

**Bachelor Project**



**Czech  
Technical  
University  
in Prague**

**F3**

**Faculty of Electrical Engineering  
Department of Software engineering and technology**

## **Prototype of the new information system of the CTU dormitory rental**

**Dastan Sadyraliyev**

**Supervisor: Ing. Karel Frajták, Ph.D.**

**Field of study: Software engineering and technology**

**Subfield: Enterprise systems**

**May 2024**



## I. Personal and study details

Student's name: **Sadyraliyev Dastan**

Personal ID number: **497919**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Software Engineering and Technology**

## II. Bachelor's thesis details

Bachelor's thesis title in English:

**Prototype of the new information system of the CTU dormitory rental**

Bachelor's thesis title in Czech:

**Prototyp nového informačního systému pro pronájem kolejí VUT**

Guidelines:

The rental of CTU dormitories is no longer sufficient to meet the needs of students, new students are often facing the problem of discovering where and how to reserve a certain item or who to contact. The rental office has no information system, everything is handled manually in the old-fashioned way.

The IS will cover a number of areas and a system built on microservices seems like a modern solution.

Gather the requirements on new IS with an emphasis on extensibility, performance, reliability, ease of use and scalability.

Create detailed wireframes for the new solution and validate the models.

Implement the core microservices and test the prototype.

Bibliography / sources:

ABGAZ, Yalemisew, et al. Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. IEEE Transactions on Software Engineering, 2023.

VELEPUCHA, Victor; FLORES, Pamela. A survey on microservices architecture: Principles, patterns and migration challenges. IEEE Access, 2023.

LI, Shanshan, et al. Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. Information and software technology, 2021, 131: 106449.

Name and workplace of bachelor's thesis supervisor:

**Ing. Karel Frajták, Ph.D. System Testing IntelLigent Lab FEE**

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **01.02.2024**

Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

\_\_\_\_\_  
Ing. Karel Frajták, Ph.D.  
Supervisor's signature

\_\_\_\_\_  
Head of department's signature

\_\_\_\_\_  
prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

## III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature



## Acknowledgements

I would like to express special gratitude to the supervisor of the bachelor's project Ing. Karel Frajták, Ph.D. for his desire and help throughout the project.

And I would like to thank the semester project supervisor Ing. Václav Smítka.

I would also like to express my gratitude to my entire family for their great support, and to my friends who took part in testing wireframes. Thank you for your time and for providing me with feedback.

## Declaration

I declare that I have prepared the submitted work independently and that I have listed all the information sources used in accordance with the methodological guidelines on compliance with the principles in the preparation of University theses. Prague. 05 March 2024.

## Abstract

The bachelor's thesis is devoted to the design of an information system for the equipment rental system in the Strahov dormitory. The information system is called Rentopia. Its main task is to facilitate the rental of necessary equipment by users through a user-friendly interface. This work includes analysis of interviews and comparison of Eureka models and alternatives, design and implementation of the system prototypes and finally, testing the achieved results.

**Keywords:** Information system, Prototypes, Detailed Wireframes, Microservices, Service discovery, Eureka

**Supervisor:** Ing. Karel Frajták, Ph.D.

## Abstrakt

Bakalářská práce je věnována návrhu informačního systému pro systém půjčovny vybavení na koleji Strahov. Informační systém se nazývá Rentopia. Jeho hlavním úkolem je usnadnit uživatelům pronájem potřebného vybavení prostřednictvím uživatelsky přívětivého rozhraní. Tato práce zahrnuje analýzu rozhovorů a porovnání modelů a alternativ Eureka, návrh a implementaci prototypů systému a nakonec testování dosažených výsledků.

**Klíčová slova:** Informační systém, Prototypy, Detailní drátové modely, Mikroslužby, Service discovery, Eureka

**Překlad názvu:** Prototyp nového informačního systému půjčovny kolejí ČVUT.

# Contents

<b>1 Introduction</b>	<b>1</b>		
1.1 Motivation	1		
1.2 Objectives	1		
<b>2 Analysis of the current state and definition of requirements for new solutions</b>	<b>3</b>		
2.1 Analysis of current state	3		
2.1.1 Analysis of existing web-service	3		
2.1.2 Analysis of current rental system	4		
2.1.3 Conclusion	5		
2.2 AS-IS diagram	6		
2.3 Requirements definition	6		
2.3.1 Functional requirements	6		
2.3.2 Non-functional requirements	7		
2.4 Prioritization	7		
2.4.1 Definition of stakeholders	7		
2.4.2 Functional requirements	8		
2.4.3 Non-functional requirements	9		
2.4.4 Conclusion	10		
2.5 Back-end	10		
2.6 Data tier	11		
<b>3 Creating a detailed wireframes for a new solution</b>	<b>13</b>		
3.1 Definition of “Rentopia” scope	13		
3.2 To-Be diagram	13		
3.3 UML class diagram	14		
3.4 Use-Case diagram	15		
3.5 Wireframing tool selection	19		
3.5.1 Conclusion	19		
3.6 Annotation of wireframe	20		
<b>4 Verification of the quality of the wireframes</b>	<b>27</b>		
4.1 Feedback	27		
4.2 Documentation	28		
4.3 Review	28		
<b>5 Microservices architecture</b>	<b>29</b>		
5.1 Introduction to microservices	29		
5.2 Patterns for microservices	29		
5.3 Challenges	30		
5.4 Visualizing microservice architecture	31		
<b>6 Service discovery</b>	<b>33</b>		
6.1 Understanding service discovery	33		
6.1.1 The benefits of service discovery	33		
6.2 Define of evaluation criterias	34		
6.3 Eureka	34		
6.4 Comparative Analysis	36		
6.5 Decision	37		
<b>7 Implementation of prototypes</b>	<b>39</b>		
7.1 Structure of project	39		
7.2 Critical functionality	40		
7.3 Sequence diagram	41		
7.4 Graphical prototypes	42		
7.5 Showcase interactions	43		
7.6 Testing	45		
7.6.1 JUnit tests of the services	46		
7.6.2 Eureka service discovery testing	46		
7.6.3 API Gateway testing	47		
7.6.4 Manual API testing with postman	47		
7.6.5 Examples of requests	48		
7.6.6 Responses	49		
7.7 Review	50		
7.8 Future developments	50		
<b>8 Conclusion</b>	<b>51</b>		
<b>A List of abbreviations</b>	<b>53</b>		
<b>B List of links</b>	<b>55</b>		
<b>C Bibliography</b>	<b>57</b>		

## Figures

<b>2.1</b>	Grill center web-service. . . . .	3
<b>2.2</b>	Grill center reservation page. . . . .	4
<b>2.3</b>	AS-IS diagram. . . . .	6
<b>3.1</b>	To-Be diagram. . . . .	13
<b>3.2</b>	UML class diagram. . . . .	14
<b>3.3</b>	Use-Case diagram. . . . .	15
<b>3.4</b>	Wireframe 3.4. Registration form. . . . .	20
<b>3.5</b>	Wireframe 3.5. Authorization form. . . . .	21
<b>3.6</b>	Wireframe 3.6. Equipment view. . . . .	22
<b>3.7</b>	Wireframe 3.8. Equipment management. . . . .	23
<b>3.8</b>	Wireframe 3.8. Equipment management. . . . .	24
<b>3.9</b>	Wireframe 3.9. Equipment reservation for users. . . . .	25
<b>5.1</b>	Microservice Architecture. . . . .	31
<b>7.1</b>	Structure of project. . . . .	40
<b>7.2</b>	Structure of project (part 2) . . . . .	40
<b>7.3</b>	Structure of project (part 3) . . . . .	40
<b>7.4</b>	Sequence diagram: Equipment reservation. . . . .	41
<b>7.5</b>	Eureka technology. . . . .	42
<b>7.6</b>	First scenario prototype. . . . .	43
<b>7.7</b>	Second scenario prototype. . . . .	44
<b>7.8</b>	Successful eureka client registration output. . . . .	46
<b>7.9</b>	Eureka dashboard: Registered instances. . . . .	46
<b>7.10</b>	Eureka dashboard: One of the instances is DOWN. . . . .	47
<b>7.11</b>	API gateway routing. . . . .	47
<b>7.12</b>	Account-management-service response. . . . .	49
<b>7.13</b>	Equipment-management-service response. . . . .	49
<b>7.14</b>	Equipment-reservation-service response. . . . .	49

## Tables

<b>6.1</b>	Score voting results. . . . .	36
------------	-------------------------------	----



# Chapter 1

## Introduction

### 1.1 Motivation

Strahov dormitories are currently facing significant difficulties in managing the rental service. The main reason for this is the lack of a centralized, efficient and transparent system that allows users, new and old residents of dormitories, to navigate the rental service. The lack of an information system leads to high inefficiency. This inefficiency not only leads to confusion and frustration among students, but also hinders the effective management of dormitory resources. The growing field of microservices architecture represents a promising solution to such problems. Using modern technological advances, it is possible to change the dormitory rental system, making it more accessible, responsive and convenient for both students and the administration.

### 1.2 Objectives

1. Analysis of the current state and definition of requirements for new solutions.
2. Creating a detailed wireframes for a new solution with an emphasis on responsiveness and transparency.
3. Verification of the quality of the wireframes
4. Analysis of Eureka with alternatives in order to identify the best choice in the context of the information system. You can consider criteria such as performance, reliability, ease of use and scalability.
5. Prototyping microservices.
6. Prototype testing.



## Chapter 2

# Analysis of the current state and definition of requirements for new solutions

### 2.1 Analysis of current state

In this section, I want to analyze the current situation related to the equipment rental system at the Strahov dormitory. The analysis includes analysis of the web service and the rental service itself.

#### 2.1.1 Analysis of existing web-service

On one of the 11 subsites of the dorm's main website user can find link to the service Grill Center. With the help of this service, users have the opportunity to reserve a portable grill and a grill area on the territory of the dorm.

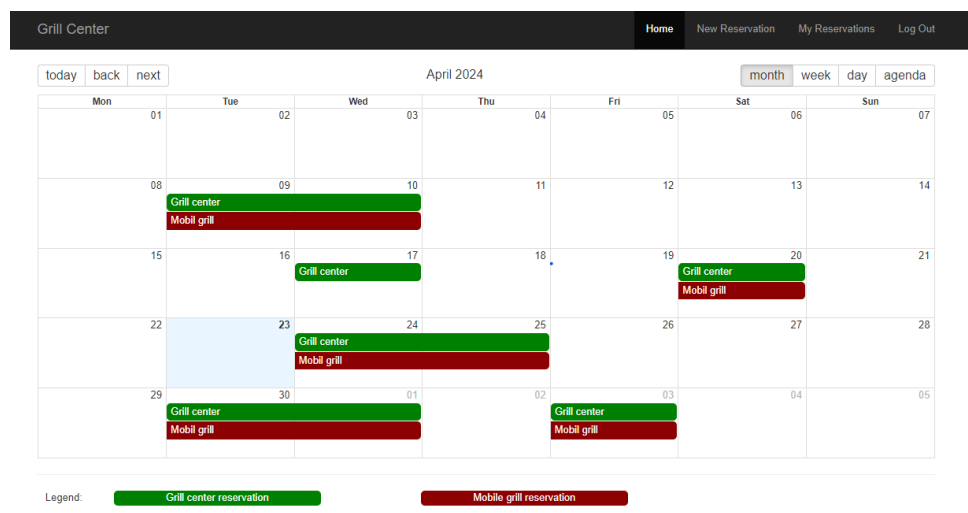


Figure 2.1: Grill center web-service.

**Figure 2.2:** Grill center reservation page.

This web service provides only a basic web page for viewing existing reservations and creating your own reservation. But the user interface is not user-friendly and does not provide the necessary information. I want to provide more detailed information about the disadvantages of this service as one of the quotes from the administrator.

### **Administrator's feedback.**

”The current web service of the grill center only includes booking a mobile grill. There is no information system for the rental system in our dormitory. We would be glad to have a modern information system with a user-friendly interface. The information system should simplify the rental of equipment for residents and administrators. The main thing is that the system must meet all the requirements of the residents of the Strahov dormitory.”[1]

### **2.1.2 Analysis of current rental system**

One of the 11 subsites of the dorm’s main website is where the information about the rental system is found. According to the information that is currently available, or more accurately the official web-page of the dorm, the system for the Strahov rental system appears to be primarily manual and minimal in scope.[2]

#### **This is how the situation is currently described:**

1. **Restricted Online Data.** Only the most basic details about the rental service are available on the dorm’s official website. It gives information regarding the rental procedure and notes that rental equipment is available on the third level south.
2. **Manual Process.** The renting procedure appears to be primarily manual. Renting equipment in person need residents or students to come

to the third floor south location. The website does not include an online reservation mechanism.

3. **Equipment Types.** The information mentions that different types of equipment are available for rent. For example: tools, sports equipment, and other equipment. However, the details about the range of available equipment, kinds, quantities, status and conditions are not provided.
4. **Return Conditions.** It is mentioned that rented equipment must be returned in the same condition as when borrowed and that any defects should be reported to the administrator. However, there is no information about how these returns and reports are processed, which indicates the potential loss of a formalized system.
5. **Recording Borrowed Items.** There is a reference to recording the "agreed return time" in a notebook. This suggests that the tracking of borrowed items rely on manual recording, which could be reason to errors.
6. **Bans.** Information about penalties for failure to return rented items is also available here. Vacuum cleaners have a 7-day ban period, drills, and grills have a 5-day ban period, while all other items have a 3-day ban period. This information is recorded manually, which can lead to errors and issues in the rental management system. There is no information about fines for damage to rented items, which leaves students unsure of the potential consequences.
7. **Communication with administrators.** Information about the administrators and their e-mails for the rental room is available on the web-page, but there are no other kinds of communication. In other words, if a student urgently needs any rented equipment, he will not be able to contact the administrators quickly, the only way is to knock on their door, but there is a risk that the administrator will not be in his room. The chance that a student will be able to rent the equipment he needs at a certain hour is 50 percent. At the same time, he will have to spend time and effort searching for an available administrator. This leads to a decrease in the desire to use this service among students.

### 2.1.3 Conclusion

In general, the current state of the information system of the dormitory rental service seems does not correspond to digitalization and automation. It is based on manual processes, and there is a limited amount of online information available to students or residents to access and manage equipment rentals. There is potential for improvement through the introduction of a more organized and efficient equipment rental information system, inventory tracking and ensuring better interaction with users through a convenient online platform. This is an improvement I want to provide. My solution is

an information system for a rental room. Name of information system is "Rentopia".

## 2.2 AS-IS diagram

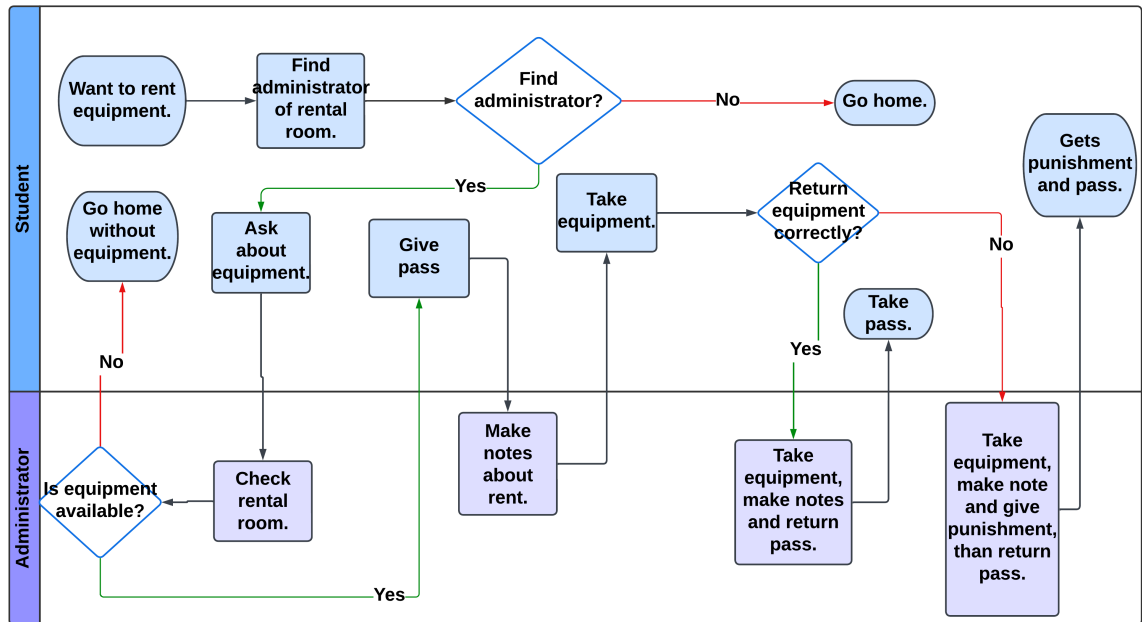


Figure 2.3: AS-IS diagram.

This AS-IS diagram represents the current state of the process: Equipment Rental. This diagram shows that the current process is filled with different conditions, which with a 50 percent chance (based on the statistics) lead to failure and to an undesirable result for users.

## 2.3 Requirements definition

This section will be devoted to defining functional and non-functional requirements based on a questionnaire. In the attachments you can view the results of the survey in an Excel table.

### 2.3.1 Functional requirements

1. Allow user registration.
2. Support equipment reservation.
3. Display equipment availability and status.
4. Enable users to contact administrators.

5. Provide overall information about the room and its rules.
6. Offer of new equipments for the rental room.
7. Implement CRUD <sup>1</sup> operations for equipment management, including the ability to add, edit, and delete equipment.
8. Allow administrators to contact with users.
9. Add time for meeting with user for their reservation.
10. Implement a system of imposing fines on users with conditions and consequences.

### 2.3.2 Non-functional requirements

1. Data security (encryption, secure access).
2. Speed and performance (fast application response).
3. Accessibility (responsive design, accessibility for users with special needs and user-friendly interface).
4. Scalability (expandable for more users).
5. Testing and debugging (easy testing and error detection).

## 2.4 Prioritization

This section includes the definition of stakeholders and functional ones; non-functional requirements.

### 2.4.1 Definition of stakeholders

- Administrator responsible for Rental Room - This stakeholder is likely concerned with the effective management of the rental process, inventory, and overall system administration. Their input may be crucial for defining system features related to rental operations, rules, and fines.
- Students and Non-students (End-users) - End-users are a critical stakeholder group, because they directly interact with the system. In my scenario, this applies to both students and non-students who use the equipment rental service. Collecting their requirements and feedback is necessary to create an user-friendly and efficient system.

---

<sup>1</sup>CRUD: "Create, Read, Update, and Delete (CRUD) are the four basic functions that models of database should be able to do, at most."

- Developers (In this case, myself) - My understanding of the technical aspects, limitations, and opportunities is crucial. Additionally, as the developer, I'll have insights into the system's architecture, design decisions, and implementation challenges.

To categorize requirements I am going to use MoSCoW method.

“MoSCoW prioritization, also known as the MoSCoW method or MoSCoW analysis, is a popular prioritization technique for managing requirements. The acronym MoSCoW represents four categories of initiatives: must-have, should-have, could-have, and won't-have, or will not have right now.”[3]

MoSCoW method to categorize requirements:

- **Must-haves (M)**: Requirement, which must be have for successful application.
- **Should-haves (S)**: Important requirement, but are not as needed as core (M) requirements
- **Could-haves (C)**: Requirements, which will have a small impact if left out.
- **Won't-haves (W)**: Requirement that are not high priority for this specific time.

## ■ 2.4.2 Functional requirements

1. Facilitate user registration (M):
  - a. This requirement marked as a "Must-have" since user registration is fundamental for user engagement and use of the system.
2. Support equipment reservation (M):
  - a. Marking this as a "Must-have" aligns with the core functionality of the system, ensuring that users can efficiently reserve equipment.
3. Display equipment availability and status (M):
  - a. This is correctly categorized as a "Must-have" to provide users with important information for making reservations.
4. Enable users to contact administrators (S):
  - a. Marking this as a "Should-have" suggests that while it's important for users to contact administrators, it may not be as critical as the core functionalities.
5. Provide comprehensive information about the room and its rules (S or C):



- a. Depending on the specifics and priorities, this could be categorized either a "Should-have" or a "Could-have." If it's deemed crucial for user understanding and compliance, it might be a "Should-have."
6. Offer of new equipment for the rental room(W):
  - a. Marking this as a "Won't-haves" indicates that for current time this is not a necessary technology, but it may be added in the future.
7. Define and implement CRUD operations for equipment Management (M):
  - a. Marking this as a "Must-have" is fitting since it forms the basis of equipment management, ensuring necessary functionalities for system operation.
8. Allow administrators to initiate contact with users (C):
  - a. Marking this as a "Could-have" indicates that while it would be desirable for administrators to contact users, it's not necessary for the initial system release.
9. Add time for meeting with user for their reservation (S):
  - a. Marking this as a "Should-haves" indicates that adding this function will be good decision, but can be automatically by the system or manually by admins.
10. Define and implement a system for imposing fines on users (C):
  - a. Categorizing this as a "Could-have" aligns with the idea that while it adds value to the system, it's not a critical component for the initial system release.

### 2.4.3 Non-functional requirements

1. Data security (Encryption, Secure access)(M):
  - a. Positive aspects:
    - (i) Identifies the need for encryption and secure access.
  - b. Considerations:
    - (i) Specify encryption standards and secure access mechanisms.
2. Speed and performance(S):
  - a. Positive aspects:
    - (i) Clearly create the expectation for a responsive application.
  - b. Considerations:
    - (i) Define specific performance metrics or response time targets.

3. Accessibility(M):

a. Positive Aspects:

- (i) Recognizes the importance of an accessible design and user-friendly interface.

b. Considerations:

- (i) Specify accessibility standards to be followed.

4. Scalability(C):

a. Positive aspects:

- (i) Acknowledges the need for scalability in terms of technology and user base.

b. Considerations:

- (i) Detail how the application will adapt to new technologies.

5. Testing and debugging(M):

a. Positive aspects:

- (i) Highlights the need for a test and quick resolution of problems.

b. Considerations:

- (i) Specify the testing methodologies to be employed (e.g., unit testing, integration testing).

#### 2.4.4 Conclusion

These functional and non-functional requirements provide a solid foundation for developing a reliable and user-friendly application.

## 2.5 Back-end

**Java** is a multi-platform object-oriented programming language. It supports web applications, smartphone operating systems and many well-known programs. Java is currently the most popular programming language for application developers. In my opinion, Java is the best choice for creating the Rentopia information system, because it includes all the necessary technologies and there are a huge number of useful frameworks.[4]

**Spring Boot** is an open source Java-based framework. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.[5]

My preference goes to **Java** and **Spring Boot**, because I have much more experience working with the Java language and Spring framework.

## ■ 2.6 Data tier

Data-tier will be based on relational database PostgreSQL and non-relational database MongoDB.

**PostgreSQL** is a powerful, open source object-relational database system that uses and extends the SQL <sup>2</sup> language combined with many features that safely store and scale the most complicated data workloads.[6]

**MongoDB** is a non-relational document database that provides support for JSON <sup>3</sup> -like storage. In my case, for my Rentopia app, MongoDB will be used to store data about equipment. Thanks to the JSON format, it will be easy and convenient to work with data that describes the equipment.[7]

I prefer PostgreSQL and MongoDB because I have a lot of experience with PostgreSQL and as I mentioned earlier, the JSON format in MongoDB will be very convenient for storing hardware information.

---

<sup>2</sup>SQL: "Structured Query Language"

<sup>3</sup>JSON: "JavaScript Object Notation"



## Chapter 3

# Creating a detailed wireframes for a new solution

### 3.1 Definition of "Rentopia" scope

The main goal of my work is to simplify and improve the use of the equipment rent service in dormitories. My solution is an information system that will include all the functions and requirements identified through an analysis of a user questionnaire diagrams. The basis of the "Rentopia" application is a user-friendly interface that will be designed for any type of user and will separate functions for administrators and users.

### 3.2 To-Be diagram

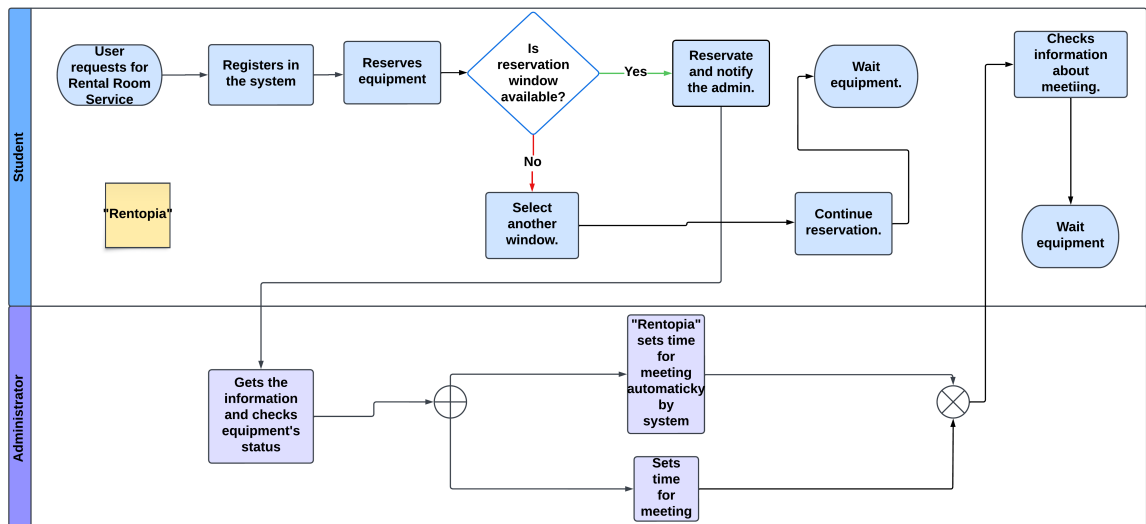
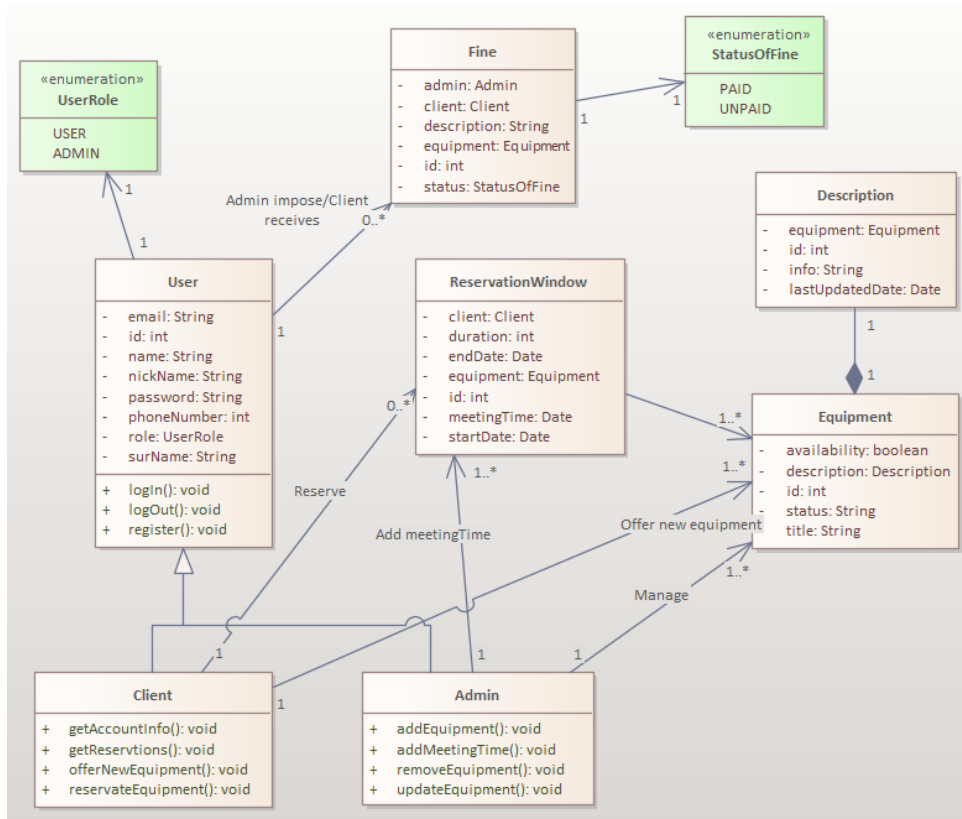


Figure 3.1: To-Be diagram.

This TO-BE diagram represents the future state of the process using the Rentopia information system: Equipment Rental. The chance of achieving

the desired result compared to the AS-IS schedule is 95 percent.

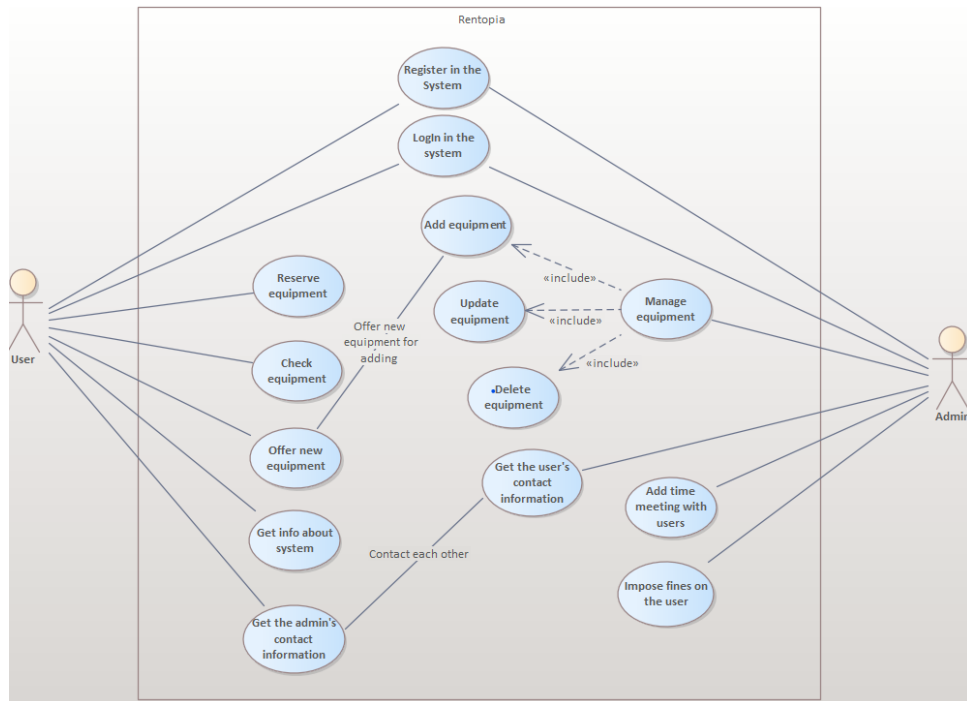
### 3.3 UML class diagram



**Figure 3.2:** UML class diagram.

This diagram is a UML class diagram that describes the entities of the future Rentopia system and the relationships between them.

## 3.4 Use-Case diagram



**Figure 3.3:** Use-Case diagram.

### Scenario 1: Registration in the System.

#### Actors:

- User
- System

#### Flow of Events:

- Step 1: User:** The user requests the registration function from the information system.
- Step 2: System:** The system provides the user with a registration form.
- Step 3: User:** The user fills out the form and registers in the system.
- Step 4: System:** The system provides comprehensive information to the user, offering an overview of its features, rules, functionalities, and navbar.

### Scenario 2: Log in and reservation equipment.

#### Actors:

- User
- System

**Flow of Events:**

- Step 1: User:** The user logs in.
- Step 2: System:** Upon successful verification, system provides an interface, presenting the relevant functionalities and information based on the roles and permissions of the logged-in user.
- Step 3: User:** The user requests for page with real-time status and availability of equipment through the system interface.
- Step 4: System:** The system provides page with equipment.
- Step 5: User:** The user selects equipment.
- Step 6: System:** The system provides page with equipment reservation.
- Step 7: User:** The user selects reservation window and reserves equipment.
- Step 8: System:** The system processes the reservation, updating the status of the reserved equipment.
- Step 9: User:** The user checks created reservation.

**Scenario 3: Equipment management.**

**Actors:**

- Admin
- System

**Flow of Events:**

- Step 1: Admin:** The admin registers in the system and logs in.
- Step 2: System:** The system provides comprehensive information to the admin, offering an overview of its features, rules, functionalities and navbar.
- Step 3: Admin:** The admin requests for page with equipment management.
- Step 4: System:** The system provides information and an interface for equipment management.
- Step 5: Admin:** The admin adds, removes equipment and updates information status and availability of equipment.
- Step 6: System:** The system processes administrator requests for equipment management.



**Scenario 4: Imposing fines on users.**

**Actors:**

- Admin
- User
- System

**Flow of Events for Admin:**

**Step 1: Admin:** The admin logs in.

**Step 2: System:** The system provides comprehensive information to the admin, offering an overview of its features, rules, functionalities and navbar.

**Step 3: Admin:** The admin requests FineSystem.

**Step 4: System:** The system provides interface for imposing fines on users.

**Step 5: Admin:** The admin imposes fines on users.

**Step 6: System:** The system processes administrator requests for imposing fines on users.

**Flow of Events for User:**

**Step 7: User:** The user logs in.

**Step 8: System:** The system provides comprehensive information to the user, offering an overview of its features, rules, functionalities and navbar.

**Step 9: User:** The user requests information about his account.

**Step 10: System:** The system provides interface with information about user's account.

**Step 11: User:** The user gets information about fines.

**Scenario 5: Getting contact information.**

**Actors:**

- Admin
- User
- System

**Flow of Events for Admin:**

**Step 1: Admin:** The admin logs in.

**Step 2: System:** The system provides comprehensive information to the admin, offering an overview of its features, rules, functionalities and navbar.

**Step 3: Admin:** The admin requests contact information of users.

**Step 4: System:** The system provides interface with contact information of users.

**Flow of Events for User:**

**Step 5: User:** The user logs in to the system.

**Step 6: System:** The system provides comprehensive information to the user, offering an overview of its features, rules, and functionalities.

**Step 7: User:** The user requests contact information of admin.

**Step 8: System:** The system provides interface with contact information of admins.

**Scenario 6: Adding time for meeting.**

**Actors:**

- Admin
- System

**Flow of Events:**

**Step 1: Admin:** The admin logs in.

**Step 2: System:** The system provides comprehensive information to the admin, offering an overview of its features, rules, functionalities and navbar.

**Step 3: Admin:** Upon successful reservation, the admin can add time for meeting, that will be shown for user in reservation window.

**Step 4: System:** The system processes request and updates interface of reservation window.

**Scenario 7: Offer of new equipment.**

**Actors:**

- Admin
- User
- System

**Flow of Events:**

**Step 1: User:** The user logs in.

**Step 2: System:** The system provides comprehensive information to the user, offering an overview of its features, rules, functionalities and navbar.

**Step 3: User:** The user decides to offer a new equipment.

**Step 4: System:** The system provides interface for offering a new equipment.

**Step 5: User:** The user offers a new equipment.

**Step 6: System:** The system processes request and updates interface for admin.

**Step 7: Admin:** The admin checks new offered equipment in equipment management and thinks about adding new equipment.

## 3.5 Wireframing tool selection

### Sketch.

Sketch is a vector graphics editor that must be installed locally on your computer. It changed the design world because it was created specifically for digital design, not for print, like Photoshop.[8]

### Figma.

**Why Figma?** There are many design apps available on the market right now that can be used to solve any kind of graphical ideas. But Figma is one of many designers favorite tools and is becoming more and more popular.[9] There are many good reasons for this.

- First, Figma allows designers and other teammates to work simultaneously in real time which takes the collaborative workflow to a whole new level.
- Second, Figma covers about everything you need to create a complex interface, from brainstorming and wireframing to prototyping and sharing assets.
- Finally, Figma is not only a design app but also a community and platform for sharing ideas and solutions.

### 3.5.1 Conclusion

Figma and Sketch are popular user interface development platforms where users can create user-friendly websites.

If users prefers offline working, using an extensive integration library to create a more personalized interface, then Sketch is a good choice.

If users need a free option, flexible collaboration and flexible vector management, then Figma may be the best solution.

In the case of creating frameworks for Rentopia, more flexible vector management is required. The best solution would be Figma.

**Create a wireframe.**

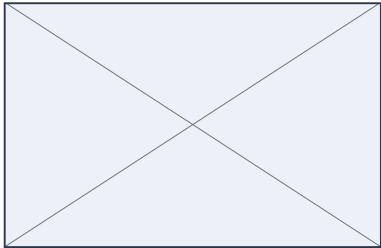
My goal is to create wireframes that can visually show my idea. My idea is to create an intuitive, efficient system with a transparent user-friendly interface. My task is also use the scenarios and use-cases that were described in Section 3.4.

By this link, you can transfer to Rentopia Prototypes, where you can by scenarios test prototypes. **Link to Figma prototype** <https://shorturl.at/talqY>

### 3.6 Annotation of wireframe

## Registration

Welcome to Rentopia!  
Please register in the system.



Write your email\*

Write your nickname\*

Write your name\*

Write your surname\*

Write you phone number\*

Write your password\*

Repeat your password\*

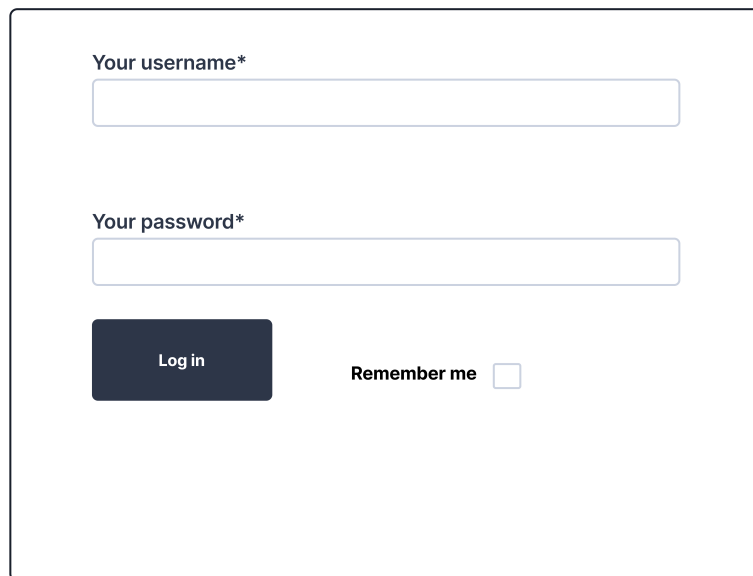
Register

**Figure 3.4:** Wireframe 3.4. Registration form.

**Annotation:**

Wireframe 3.4 shows the forms for registration of both users and administrators. At this stage, users enter their data, and the system processes this information to initiate the registration process. This wireframe meets the functional requirements, including registration.

## Log in



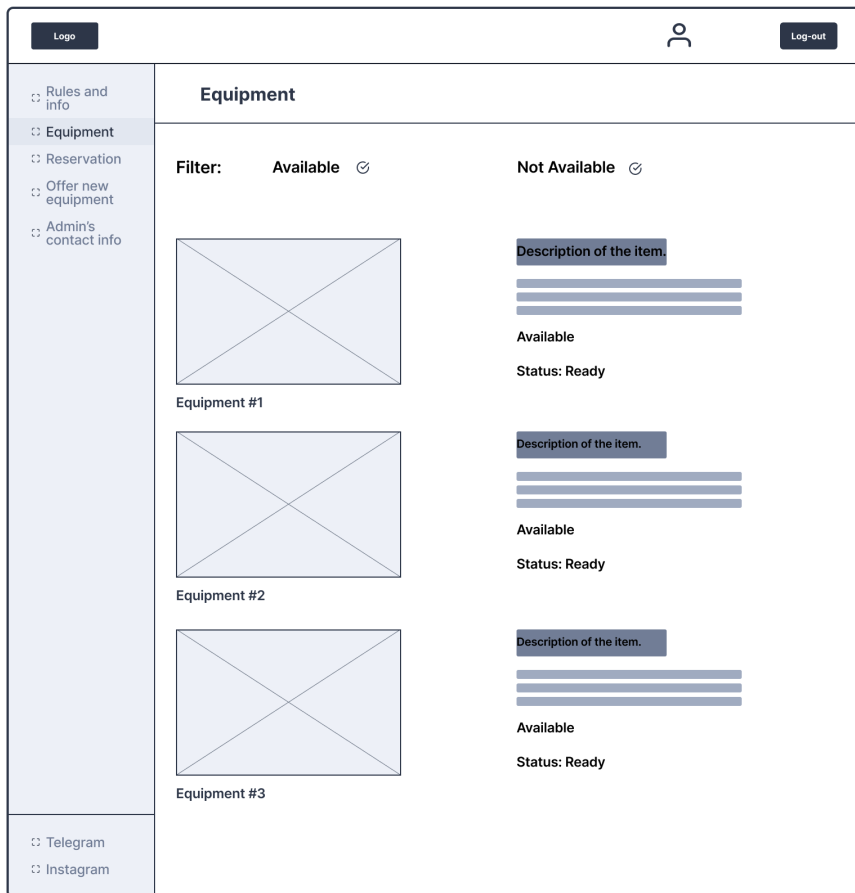
The wireframe shows a login form with the following elements:

- A title "Log in" centered at the top.
- A label "Your username\*" above a text input field.
- A label "Your password\*" above a text input field.
- A dark blue button labeled "Log in" positioned below the password field.
- A "Remember me" label followed by an unchecked checkbox, positioned to the right of the "Log in" button.

**Figure 3.5:** Wireframe 3.5. Authorization form.

**Annotation:**

Wireframe 3.5 shows the forms for authorization of both users and administrators. After successful registration, the system provides an authorization process for the corresponding user or administrator. This wireframe meets the functional requirements, including authorization.



**Figure 3.6:** Wireframe 3.6. Equipment view.

**Annotation:**

Wireframe 3.6 serves as a comprehensive repository of information belonging to equipment, including details about their availability and status.

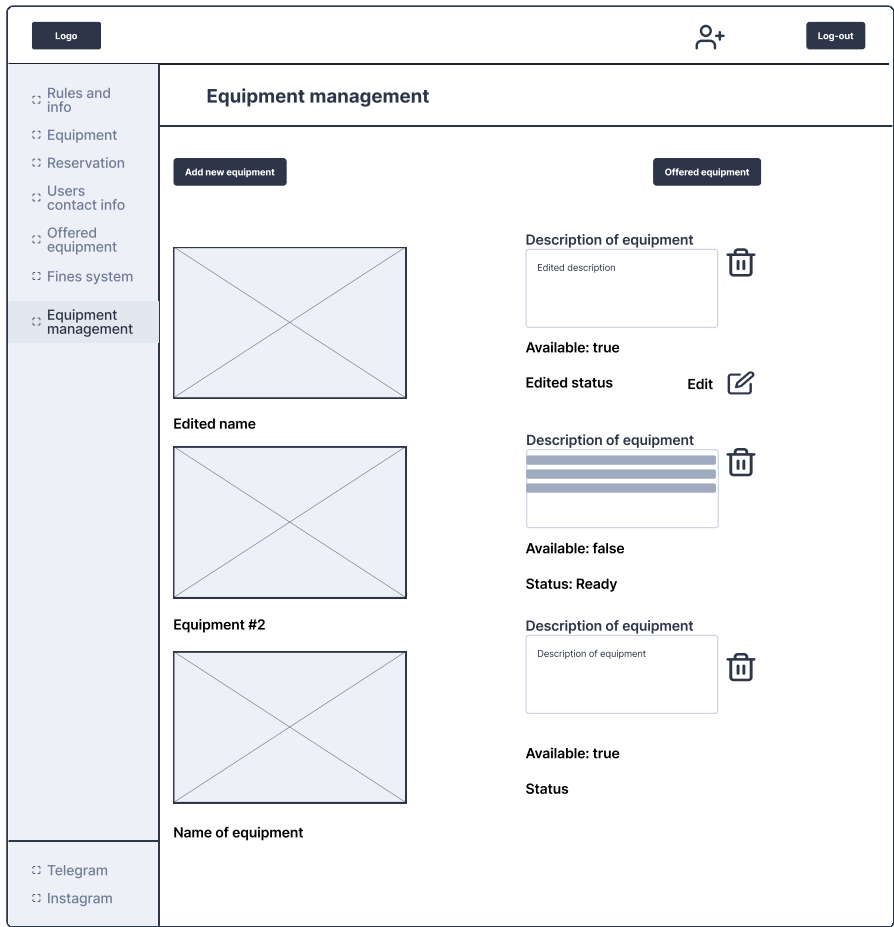


Figure 3.7: Wireframe 3.8. Equipment management.

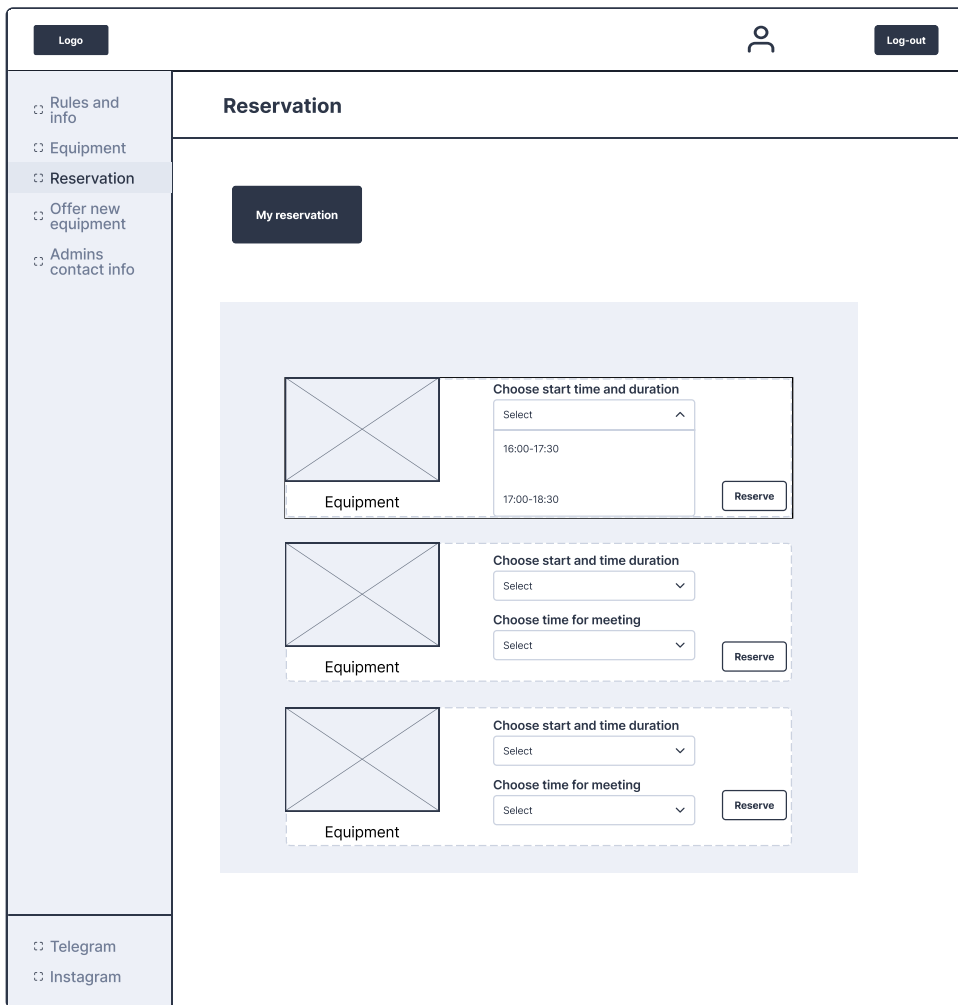
The image displays two wireframe screens for equipment management. The top screen, titled "Edit equipment", features three text input fields labeled "Name of equipment" (with placeholder "Edited name"), "Description of equipment" (with placeholder "Edited description"), and "Status of equipment" (with placeholder "Edited status"). Below these is a checkbox for "Availability" and a button with a photo icon labeled "Add photo". A "Save" button is positioned at the bottom right. The bottom screen, titled "Add your equipment", has three input fields: "Name of equipment" (placeholder "Value"), "Description of equipment" (placeholder "Value"), and "Status" (a dropdown menu with "Select" and a downward arrow). It also includes a checkbox for "Availability", an "Add photo" button, and an "Add" button at the bottom right.

**Figure 3.8:** Wireframe 3.8. Equipment management.

**Annotation:**

On the administrative side, the Wireframe 3.8 is carefully crafted for equipment management, incorporating all CRUD features in accordance with the specified requirements.





**Figure 3.9:** Wireframe 3.9. Equipment reservation for users.

**Annotation:**

Wireframe 3.9 is utilized for viewing available windows for reservation equipment. Users must enter different data required for the reservation.



## Chapter 4

# Verification of the quality of the wireframes

### 4.1 Feedback

Wireframes were shared with colleagues, friends and administrator who live in the Strahov dormitory. They gave their feedback.

#### **First resident's feedback.**

“The design of Reservation, Rules and Info, and Equipment pages is user-friendly, incorporating a thoughtful and functional minimalist approach. The use of a Single Page Application (SPA) enhances the overall user experience. Additionally, I'm impressed by the well-thought-out design and functionality, including easily accessible social media links. The Equipment Management page's commendable lo-fi design creates a unique and memorable visual impression. It's suggested to streamline functionality by consolidating the "Not Available" button with the "Available" button.”[10]

#### **Second resident's feedback.**

“I didn't like that the time selection for reservations is done using Options; it could have been implemented more intuitively, perhaps with a calendar interface similar to Google Calendar. However, I appreciate the ability to suggest personal inventory items. I would rename "Status" when modifying inventory. Overall, I really liked the intuitiveness and transparency of the interface. Fine-tuning minor details would make it an effective application that I would use when needed.”[11]

#### **Administrator's feedback.**

”As an administrator, I can say that the future information system will be very convenient and will facilitate the equipment rental service for both students and administrators. I liked the opportunity to specify the time of the administrator's meeting with a resident of



## Chapter 5

# Microservices architecture

### 5.1 Introduction to microservices

The Rentopia project is my first experience with microservice architecture. To point out the benefits and capabilities of this architecture, I will refer to the monolithic architecture for comparison.

Microservices architecture is a system design approach that breaks a complex system into multiple independent microservices.[14]

A microservice is an element of an application where each microservice is developed to perform a specific business functionality rather than being developed in logical layers like a monolithic application.[14]

These small services can be developed independently, run just like a monolithic application and communicate with each other through APIs.[14]

Microservice architecture has many advantages.

**Some of them are listed below:[12]**

- Flexible connection.
- High functionality.
- Strong cohesion.
- Accelerated scalability.
- Improved fault isolation.
- Data integrity.

### 5.2 Patterns for microservices

It is important from the beginning of development to establish correct design patterns.

Some of the most important patterns used when designing microservices are explained below:[14]

- Domain Driver Design.

- Domain-Driven Design is a way of building software by focusing on understanding the problem is solving really well.
  
- Data-Driven Design.
  - Data-Driven Design is the practice of basing your design decisions on data rather than intuition or personal preference.
  
- Service discovery pattern.
  - Studying this design template is one of the important aspects of my bachelor thesis. Details can be found in the Section 6.1.

## ■ 5.3 Challenges

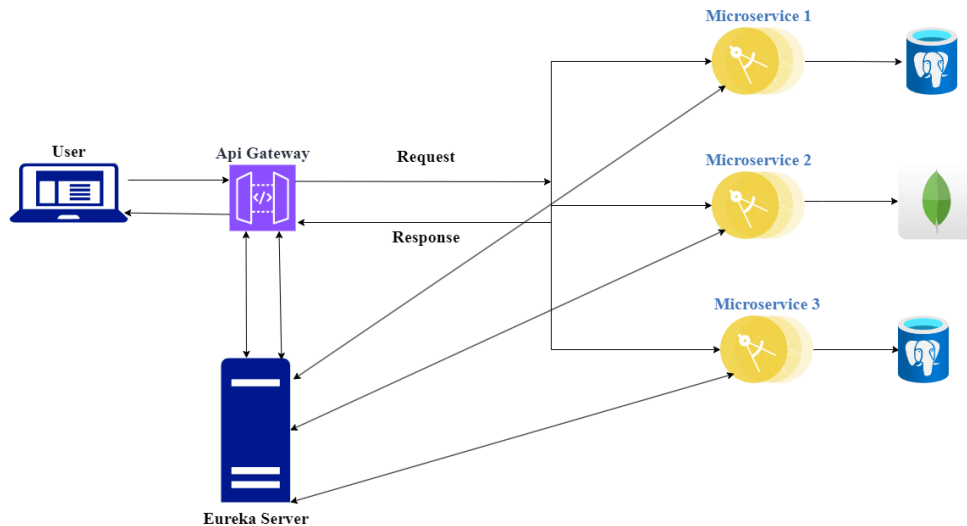
The crucial part of choosing microservices over monolithic applications is understanding the challenges that arise.

**The most relevant are detailed below:** [14]

- Data duplication.
  
- Network overhead.
  
- Understanding business logic.
  
- Understanding the current (as-is) and future architecture (to-be).
  
- Understanding only the required business functionality.

During the development of prototypes of a new information system for the rental system in the Strahov dormitory, the above challenges were processed and decisions were made to them.

## 5.4 Visualizing microservice architecture



**Figure 5.1:** Microservice Architecture.

The image shows a visualization of the microservice architecture.

### Architecture elements:

- User.
- API Gateway.
- 3 Microservices.
- Databases.

The user submits requests that go through an API gateway, which routes them to specific microservices based on the user's request and the business functionality of the individual service. The same principle was used for the architecture in Rentopia, addition information about Eureka discovery is available in the Section 7.5.

Eureka's discovery service is one of the most important aspects of my bachelor's thesis. The next chapter provides a detailed analysis of the service discovery pattern and Eureka.





## Chapter 6

### Service discovery

The purpose of this chapter is to analyze what service discovery is and determine why it is so important. The next step is to compare Eureka technology with alternatives in the Rentopia information system. In the following subsections, I'm going to analyze service discovery, define evaluation and selection criteria, describe Eureka technology, specify alternatives, and conduct a comparative analysis with Eureka.

#### 6.1 Understanding service discovery

When creating microservices, an API is usually used that provides the ability to interact with the back-end. We need to know where these services are hosted. The idea is that the services themselves are registered in an object called **Service Registry** at startup. So that when need to use a service, registered Eureka clients search for available instances in a server that stores data about available services. Each service at startup indicates its name and the address at which it is located.[14]

##### 6.1.1 The benefits of service discovery

###### **Heartbeat.**

Service Registry should determine which services are still active. The **heartbeat** is used for this. In addition to the initial registration of the service, signs of life drop from time to time so that the Registrar of Services knows that the service is still available. If any service doesn't show signs of life, the service registry receives information that something is wrong with this service and assumes that it is unavailable, so all requests will be redirected to other instances of the required service.[15]

###### **Load Balancer.**

Load balancing is a tactic of distributing incoming load, determined by the number of requests, to a separate microservice between its numerous instances. In this way, queries can be executed in a way that maximizes speed



**Benefits:** [13]

1. Ease of Use:
  - a. Netflix Eureka offers an easy setup process with minimal configuration requirements.
  - b. Integration with Spring Cloud components improves overall development efficiency.
2. Resilience:
  - a. The peer-to-peer replication model, which distributes the service registry across multiple instances, is part of the Eureka sustainability-oriented project.
  - b. Even during network partitions or node failures, the self-preservation mode guarantees the service registry's stability.
3. Flexibility:
  - a. Eureka provides flexibility in service discovery with support for multiple modes.
4. Competent Failure handling:
  - a. The heartbeat mechanism in Eureka provides monitoring of the health of registered services allowing the system to retain information about system failures and provide information about available services.
5. Customization Options:
  - a. Developers have the flexibility to customize registration and discovery behaviors according to specific project requirements.
6. Performance
  - a. Eureka demonstrates exceptional performance in the context of service registration and discovery. It is characterized by the ability to quickly register services. This emphasize that performance of Eureka is more suitable for the needs of Rentopia information system, which will dynamically develop in connection with the requirements of users.
7. Reliability
  - a. Eureka's self-preservation mode ensures the stability of the service registry, even during network partitions or node failures. This means that Eureka can continue to provide service discovery functionality even under various errors and problems.
8. Scalability



## ■ 6.5 Decision

After conducting a detailed comparative analysis, I came to the conclusion that I would choose Eureka. All technologies have similar characteristics that set them apart, but given the context of my information system and my skills and knowledge, I'm leaning towards Eureka.

Eureka serves as a reliable service registry and discovery tool that meets Rentopia requirements and surpasses Apache ZooKeeper in most evaluation criterias.





## **Chapter 7**

### **Implementation of prototypes**

This chapter will be devoted to the implementation of the prototypes of the Rentopia information system. This includes building a core microservices architecture, implementing graphical prototypes, and Eureka integration.



#### **7.1 Structure of project**

In this section below, you can familiarize yourself with the structure of the project, represented as a tree.

The structure of my project is a microservices architecture with a Rentopia root module and child modules representing services that have their own role and tasks.



**Figure 7.1:** Structure of project.

As you can see in figures 7.1, 7.2, and 7.3, there is a reference for my Structure of project.

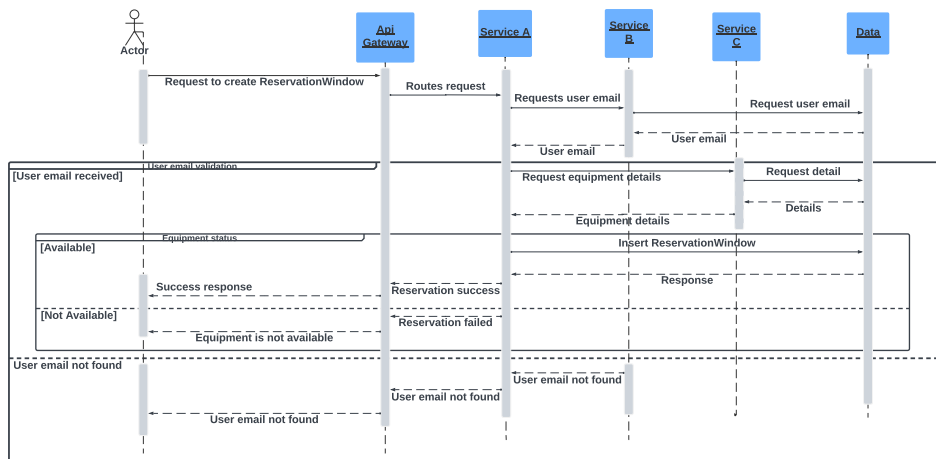
## 7.2 Critical functionality

The critical functionality that the prototypes will be based on will be the first scenario and the second scenario. First scenario: Registration in the system. Second scenario: Authorization in system and reservation of equipment in Rentopia. You can read these scenarios above in Section 3.4.



## 7.3 Sequence diagram

The Sequence diagram on image 7.4 describes the equipment reservation scenario.



**Figure 7.4:** Sequence diagram: Equipment reservation.

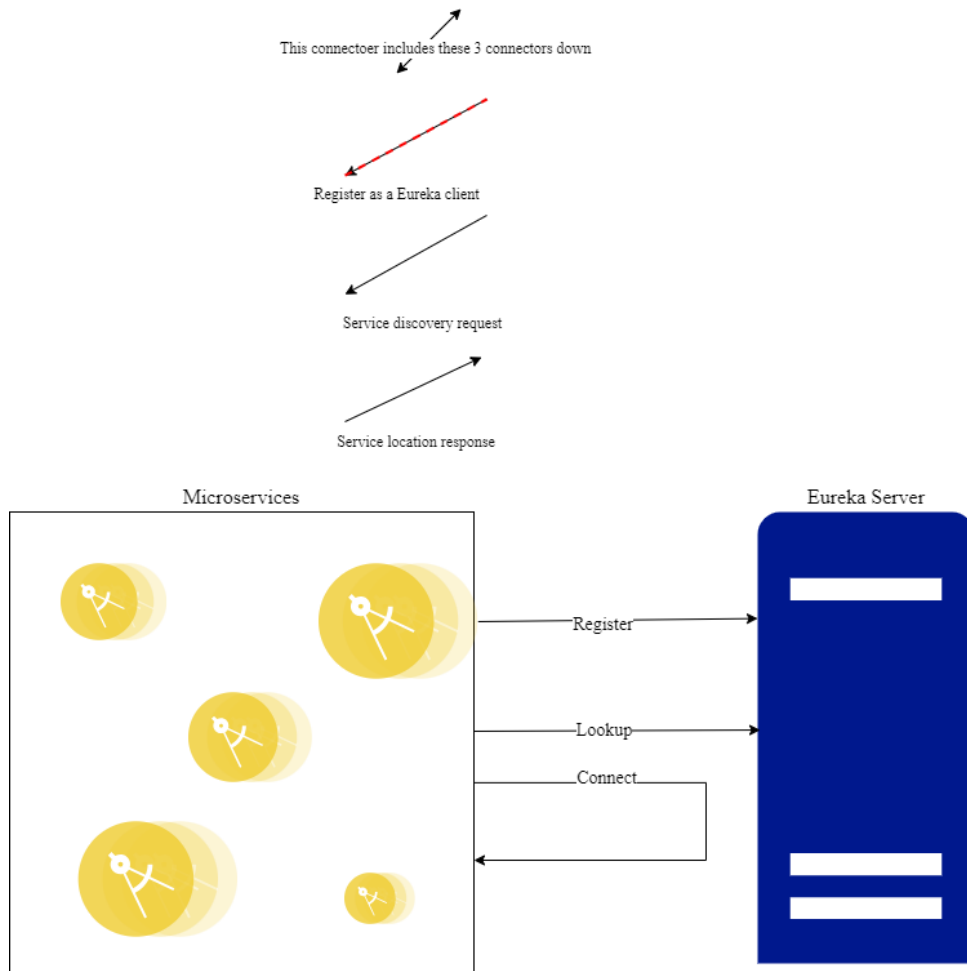
After successful authorization, the user can create a reservation window. It sends a request that passes through the API Gateway and is sent to the service that is responsible for working with the user account.

It sends a request to the database to determine if such a user exists. The database sends a response: If the user exists, the reservation process begins, if not, it issues an error and offers authorization again.

The reservation process is as follows. The API Gateway routes the request to the service responsible for equipment management and checks the availability and status of the desired equipment.

If the equipment is available for reservation, the request is routed to the service responsible for the reservation, where the reservation process takes place. If the equipment is not available, the user will receive an appropriate response.

## 7.4 Graphical prototypes



**Figure 7.5:** Eureka technology.

### How is Eureka working?

- Client Registration: Instances of microservices automatically register themselves with Eureka Server.
- Service discovery: Eureka Server maintains a registry of all client applications running on different ports and IP addresses. Eureka Server stores metadata of microservices such as port and IP address.
- Eureka Client(Microservice) sends service discovery request to Eureka Server to get metadata of other services.
- Eureka Server sends service discovery request to Eureka Client with metadata.
- Eureka Client gets needed microservice metadata.

### Graphical representation of the Eureka functionality.

The image above contains four connectors.

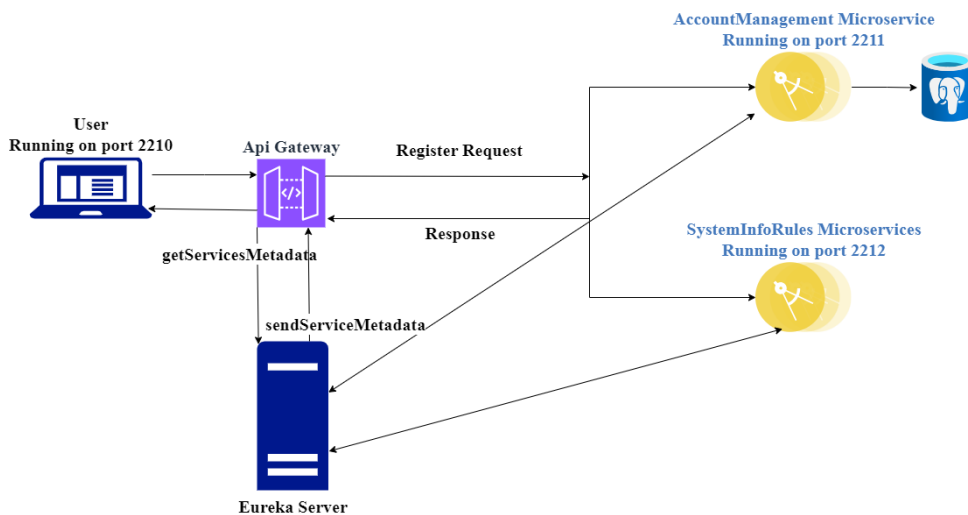
The first connector depicts the microservice registration function in the Eureka Server as the Eureka Client. The red dotted line indicates the Eureka heart-Beating technology which periodically send periodic heartbeats to indicate their availability.

The second connector depicts the microservice sending service discovery request. Microservices send this request to get metadata of target microservice.

The third connector depicts the Eureka Server sending Service location response, which contains metadata of target microservice.

The two-side pointer connector includes three connectors. I decided to go this way to ensure high understanding and ease of reading of the prototype.

## 7.5 Showcase interactions

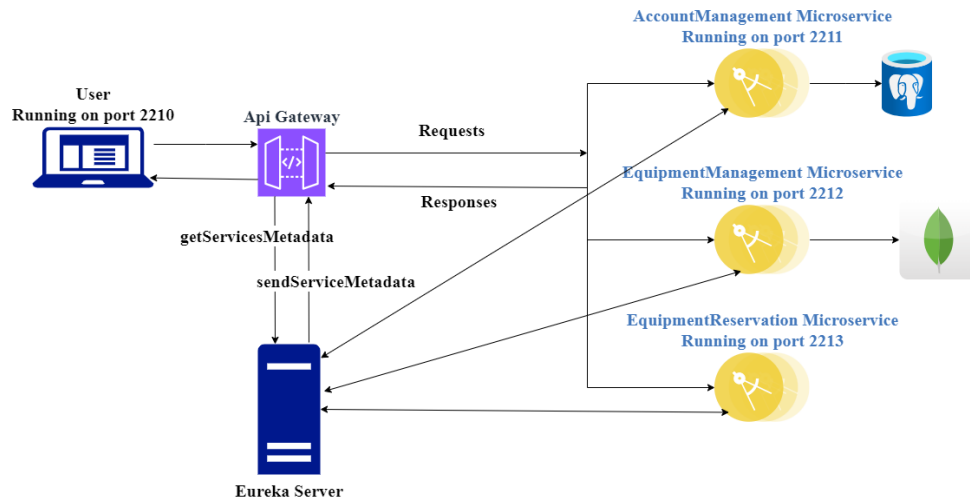


**Figure 7.6:** First scenario prototype.

### First scenario prototype.

1. Microservices start-up and register in Eureka Server as Eureka Clients. Eureka Server saves metadata, such as host and port. Microservices that register with Eureka Server periodically sending heartbeats to indicate their availability.
2. The user fills a form and sends user's registration request through API Gateway, which handles security, filter and routing.
3. The API Gateway sends a service discovery request to get information about available services.
4. The Eureka Server sends service discovery response.

5. The API Gateway routes to the Account-Management service, that should handle the user's registration request.
6. After successful registration, the API Gateway routes to the main page, where he can get acquainted with the information and rules regarding the Strahov dormitory rental system.



**Figure 7.7:** Second scenario prototype.

### Second scenario prototype.

1. Microservices start-up and register in Eureka Server as Eureka Clients. Eureka Server saves metadata, such as port and host. By this host and port will be working service discovery. Microservices that register with Eureka Server periodically sending heartbeats to indicate their availability.
2. Rentopia provides a window with form for authorization.
3. The user fills a form and sends an authorization request to the application through API Gateway.
4. The user's authorization request is sent through API Gateway, which handles security, filter and routing.
5. The API Gateway sends a service discovery request to get information about available services.
6. The Eureka Server sends service discovery response.
7. The API Gateway routes to Account-Management service, that handles user's authorization request.
8. After successful authorization the user is transferred to the main page, where he can get acquainted with the information and rules regarding the Strahov dormitory rental system.

9. The user sends a request for showing information about equipment through API Gateway, which handles security, filters and routing.
10. The API Gateway sends a service discovery request to get information about available services.
11. The Eureka Server sends service discovery response.
12. The API Gateway routes to Equipment-Management service, that handles user's request.
13. The Equipment-Management service gets a request and sends a response with required data, including equipment information.
14. The user sends request for reservation equipment through API Gateway, that handles security, filters and routing. The API Gateway determines which microservice should handle the user's reservation request.
15. The API Gateway sends a service discovery request to get information about available services.
16. The Eureka Server sends service discovery response.
17. The API Gateway routes to Equipment-Reservation service, that handles user's reservation request.
18. The Equipment-Reservation service gets a request and processes it. Equipment-Reservation service sends response through API Gateway to user with required data.

## 7.6 Testing

The testing of the Rentopia information system prototypes includes 3 different tests.

- The JUnit tests focus on the functionality and behavior of the account-management and equipment-management services.
- In the Eureka service discovery testing, two scenarios are outlined.
- The API Gateway testing section showcases the functionality of the API Gateway in routing requests to the appropriate services based on their names.
- Manual API testing with Postman manually showcases sending requests via Postman to check the endpoints of controllers and ensure that the responses match the expected results.

### 7.6.1 JUnit tests of the services

The JUnit tests focus on the functionality and behavior of the account-management and equipment-management services. These tests verify the correctness of the service logic and ensure that they perform as expected under various scenarios.[17] All running tests returned the expected result, which can be considered a success.

### 7.6.2 Eureka service discovery testing

#### The first scenario: Successful eureka client registration.

1. The first step is to start the Discovery-server-service. This service is a Eureka server. Services like Eureka clients will be registered through it.
2. The second step is to start API Gateway service. Register the API Gateway on port 8080, which will be responsible for routing requests.
3. Start and register the remaining services. The order doesn't matter.
4. To determine that the registration was successful, you can use the output and on the Eureka dashboard at localhost:8761. They can be found here: Successful output 7.8 and Eureka dashboard 7.9.

```

applicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_EQUIPMENT-RESERVATION-SERVICE/host.docker.internal:equipm
applicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_EQUIPMENT-RESERVATION-SERVICE/host.docker.internal:equipm
main] e.EquipmentReservationServiceApplication : Started EquipmentReservationServiceApplication in 6.917 seconds (process
nExecutor-0] com.netflix.discovery.DiscoveryClient : Disable delta property : false
nExecutor-0] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : null
nExecutor-0] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
nExecutor-0] com.netflix.discovery.DiscoveryClient : Application is null : false
nExecutor-0] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : true
nExecutor-0] com.netflix.discovery.DiscoveryClient : Application version is -1: false
nExecutor-0] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the eureka server
nExecutor-0] com.netflix.discovery.DiscoveryClient : The response status is 200
  
```

Figure 7.8: Successful eureka client registration output.

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
ACCOUNT-MANAGEMENT-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:account-management-service:0
API-GATEWAY	n/a (1)	(1)	UP (1) - host.docker.internal:api-gateway
EQUIPMENT-MANAGEMENT-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:equipment-management-service:0
EQUIPMENT-RESERVATION-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:equipment-reservation-service:0

Figure 7.9: Eureka dashboard: Registered instances.

#### The second scenario: One of the two instances is down.

1. The first step is to start the Discovery-server-service. This service is a Eureka server. Services like Eureka clients will be registered through it.
2. The second step is to start API Gateway service. Register the API Gateway on port 8080, which will be responsible for routing requests.
3. Start and register the remaining services. The order doesn't matter.

4. For this test, I conducted a simulation when two instances of the same equipment-management service are running and one of them is DOWN(not available).
5. On the Eureka dashboard, you can track the status of service instances. Eureka dashboard 7.10
6. I sent a request and received the expected response. Thanks to the API Gateway service, which identified an unavailable instance and thanks to Load Balancer, selected an available instance.

Application	AMIs	Availability Zones	Status
ACCOUNT-MANAGEMENT-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal.account-management-service-0
API-GATEWAY	n/a (1)	(1)	UP (1) - host.docker.internal.api-gateway
EQUIPMENT-MANAGEMENT-SERVICE	n/a (2)	(2)	UP (1) - host.docker.internal.equipment-management-service-808z DOWN (1) - host.docker.internal.equipment-management-service-0
EQUIPMENT-RESERVATION-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal.equipment-reservation-service-0

**Figure 7.10:** Eureka dashboard: One of the instances is DOWN.

### 7.6.3 API Gateway testing

API-Gateway testing was conducted during the life of Rentopia. I have set up routes for all services. For showing, that routing is working, i added image with output of API-Gateway service 7.11.

The API gateway functionality was clearly tested during sending request for reservation equipment in equipment-reservation service. This request includes the provision of information about user and equipment from other services.

The API Gateway is also registered as a Eureka client. Thanks to Eureka's integration with the API gateway, the gateway can dynamically search for the location of services based on their names, rather than hard-coding specific URLs. This provides flexibility and scalability, as services can be added and removed without the need to manually update the API Gateway configuration.

```

terFunctions : LoadBalancerClientFilter url chosen: http://host.docker.internal:55579/api/client/register
prFilter    : Weights attr: {}
terFunctions : LoadBalancerClientFilter url chosen: http://host.docker.internal:55579/api/client/allClients
prFilter    : Weights attr: {}
terFunctions : LoadBalancerClientFilter url chosen: http://host.docker.internal:55579/api/admin/allAdmins
prFilter    : Weights attr: {}
terFunctions : LoadBalancerClientFilter url chosen: http://host.docker.internal:55579/api/client/getClientByEmail?email=test.client@test.t
prFilter    : Weights attr: {}
terFunctions : LoadBalancerClientFilter url chosen: http://host.docker.internal:55594/api/reservationWindow/createReservationWindow
prFilter    : Weights attr: {}
terFunctions : LoadBalancerClientFilter url chosen: http://host.docker.internal:55594/api/reservationWindow/getReservationWindow?title=tes
prFilter    : Weights attr: {}
terFunctions : LoadBalancerClientFilter url chosen: http://host.docker.internal:55594/api/reservationWindow/addMeetingTime

```

**Figure 7.11:** API gateway routing.

### 7.6.4 Manual API testing with postman

This type of tests was created to test the operation of controllers in microservices. Success can be defined by returning the expected responses. Below are examples of requests and endpoints.

Also, as you can see, the localhost port 8080 is indicated everywhere, although the services run on different ports. This was achieved thanks to API Gateway and Eureka. Also below 7.6.6 are testing of requests and expected responses.

For this test was used Postman.

## Postman.

”Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.”[18]

### 7.6.5 Examples of requests

#### Registration request:

*Endpoint:* localhost:8080/api/client/register

```
"email": "dos777@gmail.com",
"name": "Dastan",
"surname": "Sadyraliyev",
"nickName": "sadyrdas",
"phoneNumber": "777-777-777-777",
"password": "boss"
```

#### Adding new equipment request

*Endpoint:* localhost:8080/api/equipment/management/newEquipment

```
"status": "IMMACULATE",
"title": "vacuumcleaner"
```

#### Adding description to equipment request

*Endpoint:* localhost:8080/api/equipment/management/descriptionToEquipment

```
"title": "vacuumcleaner",
"description": "New vacuumcleaner, that you can
use to clean your rooms"
```

#### Create reservation window request

*Endpoint:* localhost:8080/api/reservationWindow/createReservationWindow

```
"duration": 1,
"title": "VacuumCleanerForDastan",
"startDate": "2024-06-22 12:30",
"equipmentTitle": "vacuumcleaner",
"clientEmail": "dos777@gmail.com"
```



## 7.6.6 Responses

```

Iteration 1
POST registerClient
localhost:8080/api/client/register 201 Created 541 ms 128 B
|
| No tests found
|
GET allClients
localhost:8080/api/client/allClients 200 OK 178 ms 266 B
|
| No tests found
|
GET allAdmins
localhost:8080/api/admin/allAdmins 200 OK 10 ms 166 B
|
| No tests found
|
GET userByEmail
localhost:8080/api/client/getClientByEmail?email=test.client@test.test 200 OK 27 ms 264 B
|
| No tests found

```

**Figure 7.12:** Account-management-service response.

```

GET allEquipments
localhost:8080/api/equipment/management/allEquipments 200 OK 23 ms 226 B
|
| No tests found
|
PUT makeEquipmentAvailable
localhost:8080/api/equipment/management/makeEquipmentAvailable?availability=true 200 OK 38 ms 123 B
|
| No tests found
|
POST descriptionToEquipment
localhost:8080/api/equipment/management/descriptionToEquipment 200 OK 15 ms 185 B
|
| No tests found
|
GET EquipmentByTitle
localhost:8080/api/equipment/management/getEquipmentByTitle?title=ballForVolleyball 200 OK 46 ms 224 B
|
| No tests found
|
GET equipmentByAvailability
localhost:8080/api/equipment/management/getEquipmentByAvailability/false 200 OK 13 ms 166 B
|
| No tests found

```

**Figure 7.13:** Equipment-management-service response.

```

POST createReservationWindow
localhost:8080/api/reservationWindow/createReservationWindow 200 OK 126 ms 123 B
|
| No tests found
|
GET reservationWindow
localhost:8080/api/reservationWindow/getReservationWindow?title=test 200 OK 18 ms 334 B
|
| No tests found
|
PUT addMeetingTime
localhost:8080/api/reservationWindow/addMeetingTime 200 OK 39 ms 213 B
|
| No tests found

```

**Figure 7.14:** Equipment-reservation-service response.

## 7.7 Review

By covering unit tests for service functionality, Eureka service discovery testing for dynamic service registration and management, and API Gateway testing for request routing, the system's scalability, performance, ease of use and reliability are thoroughly evaluated.

Ease of use is achieved thanks to the user-friendly Web UI that displays the Eureka dashboard. Eureka dashboard provides valuable information about registered Eureka clients (services), including their status and network location.

Scalability and reliability are achieved thanks to combination of Eureka and API Gateway. Using Eureka, the API Gateway can request the locations of necessary services in the Discovery server. This simplifies service discovery and provides dynamic routing of requests within the Rentopia architecture even in situations, when instance of service is not available.

Overall, this testing section provides a clear overview of the testing strategies employed to validate and ensure the quality and performance of the Rentopia information system prototypes.

## 7.8 Future developments

In this section, ideas are written that were invented, but are not implemented within the The Bachelor's thesis and belong to the future extension of the application.

- Implementation of the mobile version.
- Adding the remaining functional requirements.
- Web-UI implementation.
- Implementation of the auth-service for authentication and authorization. I couldn't implement this service because I wanted to integrate the Silicon Hill database, but the administration couldn't give me access. I wanted to add the Silicon Hill Club member Id to the registration in order to compare it with the Id from the Silicon Hill database. In the same way, it was possible to provide exclusive access to the Rentopia system for residents of the Strahov hostel.
- Add a language change feature so that foreign residents do not have problems using Rentopia.
- Containing services using Docker and deploying them to a selected cloud service platform (for example, AWS, Google Cloud, Azure) to ensure convenient application management and scaling.

## Chapter 8

### Conclusion

The purpose of this work was to design and create prototypes of an information system for the rental system at the Strahov hostel. The main task is to optimize and increase the accessibility, responsiveness and convenience of the rental system for both students and administration in using the rental system.

The information system is called Rentopia. I want to say that this work achieved its goals. The analytical part was devoted to analyze the existing system, which is a grill-only rental system. It was discovered that there was no information system for the rental system. The analytical part also was devoted to defining the problem and collecting the requirements and wishes of future users: Strahov dormitory residents and administrators. The design part lists all functional and non-functional requirements.

The next stage of the bachelor's work was the creation of detailed wireframes based on the requirements, which represented the appearance of the future Rentopia system. I shared them with future users and received feedback, which included both positive feedback and wishes and instructions from the administrator.

Also, the second part of the analysis was an analysis of the microservice architecture and, in particular, the Eureka tool discovery service.

Having carried out this analysis, the stage of implementation of prototypes of the future system began. Since this was my first experience with microservice architecture, some difficulties arose, which I successfully dealt with.

Prototypes testing included 4 stages:

- The JUnit tests of the services
- Eureka service discovery testing
- The API Gateway testing
- Manual API testing with Postman

Having conducted these tests, I can indicate that Rentopia will be able to meet the expected requirements.

Due to the high volume of work, the requirements were divided into those implemented within the framework of the bachelor's thesis and those that

provide the opportunity to improve and expand the system in the future. As part of further expansion and development, priority will be given to the implementation of these future goals. The system is written in such a way that it can be expanded with additional functionality, which is indicated in the chapter on future development.

To sum it up, I would like to say that Rentopia has got its start, and in the future it can become an integral part of life in the Strahov dorm.



## Appendix A

### List of abbreviations

- JSON: "JavaScript Object Notation".
- API: "Application Programming Interface".
- BANY: "This is a system of punishment for late return and damage to equipment. Prohibition of use for late return: vacuum cleaners - 7 days, grill - 5 days, everything else - 3 days".
- TROSAD: "The registry of services and discovery tools".
- PostgreSQL: "Postgres Structured Query Language".
- MongoDB: "Humongous DataBase".
- CRUD: "Create, Read, Update, and Delete (CRUD) are the four basic functions that models of database should be able to do, at most".
- JUnit: "A unit testing framework for the Java programming language".





## **Appendix B**

### **List of links**

- GitHub: <https://github.com/sadyrdas/Rentopia>
- GitLab: <https://gitlab.fel.cvut.cz/sadyrdas/bachelor-projekt>
- Wireframes Figma: <https://shorturl.at/talqY>
- Questionnaire: <https://forms.office.com/r/SruqYyG5Bq>





## Appendix C

### Bibliography

- [1] Asel Kalibaeva one of the administrator of Silicon Hill organization and resident of Strahov Dormitory.
- [2] Silicon Hill web-page. “*Current state of rental service*”. Accessed on March 3 2024. Retrieved from [https://wiki.sh.cvut.cz/kolej/bloky/blok\\_6](https://wiki.sh.cvut.cz/kolej/bloky/blok_6)
- [3] ProductPlan. “*What is MoSCoW prioizitation?*”. Accessed on March 6 2024. Retrieved from <https://www.productplan.com/glossary/moscow-prioritization/>
- [4] Microsoft Azure. “*What is Java?*”. Accessed on March 15 2024. Retrieved from <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-programming-language>
- [5] Tutorialspoint. “*What is Spring Boot?*”. Accessed on March 15 2024. Retrieved from [https://www.tutorialspoint.com/spring\\_boot/spring\\_boot\\_introduction.htm](https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm)
- [6] PostgreSQL official web-page. “*What is PostgreSQL?*”. Accessed on March 15 2024. Retrieved from <https://www.postgresql.org/about/>
- [7] AWS Amazon. “*What is MongoDB?*”. Accessed on March 15 2024. Retrieved from <https://aws.amazon.com/ru/documentdb/what-is-mongodb/#:~:text=MongoDBisanon-relational,withrichandintuitiveAPIs.>
- [8] Medium. “*What is Sketch?*”. Accessed on March 22 2024. Date of Publication: 20 February 2024. Retrieved from <https://shorturl.at/kPnX2>
- [9] Author: Staiano,F. (2022). “*Designing and prototyping interfaces with Figma : Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop.* Date of publication: February 2022. Retrieved from <https://shorturl.at/Akzeo>
- [10] Arlan Nurkhozhin 3rd year student of the CTU FEL SIT and resident of Strahov Dormitory.

- [11] Vyacheslav Tsay a 3rd year student of the CTU FEL SIT and resident of Strahov Dormitory
- [12] ATLISSIAN. "*Benefits of microservices*". Accessed on April 15 2024. Retrieved from <https://www.atlassian.com/microservices/cloud-computing/advantages-of-microservices>
- [13] GeeksForGeeks. "*What is Eureka?*". Accessed on April 18 2024 Retrieved from [https://www.geeksforgeeks.org/spring-boot-eureka-server/?ref=header\\_search](https://www.geeksforgeeks.org/spring-boot-eureka-server/?ref=header_search)
- [14] IEEE Xplore. Title: A survey on microservices architecture: Principles, patterns and migration challenges. Authors: Victor Velepucha, Pamela Flores atd. Date of Publication: 15 August 2023. Retrieved from <https://ieeexplore.ieee.org/abstract/document/10220070>
- [15] Science Direct. Title: Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. Authors: Shanshan Li, He Zhang atd. Date of Publication: March 2021. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0950584920301993#sec0001>
- [16] Election Science. "*What is Score Voting?*". Accessed on April 23 2024. Retrieved from <https://electionscience.org/library/score-voting/>
- [17] Medium. "*JUnit testing with spring boot*". Accessed on May 5 2024. Date of publication: 25 March 2023. Retrieved from <https://medium.com/@AlexanderObregon/mastering-spring-boot-testing-with-junit-and-mockito-8bec9b4911fc>
- [18] Postman's official website. "*What is Postman?*". Accessed on May 15 2024. Retrieved from <https://www.postman.com/product/what-is-postman/>