

Unibook - Design and implementation of an IT-System



P4 PROJECT
GROUP BI409F20
INFORMATIONSTEKNOLOGI & INFORMATIK
AALBORG UNIVERSITY
MARCH - JUNE 2020



AALBORG UNIVERSITET
STUDENTERRAPPORT

Information technology and Informatics

Department of Computer Science

Aalborg University

Selma Lagerlöfs Vej 300

9220 Aalborg East, DK

Title:

Unibook - Design and implementation of an IT-System

Project:

P4-project: Development of a database system for a specific application.

Project period:

Februar 2020 - Maj 2020

Project group:

BI409F20

Participants:

Kasper Madsen

Mads Hermann

Joachim H. Ø. Juul

Jacob H. Christensen

Supervisor:

Van Ho Long

Abstract:

The project's purpose has been to develop a database system with a specific use case and incorporate the principles and concepts of object-oriented analysis, design, and programming into the development process. The analysis and design of this project are conducted according to the Object-Oriented Analysis and Design method introduced in the course of the same name as well as visual design methods introduced to us during our 2nd semester. The system design is implemented using Bootstrap, Entity Framework, and ASP.NET, which help us build a website using HTML, CSS, and C#. The result of this project is unibook.me, an online marketplace for selling used books. The website is connected to a database that stores information about the different books, listings, and users.

Keystrokes: ****

Number of pages: ****

Appendix: ****

Finished 27/05/2020

The contents of the paper are publicly accessible, but publication (with citations) can only happen in agreement with the authors.

Preface

This report is written by 4th semester students enrolled in the bachelor of Information technology at Aalborg University, as part of a study program. It is based on the superior topic of *designing a database system for a specific application*.

This project is an extension of previous semesters, which focused on the ability to analyze and construct systems for a specific context with a general understanding of programming in mind.

We wish to thank our supervisor Van Ho Long for academic support as well as constructive criticism.

Joachim Hylleberg Ørsøe Juul

Kasper Medom Madsen

Jacob Hougaard Christensen

Mads Hermann

Reading guide

This report is divided into sections that should be read in chronological order as follows:

- Introduction
- Methodology
- Problem analysis
- System analysis and design
- Implementation
- Evaluation
- Conclusion
- Discussion

Starting with an introduction and the scope of the project. After that, the methodology is presented, which acts as the foundation of the approach. Then follows the problem analysis, which presents the problem statement at the end. The analysis and design of the system are followed by the implementation of the system, which is then evaluated and concluded. Finally, the findings are discussed. The citations are based on the Vancouver method, which uses numeric sequences in the bibliography. References to documents and other external material are done in the appendix.

Contents

1	Introduction	1
1.1	Project scope	1
2	Methodology	2
2.1	Iterative approach	2
2.2	Designing interactive systems	2
2.3	Object-oriented Design and Analysis	3
3	Problem analysis	5
3.1	Initial problem	5
3.2	Understanding	5
3.2.1	PACT analysis	6
3.2.2	MoSCoW	9
3.2.3	Rich picture	9
3.3	Problem statement	10
4	System Analysis and Design	11
4.1	FACTOR	11
4.1.1	System Definition	12
4.2	Problem-domain Analysis	12
4.2.1	Class Diagram	14
4.2.2	Behavioural Pattern	15
4.3	Application Domain	16
4.3.1	Usage	17
4.3.2	Functions	19
4.4	Interface	20
4.4.1	Sketches and wireframes	20
4.5	Architectural design	27
4.5.1	Criteria	27
4.5.2	Components	29
4.6	Component Design	30
4.6.1	Interface component	31
4.6.2	Database component	31
4.6.3	Model Component	32
4.6.4	Function component	33
5	Implementation	37
5.1	Languages	37
5.1.1	HTML	37
5.1.2	CSS	37
5.1.3	C#	38

5.2	Frontend	38
5.2.1	Bootstrap	38
5.3	Database and server	45
5.3.1	MySQL	45
5.4	ASP.NET	49
5.4.1	MVC and Razor Pages	49
5.4.2	The default classes in ASP.NET	52
5.4.3	Startup class	53
5.5	Functionality	55
5.5.1	Searching	55
5.5.2	Creating	57
5.5.3	Editing	60
5.5.4	Calculate	62
6	Evaluation	64
6.1	Usability Evaluation	64
6.1.1	Method	64
6.1.2	Debriefing - System Usability Scale	68
6.1.3	Usability Problems	69
6.1.4	Problems with the usability test	71
7	Conclusion	72
8	Discussion	73
8.1	System Analysis and Design	73
8.2	Implementation	73
8.2.1	Security	73
8.3	Evaluation	74
8.4	Covid-19	74
	Bibliography	75

For this project, the aim is to develop a system connected to a dedicated database and serve as a possible solution for a problem that has been identified beforehand. To do this, we determine an overall definition of a system that would solve the problem. This definition is based on a PACT and FACTOR analysis, which provides us with information regarding the requirements of the system. The system definition is then used as the foundation for the object-oriented analysis and design, in which the problem and application domain is analyzed further. Likewise, the classes, functions, and components of the system are designed and visualized. After this, the implementation of the design can begin, and the project enters the development phase. Once this is done, and the system has taken shape, we test and evaluate the system and determine whether or not it fulfills the requirements. Based on this, we create a basis for future work and discuss the conclusion and project as a whole.

1.1 Project scope

This project will focus on the development of a system connected to a database, based on a case that was derived, from a problem we faced during our time at Aalborg University. Therefore, the project work will not involve any partners or clients, and we will have to base most of our requirements and evaluation on internal assumptions and predictions of what is expected and needed of the system. We are aware of the consequences of this approach and will be discussing it, when relevant to the results and conclusions we present. The case is not necessarily limited to only be relevant for students at Aalborg University. However, due to practical considerations, we have decided to focus on our own experiences as students of Aalborg University, which will serve as the primary context for the scope of the project.

Due to the national prohibition of physical contact caused by the Covid-19 virus outbreak, we have been forced to limit the testing of the system to unconventional testing methods. Under normal circumstances, the testing would have included a usability test with the physical participation of potential users of the system, but this is not a possibility due to the unusual situation. Therefore, the testing is done online but based on the same principles as a regular usability test. This will be included and discussed further in the relevant sections of the report.

Methodology 2

Our selection of methods and approach is based on the courses attended during the fourth semester, which consists of Database Development (DBU), Fundamental Object-oriented Programming (GOOP), and System Analysis and Design (SAD). Our approach to this project is based on our knowledge and experience with the development of IT-systems.

2.1 Iterative approach

For this project, we have utilized a primarily iterative approach to analyze the problem and develop the solution. This iterative approach is a standard method for developing IT-systems and software, as it ensures that nothing is set in stone and that developers can improve the system design over time as new information and knowledge is uncovered. However, due to the hard deadline that exists for this project, our iterative approach will be limited to a finite amount of time being spent on the different phases of the project. This will introduce characteristics of a waterfall model, where the project follows a sequence of planned phases. By mixing an iterative approach with parts of a waterfall model, it allows us to process information as we move along while having a plan for when the different phases of the project should be completed. [1] This approach also works well with the top-down approach presented with the Object-Oriented Analysis and Design method, which is the basis for the analysis and design phases of this project. [2] Finally, our primary reasoning behind the choice of this approach is practicality. It allows for flexibility in the work that we perform, as mistakes or poor decision-making can be corrected at a later time, limiting the impact of the lack of information that might exist in the early phases of the project. It ensures that we remain critical of our work throughout the project, increasing the quality assurance of the completed solution at the time of the deadline.

2.2 Designing interactive systems

When working with the design and development of IT-systems meant to be used by people, one should consider the field of Human-Computer Interaction (HCI). One way to approach HCI is to consider how an IT-system can be designed with the human user playing the central role in the design considerations. In the book *Designing Interactive Systems*, David R. Benyon proposes a method for system design where the human user plays such a central role. Benyon's method focuses on human ability and preference as well as understanding the context from an accessibility, usability, and acceptability point of view. [3]

For this project, we are using a PACT-analysis (People, Activities, Contexts, Technologies) based on Benyon's principles for establishing an understanding of the context of the system design. This provides a thorough and useful understanding of the characteristics of the users, the users'

activities, the context of these and the technological consideration related to the system's purpose and function.[3]

To create a practical structure regarding the different requirements of the system and the priority of these, we will use the MoSCoW model when establishing the priority hierarchy of the requirements. This way, we will know which requirements are a must-have, should-have, could-have, and won't-have when planning our time in the development phase.[3]

To visualize the design of the system's user interface, we will use sketches and wireframes in what Benyon describes as the envisionment phase. Sketches will aid the creativity of the visual design, and the wireframes will be helpful in the implementation phase as it will serve as a rough blueprint of the system's final user interface.[3]

Overall we will be looking to Benyon for inspiration in our understanding and envisionment of this project and selectively combine principles from his method with the Object-Oriented Analysis and Design (OOA&D) method. The reasoning behind this is that OOA&D works well when the project involves object-oriented programming languages such as C#. However, we would argue that Benyon's principles of understanding and envisionment supplement the initial phases of OOA&D well, and therefore we have decided to pursue a fusion of the two.

2.3 Object-oriented Design and Analysis

This project is based on Object-Oriented Analysis and Design, which is a method for software development that is based on objects as building blocks. Every object is described with a specific identity, state, and behavior. These objects help the user express their point of view. User involvement is a continuous part of the OOA&D method, which differentiates between the problem-domain and the application-domain. The most significant advantage of OOA&D is the ability to express the phenomena as natural language, instead of being overly technical. This enables user involvement to be more specific, and it provides context. As systems switch from automation and uniform data to more complex organizational tasks, there needs to be equal focus on the system and the context, as well as cooperation between developers and users.[2]

The OOA&D method mainly consists of a series of activities that serve as a part of an iterative process. The primary purpose of these activities is to use the information related to the system's context and definition as well as establishing an accurate description of classes, objects, and events. The four main activities consist of small, less abstract tasks, and the organization of these tasks depend on the strategy that is most suited for a specific project. This means that the order in which these tasks are completed can be arranged to fit different situations related to different types of projects.

An example of a specific approach to this method is a traditional top-down approach, which is visualized on figure 2.1.

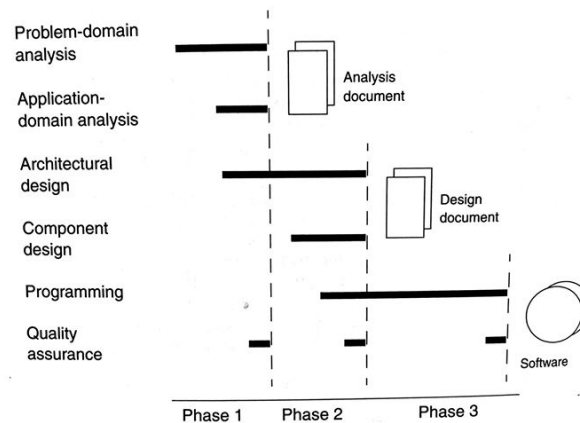


Figure 2.1. The traditional top-down approach

In this approach, the process begins with the problem-domain analysis, which leads to the other activities by using the information created as a result of previously completed tasks. This approach is still fundamentally iterative as its use in practice, requires the project team to go back and forth between the different activities. When new information reveals itself the evaluation of the information changes based on new knowledge. Ultimately the choice of approach must be determined based on the circumstances of the context and which uncertainties the project faces. When these circumstances and uncertainties have been identified, the method and process can be organized accordingly.

A differentiation is made between what the system does and how it operates as the developer models the context and their understanding of the system's practical application. These are called the problem-domain and the application-domain. The problem-domain is described as *"That part of a context that is administered, monitored or controlled by a system"* [2]. The operation of the problem-domain is controlled by the application-domain, described as *"The organization that administrates, monitors or controls a problem-domain"* [2]. Analyzing the domains independently and thoroughly allows the developers to create a system modelled explicitly for the given context. The success of a system relies significantly on the ability to intertwine these analyses.

Modeling of the system is, as mentioned before, equally important. Here the developers differentiate between architectural design and component design. The system and its architecture should be easily understood and flexible, as well as provide a basis for decision making. A system is defined as *"A collection of components that implement modeling requirements, functions, and interfaces"* [2]. The primary purpose of this architectural design in OOA&D is to design a system without any significant uncertainties by giving an accurate overview of the system. The component design is about identifying relevant components, which is a highly iterative process in OOA&D. Components are described as *"A collection of program parts that constitutes a whole and has well-defined responsibilities"* [2]. Systems often have a user interface, functions, and a model component. By decreasing complexity in components and enabling flexibility, components can be reused in OOA&D.

Problem analysis 3

3.1 Initial problem

The Danish educational system relies heavily on the use of books as a part of the learning process for most subjects taught on all levels of education. These books are provided free of charge for students up until they reach the higher levels of their education, at which students will have to buy the necessary books themselves. The magnitude of this expense can vary from study to study but is an expense nonetheless for every student on universities throughout the country. Furthermore, the custom of most classes and subjects is for each student to own a copy of the necessary reading material and buy it brand new. This leads to a wasteful and unsustainable consumer culture where books with a lot of remaining use are discarded or forgotten, once they are no longer useful to the owner.

This is not to mention the environmental footprint that the production and distribution of books have. Aalborg University continuously strives to offer sustainable education, ranking 23rd on the Times' Higher Education sustainable report of 2020. This project wishes to make a sustainable contribution to the problem revolving printed books, through the online book marketplace: Unibook[4].

Combining these two problems results in the demand for a solution that can provide students with a cheaper alternative to buying brand new books and getting more use out of the books they are no longer using. Such a solution is an online marketplace specifically designed for students where it is possible to sell and purchase used textbooks.

An online marketplace is not a revolutionary solution, and there are other such websites in existence online. However, it is few that are explicitly designed for selling books, and that is mainly targeted towards university students. Therefore, we argue that a gap in the market could be filled by our online marketplace, which justifies its development.

3.2 Understanding

Benyon divides the design process into four activities: Understanding, Design, Envisionment, and Evaluation. For now, the focus will be on understanding. Before designing an interactive system that is human-centered, it is of utmost importance that the designer has a great understanding of the *people* who will be involved with the system, the *activities* that are the focus of the design, the *contexts* in which those activities take place and the implications for the design of *technologies*: 'PACT' [3]. Furthermore, the problem-domain and application-domain analysis will also gather a lot of information and understanding of the system and what features and functions it should support. Together with the PACT analysis, the outcome of this will be a set

of system requirements. These requirements will be prioritized with the MoSCoW rules. The MoSCoW rules classify the requirements into:

Must have: these are the requirements that are essential and the backbone of the system. Without these, it won't be workable and useless.

Should have: if more time is available at hand these are the next requirements to incorporate although the system should be useable without them.

Could have: these requirements are of lesser importance and can, therefore, be left out of the design more efficiently.

Want to have but won't have this time: requirements that can wait for a later iteration of the system.

This method is a way to prioritize requirements so that a business can control the cost of development [3].

3.2.1 PACT analysis

When using a PACT analysis, we are making sure that our design process is human-centered, which is essential when designing an interactive system for people to use. PACT is an acronym for the four words that are people, activities, context, and technology.

Therefore, it is essential to consider this. The people that must make use of the system can vary a great deal. It takes no scientist to notice that humans differ quite a lot in various aspects such as tall, low, short-sighted, long-sighted, handicaps, etc. Therefore, a designer of an interactive system must have this in mind.

Furthermore, these different activities have diverse characteristics regarding the task the system should support or do. Thus, the designers need to ensure that the functions that have a high frequency of use are easy to do, and those with a lower frequency should be easy to learn. Furtherly, the designers need to have in mind that the system will experience different peak hours of activity. This would mean that a system might work well when there is a low usage but might face issues when the usage is high.

All these activities would always happen in a context. Benyon refers to three different types of contexts that are distinguishable. These are organizational context, social context, and physical circumstances. In this project, the organization is the designers of the system, and as such, it will not have been something that would require going into further. The physical circumstances that a designer needs to consider are such things as how sunlight will affect the screen of the user device. Are there any security measures that should be taken into consideration? Benyon uses the example of an activity where the user has to withdraw cash from an ATM. The designer should, in such a case, take location, camera observation, and maybe also distance markers into his design process. Further, using the example for social context, the considerations would include time taken for the transaction and the need for the queue [3].

By taking these considerations into account and using PACT as a framework for our design process, we understand the system's potential users. We can, therefore, cater the design to their needs.

People

The system will consist of two user groups - a primary and a secondary. The primary user group will be the users of the system, and the secondary user group will comprise the administrators of the IT system.

The primary group will be the ones who are making use of the system to buy and sell books through listings. They will also be the ones responsible for rating each other after a deal is completed. The secondary group will have the responsibilities for all the administrative tasks, including the moderation of listings and users of the primary group.

The primary user group of this system is students at universities in Denmark. Based on the statistics pulled from Statistikbanken (graph below), we can conclude that our primary user group will be in the age group of 20-29. The secondary user group will most likely be based on the same demographic as the main one, and will initially be based on volunteers. It can also be expanded to include people who are employed to moderate the site.

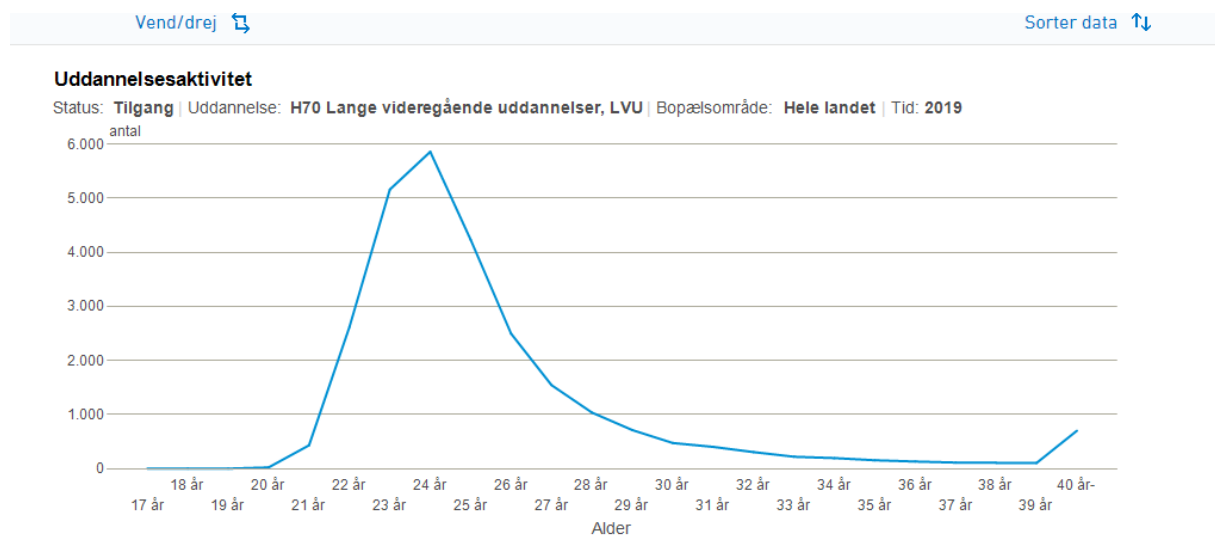


Figure 3.1. Graph of different ages at danish universities.

The following characteristics can also describe the primary group and these characteristics must be kept in mind while designing the system:

- There will be a varied understanding of IT-systems.
- Not all of the users will understand Danish.
- The common language would be English.

The following characteristics can also describe the secondary group and these characteristics must be kept in mind while designing the system:

- The primary responsibility for the moderation of the listings.
- Experience in moderating online boards etc.

The users are, for the most part, bilingual. The common language for the user group will be English. It would be 'nice to have' support for other languages. The system must also be designed

according to those who have poor spatial attention. This is done through the incorporation of proper signage and clear direction of where to go in the system for the various functions it supports [3].

The most common physical attributes the system has to take into consideration is color blindness and visual impairment. These two types of handicaps are the most relevant to take into account when designing the system. This is also done through proper signage and clear direction but also the consideration of design colors and size of signs. Furthermore, we need to make sure that the design of the system's functions are large enough for our fingers to hit them if the system is used on a smartphone [3].

Activities

For this system, there are certain activities that the users will take part in that we, as developers, need to understand. Activities such as making a new listing within the system and searching for specific books must be simple and easy to do since these activities will be carried out by the user frequently. Other functions that will not be used as often, and for the most part, rarely used should not be the main focus on making it easy to undertake, but they should be easy to learn either way. These include but are not limited to: Reporting a user to the moderators, contacting support, etc. The system is to be used daily, as it is a marketplace for buying and selling books. This means that the system should be capable of being up 24/7 and should have minimal downtime.

We also need to consider whether or not the user might want to be able to turn their upcoming listing into a draft while making it, as they may be interrupted in the making of the listing, and wish to continue with the making of the listing at a later time. Another activity we need to consider is the possibility for the user to make mistakes within the system when they are creating a listing. The users should be able to edit their listing if they spot an error they made within the original listing. This could be a typo, a wrong selection of book information, etc.

Context

Regarding this system, both the physical environment and organizational context won't be discussed, since we consider these irrelevant for the system since it is very arbitrary when the user will use the system and the fact that it will not be used to support work-tasks.

Regarding social context, we need to consider the fact that users are interacting with each other when buying and selling books. We need to find a solution for communication between the users, as well as privacy concerns. Regarding communication solutions, we will not make an integrated chat-system. Instead, we will let the users put in their contact information on the listing page. Also, the system will contain a minimum of personal data of the users to combat the privacy issues.

Technology

The designers of the IT-system has to take into consideration that it should be accessible from both computers and smartphones as both devices are viable methods of interacting with the system. Therefore, the system should be flexible and scalable to each device. This would also mean that the system has to be designed for both touchscreens and input devices such as a

keyboard, mouse, and trackpad. Furthermore, the system is going to be online based and open 24 hours a day. Therefore, it should always have access to the servers that also need to be reliable and consistent.

3.2.2 MoSCoW

Using the MoSCoW analysis is a way for us to prioritize and streamline what features are essential in the system's design process. It is used to identify which functions are essential for the system and which functions can be delayed for further development if time and resources are in hand.

#	MoSCoW	Requirements of functions	Fulfilled
1	Must	Be able to support the creation and edit of listings	
2	Must	Be able to support the creations and edit of users	
3	Must	Be able to show the contact information of sellers to users that are logged in	
4	Must	Be able to rate users and show this rating	
5	Must	Be able to search for listings and filter the results based on criteria	
6	Should	Be able to support users to save interesting listings	
7	Should	Be able to request a new password if forgotten	
8	Could	Be able to track recently visited listings and show them to the user	
9	Could	Be able to track the number of visits to a listings	
10	Won't	Be able to support chat between users	
11	Won't	Be able to promote specific listings	
12	Won't	Be able to support the sale of other items	

Figure 3.2. Table of system requirements with MoSCoW

3.2.3 Rich picture

It is beneficial to make a rich picture when creating a system. A rich picture is excellent for getting an overview of the situation for which a system is developed. A rich picture is a drawing made based on the illustrators' understanding of a given situation. A rich picture should focus on the most critical aspects of a given situation, while also being broad enough to allow for alternative interpretations of the situation. The illustrator decides the most important elements. When working with rich pictures, it is essential to obtain insight into and a feel for the most critical aspects of the situation. If, for example, a system is being created for an organization, it is important to understand the user culture and understand what matters. To do this, it is

crucial to visit the organization and see how it operates. It is also important to talk to a wide range of people about what's going on in the organization and perhaps conduct formal interviews. To understand a situation and familiarize yourself with it, it is important to involve all parties in making a rich picture and especially having future users of the system in mind. This does not mean that the users must draw a rich picture nor see it, but rather cooperate in making it by being involved in interviews. The rich picture is merely for the developers' understanding of the situation. Although, with that being said, the rich picture can function as a great way of communicating your understanding of the situation to users. We have made a rich picture of our understanding of the situation, and it looks as follows:

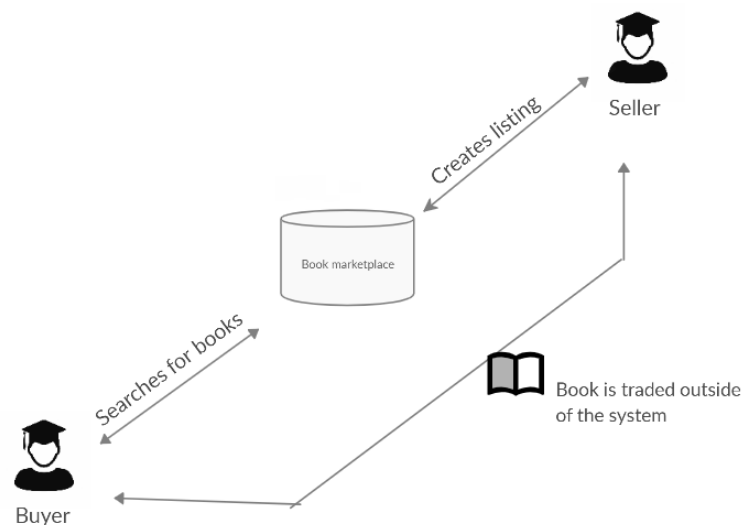


Figure 3.3. Rich picture

For our system, our situation is the selling of books as well as buying. Our system primarily needs to facilitate the instantiation of the trade. This means that it does not support the sale of the book, but rather the initial communication between buyer and seller.

3.3 Problem statement

For this project the main focus points of the solution will be:

- The solution will be financially beneficial for both buyer and seller.
- The solution will strive to live up to the UN's 12th global goal. To ensure sustainable consumption and production patterns.
- The solution will provide sentiment for the legal purchase of books by providing a viable alternative to the illegal acquirement of books.

The problem can be summed up in the following problem statement:

How can we create an IT-system, which aims to help students buy and sell used books?

System Analysis and Design 4

4.1 FACTOR

In the early stages of any kind of software development or project, it can be hard to identify the most substantial challenges that lie ahead. One way of tackling these challenges in an efficient way is for the entire team to be on the same page. A system can be perceived from many different perspectives, and individuals can have differentiating ideas. Discussions of different definitions and purposes can sometimes spark great debates and contrary opinions. Agreeing on a single, thoroughly discussed, system definition is one way of ensuring that every participant is in line with the others. A system definition gives an overview of the entire system, expressed in natural language. It is brief and concise, containing only the fundamentals. The definition and section below are created on the foundation of the FACTOR principles, with each letter referring to a key element in object-oriented software development. It consists of the following six elements:

- **Functionality:** The system functions that support the application-domain tasks
- **Application domain:** Those parts of an organization that administrates, monitor, or control a problem-domain.
- **Conditions:** The conditions under which the system will be developed and used.
- **Technology:** Both the technology used to develop the system and the technology on which the system will run.
- **Objects:** The main objects in the problem-domain
- **Responsibility:** The system's overall responsibility in relation to its context.[2]

In the following section the six elements in the FACTOR will be described in relevancy to this project.

Functionality: Register user profiles with relevant information and store them in a database. Registered users should be able to edit and delete their user profiles. Support for searching and filtering of books available in the system within specified search criteria, such as study and semester. Support for the creation of sales listings by users, allowing the user to edit and delete their listings. Support the finding of available books and the contact information of the user selling the book. Register a user rating and show each user's average rating score.

Application-domain: The system will be used by students who need to buy or sell books for a specific course and semester related to a particular field of study. A group of administrators will maintain the administration of the system with ownership of the system.

Conditions: The system must enable users to sell used books to other users at a price they deem appropriate. This will ensure the motivation among users to utilize the system as there will be a financial benefit for both parties involved, which is enabled by the system. The system

must also rely on the morale and integrity of the users to not create listings intended to fraud other users. The system must be easy to use, as users will have different levels of IT expertise and understanding. Finally, the system must exercise an up-time of at least 99.9%.

Technology: The system will be web-based and will require a connection to the internet and a device that can run a web browser. The system will be created using a combination of HTML5 and CSS for the user interface, C# for the system functionality, and MySQL for storage of data in the database.

Objects: The system is relatively simple in structure and design, so it is only possible to identify three objects: user, book, and listing.

Responsibility : The system is responsible for providing the user with a digital platform that can collect the relevant listing information and make the listing searchable based on that information. The system is also responsible for facilitating the search for specific books based on the information input from the user and show listings that match this information input. Finally, the system will be responsible for displaying the relevant information about the listings and the seller so that the buyer can verify the book and contact the seller.

4.1.1 System Definition

The system will be a digital platform that serves as an online marketplace for the sale and purchase of used books. The users will be able to sign up on the platform and provide their profiles with personal and contact information. The users will then be able to create, edit, and delete their own sales listings for books, that contain relevant information. Unregistered users will be able to search for listings based on predefined search criteria such as title, author, ISBN etc. However, to access the seller information they must be a registered user in the system. The listing and user data will be stored in a database. The system will be accessible through a web browser on devices with access to the internet. The system will be easy to use and thus allowing users with limited IT experience to utilize the solution.

4.2 Problem-domain Analysis

This project analyses the problem-domain first, followed by the application-domain. This was done because of the understanding of the business concept outweighed that of the functions and interfaces. This also allowed the object-oriented analysis and design part of this project to follow the course chronologically.

The purpose of the problem-domain analysis is to lay out what the system handles. Technical difficulties are not given any thought, as the focus is to make a model that represents how future users understand the system. This model acts as the foundation for the design and implementation of the system at a later stage of development. Every step of this analysis is done systematically and iteratively.

The offset for the problem-domain analysis is the system definition. The three following activities make up the problem-domain modeling: Classes, structures, and behavior. The object-oriented approach is, as the name suggests, based around objects. Objects are described as *"An entity with identity, state, and behavior."* [2] A collection of objects that share the same behavior and structure are made into classes. We classify certain phenomena to understand the tasks at hand.

These objects, which are entities, can be tangible items such as people but also less tangible items such as resources. We can then connect these objects to the events in which they are present. These events are abstractions of processes that include certain objects. Events can be thought of theoretically and might be a more natural way of identifying objects.

To produce excellent problem-domain analysis, collaboration with users of the target group is essential. In this project, where the target group is university students, the group members can act as prospective users, which makes finding classes easier. By writing down all possible class and event candidates, no possibilities are left unturned. As this process is highly iterative, there will be added and removed elements. In the end, this activity results in an event table, which shows the events vertically and the classes horizontally. A '+' mark shows the event happening just once in connection with a class, while a '*' mark allows that event to repeat indefinitely. Below is the event table for the system that this project handles.

Event \ Classes	User	Book	Listing
Listing updated	*		+
Listing created	*		+
Listing deleted	*		+
User created	+		
User updated	*		
User deleted	+		
Book created		+	
Book updated		*	
Book deleted		+	
User rated	*		

Figure 4.1. Event Table

The main function of the system is to enable users to sell and buy books. However, the system does not keep a record of the physical books. Instead, these are classed as listings, which are intangible sales listings of the books. The events related to this class are, therefore, the creation, updating, and deletion of said listings. The system also keeps track of users, and they are, hence a class in the system. The user is related to all events, including the creation, updating, and deletion of said user. Also, the users can rate other users with whom they have exchanged books.

The search criteria class relates to elements such as the university, semester, and study, which helps sort the books into relevant lists. Whenever the criteria are updated, created, or deleted, they affect the class search criteria. The events are instantaneous and atomic and can easily be identified, which are the most important criteria for evaluating the events in the event table [2].

4.2.1 Class Diagram

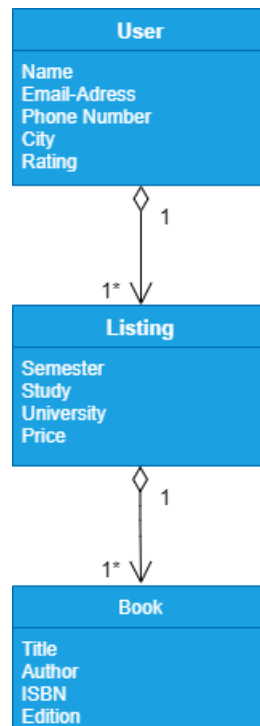


Figure 4.2. Class Diagram

The class diagram, such as the one above, uses structures to describe the relations between classes. In this case, the diagram contains three classes: User, Listing, and Book. The user, which includes attributes such as name and email address, are registered users in the system. These users have listings, and a listing is, in other words, the sales page, where the user can create their listings and list them for sale. These listings contain books, which is the final class. The 'book' class includes attributes such as title, author, and ISBN, which allows the user to sort listing searches by these attributes.

It is an extension of the event table, and the focus is on the interrelations between the classes. The structures are studied dynamically and statically by identifying generalizations, aggregations, associations, and clusters. Every class is held up against every other class, to see if there is a relation, and which kind of relationship there is between the two. The structures in the diagram are used correctly by using linguistic expressions to make sure the correct structures are used. The concepts are named to mimic that of a user's understanding of the future system. Although simplicity is a virtue, and something to strive for, the class diagram above is highly complex and simple at the same time. The entire system is based on those three classes, so while they are few, they are individually complex.

There is an aggregation structure between the class 'Listing' and 'User', which indicates that one is a defining part of the other. In this case, the listing is-owned-by the user. The 'User' class is therefore placed above the 'Listing' class. In this case, multiplicity shows that every listing has a user who owns it, but every user can have multiple listings, denoted with the 1* (1 to many) number. The 'Book' class has an aggregation structure to the listing, meaning that it is a part of a listing. The aggregation implies a strong link.

4.2.2 Behavioural Pattern

The final activity in the problem-domain modeling is the behavior of the classes. A behavioral pattern shows all the possible event traces of that class. This is done to model the dynamic reality that systems often find themselves in. This is also necessary since most problem-domain models demand some kind of event order. The classes from the class diagram are expressed with a statechart diagram that shows all possible event traces. This is done by studying every single possibility from the initiation to the end of a specific task.

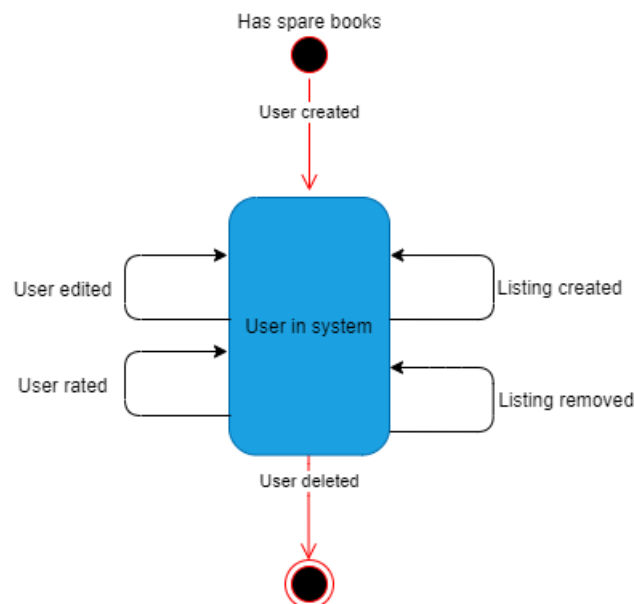


Figure 4.3. Behavioural Pattern for the 'User' class

For the "User" class, which is graphically expressed above, the start point is having spare books and, therefore, deciding to create a user on the platform. This creates a new state, where the user is now in the system. From here, the control structure is completely iterative. The user can create listings, buy other's listings, remove listings, and rate other users. These events can happen an indefinite amount of times before the user decides, for one reason or the other, their use is to be deleted. In that case, there are no more possible event traces for the "User" class.



Figure 4.4. Behavioural Pattern for the 'Listing' class

For the "Listing" Class, which is dictated primarily by sequence, with an iteration structure between the states of negotiation. The other event traces, and states are sequential, which means that one happens after the other is completed. Here the pattern starts with the creation of a listing, which places the book on the marketplace. Here the listing is subject to negotiations until a deal is agreed upon, in which case the details of the exchange are agreed upon. After the deal has gone through, the listing is removed from the marketplace.

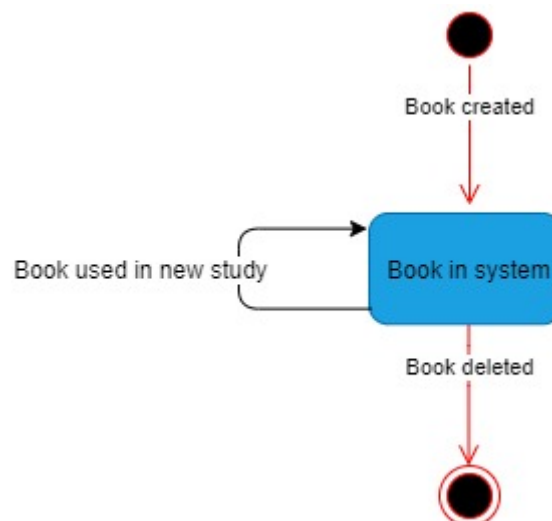


Figure 4.5. Behavioural Pattern for the 'Book' class

The pattern for the 'Book' Class is similar to that of the book, as it is sequential with a touch of iterative while the book is in the system. Each book is created in the system with the attached attributes. While it is in the system, it is subject to change in the form that there might be a new edition of the book, or the book is used in a new study that is created. The book stays in the system until it is deleted, which would happen when it is no longer being used by any study at the university.

4.3 Application Domain

While the problem-domain analysis primarily focused on what information the system should handle, the application-domain refers to the usage of the system. The purpose is to define

requirements for the functions and interfaces within the system. The problem-domain analysis was done first to understand the business concept of the project properly and to get everyone on the same page. Furthermore, it allowed the analysis to follow the course chronologically. The two events can be swapped if the focus is on the user's work. The best way to perform an application-domain analysis is to include users. This is because a lot of the application-domain analysis is based on use cases. In this project, which revolves around student users, the use cases are based on the students within the study as potential users. The application-domain analysis activities are also highly iterative, switching between use cases, interfaces, and functions. To establish order, the activities are done in turn, starting with the use case diagram.

4.3.1 Usage

Use case\ Actor	User	Marketplace manager	Guest
Listing created	+		
Listing removed	+		
Listing information	+	+	+
Search for listing	+		+
User information	+		+
Review	+		
Account access	+	+	
Register user	+	+	+

Figure 4.6. Actor Table

The use case diagram shows all eventual use cases, and the relation to the actors that perform these use cases. It could also be other systems. In this case, however, the system stands alone. Every use case is a small part of the system, and each can be initiated by an actor. This process is highly analytical while being creative. Every possibility must be evaluated systematically. The actor table is preferred over the statechart diagram, which shows the same information but in a more graphical way. This is due to how much time and space they each take up.

Description of each use case:

- Listing created: Every time a user creates a listing, it is saved in the database.
- Listing removed: After a book is sold or there is no sign of interest from other users, the listing is removed from the system and deleted in the database.

- Listing information: There are several attributes or information about every listing that the user fills out when creating the listing. This helps other users find the listing.
- Search for listing: A listing can be searched for either by information about a certain book, such as author and title, but also by a certain study at the university.
- User information: Information about other users can be found when there is a need to contact another user about a certain listing.
- Review: Users can review each other when a deal is finished and rank their experience with the other user.
- Account access: When logging in or out of an account to use some of the functionality that the platform has.
- Register user: When guests find listings that are of interest, they can sign up to gain the contact information from other users to negotiate a deal.

In the table above, there are three actors listed: User, guest, and marketplace manager. These are the primary users that can initiate one or more use cases. The users are the primary actor, around whom the listing functions origin, which is the central element of the system. The marketplace manager is there to ensure that the system is being used as intended, as well as updating, creating, or removing certain search criteria. The guest actors can browse the listings, but are intended to register should they need the rest of the functionality. The actor table above serves as the primary description of the relevant parts of the application domain. Furthermore, it provides descriptions and an important step towards the requirement specification.

4.3.2 Functions

Check book information	Simple	Read
Check listing information	Simple	Read
Check user information	Simple	Read
Create listing - Multiple criteria - Picture	Complex	Update
Remove listing	Simple	Update
Calculate rating	Simple	Compute
Rate user	Medium	Update
Log in	Medium	Read
Log out	Simple	Read
Register as user - Insert into DB - Manager or not	Complex	Update
Search - List listings based on criteria from DB.	Complex	Read

Figure 4.7. Function List

The second activity is the functions, with the purpose of determining the system's information processing capabilities. Functions act as the model that actors use when using the system. There are different types of functions, and these have different complexity levels. The list, therefore, provides both complexity and function type. The complexity is based on a four-point scale, with simple, medium, complex, and very complex, respectively. The function list is mostly used as a part of the system requirement negotiation, where the complex levels act as the basis for negotiation. The function types are also categorized into: Update, Signal, Read and Compute functions.

The list above further develops on the actor table, describing the functions that are used in those use cases. The 'check' functions are all 'read' functions activated by a need for information in a work task, returning the relevant information.

Creating and removing listings are updated functions, which means they are activated by an event and change the state of the problem-domain model. These are created by asking critical questions, which provides a reasonable basis for identifying and evaluating functions. The complete list of the functions, along with the actor list, serves as the basis of the ambition level of the system. They are also in line with the system definition and the problem-domain model.

Description of functions

User:

- Search listing: A search is based on criteria. The criteria include semester, study, and university. The query is initiated, and it retrieves information from the database and shows relevant listings.
- View listings: The user can view the individual listings, which they receive from the search.
- Rate users: After a deal has gone through, the user can rate the user they have made a deal with.
- View user profiles: The user can browse the profiles of other users and see their listing history as well as their rating.
- Login/Logout: The user should be able to log in and out of the system.

Rating:

- Calculate mean rating: The system can calculate the mean rating based on all ratings divided by the number of ratings
- Store ratings: The system can store how many N of X ratings the user has, and it can display this information to the user.

Listing:

- Create listing: A listing can be created by users and defined by search criteria such as semester, study, and university. Each listing contains several books.
- Remove listing: The user can remove their listing, and thus able to remove it from the database.
- Edit listing: The user can update their listing as they see fit.

Book:

- Filter by title: The listing search can be filtered by title.
- Filter by author: The listing search can be filtered by the author.
- Filter by ISBN: The listing search can be filtered by ISBN.

4.4 Interface

4.4.1 Sketches and wireframes

As a part of the design process, we have created sketches and wireframes to represent the visual characteristics of the systems user interface. This is done to complement the object-oriented design, assist us in planning, and visualize the final product of the implementation process. This

method of creating sketches followed by wireframes enables us to be creative in our design choices while ensuring that the requirements for the system are still fulfilled.

We started the process by creating rough sketches because it allows us to envision our design of the user interface without constraints. We are free to explore different design options and styles before moving on to the creation of wireframes, which serve as a more realistic and robust design plan for the user interface. Figure XX shows one of our sketches that depict the page where the user can create a new listing. The inspiration for the sketch does not come from one source alone, as it is a combination of our experiences with different types of website design and particularly websites that also serves an online marketplace. (Designing Interactive Systems, David. R. Benyon)

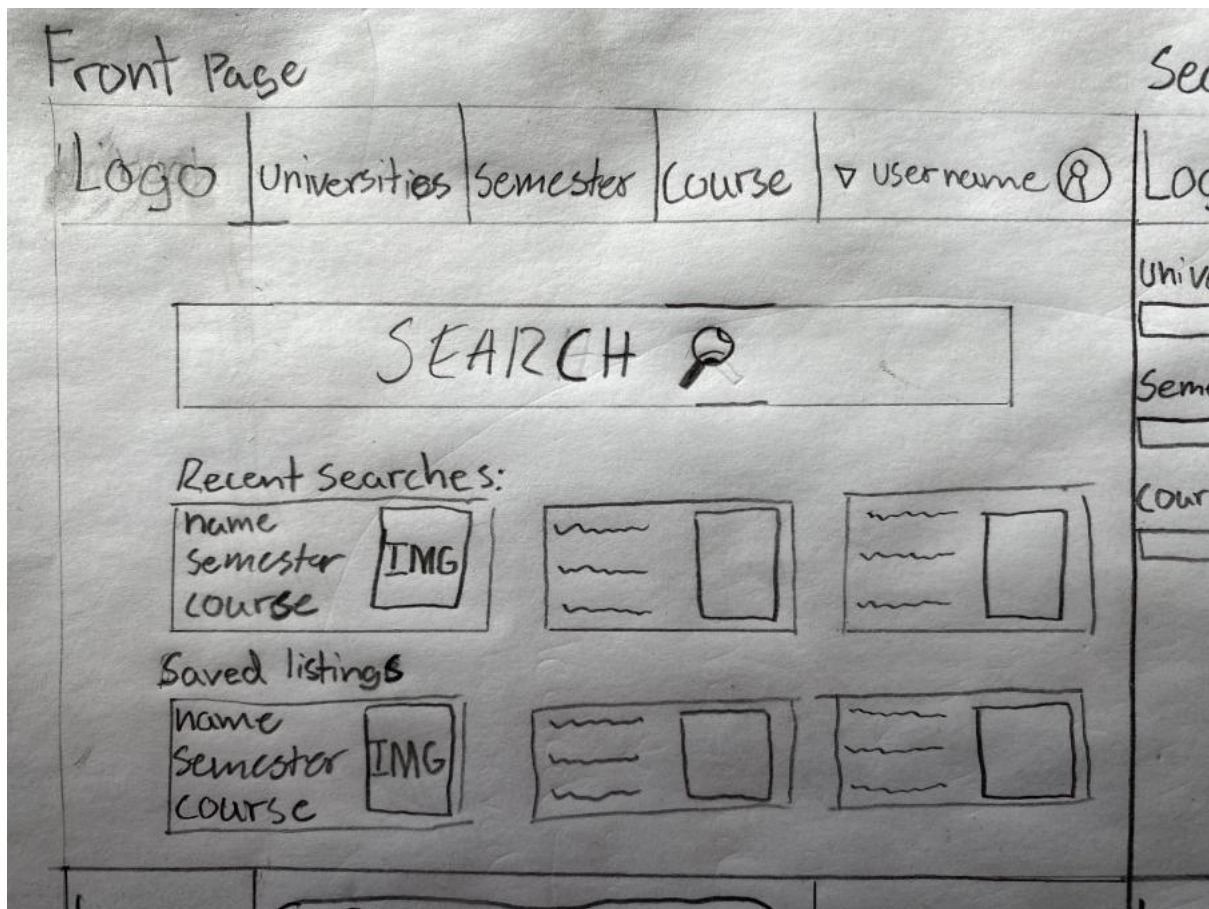


Figure 4.8. Sketch of front page

After multiple iterations of sketches had been created and evaluated, we decided on moving forward with simple and common design to increase the usability of the system. From these sketches, we have created the following set of wireframes that implement more realistic elements from general web design and can be used to design the layout of the user interface accurately.

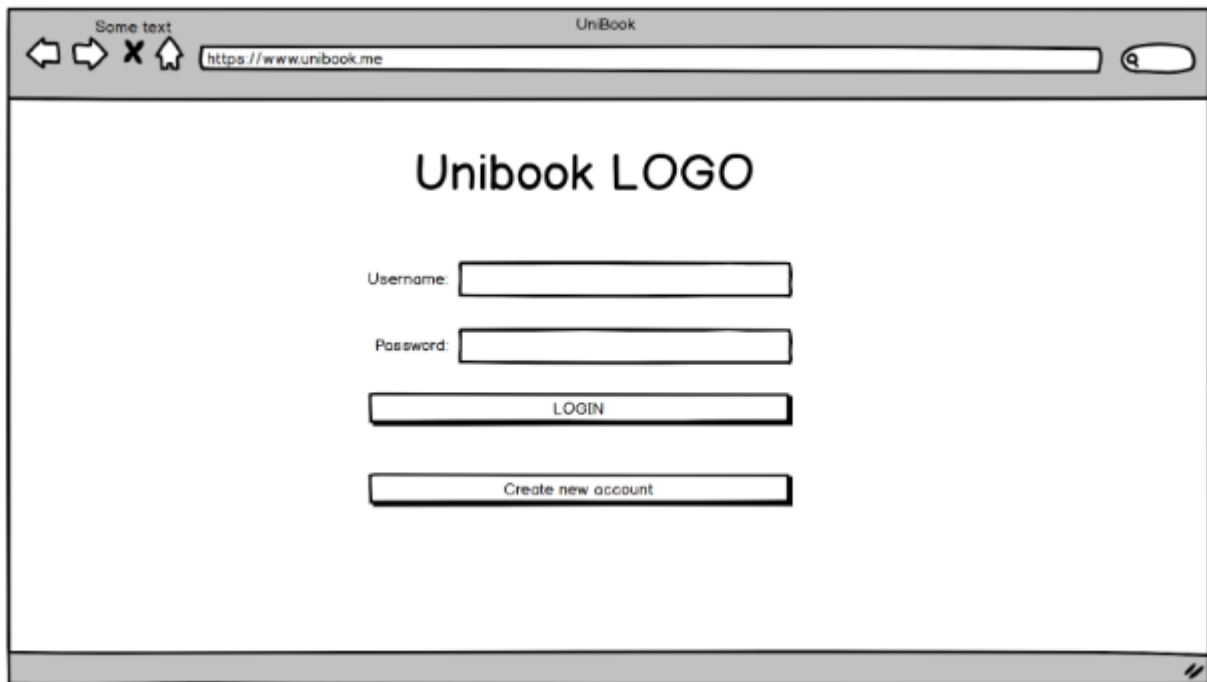


Figure 4.9. Wireframe 1

Wireframe 1 shows the login page where users can log in to the website and create a new user account. The users will have to be logged in to create listings, see the contact information of other users, and rate the users.

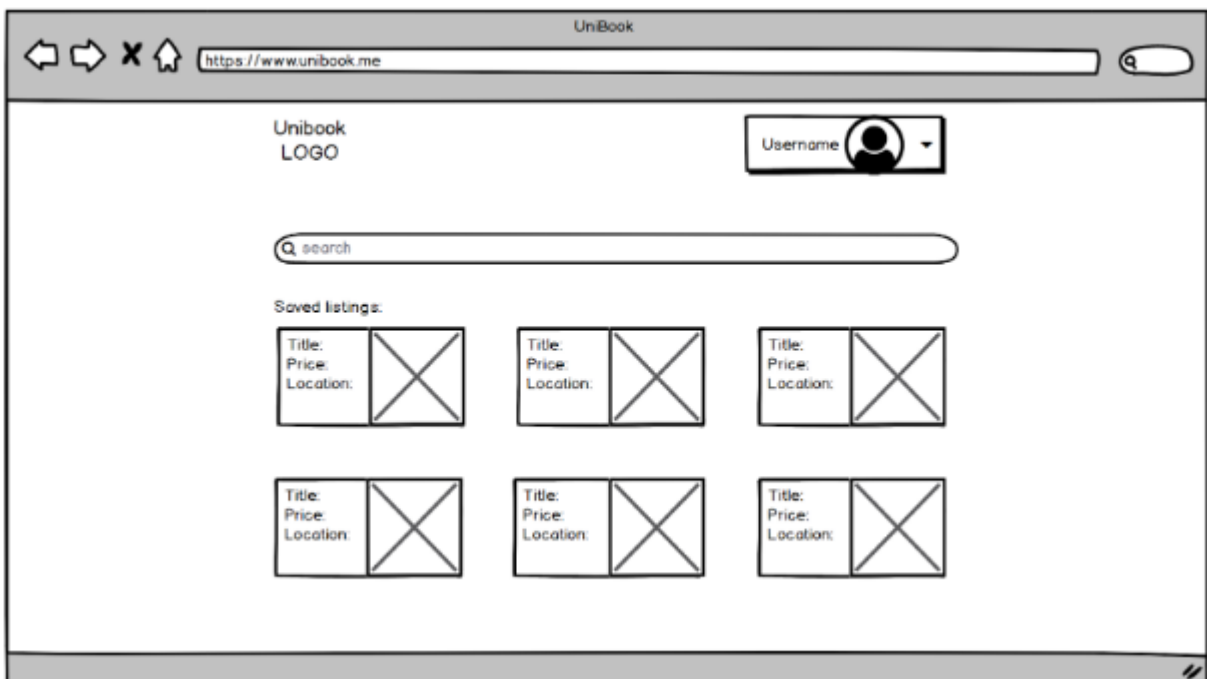


Figure 4.10. Wireframe 2

Wireframe 2 shows the front page of the website, which also serves as the landing page for the URL www.unibook.me. From the landing page, the users can search for listings, see their saved

listings if they are logged in. In the top left corner of the page, there is a website-logo. The logo is visible on all the pages of the website. The logo also serves as a link back to the front page. In the top right corner, there is a drop-down menu that serves as the main menu and will display the username and profile picture if the user is logged in. The drop-down menu is also visible on all pages of the web site except the login page. In the drop-down menu, users can log in and log out of the system, go to their profile page and create a new listing.

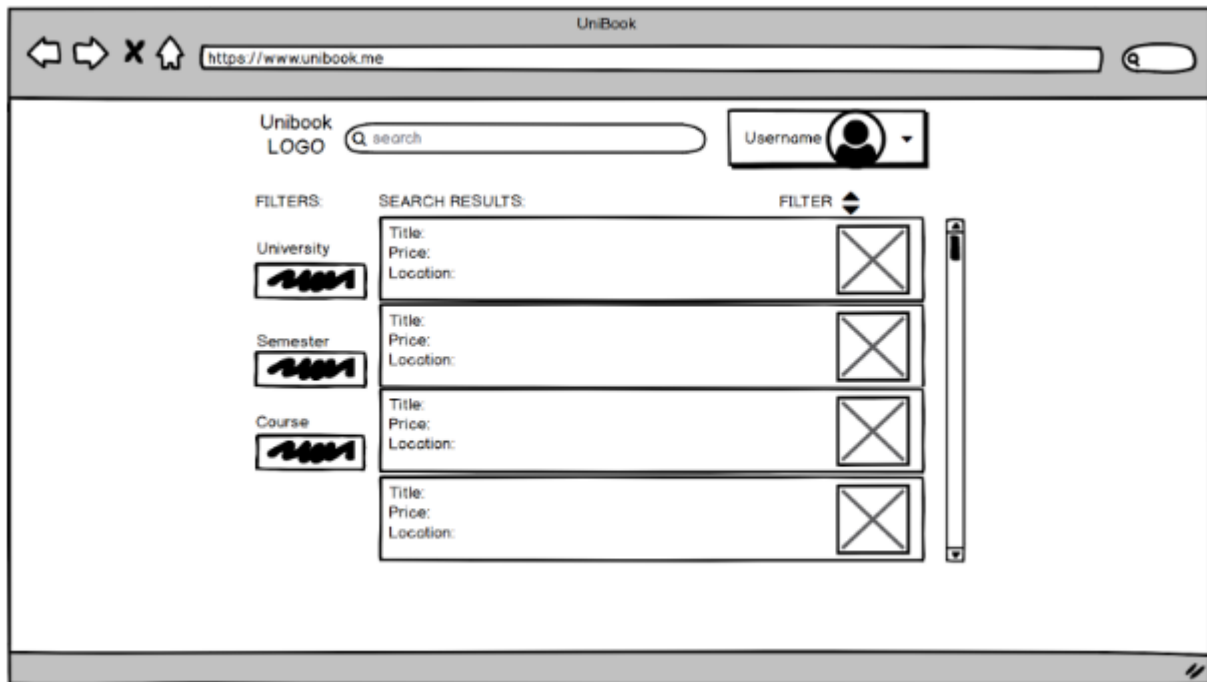


Figure 4.11. Wireframe 3

Wireframe 3 shows the page where search results would appear based on what has been searched for in the search field. The search field is moved to the top of the page compared to its position on the front page. This is done to maximize the amount of space for displaying the search results, and the search field has this position on all other pages with content to display other than the front and login page. Each element in the list of search results is a link that leads to the page of that specific listing. By applying filters to the search results, it is possible to alter the list.

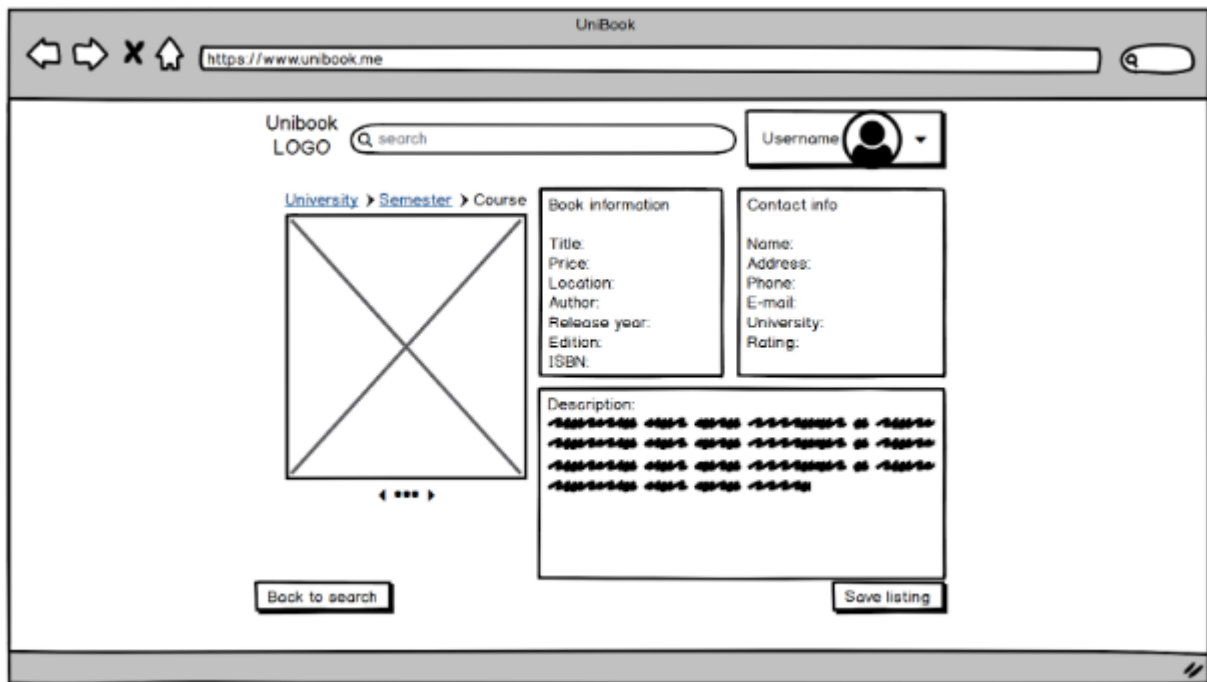


Figure 4.12. Wireframe 4

Wireframe 4 is the page showing a listing which contains images and information about the book for sale, as well as the seller. If the user is logged in, the seller's contact information is visible, and it is possible to access the seller profile page. If logged in, it is also possible to save the listing to the user's list of saved listings visible from the front and profile page.

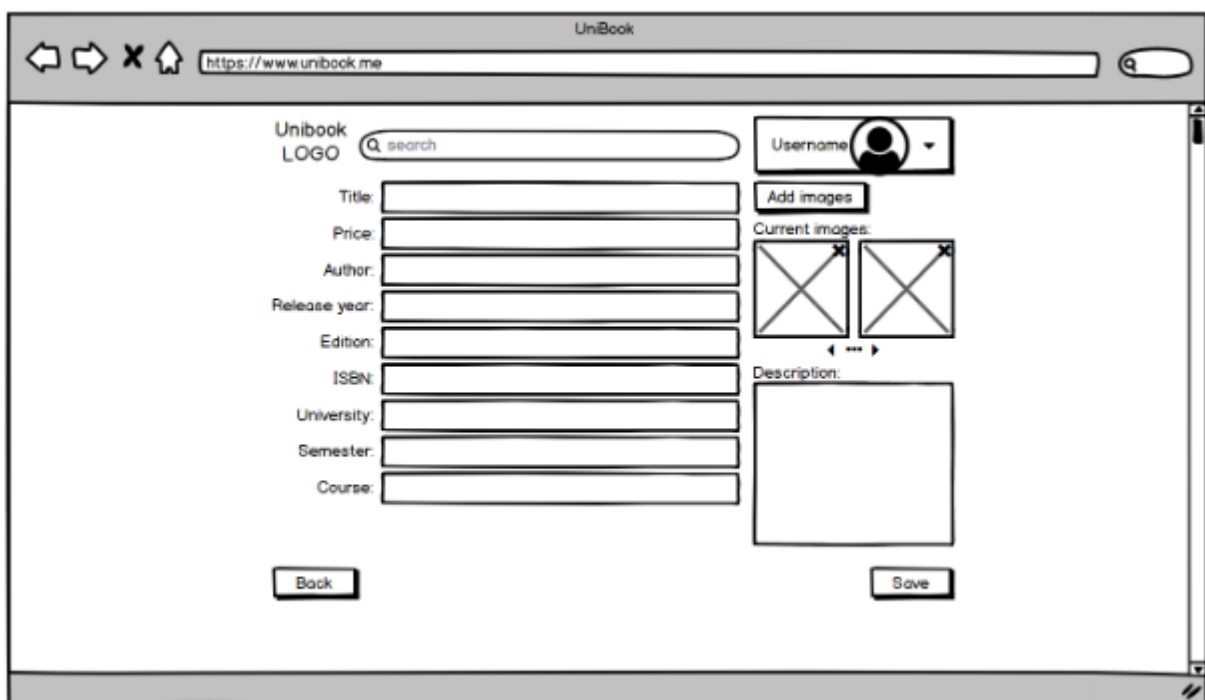


Figure 4.13. Wireframe 5

Wireframe 5 is the page that serves as both the edit- and create-page for a listing. Here, the user can add information about the book. When the listing is done, the user can click save, and the listing will be published. The user can then edit the listing later from the user profile page.

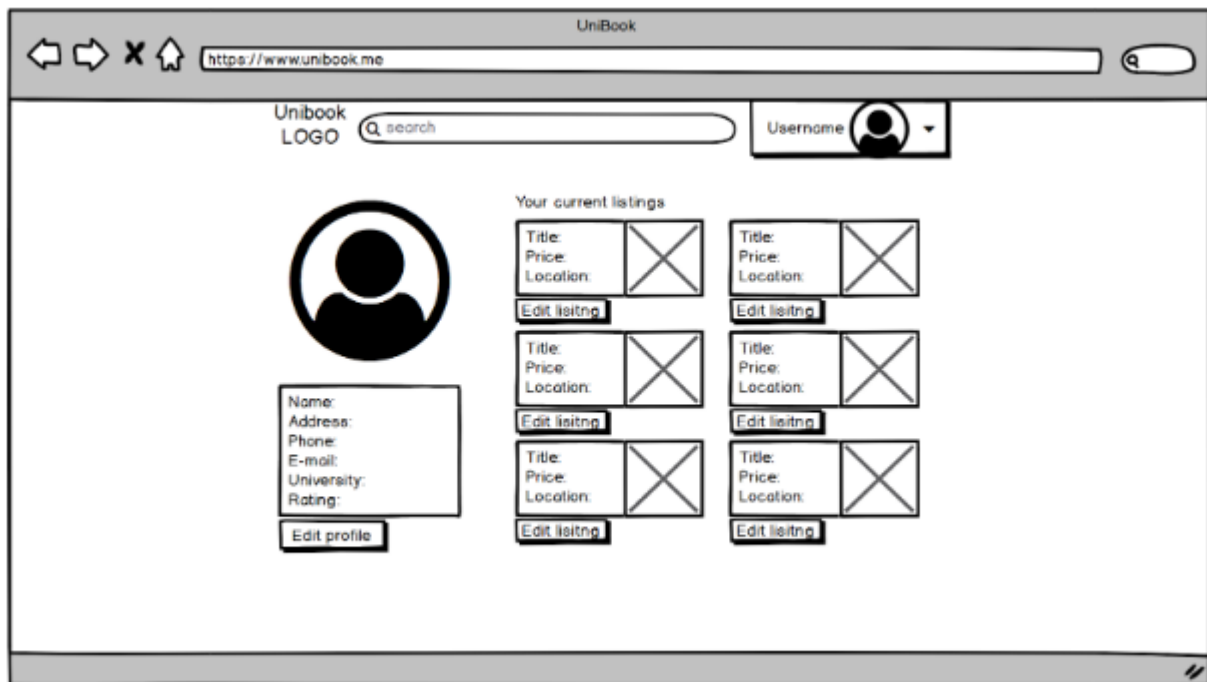


Figure 4.14. Wireframe 6

Wireframe 6 is how the user profile page will look when the user accesses their profile. The user will be able to see the contact information that is being shown and a list of the user's currently active listings. The user can edit the contact information, a button that takes the user to the editing user profile page. Also, under each listing, there is an 'edit listing' button, which takes the user to the edit listing page for that specific listing.

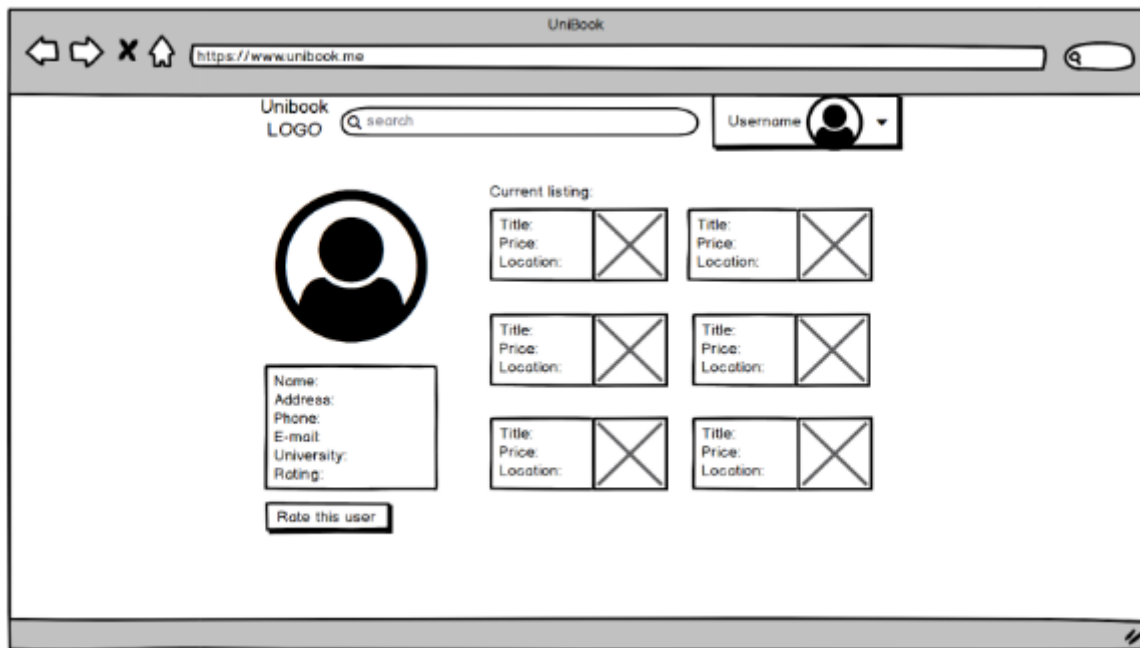


Figure 4.15. Wireframe 7

Wireframe 7 shows the profile page of a user, as other users see it. This will only be possible to get to if the user is logged in. On this page, the users will be able to rate each other, as long as a deal has gone through, as well as seeing the listings that the other user currently owns.

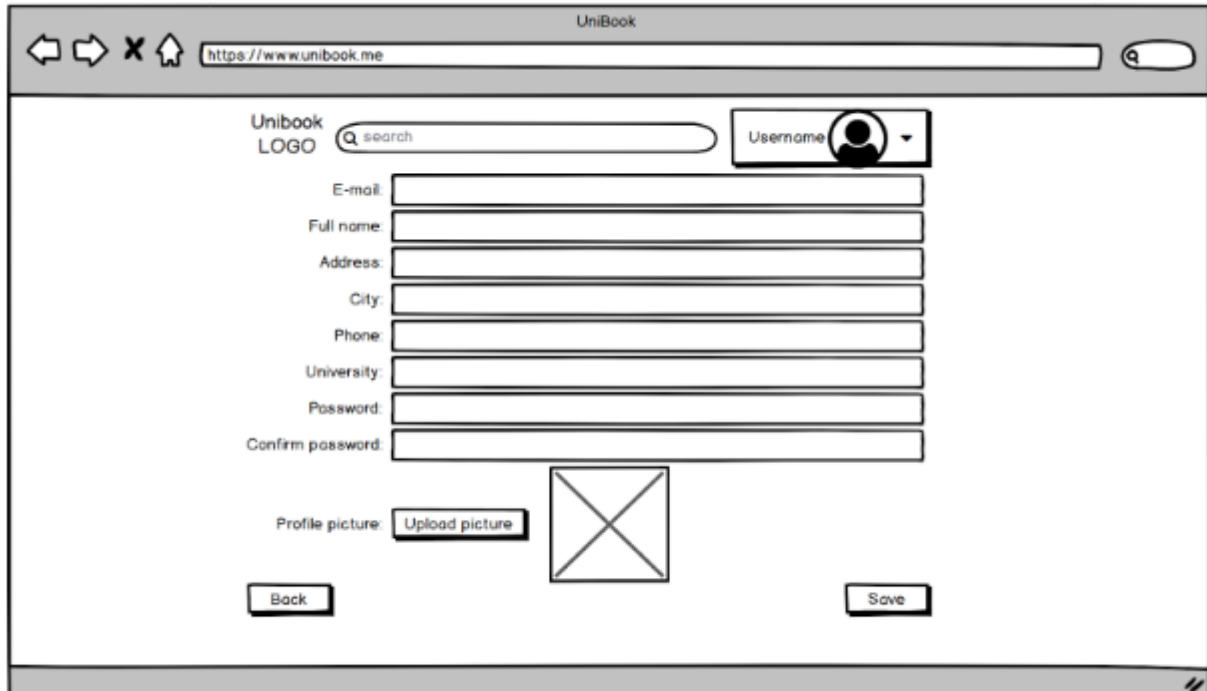


Figure 4.16. Wireframe 8

Wireframe 8 is the page that serves as both the create- and edit-page for the user profile. It is accessed when a user clicks the 'create new account button' on the front page and when the

user clicks the 'edit profile' button. The user can edit their contact information, password, and profile picture on this page.

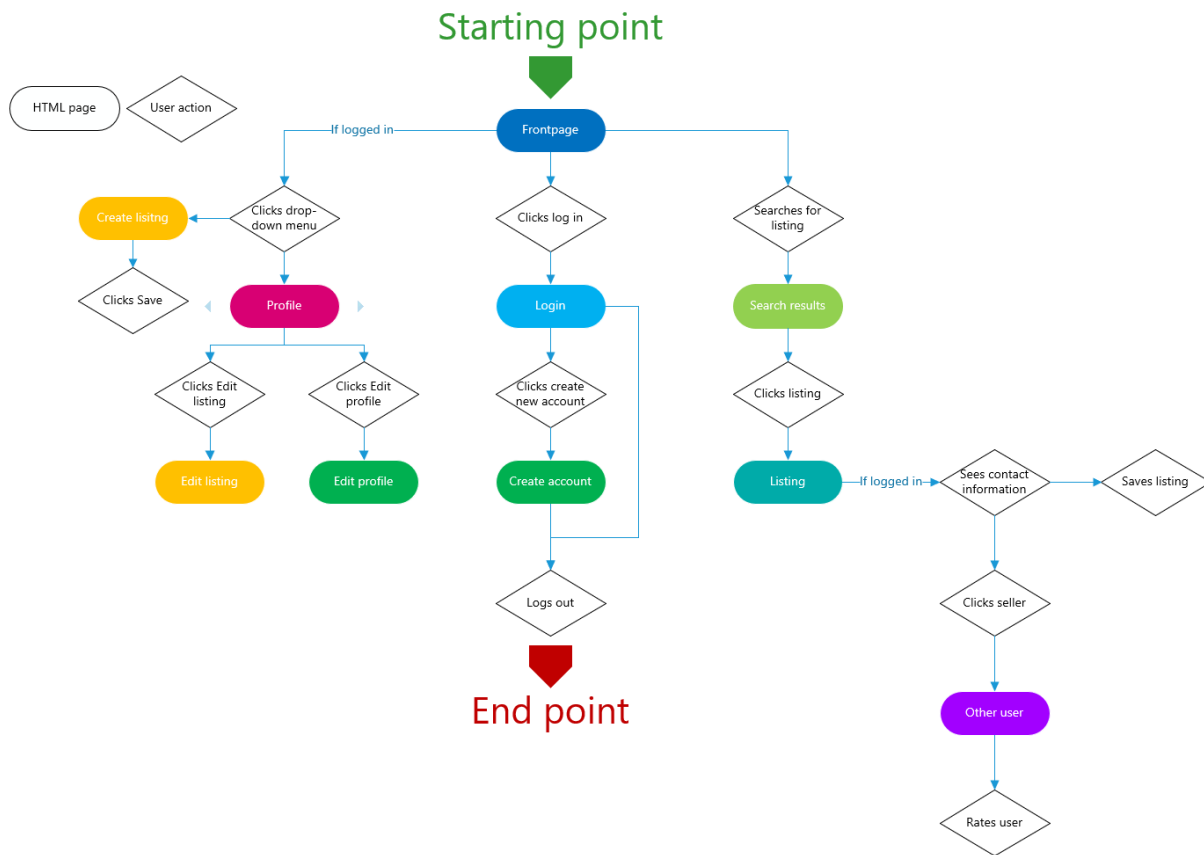


Figure 4.17. Navigation map of the site

The navigation map shows the navigation structure of the websites and how the pages are connected to each other. The starting point for most users will be the front page from where they can access the rest of the website. The navigation map also includes visible representations of the parts of the website that are restricted to users that are not logged in. The end point is when the users log out of the system as. Both the starting and end point of the website is theoretical as user will be able to leave the website at any point and also navigate through the site completely at their own will. Therefore, this navigation serves a primarily practical purpose as it helps us visualize the website structure and how different files of our system should be connected.

4.5 Architectural design

4.5.1 Criteria

The system is primarily evaluated based on its ability to fulfill the requirements established as a result of the analysis. However, the system will also be assessed on the quality of the design, and quality is achieved through the absence of errors and flaws in the system. There are many different properties of the design that can have different levels of quality, and these properties are described as criteria. The excellent design needs to focus on the criteria that are important,

the type of system that is being designed, and a good design will strive for a balance between focusing on the right criteria and limiting the amount of work necessary to achieve the desired quality. Therefore it is an integral part of the design process to prioritize each criterion based on its importance for the quality of the system.

The criteria that have been identified as a part of the OOA&D method are:

- Usable: The system's ability to adapt to the organizational, work-related, and technical contexts
- Secure: The precautions against unauthorized access to data and facilities
- Efficient: The economic exploitation of the technical platform's facilities
- Correct: The fulfillment of requirements
- Reliable: The fulfillment of required precision in function execution
- Maintainable: The cost of locating and fixing the system defects
- Testable: The cost of ensuring that the deployed system performs its intended function
- Flexible: The cost of modifying the deployed system
- Comprehensible: The effort needed to obtain a coherent understanding of the system
- Reusable: The potential for using system parts in other related systems
- Portable: The cost of moving the system to another technical platform
- Interoperable: The cost of coupling the system to other systems

Of these criteria, there are three that are considered general criteria that have universal validity for almost any system's design. These are usable, flexible, and comprehensible as they all relate to the use of the system, which is the existential basis for most systems.

In order to properly prioritize these criteria, it is essential also to consider the conditions that exist within the context of the design. In terms of technical conditions, there are very few elements that need to be considered. This project will not use any cutting edge technology or advanced software and will use already established ideas from other online marketplaces. Likewise, there exist few organizational conditions since this project does not have any partners to consider. It is, however, vital that we keep the future development options of the system in mind and be sure to consider the division of labor between the members of the project team. Finally, the human conditions are especially relevant for this project as the design competencies and overall experience of the project team are at a novice level. The consideration of these conditions will help us be more effective and realistic in prioritizing the criteria.

In the following table, the different criteria are prioritized as either very important, important, less important, or irrelevant. Furthermore, if a criterion is evaluated as not requiring as much work as another criterion during the design and implementation phase, it is prioritized as easily fulfilled.

Criterion	Very important	Important	Less important	Irrelevant	Easily fulfilled
Usable	X				
Secure			X		
Efficient		X			
Correct		X			
Reliable		X			
Maintainable			X		
Testable					X
Flexible		X			
Comprehensible		X			
Reusable				X	
Portable				X	
Interoperable				X	
Additional criteria					

Figure 4.18. Criterion

4.5.2 Components

Components make up the basis of the component architecture. The idea of components is to make "A collection of program parts that constitutes a whole and has well-defined responsibilities." (OOAD Page 191). These components are connected to make up the system structure, also called the component architecture. These components should have very well described responsibilities, as well as little to no overlapping. This is done to reduce complexity and increase flexibility. One way of separating the different components is to layer them. A layered architectural pattern is often used in software development. Each layer is then composed of various components. The generic architecture pattern, which is used in this section, is also a layered pattern. The component diagram is often denoted as a class diagram, within the UML paradigm. The arrows show dependencies, which means that one element can change certain parts of another element. Very similar to the class diagram described previously in the paper. In the diagram below, the «» notations are the singular components, while the 'interface' and 'technical platform' are layers that contain more than one component. It is separated this way into the user interface and the system interface. These are at the bottom, with the model in the middle and the interface at the top.

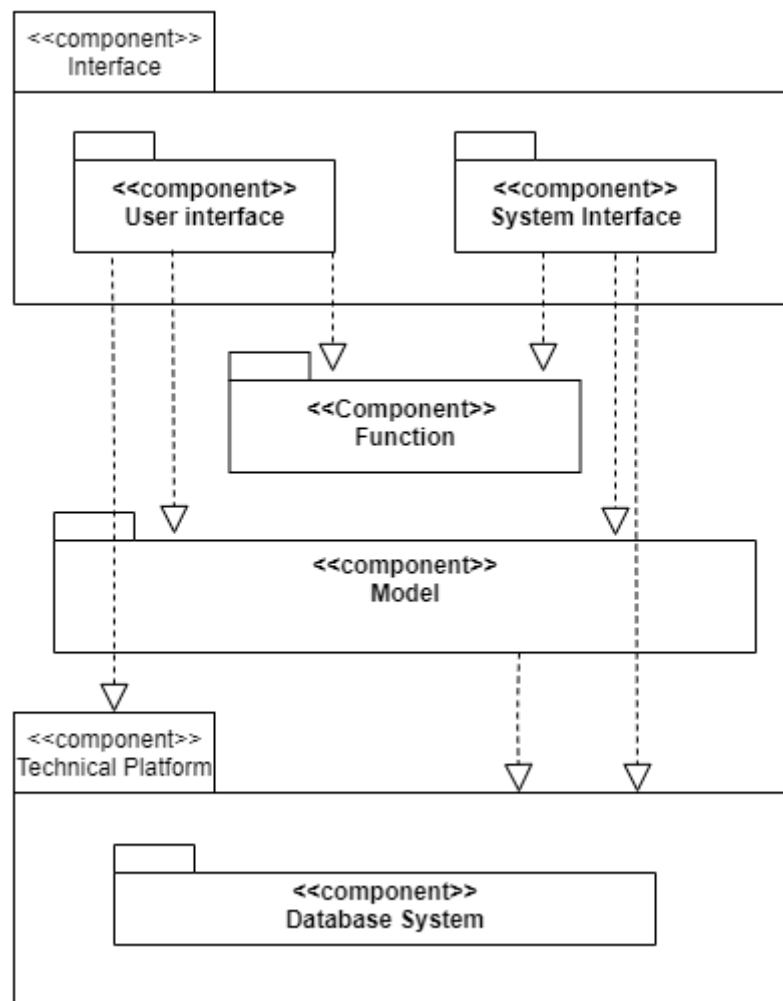


Figure 4.19. Component architecture diagram

The responsibility of the interface is to show and display relevant information about the listings and books in the system to the user. Both of these components are encapsulated in the interface layer, which means that the two components in this layer have been decomposed as they are not the same. This also allows for some flexibility. The primary objective of the interface components is to handle the interaction between the actors and the functions. The function component is the primary factor of the functionality. In this case, the functionality is rather simple, and there is no need to decompose this component in the given context. Placed above the model component, the function component adds functionality to the model, which is then displayed in the interface layer above. The technical platform at the bottom is made up of libraries that the other components used to carry out their responsibilities in the system context.

4.6 Component Design

The component design follows the component architecture diagram that was made in the previous chapter. This chapter focuses on the model and function components. With the architecture in mind, the purpose is to implement requirements. The design of the components must be adapted to the technical possibilities. The model component section addresses the implementation of the

problem-domain model into the OOP (Object Oriented Platform). The functions component implements the functional requirements by organizing and interrelating the components. (OOAD Page 233)

4.6.1 Interface component

Starting from the top of the component architecture diagram, the interface component, whose primary purpose is to display relevant information to the user, is at the top.

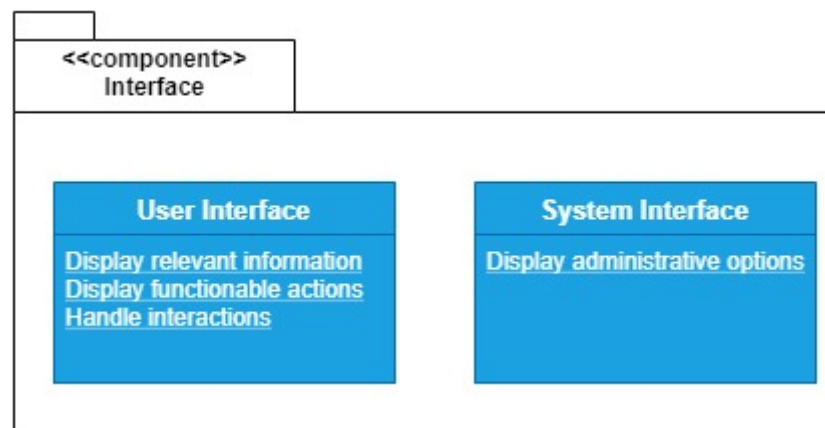


Figure 4.20. Interface component

The Interface component handles the displaying of the information from the model and database while sending signals to the functionality and the database when interacted with by the user. In other words, the interface component does not have any standalone functions or the ability to have such functions. It relies on the database, model, and functions for its displays, but it also goes the other way. Without the interface, the database would not know which data to display, as this is based on the user input from the interaction with the interface. (OOAD Page 206) The distinction between the User Interface and the System Interface is made to separate the two types of users. Regular users should see administrative options, as this would diminish the actual use of the administrators. The system does not have an emphasis on the interface component but recognizes the importance of a well-designed interface component, which is explained in a previous chapter.

4.6.2 Database component

The primary purpose of the database is to store relevant information about the listings, users, and books. The database is a library of information that can be accessed by other components. The interface component can display data from the database, while the function component can insert data into the database or update it. In other words, the database keeps track of all observable changes in the system.

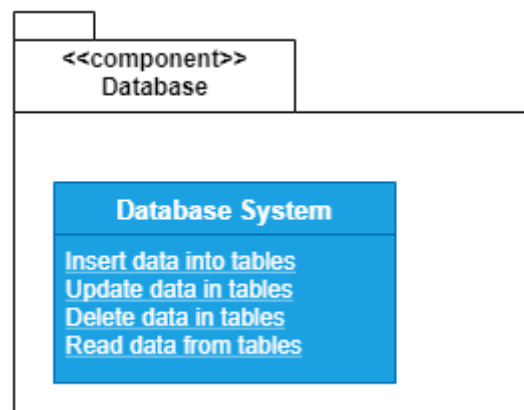


Figure 4.21. Database component

Figure XX - Database component The database component is at the bottom of the layered architecture diagram, but it is far from the least important component. The database provides all other components with needed information for them to work as intended. Therefore it is crucial that the database works as intended and can handle the changes that it is subject to. (OOAD Page 207)

4.6.3 Model Component

For the model component, the central concept is structure. The purpose is to deliver current and historical data to the functions as well as the interfaces, users, and other systems. This information is stored and is related to the system's problem-domain. The model should reflect the problem-domain's relevant conceptual relations. The foundation of this component is the object-oriented model from the analysis activity. The model component describes the problem-domain by using classes, objects, shared structure, and behavior. The model component implements the system requirements, which are stated in the problem-domain analysis. It is defined as "A part of a system that implements the problem-domain model." (OOAD, side 238). The result of the model-component activity is a revised version of the class diagram. By looking at our events and whether they are iterative, we can decide if a new class is necessary. We could see that a class for ratings was necessary.

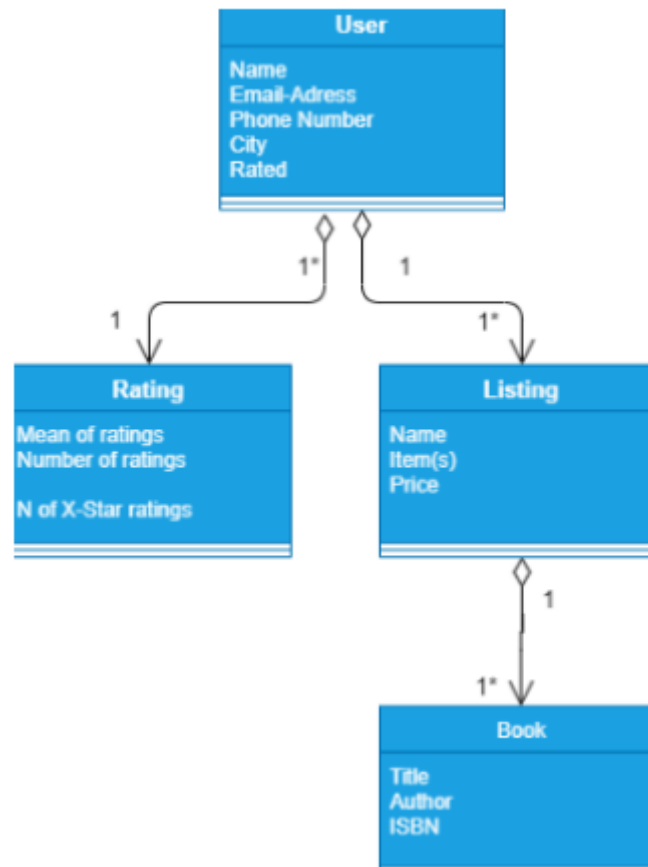


Figure 4.22. Revised class diagram for the model component

The revised class diagram features a new class for the user ratings. By revising the class diagram from the analysis chapter systematically, it now represents the objects' events with a new class. This is done in the most straightforward way of representation. When users rate each other after a deal has gone through, the rating of each user must be updated with a new mean. The rating should be able to contain more than one value, and therefore it now has its class. The rating class will contain all of the ratings that a user has received and calculate a mean rating based on X amount of ratings. This represents an event as a class with structure and attributes since it inherits the attributes of the ratings.

4.6.4 Function component

After making the model component, we need to create the function component. The purpose of the function component is to determine the implementation of functions. There are two concepts that we are working on within this step; Function component and operation. The definition of the function component is a part of a system that implements functional requirements. The definition of operation is: a process property specified in a class and activated through the class' objects. The purpose of the function component is to give the components of the system access to the model. This means that the functional component is the link between the model and the use. A function, in this context, is an externally observable behavior directly related to the users' work.

An operation is a description of behavior that is activated through an object. Operations can be understood as procedures. An operation is activated by a message, and it carries out what data should be processed and returns the calling point. When designing the function component and its operations, it is important not to program them but rather design them. It's too time-consuming to program them, and it's more important to focus on the design of them. It can also be too time-consuming to make the functions too detailed. The function component should thus give an overview of the function and how they are implemented, but not dwell on the programming of them. In design, we implement functions as operations on classes.

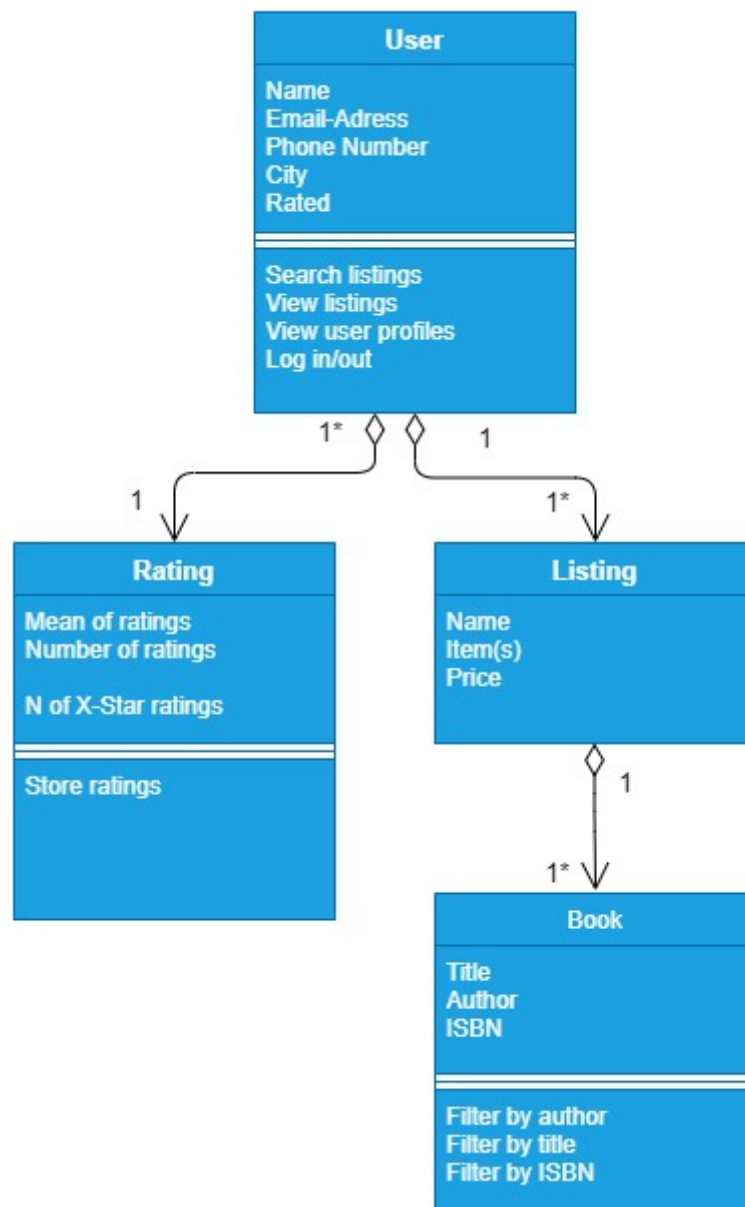


Figure 4.23. Revised class diagram with operations

By revising the class diagram and adding the operations of each class, designing and implementing the required functions is easier. Operations are: "Process properties specified in a class and

activated through the class' object" (OOAD Page 254). They describe the behaviors that can be activated or started through an object. The functions are activated on a class, instead of in a class. In this design, the functions are based on the function types mentioned in table XX from section XX. The class diagram with operations is based on the class diagram from figure XX in section XX.

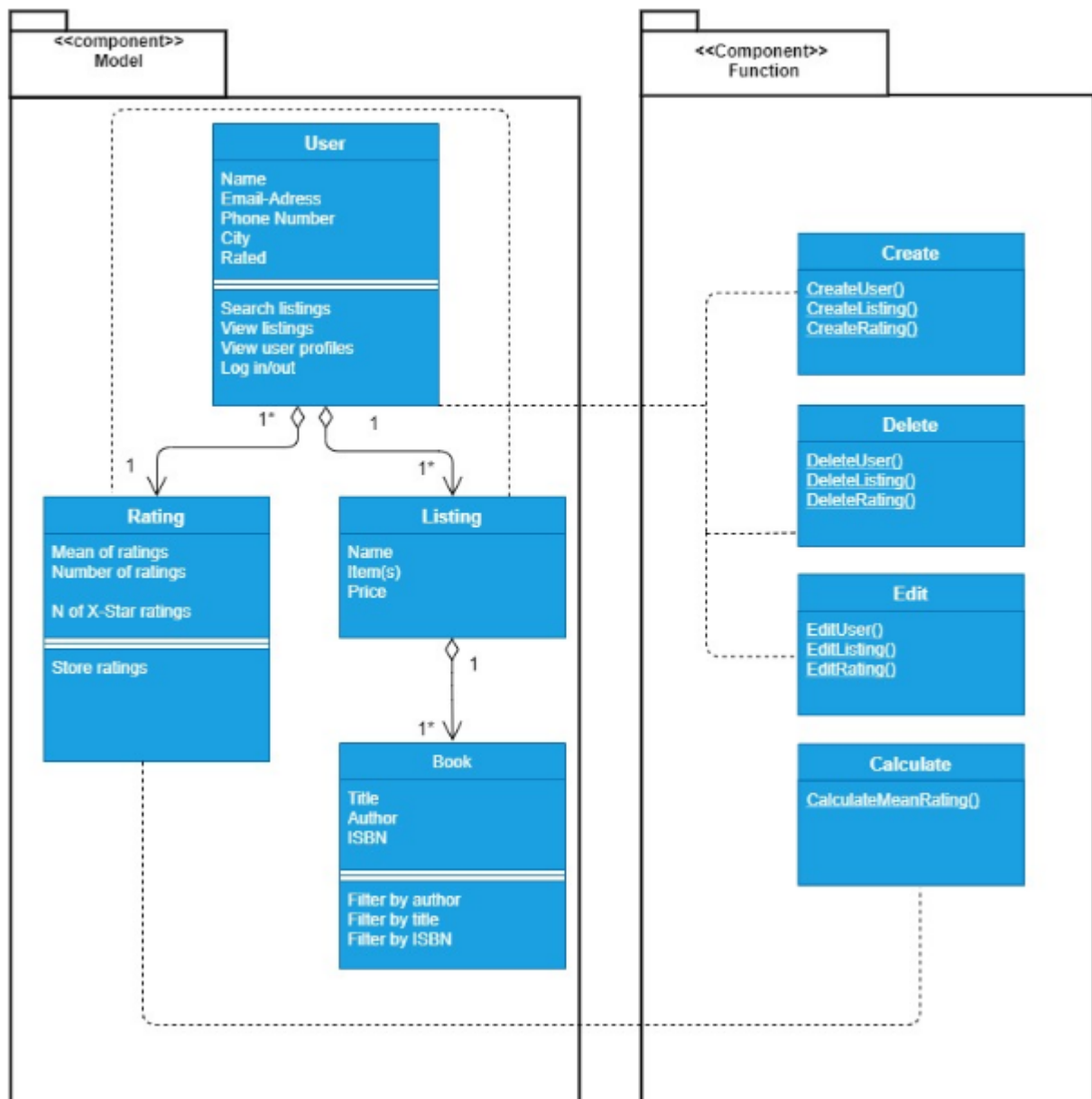


Figure 4.24. Revised class diagram with operations

Finally, the model component and the function component are connected. The functionality of the classes in the model component is added by those in the function component. Functions distinct themselves from operations in being externally observable. A user might search for listings and thereby gathering information from the database, but there is no change. When a new user is created, a row is inserted into the database, and the CreateUser() function is placed in the function component (OOAD Page 254). The create, delete, and edit functions

are very similar, as they impact the same classes in the model component. The last function is the `CalculateMeanRating()` function, which calculates the mean rating of a user and thereby updating it in the database.

Implementation 5

The implementation phase aims to create a working system based on the analysis and design. The system will be built using the object-oriented programming language C# as well as MySQL for the connection between the database and the system. In relation to the implementation of the database and system functions, we will be utilizing the ASP.NET framework to get access to useful libraries and the Entity framework for querying the data in our database. Furthermore, the website will be built using HTML, CSS, and JS as a part of the Bootstrap framework. In this part of the report, we will go into further detail regarding the choices made for using these tools, languages, frameworks, and the programming conventions that we have followed in general.

5.1 Languages

The frontend of our implementation will be responsible for creating the visual elements and structure of our website as well as linking the different pages together so that it is easy to navigate. It will also allow the user of the website to access the backend functionality of our platform so that the user can read and write data in the database.

5.1.1 HTML

HTML (HyperText Markup Language) is used on webpages to make it possible to present material on the web. Webpages translate the HTML into something that the user can see and interact with. HTML consists of tags that are used to format the HTML. An example is to create a title, you need to use the tag `<title> </title>`, and in between the two, you write the title of the page. With HTML5, there was a leap forward in web design, layout, and usability. It provided a more straightforward method to design graphics without using plugins such as Flash and offers ways to install audio and video into the webpage without plugins.

5.1.2 CSS

CSS (Cascading Style Sheets) is a crucial companion to HTML, ensuring that the HTML text and embedded images are laid out consistently and in a manner for the user's screen. It is used to organize the look of the homepage by changing the typography, size of the images and text, the colors of the design, and so forth. Using CSS can be simple as one would only need to insert a couple of code lines to change something on the web page e.g., R. Nixon uses the following code to demonstrate this:

Listing 5.1. CSS example

```
1 <style>
2                               p{
3     text-align: justify;
4     font-family: Helvetica;
5 }
6 </style>
```

These lines of code affect the 'p' tag, which represents the paragraphs. From now on, all paragraphs that are contained in the web page by default are justified and use the Helvetica font [5].

5.1.3 C#

C# is a simple general-purpose language that can be used to desktop applications, server-side code for websites, and even video games. It provides ease of use and power. It relies on the .NET Framework, which is a large platform that primarily consists of two parts.

The Common Language Runtime (CLR) and the Base Class Library (BCL). The CLR is a virtual machine that runs the code so that a physical computer does not run it. The BCL is a massive library of code that can be reused in our program to accelerate the process of building the project. The goal of the CLR as a virtual machine is to provide a single architecture to our code without regard to the physical hardware or OS used.

The process that the code goes through to execute is a bit longer and more complex because of the .NET framework. The C# compiler turns the code into Common Intermediate Language (CIL or IL). To share the CIL with others, it is packaged into a .exe or .dll file. The CIL code in our .exe or .dll file will run through the CLR to run the program.

Further advantages of the CLR is that it also serves as memory management and has a high level of control over what code can access the hard drive, the network, and other hardware. The CLR will, in order to keep things organized, move things around according to whether it is being used or not. Furthermore, since it is a virtual machine, a program cannot get access to the memory of other programs preventing code from adopting a virus-like behavior.[6]

5.2 Frontend

The frontend of our implementation will be responsible for creating the visual elements and structure of our website as well as linking the different pages together so that it is easy to navigate. It will also allow the user of the website to access the backend functionality of our platform through a guided user interface so that the user can read and write data in the database.

5.2.1 Bootstrap

Bootstrap is a frontend framework and open source project used for building responsive mobile-first sites [7]. It supports the latest, stable releases of all major browsers and platforms. It incorporates the languages CSS, HTML, and JS to make the components function and it requires that HTML5 doctype is used. It uses a responsive grid system that makes it easy to create a responsive design so that the webpage is functional on different devices [8].

The main reason we have chosen to use the Bootstrap framework is to create a website that is responsive and can adapt to different devices. This removes the need to worry about how different screen formats and resolutions affect the elements on the website. Another reason is that it limits the usage of inline CSS in our code. We still have to specify some special styling for some of the elements on the website to achieve the design we are looking for, but the scope of this has been sharply reduced because of the Bootstrap framework.

Another benefit of using Bootstrap is that the design of the website will be familiar to our users as many other websites also use Bootstrap. This will result in the element design and design conventions to be more easily understood. This, in return, will increase the usability of our system and improve the user experience.

The website's structure consists of multiple .cshtml files where each file represents a webpage on our website. Furthermore, we have a Layout.cshtml file that creates the elements that are present in the header and footer regions of all the pages. Each file will use the grid system of Bootstrap to arrange the content in columns. The width of these columns will be specified in percentage of the screen to further support the use of the system on different devices.

Navigation bar

Below is a representation of the Layout.cshtml file and the code that creates the navigation bar that is visible throughout the website.

Listing 5.2. Layout page

```

1      <header>
2          <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light
3              bg-dark border-bottom box-shadow mb-3">
4              <div class="container">
5                  <a class="navbar-brand" asp-area="" asp-page="/Index"></a>
8                  <button class="navbar-toggler" type="button" data-toggle="
9                      collapse" data-target=".navbar-collapse" aria-controls="
10                         navbarSupportedContent"
11                         aria-expanded="false" aria-label="Toggle navigation">
12                      <span class="navbar-toggler-icon"></span>
13                  </button>
14                  <div class="navbar-collapse collapse d-sm-inline-flex flex-sm
15                      -row-reverse">
16                      <ul class="navbar-nav flex-grow-1">
17                          <li class="nav-item">
18                              <a class="nav-link text-white" asp-area="" asp-
19                                  page="/Index">Home</a>
20                          <li class="nav-item">
21                              <a class="nav-link text-white" asp-area="" asp-
22                                  page="/Privacy">Privacy</a>
23                      </li>
24                      </ul>
25                  </div>
26                  <div>
27                      <ul class="navbar-nav navbar-right">
28                          <li>
29                              <partial name="_LoginPartial" />

```

```

23         </li>
24     </ul>
25 </div>
26 </div>
27 </nav>
28 </header>

```

For the navigation bar, we use the navbar class in Bootstrap as this creates a navigation bar that collapses depending on the screen size. This means that the navigation bar provides a better user experience for users with mobile devices. The elements that relates to the login and logout process are handled by the LoginPartial.cshtml file and to include these elements in the navigation bar we reference the partial file in its own list item on line 32. The content of the LoginPartial.cshtml file is visible below.

Listing 5.3. LoginPartial page

```

1  @using Microsoft.AspNetCore.Identity
2  @using unibook.Models
3
4  @inject SignInManager<User> SignInManager
5  @inject UserManager<User> UserManager
6
7  <ul class="navbar-nav">
8      @if (SignInManager.IsSignedIn(User))
9      {
10         <li class="nav-item">
11             <a id="manage" class="nav-link text-white" asp-area="Identity"
12                " asp-page="/Account/Manage/Index" title="Manage">Hello
13                @UserManager.GetUserName(User)!</a>
14         </li>
15         <li class="nav-item">
16             <form id="logoutForm" class="form-inline" asp-area="Identity"
17                asp-page="/Account/Logout" asp-route-returnUrl="@Url.Page(
18                ("/Index", new { area = "" })">
19                 <button id="logout" type="submit" class="nav-link btn btn
20                    -link text-white">Logout</button>
21             </form>
22         </li>
23         <li class="nav-item">
24             <a id="manage" class="nav-link text-white" asp-area="Identity"
25                " asp-page="/Account/CreateListing" title="Manage">Create
26                a new listing</a>
27         </li>
28     }
29     else
30     {
31         <li class="nav-item">
32             <a class="nav-link text-white" id="register" asp-area="
33                Identity" asp-page="/Account/Register">Register</a>
34         </li>
35         <li class="nav-item">
36             <a class="nav-link text-white" id="login" asp-area="Identity"
37                asp-page="/Account/Login">Login</a>
38         </li>
39     }
40 </ul>

```

The LoginPartial.cshtml primarily consists of an if-statement that checks if the user is signed in. If the user is signed in, the navigation bar will display the username and an option to logout of the system. If the user is not logged in the navigation bar will show the login option as well as the option to register a new user account.

Forms

For pages where we want to have the user input some information into the system, we use the form class from Bootstrap. This allows us to create input fields on a page that can save information to our database once the user clicks the submit button on the page. Each text field uses the name of the relevant database column so that the user knows what information should be entered into the specific text field. The pages that include forms are the pages for registering a user, creating a listing, and logging in. Below is a representation of the register page and the code that creates the input form.

Listing 5.4. Register form

```

1 <div class="row">
2   <div class="col-md-4">
3     <form asp-route-returnUrl="@Model.ReturnUrl" method="post" enctype="
      multipart/form-data">
4       <h4>Create a new account.</h4>
5       <hr />
6       <div asp-validation-summary="All" class="text-danger"></div>
7       <div class="form-group">
8         <label asp-for="Input.Email"></label>
9         <input asp-for="Input.Email" class="form-control" />
10        <span asp-validation-for="Input.Email" class="text-danger"></
          span>
11      </div>
12      <div class="form-group">
13        <label asp-for="Input.Name"></label>
14        <input asp-for="Input.Name" class="form-control" />
15        <span asp-validation-for="Input.Name" class="text-danger"></
          span>
16      </div>
17      <div class="form-group">
18        <label asp-for="Input.University"></label>
19        <input asp-for="Input.University" class="form-control" />
20        <span asp-validation-for="Input.University" class="text-
          danger"></span>
21      </div>
22      <div class="form-group">
23        <label asp-for="Input.PhoneNumber"></label>
24        <input asp-for="Input.PhoneNumber" class="form-control" />
25        <span asp-validation-for="Input.PhoneNumber" class="text-
          danger"></span>
26      </div>
27      <div class="form-group">
28        <label asp-for="Input.Address"></label>
29        <input asp-for="Input.Address" class="form-control" />
30        <span asp-validation-for="Input.PhoneNumber" class="text-
          danger"></span>

```

```

31         </div>
32         <div class="form-group">
33             <label asp-for="Input.City"></label>
34             <input asp-for="Input.City" class="form-control" />
35             <span asp-validation-for="Input.City" class="text-danger"></span>
36         </div>
37         <div class="form-group">
38             <label asp-for="Input.PostalCode"></label>
39             <input asp-for="Input.PostalCode" class="form-control" />
40             <span asp-validation-for="Input.PostalCode" class="text-danger"></span>
41         </div>
42         <div class="form-group">
43             <label asp-for="Input.Password"></label>
44             <input asp-for="Input.Password" class="form-control" />
45             <span asp-validation-for="Input.Password" class="text-danger"></span>
46         </div>
47         <div class="form-group">
48             <label asp-for="Input.ConfirmPassword"></label>
49             <input asp-for="Input.ConfirmPassword" class="form-control" />
50             <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
51         </div>
52         <div class="form-group">
53             <label asp-for="ImageNameInput">Add a profile picture</label>
54             <input type="file" asp-for="ImageNameInput" />
55         </div>
56         <button type="submit" class="btn btn-primary">Register</button>
57     </form>
58 </div>
59 </div>

```

Each input field uses the form-group class as well as a label and an input specification. The "asp-for" is an input tag helper that is available in ASP.NET. It makes sure that the form uses the HTML type attribute value based on the model type and data annotation attributes for the model property. The model properties for the register page can be seen below: (<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-3.1>)

Listing 5.5. Register model properties

```

1 public class InputModel
2     {
3         [Required]
4         [EmailAddress]
5         [Display(Name = "Email")]
6         public string Email { get; set; }
7
8         [Required]
9         [Display(Name = "Name")]
10        public string Name { get; set; }
11
12        [Display(Name = "City")]

```



```

13         public string City { get; set; }
14         [Display(Name = "Postal code")]
15         public string PostalCode { get; set; }
16
17         [Display(Name = "Address")]
18         public string Address { get; set; }
19
20         [Display(Name = "Phone number")]
21         public string PhoneNumber { get; set; }
22
23         [Display(Name = "University")]
24         public string University { get; set; }
25
26         [Display(Name = "Image")]
27         public string ImageName { get; set; }
28
29         [Required]
30         [StringLength(100, ErrorMessage = "The {0} must be at least {2}
31             and at max {1} characters long.", MinimumLength = 6)]
32         [DataType(DataType.Password)]
33         [Display(Name = "Password")]
34         public string Password { get; set; }
35
36         [DataType(DataType.Password)]
37         [Display(Name = "Confirm password")]
38         [Compare("Password", ErrorMessage = "The password and
39             confirmation password do not match.")]
39         public string ConfirmPassword { get; set; }

```

Displaying information

On some pages, we want to display information to the user such as different listings, their title, price, and also the contact information of the sellers. For these pages, we use the `@HTML.DisplayFor` function to show the information from the database relevant for that specific page. For the seller's contact information we list such elements as name, email, etc. as a list-group-item within a card class which creates a box with an image at the top and text at the bottom. This is meant to display the information in a very readable way to the user and also mirrors the representation of the information in the form on the registration page, which should help the users get familiar with the system. Below is the code that displays the seller's contact information:

Listing 5.6. Display contact information

```

1         <div class="card" style="width: 18rem;">
2             
4             <div class="card-body">
5                 <ul class="list-group list-group-flush">
6                     <li class="list-group-item">Name: <br> @Html.
7                         DisplayFor(model => model.User.Name) </li>
8                     <li class="list-group-item">Address: <br> @Html.
9                         DisplayFor(model => model.User.Address) </li>
10                    <li class="list-group-item">Phone: <br> @Html.
11                        DisplayFor(model => model.User.PhoneNumber) </li>

```

```

8         <li class="list-group-item">E-mail: <br> @Html.
          DisplayFor(model => model.User.Email)</li>
9         <li class="list-group-item">University: <br> @Html.
          DisplayFor(model => model.User.University)</li>
10        <li class="list-group-item">Rating: <br> @Html.
          DisplayFor(model => model.User.Rating)</li>
11    </ul>
12    <br>
13    <a href="#" class="btn btn-primary">Edit profile</a>
14</div>
15</div>

```

For displaying the listings we have gone for a design that also uses the card class to put an emphasis on the listing image and the most important information regarding each listing. This will improve the user experience when scrolling through the listings and help the user decide which listings could be interesting. Here you can see the code that creates a listing:

```

1        <div class="card" style=" margin-right: 1%; margin-bottom: 1%;
          width: 24%;">
2            
3            <div class="card-body">
4                <h5 class="card-title">
5                    @Html.DisplayFor(model => listing.Book.Title)
6                </h5>
7                <p>
8                    <b>
9                        @Html.DisplayNameFor(model => listing.
                          Price):
10                    </b>
11                    @Html.DisplayFor(model => listing.Price) kr
12                </p>
13                <p>
14                    <b>
15                        @Html.DisplayNameFor(model => listing.
                          BookISBN):
16                    </b>
17                    @Html.DisplayFor(model => listing.BookISBN)
18                </p>
19                <a asp-page="./Listing" asp-route-id="@listing.Id
                  " class="btn btn-primary">Go to listing</a>
20            </div>
21        </div>

```

Listing 5.7. Create Listing

This code is wrapped in a for-each loop that is used to make sure that a listing card is created for each listing item in the database. For the Index page, the for-each loop includes each listing item in the database and creates a card for each listing unless the user searches for a specific input. Then only listings related to that input will be shown as cards. However, on the user profile page, the for-each loop will only create listings that are related to that specific user.

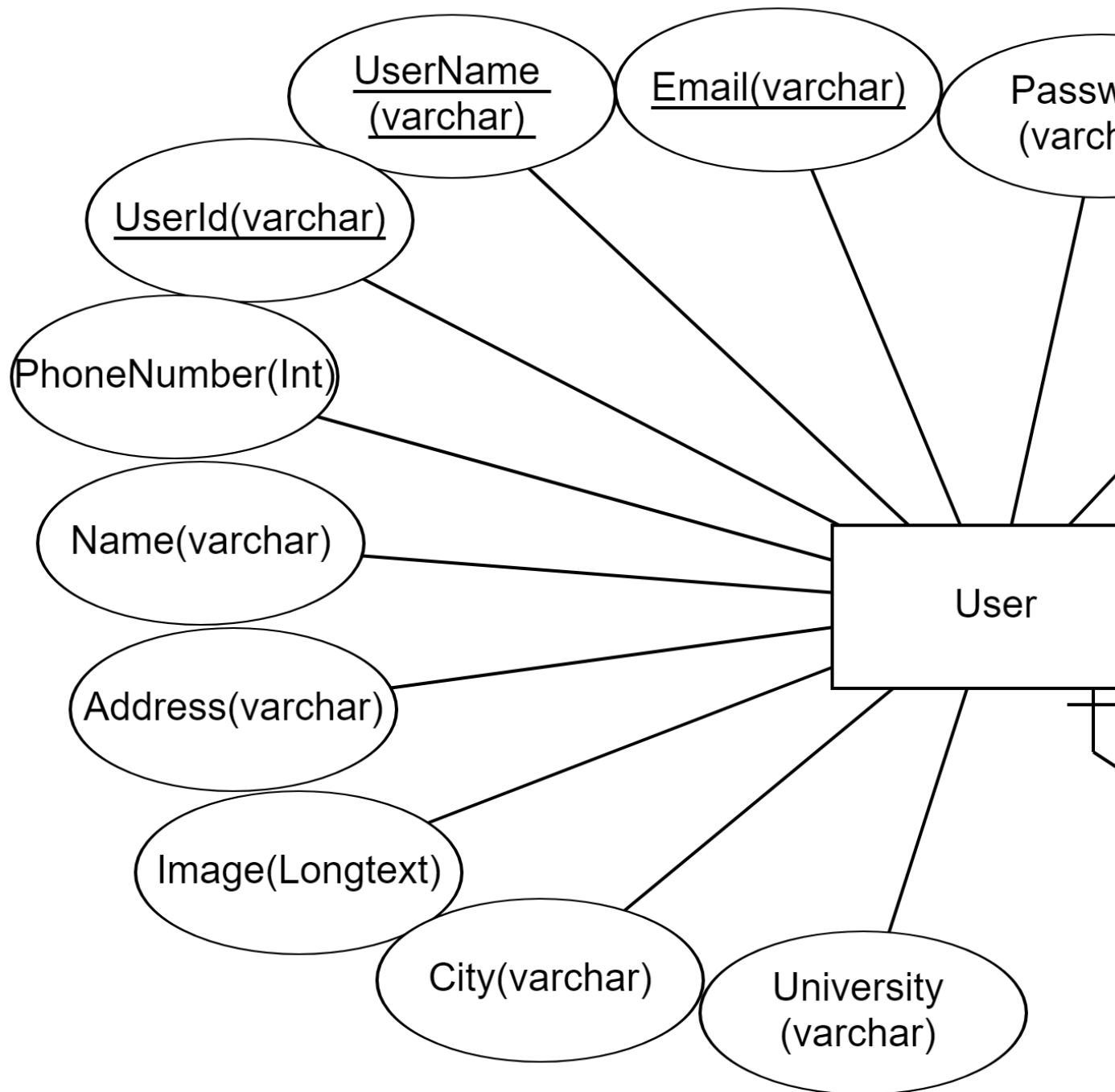
5.3 Database and server

Our server is based on Ubuntu Server ver. 18.04 and Nginx for web hosting. DigitalOcean has been chosen as our host. For choosing the domain name, we have used Namecheap. They have all been chosen through GitHub Student Developer Pack as the tools are free to access. For making our platform, we are using a .NET framework so that it is compatible with C#, and to extend the .NET framework further, we are using ASP.NET that results in further tools and libraries for building web applications. (<https://www.digitalocean.com/>) (<https://www.namecheap.com/>) (<https://education.github.com/pack>)

5.3.1 MySQL

With over 10 million installations, MySQL is probably one of the most popular database management systems for web servers. It was developed in the mid-90s and is today a mature technology that powers many internet destinations. We have chosen MySQL because it is a fast and powerful tool that can run on the most basic hardware. Another reason is that MySQL is highly scalable in the sense that it can grow with the website. In MySQL, the database is divided into tables that each contain different information. A table is consisting of rows and columns, and a database can consist of several tables.

The language used in MySQL is loosely based on English and is designed to allow simple requests from a database using commands. SQL stands for 'Structured Query Language.' (p.165-66, Nixon 2018) MySQL was preferred over Microsoft SQL Server as it only takes up 1GB of memory on our server, while SQL Server takes up 2GB, which we do not have. Each of the classes in our class diagrams has been allocated to their table. These tables have columns that are specific to that class. Each of the classes in our class diagrams has been allocated to their table. These tables have columns that are specific to that class. This is shown on the ERD Diagram below:



Entity-relation diagram

When creating a database, it is a great idea to start with modeling the entity-relation diagram (ERD). The E-R model is developed in conceptual design and consists of a collection of entities and relationships between these. The result of E-R modeling is an E-R diagram or schema. We use the model in database design to facilitate by allowing for the specification of the E-R diagram in such a way that the logical structure of a database is represented. The E-R model employs three basic concepts: entity sets, relationship sets, and attributes.

Entities

An entity is represented by a set of attributes such as a person being described by a name, address, phone, etc. The individual entities make up an entity set wherein the entity is stored, for example, persons, items. These are all entity sets that contain multiple persons and items. To describe an entity even further, entity types are created, which could be, for example, person. The entity type is then related to the attributes of the entity type, i.e., name, phone, address, etc.

What is important is that it can be uniquely identified through its attributes from other objects. This is also called a primary key. An example is students at a college. The primary key that is unique to all students is the student ID. Using this, we can identify an entity within an entity set.

An entity is an object which can be found in the real world. It has an independent existence and can thus be differentiated from other objects. Entities are often classified based on their strength, that is, whether its tables are existence dependent. If an entity can not exist without the relationship with another entity, it is called a weak entity. It would also mean that its primary key would be derived from the parent entity's primary key. A strong entity, on the other hand, is an entity that can exist apart from all of its related entities.

Attributes & relationships

Attributes are, as mentioned before, used to describe entities. If a book is an entity in a database, its attributes could be things such as author, ISBN, and edition. There are different kinds of attributes. There are simple single-valued attributes. In other words, these are atomic. These could be, using our book example, ISBN and edition. There are also multivalued attributes. Using our book example, these could be author, since a book can have multiple authors.

There are three types of relationships when it comes to the modeling of an ERD:

One to many (1:N): one to many relationships is one of the most common relationships in ERD. Using our book example, a book can be in many universities.

One to one (1:1): a one to one relationship is where an entity is only related to one other entity. This is rarely used in a database and can indicate that these two entities should be combined.

Ternary:, a ternary relationship, involves many to many relationships between three tables.

It is a one to many relationships from the books, as one listing only can contain one book, but an exact copy of the said book can be a part of many listings. The listings "have a" book attached to them, while the listings are attached to a user. This way, the ERD Diagram and thereby the

database model, is very similar to the class diagram, where the books aggregate the listings, and then the listings aggregate the users. The database represents the vision for the system design that was intended. (ERD Kilde)

The Book table has columns such as ISBN, Author, Edition, and Title. The ISBN column is set to be variable characters as it can contain both numbers and dashes. The rest are longtexts that contain strings. The ISBN is the primary key, as it is unique for only one book. This makes it perfect as a primary key. [9]

Field	Type	Null	Key	Default	Extra
ISBN	varchar(255)	NO	PRI	NULL	
Author	longtext	YES		NULL	
Edition	longtext	YES		NULL	
Title	longtext	YES		NULL	

Figure 5.2. Books table

The Listings table acts as the middleman in the database structure. It contains columns relevant for the listing, such as a semester, study, university, description, and a price that takes in an integer as it is a number. Each listing is given an ID that auto increments, meaning that every time a new listing is created, it will take the next free ID number. It also contains an ISBN for the book that is being sold, as well as a UserID that ties it to a specific user. These are both foreign keys from the two other tables.[9]

Field	Type	Null	Key	Default	Extra
Id	int(11)	NO	PRI	NULL	auto_increment
Semester	longtext	YES		NULL	
Study	longtext	YES		NULL	
University	longtext	YES		NULL	
Description	longtext	YES		NULL	
Price	int(11)	NO		NULL	
UserId	varchar(255)	YES	MUL	NULL	
BookISBN	varchar(255)	YES	MUL	NULL	

Figure 5.3. Listings table

The users table also has an ID that auto increments, while also containing a user-defined UserName, as well as other information about the user, that might be relevant in an exchange. The password is hashed, which is done by the ASP.NET Identity system mentioned in section X.X. The unique UserID is varchar, but this is also a feature in the Identity system, which makes the UserIDs a mix of numbers and letters separated by dashes.

Field	Type	Null	Key	Default	Extra
Id	varchar(255)	NO	PRI	NULL	
UserName	varchar(256)	YES		NULL	
Email	varchar(256)	YES		NULL	
PasswordHash	longtext	YES		NULL	
PhoneNumber	longtext	YES		NULL	
Name	longtext	YES		NULL	
Address	longtext	YES		NULL	
Rating	double	NO		NULL	

Figure 5.4. Users table

5.4 ASP.NET

ASP.NET is an open-source web framework for building fast and secure web apps. It is an extension of the .NET framework. ASP.NET helps the developer with creating websites using HTML, CSS, and JavaScript and lets the websites use the build-in features. ASP.NET uses a programming syntax called "Razor" to allow the developer to create dynamic web pages. Razor allows the developer to run server-based code (C#) on their website. Razor is fundamentally just HTML and C# combined. When creating a Razor page, the user can input all the HTML they want to use for their website and then type in an "@" to start writing C# code.

5.4.1 MVC and Razor Pages

When creating a website using ASP.NET, the developer can use either MVC (Model, View, Controller) or Razor Pages to handle the individual pages. We decided to use Razor Pages. These two are very similar in their operations. It is only the environment within Visual Studio that is different. With MVC, the application is separated into three main layers: models, views, controllers. This makes each layer have one responsibility, also known as Single Responsibility Testing (SRP) and Separation of Concerns (SoC). In software engineering, it is considered good practice to keep similar functionalities as a unit. With this approach, a user request is routed to the controller, which uses the Model to retrieve data and perform actions. The controller selects a corresponding View for the user and provides the relevant data from the Model.

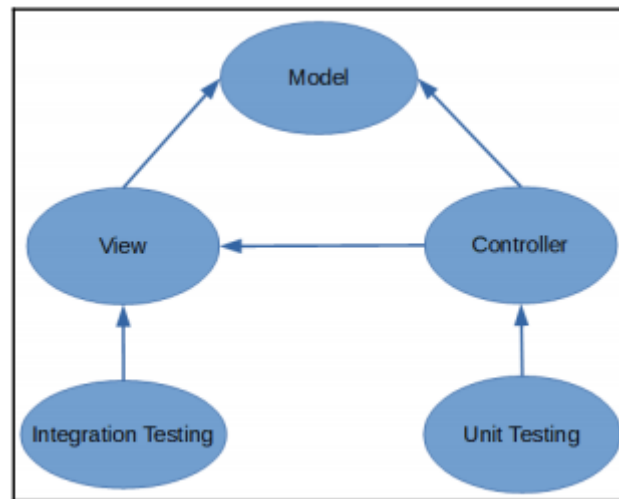
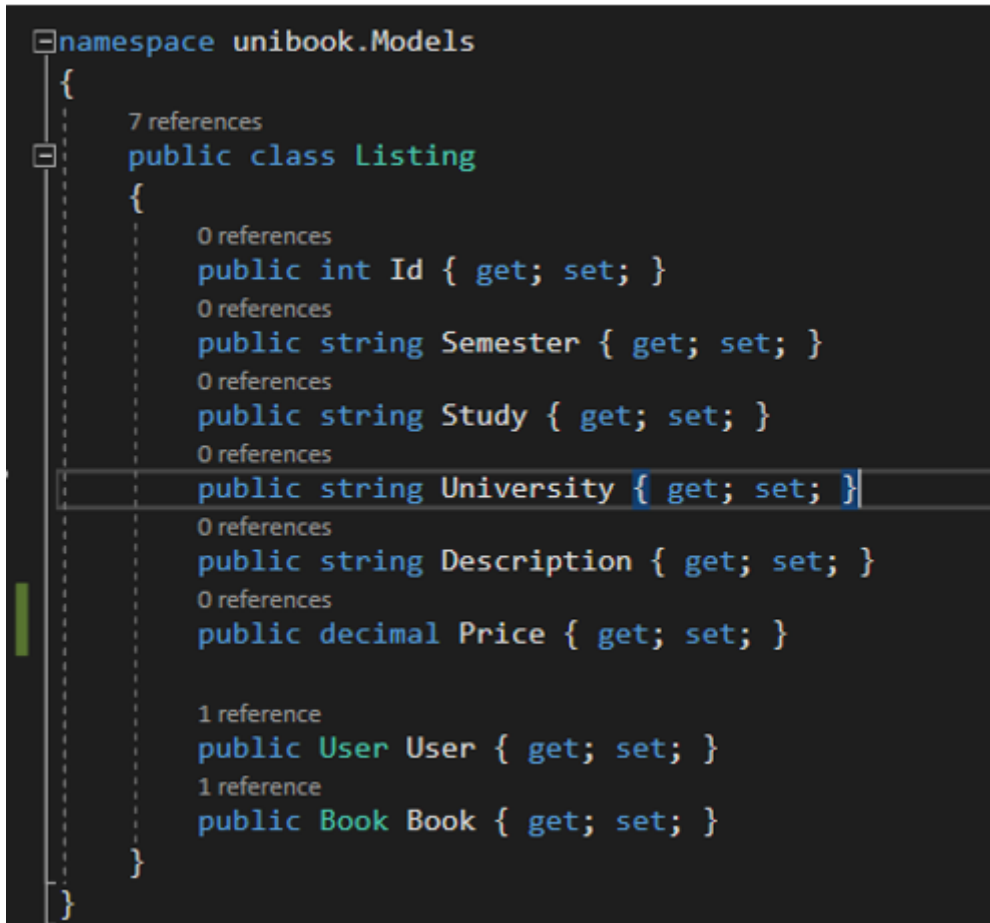


Figure 5.5. (Learn ASP.NET Core 3 Second Edition, Kenneth Yamikani Fukizi Jason De Oliveira Michel Bruchet, 2019)

Model

The model contains the logical data structures as well as the data of the given application. To make a database and an ASP.NET application communicate, it is a good idea to use the Entity Framework, which makes it easy to communicate with the database using C# syntax.

A screenshot of a code editor showing the implementation of a C# class named 'Listing' within the 'unibook.Models' namespace. The class has several public properties with get and set accessors: 'Id' (int), 'Semester' (string), 'Study' (string), 'University' (string), 'Description' (string), and 'Price' (decimal). It also has two public properties for associations: 'User' (type 'User') and 'Book' (type 'Book'). The code is color-coded, and the 'University' property line is highlighted with a mouse cursor. A vertical scrollbar is visible on the left side of the code editor.

```
namespace unibook.Models
{
    7 references
    public class Listing
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Semester { get; set; }
        0 references
        public string Study { get; set; }
        0 references
        public string University { get; set; }
        0 references
        public string Description { get; set; }
        0 references
        public decimal Price { get; set; }

        1 reference
        public User User { get; set; }
        1 reference
        public Book Book { get; set; }
    }
}
```

Figure 5.6. Model for listings

We use our model as an entity model, since we use Razor Pages, and we do not have to create a model folder, as one would do in MVC.

View

The view is responsible for creating the visual representation and user interface of the application. In ASP.NET, these are created using HTML, and Razor markup as well as other Razor components. Most often, the view has a '.cshtml' extension. A view can contain a complete web page, and web page part (partial view), or a layout. In our case, the view is pages. Our pages control everything. In the context of MVC, here lies both the controller and the model.

Controller

A controller manages the interactions between models and views. It controls the logical behavior of the application. It chooses which view is to be rendered for a specific user request.

Razor Pages

In Razor Pages, the model and the controller is implemented within the page itself (View). This is the main difference between MVC and Razor Pages. This makes our environment look like so:

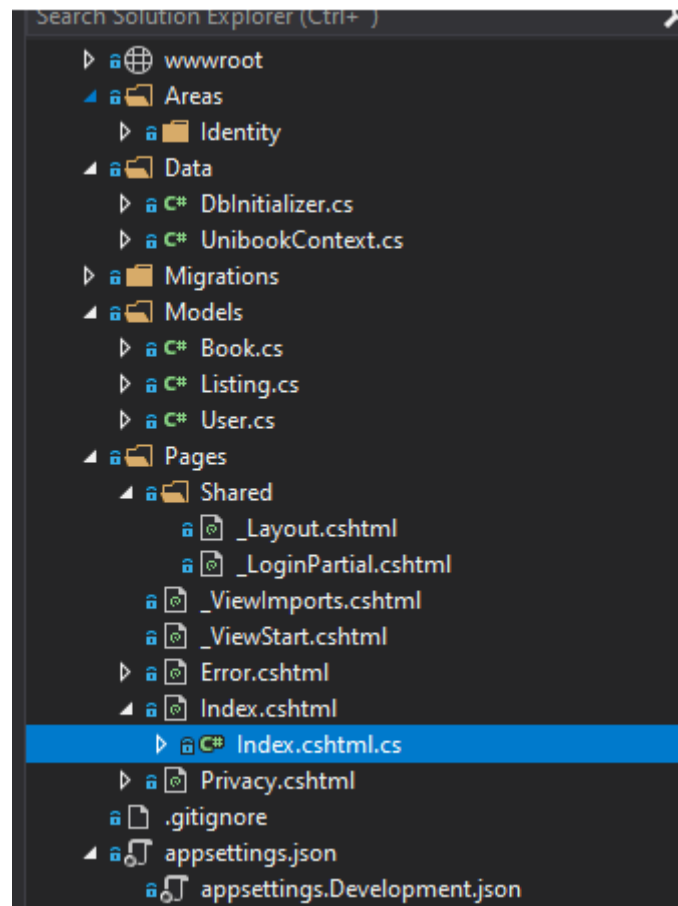


Figure 5.7. Razor pages environment

5.4.2 The default classes in ASP.NET

When creating an ASP.NET project, some default classes are implemented automatically. These include the Program class and the Startup class. These two classes are essential for the program even to start.

Program class

The Program class is the main entry point for ASP.NET applications. This is similar to the console apps that can be created using the .NET framework. The ASP.NET applications' Program class, like the console applications, has a Main method, which is executed when running the application. This is because an application based on ASP.NET is a console application hosting a web application.

Usually, it is not necessary to change anything in the Program class, since it is configured by default and thus includes everything necessary to run the application.

By default, the Program looks a follows:

```
1 namespace unibook
2 {
3     public class Program
4     {
5         public static void Main(string[] args)
6         {
7             CreateHostBuilder(args).Build().Run();
8         }
9         public static IHostBuilder CreateHostBuilder(string[] args) =>
10             Host.CreateDefaultBuilder(args)
11                 .ConfigureWebHostDefaults(webBuilder =>
12                 {
13                     webBuilder.UseStartup<Startup>();
14                 });
15     }
16 }
17
```

Listing 5.8. Program Class

5.4.3 Startup class

The Startup class is responsible for the preloading and configuration of services. The Startup class and the Program class are the foundation of any ASP.NET applications.

Within the Startup class, there are certain methods which are important to acknowledge, since they will require the developer's attention on quite a frequent basis.

The first of these is the ConfigureServices method, which handles dependency injection. This method is called by routine and is used to add different services to the application. This application uses the AddRazorPages method to enable the solution to run Razor Pages. Furthermore, it uses the DbContext method, which takes the UnibookContext class, the custom context class, as a parameter. The MySQL connection is then typed out with the connection details. This includes the server, the port, and user credentials. This makes sure that the connection to the database happens on startup.

Lastly, the AddIdentity service is used to enable the use of ASP.NET's Identity platform. This method takes the user and their role as parameters. Identity contains a userManager as well as a signInManager. It also adds columns to the User table in the database, such as email, password, and comprehensive security measures such as the option to include two-factor authentication. The EntityFrameworkStores are also added to the services.

The second method is to Configure. It is used to configure the HTTP request pipeline. It also handles middleware and routing. In this case, it contains mostly the generic ASP.NET Config, with the ability to use a developer or production environment by the parameter of the IWebHostEnvironment. (<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-3.1#tabs=windows>)

The Unibook Startup class looks as follows:

```
1 namespace unibook
2 {
3     public class Startup
4     {
5         public Startup(IConfiguration configuration)
6         {
7             Configuration = configuration;
8         }
9         public IConfiguration Configuration { get; }
10        // This method gets called by the runtime. Use this method to add
11        // services to the container.
12        public void ConfigureServices(IServiceCollection services)
13        {
14            services.AddRazorPages();
15
16            services.AddDbContext<UnibookContext>(options =>
17                options.UseMySQL("Server=127.0.0.1;Port=3306;Database=
18                    UnibookDatter;User=root;Pwd=*****;Connection
19                    Timeout = 120;"));
20            services.AddIdentity<User, IdentityRole>()
21                .AddEntityFrameworkStores<UnibookContext>();
22        }
23        // This method gets called by the runtime. Use this method to
24        // configure the HTTP request pipeline.
25        public void Configure(IApplicationBuilder app, IWebHostEnvironment
26            env)
27        {
28            if (env.IsDevelopment())
29            {
30                app.UseDeveloperExceptionPage();
31            }
32            else
33            {
34                app.UseExceptionHandler("/Error");
35                app.UseHsts();
36            }
37            app.UseStaticFiles();
38            app.UseAuthentication();
39            app.UseRouting();
40            app.UseEndpoints(endpoints =>
41            {
42                endpoints.MapRazorPages();
43                endpoints.MapControllers();
44                endpoints.MapControllerRoute("default", "{controller=Home}/{
45                    action=Index}/{id?}");
46            });
47        }
48    }
49 }
```

Listing 5.9. Startup class

5.5 Functionality

In this section, the system's main functionality will be presented so that each function's purpose and structure are clear. The system consists of many parts that could be considered as functionality. However, this rapport will only go into detail regarding those functions and operations that were recognized as a part of the component design. The functions search, create, edit, and calculate, as well as the operation search.

5.5.1 Searching

The search operation is a part of the index page that serves as the landing page for our website. To interact with the search operation the user needs to click the search field and type words or characters related to the book that the user would like the system to search for. The system will then search for books related to the user input and display the listings that contain the related books as the search result. The search result is then displayed on a separate page and the user will be able to click on the listings displayed or search for something else. The code for displaying the search field and for handling the operation itself is shown below:

```
1 <div class="col-md" style="padding-left: 0%; padding-right: 1%;">
2     <form method = "GET" asp-page="./SearchResult">
3         <p>
4             <input class="form-control" type="text" placeholder="Search"
5                 aria-label="Search" name="SearchString">
6         </p>
7     </form>
8 </div>
```

Listing 5.10. Search bar on Index page

```
1 public IActionResult OnPost(string SearchString)
2     {
3         SearchString = (SearchString ?? "").ToLower();
4
5         Listings = _context.Listings
6             .Where(b => b.Book.Title.ToLower().Contains(SearchString) ||
7                 b.Book.ISBN.ToLower().Contains(SearchString) || b.Book.
8                 Author.ToLower().Contains(SearchString))
9             .ToList(); // SELECT * FROM Listings WHERE title LIKE '%test
10                        %';
11         return RedirectToPage("./SearchResults");
12     }
```

Listing 5.11. EF Core ORM Query

The critical thing to note here is the name of the text input field, which is then used as a parameter for the OnPost() method that is placed in the behind lying page model. The text is put into the following code when the user hits enter after having inputted the search criteria. This then redirects to the search results page. Here it takes every element from the listings, where the lambda expression of `b=> b.Book.Title` contains the query. It does this for the Title, ISBN and Author, before finally turning them into a list. This list is then presented to the user on the same page, by a for each loop.

Rather than using standard SQL Queries, the application will be using Entity Framework Core to handle the data access. EF Core is used as an object-relational mapper (ORM), which allows the developers to work with our MySQL database while using .NET objects. This eliminates a lot of manual queries, as properties and methods replace them.

Here the string is first made to lower so that there is no case sensitivity. Then the database sets the list of Listings to equal `_context.Listings`, which is the database link property defined in the constructor. The `ILogger` is a type that enables logging on the page. This can be seen in the image below:

```
1 private readonly ILogger<IndexModel> _logger;  
2 private readonly UnibookContext _context;  
3 public IndexModel(ILogger<IndexModel> logger, UnibookContext context)  
4 {  
5     _logger = logger;  
6     _context = context;  
7 }
```

Listing 5.12. IndexModel constructor with DbContext and ILogger

Each listing within the list of listings, which contains the posted query is displayed to the user. This is done by using the card form and the `@Html.DisplayFor`, which allows for the use of variables and methods within the HTML code. This is also why the pages are of the file format `cshtml`. Each listing has a picture, which has a dynamic image source depending on the listing that is being displayed. The title and price, as well as the ISBN of the book being sold, are then displayed below. The card also includes a button that contains a dynamic link that takes the user to the individual listing site.

```

1 <div class="row" runat="server" style="margin-left: 0px; margin-right: 0px;
  width: 100%;">
2   @foreach (var listing in Model.Listings.Where(l => l.University == "
    Aalborg University"))
3   {
4       <div class="card" style="margin-right: 1%; margin-bottom: 1%; width:
        24%;">
5           
6           <div class="card-body">
7               <h5 class="card-title">
8                   @Html.DisplayFor(model => listing.Book.Title)
9               </h5>
10              <p>
11                  <b>
12                      @Html.DisplayNameFor(model => listing.Price):
13                  </b>
14                      @Html.DisplayFor(model => listing.Price) kr
15              </p>
16              <p>
17                  <b>
18                      @Html.DisplayNameFor(model => listing.BookISBN):
19                  </b>
20                      @Html.DisplayFor(model => listing.BookISBN)
21              </p>
22              <a asp-page="./Listing" asp-route-id="@listing.Id" class="btn
                btn-primary">Go to listing</a>
23          </div>
24      </div>
25  }
26 </div>

```

Listing 5.13. Foreach-loop displaying listings

5.5.2 Creating

There are two types of data that is being created within the Unibook application. The creating of users and the creating of listings are the two functions that inserts new information into the database. Both use forms that contain textfields in which the user inputs the necessary information. Both are also scaffolds as Razor Pages.

Listing creation

The listing creation is based on an inputmodel that contains properties for the values. Every property except the image is marked with a [required], which means that the user has to fill those

out when creating a new listing. The writing to the database is then done in the `OnPostAsync` method within the modelpage of the `CreateListing` model.

```

1 public async Task<IActionResult> OnPostAsync(string returnUrl = null)
2     {
3         returnUrl = returnUrl ?? Url.Content("~/");
4         if (ModelState.IsValid)
5         {
6             var book = await _context.Books.FirstOrDefaultAsync(b => b.
                ISBN == Input.ISBN);
7             if (book == null)
8             {
9                 var bookcreator = new Book()
10                {
11                    Title = Input.Title,
12                    Author = Input.Author,
13                    Edition = Input.Edition,
14                    ISBN = Input.ISBN,
15                };
16                _context.Books.Add(bookcreator);
17            }
18            var fileName = ListingImageInput == null ? "DefaultListing/
                Book-Placeholder.png" : GetUniqueName(ListingImageInput.
                FileName);
19            var Images = Path.Combine(hostingEnvironment.WebRootPath, "
                Images/ListingImages");
20            if (ListingImageInput != null)
21            {
22                var filePath = Path.Combine(Images, fileName);
23                this.ListingImageInput.CopyTo(new FileStream(filePath,
                    FileMode.Create));
24                this.Listing.ListingImage = fileName; // Set the file
                    name
25            }
26            var listingcreator = new Listing()
27            {
28                Book = book,
29                User = await _userManager.GetUserAsync(User),
30                Price = Input.Price,
31                University = Input.University,
32                Semester = Input.Semester,
33                Study = Input.Study,
34                Description = Input.Description,
35                BookISBN = Input.ISBN,
36                ListingImage = fileName
37            };
38            _context.Listings.Add(listingcreator);
39            await _context.SaveChangesAsync();
40        }
41        return RedirectToPage("./Index");
42    }

```

Listing 5.14. Listing creation model

[caption=Get unique name method,captionpos=b] First the method checks if the modelstate is valid, which means that there is no errors in the input. An example of an invalid modelstate

would be a non-number entry to a numeric input. It then checks if the book that is being created in the listing is already in the database, in order to avoid duplicates. If the book is new to the system it is created with the appropriate information. It then handles the image input via a property `ListingImageInput` that is of the type `IFormFile` which enables it to handle image files. Then it uses a method `GetUniqueName` that gives the image a unique name in the database. The method is shown below:

```
1 private string GetUniqueName(string fileName)
2     {
3         fileName = Path.GetFileName(fileName);
4         return Path.GetFileNameWithoutExtension(fileName)
5             + "_" + Guid.NewGuid().ToString().Substring(0, 4)
6             + Path.GetExtension(fileName);
7     }
```

It uses GUID (global unique identifier) to randomize the filename. The original filename is followed by an underscore and then the randomized number. If no image is uploaded it uses a default picture as a placeholder. Then it creates the listing by using the new operator that contains the book from previously, as well as the user that creates the listing. The rest are taken from the input from the form. It then adds the listing creator to the database with an ID that is automatically incremented. Finally it redirects to the index page where the listing is then displayed under the university of choice.

User creation

The user creation uses the same type of input model with the appropriate properties. It does however contain a password that is hashed by the ASP.NET Identity platform which is denoted by the `DataType.Password`. It reuses the methods for image handling but instead it puts the files in the `UserImages` folder, in order to separate the user images and the listing images.

```
1 var user = new User { Email = Input.Email,
2                       UserName = Input.Email,
3                       Name = Input.Name,
4                       University = Input.University,
5                       City = Input.City,
6                       Address = Input.Address,
7                       PostalCode = Input.PostalCode,
8                       ImageName = fileName };
9 var result = await _userManager.CreateAsync(user, Input.
    Password);
10
11 if (result.Succeeded)
12 {
13     _logger.LogInformation("User created a new account with
        password.");
14
15     await _signInManager.SignInAsync(user, isPersistent:
        false);
16     return LocalRedirect(returnUrl);
17 }
18 foreach (var error in result.Errors)
19 {
```

```

20         ModelState.AddModelError(string.Empty, error.Description)
21     };

```

Listing 5.15. User creation

It uses the `_userManager` property to access the `userManager` within the Identity platform. Since the `OnPost` is an asynchronous method, the creation of the user is noted with an "await". The result variable is also used with the logger to log the creation of a new user. The user is automatically signed in following the registration process. Should any error occur, the foreach loop of `result.Errors` will display them with a description of which error occurred in the model state.

5.5.3 Editing

Just like the create function, two types of data can be edited in the Unibook application. Users can edit listings and their profiles if the information contained within those needs to be updated. A user might have taken a bad picture or made a typo in the creation of their listing/user, so instead of having to create a new one, the ability to edit was implemented. Furthermore, it is also possible to completely delete a listing or one's user from the system. If a user still has a listing associated with the user profile, the system will delete these listings as well as the user profile itself.

Both the edit and delete function for both the listings and the users are programmed using scaffolding in RazorPages. The edit function for both listings and users are also very similar to the create function as it uses the same structure in the input form. The only significant difference is that the edit listing function does not allow for users to edit the information related to the book, as this will lead to discrepancies in the relationship between the listing and the book associated with that listing.

Edit listing

The edit listing model uses a bind property with the listing type to display the information already stored in the system about the specific listing. The property gets this information by calling "await `_context.Listing`" within the `OnGet` method that runs when the edit listing page is opened. The `await _context.Listing` also includes the book and user information for the specific listing when called. This can be seen in the code below:

```

1  public async Task<IActionResult> OnGetAsync(int? id)
2  {
3      if (id == null)
4      {
5          return NotFound();
6      }
7
8      Listing = await _context.Listings
9          .Include(l => l.Book)
10         .Include(l => l.User).FirstOrDefaultAsync(m => m.Id == id);
11     if (Listing == null)
12     {
13         return NotFound();

```

```

14         }
15         ViewData["BookISBN"] = new SelectList(_context.Books, "ISBN", "
            ISBN");
16         ViewData["ListingImage"] = new SelectList(_context.Listings, "
            ListingImage", "ListingImage");
17         return Page();
18     }

```

Listing 5.16. OnGet method for edit listing

When this user has finished with the changes to the listing and clicks Save, the OnPost method is called which is very similar to the OnPost method in the create listing section. The only major difference is the following lines of code:

`await _context.SaveChangesAsync();` The first line attaches the entity state mark named "modified" to the state of the entity named listing. This results in a database update for the specific listing item with the new data when the second line of code is activated, which uses the `SaveChangesAsync` method to save the updates to the database.

Edit user

The edit user button is accessed from the profile page, and allows the user to edit their profile or administrative actions such as change password and delete personal data. The user model, based on the Identity platform, was lacking some properties and therefore has been customized. This also meant that the edit page needed if statements to overwrite the input. These statements look as follows:

```

1     if (Input.Name != user.Name)
2     {
3         user.Name = Input.Name;
4     }
5     if (Input.City != user.City)
6     {
7         user.City = Input.City;
8     }
9     if (Input.PostalCode != user.PostalCode)
10    {
11        user.PostalCode = Input.PostalCode;
12    }
13    if (Input.Address != user.Address)
14    {
15        user.Address = Input.Address;
16    }
17    if (Input.University != user.University)
18    {
19        user.University = Input.University;
20    }

```

Listing 5.17. If statements in the user model

The edit of the user is also unique in the sense that it uses the `UpdateAsync` method for the user once the modelstate is valid and the changes have been made in the form and has gone through the images code as well as the if statements. This method is unique to the users in Identity, and

therefore takes only the parameter of a single user. Once the information of the user is updated, the page is refreshed with the new login information. Finally the user is redirected after receiving a status message.

```

1 public async Task<IActionResult> OnPostAsync()
2     {
3     var user = await _userManager.GetUserAsync(User);
4         if (user == null)
5         {
6             return NotFound($"Unable to load user with ID '{_userManager.
              GetUserId(User)}'");
7         }
8         if (!ModelState.IsValid)
9         {
10            await LoadAsync(user);
11            return Page();
12        }
13
14    await _userManager.UpdateAsync(user);
15
16    await _signInManager.RefreshSignInAsync(user);
17    StatusMessage = "Your profile has been updated";
18    return RedirectToPage();
19

```

Listing 5.18. Edit user OnPost method

5.5.4 Calculate

The calculate function is used in relation to the rating feature of the system. This feature allows users to rate other users based on the experience they have had buying books for this other user. The rating is between 1 and 5 and is shown as an average of all the separate ratings the user has received. The code for displaying the rating feature is shown below:

```

1     @if (!Model.ShowEdit && Model.ShowRating)
2     {
3         <form asp-route-id="@Model.Profile.Id" method="post"
              enctype="multipart/form-data">
4             <div class="form-group" style="padding-left: 7%">
5                 <label asp-for="Input.Rating"> Rate this user
6                     </label>
7                 <select asp-for="Input.Rating" class="form-
              control">
8                     <option>1</option>
9                     <option>2</option>
10                    <option>3</option>
11                    <option>4</option>
12                    <option>5</option>
13                </select>
14            </div>
15            <div style="padding-left: 7%">
16                <button type="submit" class="btn btn-primary"
17                    >Rate</button>
18            </div>
19        </form>
20    }

```

Listing 5.19. Rate feature

The if statement makes sure that the rating feature is only visible to the user if they are logged in to the system, and the rating is done by selecting one of the five options displayed in the dropdown menu. When the user clicks the "Rate" button, the OnPost method is called and run. The OnPost method can be seen below:

```

1 public async Task<IActionResult> OnPostAsync(string id, string returnUrl =
  null)
2     {
3         await LoadProfileAsync(id);
4
5         returnUrl = returnUrl ?? Url.Content("~/");
6         if (ModelState.IsValid)
7         {
8             var ratingcreator = new Ratings()
9             {
10                 Rating = Input.Rating,
11                 UserId = Profile.Id,
12                 Rater = await _userManager.GetUserAsync(User)
13             };
14             _context.Ratings.Add(ratingcreator);
15             await _context.SaveChangesAsync();
16         }
17         return Page();
18     }

```

Listing 5.20. OnPost method for rate feature

The OnPost method creates a new rating object in the "ratingcreator" variable if the model state is valid. The item is created with the rating from the user input, the ID of the user that is being rated and the user ID of the person giving the rating. This data is saved on the properties of the rating class and the new object in the "ratingcreator" variable is then used as a parameter in the Add method that adds the object to the Ratings table in the database. The actual calculation of the rating average is handled by the rating property in the user model as seen below:

```

1 public double Rating => Ratings.Any() ? Ratings.Select(r => r.Rating).Average
  () : 0;

```

Listing 5.21. Rating p

The rating property is set to the average of the ratings in the "ratings" table if there are any. This makes sure that the calculation does not fail by accidentally dividing by zero. The rating property can then be called on the profile page to display the calculated average of the rating related to that user.

Evaluation 6

6.1 Usability Evaluation

6.1.1 Method

To ensure that the system would be easily understood by the potential users, a usability test has been conducted. The test users were chosen to be as representative as possible for the intended users of the system. The main requirements were that the users were students at a university and were presentable for the appropriate age group. Therefore, the test users were in the age group of 20-29 years old. The goal of the usability test was to identify problems with the design of the system. The usability test will result in a greater understanding of the system in such a way that the test users will show which parts are well functioning and which parts of the design that the test users have difficulties with. The usability problem that the test users run into during the test will be ranked on a list[10].

The test users were informed of the scenario, in which the tasks take place, before getting the tasks from the interviewee. The test user was asked to confirm that the question was understood, and it was made clear that if they had any questions regarding the tasks, they could ask right away. This was a possibility since the tasks were formulated in English, whereas the test users were all Danish. The tasks were completed when the test user had met the success criteria for the given task.

For the usability test, eight test users were chosen at the novice-user level as, according to Jakob Nielsen, this would yield a 90% confidence interval with a 25% deviation. This interval is deemed sufficient for the usability test.[10]

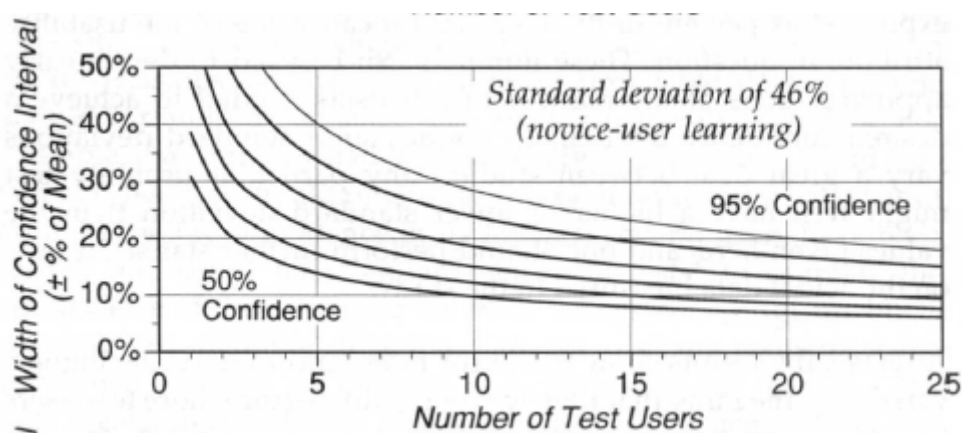


Figure 6.1. Confidence interval for novice users

Furthermore, the more test users that are included in the usability test, the more the results are reliable, but Nielsen also points out that the pay-off ratio between the number of test users and benefits starts dropping off. For this reason, eight test users were chosen.[10]

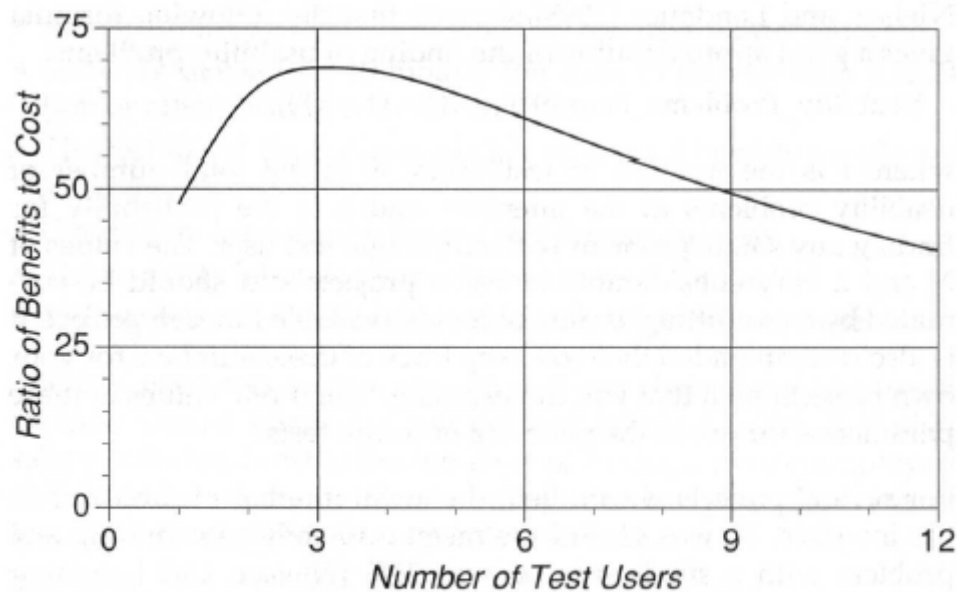


Figure 18 The pay-off ratio (how much larger the benefits are than the costs) for user tests with various numbers of test users under the assumptions for a “typical” medium-sized project described in the text.

Figure 6.2. Relationship between the benefits to cost and number of test users

Pilot test

Before executing the usability test, a pilot test was made to minimize faults during the test of the system or misunderstandings in the tasks given to the test users. From the pilot test, we learned that we had forgotten to include the ISBN of the book that the evaluators were asked to make in task 5. After our pilot test, we also concluded that some kind of indication that the listing had been created as needed. We decided to redirect the user to the index page after they had created a listing. Furthermore, some technical errors within the site that hindered the test was also discovered and corrected before the usability tests.[10]

This resulted in the table below that represents the final edition of the tasks the test users were asked to perform during the usability test.

Assignments	Task Scenario	Success criteria	Justification
1	You have finished 4. semester and want to sell your books and at this time you stumble upon Unibook. Go to unibook.me	The user goes to unibook.me	In this system, the user will have to access the system through their browser. The system also allows guest users. Therefore, it is necessary that we test this feature in our usability test
2	You want to be prepared for next semester. You need a book called "Computer Systems: A Programmer's Perspective"	The user finds the correct book	The users may from time to time access the website because they want to buy a book which they need for a certain semester. The user should be able to find the books they need on the website.
3	You realize you can't find any information about the seller before you have signed up. Go ahead and sign up.	The user registers for the site	To find user information on the website, the user needs to be signed in, since the guest users are not able to access this information. The user should be able to sign up for the website without it being too cumbersome.
4	Find seller information for the book.	The user finds the seller information.	Since all communication between the users is acted out outside the system, the users should be able to easily find the necessary information about the users, which they want to buy from.
5	You have just completed the course GOOP for Ball 4th semester at Aalborg University. You wish to sell your book for this course. Go ahead and create a listing. Book information: Title: C# player's guide ISBN: 978-3-16-148410-0 Price: 249 Author: RB Whitaker Edition: 2nd University: AAU Study: Ball	The user successfully creates a listing for the correct book	The user may want to sell a book of their own. It should be easy for the user to create a listing of their own.
6	You realize that the price you entered for the book, wasn't the correct one. Change the price of the book to 349	The user edits the listing's price to 349	Users may enter some incorrect information in their listing information. The user may also want to lower the price or increase it.
7	You remember that you also need a book for next semester. The book is called Introduction to Algorithms. Go ahead and find it on the site.	The user finds the correct book	The user should be proficient with their navigation on the site and be able to easily find the books they need, since this is one of the fundamental parts of the system.
8	You found the listing. Now find contact information for the seller.	The user finds the contact information	It should be easy to find information about the seller they want to buy from.
9	Before contacting the seller, you realize that you need to change your profile picture, because the picture you have right now, isn't very good.	The user changes their profile picture	If the user hasn't made a profile picture yet, and they want one, they should be able to find the "edit profile"-feature of the site. From here, they can also update their contact information.
10	The deal went through and you got the new book. Go ahead and rate the user you made a deal with.	The user rated the other user	The system is reliant on people being honest with each other. The user should thus be able to give feedback about individual sellers for others to see.
11	You realize that you don't want to have a user on the site anymore. Go ahead and delete your user and all your userdata.	The user successfully deletes their data.	To be GDPR-compliant, we want the user to be able to delete all the information we have on them in the system, so that they feel secure about entering their personal information on the site.

Figure 6.3. Final edition of the usability task

SUS - System Usability Scale

As a debriefing method of the usability test, the System Usability Scale (SUS) questionnaire is issued to the test users. It is a method that is quick, cheap, and reliable that has been tested thoroughly throughout the years through different studies, which also makes it valid. The SUS questionnaire contains ten questions that each test user is asked to answer on a scale from '× Strongly Disagree' to 'Strongly Agree.' Each question has a weight of 10 points resulting in a max score of 100 points. The average SUS score is 68, which means that an average score of 68 from our test users would put us in the 50th percentile (<https://uiuxtrend.com/measuring-system-usability-scale-sus/interpretation>).

Usability test environment

Due to the current world situation, it was not possible to meet up with the test users and conduct the tests in a facility, lab, or office. Instead, the project group decided to conduct the usability test on the communication platform, Discord, which enabled the test to be conducted with audio and screen sharing. This swayed the need for transportation and made it possible for the project group to reach out to potential test users across the nation, making for a more representable population of test users. Furthermore, the test was conducted with screen recording, which made it able to oversee the tests further and investigate each usability problem for each test user. It also serves as a method to reexamine the test for any further usability problems that were missed during the live test.

6.1.2 Debriefing - System Usability Scale

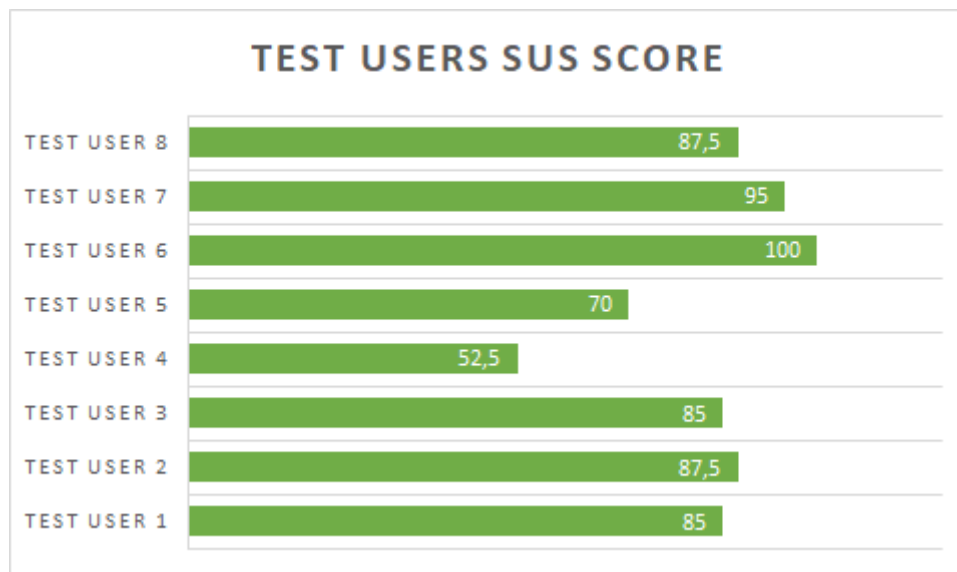


Figure 6.4. Test Users SUS score of the system

The above table shows the different SUS scores from each test user. There is an outlier with test user 4's SUS score compared to the others. This is not surprising as the test user was the one who sustained most errors during the test, so it would only be natural that it is reflected on the SUS score. The SUS score avg can be calculated by dividing the total sum of all the SUS scores by the number of SUS answers.

$$\text{avg sus score} = 82,81$$

According to those calculations, our system has a SUS score of 82,81, which is considered to be an excellent score for our system. Furthermore, item 4 and 10 on the questionnaire supports the learnability segment of usability. In 7 out of 8 responses on the SUS, we have a predominantly tendency towards a positive response on those items and, thus, can derive that the system was intuitive and easy to learn for the test users. In the one case of test user 4, he left the items to be neutral, so even though the test user experienced issues, it did not overshadow the system's ease of use and intuitiveness (<https://measuringu.com/sus/>).

Test users	SUS score
Test user 1	85
Test user 2	87,5
Test user 3	85
Test user 4	52,5
Test user 5	70
Test user 6	100
Test user 7	95
Test user 8	87,5
Total sum	662,5
Avg SUS score	82,81
Std Deviation	14,05555
Significans	0,05
Population	8
Confidens interval	9,74

Figure 6.5. Statistics on the SUS scores

Some further statistics were run on the data from the SUS score. With a 95% confidence, we expect the results for the SUS score to be the same for 95 times if 100 were done, which means that in 5 cases, the values should be beyond the interval. Moreover, the confidence interval shows us that with a 95% confidence that the SUS score should be in the parameter of $82,81 \pm 9,74$.

6.1.3 Usability Problems

Task 3 - Register a user

All test users had issues with password requirements. They were accustomed to the requirements of mixing small and capital letters and using a number, but the requirement that the password should contain at least one non-alphanumeric character seemed to cause trouble for most test users. This usability problem is categorized as critical since 8 out of 8 test users had trouble with the password requirements. A fix to this issue would be to lessen the requirements. We could also make it more clear before the users enter their desired password, what the requirements are.

Listing error occurred when there was no description

When the test user tried to create a listing, the test user left out the description box. This caused an unforeseen bug with the system, in which it did not create the listing. Furthermore, the system did not give the test user any information as to why it would not create the listing. The bug occurred due to some changes made to the create listing function after the pilot test. The problem should be resolved with some further debugging of the system. The problem is categorized as severe for the system and should be fixed before going live.

This error was corrected through code that made it a requirement to write a description for the book. After further internal testing the error did not occur again.

Nginx error & error with multiple listings

One user experienced an Nginx error when uploading a picture to the website. The Nginx error claimed that the entity was too large, meaning the image's file size was too big. Before the test, we had increased the file size limit for uploads. Another user uploaded the same image file without getting this error message. The user who got the Nginx error, also experienced having multiple listings, even though they only created one. We suspect that the listing was made even though they got the Nginx error, but the listing would not have an image. Moreover, we suspect Nginx error to be browser-related. Test user 4 was the only one to occur these errors and the only one to use another browser than the Google Chrome browser, which was Microsoft Edge. These errors are considered catastrophic for the system and need to be fixed before the system launches. In the future, we can try to test the system on multiple browsers.

When trying to replicate the error, it wasn't possible for the dev team using several different browsers. This makes it difficult to fix. It also leads to believe that the error may not have been browser related. The server settings are set to be able to handle images of greater sizes than the one provided to the test user.

Could not find seller information

In task 4, some of the test users had trouble understanding that they had not found the seller's information. On the listing page, there is some information about the seller, e.g., name and university. The test users often took this as the required information. The test users had to find all the information about the seller. This was not specified in the task and maybe what lead to this problem occurring. It may also be unclear that there exists additional information of the seller from the listing page. This could be exhibited more clearly on the listing page. Therefore, this usability problem is categorized as a cosmetic.

Other notable outliers and problems in usability test

The outlier in task 2 for test user 4 occurred because the test user was first trying to scroll through all listings before deciding on using the search function.

We noticed that many users were not using the search function on the site, and instead opted to scroll to find their desired listing. It worked for them because they knew that the listing was on the site, but this is not an ideal solution for the users. We suspect that the search function is not clear enough for the users to notice, and they may have went passed without even knowing it exists.

We could make the search function more transparent so that users instantly notice it when they access the site. We categorize this as a cosmetic error.

In task 3, as mentioned above, the test users faced issues with the password requirements. This was the same for test user 8 as he had trouble understanding the description of password requirements.

Another outlier in task 5 for test user 8 occurred because test user 8 was the only test user who manually wrote the info given in the task. The other test users used the copy-paste function.

In task 6, there were two outliers for test user 4 and 8. Test user 4 experienced the Nginx bug

that created multiple listings, confusing the test used for a couple of seconds before proceeding with the given task. Test user 8 wanted to go to their listing, but instead of using the built-in function, where the user could go to their profile page and then access their listing, the user opted to clicked on a listing which was identical to one they had made but was not made by them.

In task 7, the test user 4 misspelled the book's name when using the search function. This made the test user double-check the name of the book and the spelling.

Lastly, in task 9, both test user 1 2 spent time looking for an appropriate picture to upload as a profile picture.

6.1.4 Problems with the usability test

The test lacked an experienced Usability specialist, which most likely would have returned more found usability problems if conducted by this specialist. It would also have been ideal to use a proper usability laboratory instead of using the Discord service to conduct the tests. With a proper usability laboratory, we could have given the test users a physical book, which they would have had to get the information from. The would represent the use case a lot better than just being given some information by us. In a proper usability laboratory, we could also have had the necessary image files for the uploads already on the test computer, instead of relying on the user downloading some images which we provided them.

On further review of the videos of the testing, we noticed that the usability moderator was, in some cases, too quick to call the completion of the tasks and also at some points, helped the test user too quickly. If further testing is to occur, that is something to have in mind and make sure it will not happen again.

The test users chosen for the usability test consisted of friends and acquaintances, and this might have skewed the data towards a positive scoring in the SUS as the test users were biased.

Conclusion 7

Through our own experience as students at Aalborg University, we have encountered the trouble of purchasing books first hand. Not only can it be a financial burden for students who buy new books for each semester but have them thrown out or stored with no value. This leads us to the problem statement:

“How can we create an IT-system, which aims to help students buy and sell used books?”

Based on our system definition, we were able to analyze the problem-domain, which resulted in a coherent model, including classes, structure, and behavior. The system definition was also used to analyze the application domain, leading to a list of requirements based on the usage, interfaces, and functions found. The resulting analysis document acted as the foundation for the architectural design of the components, which then is used in the final component design. Thus, we were able to design a system based on the conventions of the object-oriented design and analysis while using sketches and wireframes to visualize the design.

We have made a fully functional website using the ASP.NET framework and the object-oriented programming language C#. Furthermore, the system uses a MySQL database to store data regarding listings, users, ratings, and books.

The system fulfills all the must-have requirements from the MoSCoW analysis. Unibook has the functionality to register users, let them create listings, and rate other users with whom they have dealt. The system also lets the user edit their listing or user profile, should there be typos or other errors in their input. Each user has their login and ID, which ties them to the listings that they have created. Requirements #10, #11, and #12 will not be implemented in this development, as they are time-consuming and not fundamental to the system at this point. They would be an excellent starting point for a second iteration of the system.

We conducted a usability test in which we found various usability errors, which should be solved before launch. Furthermore, we found that various users did not use the built-in search function as was intended. We concluded that it might be a good idea to make the search bar more apparent to the user.

During the debriefing using the system usability scale, we learned that the Unibook was easy to learn and intuitive for most of the test users.

Finally, it can be concluded that a fully working web-application has been developed and deployed. The system provides a platform for students to buy and sell books, which was the goal we set out to solve. The system fulfills all requirements, with a few usability problems. The system fulfills all requirements, with a few usability problems.

Discussion 8

During this project, some choices, which were thought to be the best, were made. This naturally led to some options being discarded. These will be discussed in the following section. Not only the choices that were discarded but also what they could have contributed to the project. These are valuable considerations to do, as it could improve further projects.

8.1 System Analysis and Design

For system analysis and design, we decided to not go into processes. We chose not to since we have not had any lecturing on this topic. Moreover, we decided that this would not contribute much substantial material for the project.

The choice of starting with the problem-domain analysis rather than the application-domain gave the project a more context-based development. Had the application-domain analysis been chosen as the former, the context would have been adjusted to the technology. This could have given the system a different outlook.

8.2 Implementation

Regarding the implementation, we decided to use some concepts that were outside of the curriculum. The use of entity framework, migrations, and the use of ASP.NET, was outside of the curriculum and were recommended to us by another student. Although these concepts were a great idea to use and made some processes more manageable, they also caused some hurdles in the implementation, since we had to learn how to use them.

As mentioned in the report, we chose to use MySQL rather than Microsoft SQLServer for our database, due to our ubuntu server's specifications. Given the fact that we used Microsoft Visual Studio as the IDE for development, which cooperates internally with the SQLServer, this could have been a good solution. However, it is hard to say how much it would have contributed to the efficiency of the development, as none of the project members had any experience with Microsoft SQLServer.

8.2.1 Security

Security was not a significant concern for us when we decided to make this system. This is also reflected in the final product. We do not have any security measures implemented besides the standard ones that are implemented with Microsoft Identity. Identity covers the users and their personal information and is responsible for hashing user passwords. We have looked into some areas of our model, specifically the use of addresses. It might not be a great idea to store user's

addresses in the system and making it available for other users to see. It was a design decision that allowed the users to discard listings that were geographically out of their reach.

8.3 Evaluation

Heuristic evaluation is a method to gain feedback quickly and relatively cheap, but it is only as good as the person we get to do it. Due to time constraints and a non-existent budget, the candidates available were deemed unqualified and inefficient as they would most likely consist of fellow students. Moreover, for the same reason, usability testing was deemed a better approach to evaluate the system. With eight novice users that suited our demographic, we could test how the system would be perceived regarding ease to learn, efficiency, and how fast the system would be for the users to use.

8.4 Covid-19

This project was developed during the global pandemic of the coronavirus. This meant that the government forced regulations on the university. This also meant that the members of the project group and our cooperation was done solely digitally. Furthermore, supervision was also performed digitally, which made communication a lot harder. It is hard to say precisely how and where this impacted the project. Indeed, it forced the members and supervisor to work under unorthodox conditions, which may have lead to inefficient communication, both within the group and with the supervisor. It also impacted the evaluation, as this had to be done digitally as well.

Bibliography

- [1] Bo Dalholm Lars Mathiassen. *Computers in context*. Wiley-Blackwell, 1993.
- [2] Lars Mathiassen et al. *Object Oriented Analysis Analysis*. Metodica ApS, 2018.
- [3] David R. Benyon. *Designing Interactive Systems*. Pearson, 2014.
- [4] Aalborg University. Aau leverer verdens bedste bæredygtige undervisning. 2020.
- [5] Robin Nixon. *Learning PHP, MySQL, JavaScript, CSS HTML5 4th Edition*. O'Reilly, 2014.
- [6] RB Whitaker. *The C Player's Guide*. Starbound Software, 2017.
- [7] Bootstrap. Layout - overview, . URL <https://getbootstrap.com/docs/4.3/layout/overview/>. (Sidst besøgt d. 26/05-2020).
- [8] Bootstrap. Overview, . URL <https://getbootstrap.com/docs/4.3/about/overview/>. (Sidst besøgt d. 23/05-2019).
- [9] Lynn Beighley. *Head First SQL– A Learner's Guide*. O'Reilly Media; 1 edition, 2007.
- [10] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann, 1993.

Listings

5.1	CSS example	38
5.2	Layout page	39
5.3	LoginPartial page	40
5.4	Register form	41
5.5	Register model properties	42
5.6	Display contact information	43
5.7	Create Listing	44
5.8	Program Class	53
5.9	Startup class	54
5.10	Search bar on Index page	55
5.11	EF Core ORM Query	55
5.12	IndexModel constructor with DbContext and ILogger	56
5.13	Foreach-loop displaying listings	57
5.14	Listing creation model	58
5.15	User creation	59
5.16	OnGet method for edit listing	60
5.17	If statements in the user model	61
5.18	Edit user OnPost method	62
5.19	Rate feature	62
5.20	OnPost method for rate feature	63
5.21	Rating p	63