answer. This is especially true when their answers could harm them in some way, if for example, the questions are about criminal activity, infidelity etc. Maybe the participant does not like the company doing the survey or the research being done with the questionnaire. In this case they might want to answer predominantly negative to skew the results [23].
To ensure honesty it is important to reduce the amount of "trigger" questions. These are questions that relate to behavior, beliefs or belonging. These kinds of questions are often necessary though, therefore the questions must be worded carefully to reduced how much they affect the person taking the survey.
It might be possible to catch a liar in a survey by using misleading answers and redundant questions. For example, if the questionnaire asks the participant about which city they would rather live in, and among the answers there is a fake city included or just a city that is not in the respective country. The liar can then be spotted if the participant chooses the misleading answer. In the case of redundant questions, you can ask two questions that are very similar, but the participants answer varies wildly this is also a sign of dishonesty. However, if catching a subject in a lie ends up coming into the open, the questionnaire has already been polluted by these lies and has become that less accurate as a result. Therefore, the focus should be far less on catching potential liars, and instead on ensuring that their likelihood of occurrence is as slim as possible.

Another useful way to improve honesty is to make the survey anonymous. For example, if the questionnaire is about politics and is taken over the phone or in person, the participant might tense up and provide answers that are more neutral or less controversial than what they think themselves and thereby provide dishonest answers. In an anonymous survey there is not any social pressure on the participant, and they can therefore provide a more honest answer [23].

Why do we need to ensure that it is truthful?

As mentioned before, the project is based on the questionnaire, therefore it is important to make sure that the applicants answer the questions with honesty. Otherwise the result will not meet the projects purpose.
As described in the previous section, peoples answers will potentially vary depending on factors such as the question itself. Since the questionnaire for the students wont include redundant questions, the questions can be deemed as having an actual impact corresponding to the answer. Because of this, while it may be outside of our scope to come up with the questions that are definitively the most relevant for any particular tested subject, any deviation from a truthful answer on the applicants part will lead to an incrementally more inaccurate weighing of their potential living quarters.
It is imperative that such situations are to be avoided, specifically as it would impact and potentially have a strong negative effect on the accuracy of the algorithm. Therefore the methods for not just catching lies, but preventing them, is necessary for the algorithm to give a result that is as accurate as we can make it.

2.5 Impacts of having roommates

Most students are willing to share their room or apartment with one or more roommates. Some of the universities require having the first year of study on campus. Therefore, it is important to know about the impacts of having roommates on academic performance. The percentage of first-year students taken from a national data set of 128 institution in the United States with and without roommates is shown in figure 2.1 [33].

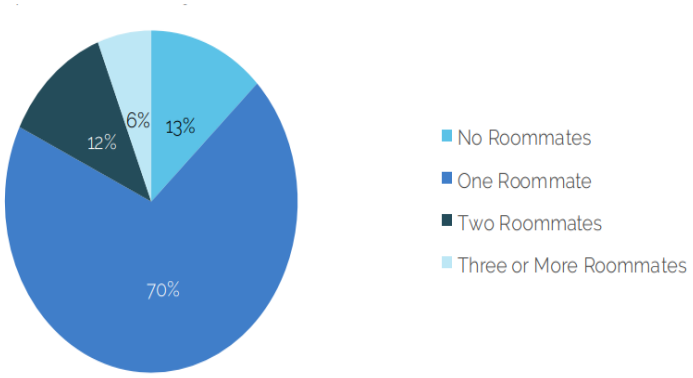


Figure 2.1: Percentage of first-year students with the number of roommates [33].

Institutional commitment, satisfaction and homesickness distress are the other factors that is related to having a roommate on the first year of study. In figure 2.2 the first-year students with roommates have 5% higher satisfaction average and 4% higher commitment compared with the students without roommates. Even though the difference is not so big, it is significant [33].

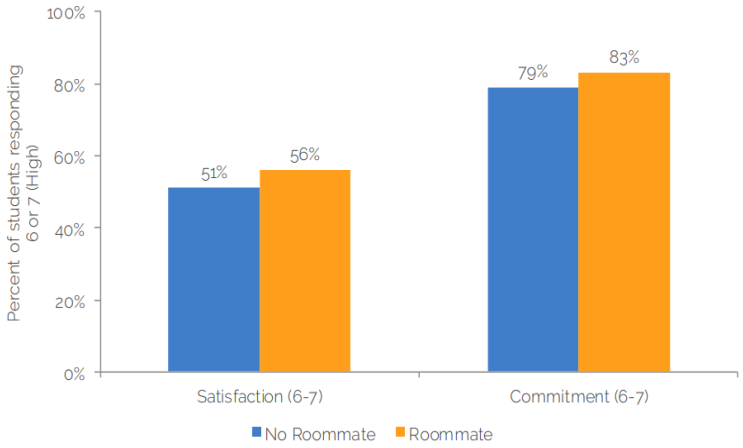


Figure 2.2: Percentage of first year students responding average 6 or higher on questions related to institutional commitment and satisfaction[33].

Some of the first-year students are dealing with homesickness distress, but living with a roommate helps to reduce it. In the figure 2.3 you can see that the students who live with a roommate are less involved in this issue [33].

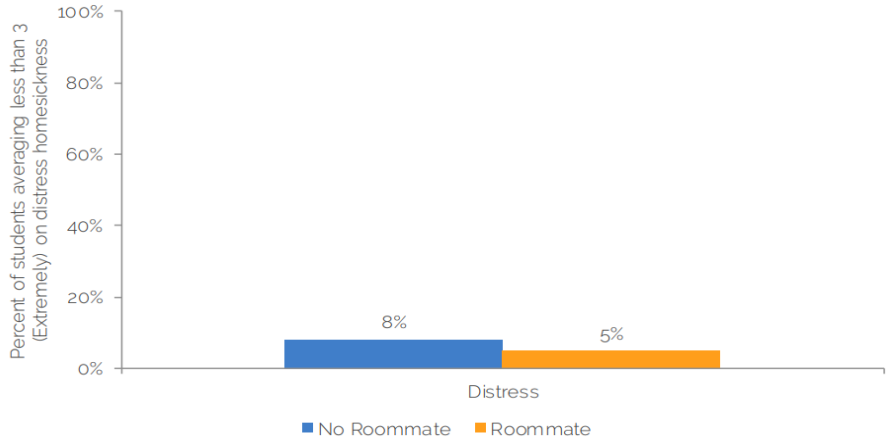


Figure 2.3: Homesickness distress [33].

Academic performance

Taking reference in figure 2.4, the students who live with roommates have almost a .10 higher GPA in the first and second semester of their study compared to students without a roommate. It can be concluded that having one or more roommates has a significant impact on students academic performance [33].

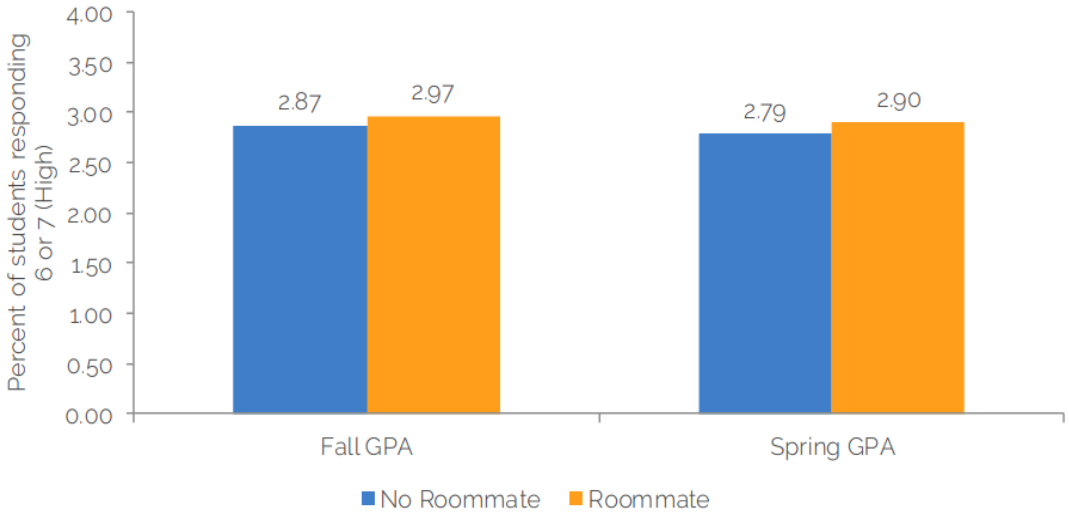


Figure 2.4: Percentage of average GPA of first year students with or without roommate [33].

2.5.1 Mental Health

Good mental health is a key part of education and learning, as poor mental health can lead to stress, anxiety and/or depression. Stress can cause memory difficulties, difficulty with concentration, sleep problems and losing the desire to learn, just to name a few consequences.
Students who do not often see family or friends, like an international student, will have a lack of emotional and practical support. This can lead to a higher chance of the individuals mental health being poor or worsening [20].
The residents at a students dormitory can affect that said student's mental health, for better or worse. If a student's relationship with their roommate is great, that might help to improve both roommates mental health, as an example the roommate might encourage them to go to bed earlier, wake up earlier, maybe start working out or a simple thing such as eating better and healthier. Just like family, a student's roommate might notice the day-to-day difference between changes in feelings or behavior. The opposite can be said as well, if students have a bad relationship with their roommate. As an example if the roommate annoys the student with how they live or handle certain things, while being stressed from work or school, that can lead to poor or worsening mental health [38][2].

2.5.2 Roommates matching benefits

When students decide to live with a roommate, they may not be aware of how complicated it might be. Because a student's roommate are most likely, the first non-family member they are going to live with and it can cause doubled challenges. In a nationwide research made among 31.500 American undergrad students, 50.1% of women and 44.1% of men experienced repeatedly or periodic conflicts with their roommates. The research also showed 5.6% of undergraduate students announced that having problems with their roommates had weakened their academic performance. The statistic show how important it is to have a good relationship with your roommates [34].
Other research shows, that some factors like personality traits and behavior patterns can have good impacts on having a healthy relationship with a roommate. In the research, where 84 roommate pairs of women were studied, the impact of similarity in some of personality factors like conscientiousness and self-determination were inescapable [34].
Some of the conflicts between roommates come from diversity in personality, culture, sleep and study habits, life style etc. so the idea of matching students can reduce the amount of conflicts and lead to a better relationship [28].

2.6 Scope of Project

In the past few years about 150.000 students have enrolled in danish universities [30]. Many of these students are also moving out for the first time. These students must, at the same time, figure out their new life as a university student, and how to live without their parents, whether it will be alone or with a roommate. As all group members are students who have gone through this process, we know that it can be a little confusing, trying to find a roommate that is just right for you. A dormitory might be specifically tricky since here you will end up living quite close with a lot of other strangers that you might be more or less socially compatible with. Therefore, we wish to simplify this with our project. Based on this, the proposed solution to the problem will aim to take much of the guesswork out of searching for the most compatible roommate and thus provide housing based on individual preferences.

2.6.1 International Students

Among the many students enrolling in danish universities each year, a small but important part are the international students. Accounting for just 2440 students in 2021, these students are not just experiencing the same obstacles as ordinary danish university students, but also have the added pressure of everything being new and unfamiliar. Pressure of fitting into danish culture, most likely being in need of housing and most importantly for this project, currently without the ability to find a fitting roommate with the ease of action that we think could be provided are all problems new students have upon coming to Denmark. All of these obstacles will be taken into account for as smooth of a transition as possible to their new life with our given solution. To best help describe the scope of the project, delimitations of both the project as a whole and the product is presented next.

2.6.2 Project delimitation

In our project we will simplify the solution to only include dormitories in Aalborg and the surrounding areas. We have chosen this to narrow the scope of the project so we within the given time can make as polished a product as possible. The paper covers Aalborg specifically because that is the municipality the paper was being written in. Additionally it was chosen to focus solely on one city, based on the nature of the project. The project is thought to be a way of finding a fitting roommate and then presenting the applicant with a roommate matching the criteria of both the applicant and roommate, in an easy and effective way. What this essentially means for the project is that, the priority will be on finding the applicant an appropriate roommate, rather than the priority being finding the applicant a place to live.

2.6.3 Product delimitation

In our product we will simplify the solution to only include the roommate options by compatibility, name, email, if the apartment is handicap friendly, the price and at last the location. For the product, made up data will be used to represent information held by the individual current residents. The made up data will then be used to insert into an algorithm which in turn will provide an output in the form of text and an accompanying e-mail describing the result of the questionnaire with a corresponding method of application.

2.7 Problem statement

To help form the problem statement, specific key issues have been selected to narrow down the area of importance and the problem statement will therefore be rooted in these key issues.

Of these key issues we have that:

- We have to be inclusive when taking our target audience into consideration; domestic and international.
- Due to the problem that living in a dormitory often will be a gamble in terms of compatibility, the individual student's preferences should be taken into account.
- The preferences will have to be gathered in an easy and approachable manner and the preferences will have to be usable as input for algorithm.

After having conducted sufficient research and found these key issues, we are able to more accurately determine what exactly this project is about, meaning the problem statement can now be formulated as such:

How can a software solution be made for students, domestic and international, to help them find the most compatible roommate and accommodation options based on personal preferences?

To further describe the problem, sub questions can be set up:

- How can a software solution be made to make it easier for students to find the most compatible roommate that can benefit their study in Aalborg?
- How can an algorithm be made with preferences and requirements as inputs to effectively assign the students the optimal option for accommodation?

By setting up both a problem statement and accompanying sub questions, it will allow for more appropriate and precise further work of the project.

Chapter 3

Program Design

Now that the problem statement has been defined, we have to think about the software solution. The group unanimously agreed that the software that should be used, was a website to solve the problem statement. There is a lot that has to be thought about before beginning to create the software solution. First off the group made a MoSCoW-analysis to see which things we must have, should have, could have and won't have. Once that had been agreed upon we sketched out the landing page of the website to give a rough markup of what it should eventually look like. So how do you solve the "most compatible" part of the problem statement, you do that with an algorithm. There are millions of different algorithms, but the one we found most compatible for us was Brute Force Optimization.

3.1 MoSCoW

Almost no matter how you set the scope at the beginning or during the development of the program, different elements may be prioritized differently. As such, these different elements can be represented using the MoSCoW-method covering what is seen as a must have, should have, could have and won't have. By doing this, the project can gather a more specific scope through prioritizing the elements which have the most relevance, or perform best in a cost/benefit analysis:

Must	Should	Could	Won't
Questionnaire	GUI	Real input	Apply for dormitories
Weighting system for questionnaire	Actual dorms/roommates	Link as result (Application form)	User login
% compatibility	Admin login	Personality test	Social network
Brute force optimization algorithm	GDPR Regulations	Language DA/EN	Payment system
Comprehensive guide for international students	Email with results	Support for multiple person questionnaire	Available in other countries
Contact us/Feedback	Privacy policy	Multiple cities	Apply for Boligstøtte, SU and/or Utilities
Roommate as a result, text only	Database	Currency selection	
About us section		EULA - End User License Agreement	
Web server		Information about output dorms	
Pseudo input			

Table 3.1: A MoSCoW-analysis was developed on the basis of the programs requirements.

Now that the MoSCoW-analysis has been conducted, a further explanation will be made to the specific parts in section 3.1.1 and further beneath. Through this, it will be explained why the different functions and parts of the project are placed in the way that they are.

3.1.1 Must have

The category "Must have" relates to all the functionalities that are essential/vital to the program. This means that without the "must have" functionalities it would not be possible for the group to make use of the program to fully fulfill the requirements set.

"Questionnaire"
The questionnaire is a vital part of the project. The questionnaire's job is to get the user's preferences on different questions, in order to help them get the most compatible roommate.

"Weighting system for questionnaire"
Some preferences might not be as important to one person as it is to another. The proposed solution is to give the applicant the possibility of choosing the most important questions them self. Selected questions will have the most weight.

"Percentage Compatibility"
Another feature which is exceptionally convenient for the user, is having a percentile score for one's compatibility with the featured top roommates. This makes

it more manageable than the result being presented as a large bullet pointed list which shows each question answered and how they line up with the particular roommate. By having the algorithm's weighting system adjoined to a percentile base, the end result given to the respondent is far more readable than it could be if they needed to go through each answered question and check their compatibility that way.

"Brute force optimization algorithm"
The Brute force optimization algorithm will be the first one to run after the user has submitted their questionnaire. It will be split up into two parts.

First part:
The first part of the algorithm consists of getting in the raw questionnaire/form data from the user, then calculating the different weights the user inputted. Thereafter the results after the weighting will be put into either an array or an object to prepare it to be compared.

Second part:
After the weightings have been calculated we compare the answers that the applicant has given and the answers of all the different current residents. Once these has been calculated we multiply them with the weightings from the first part. What we end up with is a percentage compatibility between the applicant and all the different current residents.

"Comprehensive Guide for International Students"
Despite being inclusive of students of all nationalities, extra effort is made to ensure international students feel at home and are able to carry out a normal, productive life. A comprehensive guide is not deemed as a "must have" since it is not an absolute necessity to life, but based on the fact that the program's intent is making everyday life easier, it has been placed under "must have".

"Contact Us/Feedback"
Whether the program stays within the scope of a project, or expands to include a wider space of dormitories, a pipeline for any kind of feedback is valuable to optimize the usability of the program. This would coincide with a method for contacting administrators, which would be a separate page that includes a feedback form that may have different, categorized preset options for what feedback may be given.

"Roommates as a result. Text only"
The baseline for the algorithm having run its course, the final result should at

minimum be a text-based response that shows the 3 roommates which is the most compatible with the applicant based on the answers that they have given. As this is a bare-bones feature, it includes nothing but the most relevant information, which would be the official title of the relevant roommate.

"About Us"
A minor convenient feature, an "About Us" page for the site would be convenient in explaining what exactly the point of the questionnaire and site is, and should likely be placed front and center upon entry. This gives an immediate baseline for what the average user should know before taking the questionnaire and being sorted using the algorithm, and could give information such as the importance of honesty or anything a first time user should be aware of.

"Web server"
To best represent how this service would be seen and used in a real life scenario, it is seen as beneficial to use a web server to bring in the GUI. Without a web server, the user would not be able to access the contents of the website at all, so in order to bring the program into a more publicly accessible state it is deemed necessary to include a web server.

"Pseudo input"
As is also described a little further in the report, the inputs used for the algorithm will not be based on or taken from any specific source. By merely using pseudo inputs for the algorithm it is possible to show the intent of the program. As far as future development of the program goes, it would be a possibility to either use more tailored, measured inputs, which for example could be based on research done by others, or it could be based on actual data from existing roommates. None of these possibilities have been placed under "must have" since they are not vital to show the program's use, related to the problem statement.

3.1.2 Should have

The category of "Should have" deals with functions that are pertinent to the functionality of the software, but not explicitly necessary to succeed in solving the problem at hand. Without these, the program will fulfill the base requirements set up, but would not have the desired functionality that the group feels is necessary.

"GUI"
In order to make the program easier for the user to understand and use, a more cosmetic approach is taken to make it easier for the user to interact with the program as it will provide more visual feedback rather than just pure strings of text.

This is a "should have" due to how the questionnaire will be structured, as strings of text in a command prompt would not be sufficiently for displaying such things as sliders or bulleted lists.

"Actual dorms/roommates"
Having actual roommates to measure a percentile compatibility with would be ideal in most scenarios, as these real persons would give a more accurate concept of what the algorithm actually succeeds in doing, rather than having arbitrary hypothetical roommates. This comes with the factor of human error, as no matter how much anyone can try to make a realistic example of how things are set up, there is a constant margin of error based on the uncertainty of reality. Therefore, to get the most accurate results, said results must be based in actual reality. This is categorized as a "Should have" as while it is extremely important to include in a scenario such as future development, it is not necessary to show the principal functionality of the algorithm and accompanying program due to the use of pseudo-inputs.

"Admin Login"
A login specifically for Administrators that gives "behind the scenes" access to various functions of the site would have a mutual benefit, as this would make it easier to add such factors as new roommates to the list of options without needing to rewrite the baseline code of the site. Additionally, with having an explicit Admin Login without one for the user, it lessens a potential privacy concern as an applicant would not need to give away any sort of personal information that would remain in the system to refer to that specific user.

"GDPR Regulations"
Following GDPR is necessitated by European Law, and if the algorithm and associated site should ever see any kind of public use, it must follow these regulations on all fronts. This includes having a log of what potential data is kept and where, how long it is kept for, and how it is being kept, while also logging the reasoning behind all data that is being obtained and following the GDPR principles for responsible data harvesting.

"E-mail with Results"
For the project, a decision was made to stray away from the idea of using a login-system for the user. Instead it was chosen that the processed result of questionnaire will be presented on a separate page and an e-mail will be sent to user with the results for later use.

"Privacy Policy"
In the same vein as following GDPR regulations, having a solid privacy policy is a requirement by law, and it is therefore necessary, should the program ever be launched into being anything more than a project. However, this is more within the scope of future development, and is therefore not prioritized in favor of must have functions that are essential to make the algorithm work.

"Database"
An entry in the MoSCoW-table such as "database" represents the back-side of the comparison algorithm, in that it would be essential for the program to have existing data to compare fresh, new input to. This would be an algorithmic compilation of a collection of surveys given to every current resident of the accommodations, which is what every new applicant would have their answers compared to, to find their final compatibility with each roommate.

3.1.3 Could have

The entries in the category "could have" deal with functionalities which are not essential for the program, but would still have a position influence if resources allow it.

"Real input"
As opposed to the pseudo-inputs that are required, real inputs are functionally almost exclusive to the scope of future development. The only reason that they are placed in "Could have" despite their perceived value is due to the fact that pseudo-inputs are equally valuable in showing the functionality of the program, while being significantly easier to replicate without a large scale process of getting results from actual respondents.

"Link as result (Application form)"
Giving an applicant a direct link to an application form for their chosen accommodation is a feature which would be convenient to have, both for streamlining and simplifying the process. However, this requires an amount of resources that are not available to the project at hand, as it requires a collaboration with the services that are in charge of such applications, such as AKU-Aalborg and similar services. As such collaborations are not within the scope of program, this has been relegated to "could have" despite its functional convenience.

"Personality test"
A personality test, will help students, to find roommates, based on personality. But

as the existing personality tests takes 10-15 minuets to take, it was decided to put the personality test in "could have" section.

"Language DA/EN"
Having an option between the website being presented in Danish or English would be a good thing to include especially seeing as expanding to include Danish would allow for easier use for domestic students more so than having the site in English would allow for. However, as it is not as important or necessary as other functionalities such as the questionnaire or database, we decided to not waste time and workforce on it.

"Support for multiple person questionnaire"
A questionnaire for multiple people might be useful if someone wants to move together with a friend or another person, as it would then include the preferences of both of them, and see if they both are compatible with the various roommates.

"Multiple cities"
The only city/region taken into account is Aalborg, as it was deemed unnecessary to widen the scope of the project and just focus on one city/region, but it would be a positive inclusion to include multiple cities.

"Currency selection"
Selection what type of currency is shown on the website would help international students' understanding of the price of various accommodation's costs, such as rent.

"EULA - End User License Agreement"
As the website is free, and there is nothing to sell or be sold in the website, an user license agreement is not important to include.

"Information about output dorms"
Information about the accommodation other than price and location will not be written in the results the user gets. However the results could have a link to said accommodation, so the user would get information about the housing the roommates live in.

3.1.4 Won't have

All the items under "Won't have", are topics, that are not going to be include in the project.

"Apply for dormitories"
It wont be possible to apply for an accommodation. The outcome of the questionnaire contains some suggestions of accommodation options. Although they can use the "Comprehensive guide" to apply for the suggested accommodations.

"User login"
There will be no need for a "User login" system, since the users are going to use the portal, when they want to find a residence. If they decide to choose another place to stay, they can easily start the process from the beginning and fill out the questionnaire according to their new preferences. Any information kept by the website would be handled exclusively if the applicant finds their appropriate accommodation, in which case their answers in the questionnaire will be recorded for the duration of their stay in the apartment.

"Social network"
Including social network in the platform necessitates a high degree of security. Furthermore, there will not be allowed for tenants and house lords to contact each other for security reasons.

"Payment system"
Since the platform is going to be a free platform, there will be no need for a payment system.

"Availability in other countries"
The Web site is going to suggest users some accommodation options just in Aalborg, Denmark. And the other cities or countries will be unavailable.

"Apply for Benefits"
Like accommodations, the user has to manually apply for benefits, such as SU or boligstøtte, although the website could help the user find where and how to apply for said benefits. This would go under the "Comprehensive guide" part of "Should have".

3.2 Website design

The website design is crucial to the users impression of the website. If the user does not like the landing page or the overall flow of the website they might not want to spend any time on it, and we do not want that. So one thing we did not want to double down on was the design of the website.

3.2.1 Initial visualization

To present the algorithm on which the solution to the problem described in the report is based on, it was chosen to develop a website reminiscent of a site dealing with a similar concern. The structure was first drawn to give an initial impression of the flow of the program as presented below. It is here important to remember that initial visualization may include elements or functionalities which are not present on the final version and that the final version might have elements or functionalities which at the time were not yet considered.

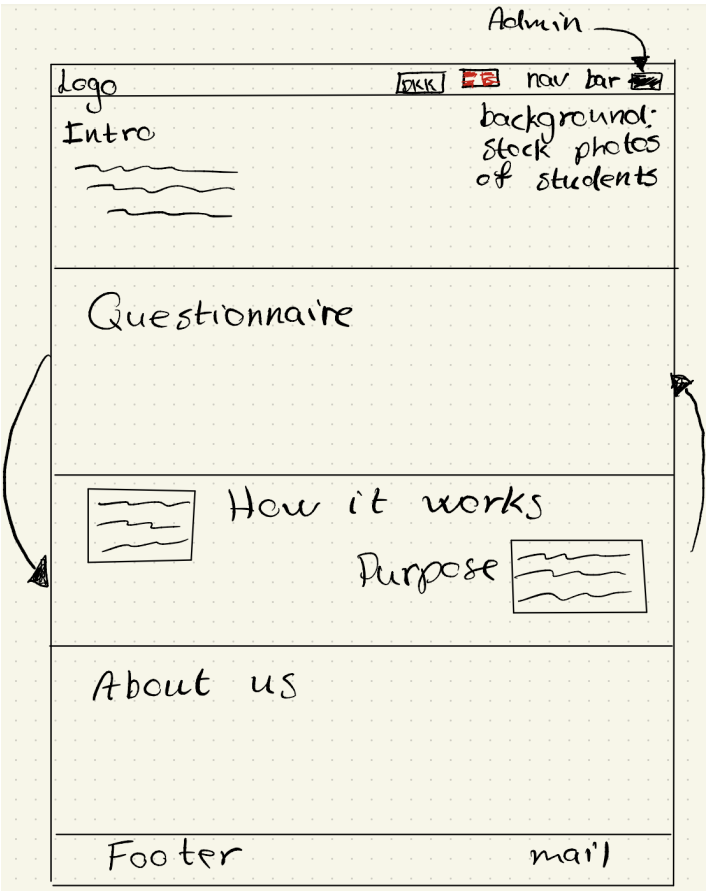


Figure 3.1: Initial front page design.

Nav bar

For the nav bar we want to give the user the possibility to change the language of the website, hereby increasing the inclusivity of international students. Similarly the nav bar will have the ability to change the currency used on the website. This is done so the student can be presented with an apartment after finding a compatible roommate in their preferred currency. The website is not going to have a login-system, hence why there is only an option for administrators to log in. Lastly is the drop-down menu which allows the user to select where on the page they want to navigate to.

Questionnaire

In this section the questionnaire will be placed, which will be constructed of multiple pages with a submit button at the end. Upon clicking the submit button, the data entered in the questionnaire will be sent to the database and used to calculate the compatibility of the applicant with every existing resident.

How it works

As is demonstrated with the pointing arrows, the order of the questionnaire and the "How it works" section is as of the initial visualization not yet determined. However, the HIW-section is going to provide information on how the process of searching for a roommate will proceed upon starting. Additionally, the purpose of the questionnaire/project will be presented.

About us

Even though it can be argued a section "About us" is not necessary, we feel it is a good practice to include, since in the situation of future development beyond the project it would be beneficial to create a sense of trust and reasoning.

Footer

Alongside the nav bar, the footer is a part of the website, which is considered reusable in other parts of the website. Keeping this in mind, it is important that every reusable part of the website is somewhat vague as to not have change anything. Based on this, the footer will include an email, a phone number and possibly other information seen as essential.

3.3 Questionnaire

To help better the user experience of going through and answering the questionnaire, the flow of the questions have been visually set up in a flowchart. The questionnaire is set to start with getting the general information of the applicant, which is then followed by the applicants personal preferences about their potential roommate. On the 4th page of the questionnaire the applicant will answer questions about the dormitory and/or apartment they are applying to itself.

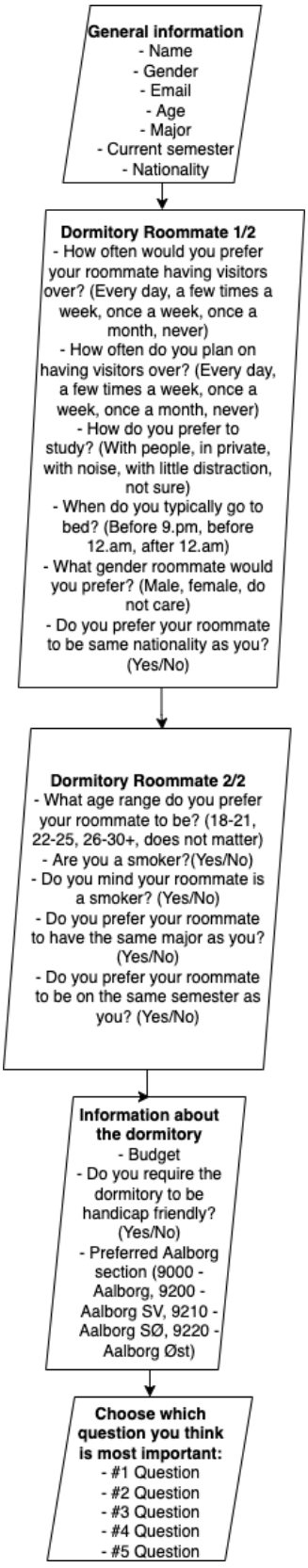


Figure 3.2: Questionnaire flowchart.

General information

The first "page" you see in the questionnaire is the general information, this asks the user about basic information such as name and so on. Some of these questions such as gender, age and nationality are being used for the algorithm to compare with the current resident.

Dormitory Roommate 1/2

The second "page" asks multiple questions according to the users preferences that are being used in the algorithm as well. This could be "How often would you prefer your roommate having visitors over?".

Dormitory Roommate 2/2

The third "page" is a continuation of the previous one, that continues to asks for more personal preferences that the user has, that. could be "Are you a smoker?".

Information about the dormitory

This is more focused questions to the user, and only the user. It can be seen as a filter to who the user gets compared to.

Choose which questions you think is most important

The last "page" is asking the user to choose which of 9 questions they think is the most important, these are the questions that gets the highest weight when comparing.

3.4 Optimization problem

Facing a problem makes us find different solutions, and finding the best solution among all the existing solutions, is often referred to as an optimization problem. For having an optimization problem, we normally need to maximize or minimize a value from a function. For example, if you want to buy an airplane ticket, the most optimized one, is the ticket that has the minimum cost, if your preference is to buy the cheapest one [25].
In our project, the problem we try to solve is to find the best matching roommate for students applying using our website. For this, we define an objective function which can be read more about in the next sections. In our case this function must be maximized, to give the most compatible roommate option based on the students' preferences.

Decision variable

Every optimization problem consists of variables called decision variables. Decision variables are undetermined, and have a domain. The values in a domain constitute all possible values for the certain decision variable [19]. Decision variables in our case will be the available roommates. We assume the variable X for it and the domain is as follows: $X = \{X_1, X_2, X_3, X_4, X_5, ..., X_r\}$ where X is the Decision variable and r is the last available roommate.

3.4.1 Objective function

After specifying decision variables, the next step is to define the formula we are going to use. Accordingly, we remodeled the function from [8] called fitness function.

Remodelled formula

In our case we will make it available for two people to live together, therefore we need to remodel the original formula from Chih-Ching Chang and Che-Chern Lin's report[8], since this loops through the number of available roommates.So the remodeled formula we use is:

$$f = \sum_{i=1}^{15} D_i \times W_i \tag{3.1}$$

i is the number of questions
D_i is the Delta variable related to i'th question (more description in 3.4.4)

W_i is Weighting of i'th question (more description in 3.4.5)

As mentioned before, for having an optimized solution (the most compatible roommates), we maximize the function, and here is the objective function:

$$maxf = max \sum_{i=1}^{15} D_i \times W_i \tag{3.2}$$

Constraints

In the real world, it is most likely to happen that problems have some sort of specific conditions. These conditions that limit the problems are called constraints. The provided solution must solve problems yet still satisfy all the given constraints, for having an optimization problem. Doing this will give a so called feasible solution [26] [18].
As an example, in our project, it might happen that some students have a limited budget for renting a residence. We assume this case as a constraint, since if the budget is limited, the student might refuse considering the residence as an option. The other constraint is about handicap friendliness. If the current resident's answer to the related question is "Yes", it means that the accommodation is handicap friendly. Third constraint is related to the Aalborg section applicant chooses to live in.

3.4.2 Maximization linear program

Solving the following linear program will be the optimal solution we were looking for.

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{15} (D_i * W_i) \\ \text{subject to} \quad & Q_{19y} \geq Q_{19x} \\ & Q_{20x} = Q_{20y} \\ & Q_{21x} = Q_{21y} \end{aligned} \tag{3.4}$$

The first part of this linear program is the maximization of the objective function we use, to find the most compatible option.
The second part is where we subject to constraints. $Q_{19y} \geq Q_{19x}$, which is related to question number 19, means that the applicants budget, must be greater or equal than the current roommates accommodation cost. The second constraint ($Q_{20x} = Q_{20y}$), means if the applicant is looking for a handicap friendly accommodation, the other options without the facility will be removed from the final list. The third constraint ($Q_{21x} = Q_{21y}$), checks if the applicants selected Aalborg section is the

same as the current resident, and if so, removes the other roommate options who do not live in that specific Aalborg section from the result list.

Brute force algorithm

To solve optimization problems it is necessary to use an algorithm. Algorithms are one of the fundamental and integral parts of computer science. To solve a computational problem, there will always be need for methods that represent how to solve the problem systematically. In fact, algorithms are step-by-step instructions, that take values as input and generate new values as output [18].
There are several algorithms for solving problems, but not all of them are applicable. Choosing the appropriate algorithm, can depend on factors like type and size of the data, the problem statement, type of the output, available computational time and so on [5]. Brute force is a searching method that checks all the possible solutions, therefore the algorithm's run time depends on how big the solution set is [7].

In the project, we decided on using this method because of two factors; we will not necessarily have to run through every option because of constraints and since the amount of danish accommodation for students and thus potential residents/applicants are not of unmanageable size. Depending on the type of constraint, the algorithm may become more or less complicated, which could impact efficiency, but in the case of our algorithm, the constraints will remove some of the options from the list and in return make it more efficient at reaching the desired results. To take advantage of this method we are using a function which measures compatibility and we are going to maximize it, to see exactly which option is the most compatible.

Final output

The final output of the brute force optimization algorithm must be the roommate that has the maximum percentage of compatibility, and can therefore be written as:

$$Result = argmax(f(x)) \tag{3.3}$$

The argmax is the most optimized argument returned from the algorithm. For the case of the program, the argmax variable is the resident which the algorithm returns as the most compatible [37]. What this is essentially saying is that given a function like this project's optimization algorithm, which takes potential roommates as arguments, argmax can specify the most compatible option, which is the roommate with maximum compatibility percent.

3.4.3 Questionnaire

In this section, the questions from the questionnaire can be seen.
It is assumed that these questions was filled out by the current resident of an apartment. Since such information is not currently available, and it is also time consuming to collect, the group decided to use pseudo inputs.
These inputs are supposed to be compared with the applicants answers and give the final recommendation list of available options.
Questions that are marked in **bold**, are the ones we use in our brute force algorithm, and the ones in italic are used as constraints.

- Q_1 is "First Name"
- Q_2 is "Last Name"
- Q_3 is **"Gender" (Male, Female)**
- Q_4 is "Email address"
- Q_5 is **"Age" (18-21(1), 22-25(2), 26-30+(3))**
- Q_6 is **"What Major are you studying?" (List of all AAU majors)**
- Q_7 is **"What semester are you on?" (1-10)**
- Q_8 is **"Nationality" (List of all countries)**
- Q_9 is **"How often would you prefer your roommate having visitors over?" (Every day(5), A few times a week(4), Once a week(3), Once a month(2), Never(1))**
- Q_{10} is **"How often do you plan on having visitors over?" (Every day(5), A few times a week(4), Once a week(3), Once a month(2), Never(1))**
- Q_{11} is **"How do you prefer to study?" (With people(5), In private(4), With noise(3), With little distraction(2), Not sure(1))**
- Q_{12} is **"When do you typically go to bed?" (Before 9. PM(3), Before 12. AM(2), After 12. AM)(1)**
- Q_{13} is **"What gender roommate would you prefer?" (Male(3), Female(2), Does not matter(1))**
- Q_{14} is **"Do you prefer your roommate to be the same nationality as you?" (Yes/No)**
- Q_{15} is **"What age range do you prefer your roommate to be?" (18-21(1), 22-25(2), 26-30+(3), Does not matter(4))**

- Q_{16} is **"Are you a smoker?" (Yes(1)/No(2))**
- Q_{17} is **"Do you prefer your roommate to have the same major as you?" (Yes/No)**
- Q_{18} is **"Do you prefer your roommate to be on the same semester as you?" (Yes/No)**
- Q_{19} is *"Budget" (<2000 DKK, <3000 DKK, <4000 DKK, <5000 DKK, <6000 DKK)*
- Q_{20} is *"Do you require the dormitory to be handicap friendly?" (Yes/No)*
- Q_{21} is *"Which section of Aalborg do you wish to live in?" (9000 - Aalborg Central, 9200 - Aalborg South West, 9210 - Aalborg South East, 9220 - Aalborg East)*

3.4.4 Calculating D_i

Since it is needed to determine whether a result should raise or lower compatibility, an D_i value is determined for certain questions. This is either presented as the Delta between results from x and y, where x is the person that answered the questionnaire and y is a resident that currently lives in the dormitory or by short pseudo code determining the outcome based on the questions. Looking at D_1, which is of the former kind, it can be seen that the greater the Delta the more negative the effect will be on the final outcome, whereas the opposite is true for a lesser Delta. For example, for calculating D_1, we define a delta variable, if the person applying in the website answer in question 9 that he/she is ok with that the resident has visitors every day (the answer number 5), and the current resident has also answered that he/she has visitors every day (answer number 5), the Delta is 5-5=0, and that means the situation where delta is equal to zero is the most compatible situation. accordingly, D_1 gets value 1, which is the highest value.

Table 3.2: Comparisons between student answering the questionnaire and student already living in the dormitory. (x is student answering, y is student already living there)

D_i	**Calculation**
D_1	$\Delta = \lvert Q_{9x} - Q_{10y} \rvert$ $\Delta == 0 \rightarrow 1$ $\Delta == 1 \rightarrow 0.5$ $\Delta == 2 \rightarrow 0$ $\Delta == 3 \rightarrow -0.5$ $\Delta == 4 \rightarrow -1$

Table 3.2 continued from previous page

D_i	**Calculation**
D_2	Δ = | Q_{10x} - Q_9y | $\Delta == 0 \rightarrow 1$ $\Delta == 1 \rightarrow 0.5$ $\Delta == 2 \rightarrow 0$ $\Delta == 3 \rightarrow -0.5$ $\Delta == 4 \rightarrow -1$
D_3	Δ = | Q_{11x} - Q_{11y} | $\Delta == 0 \rightarrow 1$ $\Delta == 1 \rightarrow 0.5$ $\Delta == 2 \rightarrow 0$ $\Delta == 3 \rightarrow -0.5$ $\Delta == 4 \rightarrow -1$
D_4	$\Delta = \lvert Q_{12x} - Q_{12y} \rvert$ $\Delta == 0 \rightarrow 1$ $\Delta == 1 \rightarrow 0$ $\Delta == 2 \rightarrow -1$
D_5	If Q_{13x} == 'Does not matter(3)' || $\Delta = \lvert Q_{13x} - Q_{3y} \rvert == 0$ D_5 = 1 Else $D_5 = -1$
D_6	If Q_{13y} == 'Does not matter(3)' || $\Delta = \lvert Q_{13y} - Q_{3x} \rvert == 0$ D_6 = 1 Else $D_6 = -1$
D_7	If Q_{14x} == 'Yes' If Q_{8x} == Q_{8y} $D_7 = 1$ Else $D_7 = 0$
D_8	If Q_{14y} == 'Yes' If Q_{8y} == Q_{8x} $D_8 = 1$ Else $D_8 = 0$

Table 3.2 continued from previous page

D_i	**Calculation**
D_9	If Q_{15x} == 'Does not matter(4)' || $\Delta = \|Q_{15x} - Q_{5y}\| == 0$ D_9 = 1 Else if $\Delta == \|Q_{15x} - Q_{5y}\| = 1$ $D_9 = 0$ Else $D_9 = -1$
D_{10}	If Q_{15y} == 'Does not matter(4)' || $\Delta = \|Q_{15y} - Q_{5x}\| == 0$ D_9 = 1 Else if $\Delta == \|Q_{15y} - Q_{5x}\| = 1$ $D_9 = 0$ Else $D_9 = -1$
D_{11}	If Q_{16x} == 'smokerYes' || Q_{16x} == 'smokerNo' If $Q_{16x} == Q_{16y}$ $D_{11} = 1$ Else $D_{11} = -1$
D_{12}	If Q_{17x} == 'Yes' If $Q_{6x} == Q_{6y}$ $D_{12} = 1$ Else $D_{12} = 0$
D_{13}	If Q_{17y} == 'Yes' If $Q_{6y} == Q_{6x}$ $D_{13} = 1$ Else $D_{13} = 0$
D_{14}	If Q_{18x} == 'Yes' If $Q_{7x} == Q_{7y}$ $D_{14} = 1$ Else $D_{14} = 0$
D_{15}	If Q_{18y} == 'Yes' If $Q_{7y} == Q_{7x}$ $D_{15} = 1$ Else $D_{15} = 0$

3.4.5 Question weighting W_i

After the students answer all questions in the questionnaire, they will be able to choose 5 questions, that has the most degree of importance for them. These 5 questions get the highest weighting which is 10% and the other ones get the minimum weighting which is 5%. That is how the algorithm lists the most compatible options based on the students preferences. Since the questionnaire has 15 questions and 5 of them get 10% the 10 questions that are left get 5% which in the end equals 100%.

Chapter 4

Program Documentation

In order to ensure sufficient depth of the documentation of the program, the programs accompanying code will be broken into smaller sections. The explanations to the code beneath will refer to the line numbers in the report, and not in the actual code it self. To avoid confusion, the code will be split into the front-end and the back-end, and the code will additionally be divided into fitting subsections.

4.1 Programming Languages and Frameworks

For a more well-rounded experience of developing the program, different programming languages and frameworks were used, all of which will be introduced beneath.

4.1.1 Languages

As the group are making a website it is important to talk about which languages the program will be written in. These languages are the bare bone basics of a website.

HTML & CSS

Despite all the different opinions on whether or not HTML and CSS are programming languages, they are listed as such here. The way HTML was used in the program meant, that it would almost always be working together with either CSS or the framework EJS. Using HTML and CSS together provide the backbone of what web-development really is. What lies behind this claim is that HTML is great at creating the program structure while CSS provides the styling [16].

JavaScript

For the program a general rule of thumb is that whenever data is being manipulated, it is done using JavaScript. For JavaScript it is also worth noting that JavaScript can be used for both front-end and back-end, which is a very valuable attribute. In relation to this project, JavaScript was used to add behavior to elements like the questionnaire, like what happens when the next and previous button is pressed. Also, JavaScript is used to add interactivity, like the routing happening when specific actions happen e.g. clicking a link [16].

4.1.2 Frameworks

The frameworks of the website is what makes it special, and it makes it easier for the developer to express what they want.

Expressjs

Many of the functionalities in the program rely on a framework in one way or another. This could for example be seen in the "app.js" file, where an instance of the express framework is initialized to create a server [24].

Bootstrap

Bootstrap is a framework aimed at making the process of styling an application easy and manageable. Generally seen the process of styling could be replicated with plain CSS, but the advantages of Bootstrap is the ease of implementation as well how it facilitates responsive design, meaning adapting to other devices would be a lot less painful [10].

EJS

It was mentioned that the EJS framework is often used with HTML, and actually requires the file extension to be .ejs. This means that ejs replaces the HTML files. The difference between HTML and EJS is however not that significant, at least not in the way it was used in this program. EJS was used to allow the use of variables used in the JavaScript code to display in an HTML format [15].

4.2 Front-end

The front-end of the program is what the user is able to see, and is everything that the user interacts with. The front-end as previously mentioned is very crucial, and is not something that should lack anything.

4.2.1 Partials

When writing code it is always going to be superior to write sustainable and reusable code. This way, you save time and potentially money as well as lessening the chance of bugs and increase readability. Based on this, partials, which are parts of a specific page, which might be used on another page are made.

Header

The header consists of the vital things, that the browser needs to know about the website. This includes what the user sees on the tab in terms of the website name and the accompanying logo. Besides this, the header is the central place to include references such as bootstrap.

Listing 4.1: Header - 1-16 - /views/partials/head.ejs

```
1  <meta charset="UTF-8" />
2  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
3  <meta name="viewport" content="width=device-width,
       initial-scale=1.0" />
4  <title>Roomscape</title>
5  <link rel="icon" href="/images/web_icon.svg" />
6  <link
7      href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/
           css/bootstrap.min.css"
8      rel="stylesheet"
9      integrity="
           sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftj
10          DbrCEXSU1oBoqyl2QvZ6jIW3"
11     crossorigin="anonymous"
12 />
13 <link
14     rel="stylesheet"
15     href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1
           .3.0/font/bootstrap-icons.css"
16 />
17 <link rel="stylesheet" href="/stylesheets/style.css" />
```

Navbar

Since the website could benefit from easy navigation, a navbar is introduced, which is used to navigate to either different parts of the landing page or to a different page all together. The navbar essentially consists of the bar-element itself, a logo with accompanying text, a drop-down menu consisting of 3 hyperlinks and lastly three extra hyperlinks also used for navigation. This is accomplished with bootstrap by setting up the different parts. For all the elements listed below, all of them will be wrapped in the bar element which makes up the navbar.

Listing 4.2: Drop-down menu - 1-67 - /views/partials/navbar.ejs

```
<div class="collapse navbar-collapse" id="navmenu">
    <ul class="navbar-nav ms-auto">
        <li class="nav-item dropdown">
            <a
                class="nav-link dropdown-toggle"
                href="#"
                id="navbarDropdown"
                role="button"
                data-bs-toggle="dropdown"
                aria-expanded="false"
            >
                Navigation
            </a>
            <ul class="dropdown-menu">
                <li>
                    <a class="dropdown-item" href="/#
                       howitworks"
                        >How It Works</a
                    >
                </li>
                <li>
                    <a class="dropdown-item" href="/#
                       questionnaire"
                        >Questionnaire</a
                    >
                </li>
                <li>
                    <a class="dropdown-item" href="/#
                       questions">FAQ</a>
                </li>
                <li>
                    <a class="dropdown-item" href="/#aboutus
                       "
                        >About us</a
                    >
```

```
32                  </li>
33              </ul>
34          </li>
35
36      </ul>
37  </div>
```

Listing 4.3: Navigation buttons - 1-67 - /views/partials/navbar.ejs

```
1  <li class="nav-item">
2      <a href="/guide" class="nav-link">Guide</a>
3  </li>
4  <li class="nav-item">
5      <a href="/contact" class="nav-link">Contact Us</a>
6  </li>
7  <li class="nav-item">
8      <a href="#" class="btn btn-primary btn-ml">Admin</a>
9  </li>
```

Footer

Similar to the navbar, the footer is to be used on every single public page we have on the website to keep it consistent across the website. Utilizing bootstrap allows easy creation of visuals, such as the footer. The footer itself is only styled on the line 1 and 2, which sets the positioning, color and text styling. This whole footer element then wraps to contain the remaining elements: a small paragraph and a hyperlink styled as a button referring to a specific position on the page.

Listing 4.4: Footer - 1-8 - /views/partials/footer.ejs

```
1  <footer class="p-5 bg-dark text-white text-center
       position-relative">
2      <div class="container">
3          <p class="lead">Copyright &copy; 2022 Aalborg
               University</p>
4          <a href="#" class="position-absolute bottom-0 end-0
               p-5">
5              <i class="bi bi-arrow-up-circle h1"></i>
6          </a>
7      </div>
8  </footer>
```

Script

The code seen in the snippet below can be thought of as a template of what needs to be included in every EJS file. These are the link needed to use Bootstrap as well

a pointer to an external JavaScript file, "/javascripts/script.js". How this code is included in EJS files can be seen in the code snippet below the code in listing 4.6.

Listing 4.5: Script - 1-6 - /views/partials/script.ejs

```
<script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/
       js/bootstrap.bundle.min.js"
    integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+
       IlRH9sENBO0LRn5q+8nbTov4+1p"
    crossorigin="anonymous"
></script>
<script src="/javascripts/script.js"></script>
```

Utilizing partials

Now because these partials have been made, we are able to use them on the pages that we have made. As an example, you can see how we use the header.ejs and navbar.ejs file in practice, that is listed in listing 4.1 & 4.2 on the /views/pages/index.ejs file.

Listing 4.6: How to use partials - 1-11 - /views/pages/index.ejs

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <%- include('../partials/head') %>
    </head>
    <body>

        <!-- Navbar -->
        <%- include('../partials/navbar') %>

        .
        .
        .
        <%- include('../partials/script') %>
    <\body>
<\html>
```

4.2.2 Pages

With the partials created, they can now be implemented as stable pieces of the different pages. Utilizing partials as such leads to a consistent design less prone to errors. The different pages the user will be presented with can be seen below.

index.ejs

Index.ejs consists of our homepage/landing page. This is the first page the user is met with. It cannot be stressed more that this website is heavily lifted by the Bootstrap framework. Each of these subsections below are embedded in the body of the HTML.

Showcase

Showcase is the first thing you see on the website when you arrive. Is displays the title that explains what the whole website is about, and a short text. The user is able to press the "Get started!" button that takes them to the questionnaire, or they can scroll down to the "How it works" section.

Listing 4.7: Showcase - 12-37 - /views/pages/index.ejs

```
<section
            class="bg-dark text-light p-5 p-lg-0 pt-lg-5
               text-center text-sm-start"
        >
            <div class="container">
                <div
                    class="d-sm-flex align-items-center
                       justify-content-between"
                >
                    <div>
                        <h1>
                            Find the most
                            <span class="text-warning"
                               >COMPATIBLE</span>
                            roommate for You!
                        </h1>
                        <p class="lead my-4">
                            Take our questionnaire today and
                                find your very own!
                        </p>
                        <a href="#questionnaire" class="btn
                           btn-primary btn-lg">Get started!<
                           /a>
                    </div>
                    <img
                        class="img-fluid w-100 d-none
                           d-sm-block"
                        src="/images/searching.svg"
                        alt=""
                    />
                </div>
```

```
25              </div>
26          </section>
```

How it works

The "How it works" section of the website contains 3 text sections that in short explains how the algorithm works. This is to give the user a better understanding of what the website actually is and how they get the answer that they get.

Listing 4.8: How it works - 62-77 - /views/pages/index.ejs

```
1  <section id="howitworks" class="p-5">
2      <div class="container">
3          <div
4              class="row align-items-center
                   justify-content-between"
5          >
6          <div class="col-md">
7              <img src="/images/howitworks_1.svg" class="
                   img-fluid" alt="">
8          </div>
9          <div class="col-md p-5">
10             <h2>How it Works</h2>
11             <p class="lead">By taking our questionnaire,
                   your personal preferences are evaluated
                   against residents in your selected area, so
                   go ahead and get started below.</p>
12             <p></p>
13
14         </div>
15     </div>
16 </section>
```

Questionnaire / Form

There is multiple different sections here. There will be a few examples from the form, and also some JavaScript + CSS of how the animations and the whole form works.

Firstly a look at some of the form elements:

Listing 4.9: First div in questionnaire section - 122-491 - /views/pages/index.ejs

```
1  <div class="mb-5 card-step" data-step>
2      <h3 class="step-title">Personal Information</h3>
3      <div class="form-group">
4          <label for="firstName">First Name</label>
```

```
5           <input type="text" class="form-control" name="
               firstName" id="firstName" placeholder="Jens"/>
6       </div>
7       <div class=" form-group">
8           <label for="lastName">Last Name</label>
9           <input type="text" class="form-control" name="
               lastName" id="lastName" placeholder="Jensen"/>
10      </div>
11      <div class="form-group">
12          <label for="gender">Gender</label>
13          <select class="form-select" name="gender" id="gender
               ">
14              <option selected>Select gender.</option>
15              <option value="1">Male</option>
16              <option value="2">Female</option>
17          </select>
18      </div>
19      <div class="form-group">
20          <label for="email" class="form-label">Email address<
               /label>
21          <input type="email" class="form-control" name="email
               " id="email" placeholder="name@example.com"/>
22          <div id="emailHelp" class="form-text">We'll never
               share your email with anyone else.</div>
23      </div>
24      <div class="form-group">
25          <label for="age">Age</label>
26          <select class="form-select" name="age" id="age">
27              <option selected>Select your age.</option>
28              <option value="1">18-21</option>
29              <option value="2">22-25</option>
30              <option value="3">26-30+</option>
31          </select>
32      </div>
33      .
34      .
35      .
36      <button type="button" class="btn btn-primary"
           data-next>Next</button>
37  </div>
```

In the code above, the code representing the part of the first "page" of the questionnaire can be seen. In total it therefore contains 5 input fields; firstName, lastName, gender, email and age. To simplify only the first and last are looked at. The code for the first input field starts on line 3 in the code above, where a div tag with the class "form-group" is started. Inside the div tag a label element is used to create

text indicating what the intended input for the input field below is. On line 5 the actual input field is declared as a single-line field meant for text. The field is given both the name and ID of firstName and lastly a placeholder is made, given an example to the user of might be put in the field.

The last input field in the code above starts on line 24, where a div tag is made with the class form-group. Using a label element, text is placed above the input field which reads "age". Next, on line 26 a select menu is made with the "form-select" class from Bootstrap with the name and age being set to "age". Inside the form-select 4 options are placed. As can be seen on line 27, the "selected" keyword is used to preselect the option reading "Select your age".

Now lets take a look at the CSS that is attached to the form:

The CSS in listing 4.10 is mainly for the form animation. If you take a look at the form and interact with it, it shows multiple pages with a sliding animation. The visual part of it is purely done with the CSS displayed beneath.

Listing 4.10: Questionnaire CSS - 16-84 - /public/stylesheets/style.css

```
1  .card-step {
2      background-color: white;
3      border: 1px solid #333;
4      border-radius: .5rem;
5      padding: .5rem;
6      max-width: 450px;
7      margin: 0 auto;
8      animation: fade 250ms ease-in-out forwards;
9  }
10
11 .form-group {
12     display: flex;
13     flex-direction: column;
14     margin-bottom: .5rem;
15     gap: .25rem;
16 }
17
18 .form-group:last-child {
19     margin: 0;
20 }
21
22 .step-title {
23     margin: 0;
24     margin-bottom: 1rem;
25     text-align: center;
26     color: #333;
```

```
}

.card-step.active {
    animation: slide 250ms 125ms ease-in-out both;
}

.multi-step-form {
    overflow: hidden;
    position: relative;
}

.hide {
    display: none;
}

@keyframes slide {
    0% {
        transform: translateX(200%);
        opacity: 0;
    }

    100% {
        transform: translateX(0);
        opacity: 1;
    }
}

@keyframes fade {
    0% {
        transform: scale(1);
        opacity: 1;
    }

    50% {
        transform: scale(.75);
        opacity: 0;
    }

    100% {
        opacity: 0;
        transform: scale(0);
    }
}
```

And at last the JavaScript:

The following code in listing 4.11 is the JavaScript made for the form.
In line 1 and 2 we start by defining the variables, multiStepForm and formSteps. These two variables are assigned to query selectors that returns the first Element within the document that matches the specified selector, or group of selectors, which gets the two id's "data-multi-step" and "data-step" that are both assigned to respectively the form and each page/div of the form. Line 4-6 assigns the current "page" of the form to "currentStep", this is to make sure that the program always know where it is at. Line 8-12 makes sure that the first "page" of the form has the value of 0, and then assign the class of "active" to the first div. Each of the buttons on each page is either "Previous" or "Next", the code from line 14 to 32 reacts to the action of clicking on the two buttons, and increments or reduces the variable called "incrementor". The const called inputs on line 24 grabs all HTML inputs that is on the respective page. Line 25 then looks at all of these inputs to see if they have been filled with the minimum requirements, this could be the first name input, and that just requires that it contains text. The if statement from line 26 to 29 checks if the boolean variable "allValid" is true, and if it is, it adds the "incrementor" variable to the global variable called "currentStep" that keeps track of which page the client is on. At last it calls the function called "showCurrentStep" that toggles the class "active" on the current page which is equal to the "currentStep" variable.
As the questionnaire is actually one big page, the different "pages" is only thanks to JavaScript and CSS, so the forEach statement from line 34-42 takes care of the animation when you press the "next" and "previous" button, and adds the class "hide" the the pages you currently are not looking at.

Listing 4.11: Questionnaire JavaScript - 10-57 - /public/javascripts/script.js

```
1  const multiStepForm = document.querySelector("[data-multi-
       step]");
2  const formSteps = [...multiStepForm.querySelectorAll("[data-
       step]")];
3
4  let currentStep = formSteps.findIndex(step => {
5      return step.classList.contains("active");
6  });
7
8  if (currentStep < 0) {
9      currentStep = 0;
10     formSteps[currentStep].classList.add("active");
11     showCurrentStep();
12 }
13
14 multiStepForm.addEventListener("click", e => {
15     let incrementor;
16     if (e.target.matches("[data-next]")) {
17         incrementor = 1;
```

```
18      } else if (e.target.matches("[data-previous]")) {
19          incrementor = -1;
20      }
21
22      if (incrementor == null) return;
23
24      const inputs = [...formSteps[currentStep].
            querySelectorAll("input")];
25      const allValid = inputs.every(input => input.
            reportValidity());
26      if (allValid) {
27          currentStep += incrementor;
28          showCurrentStep();
29      }
30
31      showCurrentStep();
32  });
33
34  formSteps.forEach(step => {
35      step.addEventListener("animationend", e => {
36          formSteps[currentStep].classList.remove("hide");
37          e.target.classList.toggle(
38              "hide",
39              !e.target.classList.contains("active")
40          );
41      });
42  });
43
44  function showCurrentStep() {
45      formSteps.forEach((step, index) => {
46          step.classList.toggle("active", index ===
                currentStep);
47      });
48  }
```

FAQ

The FAQ is the section beneath the form. This section contains 5 accordions with frequently asked questions that the user might need an answer to. As an example could this be "Can we guarantee a spot at a dormitory?". Beneath the FAQ section is the "About us" section.

Listing 4.12: Item 1 in FAQ accordion - 750-760 - /views/pages/index.ejs

```
1  <!-- Item 1 -->
2  <div class="accordion-item">
3    <h2 class="accordion-header">
```

```
4       <button class="accordion-button collapsed" type="button"
            data-bs-toggle="collapse" data-bs-target="#
           question-one">
5         Can we guarantee a spot at a dormitory?
6       </button>
7     </h2>
8     <div id="question-one" class="accordion-collapse collapse"
         aria-labelledby="flush-headingOne" data-bs-parent="#
        questions">
9       <div class="accordion-body">The short answer is, no. As
           our expertise lies in matching students with their
           most compatible roommates, Roomscape cannot guarantee
            accommodation at a dormitory not affliated with
           Roomscape</div>
10    </div>
11  </div>
```

About us

The "About us" sections consists of 3 cards that display each group member, this is available for the user to get an understanding of who the developers are and their socials. This is non-vital.

Listing 4.13: About us - 810-854 - /views/pages/index.ejs

```
1  <section id="aboutus" class="p-5 bg-primary">
2      <div class="container">
3          <h2 class="text-center text-white">About Us</h2>
4          <p class="lead text-center text-white mb-5">Group
               A218b</p>
5          <div class="row g-4">
6              <div class="col-md-6 col-lg-4">
7                  <div class="card bg-light">
8                      <div class="card-body text-center">
9                          <img src="/images/BanafshehPic.jpg"
                               class="rounded-circle mb-3" alt="
                               " height="130">
10                         <h3 class="card-title mb-3"
                               >Banafsheh Nourizadehbarabi</h3>
11                         <p class="card-text">Lorem ipsum,
                               dolor sit amet consectetur
                               adipisicing elit. Nihil odio
                               nobis nam ea, omnis officia.</p>
12                         <a href="#"><i class="bi bi-twitter
                               text-dark mx-1"></i></a>
13                         <a href="#"><i class="bi bi-linkedin
                                text-dark mx-1"></i></a>
```

```
14                          <a href="#"><i class="bi
                               bi-instagram text-dark mx-1"></i
                               ></a>
15                      </div>
16                  </div>
17              </div>
18              <div class="col-md-6 col-lg-4">
19                  <div class="card bg-light">
20                      <div class="card-body text-center">
21                          <img src="/images/Thomas_billede.jpg
                               " class="rounded-circle mb-3"
                               alt="" height="130">
22                          <h3 class="card-title mb-3">Thomas
                               Bjeldbak Madsen</h3>
23                          <p class="card-text">Lorem ipsum,
                               dolor sit amet consectetur
                               adipisicing elit. Nihil odio
                               nobis nam ea, omnis officia.</p>
24                          <a href="https://twitter.com/
                               ThomasBjeldbakM"><i class="bi
                               bi-twitter text-dark mx-1"></i></
                               a>
25                          <a href="https://www.linkedin.com/in
                               /thomasbjeldbakmadsen/"><i class=
                               "bi bi-linkedin text-dark mx-1"><
                               /i></a>
26                          <a href="https://www.instagram.com/
                               eshes1509/"><i class="bi
                               bi-instagram text-dark mx-1"></i
                               ></a>
27                      </div>
28                  </div>
29              </div>
30              <div class="col-md-6 col-lg-4">
31                  <div class="card bg-light">
32                      <div class="card-body text-center">
33                          <img src="https://randomuser.me/api/
                               portraits/men/18.jpg" class="
                               rounded-circle mb-3" alt="">
34                          <h3 class="card-title mb-3">Tobias
                               Christian Hansen</h3>
35                          <p class="card-text">Lorem ipsum,
                               dolor sit amet consectetur
                               adipisicing elit. Nihil odio
                               nobis nam ea, omnis officia.</p>
36                          <a href="#"><i class="bi bi-twitter
                               text-dark mx-1"></i></a>
```

```
37                          <a href="#"><i class="bi bi-linkedin
                                text-dark mx-1"></i></a>
38                          <a href="#"><i class="bi
                               bi-instagram text-dark mx-1"></i
                               ></a>
39                      </div>
40                  </div>
41              </div>
42
43          </div>
44      </div>
45  </section>
```

Contact

Beneath the "About us" section is the "Contact" section. This section contains 3 things, the location where the group is, the phone number for the group which is just marked with "XXXXXXXX" since we do not have a mutual phone number, and at last the groups email if a user should contact us on there. The content of this section is to give the user different information on where to contact us.

Listing 4.14: Contact - 857-879 - /views/pages/index.ejs

```
1  <section class="p-5">
2      <div class="container">
3          <div class="row g-4">
4              <div class="col-md">
5                  <h2 class="text-center mb-4">Contact</h2>
6                  <ul class="list-group list-group-flush lead"
                       >
7                      <li class="list-group-item">
8                          <span class="fw-bold">Location:</
                               span> Strandvejen 12-14, Aalborg
                               9000
9                      </li>
10                     <li class="list-group-item">
11                         <span class="fw-bold">Phone:</span>
                               XXXXXXXX
12                     </li>
13                     <li class="list-group-item">
14                         <span class="fw-bold">Email:</span>
                               fs-21-sw-2-a218b@student.aau.dk
15                     </li>
16                 </ul>
17             </div>
18             <div class="col-md">
```

```
19                  <img src="/images/contact.svg" class="
                        img-fluid w-50 d-none d-sm-block"" alt=""
                        >
20              </div>
21          </div>
22      </div>
23  </section>
```

Results.ejs

Results.ejs is the page that you see whenever you submit the form. The page contains 3 Bootstrap cards that has the content of a picture to resemble the resident, the compatibility percentage between the applicant and the current resident, the residents name, email, if the apartment that the resident lives in is handicap friendly, the price of the apartment and location where the resident lives.

Listing 4.15 contains the Bootstrap alert that gets generated for every error there is if someone does not input the required inputs we expect into the database, server side validation will be explained later in section 4.3.1.

Listing 4.15: Error alert - 16-30 - /views/pages/results.ejs

```
1  <% if(typeof alert != 'undefined') { %> <% alert.forEach(
       function(error)
2  { %>
3  <div
4      class="alert alert-warning alert-dismissible fade show"
5      role="alert"
6  >
7      <%= error.msg %>
8      <button
9          type="button"
10         class="btn-close"
11         data-bs-dismiss="alert"
12         aria-label="Close"
13     ></button>
14 </div>
15 <% }) %> <% } %>
```

The cards mentioned earlier are created in the code beneath, in this listing it is only the first of three cards. Cards are a part of the Bootstrap framework and are nothing else but visual. From line 1-6 is where the image to the card is linked, the source of the image is an API that has different available portraits that is free to use. The body of the card starts on line 7 and ends on line 40, the content of this card is the most compatible resident. The resident[0] is the first person in the resident array which gets sorted and passed to results.ejs in indexRoute.js

which will be explained later in section 4.3. This array is a JavaScript object, so each resident has different variables to access, in this listing we access the .name, .email, .handicapFriendly, .budget and the .aalborgSection, each of these variables are unique to the respective resident. In line 10 we want to display the compatibility percentage that the applicant and resident has, we access this by typing resident[0]?.fitness, and this displays a value between 1-100 which resembles their compatibility. The card takes use of JavaScript that EJS makes available to us, so from line 18-28 we want to display the price, since the value we get back from resident[0]?.budget is a value between 1-5, 1 being the cheapest(<2000 DKK) and 5 being the most expensive(<6000 DKK) we want to display something that the client is able to understand, we therefore have to use JavaScript's if statement, so on line 18 if resident[0]?.budget is equal to "1" it displays "<2000 DKK" and in the following else if statement if resident[0]?.budget is equal to "2" it displays "<3000 DKK". The same thing happens on line 13 for the handicap friendly part and from 29-37 for the section of which the resident lives.

Listing 4.16: Cards - 33-65 - /views/pages/result.ejs

```
1  <div class="card">
2      <img
3          src="https://randomuser.me/api/portraits/men/1.jpg"
4          class="card-img-top"
5          alt="..."
6      />
7      <div class="card-body">
8          <h5 class="card-title">Best Match</h5>
9          <p class="card-text">
10             Compatibility %: <%= resident[0]?.fitness %><br
                   />
11             Name: <%= resident[0]?.name %> <br />
12             Email: <%= resident[0]?.email %><br />
13             Handicap friendly: <% if (resident[0]?.
                   handicapFriendly === "handicapFriendlyNo") {
                   %>
14                 No
15             <% } else if (resident[0]?.handicapFriendly ===
                   "handicapFriendlyYes") { %>
16                 Yes
17             <% } %><br />
18             Price: <% if (resident[0]?.budget === "1") { %>
19                 <2000 DKK
20             <% } else if (resident[0]?.budget === "2") { %>
21                 <3000 DKK
22             <% } else if (resident[0]?.budget === "3") { %>
23                 <4000 DKK
24             <% } else if (resident[0]?.budget === "4") { %>
```

```
25                  <5000 DKK
26              <% } else if (resident[0]?.budget === "5") { %>
27                  <6000 DKK
28              <% } %> <br />
29              Location: <% if (resident[0]?.aalborgSection ===
                    "sectionAalborgC") { %>
30                  9000 - Aalborg Central
31              <% } else if (resident[0]?.aalborgSection === "
                    sectionAalborgSW") { %>
32                  9200 - Aalborg South West
33              <% } else if (resident[0]?.aalborgSection === "
                    sectionAalborgSE") { %>
34                  9210 - Aalborg South East
35              <% } else if (resident[0]?.aalborgSection === "
                    sectionAalborgSW") { %>
36                  9220 - Aalborg East
37              <% } %>
38          </p>
39      </div>
40  </div>
```

Guide.ejs

A link we have in our Navigation bar is for the guide, as this website is primarily focused for international students, we thought that it would be a good idea to have a short guide to the basics of Denmark. So we created that. It consists of 7 different sections that cover housing as seen in the listing below, CPR, Rejsekort, Boligstøtte, SU, internet and phones.

The code is very basic, each topic has their own section, and in each section there is a h2 title of the topic, and paragraph that contain information. For every section there is a link for further information if the client would like a more comprehensive guide.

Listing 4.17: Housing section - 37-53 - /views/pages/guide.ejs

```
1  <h1 class="mt-5 text-center">Guide</h1>
2  <section class="p-5">
3      <div class="container">
4          <div
5              class="row align-items-center
                   justify-content-between"
6          >
7          <div class="col-md p-5">
8              <h2>Housing</h2>
```

```
9               <p class="lead">Housing in Denmark requires lots
                    of information. It is even more complicated
                   for the international students.  </p>
10              <p> The link below is a survival guide for
                   international students including usefull
                   information about housing: <br> <a href="
                   https://esknet.dk/wp-content/uploads/2021/05/
                   Survival-Guide_2021.pdf">International
                   students survival guide</a> </p>
11          </div>
12          <div class="col-md">
13              <img src="/images/howitworks_2.svg" class="
                   img-fluid" alt="">
14          </div>
15      </div>
16  </section>
```

Contact.ejs

If the client should wish to contact us via mail, we have placed a "Contact us" link in the navigation bar. The contact page is very simple and contains a form that asks for the clients first name, last name, email address and whatever text they want to send to us. Whenever the client is ready to send the mail, they press on the "Send mail" button, what this does is take all of the fields that has an input, opens their default mail application and everything is already in the content of the mail, all the client has to do is press "Send".

Listing 4.18: Contact form - 10-73 - /views/pages/contact.ejs

```
1  <h1 class="mt-5 text-center">Contact us</h1>
2  <form
3      id="survey-form"
4      data-multi-step
5      class="multi-step-form"
6      action="mailto:fs-21-sw-2-a218b@student.aau.dk"
7      method="POST"
8      enctype="text/plain"
9  >
10     <div class="mb-5 card-step" data-step active>
11         <div class="form-group">
12             <label for="firstName" id="firstNameLabel"
13                 >First Name</label
14             >
15             <input
16                 type="text"
17                 class="form-control"
```

```
18                  name="firstName"
19                  id="firstName"
20                  placeholder="Jens"
21              />
22          </div>
23          <div class="form-group">
24              <label for="lastName" id="lastNameLabel">Last
                    Name</label>
25              <input
26                  type="text"
27                  class="form-control"
28                  name="lastName"
29                  id="lastName"
30                  placeholder="Jensen"
31              />
32          </div>
33          <div class="form-group">
34              <label for="email" class="form-label">Email
                    address</label>
35              <input
36                  type="email"
37                  class="form-control"
38                  name="email"
39                  id="email"
40                  placeholder="name@example.com"
41              />
42              <div id="emailHelp" class="form-text">
43                  We'll never share your email with anyone
                        else.
44              </div>
45          </div>
46          <div class="mb-3 form-group">
47              <label for="text" class="form-label">Text</
                    label>
48              <input
49                  type="text"
50                  class="form-control"
51                  name="text"
52                  id="text"
53                  placeholder="Input what you want to contact
                        us about"
54              />
55          </div>
56          <button type="submit" class="btn btn-success">Send
                mail</button>
57          <div id="contactHelp" class="form-text">
```

```
58              Once you press "send mail" it will open your
                    default mail
59              client and you will have to send the mail there.
                     Information
60              will be prefilled from the form.
61          </div>
62      </div>
63  </form>
```

4.2.3 How does it look?

This section will display how the different parts of the website looks, this is to give an understanding of how the code previously shown looks.

Index.ejs / Landing page

Figure 4.1 displays listings 4.2 - Navbar & 4.7 - Showcase.

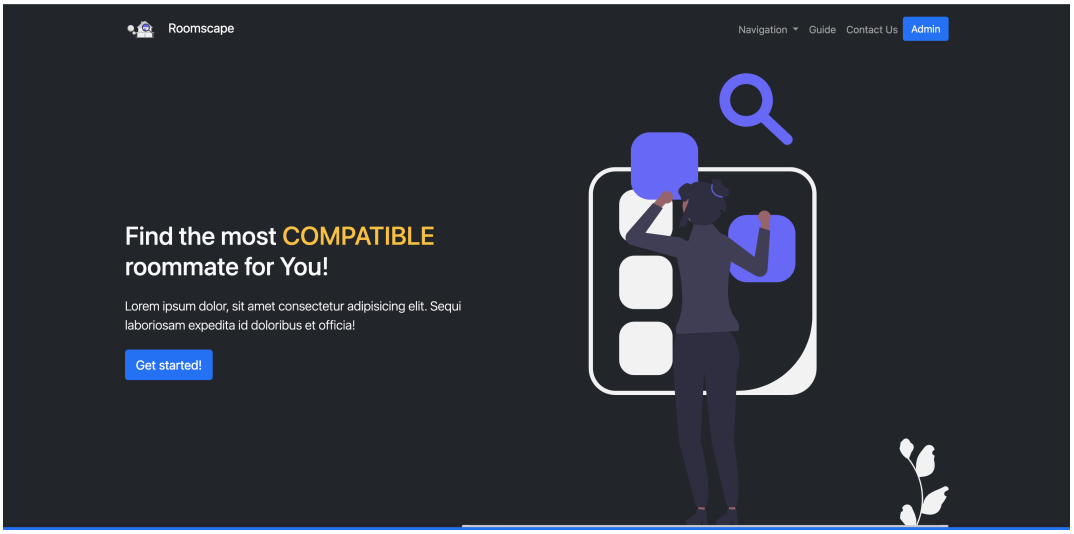


Figure 4.1: Front page - Website

Figure 4.2 displays listing 4.8 - How it works.

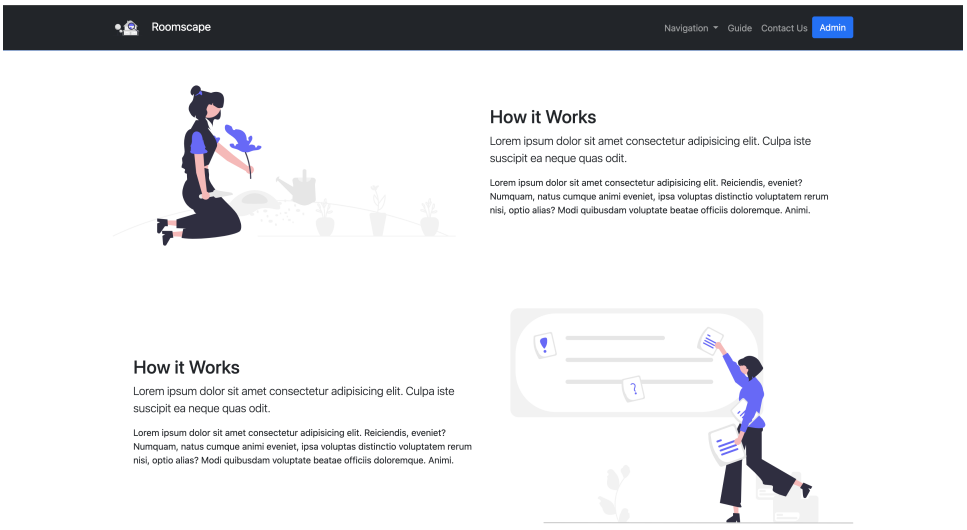


Figure 4.2: How it works - Website

Figure 4.3 display listing 4.9 - First div.

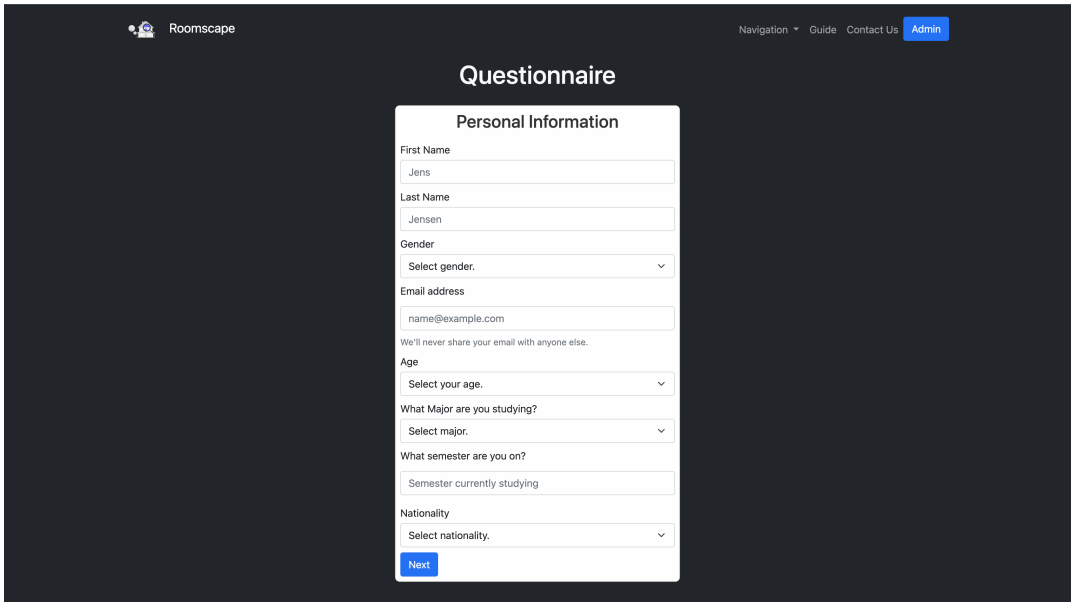


Figure 4.3: Questionnaire - Website

Figure 4.4 displays listings 4.12 - FAQ & 4.13 - About us.

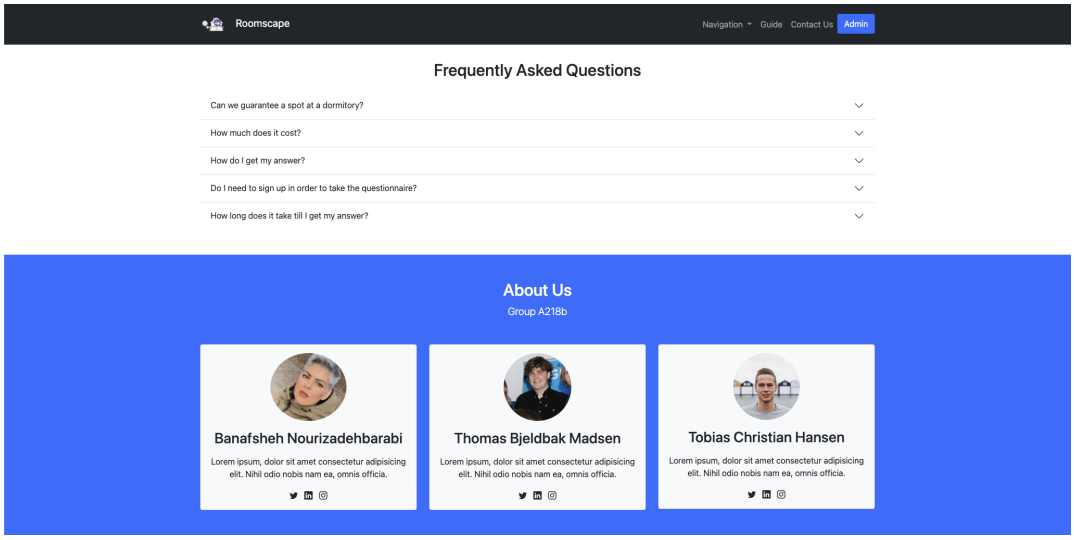


Figure 4.4: FAQ & About us - Website

Figure 4.5 displays listings 4.14 - Contact & 4.4 - Footer.

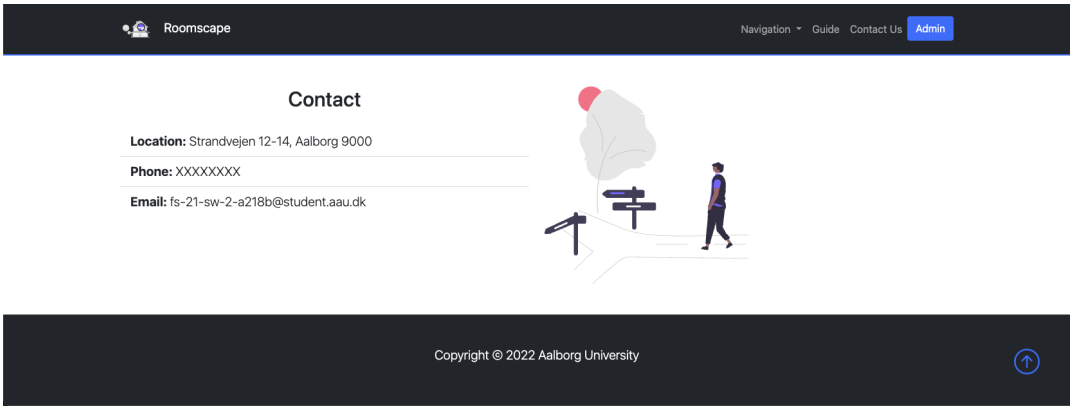


Figure 4.5: Contact & Footer - Website

Results.ejs

Figure 4.6 display listing 4.16.

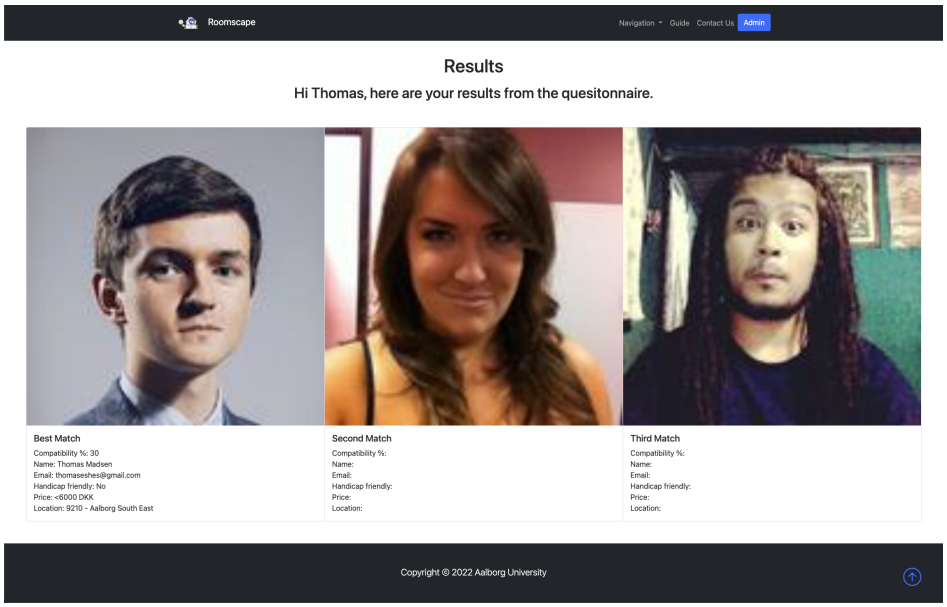


Figure 4.6: Results.ejs - /results - Website

Guide.ejs

Figure 4.7 display listing 4.17.

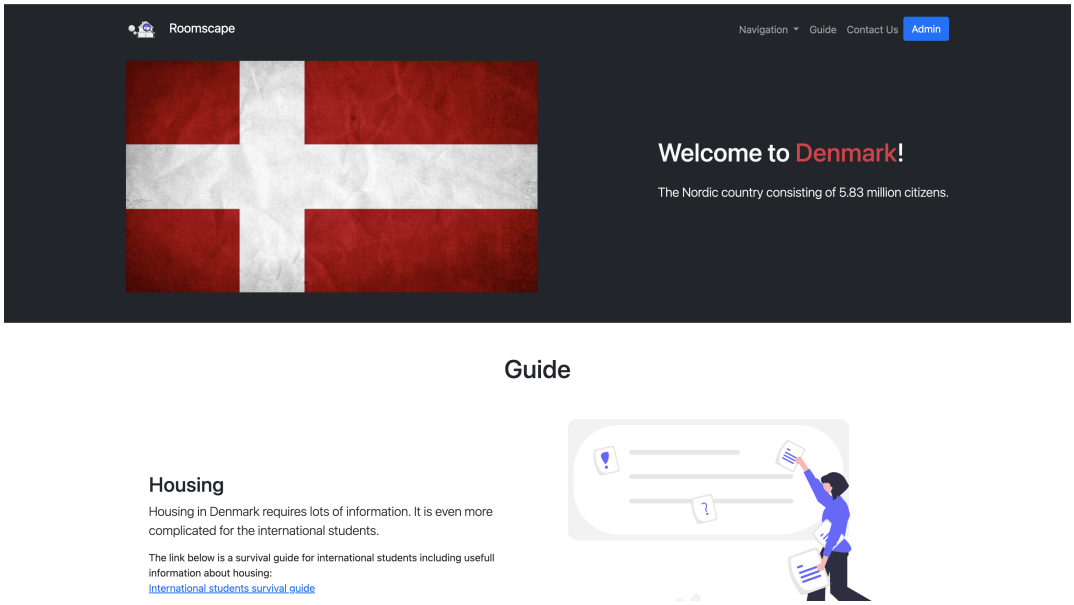


Figure 4.7: Guide.ejs - /guide - Website

Contact.ejs

Figure 4.7 display listing 4.17.

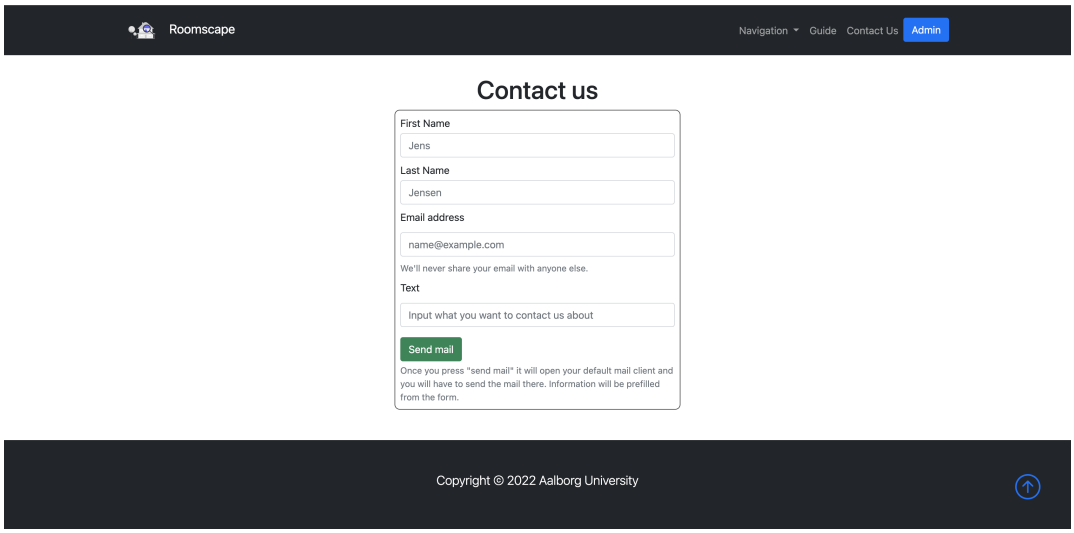


Figure 4.8: Contact.ejs - /contact - Website

4.3 Back-end

The back-end is everything that the user does not see, this is where the server is and the different parts, like the one that calculates the compatibility between the applicant and the current resident. The server is being ran through Node.js, with the server framework Express.js. Node.js and Express.js enables the use of JavaScript in server programming which makes it intuitive and easy to understand. In this section, the whole back-end of the website will be described.

app.js

Initially in app.js, the port which is going to be used is set to the environment variable port, or if that is not available then set to port 5000. On line 3 to 9 the node modules are added and an instance of express is set up on line 4.

On line 11, the view engine is set to be "ejs", which is a template engine. The template files can be found within the /views directory, which is why on line 12 the standard path for anything related to views is set to /views/. Using express, the static files used such as images on the site, styling and JavaScript scripts are declared to be found in the "public" directory on line 13.
To set up a route, indexRoute is brought in from the "indexRoute" file. Setting up the route in this manner will make accessing the / path the starting point of every other path. Lastly the express app is set up to listen to the previously defined port and give an corresponding response upon action.

Listing 4.19: app.js file - 11-28

```
1  const PORT = process.env.PORT || 5000;
2
3  const express = require("express");
4  const app = express();
5  const path = require("path");
6  const indexRoute = require("./routes/indexRoute");
7
8  app.set("view engine", "ejs");
9  app.set("views", path.join(__dirname, "/views/"));
10 app.use(express.static("public"));
11
12 app.use("/", indexRoute);
13
14 // Localhost
15 app.listen(PORT, function (err) {
16     if (err) console.log("Error in server setup");
17     console.log("Server listening on Port", PORT);
18 });
```

indexRoute.js

The first thing done in the indexRoute-file is setting up the node modules as well as setting up an instance of express. The code in the subsection indexRoute.js is to be found at /routes.

Listing 4.20: indexRoute - 13-17

```
1 const express = require("express");
2 const router = express.Router();
3 const { MongoClient } = require("mongodb");
4 const bodyParser = require("body-parser");
5 const { check, validationResult } = require("express-
    validator");
```

Next the body.parser module is brought in for use. What line 1 in the code below is doing is telling the program that we want to handle data in JSON-format.

Listing 4.21: Body-parser - 19-20

```
1 router.use(bodyParser.json());
2 router.use(bodyParser.urlencoded({ extended: true }));
```

As previously mentioned, the website is connected to a mongoDB database. Setting up connection to the database is done as seen in the code below:

Listing 4.22: MongoDB uri - 23-26

```
1 const uri =
2     "mongodb+srv://test:test@cluster0.uwjpd.mongodb.net/
        myFirstDatabase?retryWrites=true&w=majority";
3
4 const client = new MongoClient(uri);
```

Routes:
In order to navigate the different pages on the website, routes have to be set up. This is done with the express framework as seen in listing 4.23 below. After the router is set up, it can be used to make, among others, GET and POST requests and make specific actions upon those requests.

Listing 4.23: indexRoute.js - 29-41

```
1 router.get("/", function (request, response) {
2     response.render("pages/index");
3 });
4
5 router.get("/contact", function (request, response) {
6     response.render("pages/contact");
7 });
8
9 router.get("/guide", function (request, response) {
```

```
10      return response.render("pages/guide");
11  });
```

In the above code, the first GET request on line 1, upon accessing the "/" path, a GET request is made, to which a function that takes a request and a response as input is added. The function then specifies that the response will be to render the "pages/index" path.
The same procedure happens with the GET requests starting on line 5 and 9. The differences here is merely what path has to be accessed before the GET request is made, thus rendering the page corresponding to the path.

The next snippet of code is going to be the POST request handling the results page, that the applicant is sent to after submitting the questionnaire.

Listing 4.24: indexRoute.js - 44-242

```
1  router.post("/results", async function (request, response) {
2          const errors = validationResult(request);
3          const db = client.db("myFirstDatabase");
4          let alert = await error(errors);
5
6          if (errors.isEmpty()) {
7              await client.connect();
8              await db.collection("residents").insertOne(
                   request.body);
9              console.log("Inserted applicant");
10             let resident = await db.collection("residents").
                   find({}).toArray();
11             let weight = await weighting(request);
12             fitness = compatibility(request, resident,
                   weight);
13             client.close();
14         } else {
15             fitness = [];
16         }
17
18         response.render("pages/results", {
19             applicant_name: request.body.firstName,
20             resident: fitness,
21             alert,
22         });
23     }
24 );
```

Line 1 and 2 in the code snippet above creates a POST request, taking an asynchronous function as its second parameter. Using the keyword "async" before function on line 1 simply makes the function return a promise. This is then used

in tandem with the keyword "await", making the program wait until the promise made earlier has been resolved. This is done here to make sure the database, which is connected to on line 2, is ready before it has to be used.
On line 4 the database is declared. Another call of await is made on line 5. Here it is done to make sure that all the data that has to be inserted from the request body with the call of "insertOne" does not get interrupted. This data is then sent to the mongoDB collection called "answers".
Since new data has now been entered in the database, the new and updated data is placed in an array in line 7 and 8. Yet again the await keyword is used with the mongoDB find() method and the JavaScript "toArray" method. Using the find method syntax as seen on line 7 will simply take every entry in the database and put it into the array.
Both weighting() and compatibility() and line 9 and 10 are functions covered later. The connection to the database is on line 12 closed. With all the required data now saved and found, the data is used to render the "pages/results" page with data corresponding to the person taking the questionnaire.

Evaluation functions

The evaluation functions mentioned here are only a few of the many found in the indexRoute.js file. The purpose of the functions is simply explained to return whether the data entered by the applicant is going to make them more or less compatible with the individual residents. This is done to every questions in the questionnaire, hence why only a few will be looked at, as the process of dealing with the different questions do not vary much.

Listing 4.25: indexRoute.js - 254-283

```
1  function x1(q9x, q10y) {
2      let delta, x1;
3
4
5      if (q9x > q10y) {
6          delta = q9x - q10y;
7      } else {
8          delta = q10y - q9x;
9      }
10
11     switch (delta) {
12         case 0:
13             x1 = 1;
14             break;
15         case 1:
16             x1 = 0.5;
17             break;
```

```
18          case 2:
19              x1 = 0;
20              break;
21          case 3:
22              x1 = -0.5;
23              break;
24          case 4:
25              x1 = -1;
26              break;
27      }
28
29      return x1;
30  }
```

In the context of x1, the questions taken into consideration are question 9 in the form, which is about the resident (q9y) and the second question taken into consideration is question 10 in the form, which is about the applicant (q10x).

q9x = How often would you prefer your roommate having visitors over?
q10y = How often do you plan on having visitors over?

The way the question is set up, is that a more infrequent answer will give a higher value. To make sure the delta or difference between answers is positive, an if-statement is introduced on line 5 in the code snippet above. This if-statement test of the answer given by the resident is more infrequent than the applicant. If that is the case, the delta is given by line 6. If that is not the case, the delta is given by line 8. In the case that both applicant and resident has given the same answer, it will merely fall through to the else part of the if-statement, meaning that the delta will just end up being 0. After calculating the delta, the result will go through a switch starting on line 11. Having the delta go through the switch means the bigger the difference in answers between applicant and resident, the less compatible they are according to the evaluation algorithm and vice versa.

The other way of evaluating answers is by checking the equality.

Listing 4.26: indexRoute.js - 484-496

```
1  function x11(q16x, q16y) {
2      let x11;
3
4      if (q16x === "smokerYes" || q16x === "smokerNo") {
5          if (q16x === q16y) {
6              x11 = 1;
7          } else {
8              x11 = -1;
9          }
```

```
10      }
11
12      return x11;
13  }
```

As can be seen in the code snippet above, if the applicant answers either yes or no to being a smoker, another if-statement is hit checking if the answers from the applicant and resident is the same.

Weighting function

To go with the evaluated answers, a weighting is brought in for x1 to x15.

Listing 4.27: indexRoute.js - 562-595

```
1  function weighting(request) {
2      let weight = [];
3      const x = [
4          "planVisitorsRoommate",
5          "preferStudy",
6          "bedtime",
7          "roommateGender",
8          "roommateLanguage",
9          "preferredAgeRange",
10         "btnradio4",
11         "roommateMajor",
12         "roommateSemester",
13     ];
14     let answerID = [
15         request.body.Q1,
16         request.body.Q2,
17         request.body.Q3,
18         request.body.Q4,
19         request.body.Q5,
20     ];
21     for (let i = 0; i < 15; i++) {
22         for (let j = 0; j < 5; j++) {
23             if (answerID[j] === x[i]) {
24                 weight[i] = 0.1;
25                 break;
26             } else {
27                 weight[i] = 0.05;
28             }
29         }
30     }
31     return weight;
32 }
```

From line 3 to 13 in the above code, questions which the applicant might have a stronger opinion about and thus might want to give a higher weighting are declared. The applicant is able to chance the weighting of 5 of the questions. These inputs from the form are stored in the array declared from line 14 to 20. On line 21 and 22 an outer and inner for-loop are setup. The outer loop is set to iterate through all the 15 questions in the form, and the inner loop will run through the 5 questions which were given a higher weighting. An if-statement is now setup which checks if the question which was the set to have a higher weighting is the same question in the list of all questions. If this returns true, the weight array will get a value of 0.1 on the index corresponding to the question. Should the if-statement not return true, the weight for the corresponding question is set to 0.05.

Compatibility function

Since the compatibility function is quite large and somewhat repetitive, only some of the repetitive parts will be presented in the code snippet.

Listing 4.28: indexRoute.js - 598-607

```
1  function compatibility(request, resident, weight) {
2      let fitness = [];
3      for (let i = 0; i < resident.length; i++) {
4          let array = [];
5          let sum = 0;
6
7          let X1 = x1(
8              request.body.planVisitorsRoommate,
9              resident[i].planVisitors
10         );
```

As seen in the code snippet above, a for-loop is initialized, which will span most of the function. The loop will iterate through every resident one by one. The different x's that are formulated in the evaluation function are now assigned a new variable, capitalized X with the form data from the request body and the i'th resident's corresponding answer. This is done from X1 to X15.

Listing 4.29: indexRoute.js - 650-681

```
1  if (
2      resident[i].btnradio7 === "handicapFriendlyNo" &&
3      request.body.btnradio7 === "handicapFriendlyYes"
4  ) {
5      break;
6  } else if (
7      parseInt(resident[i].budget) > parseInt(request.body.
           budget)
```

```
8  ) {
9      break;
10 } else if (
11 request.body.aalborgSection !== resident[i].aalborgSection)
       {
12     break;
13 } else {
14     array[0] = X1;
15     array[1] = X2;
16     array[2] = X3;
17     array[3] = X4;
18     array[4] = X5;
19     array[5] = X6;
20     array[6] = X7;
21     array[7] = X8;
22     array[8] = X9;
23     array[9] = X10;
24     array[10] = X11;
25     array[11] = X12;
26     array[12] = X13;
27     array[13] = X14;
28     array[14] = X15;
29 }
```

For the algorithm, some of the questionnaire questions can be seen as constraints. It was therefore chosen to implement a sort of filter to make sure the constraints are followed, thereby making sure the end result is a feasible solution. This is done with an if-else chain, that on line 2 and 3 first checks equality between the resident and then the applicant. On line 8 the applicant's budget is compared to the actual rent. Lastly on line 11 the inequality of applicant's proposed section of Aalborg to live in is checked again the resident's current section of Aalborg. What this filter essentially does is that it makes sure the housing option is handicap friendly if they require it, they can actually afford it and that it is located in the correct section. If this is fulfilled, an array is filled with the capitalized X's value.

Listing 4.30: indexRoute.js - 683-696

```
1          for (let j = 0; j < 15; j++) {
2              sum += array[j] * weight[j];
3          }
4
5          let fit = {
6              fitness: Math.round(sum * 100),
7              email: resident[i].email,
8              name: resident[i].firstName + " " + resident[i].
                   lastName,
9              aalborgSection: resident[i].aalborgSection,
```

```
10              budget: resident[i].budget,
11          };
12          fitness.push(fit);
13  }
```

At this point in the code, the impact of the difference in answers as well as the weighting for each individual question has been calculated, which means they can be summed together to represent a fitness-value. Looking at the code snippet above, this is first done by initializing a for-loop, which iterates through the 15 questions and multiplies each entry in the array declared in listing 4.29 and the weight array. A new variable, fit, is now declared. This variable is then filled with the information to be displayed on the results page and the fit variable is hereafter pushed onto the fitness array, which can be seen on line 12 in the above snippet. After the fit variable is pushed onto the fitness array, the outer most for-loop is exited and the fitness array is returned for use elsewhere.

4.3.1 Server-side validation

One of the most important things when adding information to a database that a client has typed is server-side validation. One of the first things you learn as a web-developer is to never trust any data that originates from the client side, you could say that not validating user input is like trusting people and not installing locks on your house. If a server does not have server-side validation, the user would be able to do anything, you could be allowing users to drop entire databases(or worse, leak them), modify anything they like(or worse, read anything they like).

In our case, we have both client-side and server-side validation, but the client side is not enough as this is only plain HTML or JavaScript, and the user is able to turn off JavaScript in they browser and easily modify the HTML.
We use the Express middleware called Express-validator, this gives us the ability to validate and get some errors back that we can act on. We start by defining "check" and "validationResult" to be equal to "require("express-validator")".

Listing 4.31: Require express-validator - 17-17 - /routes/indexRoute.js

```
1  const { check, validationResult } = require("express-
       validator");
```

Once we have done that, we want to define what the express-validator should validate. We do that by passing different checks as seen in listing 4.32 as a parameter to the post request we make whenever the page gets called.

Listing 4.32: Express-validator check - 44-219 - /routes/indexRoute.js

```
1  router.post(
2      "/results",
```

```
3     [
4         check("firstName", "Enter first name").isString().
              notEmpty(),
5         check("lastName", "Enter last name").isString().
              notEmpty(),
6         check("gender", "Choose a gender").equals("1", "2"),
7         check("email", "Email is not valid").isEmail().
              normalizeEmail(),
8         check("age", "Choose an age").isIn(["1", "2", "3"]).
              exists().isString(),
9         . . .
10        . . .
11        . . .
12    ], async function (request, response) {
```

On line 4 and 5 we want to check the first and last name. As these two inputs in the form are text inputs there is not much we can do other than using .isString() and .notEmpty(). These two functions make sure that the input we get is a string, so text, and that the input is not empty as we do not want that. Another one to look at is line 8, this check uses .isIn([]) to see if the age input is either "1", "2" or "3", nothing else is able to be inputted into the database, the second function we use is .exists() this is only to check that there actually is an input called "age".

Listing 4.33: /results post request body - 219-242 - /routes/indexRoute.js

```
1  async function (request, response) {
2      const errors = validationResult(request);
3      const db = client.db("myFirstDatabase");
4      let alert = await error(errors);
5
6      if (errors.isEmpty()) {
7          await client.connect();
8          await db.collection("residents").insertOne(request.
               body);
9          console.log("Inserted applicant");
10         let resident = await db.collection("residents").find
               ({}).toArray();
11         let weight = await weighting(request);
12         fitness = compatibility(request, resident, weight);
13         client.close();
14     } else {
15         fitness = [];
16     }
17
18     response.render("pages/results", {
19         applicant_name: request.body.firstName,
20         resident: fitness,
```

```
21          alert,
22      });
23  }
```

If anything would happen with the checks and the user inputs, something we do not expect or want it creates an error, these errors get assigned to the variable errors. All of our database and fitness calculation takes place in the body of this post request, and we do not want that to run if there is an input error so that the database could be filled with a bad input, we therefore encompass it in an if-else statement. The if-else statement checks if the variable errors is empty, if it is, it runs the whole database and calculation, since we assume that the input is valid, if there is any errors it jumps to the else statement and assigns an empty array to fitness.

Listing 4.34: Error function - 244-251 - /routes/indexRoute.js

```
1  function error(errors) {
2      let alert = [];
3      if (!errors.isEmpty()) {
4          alert = errors.array();
5          console.log(alert);
6      }
7      return alert;
8  }
```

The listing above is a function called error, its input is the variable "errors" from listing 4.33. The function defines an empty array to "alert" it checks if "errors" is not empty, and assigns "errors" as an array to "alert" and returns "alert". This "alert" array we now have inside the listing 4.33 we pass in our response.render call. The render call is what happens whenever the post request is done, and the reason behind we pass the array variable "alert" is because we use that to create the errors on the client side you saw in section 4.2.2, so that the user is able to see which errors there are.

4.3.2 MongoDB

In the section "Routes" it was briefly mentioned that mongoDB was chosen as the database for the project, but it was never explained what mongoDB actually is. Firstly, mongoDB is a non-relational database, meaning that it does not conform to traditional use of schemas with rows and columns. Instead mongoDB allows for much more flexibility. In the case of this project, data is stored as simple key/value pairs. The way the key/value pairs are set up can be seen in the code snippet below. These key/value pairs are then stored in so-called documents, which sit in a collection. Another great positive for mongoDB is the possibility to easily use locally as well as remotely, with the ability for multiple people to manage the data.

For the use in a project like this, mongoDB being free to use is a positive. Should the project be taken to a bigger scale, then mongoDB would still be able to keep up in terms of storage and speed. In the code snippet below the method of accessing and manipulating the database is presented.

Listing 4.35: indexRoute.js - 219-242

```
1  async function (request, response) {
2          const errors = validationResult(request);
3          const db = client.db("myFirstDatabase");
4          let alert = await error(errors);
5
6          if (errors.isEmpty()) {
7              await client.connect();
8              await db.collection("residents").insertOne(
                   request.body);
9              console.log("Inserted applicant");
10             let resident = await db.collection("residents").
                   find({}).toArray();
11             let weight = await weighting(request);
12             fitness = compatibility(request, resident,
                   weight);
13             client.close();
14         } else {
15             fitness = [];
16         }
```

If the if-statement returns true, the order of operation is as follows:

1. Line 7: Connect to the database
2. Line 8: With "insertOne", insert data coming from the request body in the collection "residents"
3. Line 9: Acknowledge the data was inserted with console.log
4. Line 10: For easier use in the program, the data in the collection is assigned to the variable "resident" (which is an array)
5. Line 11: Using the request, the weighting for the individual questions are brought in and assigned to the "weight" variable.
6. Line 12: Along with the request, with resident and weight variables are now used in conjunction with the compatibility function to fill the "fitness" array with the fitness value to be presented on the results page.

Lastly the connection to the database is closed and the else-part of the if-statement is reached. Had the if-statement returned false, the fitness array would have been set to be empty.

Chapter 5

Testing

As we spend time learning about different programming languages, it is also a good idea to spend some time to learn about the testing methods. It is most likely to happen, that the written program does not meet the expectations of the programmer themselves. This of course becomes a big deal when considering there might also be requirements set by an outside person, such as a client. Different methods of testing is then brought in to make sure the programs reliability, security and performance is satisfactory [32].
There exist different types of testing:

- **Unit testing**; is a method that tests the individual parts of a program by isolating a section of code, to guarantee its correctness, during the development phase [36]. This could potentially be done by separating the source code into the individual functions and then testing each one.
- **Integration testing**; will as the name might suggest focus on the integration of code. This is mainly done since different programmers might have written code that interact with each other [32].
- **Acceptance testing**; can be seen as the final approval of a product. Here the product thoroughly tested to make sure it is ready to be given to the end-user while still satisfying the acceptance criteria [3].

5.1 Jest testing

There are different testing libraries for a JavaScript based program. Jest is one of the most popular and highly preferred frameworks that requires zero configuration, it is fast, it uses built in tools for code coverage, and it includes snapshot testing support [22]. These specifications were enough for the group to choose Jest as a testing framework. Jest was used to test 2 functions, which of course is only

a small part of the program as a whole, but the technique behind using Jest to perform unit/integration testing remains virtually the same.

5.1.1 Testing of x1 function

In order to perform Jest testing, a folder called Testing is created, which will contain a JavaScript file with the function to be tested, and it will have a fileName.test.js file. The functions tested are the x1 and weighting functions, which can both be found in indexRoute.js.

Listing 5.1: Testing/x1.js - 1-31

```
1  function x1(q9x, q10y) {
2      let delta, x1;
3
4      if (q9x > q10y) {
5          delta = q9x - q10y;
6      } else {
7          delta = q10y - q9x;
8      }
9
10     switch (delta) {
11         case 0:
12             x1 = 1;
13             break;
14         case 1:
15             x1 = 0.5;
16             break;
17         case 2:
18             x1 = 0;
19             break;
20         case 3:
21             x1 = -0.5;
22             break;
23         case 4:
24             x1 = -1;
25             break;
26     }
27
28     return x1;
29 }
30
31 module.exports = x1;
```

On line 31 in the code snippet above, the function is exported, allowing use in other files.

Listing 5.2: Testing/x1.test.js - 1-7

```
1 const x1 = require('./x1');
2
3 test('x1 testing', () => {
4     expect(
5         x1(5, 1)
6     ).toBe(-1)
7 })
```

In the above code, the function is first imported on line 1. The keyword test is then used to create a test block. First parameter of the test keyword is the name of the test, followed by the function to be run - here the function expect is run. Paired with the expect function is the toBe function. Between the expect and toBe functions, the function to be tested is placed with parameters, that are realistic to the program as a whole. It can be seen on line 5 in the above code, that 5 and 1 are used as parameters for x1, used to represent input from the applicant and resident in the questionnaire. To make sure the function runs as expected, the parameter passed to the toBe function is -1. Using Jest with full coverage, it can be seen that not every line is covered. This is merely because there has not been given an expected result on every case in the switch. The group deemed this acceptable since the test is still passed.

5.1.2 Testing of weighting function

Listing 5.3: Testing/weighting.js - 1-38

```
1  function weighting(request) {
2      let weight = [];
3
4      const x = [
5          "planVisitorsRoommate",
6          "preferStudy",
7          "bedtime",
8          "roommateGender",
9          "roommateLanguage",
10         "preferredAgeRange",
11         "btnradio4",
12         "roommateMajor",
13         "roommateSemester",
14     ];
15
16     let vigtigID = [
17         Q1 = "preferStudy",
18         Q2 = "bedtime",
```

```
19          Q3 = "btnradio4",
20          Q4 = "roommateGender",
21          Q5 = "preferredAgeRange"
22      ];
23
24      for (let i = 0; i < 15; i++) {
25          for (let j = 0; j < 5; j++) {
26              if (vigtigID[j] === x[i]) {
27                  weight[i] = 0.1;
28                  break;
29              } else {
30                  weight[i] = 0.05;
31              }
32          }
33      }
34
35      return weight;
36  }
37
38  module.exports = weighting
```

The code snippet above shows the weighting function with accompanying mock input. As can be seen on line 1, the function takes a request as the only parameter. This mock input is therefore made to simulate the data that would normally be coming from the request. On line 24 and 25 an inner and outer for-loop is set up. The outer loop will iterate through the questions in the questionnaire and the inner loop will run through the 5 questions, that the applicant chose at the end of the questionnaire to raise the weighting of the corresponding question. Inside both the loops, comparing each question in the questionnaire with the questions selected to have a higher weighting. This can be seen on line 26 where, it can be seen on the array indexes that the variables i and j in the loops are also used to iterate through the arrays. Should the if-statement return true, the weighting on the i'th index of the weight array is then set to 0.1 and the if-statement is broken out of. If the if-statement returns false, it will fall through to line 30, where the weight is set to 0.05 on the i'th position. lastly once both loops have finished running, the weight variable is returned.

Listing 5.4: Testing/weighting.test.js - 1-21

```
1  const weighting = require('./weighting');
2
3  const vigtigTest = [
4      "preferStudy",
5      "bedtime",
6      "roommateLanguage",
7      "roommateGender",
```

```
8       "preferredAgeRange"
9   ]
10
11  test('weighting test', () => {
12      expect(
13          weighting(vigtigTest)
14      ).toEqual([
15          0.05, 0.1, 0.1, 0.1, 0.05,
16          0.1, 0.1, 0.05, 0.05, 0.05,
17          0.05, 0.05, 0.05, 0.05, 0.05
18      ])
19  })
```

In the weighting.test.js file, the weighting.js file is required on line 1. Another array representing the 5 questions from the end of the questionnaire is then made. On line 12 the test function is called. Once again the expect function is called followed by the function to be tested, the weighting function. Instead of using toBe with expect, the toEqual keyword is used instead and is said to be equal to the 15 questions ran through in the weighting.js file.

5.1.3 Jest testing output

When testing with Jest, the result of the test is presented as can be seen below:

```
PASS Testing/weighting.test.js
PASS Testing/x1.test.js
-------------|---------|----------|---------|---------|-------------------
File         | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------|---------|----------|---------|---------|-------------------
All files    |   68.96 |    44.44 |     100 |   66.66 |
 weighting.js|     100 |      100 |     100 |     100 |
 x1.js       |   43.75 |    28.57 |     100 |   43.75 | 7,12-22
-------------|---------|----------|---------|---------|-------------------
```

Figure 5.1: Out of the Jest testing

Like was mentioned earlier, not every line of code is covered in the testing of x1.js, but have can be gathered from this output is that both the functions have passed the Jest testing.

Further testing

The process of testing code like it is done here, is in no way exclusive to just these sections of code. In an ideal scenario, all the code would be covered and verified.

In this way the requirements of integration testing and thereby unit testing as well as acceptance testing would be covered.

Chapter 6

Discussion

Throughout the project the group has been limited to the problem matching roommates in the best way possible. This has lead to the current solution, but was not without wondering such as what is presented in 6.1.2 and 6.1.3. Additionally, how the group would proceed beyond the end of the project has been discussed.

6.1 Status of the Project

Lack of systems to distribute student accommodations in Denmark lead the group to design the related application. According to the benefits of having roommates and its effects on academic performance, the project led to model a platform of finding roommates for students. Considering the conflicts and challenges that especially the international students struggle with when they study abroad, convinced the group to create the platform based on preferences, as it can help students to have a part in selecting the people around them. Implementing such a platform requires a big database including lots of information about the accommodations and students already living in Denmark, but considering the limited time the group decided to focus on Aalborg city.

6.1.1 Questionnaire

A questionnaire was required to specify students preferences. The group took the other countries student dormitory distribution questionnaires into account and developed a 15 item questionnaire, see section 3.4.3. Having such a questionnaire, will result in finding the most compatible roommate based on the preferences chosen by the students themselves. The last part of the questionnaire that asks about five most important questions, makes it more optimized and effective to find the best matching option among the existing possibilities.

6.1.2 Is matching of roommates always a good idea?

As mentioned before, the purpose of this project is to match students with the most compatibility. There are pros and cons of pairing students in such manner. The advantages are described in the report, but there also exists some documents that demonstrate impacts of diversity. As the purpose of implementing such platform was to accommodate students who are looking for roommates with similar habits, the disadvantages are not mentioned in this report.

6.1.3 Reflection on the project

Working on this project as a group gave the individual a lots of achievements, group working skills beside the knowledge we gained of all the research we did and also the improvement of programming skills are some of them to mention. The most challenging part for the group was the programming, as for the purpose of the project, we were required to use tools, software and techniques that we were inexperienced in. The teamwork was full of advantages for us, as we could help and learn from each other in different situations. As an unpleasant experience, we can mention the separation of us three from the main group, almost in the middle of the semester. However the group of three worked well, but we could have benefited more with a group of seven under certain circumstances. But to avoid facing such problems, we plan to formalize the group contract for the next semester having more specific rules.

6.2 Future Development

As is often the case with a project like this one, imagining how the project would go on to evolve beyond the time constraints is interesting. Doing so, could give good insight in how the needs and missing parts of a project and its product could actually be carried out in reality. To help do this, it is helpful to take a look at the MoSCoW-table (see table 3.1, which was created throughout the project.)
The items that can be mentioned as future work are as follow:

- Using real data of current residents
- Implementing the application with national coverage.
- Including the currency selection
- Including the language selection
- Including a personality test
- Sending the related application link of the accommodation where the compatible roommates live.

- Support of applying for multiple roommate
- Including the possibility of define weightings to the applicant himself.
- Including questions about accommodation type
- Adding the possibility of applying for an accommodation without any roommates

It can therefore be said that looking at the future development is a continuation of the section "Status of the Project".

Chapter 7

Conclusion

While the implications of living with roommates have an impact on a seemingly ever increasing amount of people, especially students, the method of grouping has not seen much innovation. The justification of this, is that no one has succeeded in supplying a sufficient solution, preferably one that remains useful no matter the location.

Based on this lack of solution, the group developed a web application to demonstrate the functionality of calculating compatibility between an applicant and a resident. Calculating compatibility with input stemming from a form with the representing personal preferences allows for the use of a remote database to keep a record of existing applicants and residents. In addition a section on the website was made as a guide to international students. With such an implementation, a possible solution to the problem of pairing compatible roommates has been suggested. Provided the needed infrastructure, the solution would with customization be suitable for domestic as well international use.

Despite the disadvantages of pairing roommates of similar preferences not being mentioned in the report, the potential gain and advantage a similar solution could provide should speak for itself. Additionally, by catering to and embracing international students, they are more inclined to stay in Denmark after their education is over, thereby justifying the great amount of money spent on educating a student at a university, which is preferable to having them bring their newly acquired education elsewhere.

It is because of these reasons that personalized testing, where personal preferences are welcome, should be adopted and further investigated in relation to schooling.

Bibliography

[1] *10 great apps and website for finding a roommate*. URL: https://www.rent.com/blog/best-apps-finding-a-roommate/.

[2] *A comparison of mental health in dormitory and non dormitory students*. URL: https://www.cibtech.org/sp.ed/jls/2014/03/JLS-043-S3-057-SARA-A-STUDENTS.pdf.

[3] *acceptanceTesting*. URL: https://softwaretestingfundamentals.com/acceptance-testing/.

[4] AKU-Aalborg. *Make an application*. URL: https://vl.aku-aalborg.dk/OpretAnsogning.aspx.

[5] *An easy guide to help you decide which machine learning algorithm to use for your business problem*. URL: https://edvancer.in/an-easy-guide-to-help-you-choose-which-machine-learning-algorithm-to-choose-for-your-business-problem/.

[6] Patrick Atack. "Danish student housing investment opportunities highlighted". In: (2018). URL: https://thepienews.com/news/danish-student-housing-opportunities-highlighted/.

[7] *Brute force algorithm*. URL: https://www.educba.com/brute-force-algorithm/.

[8] Chih-Ching Chang and Che-Chern Lin. "Dormitory Assignment Using a Genetic Algorithm". In: *Applied Artificial Intelligence* 35.15 (2021), pp. 2276–2297. DOI: 10.1080/08839514.2021.1999595. eprint: https://doi.org/10.1080/08839514.2021.1999595. URL: https://doi.org/10.1080/08839514.2021.1999595.

[9] College Choice. *The roommate effect*. URL: https://www.collegechoice.net/college-life/the-roommate-effect/.

[10] David Cochran. *Twitter bootstrap web development how-to*. Packt Pub., 2012.

[11] Hilbert College. *Entrollment Kit - Roommate Preference Questionnaire*. URL: https://www.hilbert.edu/admissions-financial-aid/accepted-students-new/enrollment-kit/roommate-preference-form.

[12] Wells College. *Housing Roommate Questionnaire*. URL: https://www.wells.edu/files/public/forms/Housing_Roommate_Questionnaire-fillable.pdf.

[13] *College Enrollment in the US 1965-2019*. URL: https://www.statista.com/statistics/183995/us-college-enrollment-and-projections-in-public-and-private-institutions/.

[14] Danish agency for higher education and science. *Living in Denmark*. URL: https://studyindenmark.dk/live-in-denmark/housing-1.

[15] *EJS, url =*.

[16] David Flanagan. *JavaScript: the definitive guide*. 2013.

[17] Student Survival guide. "How to find student accommodation in Denmark". In: (2021). URL: https://www.studentsurvivalguide.dk/posts/how-to-find-student-accommodation-in-denmark.

[18] Thomas H.Cormen et al. *The Introduction to algorithms Book*. The MIT press, 2009.

[19] *ILOG CPLEX optimization studio*. URL: https://www.ibm.com/docs/en/icos/12.9.0?topic=types-decision-variables.

[20] *International Students' Perspectives on the Importance of Obtaining Social Support from Host National Students*. URL: https://files.eric.ed.gov/fulltext/EJ1095801.pdf.

[21] *International Students' Survival Guide to life in Denmark*. URL: https://esknet.dk/wp-content/uploads/2021/05/Survival-Guide_2021.pdf.

[22] *Jest tutorial*. URL: https://www.softwaretestinghelp.com/jest-testing-tutorial/.

[23] Jasko Mahmutovic. *7 Ways to Get the Truth on Surveys (+ Reasons for Lying on Surveys)*. URL: https://www.surveylegend.com/customer-insight/lying-on-surveys/.

[24] Azat Mardan. *Express. js Guide: The Comprehensive Book on Express. js*. Azat Mardan, 2014.

[25] Dr. Petra Menz, Nicola Mulberry, and David Guichard. *The Calculus early transcendentals Book*. Lyryx, 2017.

[26] *Optimization tutorial-defining constraints*. URL: https://www.solver.com/defining-constraints.

[27] *Room and roommate selection guide for first year students*. URL: https://www.lsu.edu/reslife/files/reslife_freshmen_room_selection_guide.pdf.

[28] *Roommate relationships*. URL: https://www.mtu.edu/deanofstudents/faculty-staff/intervention/resources/roommate-relationships/.

[29] Jerome L. Short Jeffrey W. Pollard Sarah E. Erb Keith D. Renshaw. "The Importance of College Roommate Relationships: A Review and Systemic Conceptualization". In: (2014). URL: https://mason.gmu.edu/~jshort/Erb%20Roommate%20Relationships.pdf.

[30] Statista. *Number of registered university students in Denmark*. URL: https://www.statista.com/statistics/1111224/number-of-registered-university-students-in-denmark/.

[31] Studyportals. *study in Denmark:Tuition fees and living costs*. URL: https://www.mastersportal.com/articles/342/study-in-denmark-tuition-fees-and-living-costs.html.

[32] *Testing your JavaScript code*. URL: https://codeburst.io/testing-your-javascript-code-95c171c71647.

[33] *The impacts of roommates on first-year students*. URL: http://www.skyfactor.com/wp-content/uploads/2017/02/2017-Roommate-Research-Note.pdf.

[34] *The importance of collage roommate relationship*. URL: https://www.researchgate.net/publication/263500255_The_Importance_of_College_Roommate_Relationships_A_Review_and_Systemic_Conceptualization.

[35] Ungdomsboliger. *Find accommodation*. URL: https://www.ungdomsboliger.dk/?id=navigationskort&lang=en.

[36] *Unit testing tutorial*. URL: https://www.guru99.com/unit-testing-guide.html.

[37] *What is argmax in machine learning*. URL: https://machinelearningmastery.com/argmax-in-machine-learning/.

[38] *Yes, your roommate can affect your mental health*. URL: https://www.refinery29.com/en-us/roommates-mental-health-effects.