ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hájek** Jméno: **David** Osobní číslo: **474556**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Průvodce nemocemi

Název diplomové práce anglicky:

Illness Guide

Pokyny pro vypracování:

Zdravotnictví se potýká se stárnoucí populací a zároveň nedostatkem lékařů. Přitom pacienti disponují chytrými zařízeními, které by mohly pomoci situaci zlepšit.
Cílem této práce bude vývoj systému, který bude minimalizovat čas pacienta u lékaře, zefektivní samotnou léčbu a zároveň bude pacienta motivovat dodržovat předepsané úkony.
Systém se bude skládat z webové a mobilní aplikace a bude mít za úkol provést pacienta nemocí, poskytnout mu o ní informace. Dále bude dohlížet na dodržení léčby a průběžně monitorovat stav pacienta. Údaje budou zadávány samotným pacientem - tlak, požití léku, cvičení ap. Výsledky ve formě grafu, tabulky nebo kalendáře bude možné sdílet jednoduchým způsobem s lékařem.
Požadavky
1. Zanalyzujte a popište aktuální stav software a webových stránek s informacemi o nemocech.
2. Implementujte mobilní aplikaci, která zobrazí informace o několika vybraných nemocech. Pro danou nemoc je potřeba zadat požadavky na pacienta - jaké léky a v jakých intervalech je má brát, jak a kdy má cvičit, kdy má zadávat požadovaná měření, např. tlak. Aplikace se bude aktivně připomínat.
3. Implementujte webovou aplikaci pro lékaře, který bude moci sledovat výsledky pacienta. Vyřešte jednoduchý a dostatečně bezpečný způsob autorizace přístupu k pacientovým datům.
4. Popište, jak zapadá práce s daty do GDPR.
5. Implementaci vyzkoušejte na reálných lidech.

Seznam doporučené literatury:

[1] Patterns of Enterprise Application Architecture — Martin Fowler
[2] Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications — Mark Heckler
[3] Beginning App Development with Flutter: Create Cross-Platform Mobile Apps — Rap Payne

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Petr Aubrecht, Ph.D. katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **30.01.2023** Termín odevzdání diplomové práce: **26.05.2023**

Platnost zadání diplomové práce: **22.09.2024**

Ing. Petr Aubrecht, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Master's thesis

Illness Guide

Bc. David Hájek

Supervisor: Ing. Petr Aubrecht, Ph.D.

Study Programme: Open Informatics

Field of Study: Software Engineering

May 26, 2023

Aknowledgements

I would like to thank to my thesis supervisor Ing. Petr Aubrecht, Ph.D. for his invaluable advice and help in the preparation of the master's thesis. Without his guidance, the thesis would certainly not have reached the final level of sophistication and expertise. I would also like to thank those who gave their time for professional consultations, especially the doctors.

Declaration

V Praze dne 26. 5. 2023 ....

Abstract

The healthcare sector is facing an aging population and a shortage of doctors. Yet patients have smart devices that could help improve the situation.
The goal of this work will be to develop a system that will minimize the patient's time at the doctor's office, make the treatment itself more efficient, and motivate the patient to comply with the prescribed actions.
The system will consist of a web and mobile application and will be designed to guide the patient through the disease, providing information about it. It will also supervise adherence to treatment and continuously monitor the patient's condition. The data will be entered by the patient himself – pressure, medication ingestion, exercise, etc. The results in the form of a graph, table or calendar will be shared in a simple way with the doctor.

Keywords: Software, development, web application, mobile application, Java, Spring boot, React, Flutter, healthcare

Abstrakt

Zdravotnictví se potýká se stárnoucí populací a zároveň nedostatkem lékařů. Přitom pacienti disponují chytrými zařízeními, které by mohly pomoci situaci zlepšit.
Cílem této práce bude vývoj systému, který bude minimalizovat čas pacienta u lékaře, zefektivní samotnou léčbu a zároveň bude pacienta motivovat dodržovat předepsané úkony.
Systém se bude skládat z webové a mobilní aplikace a bude mít za úkol provést pacienta nemocí, poskytnout mu o ní informace. Dále bude dohlížet na dodržení léčby a průběžně monitorovat stav pacienta. Údaje budou zadávány samotným pacientem – tlak, požití léku, cvičení ap. Výsledky ve formě grafu, tabulky nebo kalendáře bude možné sdílet jednoduchým způsobem s lékařem.

Klíčová slova: Software, vývoj, webová aplikace, mobilní aplikace, Java, Spring boot, React, Flutter, healthcare

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the healthcare system in Czechia has faced numerous challenges, including an increasing workload for healthcare professionals and limited access to quality healthcare for patients. All this was amplified by the recent epidemic of Covid-19. This Master's thesis addresses these concerns by exploring the development of a novel software application designed to revolutionize the healthcare industry. By leveraging the power of technology, this application aims to bring better healthcare services to patients across Czechia while simultaneously alleviating the burden on the healthcare system. In this introductory chapter, we lay the foundation for our research by outlining the significance of the problem and identifying the objectives of the study.

For my bachelor's thesis, I worked on a similar topic, where I implemented a prototype that suggested how this issue could be conceptualized. With the experience of doing my bachelor's thesis and my master's studies, I decided to take up the topic again and try to take it further. I am aware from my bachelor's thesis analysis that the main idea of the application can make sense, particularly in some areas of healthcare. For example, I was encouraged by a conversation with David Marx, MD, who found particular interest in the education module with its focus on personalized education and saw potential in segments such as bariatric surgery, plastic surgery, and long-term illnesses. I also discussed the idea with Eva Koblihová, MD, who specializes in bariatric surgery, and she saw the biggest benefit in having the entire topic covered in one place and thought an interactive app would be useful in that regard. [18]

1.1 Healthcare crisis

The Czech healthcare system is facing a serious crisis: lack of resources, exhausted health workers, and declining access to care are just some of the problems facing patients in the country. The struggle over the financing and budget of the healthcare system has dragged on for several years and no lasting and effective solution has yet been found. Insufficient funding stands out as a primary contributor to the crisis in the Czech healthcare system. A key comparative factor highlighting this issue is the proportion of total health expenditure in relation to the country's GDP. Throughout the years, the Czech Republic has consistently maintained a lower level in this indicator when compared to other OECD nations. Specifically, in 2018, healthcare expenditure reached only 7.5% of the country's GDP, significantly lower than the OECD average which exceeds this figure by more than one percentage point. See figure 1.1. This stark contrast indicates that the Czech health sector receives comparatively less funding than more than half of the OECD countries, leading to the widespread characterization of the sector as underfunded. In terms of healthcare costs, the Czech Republic stands out as having the most affordable healthcare price level among EU member states. See figure 1.2. Cheap medicines, price and reimbursement regulations, low salaries, and other elements of the Czech healthcare system show that we can afford a relatively high volume of care for less money.[29]

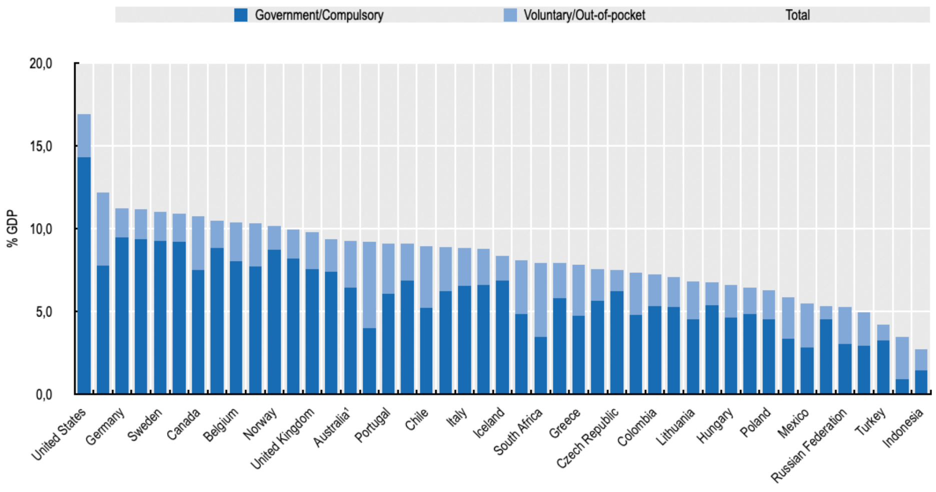


Figure 1.1: Health expenditure as a share of GDP, 2018 [35]

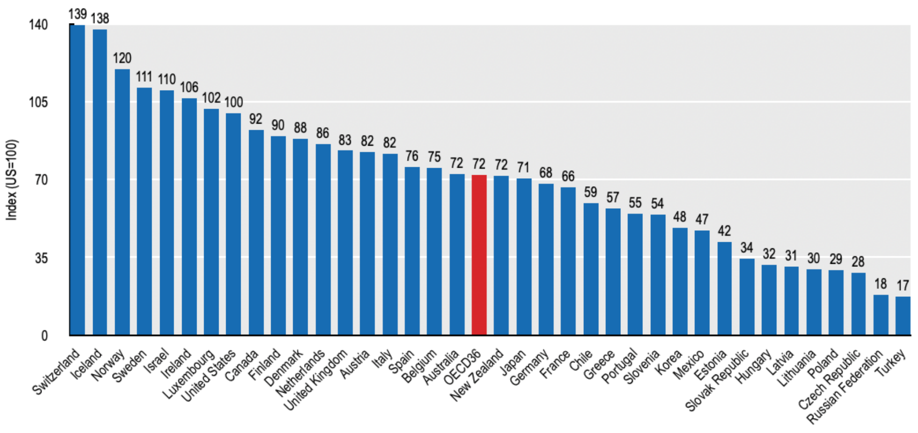


Figure 1.2: Comparative price levels for health, 2017 [35]

Another major problem is the shortage of health professionals. To worsen the situation, the existing medical staff is also aging. A significant part of the health workforce is expected to retire in the coming years. One of the recommendations for the Czech Republic by the European Commission is to ensure a sufficient number of health professionals. [34] The current shortage of healthcare professionals in Czech hospitals encompasses a deficit of more than a thousand nurses and hundreds of doctors. [19]

It is important to mention that despite facing challenges such as underfunding and a shortage of medical personnel, the Czech healthcare system continues to operate at a proficient level. Despite the relatively low share of total health expenditure in GDP and the scarcity of healthcare professionals, the system manages to deliver quality care to its population. The dedication and expertise of healthcare professionals contribute to the maintenance of a proficient level of healthcare services. The question is whether it is sustainable or not. Thus various measures, such as optimizing healthcare processes and implementing innovative technologies, could help compensate for the lack of resources and contribute to the overall effectiveness and resilience of the Czech healthcare system.

1.1.1 Patient's problems

The growing pressure on the Czech healthcare system is exacerbated by both the often ineffective treatment and inappropriate interaction between the patient and the hospital. Imagine a situation where a patient is unwell and decides to seek medical help. In the doctor's surgery, the doctor diagnoses a disease and tries to explain it to the patient in a limited amount of time, which is a problem with more complex diseases, and the patient is often upset for multiple reasons, e.g. due to the diagnosed disease. Next, the doctor prescribes treatment and again tries to explain it to the patient. Everything is done in a limited time and in a somewhat chaotic way. The patient forgets a smaller or larger part of the information and instructions before he gets home, so he starts searching for his illness online, visiting various unverified servers and social networks, where he learns misleading information about his illness. That can dangerously undermine his confidence in the treatment. This results in non-compliance with the prescribed treatment. For example, they learn from a non-professional person stating that the medicine has side effects and it is better not to undergo it because of various reasons. Another common problem relates to unintentional non-adherence to treatment. Patients often forget to do certain tasks at a specific time, such as taking antibiotic medications every 8 hours or forgetting to do the task at all. This can lead, for example, to the postponement of a scheduled operation. Together, these problems add to the workload of doctors, prolong recovery and increase the financial cost of their treatment.

The mentioned claims are backed up by research. For example, according to a survey conducted by Teva Pharmaceuticals, approximately 50% of patients diagnosed with diabetes, high cholesterol, or high blood pressure acknowledged non-compliance with their prescribed treatment. The greatest challenges reported by these patients were related to adhering to recommended dietary and exercise regimens. The survey encompassed 510 individuals receiving care from general practitioners and internists. [1]

1.2 Motivation and vision

The objective of this project is to develop a system that enhances the efficiency of patient treatment, reduces the time spent at the doctor's office, and promotes patient compliance with the prescribed treatment.

The fundamental idea is to offer patients a hospital-based service that assists them in managing their illnesses and undergoing procedures through a dedicated application. This application will serve as a guide for the patient throughout the treatment process, ensuring adherence to the treatment plan and providing reminders for necessary tasks. It will also provide interactive educational resources to help patients understand their illness better, and if needed, facilitate better comprehension for their family members. Additionally, the application will enable continuous monitoring of the patient's condition. Patients will be able to input valuable information about themselves, such as temperature or pain levels, and track the progression of their condition. The collected data will be presented in the form of graphs, tables, or calendars, making it easy to share with their doctor.

The system will comprise both a web and mobile application, designed to support patients in managing their illness, providing them with relevant information. It will also ensure adherence to the treatment plan and enable continuous monitoring of the patient's condition. The patient will be responsible for entering various data points, including blood pressure, medication intake, exercise, and more.

It is not my ambition to make a fundamental contribution to improving the financial or personnel aspect of the above-mentioned problem of the Czech healthcare system, because its solution is more of a political issue. The system I have created can primarily help to reduce the workload of health professionals, speed up and, in practical terms, qualitatively improve the treatment of patients. If it also has a positive side effect in the financial area (e.g. reduction of expenses) or the area of personnel (in the form of more efficient handling of work by a lower number of health workers), I will naturally be pleased.

Chapter 2

Analysis

In this chapter, I focus on a comprehensive analysis of the software that is being developed, taking into account the valuable knowledge and insights gained during my master's studies and leveraging the findings from my bachelor's thesis. This analysis encompasses a range of critical components that contribute to the success of the software. Firstly, functional and non-functional requirements are modeled, defining the desired functionalities and performance criteria of the software. These requirements serve as a blueprint for guiding the development process and ensuring the software meets the intended objectives.

In addition to establishing requirements, an Entity-Relationship (ER) diagram is created to visually represent the data model and relationships within the software. This diagram provides a clear understanding of the system's structure and aids in the effective design and implementation of the software. Furthermore, a comprehensive competition analysis is conducted to evaluate existing software solutions in the market. This analysis enables me to identify potential gaps, assess the competitive landscape, and refine the software's unique selling points. Additionally, a feasibility study is conducted to assess the practicality and viability of the software's implementation. Furthermore, GDPR compliance is considered to ensure that the software adheres to data protection and privacy regulations since it is dealing with sensitive patient data. Finally, a thoughtful business plan outlines the strategic direction, goals, and potential financial outcomes of the software project, taking into account both theoretical knowledge and real-world considerations.

2.1 Requirements

In the field of software development, business requirements play a crucial role in guiding the development process and ensuring that the resulting software meets the needs of the stakeholder. These requirements serve as a foundation for creating effective software solutions that align with the business objectives and strategies. Understanding and capturing accurate business requirements are vital for successful software development projects.

The purpose of this section in my master's thesis is to explore and analyze the business requirements in software development. By examining various aspects of business requirements, including their definition, I aim to shed light on their significance and impact on the development process.

Software development projects are often complex and multifaceted, involving different stakeholders with varying priorities, expectations, and constraints. Business requirements act as a bridge between the business stakeholders and the development team, ensuring that the resulting software addresses the specific needs.

Within this section, we will focus on the different types of business requirements, such as functional requirements that define the desired software functionalities and non-functional requirements that specify constraints related to performance, security, usability, and more. We will explore the process of eliciting, documenting, and validating these requirements, highlighting the importance of clear communication.

Furthermore, this section will discuss the challenges and risks associated with business requirements in software development. We will examine common pitfalls, such as ambiguous requirements, changing business needs, and scope creep, and explore strategies for mitigating these risks to ensure successful project outcomes.

By gaining a comprehensive understanding of business requirements in software development, this research aims to provide valuable insights and recommendations for both practitioners and researchers. By effectively managing business requirements, organizations can enhance their software development processes, improve stakeholder satisfaction, and achieve successful project outcomes. [47]

2.1.1 Functional requirements

Functional requirements in software development specify the particular functionalities and capabilities that a software system must possess to meet the needs of its users. These requirements define what the software should do and how it should behave from a functional standpoint. They outline the desired features, actions, and behaviors that users expect from the software. Functional requirements typically include details such as input data, processing logic, and expected outputs. [47]

ID	**Definition**
BR02	As a patient, I'll be able to see what stage of the disease I'm in.
BR03	As a patient, I will see what upcoming events I'm facing.
BR04	As a patient, I can check that I have fulfilled everything.
BR05	As a patient, I need to get information about my disease and its treatment.
BR06	As a patient, I need to see my progress to stay motivated in my treatment.
BR07	As a patient, I need to monitor myself for my doctor.
BR08	As a doctor, I will be able to see my patient's progress which the patient shares by QR code.
BR01	As a doctor, I'll be able to register a new patient.
BR09	As a patient, I need to edit my profile so that I can update my symptoms, etc.
BR10	as a patient I can check that I accomplished everything on the to-do list.

BR11	as a patient I can edit my medical anamnesis.
SR01	The system allows the patient to upload the data recorded by him.
SR02	The system allows communicating with an external service in order to save doctors' work.
SR03	The system will notify patients regularly of what they have to do so they don't forget anything.
SR04	The system will be secure to protect patients' sensitive data.
SR04	The system allows monitoring multiple parameters.

Table 2.1: Functional requirements

2.1.2 Non-functional requirements

Non-functional requirements are an addition to functional requirements. They describe the additional necessary characteristics needed by the production environment and context in which the application will operate.[47]

ID	**Definition**
NR01	Number of users - The system is able to deal with 1 500 online users.
NR02	Bandwidth - The system is able to withstand 300 requests per second.
NR03	Response time - 95 percent of requests will be resolved in under 4 seconds
NR04	Guaranteed Availability - The system has guaranteed 99,95 percent uptime.
NR05	Scaling - The system is able to scale by adding new instances.
NR06	GDPR - The system is GDPR compliant.

Table 2.2: Non-functional requirements

2.2 Competition

2.2.1 Seamless.md

There are few similar projects abroad. The most similar application is the SeamlessMD from the United States. Seamless.md is a web-based platform that aims to streamline and enhance patient-doctor interaction by providing various features and functionalities. Upon reviewing the website, it becomes evident that Seamless.md focuses on facilitating virtual consultations, appointment scheduling, and secure communication between patients and healthcare providers.

The platform offers a user-friendly interface where patients can easily book virtual appointments with their preferred healthcare professionals. It also provides a secure messaging system, enabling patients to communicate with their doctors and receive timely responses to their medical inquiries.

Seamless.md appears to prioritize convenience and accessibility by offering a range of features that allow patients to manage their healthcare remotely. These features include the ability to access medical records, view test results, request prescriptions, and receive appointment reminders.

Furthermore, the platform emphasizes the integration of telemedicine technology, enabling patients to have virtual consultations with healthcare providers through video calls. This feature eliminates the need for in-person visits, reducing travel time and increasing accessibility, particularly for patients with mobility constraints or residing in remote areas.

In terms of user experience, Seamless.md presents a modern and intuitive interface, making it easy for patients to navigate the platform and access the desired functionalities. The platform also highlights its commitment to data security and patient privacy, which is crucial in the healthcare sector. [46]

2.2.2 Moje endoprotéza

In one of the interviews mentioned in the section 8.4.3, the mobile app Moje endoprotéza was recommended to me. This is an app for patients who are going to have surgery to have an artificial joint implanted (endoprosthesis). The app prepares the patient for surgery and guides them through the surgical and post-surgical periods. It uses selected features on a specific case, which our system plans to use more universally for a wide range of diseases and medical procedures.

2.2.3 Other competitors

I already mentioned other competitors in my bachelor's thesis. Most of them are based on the eHealth and telemedicine principles. Our system does not go further in this direction and therefore it is not necessary to analyze them more deeply. [48]

2.3 Feasibility study

A Feasibility study is an analysis that takes into account all relevant factors including the economic perspective, options, possibilities, and likelihood of successful implementation. Project managers use the study to compare the positives and negatives of a project before investing time and money in it. In this project, the feasibility study is very challenging to determine. It is difficult to quantify the true benefits. It depends on many factors including, but not limited to, the clinic where the application would be used. However, the main goal of this application is to save the patient's time at the doctor's office. The amount of time saved varies based on the type of treatment. Once we know how much time the application saves in certain cases, we can then evaluate the application itself.

In this regard, it is necessary to select diseases or medical treatments suitable for this application according to whether pre-set treatment plans can be created for them. In other words, whether the time investment in creating such a plan and its subsequent management will be more expensive in terms of cost than it can save.

It is also essential to take into account the cost of treating individual diseases. If we achieve a saving of, for example, one day of hospitalization out of a total of five, then this is a greater saving for the health system than for another disease that generally does not involve hospitalization and the patient heals more quickly at home. For more high-cost diseases, then, there is no need to recruit a large patient group to make their treatment plans worthwhile.

2.4 GDPR Compliance

This is a platform in the healthcare industry, thus it's dealing with sensitive customer data. It's important that the application is GDPR compliant. What is meant by that is that we don't track the user, we are able to delete all his/her data on request and the user gives consent to work with the data. When it comes to sharing the data with a specific doctor the patient has to give time-limited consent. [20]

2.4.1 Consent to process data

At some point during the registration process, the new user has to give consent to process their data and agree with the terms and conditions (opt-in). [20]

2.4.2 The Right to Erasure (Right to be Forgotten)

Under GDPR, individuals have the right to request the erasure of their personal data. This right grants users the power to have their data permanently deleted, ensuring their privacy and control over their information. Medical software developers must implement mechanisms that facilitate the secure and efficient deletion of user data while maintaining data integrity. When it comes to deleting user data, it is crucial to follow secure practices to prevent unauthorized access or accidental data breaches. Some best practices include:

Anonymization: Consider anonymizing or pseudonymizing user data before deletion to maintain the integrity of statistical or research-related analyses, while ensuring individual identities are protected.

Encryption and Secure Erasure: Use strong encryption techniques to protect data at rest and in transit. When deleting data, ensure it is securely erased to prevent any potential data recovery.

Audit Logs and Data Traces: Implement comprehensive logging mechanisms to track data deletion activities, including user requests, deletion timestamps, and relevant metadata. This helps in demonstrating compliance and addressing any potential audit requirements.

Backups and Replication: Review and adjust backup and replication processes to ensure deleted data is not inadvertently restored or retained in backup systems. Define clear retention periods for backups and establish policies for securely deleting data from backup storage. [20]

2.4.3 Sharing data

The application allows patients to collect sensitive data about themselves. Patients may want to share this sensitive data with their doctors from time to time. If a doctor wants to see a patient's data, the patient has to give them time-limited consent in order for them to access it. After the time is up, the patient's data automatically disappear and the doctor can no longer see it. At the same time, the doctor, with the use of the app, agrees not to pass the data to anyone else (third parties) or to process it any further.[20]

2.4.4 Conclusion

As we described the issue of GDPR compliance here above, there are three general points above others; a consent to process data, the right to erasure data and to share data. It is crucial to keep in mind how sensitive user data can be processed in our system and hence, a proper and legally compliant process must be implemented.

2.5 Business plan

There would be only one instance of my application and hospitals or medical clinics would collaborate with it. Collaboration would be established by identifying suitable diseases and their treatment processes for a given medical facility. For these diseases and processes, people from the application side would start to create treatment guides. These guides would then be consulted with doctors or other health professionals. Thus, everything would be agreed upon by experts. All with a primary focus on minimizing the time required from the medical personnel.

Everyone would then have one app that would be used across the healthcare system and its facilities. The advantage will be that the patient will not have to register for each hospital separately. When it comes to monetization as I mentioned in the section 2.3, we have to focus on segments where the application can make a difference and save both money and time.

Chapter 3

Proposal

Based on the insights gained from the analysis, this chapter presents a detailed proposal for the development of a software application aimed at enhancing healthcare services in Czechia. We outline the objectives, scope, and architecture of the proposed application, highlighting its key features and functionalities. Additionally, we discuss the technologies and frameworks that will be used in the development process. Moreover, considering the complex regulatory and privacy concerns surrounding healthcare data, we address the legal and ethical considerations that need to be taken into account during the development and implementation of the application. Through this comprehensive proposal, we lay the groundwork for the subsequent chapters, which focus on the actual implementation and deployment of the software application.

In this chapter, I would like to also introduce a name that I decided to use for my project. I named my project Corposa. From now on, any mentions of Corposa mean the entire platform including all its components.

3.1 Main parts of the application

As outlined in the introduction, the application aims to guide the patient through the disease, educate the patient sufficiently and record patient data. Based on the defined requirements, I decided to divide the application into four general parts.

- Disease Guide
- Education module
- Monitoring
- Anamnesis

3.1.1 Disease Guide

This module will literally guide the patient through the disease. It will tell them what stage of the disease they are currently in. It will remind them what time to take their pills or when to stop eating. It will make sure that the patient follows the prescribed regimen. Or helps the patient to exercise properly. Check to-do lists and linking to the education module will help with the above.

The patient will be better prepared as a result. This minimizes the risk of the patient forgetting to take, for example, a pill before surgery and being unable to undergo the operation. Vice versa, the effect of the treatment on the patient himself is maximized. In the first phase of any treatment, it often happens that the patient forgets to get all the examinations. He then comes unnecessarily for a check-up, where he cannot get an appointment for surgery. This is another example where an app could help.

3.1.2 Education module

Imagine a scenario where a patient comes in for a check-up and is told their medical condition. He finds out that he will have to undergo surgery in two months and a list of things he will have to do is read to him. When he gets home, some of the things are already forgotten and the patient starts looking for information from unreliable sources (internet, forums, etc.).

This is what the education module wants to tackle. Here, we clearly and interactively convey information to the patient about the disease he is suffering from. The information will be provided directly by the hospital and the patient will not come across false information that could lead him to make decisions that would have a negative impact on his treatment. Moreover, the information will be personalized directly for the person concerned.

3.1.3 Monitoring

Monitoring will allow simple recording of basic data about the patient's condition. Importantly, the data will be uploaded to the app by the patient himself. He will therefore be able to measure his pain level, and weight or upload photos of a healing wound, for example. The patient will feel better cared for and the doctor will use the data when checking the patient again. The patient will also have interactively mediated data where they can see their progress and be motivated to continue their treatment.

3.1.4 Anamnesis

This section allows the patient to keep track of his medical history as well as his family medical history. The patient doesn't forget to mention important information once he is at the doctor's. As mentioned in one interview 8.4.3, it would be interesting if a patient could share anamnesis with other patients (primarily family relatives).

3.2 Corposa platform

Developing both a web application and a mobile application for our purposes offers numerous advantages. While the core functionality remains the same, the inclusion of both platforms caters to different user preferences and usage scenarios. A web application provides a versatile and accessible platform that can be accessed from any device with a web browser, allowing users to conveniently interact with the application using their desktops, laptops, or tablets. On the other hand, a mobile application offers the advantage of being specifically tailored for smartphones, providing a seamless and optimized user experience on-the-go. By having a dedicated mobile app, patients can enjoy the benefits of native device features and functionalities, such as push notifications and picture taking. Additionally, the web application's platform features extend its utility, enabling authorized users to efficiently manage and control the application's backend operations, user access, and content updates. This combination of a web application and a mobile application ensures a wider reach, enhanced user experience, and efficient administration, making it an advantageous approach for catering to diverse user preferences and facilitating effective application management.

3.2.1 Web application

The main point of interaction for patients will be the web application. It will contain the main functionalities that bring together the four pillars mentioned above. The app will be divided into two parts. One part is going to be for patients only and the second one is going to be for doctors and administrative purposes. The web application is intended as the main one, because it allows wider administration and the usage of advanced system functions.

3.2.2 Mobile application

The mobile application is mainly patient-oriented. The doctor only needs the mobile app to scan the patient's QR codes. As described here above, patients can benefit from application's push notifications serving as a task reminder, latest medical data input and picture taking using their personal mobile device regardless of where they are. It is also intended to enable using the Education module in order to provide verified and medically relevant information when needed.

3.3 Enterprise architecture

Our application is specifically designed for hospitals or similar medical facilities, necessitating specific characteristics and capabilities. First and foremost, it must possess robustness and reliability to handle a high workload consistently. The application should be able to effectively respond to various scenarios without crashing, even when faced with incorrect inputs. Additionally, scalability and modifiability are key aspects to consider. We refer to applications that possess these qualities as Enterprise Applications (EA) or enterprise software. While there is no formal definition, EA shares common characteristics. It is a sophisticated software platform intended for use within corporate or government environments. These applications are typically complex, scalable, and distributed, comprising multiple components based on enterprise architecture. EA solutions effectively address the challenges faced by enterprise organizations, prioritizing reliability, security, and robustness. Moreover, they often integrate or communicate with other EAs across different network technologies. Meeting stringent security requirements is a crucial aspect of EA, as they handle large data sets and operate within a demanding environment.

"Enterprise applications are about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data." - Martin Flower [13]

3.4 Multilayered vs multi-tiered architecture

Multilayered architecture, also known as n-tier architecture, separates the application into distinct layers, with each layer responsible for specific functionalities. Typically, the layers include presentation, business logic, and data access. This approach promotes modularity, scalability, and maintainability. Each layer operates independently, communicating with the layer above or below through well-defined interfaces. This architecture allows easier code reusability, flexibility in modifying or adding layers, and better separation of concerns.

On the other hand, multi-tiered architecture also referred to as a three-tier architecture, divides the system into three tiers or layers: presentation, application logic, and data storage. Each tier represents a physical or logical separation. The presentation tier handles the user interface, the application logic tier contains the business rules and processing logic, and the data storage tier manages the persistence and retrieval of data. This architecture emphasizes the physical distribution of components across different servers or systems, promoting scalability, fault tolerance, and performance optimization. Each tier can be independently scaled or replaced without affecting the other tiers. [13]

3.5 Containerization

Containerization is a technology that has revolutionized the way software is deployed and managed. This approach involves encapsulating applications and their dependencies into portable containers. By doing so, containerization enables consistent and reliable execution across diverse environments, eliminating the need for environment-specific configurations. The main containerization benefits include improved resource utilization, accelerated application deployment, scalability, and enhanced development productivity. That's why I propose encapsulating as many Corposa components as it is possible.

3.6 Data storage

For the data layer, I decided to use a database. In today's world, we have two main types of databases: relational (relation, SQL) and non-relational (NoSQL). Relational databases have a fixed structure in a table schema. NoSQL is a database concept that attempts to replace the table schema with other means. The goal is usually to create an optimized repository that is suitable for, for example, a large data load. In our case, there is no need for a non-relational database.

3.7 Entity relation diagram

An entity-relationship diagram (ER diagram) is a visual representation that depicts the relationships between different entities in a database system. It serves as a tool for designing and understanding the structure of a database, showing how entities are related to each other and how they interact.

The ER diagram helps in visualizing and analyzing the relationships and dependencies within a database system. It provides a clear understanding of the entities, their attributes, and the relationships between them, enabling efficient database design, query optimization, and data management. The ER diagram serves as a blueprint for database implementation and serves as a communication tool between developers, designers, and stakeholders involved in the database development process. [36]

When referring to the original er diagram, created in my bachelor's thesis, the goal was to extend the original er diagram as well as simplify the relations between entities.

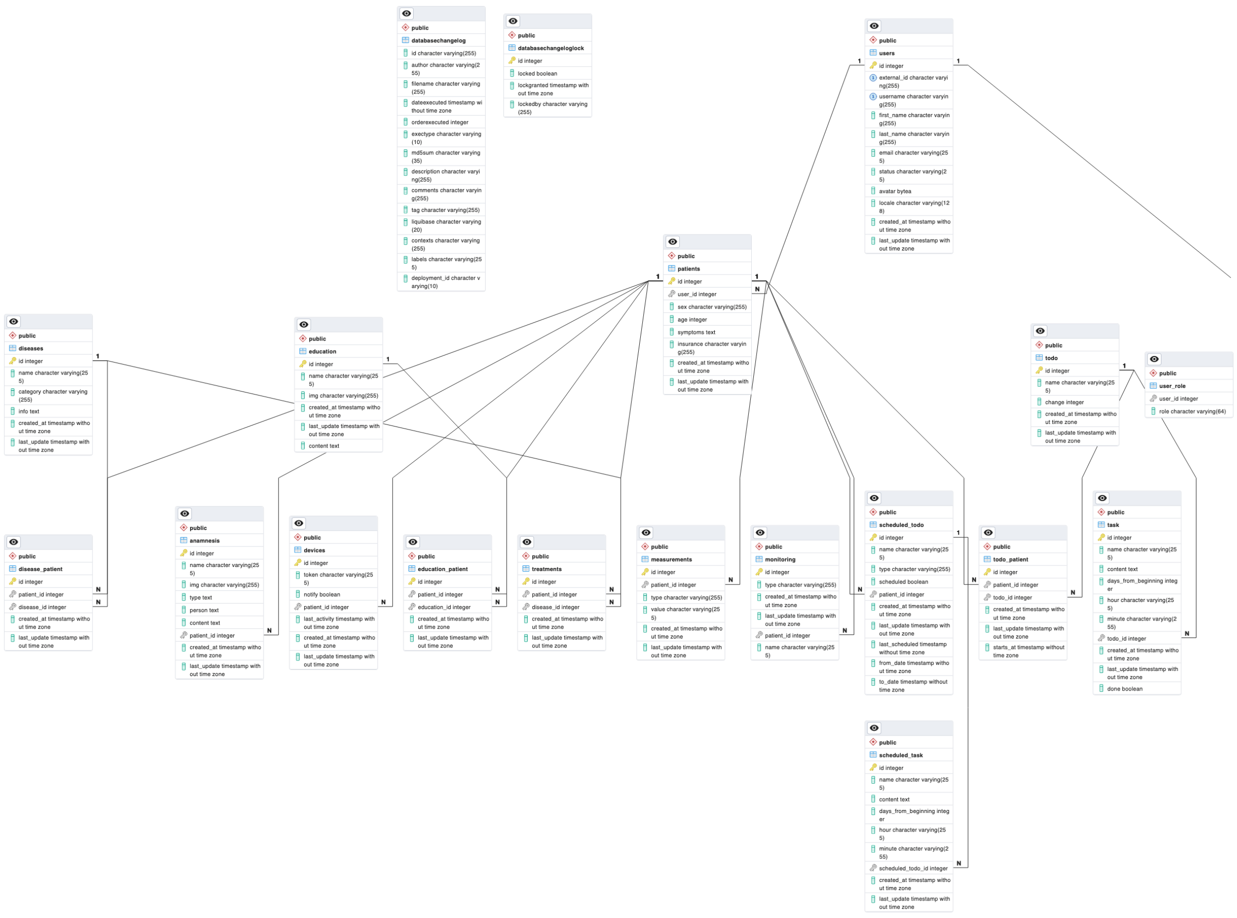

public
databasechangelog
id character varying(255)
author character varying(255)
filename character varying(255)
dateexecuted timestamp without time zone
orderexecuted integer
exectype character varying(10)
md5sum character varying(35)
description character varying(255)
comments character varying(255)
tag character varying(255)
liquibase character varying(20)
contexts character varying(255)
labels character varying(255)
deployment_id character varying(10)
public
databasechangeloglock
id integer
locked boolean
lockgranted timestamp without time zone
lockedby character varying(255)
public
users
id integer
external_id character varying(255)
username character varying(255)
first_name character varying(255)
last_name character varying(255)
email character varying(255)
status character varying(25)
avatar bytea
locale character varying(128)
created_at timestamp without time zone
last_update timestamp without time zone
public
patients
id integer
user_id integer
sex character varying(255)
age integer
symptoms text
insurance character varying(255)
created_at timestamp without time zone
last_update timestamp without time zone
public
diseases
id integer
name character varying(255)
category character varying(255)
info text
created_at timestamp without time zone
last_update timestamp without time zone
public
education
id integer
name character varying(255)
img character varying(255)
created_at timestamp without time zone
last_update timestamp without time zone
content text
public
todo
id integer
name character varying(255)
change integer
created_at timestamp without time zone
last_update timestamp without time zone
public
user_role
user_id integer
role character varying(64)
public
disease_patient
id integer
patient_id integer
disease_id integer
created_at timestamp without time zone
last_update timestamp without time zone
public
anamnesis
id integer
name character varying(255)
img character varying(255)
type text
person text
content text
patient_id integer
created_at timestamp without time zone
last_update timestamp without time zone
public
devices
id integer
token character varying(255)
notify boolean
patient_id integer
last_activity timestamp without time zone
created_at timestamp without time zone
last_update timestamp without time zone
public
education_patient
id integer
patient_id integer
education_id integer
created_at timestamp without time zone
last_update timestamp without time zone
public
treatments
id integer
patient_id integer
disease_id integer
created_at timestamp without time zone
last_update timestamp without time zone
public
measurements
id integer
patient_id integer
type character varying(255)
value character varying(255)
created_at timestamp without time zone
last_update timestamp without time zone
public
monitoring
id integer
type character varying(255)
created_at timestamp without time zone
last_update timestamp without time zone
patient_id integer
name character varying(255)
public
scheduled_todo
id integer
name character varying(255)
type character varying(255)
scheduled boolean
patient_id integer
created_at timestamp without time zone
last_update timestamp without time zone
last_scheduled timestamp without time zone
from_date timestamp without time zone
to_date timestamp without time zone
public
todo_patient
id integer
patient_id integer
todo_id integer
created_at timestamp without time zone
last_update timestamp without time zone
starts_at timestamp without time zone
public
task
id integer
name character varying(255)
content text
days_from_beginning integer
hour character varying(255)
minute character varying(255)
todo_id integer
created_at timestamp without time zone
last_update timestamp without time zone
done boolean
public
scheduled_task
id integer
name character varying(255)
content text
days_from_beginning integer
hour character varying(255)
minute character varying(255)
scheduled_todo_id integer
created_at timestamp without time zone
last_update timestamp without time zone

Figure 3.1: Entity relation diagram (Chen notation)

Chapter 4

Implementation

This chapter deals with the actual development and implementation process of the software application. Here below, I will be discussing the tools and frameworks used in the development process. By breaking down the application into its various modules and components, we explain the logic behind its functionalities. Furthermore, I will discuss the challenges encountered during the implementation phase and the strategies used to overcome them. Through this chapter, we aim to provide a comprehensive understanding of the technical aspects and intricacies involved in bringing the software application to life. The Corposa platform consists of one RESTful back-end, two front-end applications, one IAM, two databases, and AWS S3 file storage service. See figure 4.1.

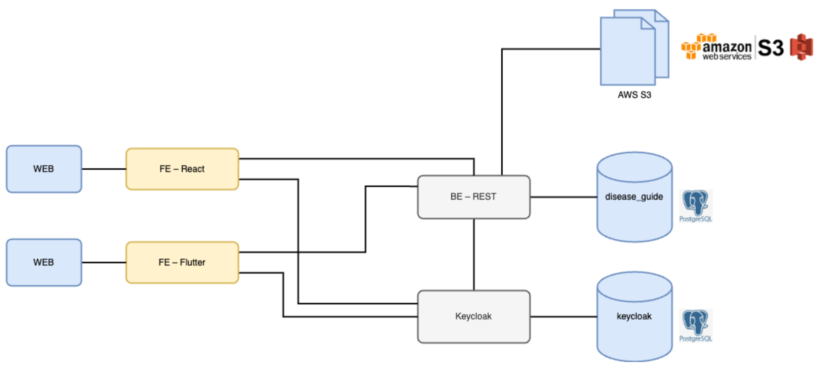


Figure 4.1: Technology diagram

4.1 BE - Java

The main technology used in this project is the Java programming language, specifically version 17. It was an easy choice for me. Java is still one of the most widely used languages in the world, even the most widely used according to the TIOBE index, especially in the enterprise applications sector. [41] It has a large community and is therefore well documented. Its syntax is extensive, but with a suitable editor, development is fast. More code brings order and quality type checking in return.

4.1.1 Spring boot

Spring is the most popular and widely used Java framework today[44]. Spring has very good documentation and common issues are discussed in detail in online discussions.

Spring boot is also a framework for enterprise application development. Unlike Spring, it focuses on code reduction and simplifies development. It is built as a stand-alone application. We don't need to deploy a WAR file and the application can run itself. It includes a Tomcat servlet server.[17]

4.1.1.1 Spring boot vs Java EE

Spring Boot and Java EE (Enterprise Edition) are two popular frameworks for building Java-based enterprise applications. Both frameworks have their own strengths and considerations, providing developers with different approaches to building robust and scalable applications.

Spring Boot, a part of the larger Spring framework ecosystem, offers a lightweight and opinionated approach to building applications. It focuses on simplicity, ease of use, and rapid development. Spring Boot provides an embedded server, auto-configuration, and dependency management, allowing developers to quickly bootstrap projects and get started with minimal configuration. The framework emphasizes convention over configuration, enabling developers to focus more on business logic rather than infrastructure concerns. Spring Boot also has extensive community support, a rich ecosystem of libraries, and excellent integration with other Spring modules, making it a preferred choice for building microservices and cloud-native applications. [17]

On the other hand, Java EE, now known as Jakarta EE, is a set of enterprise specifications and APIs provided by the Java Community Process (JCP). Java EE offers a comprehensive and standardized platform for building enterprise-grade applications. It provides a range of specifications and APIs for various services like persistence (JPA), messaging (JMS), web services (JAX-WS), and dependency injection (CDI). Java EE emphasizes portability and interoperability across different application servers and vendors. With Java EE, developers can rely on a well-defined set of standards and features, ensuring consistency and compatibility across different Java EE-compliant containers. This makes Java EE a suitable choice for larger, enterprise-scale applications that require robustness, stability, and adherence to industry standards.[3] As I stated earlier I used Java EE in my bachelor work and now I'm using Spring boot. I both used for a similar application. Therefore I can compare these two approaches quite comprehensively.

4.1.2 REST

REST stands for Representational State Transfer. It is an architectural style for building web services that communicate over the HTTP protocol. RESTful web services are designed to be simple, lightweight, and scalable, and are widely used for building APIs (Application Programming Interfaces) that allow different applications to communicate with each other.

At its core, REST is based on a set of constraints that define how resources should be represented and accessed via a uniform interface. These constraints include:

- Client-server architecture: The system is designed as a client-server architecture where the client sends requests to the server and the server responds with the requested data.
- Stateless: Each request to the server contains all the necessary information for the server to process the request. There is no need for the server to maintain any session state between requests.
- Cacheable: Responses to requests are cacheable to improve performance, and scalability and reduce the amount of traffic on the network.
- Layered system: A layered system allows for the scalability of the system by enabling the introduction of intermediate layers, such as caches or gateways, to manage the workload.

REST architecture style provides a set of principles for designing web services that are simple, scalable, and interoperable, making it an increasingly popular choice for building APIs and web services.

4.1.2.1 Scheduler

To generate todo lists and tasks from the scheduling tables I use Scheduler. It is scheduled to check what is planned for the next day and create it. See figure 4.2. I also use the scheduler to send notifications to the mobile application.

```
@Scheduled(cron ="0 0 17 * * ?")
public void cronJobAddTasks() {
   scheduledTodoService.scheduleTodo();
}
```

Figure 4.2: Scheduler

4.1.2.2 Interceptors

In order to send mobile notifications, I need to receive the device_token that is generated on the mobile device and keep track of it. One user can log in to multiple devices and each time the app is reinstalled it creates a new token. The solution I came up with is to send device_token in every request as a header field. On BE side I check for this header field in every request. When the device_token is included I first check whether it is in the database and if so I update the last_updated column to the current datetime and if not I insert a new row. You might ask why not specify it to one endpoint. The reason is the logged-in user can use different endpoints and we don't know which one to use. When I want to send a notification I query all devices for this specific user and choose only the ones I suggest as active (this can mean for example tokens with last_updated field not older than 30 days)

To check every request efficiently and look for the DeviceToken I use Interceptors similarly as I did in the mobile app. You can see the example below. Interceptors are good when you want to check the validation of some field in the header, log input data, authentication, and many other useful things. There are three main methods you can use: preHandle (checks request on its way to the controller), postHandle (action on response to client), and afterCompletion (after request and response). The method we want is preHandle because we want to check the header of the request sent from the client side. Then it's straightforward. I inject the service into the securityConfig class and update/insert the device token. [23]

```
@Slf4j
@Component
public class Interceptor implements HandlerInterceptor {

   private final DeviceServiceImpl deviceService;

   public Interceptor(DeviceServiceImpl deviceService) {
      this.deviceService =deviceService;
   }

   @Override
   public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
                                                   Object handler) {
      String deviceToken =request.getHeader("DeviceToken");

      if (deviceToken !=null) {
         log.info("Request intercepted: " +deviceToken);
         deviceService.updateTokenActivity(deviceToken);
      }

      return true;
   }
}
```

Figure 4.3: Interceptor class

```
@RequiredArgsConstructor
@Configuration
public class ConfigInterceptor extends WebMvcConfigurerAdapter {

   private final Interceptor interceptor;

   @Override
   public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(interceptor);
   }
}
```

Figure 4.4: ConfigInterceptor class

4.1.2.3 FCM integration

For sending push notifications I had to implement FCM [1] integration.

```
@Configuration
@EnableConfigurationProperties(FirebaseProperties.class)
public class FirebaseConfig {

    private final FirebaseProperties firebaseProperties;

    public FirebaseConfig(FirebaseProperties firebaseProperties) {
        this.firebaseProperties =firebaseProperties;
    }

    @Bean
    GoogleCredentials googleCredentials() {
        try {
            if (firebaseProperties.getServiceAccount() !=null) {
                try( InputStream is =
                                                         firebaseProperties.getServiceAccount().getInp
                                                         {
                    return GoogleCredentials.fromStream(is);
                }
            }
            else {
                // Use standard credentials chain. Useful when running inside GKE
                return GoogleCredentials.getApplicationDefault();
            }
        }
        catch (IOException ioe) {
            throw new RuntimeException(ioe);
        }
    }

    @Bean
    FirebaseApp firebaseApp(GoogleCredentials credentials) {
        FirebaseOptions options =FirebaseOptions.builder()
                .setCredentials(credentials)
                .build();

        return FirebaseApp.initializeApp(options);
    }

    @Bean
    FirebaseMessaging firebaseMessaging(FirebaseApp firebaseApp) {
        return FirebaseMessaging.getInstance(firebaseApp);
    }
}
```

Figure 4.5: Firebase configuration

[1] FCM – Firebase Cloud Messaging

```
public boolean sendTopicNotification(String title, String body) throws
                                                    FirebaseMessagingException {
    String topic ="dg";

    Message message =Message.builder()
            .setNotification(Notification.builder()
            .setTitle(title)
            .setBody(body)
            .build())
            .setTopic(topic)
            .build();

    String response =FirebaseMessaging.getInstance().send(message);

    System.out.println("Successfully sent message: " +response);

    return true;
}
```

Figure 4.6: Send notification method

4.1.2.4 File storage

When considering storage options for a Spring Boot application, the choice between local storage and AWS S3 (Simple Storage Service) depends on various factors and requirements.

Local storage, referring to a disk or file system on the server where the application is hosted, offers simplicity and ease of setup. It allows direct access to files, making it convenient for reading and writing data. Local storage is suitable for smaller-scale applications with limited storage needs or scenarios where data privacy and control are critical. It eliminates the need for external dependencies and reduces potential latency associated with network communication. However, local storage may have limitations in terms of scalability, redundancy, and disaster recovery. It requires careful management to ensure data backups and may not be as reliable as cloud-based solutions in terms of high availability and fault tolerance.

AWS S3, on the other hand, provides a highly scalable, durable, and secure cloud-based storage solution. With S3, data is stored across multiple servers, ensuring redundancy and high availability. It offers virtually unlimited storage capacity and provides seamless scalability as your application grows. S3 also includes features such as data versioning, access control, and lifecycle management. By leveraging S3's APIs, Spring Boot applications can easily interact with the storage service, allowing for efficient and reliable data storage and retrieval. Additionally, S3 offers various storage classes with different performance and cost characteristics, allowing you to optimize your storage strategy based on your specific needs. [22]

The final decision was based on scaling requirements 7.1.3. The application was supposed to be deployed on multiple VPS[2] and local storage didn't work this way because of the fact that we couldn't handle on which machine the files were saved.

[2]VPS – Virtual Private Server

4.2 Web FE - React

4.2.1 Choosing right UI

Choosing the right front-end technology is crucial for the success of your web application. In this section I will explore their strengths, weaknesses, and suitability for different types of projects.

JSF[3] is a Java-based web framework that focuses on server-side rendering and component-based architecture. On the other hand, React is a JavaScript library developed by Facebook, designed to build interactive user interfaces using a component-based approach.

JSF follows a more traditional approach, using Java and XML for defining components, which may require a steeper learning curve for developers. [39] React, being based on JavaScript, has a larger developer community and extensive documentation, making it easier to learn and develop with. [7]

JSF relies on server-side rendering, which can result in slower initial page load times compared to React's client-side rendering. React's virtual DOM and efficient rendering algorithms allow for better performance and responsiveness, especially for complex and dynamic user interfaces. React's scalability is also notable due to its ability to handle large-scale applications efficiently. [7]

JSF offers a rich set of pre-built UI components and an event-driven programming model. These components are tightly integrated with server-side logic, making it easier to handle complex application states and interactions. React, on the other hand, provides a flexible and reusable component model, allowing developers to build custom UI components and manage state efficiently. React's component-centric approach promotes code reusability and modularity.

JSF has been around for a longer time and has a mature ecosystem with a variety of third-party libraries, tools, and enterprise support. [39] However, React's ecosystem is larger and more active, with a vast collection of open-source libraries, tools, and an enthusiastic community. React's ecosystem also benefits from the wider JavaScript ecosystem, providing access to a broad range of resources.

JSF works effortlessly with Java-based back-end frameworks and libraries, as it is part of the Java EE (Enterprise Edition) stack. This integration makes it well-suited for enterprise applications that rely on Java-based services. React, being a JavaScript library, can work with any back-end technology through APIs and RESTful services, offering more flexibility in terms of back-end compatibility. [39]

Both JSF and React have their strengths and weaknesses, and the choice depends on the specific requirements of your project. JSF is well-suited for enterprise applications with complex server-side logic, while React is better in building interactive and performant user interfaces, especially for web and mobile applications.

Although it is very fast to get JSF up and running, the programmer can build the UI right away and with the right framework like PrimeFaces, he has a large number of components at his disposal. I decided to use React application SPA, because I have to create a REST interface for a mobile app, although the initialization of the project takes some time, so by creating simple components in JS FE the subsequent development is similarly simple.

[3]JSF – JavaServer Faces

4.2.2 SPA

A Single-page application (SPA) is a type of web application that follows a specific implementation approach. It initially loads a single web document and subsequently updates the content of the body within that document using JavaScript APIs. This architecture allows for a more dynamic and responsive user experience as the application can update specific sections of the page without requiring a full page reload. By leveraging JavaScript APIs, SPAs can fetch data from servers, interact with APIs, and update the user interface in real-time. This approach enhances performance and interactivity, providing a seamless and engaging user experience. [27]

4.2.2.1 Benefits

The primary benefit of using a Single-page application (SPA) as opposed to a regular website lies in its approach to downloading and rendering content. In an SPA, the entire HTML code, along with additional files like JavaScript and cascading stylesheets, is downloaded initially. However, when navigating between pages, the entire code and necessary files are no longer downloaded. Instead, the server is queried to retrieve specific parts of the page that need to be changed. This results in data savings as only the required content is fetched, while also reducing the browser's workload. By avoiding the need to redraw the entire page, SPAs deliver faster web responses, resulting in a more responsive and seamless user experience. The browsing experience is enhanced as the user perceives faster loading times and smoother transitions between different sections of the application.

4.2.2.2 Disadvantages

The disadvantage of SPA is directly dependent on JavaScript, but this can be disabled for the user, in some cases, this can be a major barrier to not building a website as SPA.

4.2.2.3 Conclusion

I chose to develop the application as a Single-page application (SPA) because it aligns with modern web development practices. Considering the nature of the application being developed, it is beneficial to only redraw specific parts of the website rather than the entire site when changes occur. This approach provides a more convenient user experience, as updates are seamless and do not require reloading the entire page. Additionally, the widespread use of JavaScript globally further supports the adoption of SPAs as it enables dynamic and interactive functionalities. I decided to choose React as a main framework.

4.2.3 Typescript

Typescript is a superset of JavaScript that adds optional static typing and other features to the language. It was developed and maintained by Microsoft and is designed to make large-scale JavaScript applications more manageable and scalable.

There are several reasons why developers may choose to use TypeScript:

- Type safety: TypeScript allows developers to specify types for variables, functions, and parameters, which helps catch errors during development, improves code quality, and makes it easier to maintain and refactor code.
- Tooling support: TypeScript has excellent tooling support, including code editors and IDEs, linters, and compilers, which help developers write and debug code more efficiently.
- Improved productivity: With features like optional chaining, nullish coalescing, and type inference, TypeScript can help reduce code verbosity and improve developer productivity.
- Compatibility: TypeScript is designed to be backwards-compatible with JavaScript, which means that developers can use existing JavaScript code and libraries in their TypeScript projects without having to rewrite them.
- Community and ecosystem: TypeScript has a large and growing community of developers who contribute to the language and create libraries and tools that make it easier to work with.

I choose TypeScript because I think it's a useful tool for developers working on large-scale JavaScript applications, as it adds useful features and improves code quality and maintainability.

4.2.4 Material design

I use the Material-UI framework for my React frontend development, benefiting from its robust set of pre-built components, theming capabilities, and seamless integration with React. Material-UI follows Google's Material Design guidelines, providing a visually appealing and consistent user interface. With Material-UI, I can quickly build responsive and mobile-friendly UI components, reducing development time and effort. The framework offers a wide range of customizable and reusable components, including buttons, forms, navigation elements, and data displays, enabling me to create a cohesive and intuitive user experience. Additionally, Material-UI's theming capabilities allow me to easily customize the visual aspects of the application to match branding requirements. It's comprehensive documentation and active community support provide valuable resources and assistance, ensuring a smooth and efficient development process.

4.3 Mobile FE - Flutter

Flutter is a cutting-edge framework that enables developers to create applications that work seamlessly across multiple platforms while achieving native-like performance. This is achieved by compiling the code into native ARM code, and for web pages, into JavaScript. The framework provides a vast range of visually appealing and modern components that require minimal additional code to add animations, resulting in sleek and responsive applications. [33]

Moreover, Flutter uses the Dart language, which is constantly updated with new features by its developer, Google. Dart is also designed for creating user interfaces, and as of version 2.12, it supports null safety, which ensures that undefined object references are checked. Additionally, Flutter offers many benefits for fast and efficient development, such as the ability to redraw the user interface without losing state or having to recompile the code. With its active community, the framework provides numerous libraries from both individual developers and major corporations. [33]

4.3.1 Multiplaform framework

There are several reasons why developers may choose to use a multiplatform framework like Flutter:

- Faster development: By using a single codebase, developers can create apps for multiple platforms, such as Android, iOS, web, and desktop, reducing development time and effort. [33]
- Consistent user experience: Flutter provides a set of customizable widgets that can be used across all platforms, ensuring a consistent user experience for all users.
- Native-like performance: Flutter's code is compiled into native ARM code, resulting in performance that is comparable to that of native applications.
- Hot reload: Flutter's "hot reload" feature allows developers to instantly see the changes they make to the code without the need to rebuild or restart the app, making development faster and more efficient. [33]
- Large community and ecosystem: Flutter has a large and active community of developers who contribute to the platform by creating plugins, packages, and tools, providing a rich ecosystem for developers to leverage in their app development. [33]

Using a multiplatform framework like Flutter can help developers create high-quality, visually appealing, and performance applications for multiple platforms in a more efficient and streamlined manner.

4.3.2 Implementation details

In this section, I would like to list implementation details that stand out and have brought difficulties in the development of the mobile app.

4.3.2.1 Dio

I use the Dio package in my Flutter mobile application to establish seamless communication with the RESTful backend. Dio provides a high-level API for making HTTP requests and handling responses, simplifying the integration of backend services into my app. With Dio, I can easily configure headers, parameters, and authentication tokens, ensuring secure and efficient data exchange. Its support for various request types, including GET, POST, PUT, and DELETE, allows me to perform a wide range of CRUD operations. Dio's built-in features, such as automatic JSON parsing and error handling, streamline the data retrieval process and facilitate efficient error management. Furthermore, Dio offers options for interceptors, enabling me to intercept and modify requests and responses as needed, enhancing flexibility and customization. [6]

4.3.2.2 State-full reactive

My mobile app is developed using the Flutter framework, which follows a stateful reactive architecture. Flutter embraces the concept of reactive programming, where changes in the application state trigger automatic updates to the user interface. By utilizing Flutter's reactive nature, I can build dynamic and interactive user interfaces that respond to user input and state changes in real-time. With Flutter's stateful widgets, I can manage and update the application state, ensuring that the UI reflects the latest data and user interactions. This reactive approach allows for a smooth and responsive user experience, as the UI adapts and updates dynamically based on the current state of the app. Flutter's stateful reactive architecture simplifies the development process and enables me to create highly interactive and engaging mobile applications. [25]

4.3.2.3 Observable object

In my Flutter mobile app, I use the Observable object pattern to efficiently manage and propagate changes to the application state. By using Observable objects, I can mark specific data models or classes as observable, enabling them to notify dependent widgets whenever their internal state changes. This allows for a reactive and efficient update of the user interface, ensuring that it reflects the latest data and maintains consistency. With Flutter's built-in state management solutions, I can easily wrap my observable objects and make them accessible throughout the app. This Observable object pattern simplifies the process of handling and synchronizing state changes, resulting in a more responsive and dynamic user experience for my Flutter mobile app.

4.3.2.4 Firebase

I integrate the Firebase platform into my Flutter mobile app to leverage its features, including Crashlytics, Analytics, and Firebase Cloud Messaging (FCM). Firebase Crashlytics provides real-time crash reporting, allowing me to identify and address issues promptly. With Crashlytics, I receive detailed crash reports, stack traces, and information about the affected users, enabling me to understand the root causes of crashes and prioritize bug fixes. Firebase Analytics helps me gain valuable insights into user behavior, app performance, and engagement metrics. By analyzing user interactions and tracking events, I can make data-driven decisions to optimize the user experience. Additionally, by utilizing FCM, I can implement push notifications and efficiently send targeted messages to my app's users, enhancing engagement and communication. Overall, integrating Firebase into my Flutter mobile app enhances its stability, enables data-driven decision-making, and improves user engagement through effective push notifications.

4.3.2.5 Notifications

To implement push notifications in my Flutter mobile app, I leverage Firebase Cloud Messaging (FCM) as part of the Firebase platform. With FCM, I establish a connection between my app and the Firebase servers to enable push notification functionality.

First, I configure my Flutter app to receive FCM tokens by integrating the firebase_messaging package. This package handles the registration process and generates a unique token for each device running the app. I store these tokens on my Spring Boot backend.

Next, in Spring Boot backend, I utilize the Firebase Admin SDK for Java to send push notifications. I create a notification payload, which includes the title, body, and other relevant information. Using the FCM API provided by the Firebase Admin SDK, I send the notification payload to the FCM server, targeting specific devices by their registered FCM tokens.

On the Flutter app side, I implement the handling of incoming push notifications. When a notification is received, the firebase_messaging package triggers a callback that allows me to extract the notification payload and display it to the user. I can customize the behavior of the app based on the received notification, such as navigating to a specific screen or updating the app's state.

By integrating Firebase Cloud Messaging and taking advantage of the Firebase Admin SDK in the backend, I can easily send push notifications to the Flutter mobile app. This enables effective communication with users, allowing them to stay updated and engaged with the app's content and features.

4.4 Keycloak IAM

As I was thought in school the best strategy you can have for managing your users credentials and authentication is to leave to already made solution. Thus I chose to implement in my platform an identity and access managment (IAM). The concrete solution I picked is Keycloak.

Keycloak is an open source Identity and Access Management (IAM) solution, developed by Red Hat. It provides a range of features for managing user authentication and authorization in modern applications and services. With Keycloak, developers can easily add secure user authentication to their applications, without needing to build authentication functionality from scratch. [28]

Keycloak allows for Single Sign-On (SSO) across multiple applications and can integrate with a variety of authentication protocols, including OpenID Connect, OAuth 2.0, SAML, and LDAP. It also provides features for managing user identities, such as user registration, password reset, and user profile management.

Overall, Keycloak is a powerful and flexible solution for managing user authentication and access in modern web and mobile applications. As an open source project, it is free to use and has a large and active community of contributors, making it a popular choice among developers.

4.4.1 Why IAM?

I already talked about it at the beginning of this section. It's not necessary to implement our own user management solution when there are plenty to choose from. But that's not the only reason. There are actually several reasons why one should consider using Identity and Access Management (IAM) solutions in your organization. [31]

- Improved Security: IAM solutions provide a secure way to manage user identities and access to resources. By implementing IAM, you can enforce strong password policies, set up multi-factor authentication, and control access to specific resources based on user roles and permissions.
- Simplified Management: IAM solutions provide a centralized way to manage user identities and access across multiple applications and services. This can greatly simplify the process of user onboarding, offboarding, and management, reducing the risk of errors and improving overall efficiency.
- Compliance: Many regulatory frameworks, such as GDPR, HIPAA, and PCI-DSS, require organizations to implement strong security controls for managing user identities and access. IAM solutions can help organizations meet these compliance requirements and avoid costly penalties and fines.[31]
- Scalability: As organizations grow and add more applications and services, managing user identities and access can become increasingly complex. IAM solutions provide a scalable way to manage user identities and access across multiple applications and services, without adding additional administrative burden. [31]

In the future, we can switch Keycloak to a more developed, paid solution.

4.4.2 keycloak vs Auth0

I didn't choose Keycloak straight away. First I look at the main stakeholders in the IAM field and tried to compare them to see which one suits best my needs. After basic research my decision process came to two solutions. One being Keycloak and one being Auth0.

Auth0 and Keycloak are tools designed for managing user authentication and access. Auth0 offers a comprehensive set of APIs and tools that allow for Single Sign On (SSO) and user management across multiple applications, with support for social, database, and enterprise identities. Keycloak, on the other hand, is an open-source solution that provides easy authentication and secure access to applications and services, without requiring the need to store or authenticate users manually. Both tools are useful for modern applications and services that require streamlined user management and authentication. [43]

I chose Keycloak over Auth0, because I didn't need advanced types of authentication including Google, Facebook etc and Auth0 free plan is limited to 7000 users, which I aim higher. Moreover, I find it easier to work with Keycloak when it comes to user management from my application. Keycloak has a simple API you can use. All you need is a service account to which you assign correct rights (user management). On the other hand, later when I was implementing the mobile app I found out that it would be much easier to use

Auth0 in this specific case because it has well prepared SDK ready to use and keycloak doesn't. Thus I had to implement my own solution, which includes getting access and refresh tokens, storing them inside mobile pref shared-storage, and checking the validity of tokens for every request. To be able to do that I used interceptors which are provided by the dio library.

4.4.3 Oauth2 and OpenID Connect

OAuth 2.0 and OpenID Connect are two popular authentication protocols used for secure authorization and authentication in modern web and mobile applications.

OAuth 2.0 is a protocol that allows third-party applications to access resources on behalf of a user, without requiring the user to disclose their credentials to the third-party application. It works by allowing the user to grant permission to the third-party application to access specific resources, such as a user's profile or photos. OAuth 2.0 uses access tokens to provide secure access to these resources, which are issued by the resource server after the user grants permission.

OpenID Connect (OIDC) is built on top of OAuth 2.0 and provides a standardized way for authentication and user identification. OIDC uses JSON Web Tokens (JWTs) to exchange information between the client, the authorization server, and the resource server. When a user logs in using OIDC, the identity provider (IDP) issues a JWT containing information about the user, which can be used to authenticate the user in subsequent requests. [30]

4.5 Liquibase

Liquibase is an indispensable tool that I have seamlessly integrated into my Spring Boot application to handle database migration tasks. With Liquibase, I can efficiently manage and track changes to the database schema, making it easier to evolve the application over time without disrupting existing data.

Using Liquibase with Spring Boot provides me with a standardized and structured approach to database migrations. I can define changes in a human-readable and version-controlled format (XML) and Liquibase takes care of applying these changes in a controlled and consistent manner. This ensures that my application and the underlying database are always in sync.

One of the key benefits of Liquibase is its support for rollbacks. If a migration encounters an error or needs to be reverted, Liquibase allows me to roll back the changes, ensuring the database remains in a consistent state. This rollback capability provides an added layer of reliability and helps mitigate risks associated with database changes.

Additionally, Liquibase seamlessly integrates with Spring Boot, leveraging its powerful dependency management and configuration capabilities. I can configure Liquibase to automatically apply migrations during the application's startup process, ensuring that the database is always up-to-date without any manual intervention. [38]

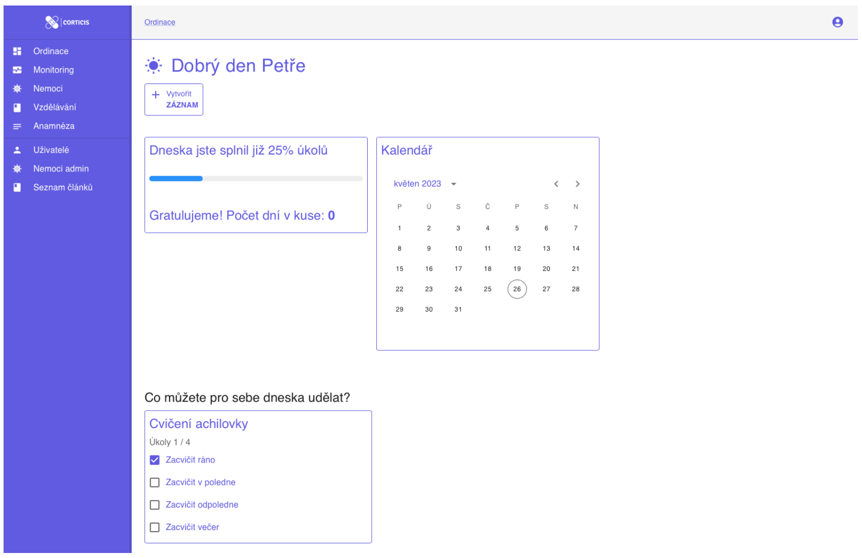
Chapter 5

Corposa

In this chapter, I will present the final results of the development of the Corposa platform, which has been developed based on the comprehensive analysis conducted in the analysis chapter 2 and the proposed solution outlined in the proposal chapter 3. This chapter represents the result of research, planning, and development efforts aimed at addressing the unique challenges and requirements. Corposa combines cutting-edge technologies, user-centric design principles, and robust functionality to deliver a compelling and intuitive web and mobile application that fulfill the envisioned objectives. Throughout this chapter, I will provide an in-depth overview of the key features and the overall outcome of the implementation, demonstrating how it successfully translates the analysis and proposal into a functional solution.

5.1 Web application

Starting with the web application, I will show its key features and explain their role in the system. It is advisable to distinguish the benefits and specifics of individual features in the web application versus in its mobile form in the sense of the different purposefulness of each of them, as we have already outlined in the chapter 3.



CORTICIS

Ordinace
Monitoring
Nemoci
Vzdělávání
Anamnéza
Uživatelé
Nemoci admin
Seznam článků

Ordinace

Dobrý den Petře

+ Vytvořit ZÁZNAM

Dneska jste splnil již 25% úkolů

Gratulujeme! Počet dní v kuse: **0**

Kalendář

květen 2023

P	Ú	S	Č	P	S	N
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Co můžete pro sebe dneska udělat?

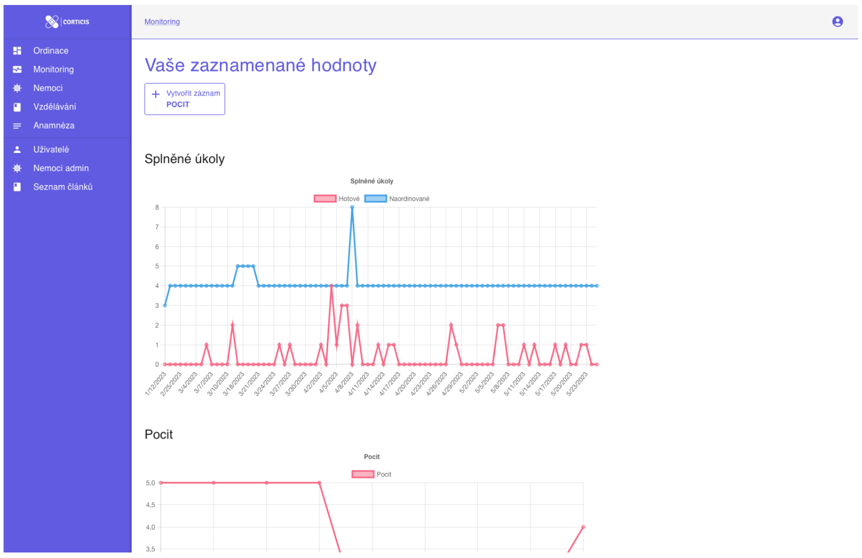
Cvičení achilovky

Úkoly 1 / 4

- [x] Zacvičit ráno
- [ ] Zacvičit v poledne
- [ ] Zacvičit odpoledne
- [ ] Zacvičit večer

Figure 5.1: Web application Dashboard/Disease guide

In figure 5.1, this page will load after filling in the login details (email address and password) on the previous page. It displays together basic information and parameters such as individual completed tasks or dates of checks or calendar operations.



CORTICIS

Ordinace
Monitoring
Nemoci
Vzdělávání
Anamnéza
Uživatelé
Nemoci admin
Seznam článků

Monitoring

Vaše zaznamenané hodnoty

+ Vytvořit záznam POCIT

Splněné úkoly

Figure 5.2: Web application Monitoring

On this page (see figure 5.2), a patient is enabled to enter requested data, e.g. physical wellness, blood pressure, heart rate, weight, oxygen saturation, body temperature, caloric values, pain levels and others if needed. These can be later evaluated compared in order to detect a possible change or progress.

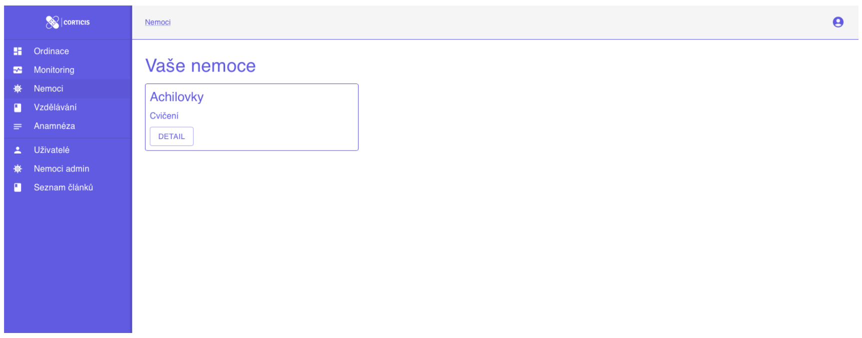


Figure 5.3: Web application Diseases

This section (see figure 5.3) provides a list of the patient's monitored diseases with basic information.

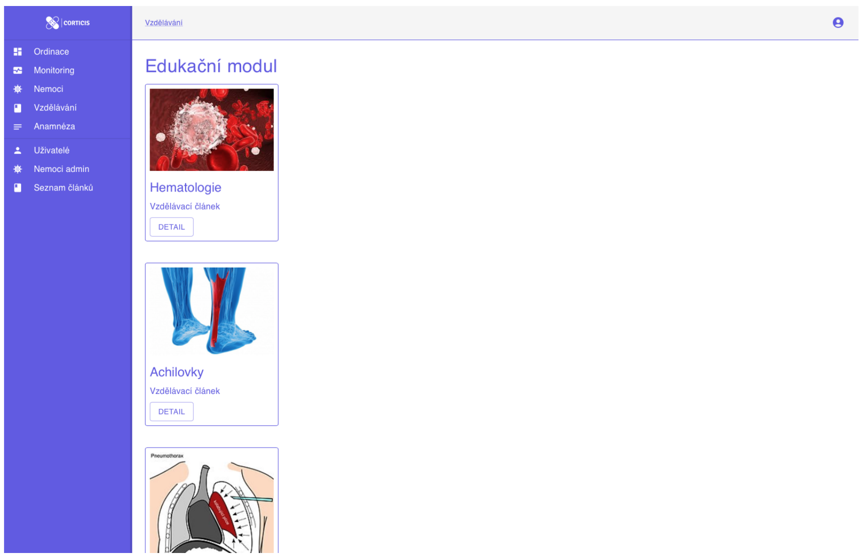
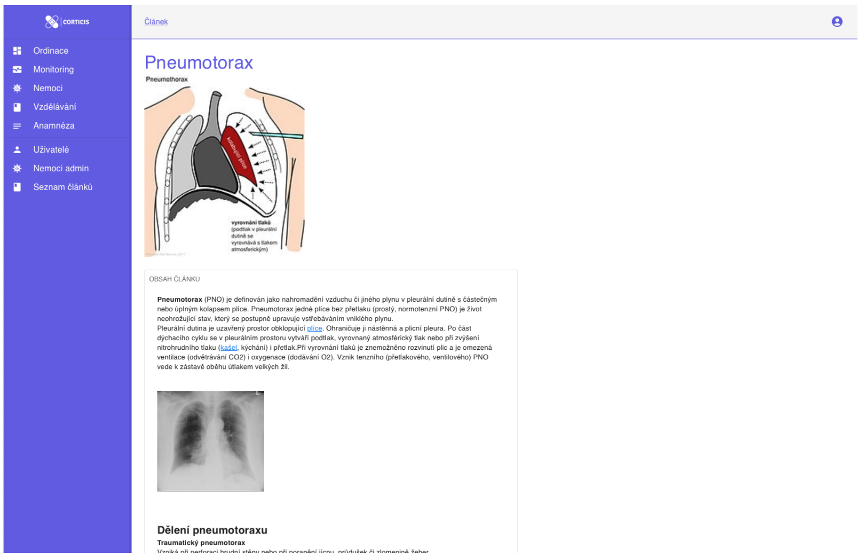


Figure 5.4: Web application Education module

As seen at figure 5.4, this page is dedicated to the topic of education on monitored diseases. Here the patient gets verified information provided by medical professionals relevant to the knowledge of his disease and the subsequent treatment set. All materials are prepared in such a way as to sufficiently inform the patient and at the same time to present this information as clearly as possible.



Pneumotorax

OBSAH ČLÁNKU

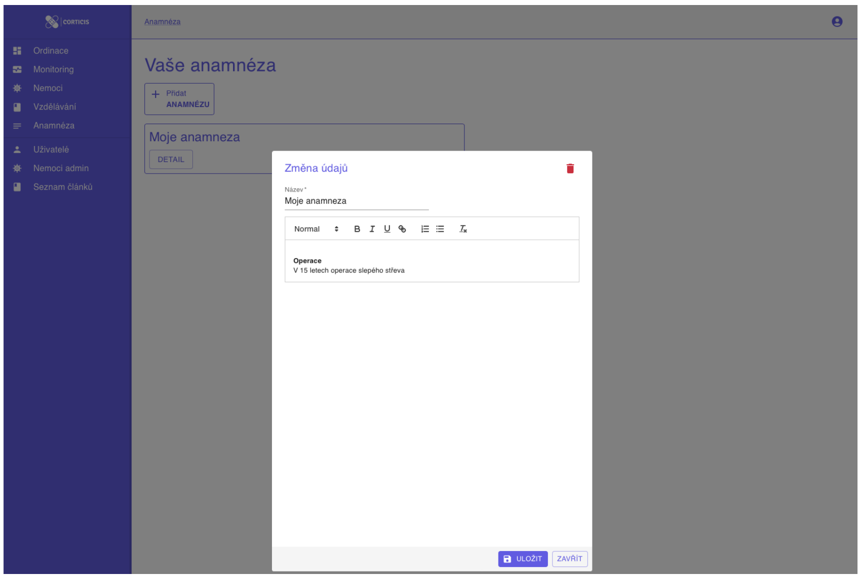
Pneumotorax (PNO) je definován jako nahromadění vzduchu či jiného plynu v pleurální dutině s částečným nebo úplným kolapsem plíce. Pneumotorax jedné plíce bez přetlaku (prostý, normotenzní PNO) je život neohrožující stav, který se postupně upravuje vstřebáváním vniklého plynu.
Pleurální dutina je uzavřený prostor obklopující plíce. Ohraničuje ji nástěnná a plicní pleura. Po část dýchacího cyklu se v pleurálním prostoru vytváří podtlak, vyrovnaný atmosférický tlak nebo při zvýšení nitrohrudního tlaku (kašel, kýchání) i přetlak.Při vyrovnání tlaků je znemožněno rozvinutí plic a je omezená ventilace (odvětrávání CO2) i oxygenace (dodávání O2). Vznik tenzního (přetlakového, ventilového) PNO vede k zástavě oběhu útlakem velkých žil.

Dělení pneumotoraxu

Traumatický pneumotorax

Figure 5.5: Web application Education module with particular disease

A detailed demonstration of a specific disease and how these data are processed in the material (see figure 5.5).



Vaše anamnéza

Moje anamneza

Změna údajů

Název *
Moje anamneza

Operace
V 15 letech operace slepého střeva

Figure 5.6: Web application Anamnesis

This detail (see figure 5.6) provides an example of patient's anamnesis. Multiple history records can be created for different family members.

5.2 Mobile application

The following demonstrations will show the final product of the mobile application. The individual functions and their benefits in terms of the requirements set out in the previous chapters will be shown in more detailed way.

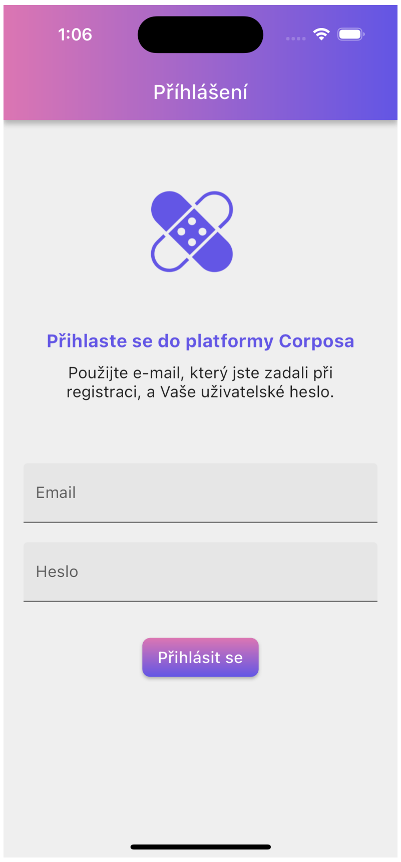
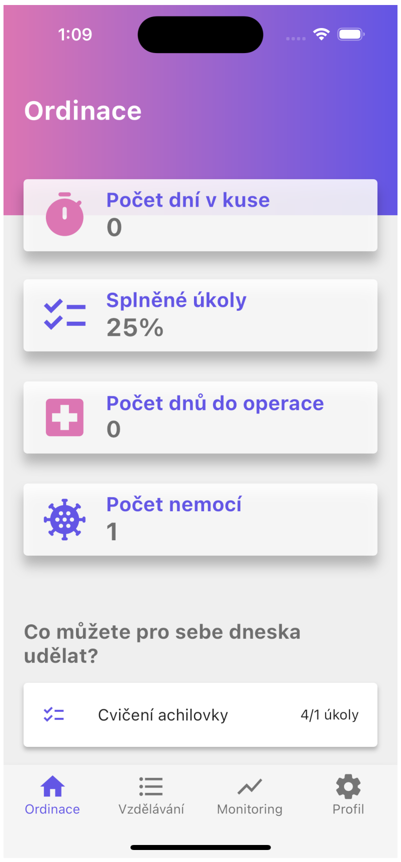

1:06
Přihlášení
Přihlaste se do platformy Corposa
Použijte e-mail, který jste zadali při registraci, a Vaše uživatelské heslo.
Email
Heslo
Přihlásit se

Figure 5.7: Mobile application login page

In order to log into the Corposa application, a user enters the email address entered during registration process and the password set (see figure 5.7).



1:09

Ordinace

Počet dní v kuse
0

Splněné úkoly
25%

Počet dnů do operace
0

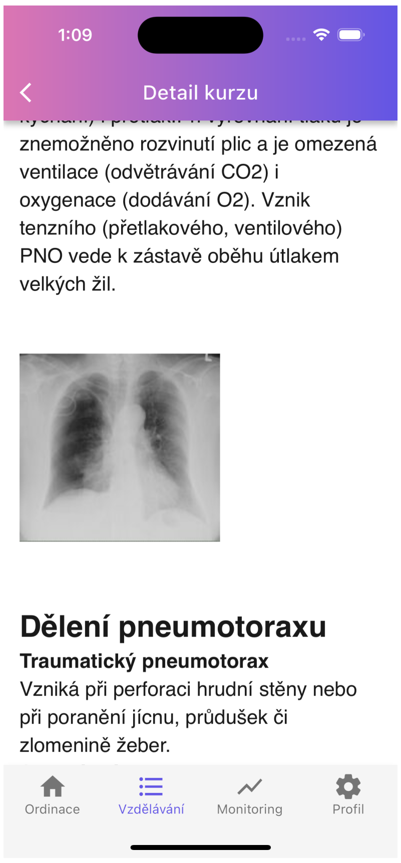
Počet nemocí
1

Co můžete pro sebe dneska udělat?

Cvičení achilovky 4/1 úkoly

Figure 5.8: Mobile application Dashboard/Disease guide

As seen at figure 5.8), several parameters and information are placed in this section for general patient information purposes. The patient can see how many days the treatment plan has been in progress, how many tasks have been completed, the number of days to scheduled surgery if any, and the current number of diseases being monitored.



znemožněno rozvinutí plic a je omezená ventilace (odvětrávání CO2) i oxygenace (dodávání O2). Vznik tenzního (přetlakového, ventilového) PNO vede k zástavě oběhu útlakem velkých žil.

Dělení pneumotoraxu

Traumatický pneumotorax

Vzniká při perforaci hrudní stěny nebo při poranění jícnu, průdušek či zlomenině žeber.

Figure 5.9: Mobile application Education module

A section of the application dedicated to the topic of education on monitored diseases. Here the patient gets verified information relevant to the knowledge of his disease and the treatment set. He can thus understand the set treatments and other actions (including the need for possible surgery) in a broader context. The content of the module is prepared in collaboration with professionals in order to be factual, informative and comprehensible. Images or video demonstrations may be embedded as part of the text (see figure 5.9).

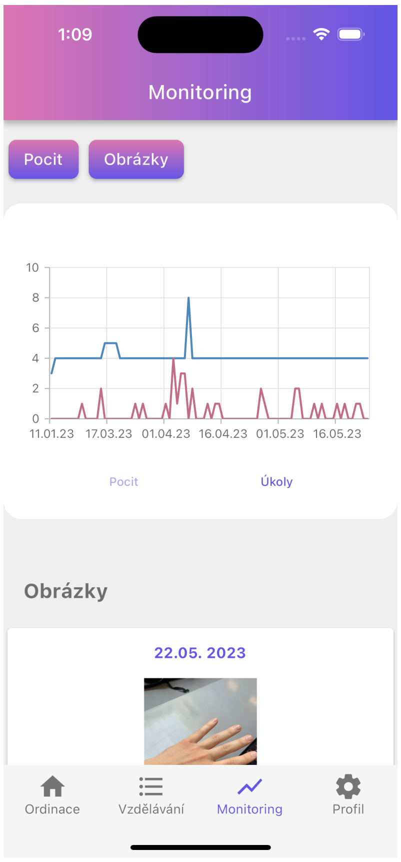
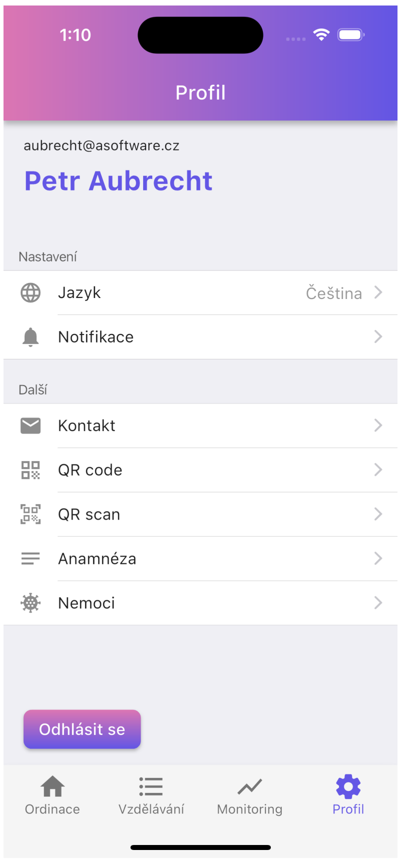


Figure 5.10: Mobile application Monitoring

A section enabling the patient to enter requested data, e.g. blood pressure, heart rate, weight, oxygen saturation, body temperature, caloric values, pain levels and others if needed. While using the mobile app, the patient can also upload photos of his body parts in case a wound after surgery is being monitored. All data are date and time stamped when entered, allowing subsequent comparison. It is also possible to trace the progress or another trend when changes in these data are detected (see figure 5.10).


aubrecht@asoftware.cz

Petr Aubrecht

Jazyk Čeština

Figure 5.11: Mobile application Profile

This section displays profile information and settings for using the mobile app, such as push notifications, interface language, QR code display and scanning, and the anamnesis section (see figure 5.11).

Chapter 6

Deployment

Part of my thesis is to get the application up and running in a production-like environment. In this chapter, I describe the approach I took for the deployment and its evolution as mainly scaling-related problems described in chapter 7 occurred. Encapsulating most of the components into containers for easy startup and handling is critical to the success of this chapter. In order to give the application a place to live, a VPS server had to be ordered. As the demands of the application increased, so did the number of VPS servers. All servers run on the Linux operating system with Ubuntu or Debian distributions.

6.1 Dockerization

As I already mentioned, the key to the success of this chapter is to containerize as many components as possible. To accomplish this, I will use Docker technology. It will streamline the deployment and execution of the components of Corposa platform. By containerizing each component using Docker, I encapsulate its dependencies, libraries, and configuration within a self-contained unit. This approach allows me to easily package and distribute the components as Docker images, ensuring consistency across different environments. [42] With Docker, I can quickly spin up instances of the platform's components on any machine that has Docker installed, without worrying about compatibility issues or manual setup. Dockerization simplifies the process of deploying, scaling, and managing the platform, offering portability, flexibility, and reproducibility. It enables me to efficiently run the various components of the Corposa platform, facilitating seamless collaboration and smooth operations. [14]

The components I was able to dockerize were the Postgres database, Keycloak IAM, and the BE together with web FE. I also manage to dockerize performance tests 8.1.2. The only thing that is not running via Docker is Nginx web server that is used as a reverse proxy and mobile app, obviously. In our case, Docker helps us to containerize components and provides a common network bridge for different applications to communicate with each other.

6.1.1 One instance

To tackle the scaling approach 1 7.1.1 I created dockor-compose.yml file, in which I defined the main application container as well as the database container. The IAM is defined separately but also under docker. Networking is set up as you can see in figure 6.1. I also use Nginx as a proxy for the incoming traffic 6.2.

```
version: '3'

services:
  disease_guide:
    build: ../..
    container_name: disease_guide
    environment:
      - DB_SERVER:db
      - DB_URL=jdbc:postgresql://db:5432/disease_guide
      - DB_USERNAME=postgres
      - DB_PASSWORD=postgres
    ports:
      - 8000:8000
    depends_on:
      - db
  db:
    image: "postgres:9.6-alpine"
    container_name: db
    restart: always
    ports:
      - 5433:5432
    environment:
      - POSTGRES_DB=disease_guide
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
volumes:
  postgres_data:
```

Figure 6.1: dockor-compose.yml for one instance

```
    server {
        server_name corposa.cz www.corposa.cz;

        location /{
            proxy_pass http://127.0.0.1:8000/;
        }
    }
```

Figure 6.2: Nginx proxy pass

6.1.2 Adding multiple instances on single VPS

To tackle the scaling approach 2 7.1.2 I created two new docker-compose.yml files, one for the database and one for the application. To set up the networking again I had to add the networks section. See figure 6.3.

```
version: '3'

services:
  disease_guide:
    build: ../..
    container_name: disease_guide
    ports:
      - 127.0.0.1:8000:8000
    volumes:
      - /home/disease-guide/data/:/home/data/
    environment:
      - DB_URL=jdbc:postgresql://postgres:5432/disease_guide
      - DB_USERNAME=postgres
      - DB_PASSWORD=postgres
      - DOC_LOCAL_STORAGE_FOLDER=/home/data
      - KC_REALM=disease-guide
      - KC_AUTH_SERVER=https://auth.corposa.cz
      - KC_SERVICE_CLIENT_SECRET=secret
      - KC_ISSUER_URI=https://auth.corposa.cz/realms/disease-guide
      - KC_CLIENT_ID=disease-guide-api
      - KC_GUI_CLIENT_ID=disease-guide-gui
      - KC_SERVICE_CLIENT_ID=disease-guide-api
      - CRON_BASIC_JOB=0 *** * ?
      - CRON_ADD_TASKS=0 *** * ?

networks:
  default:
    external: true
    name: corposa-net
```

Figure 6.3: docker-compose.yml networking

6.1.2.1 Nginx as Loadbalancer

I had to modify Nginx as a load balancer in my infrastructure to efficiently distribute incoming traffic across multiple backend instances. See figure 6.4. By configuring Nginx with a basic load-balancing strategy, I can evenly distribute requests among the available backend instances, ensuring optimal utilization of resources and improved application performance. Nginx effectively acts as a traffic manager, efficiently routing incoming requests to the appropriate backend instance based on predefined rules. [26] This setup was supposed to achieve better scalability, high availability, and improved fault tolerance by effectively distributing the workload across multiple server instances. From the performance testing section 8.1.5 we now know it did not resolve in better performance as discussed in chapter 7.

```
upstream dg-load-balancer {
    server 127.0.0.1:8001 weight=1;
    server 127.0.0.1:8002 weight=1;
    server 127.0.0.1:8000 backup;
}

server {
    server_name corposa.cz www.corposa.cz;

    location /{
        proxy_pass dg-load-balancer;
    }
}
```

Figure 6.4: Nginx as load balancer

6.1.3 Multiple instances on multiple servers

To tackle the third scaling approach 7.1.3 I had to buy external VPS servers and deploy back-end instances on them. To direct traffic I had to modify the Nginx again 6.5.

```
upstream dg-load-balancer {
    server 194.182.90.202:8080 weight=2;
    server 194.182.78.107:8080 weight=2;
    server 127.0.0.1:8000 weight=1;
}

server {
    server_name corposa.cz www.corposa.cz;

    location /{
        proxy_pass dg-load-balancer;
    }
}
```

Figure 6.5: Nginx as a load balancer outside localhost

6.1.3.1 DockerHub

Another challenge I faced during the implementation of this architecture was to efficiently distribute application instances to other servers. The solution was in uploading my built application image to the DockerHub, enabling easy accessibility and deployment on various platforms. [37] Once I build the Docker image locally on the main VPS, I push it to DockerHub, a cloud-based container registry. In the future, it is possible to upload this image to the GitLab registry as part of CI/CD[1] process instead of DockerHub. By uploading the image to DockerHub, I can securely store and share it with others. This allows me to conveniently download the image from DockerHub whenever I need to deploy my application on different environments or servers.

[1] CI/CD – Continuous integration/Continuous delivery

6.1.3.2 File storage

With distributing app instances on different servers the problem of file storage occurred. I was no longer able to store files locally since the traffic was split between instances randomly. The solution to this trouble was to implement AWS S3 file storage. Detail implementation is discussed in section 4.1.2.4 of chapter 4. This way it doesn't matter how many instances exist and where they run.

6.1.4 Adding mobile app

At this point, we are close to the finish line. Up until now, we were not taking the mobile into consideration. The mobile app deployment process is difficult and it was not part of this thesis to implement an efficient deployment strategy. However, it's important to take it into consideration as it brings a significant workload to the back-end application.

6.1.5 Final architecture

The final architecture consists of three virtual servers. On the main server, there is IAM component (Keycloak), database component (Postgres) and disease guide BE component (Spring boot REST API). All three components are fully dockerized. As a web server, I use Nginx which works as a load balancer as well and splits the traffic between instances. In addition to the main server, we have two external servers, and on each of them, there is another instance of the disease guide BE component running. All three artifacts of BE communicate with the IAM component, database component, and AWS S3 file storage service. FE components, the react web application, and the mobile application either in a web browser or on a mobile device. For details see the deployment model in figure 6.6.

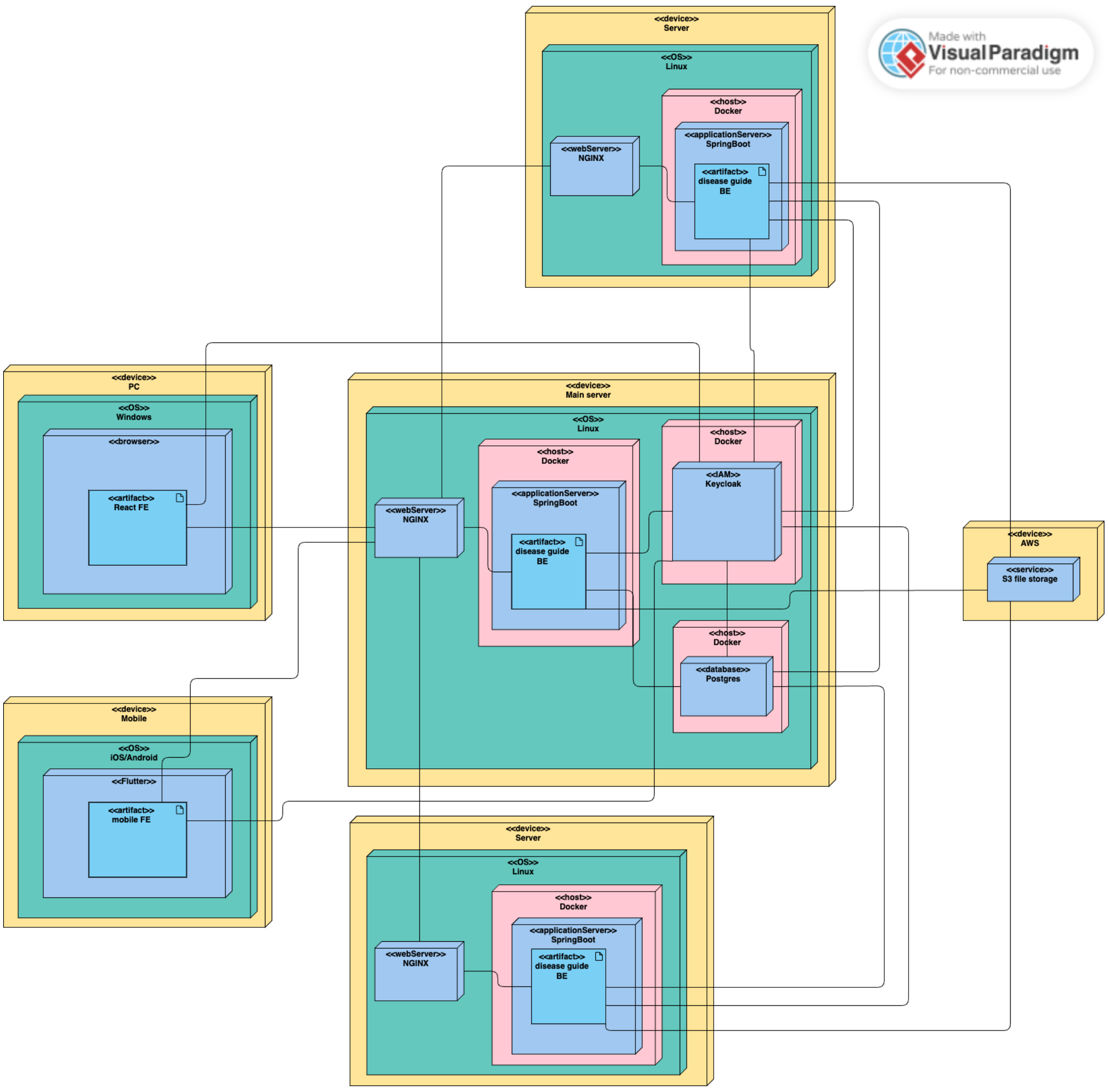


Figure 6.6: Deployment diagram

6.2 Mobile app

I was able to deploy the mobile application to several physical devices. Since I am using the Flutter framework I am able to compile my application to both iOS and Android. Unfortunately, I wasn't able to test the application on an Android device. All my test users use iOS. The deployment of the application is not automated, because in order to do so, you need a developer account, which I didn't have.

6.3 Monitoring

6.3.1 New Relic

For monitoring my VPS, I use New Relic. New Relic is a cloud-based platform for monitoring and managing application performance, user experience, and business metrics. [40]

- New Relic provides real-time insights into application performance, allowing developers and IT operations teams to quickly identify and resolve issues before they impact end-users. Its platform includes features such as:
- Application Performance Monitoring (APM): New Relic provides detailed performance metrics for applications running on a variety of platforms and languages, including Java, .NET, Ruby, Node.js, and more.
- Infrastructure Monitoring: New Relic can monitor servers, containers, and cloud infrastructure, providing visibility into the health and performance of the underlying infrastructure that supports applications.
- Synthetic Monitoring: New Relic can simulate user interactions with applications, providing insights into the user experience and identifying potential issues before they impact end-users.
- Business Metrics: New Relic can track key business metrics, such as conversion rates and revenue, providing insights into the impact of application performance on the business.

To be the first one to know when something bad is happening on my servers I set up an email notification that is triggered when resources on my VPS to high. It was proven to work when my VPS was under attack 6.4 and I immediately received an email notification. Thanks to that I was able to tack action 6.4.1 and prevent further damage.

The New Relic monitoring was helpful when I perform the distributed performance test 8.1 because I was able to see what is going on on my VPS during the test.

6.3.2 Docker

The "docker stats" command is a valuable tool that I use to monitor the resource utilization and performance of containers within my Docker environment. This command provides real-time insights into the CPU, memory, and network usage of each running container, enabling me to assess their load and identify potential bottlenecks or resource-intensive processes. By executing the "docker stats" command, I gain a comprehensive overview of the containers' resource allocation and consumption, helping me optimize their efficiency and ensure optimal performance. This command proves particularly useful in scenarios where I need to closely monitor the health and stability of my containerized applications, enabling proactive identification and resolution of varius issues. [10]

6.3.3 Container logs

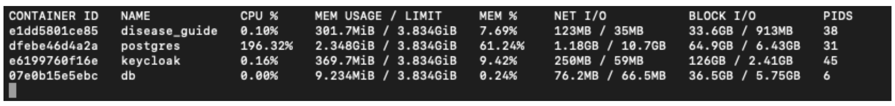
The "docker logs" command plays a crucial role in enabling efficient log management and troubleshooting in a production environment. By this command, I gain access to the logs generated by containers running within my Docker environment. This provides valuable insights into the application's behavior, error messages, and system events, helping me identify and address issues promptly. The "docker logs" command allows me to view logs from individual containers or multiple containers simultaneously, simplifying the process of monitoring and troubleshooting across different components of my application. [9] It is particularly useful in production environments where real-time visibility into container logs is essential for diagnosing and resolving issues promptly, ensuring the smooth operation of my application.

6.3.4 Mobile application

In order to gain insights into the performance and behavior of my Flutter app, I rely on Firebase Crashlytics and Firebase Analytics. These powerful tools provided by Firebase offer capabilities for monitoring and analyzing the app's performance on mobile devices. Firebase Crashlytics enables me to track and analyze app crashes and errors in real time, providing detailed crash reports that help me pinpoint the root causes of issues. By integrating Crashlytics into my app, I receive comprehensive crash logs and stack traces, allowing me to quickly identify and prioritize the most critical issues affecting user experience. This helps me proactively address and resolve any stability issues, ensuring a seamless and reliable app experience for users. [21]

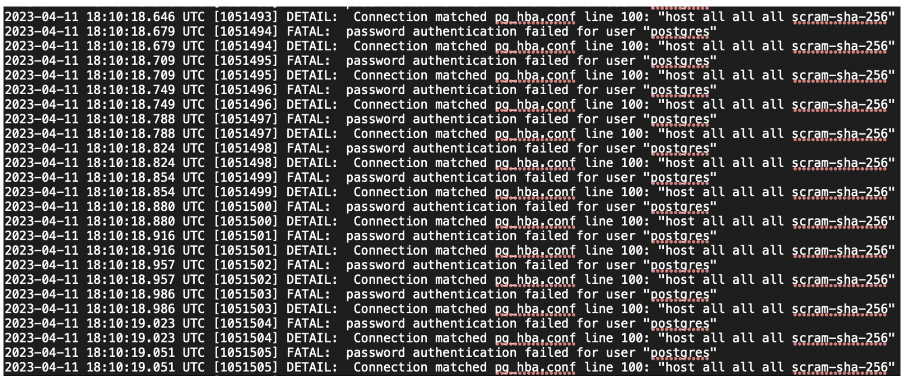
6.4 Attack

During the implementation, I encountered an attack. The attack was targeted at my database. I realized that the attack is happening when I got notified from New Relic that my memory usage is almost drained out, which was weird because at that time I was the only one who was using the app and the idle memory usage was around 5%, and now it was over 95%. So I began my investigation. First I check, whether any of my containers are in higher use. And the results were interesting. As you can see in figure 6.8, the Postgres container was consuming all the VPS resources. Then I checked the logs of this affected container. See figure 6.8. Most likely someone tried to crack the password on the known port 5432 and get into the database.



```
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %    NET I/O           BLOCK I/O         PIDS
e1dd5801ce85   disease_guide   0.10%     301.7MiB / 3.834GiB   7.69%    123MB / 35MB      33.6GB / 913MB    38
dfebe46d4a2a   postgres        196.32%   2.348GiB / 3.834GiB   61.24%   1.18GB / 10.7GB   64.9GB / 6.43GB   31
e6199760f16e   keycloak        0.16%     369.7MiB / 3.834GiB   9.42%    250MB / 59MB      126GB / 2.41GB    45
07e0b15e5ebc   db              0.00%     9.234MiB / 3.834GiB   0.24%    76.2MB / 66.5MB   36.5GB / 5.75GB   6
```

Figure 6.7: Docker stats results



```
2023-04-11 18:10:18.646 UTC [1051493] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.679 UTC [1051494] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.679 UTC [1051494] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.709 UTC [1051495] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.709 UTC [1051495] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.749 UTC [1051496] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.749 UTC [1051496] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.788 UTC [1051497] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.788 UTC [1051497] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.824 UTC [1051498] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.824 UTC [1051498] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.854 UTC [1051499] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.854 UTC [1051499] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.880 UTC [1051500] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.880 UTC [1051500] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.916 UTC [1051501] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.916 UTC [1051501] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.957 UTC [1051502] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.957 UTC [1051502] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:18.986 UTC [1051503] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:18.986 UTC [1051503] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:19.023 UTC [1051504] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:19.023 UTC [1051504] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
2023-04-11 18:10:19.051 UTC [1051505] FATAL:  password authentication failed for user "postgres"
2023-04-11 18:10:19.051 UTC [1051505] DETAIL:  Connection matched pg_hba.conf line 100: "host all all all scram-sha-256"
```

Figure 6.8: Postgres container logs

6.4.1 Solution

I changed my Postgres password from postgres to something more secure one. Then I wanted to hide the main port 5432. The problem was the external VPS where using it and in addition to that I was through this port connected to DataGrip. So to hide this port I had to block it in firewall UFW. To connect from DataGrip I used SSH tunnel. For external VPS I configured the Nginx stream so the TCP traffic on port 9876 was passed to the local port 5432 and in addition to that I restricted only external VPS ip addresses to be allowed to connect to port 9876. [45] See figure 6.9.

```
stream {
    server {
        listen 9856;

        allow 192.182.90.202;
        allow 194.182.78.107;
        deny all;

        proxy_connect_timeout 60s;
        proxy_socket_keepalive on;
        proxy_pass localhost:5432;
    }
}
```

Figure 6.9: Configuration of nginx.conf

6.5 Backups

To ensure the safety and integrity of the main database, I have implemented a daily backup strategy using the cron utility. See figure 6.10 Cron allows me to schedule regular backups of the PostgreSQL database, providing an automated and reliable solution. With the help of a cron job, a specific script is executed at a predetermined time each day, initiating the backup process. The script utilizes PostgreSQL's built-in backup tools to create a snapshot of the database, capturing all the necessary data and ensuring data consistency. I can maintain a consistent backup schedule thanks to cron, mitigating the risk of data loss and enabling the restoration of the main database to a previous state if needed. This approach provides peace of mind and allows for the seamless recovery of critical data in case of any unforeseen events or database-related issues. [15]

```
crontab -e

0 2 * * * pg\_dump -U postgres disease\_guide >/opt/backups/disease\_guide.sql
```

Figure 6.10: DB backups every day at 2AM [15]

Chapter 7

Scaling

The main goal of scaling was to create the application so that it could reflect the growth of users without major interference to the application itself, but at the same time without unnecessarily implementing a complex architecture in the sense of microservice that would not bring the benefit.

Monolith applications offer several advantages over microservices architecture, making them a suitable choice for certain scenarios. In the case of my application, which consists of a Spring Boot RESTful backend and a React frontend, a monolith architecture provides simplicity and ease of development. With a monolith, the entire application is contained within a single codebase, simplifying deployment and reducing complexity. The tight integration between the backend and frontend components enables efficient communication and reduces overhead. Additionally, a monolith architecture eliminates the need for network communication between services, improving performance. Moreover, debugging and testing the application becomes more straightforward due to the centralized nature of a monolith. While microservices offer scalability and flexibility advantages for larger and more complex systems, a monolith approach is ideal for smaller applications where simplicity and development speed are prioritized. [16]

7.1 Scaling strategies

To ensure we meet non-functional requirements that focus on scalability, we will look at different scaling strategies and compare them. The goal is to have at least 1 500 active users that are using the application without problems. The requirements are specified in the section 2.1.2 of the chapter 2.

- One instance
- Multiple instances on one machine
- Multiple instances distributed

7.1.1 One instance

This strategy consists of one instance of the app, one database, and one IAM. The nginx serves only as a reverse proxy.

7.1.2 Multiple instances on one machine

This strategy aims to improve application performance by splitting the workload among multiple instances on one machine. Even though we were splitting the traffic between multiple instances it didn't bring the desired improvements over the scaling strategy 1 and so it didn't meet the non-functional requirements.

7.1.3 Multiple instances distributed

The last strategy builds on strategy 2 however, it deploys the other instances on different virtual servers. From performance testing result 8.1.5, we see that this strategy fulfills the non-functional requirements, thus it is suitable for our purposes.

7.1.4 Future

The first strategy was simple and functional, however, it didn't allow further scaling and it already failed to meet non-functional requirements. The second strategy turned out to be more complex without bringing any improvements, on the contrary, it had even worse results. It makes sense because we are still limited by one machine. The last strategy turned out to be sufficient and met all our non-functional requirements. With a little bit of work, we can add instances to external servers and ease the workload of already running instances. The limitations of this solution lie in what the database can handle, as it represents the bottleneck of the whole architecture. Once the database reaches its limits, this solution stops working. In this scenario, it is recommended to implement master and slave replication. This way each application instance would have its own database and the scaling potential would increase. [2]

Chapter 8

Quality assurance and evaluation

Each application should have tests at several levels. During development, they are mainly used to quickly verify that the whole application is functional and that we haven't accidentally broken another part when implementing a new functionality. We have several types of tests, which are divided according to the scope of the test. [12] A common practice is to include tests in a Continuous Delivery (CD). Since I am building a layered enterprise web application where most of the methods only write or read from the database, I didn't pay as much attention to unit testing and rather focused on performance testing and user testing. However, I first started with automated tests of the REST interface. I wanted to make sure that every time I changed something, no endpoint would break. I run the tests over the test database I have in the docker, so the implementation of this test is simple and I know what endpoints return each time. Next, I tested the user interface of the main web application using Playwright. I have created five test scenarios that cover all key processes of the application. The main topic was, however, performance testing. I used the Locust library and gradually played with scalability. After finishing the implementation I handed the application over to 6 users from the family, medics, and doctors for testing. They tested both the web application and the mobile application.

8.1 Distributed performance testing

Let's start with performance testing. REST performance testing is crucial in evaluating the scalability and efficiency of Spring Boot REST applications. One tool that I rely on for this purpose is Locust. Locust is an open-source, Python-based framework that simplifies the process of simulating large-scale user loads and measuring the performance of RESTful APIs.[32]

By using Locust, I can create and execute performance tests that accurately mimic real-world user behavior from small to large scale. With Locust's intuitive syntax and flexible API, I can define user scenarios and simulate concurrent requests. This allows me to identify potential performance bottlenecks, measure response times, and assess the application's behavior under heavy load.

One of the key advantages of Locust is its distributed nature. It supports distributed load generation, enabling me to scale the test execution across multiple machines and simulate thousands of concurrent users. This distributed approach provides a more realistic representation of the application's performance under high-stress conditions and helps uncover performance issues that may not be apparent in single-user tests. I was not able to simulate stably more than 1 000 users from one machine.

Locust also offers real-time monitoring and reporting capabilities. During the test execution, I can monitor key performance metrics such as response times, error rates, and throughput through Locust's web interface. This allows me to analyze the system's behavior and identify performance hotspots to improve the overall performance of my Spring Boot REST application.

8.1.1 Why Locust?

I picked Locust for performance testing due to its simplicity, scalability, and real-time monitoring capabilities. It has an intuitive syntax it allows users to quickly define and execute load tests, simulating realistic user behavior. [11] Its distributed load generation capability enables scalability, allowing tests to be distributed across multiple machines to simulate high user loads.

8.1.2 Setup

Creating a Locust test is not complicated as you can see in the provided example code figure 8.1. The test methods annotated with @Test annotation call BE endpoints. In addition to that, you can specify the weight that ensures that the method is called more often. The challenges I had to solve were OAuth2 integration to get access to the secured endpoints, and generating unique users. OAuth2 integration is simple. Self.client is a wrapper around a python-requests, which means you only need to get the access_token from IAM at the start of each test and place it in the header for all other requests. To solve unique users problem I created a basic list with different credentials. On every test's start, I pop one pair of username and password from the list so none else can get these concrete credentials again.

```python
from locust import HttpUser, task, between
import logging

USER_CREDENTIALS =[
    ("pacient001", "Heslo001"),
    ("pacient002", "Heslo002"),
    ("pacient003", "Heslo003"),
    ("pacient004", "Heslo004"),
    ("pacient005", "Heslo005"),
]

class Test(HttpUser):
    wait_time =between(1, 3)
    host ="https://corposa.cz"

    auth_url ="https://auth.corposa.cz/realms/disease-guide/protocol/openid-connect/token"
    client_id ="client_id"
    client_secret ="client_secret"

    username ="pacient001"
    password ="Heslo001"

    @task
    def task_get_todo(self):
        self.client.get("/api/todo/")

    @task(4)
    def task_toggle_task(self):
        self.client.get("/api/todo/toggle/task/2")

    def on_start(self):
        if len(USER_CREDENTIALS) >0:
            self.username, self.password =USER_CREDENTIALS.pop()

        self.login()

    def login(self):
        response =self.client.post(self.auth_url, {
            'client_id': self.client_id,
            'client_secret': self.client_secret,
            'scope': 'openid',
            'grant_type': 'password',
            'username': self.username,
            'password': self.password
        }).json()
        self.client.headers.update({'Authorization': 'Bearer ' +response['access_token']})
```

Figure 8.1: Locust test example

8.1.3 Distributed

Locust's strength is its ability to run in a distributed mode. Once a master instance is created we can connect workers to it. The advantage is that workers can have different test scenarios. They just need to have the same class name. When the master instance is deployed on a server with a public domain, you can then connect workers from different machines. This way you can achieve a really large number of generated users that simultaneously perform actions on your tested application. [8]

8.1.4 Test cases

In this subsection, I introduce all test cases used during the performance testing.

Case ID	Name	Description
TC 1	Dashboard test	The user logs in and checks all dashboard information. That includes the longest strike, a list of all todos, events, and basic information about the user. The user marks one task as completed.
TC 2	Monitoring test	The user logs in and checks all his monitoring. That means he/she displays all his/her monitoring and then all measurements. That can vary, but usually, it's blood pressure or images.
TC 3	Education test	The user logs in and checks all educational courses that are assigned to him. Then the patient checks in detail one of the educational courses.
TC 4	Anamnesis test	The user logs in and looks at his or his family's anamnesis. The user opens one in detail and edits it.

Table 8.1: Performance test scenarios

8.1.4.1 Run

Since the performance test's main goal is to test the number of users it can withstand, it doesn't make sense to test each test case separately. On the contrary, it's desired to create the performance test as extensive as possible. Thus, I decided to run all the test cases together as one complex test case. Thanks to Locust's natural ability to perform distributed tests as mentioned in the section 8.1.2 I was able to add different workers with different test scenarios to one performance test.

During the testing, I was using three laptops each providing four or eight workers with different test scenarios. This way I was not limited by the local machine as it is often a case of a problem when dealing with generating a large number of users. For monitoring purposes, I was, of course, using mainly the Locust web UI as it provides all the important data such as response time, number of requests per second, and failures. To add more insight I monitor the VPS state through New Relic and container usage through the Docker stats command.

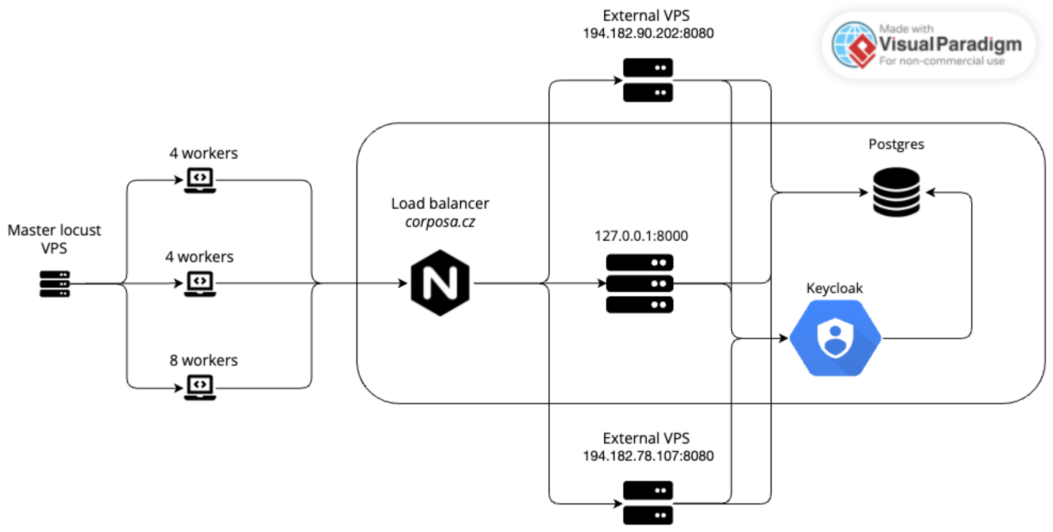


Figure 8.2: Distributed performance testing diagram

8.1.5 Results

In this section, we look at the different strategies and how they perform.

8.1.5.1 Scaling strategy 1: results

In this section, I test how many users can withstand scaling approach 1. Detail description of the approach can be found in section 7.1.1 of chapter 7. I was able to drive the number of users to **1 200** before it started to fail. For the 1 200 users, the average response time was under **2 s** and we manage to send over **300 requests per second**. See figure 8.3. VPS was running out of RAM space as shown in figure 8.4 and the container under the heaviest load was disease_guide as shown in figure 8.5.

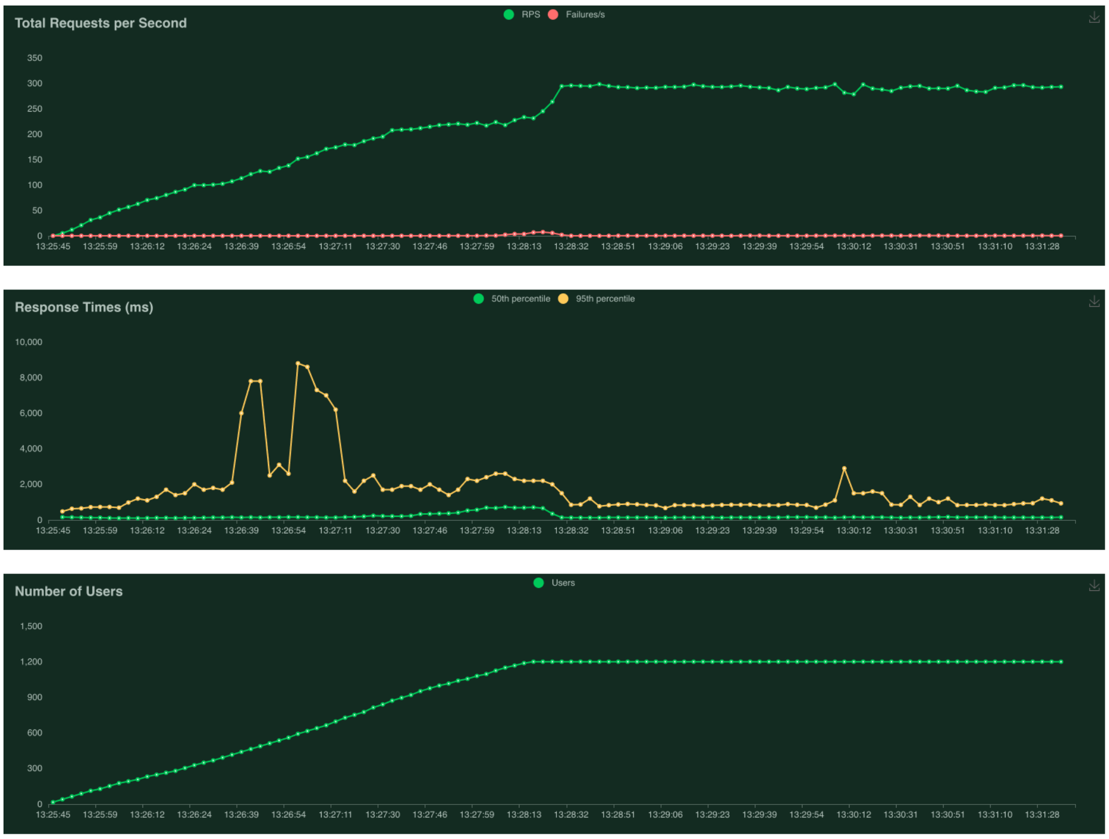


Figure 8.3: Locust statistics for strategy 1

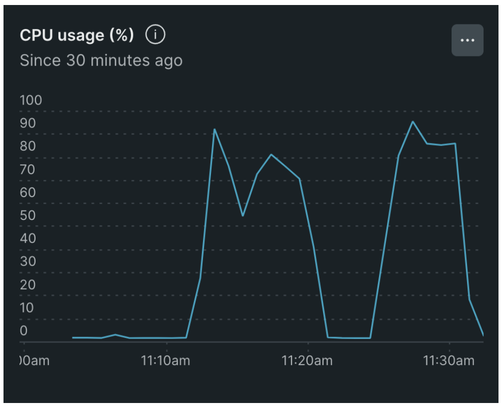
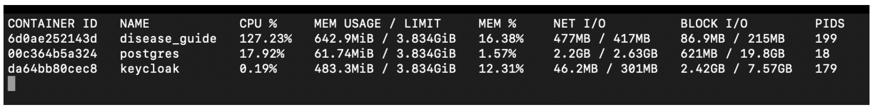


Figure 8.4: New Relic insight for strategy 1



```
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O         PIDS
6d0ae252143d   disease_guide   127.23%   642.9MiB / 3.834GiB   16.38%    477MB / 417MB     86.9MB / 215MB    199
00c364b5a324   postgres        17.92%    61.74MiB / 3.834GiB   1.57%     2.2GB / 2.63GB    621MB / 19.8GB    18
da64bb80cec8   keycloak        0.19%     483.3MiB / 3.834GiB   12.31%    46.2MB / 301MB    2.42GB / 7.57GB   179
```

Figure 8.5: Containers states for strategy 1

8.1.5.2 Scaling strategy 2: results

In this section, I test how many users can withstand scaling approach 2. Detail description of the strategy can be found in section 7.1.2 of chapter 7. Before I manage to get to the same number of users as in scaling approach 1 errors started occurring 8.7. The maximum number of requests per second was a little under 250. See figure 8.6. At that point, VPS run out of CPU power as shown in figure 8.8.

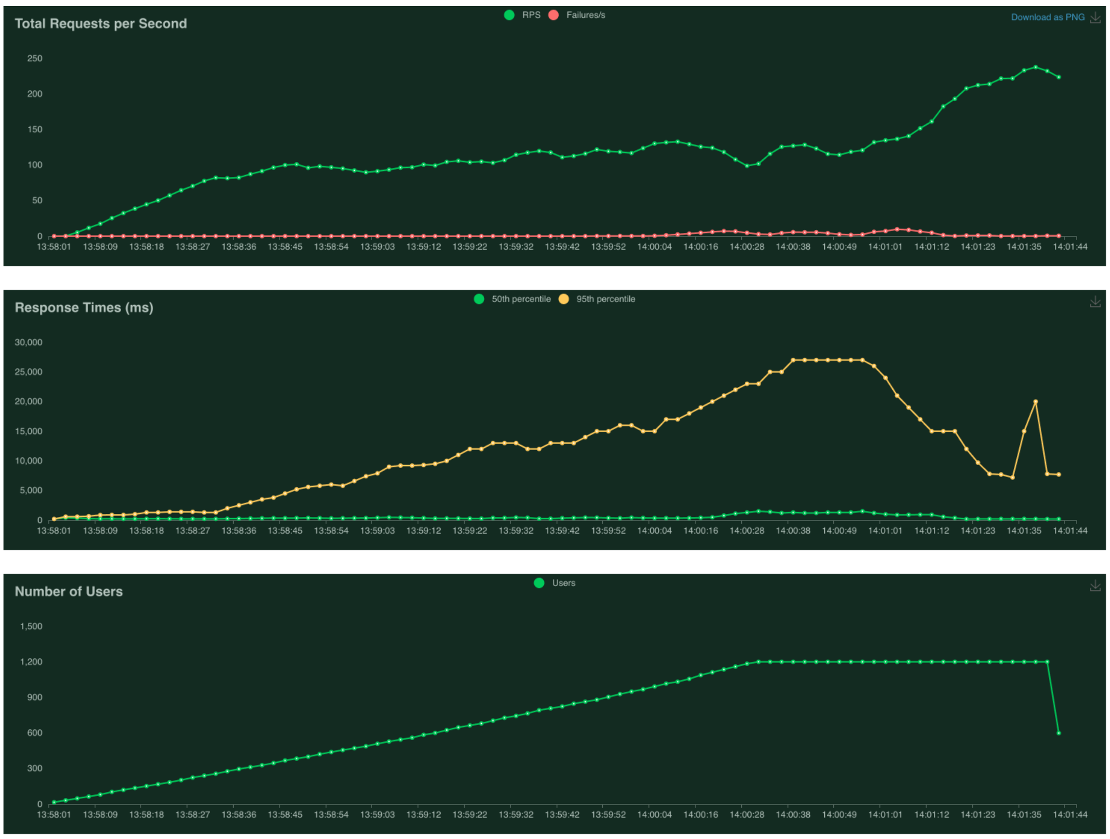
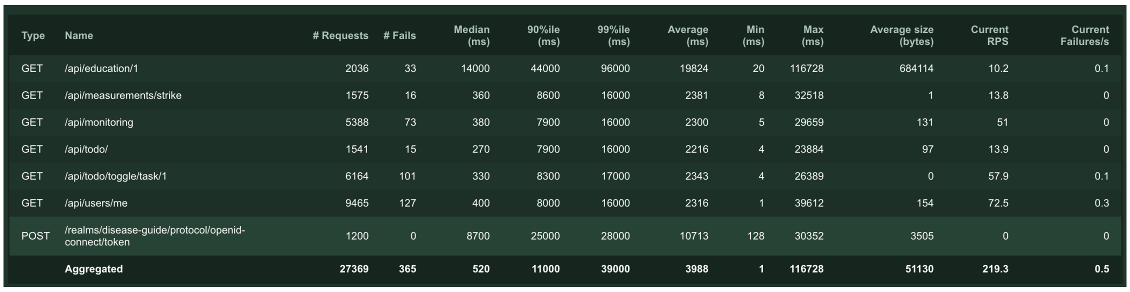


Figure 8.6: Locust statistics for strategy 2



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/api/education/1	2036	33	14000	44000	96000	19824	20	116728	684114	10.2	0.1
GET	/api/measurements/strike	1575	16	360	8600	16000	2381	8	32518	1	13.8	0
GET	/api/monitoring	5388	73	380	7900	16000	2300	5	29659	131	51	0
GET	/api/todo/	1541	15	270	7900	16000	2216	4	23884	97	13.9	0
GET	/api/todo/toggle/task/1	6164	101	330	8300	17000	2343	4	26389	0	57.9	0.1
GET	/api/users/me	9465	127	400	8000	16000	2316	1	39612	154	72.5	0.3
POST	/realms/disease-guide/protocol/openid-connect/token	1200	0	8700	25000	28000	10713	128	30352	3505	0	0
	Aggregated	**27369**	**365**	**520**	**11000**	**39000**	**3988**	**1**	**116728**	**51130**	**219.3**	**0.5**

Figure 8.7: Locust statistics aggregated for strategy 2

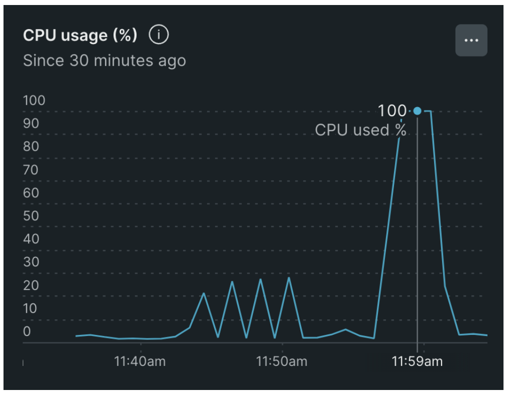


Figure 8.8: New Relic insight for strategy 2

8.1.5.3 Scaling strategy 3: results

In this section, I test how many users can withstand scaling approach 3. Detail description of the strategy can be found in section 7.1.3 of chapter 7. With this strategy, I was able to simulate 1 500 users with ease. The number of requests per second was a little under 500 and response time was between 1 and 2 seconds.

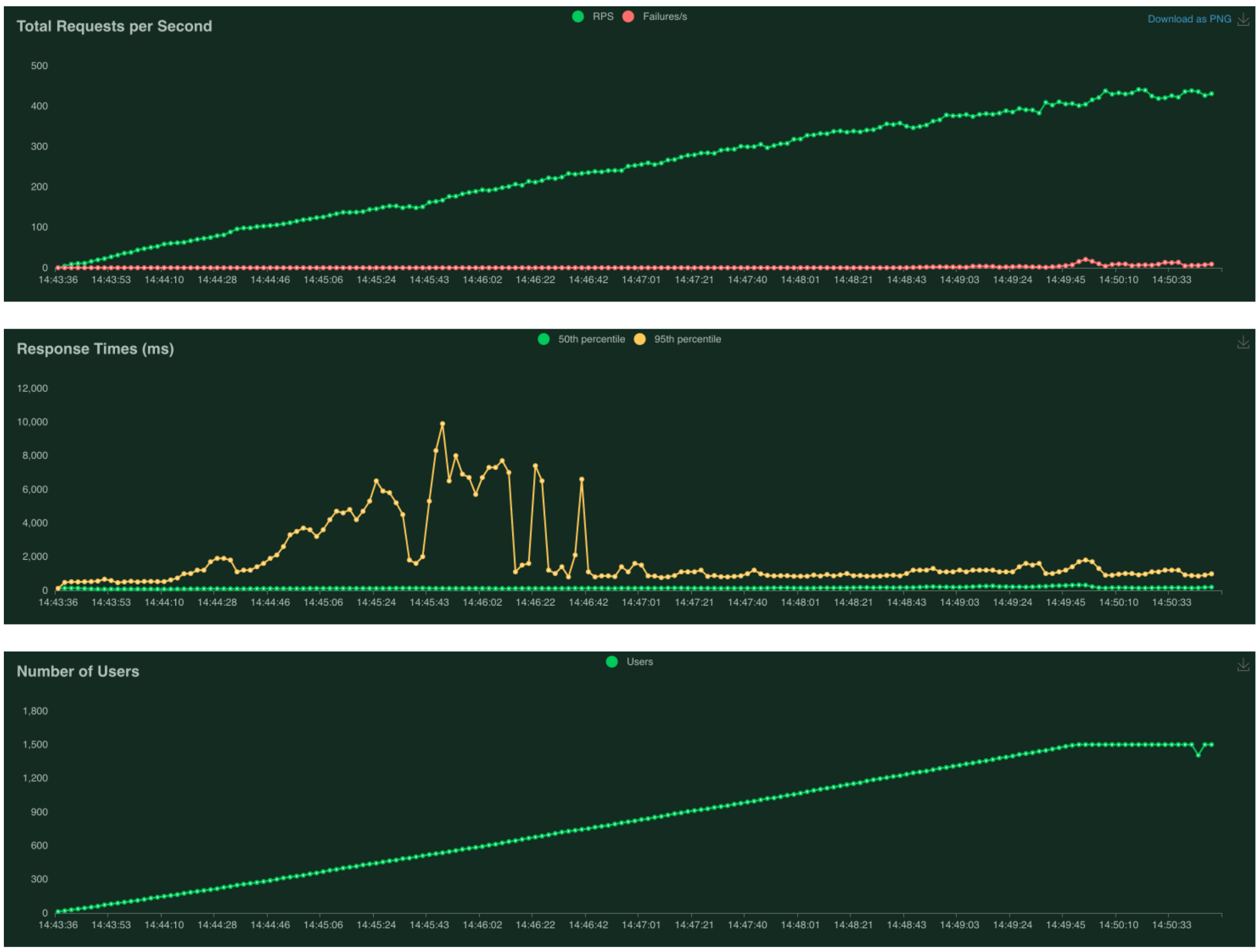
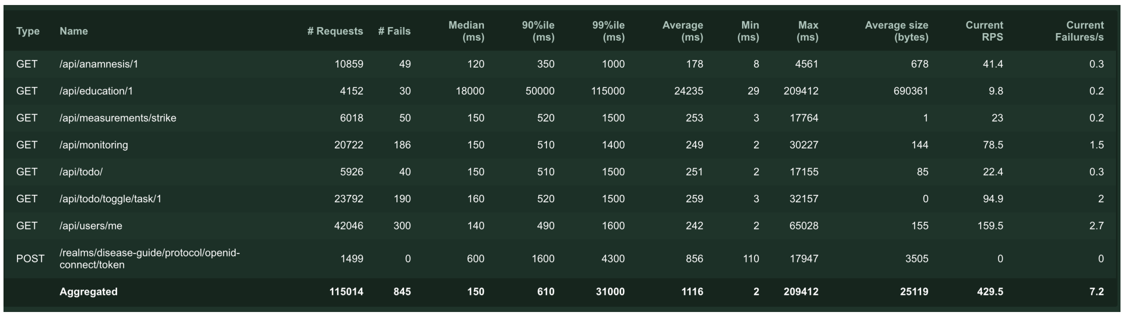


Figure 8.9: Locust statistics for strategy 3



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/api/anamnesis/1	10859	49	120	350	1000	178	8	4561	678	41.4	0.3
GET	/api/education/1	4152	30	18000	50000	115000	24235	29	209412	690361	9.8	0.2
GET	/api/measurements/strike	6018	50	150	520	1500	253	3	17764	1	23	0.2
GET	/api/monitoring	20722	186	150	510	1400	249	2	30227	144	78.5	1.5
GET	/api/todo/	5926	40	150	510	1500	251	2	17155	85	22.4	0.3
GET	/api/todo/toggle/task/1	23792	190	160	520	1500	259	3	32157	0	94.9	2
GET	/api/users/me	42046	300	140	490	1600	242	2	65028	155	159.5	2.7
POST	/realms/disease-guide/protocol/openid-connect/token	1499	0	600	1600	4300	856	110	17947	3505	0	0
	Aggregated	**115014**	**845**	**150**	**610**	**31000**	**1116**	**2**	**209412**	**25119**	**429.5**	**7.2**

Figure 8.10: Locust statistics aggregated for strategy 3

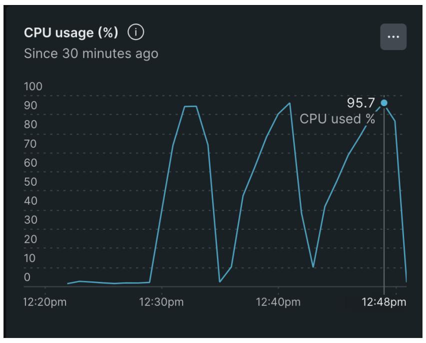
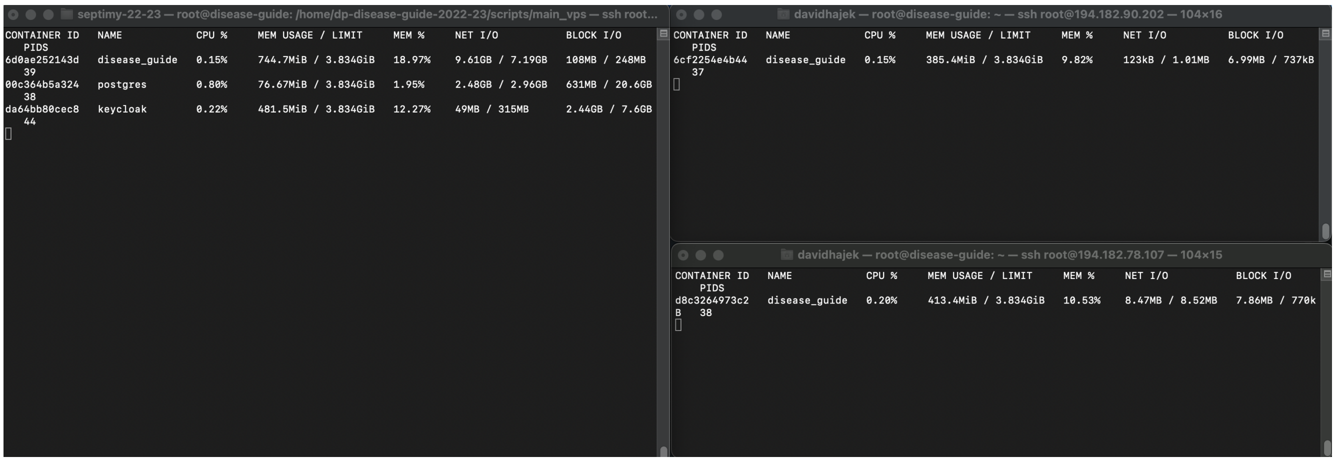


Figure 8.11: New Relic insight for strategy 3

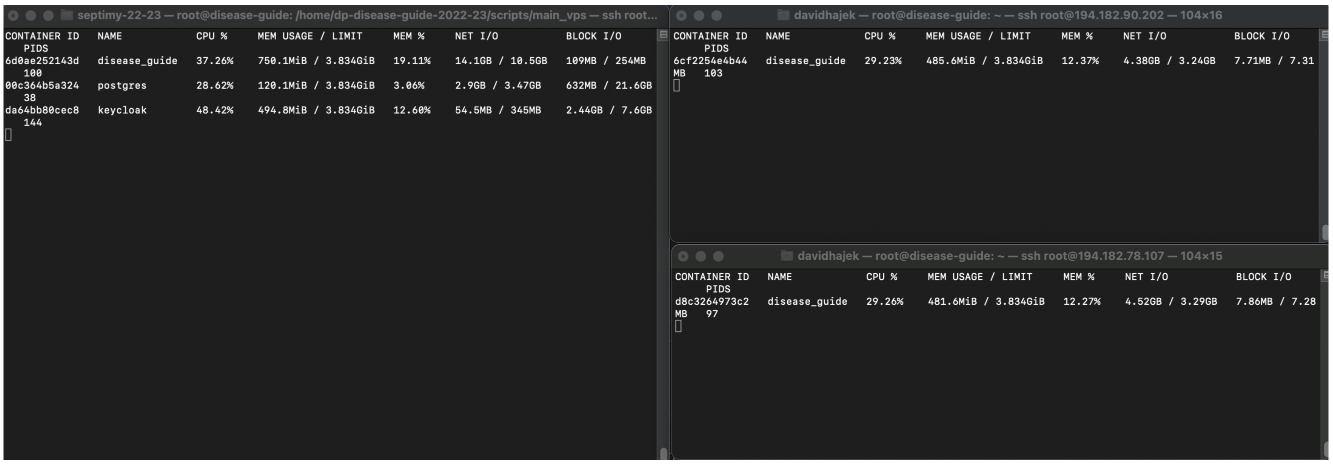


```
septimy-22-23 — root@disease-guide: /home/dp-disease-guide-2022-23/scripts/main_vps — ssh root...
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O
   PIDS
6d0ae252143d   disease_guide   0.15%     744.7MiB / 3.834GiB   18.97%    9.61GB / 7.19GB   108MB / 248MB
   39
00c364b5a324   postgres        0.80%     76.67MiB / 3.834GiB   1.95%     2.48GB / 2.96GB   631MB / 20.6GB
   38
da64bb80cec8   keycloak        0.22%     481.5MiB / 3.834GiB   12.27%    49MB / 315MB      2.44GB / 7.6GB
   44
```

```
davidhajek — root@disease-guide: ~ — ssh root@194.182.90.202 — 104×16
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O
   PIDS
6cf2254e4b44   disease_guide   0.15%     385.4MiB / 3.834GiB   9.82%     123kB / 1.01MB    6.99MB / 737kB
   37
```

```
davidhajek — root@disease-guide: ~ — ssh root@194.182.78.107 — 104×15
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O
    PIDS
d8c3264973c2   disease_guide   0.20%     413.4MiB / 3.834GiB   10.53%    8.47MB / 8.52MB   7.86MB / 770k
B   38
```

Figure 8.12: Containers states for strategy 3 - before



```
septimy-22-23 — root@disease-guide: /home/dp-disease-guide-2022-23/scripts/main_vps — ssh root...
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O
   PIDS
6d0ae252143d   disease_guide   37.26%    750.1MiB / 3.834GiB   19.11%    14.1GB / 10.5GB   109MB / 254MB
   100
00c364b5a324   postgres        28.62%    120.1MiB / 3.834GiB   3.06%     2.9GB / 3.47GB    632MB / 21.6GB
   38
da64bb80cec8   keycloak        48.42%    494.8MiB / 3.834GiB   12.60%    54.5MB / 345MB    2.44GB / 7.6GB
   144
```

```
davidhajek — root@disease-guide: ~ — ssh root@194.182.90.202 — 104×16
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O
     PIDS
6cf2254e4b44   disease_guide   29.23%    485.6MiB / 3.834GiB   12.37%    4.38GB / 3.24GB   7.71MB / 7.31
MB   103
```

```
davidhajek — root@disease-guide: ~ — ssh root@194.182.78.107 — 104×15
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O
     PIDS
d8c3264973c2   disease_guide   29.26%    481.6MiB / 3.834GiB   12.27%    4.52GB / 3.29GB   7.86MB / 7.28
MB   97
```

Figure 8.13: Containers states for strategy 3 - during

8.1.5.4 Conclusion

As we can see from the results of the tests, scaling strategy 2 did not over-perform the basic starting strategy 1 as it was able to withstand only around 1 000 users before it started to fail. Since strategy 1 still didn't satisfy our non-functional requirement, we had to implement scaling strategy 3. With this strategy, we were able to get as good results as 1 500 simultaneously connected users and a response time of around 1s. We have thus fulfilled all the key non-functional requirements. If we get to the point where our non-functional requirements run into the limits of our setup, for example, the active users growth above 1 500 users we would have to come up with a new scaling strategy.

8.2 UI testing - Playwright

In the world of software development, ensuring the quality and reliability of user interfaces is crucial. User Interface (UI) testing plays a vital role in verifying the functionality and responsiveness of applications across different browsers and platforms. To simplify and streamline UI testing in Java, developers can leverage the power of Playwright.

Playwright for Java is a Java library built on top of Playwright, the popular web automation framework. It brings the capabilities of Playwright's cross-browser automation to Java developers, empowering them to write robust and efficient UI tests. By leveraging Playwright's intuitive and powerful API, Java developers can easily automate interactions with web applications and validate their UI behavior. [5]

Playwright for Java provides a comprehensive set of features tailored for UI testing needs. It supports multiple browsers, including Chrome, Firefox, and WebKit, allowing developers to test their applications in different environments. With Playwright's consistent API, developers can write tests that work seamlessly across browsers without having to deal with browser-specific inconsistencies.

8.2.1 Challenges

Creating Playwright tests in Java is very straightforward and in combination with Playwright Inspector it's very simple to find all elements, that you want as it is often the slow thing on UI testing. However, it was not without challenges. The first problem I had to solve was signing in before each test. For that, I used @BeforeEach annotation for a method in which I fill out the login form and proceed to log in. The second problem was connected to the fact that I am testing a SPA application and it is not always simple for the testing framework to know whether the element you are looking for is still yet to be loaded or it is simply not there. Sometimes the page loads differently than we expect. A simple solution to this problem is to set the LoadState to NETWORKIDLE. [4] Both problems can be seen solved in figure 8.14.

```
@BeforeEach
void createContext() {
    String url ="https://auth.corposa.cz/realms/disease-guide/...";
    context =browser.newContext();

    // New page
    page =context.newPage();
    page.navigate(url);

    // Interact with login form
    page.getByLabel("Username or email").fill(username);
    page.getByLabel("Password").fill(password);
    page.getByRole(AriaRole.BUTTON, new Page.GetByRoleOptions().setName("Sign
                                                  in")).click();

    // Network idle
    page.waitForLoadState(LoadState.NETWORKIDLE);
}
```

Figure 8.14: Playwright beforeEach method

8.3 Rest testing postman

Postman is a powerful tool for testing RESTful services, offering developers a comprehensive and user-friendly platform to automate and streamline their API testing efforts. With its intuitive interface and extensive feature set, Postman simplifies the process of testing REST services and ensures the reliability and quality of API interactions. [24]

Using Postman for REST service testing provides several advantages. Firstly, Postman allows developers to create and manage collections of API requests, making it easy to organize and execute tests for different endpoints and scenarios. With the ability to define variables, headers, and parameters, developers can simulate various test cases and test the functionality of their REST services thoroughly.

Postman also offers a wide range of authentication methods, including Basic Auth, OAuth, and API key-based authentication, enabling developers to test different authorization mechanisms and ensure the security of their REST services. Since I use OAuth2 it's just a matter of filling up the access token URL and credentials. Especially, it comes in handy with the ability to set up authorization for the entire collection. So at the beginning of testing, I only have to receive a new token and set up the URL parameter on which I want to test the REST API, then I can run all tests at once and ensure that all endpoints are doing what I want.

For testing purposes, I created a testing database that can be deployed easily in a Docker. I use the exact same database for Playwright testing. After finishing the tests I delete the container so when I do testing again I start with fresh data.

8.4 User testing

The application was both tested and used by numerous users which I managed to convince. The output from users can be divided into two categories. One is ordinary application testing, which consists of exploring bugs and other flaws and the second category represents the benefits associated with using the application. Whether it helped them in their recovery process, motivated them in treatment plan adherence, etc. The total number of active users at the time of finishing my master's thesis is six users. All of them are being treated for different diseases and have access to the mobile and web application, although some prefer the web application and some the mobile one. In the end, I was able to get in touch with a few doctors to whom I showcased the application and presented its results. I perceive their thoughts as valuable feedback, which convinced me that this application makes sense and made me think about issues that can cause problems and try to solve them.

8.4.1 Different treatments

Each user is being treated for a unique disease, thus their experience might vary. I list for each patient a brief description of their disease and treatment and then I list their testing results as well as their overall feedback.

Achilles tendon injury: Patient 1 had suffered from an Achilles tendon injury from which he had been recovering for some time. He had been using Corposa application for an extended period of time and was thus motivated to adhere to his rehabilitation plan. The app reminded him to complete tasks, while at the same time allowing him to learn more about the injury itself and the impact on his quality of life.

Left ventricular hypertrophy: Patient 2 was diagnosed with eft ventricular hypertrophy, which means that the muscle of the heart's main pump (left ventricle) has become thick and enlarged. Subsequently, the patient was given a treatment plan in the form of arranged medical check-ups and regular home blood pressure and pulse measurements every day. The patient used the Corposa app to record these tasks, and by using it he was able to provide the doctors with the required records of the measurements.

Rehabilitation: On the basis of recurrent low back pain, Patient 3 was given a physiotherapy treatment plan consisting of regular therapeutic exercises with a doctor and a series of home exercises to be performed every morning and evening. In Corposa application, the patient could record the actual dates of the exercises with the doctor as well as the individual home exercises and the record of their completion. He could read about the benefits of the exercises on pain reduction in the educational module.

Epilepsy: Patient 4 was diagnosed with epilepsy at a young age and experienced several seizures. After a series of medical examinations, a plan was established to manage the condition in the form of daily medication, regular sleep of sufficient duration, and monitoring of selected vital functions. The application helped to remember to take the medication and to get the patient to bed on time.

Pneumothorax: Patient 5 suffered from a pneumothorax (a collapsed lung) some time ago, in the subsequent postoperative phase he had to take medication, perform regular breathing exercises and undergo oxygen inhalation procedures. The application was once again able to help him with the fulfillment of these tasks and their records, and in the educational module he was also informed about the recovery process and the positive effect of the set plan on his lungs.

Obesity: Patient 6 was diagnosed with obesity. In an effort to improve his health by reducing excess weight, he was given a treatment plan in the form of regular body weight measurements, a diet regimen for food consumption and monitoring the caloric values of individual meals. The application helped him measure these values and offered the possibility of varied food recipes in accordance with the diet as part of the education.

8.4.2 Users' review

All users rated the application positively. They saw its benefits especially in the areas of fulfilling set tasks, recording measured data and complying with overall treatment plans. Push notifications helped not to forget to complete the required tasks at the specified times, and thus ensured an uninterrupted treatment process.

At the same time, users gradually discovered some technical errors in the application. Primarily, there was repeated loading of the login page in the mobile application, then repeatedly displayed same to-do sheets at the same section. Another problem was the impossibility of uploading photos from the phone gallery, it was only possible to take a picture directly. These identified issues were gradually incorporated and fixed in the further development of the application.

8.4.3 Interviews with doctors

To show that the application makes sense I met four doctors to whom I showed the application and discussed with them how successful the application can be. All consultations were conducted in the form of an interview and took place at the end of my master's thesis work when the application was fully developed. I didn't consider it necessary to talk about it sooner as I had enough feedback on the idea itself from my bachelor's thesis, where I discussed it with Marx and the bariatric surgeon. [18] All interviews were held in the Czech language. For the purposes of this thesis, authorized interview transcripts were later translated into English. I began by explaining the concept followed by a showcase of the application. The concept was presented similarly as it is described in the proposal chapter 3 and in the Corposa chapter 5. Following that, several additional questions were asked.

8.4.3.1 MUDr. Zdeněk Poledník - orthopedic doctor

What do you think about the application? The app itself makes sense. We ourselves use an app similar to Moje endoprotéza application for one specific disease. It prepares you for artificial joint surgery, guides you through the surgical and post-surgical period. I feel positive about the education module. I agree that patients often don't remember everything they are told and there is non-adherence to treatment. If an app could help improve this, it would be useful. At the same time, I would not be opposed to the telemedicine. We tried prescribing in this way under covid-19 and it was even possible to report this treatment. Unfortunately, this was abolished after covid-19. Anyway, if the app allowed in some cases to check the patient's condition remotely and add some treatment plan based on that, then the patient wouldn't have to go to the hospital at all and the whole thing would go faster, or the number of checks could be reduced. A monitoring module makes sense, patients often use smart devices that allow this. I think this is the direction that treatment and patient contact could take.

In what area would the app make sense? The application could be used for various diseases. I can see the use in long term or chronic diseases for example. The Moje endoprotéza app is a good example for artificial joint surgery. The app is best suited for population chronic and long term, often lifelong diseases, i.e. which have the most people, where procedures can be adjusted, and at the same time programming them into the app is most worthwhile.

What are the challenges? The biggest challenge I see is the development of a database of treatment plans. This process can be challenging and the right diseases need to be identified. There are a large number of diseases and we can still treat each disease in different ways. The app is also not suitable for short illnesses where you are cured before you install the app. A significant number of patients are elderly people who do not have a smart device or do not know how to use it. For them, the app would not help. On the other hand, the proportion of patients who have a smart device is steadily improving.

What could be the next possible development? The anamnesis could be shared among the family. A patient who comes to the waiting room could have his or her health problem described in the app, along with the measured data, for example. The nurse would still scan the patient's QR code in the waiting room and everything would be displayed directly to the doctor. So when the patient comes in, we're talking straight away over specific data and a specific disease and the whole process goes faster.

How to create individual treatment plans? I agree that treatment plans would be created by a person in charge of the application who would consult it with doctors. Appropriate diseases and their treatments would have to be identified at the very beginning.

8.4.3.2 MUDr. Tomáš Kaštovský - general practitioner

What do you think about the application? It's an interesting idea. It is true that patients often do not leave the doctor's office fully informed. I myself was recently treated for a broken leg and was given no further instructions. The possibility of better information, including, for example, towards rehabilitation, would therefore be useful. I agree that patients often do not adhere to treatment, either consciously or unconsciously. An app that informs and motivates them to adhere to treatment could help. Data collection by the patient is a relatively common practice and patients are already used to it. At the same time, the use of various mobile devices is widespread even among the older population.

In what area would the app make sense? The application makes sense in a wide range of diseases. A good example is high blood pressure, where it is important to take the prescribed medication and record the necessary medical data, and for fractures it is a matter of periodic blood dilution and subsequent rehabilitation. Diabetes, cholesterol or other chronic diseases are also examples. In all cases, the disease guide, the education module and the monitoring of the patient's condition can be applied.

What are the challenges? The biggest problem is in creating treatment plans. Medical specialties and treatment procedures are enormous and consist of a vast amount of information that cannot be easily processed. It is therefore important to identify diseases where the implementation of a guide is worthwhile. Focus on common diseases where general approaches can be used and then apply these as easily as possible to individual treatment.

How to create individual treatment plans? It would be advisable for the plans to be developed in collaboration between the person in charge of the application and a specialised person, i.e. a doctor. The goal is primarily to save physicians time, and this is true even with the initial time investment when the treatment plan is created. The doctor's time should be used for consultation and approval of what the delegated person creates.

8.4.3.3 MUDr. Jan Jonáš - gynecologist

What do you think about the application? There is certainly a way in that direction. I think the idea is fundamentally good. There are many possibilities of using such an application. Especially in keeping an eye on medication, appointments and other basic tasks. Moreover, to a certain extent, not only the doctor but also other professional staff such as nurses may be authorized to approve or set medical plans.

In what area would the app make sense? Patients forget 60 % of the information they are being told during the appointment, this is a big problem. Keeping track of medication use, medication history, set and reminded dates for preventive examinations and preparation for scheduled surgeries - all of this can be covered by such an app. It is also advisable to keep the anamnesis up-to-date with information on when the patient took the medication, whether he or she had surgery or information on the anamnesis of close family members. A follow-up update can be made at the doctor's visit to increase the relevance of the information for the next period.

What are the challenges? This can be particularly problematic for complex diseases, where there are many pathways to appropriate treatment, which each doctor may determine differently depending on their own knowledge and the patient's condition. The treatment plans in the app would have to take this into account, each medical facility could have them tailored, but this then means significant initial time and financial costs.

What could be the next possible development? Linking with health facility ordering systems, creation of a card with basic data on the patient's health condition and the medication used. Other medical records, ultrasounds etc. could be uploaded as .pdf, .png files through the application. In addition, an electronic signature to give consent in accordance with legal requirements would be a good improvement, this can be done in collaboration with other technologies or platforms.

8.4.3.4 MUDr. Petr Oliva - orthopedic doctor

What do you think about the application? In general, the idea is very interesting. People are not aware of their medicines, of their diseases, they forget how long they have been sick, what symptoms they had. All this could well be recorded in such an application. We as doctors often get no information about patient's anamnesis, whether he or she uses any medication, which can complicate the determination of optimal treatment. At the moment, there is no central database providing such information. It would be very helpful if the patient had a basic summary of medical history and medications in the phone, which could be shared with the doctor, especially in acute cases.

What are the challenges? There are too many complex diseases for which such treatment plans cannot be easily prepared. Also, explaining how to use the app and the information available may take longer than today's usual explanation of the diagnosis itself.

In what other way could this application be used? One-day-surgery could be optimal, a post-surgical care, when information is provided on how to care for the treated area in the following period for a better recovery process. Also, connecting the internal system of the general practitioner with the application, where the doctor would record the detected condition and other data during the patient's visit and the selected data would be delivered to the patient's application.

8.4.3.5 Conclusion

As seen here above, I consulted with several doctors in this regard. The doctors differed in their specialties and thus each had their own perspective. Their patients and treatments differed as well. Still, I would make a few points on which they agreed. They all could imagine that the app could be used. They gave the best odds to diseases that are long-term, chronic, and for which it is not difficult to develop a treatment plan while having a large enough patient group. The operation of the app must not exceed the time that the app can save. This is not always easy to determine in advance. On the other hand, the biggest challenge in consensus is in creating treatment plans. The medical specialties are vast, there are always multiple ways to treat different diseases, and it will take time to roll this out. Ideally, treatment plans would be created by some person in charge of the application, who would then consult it with the doctors. Their invested time would be minimal.

Chapter 9

Conclusion

9.1 Summary

In my master's thesis, I focused on creating a system consisting of a mobile and web application with a goal of improving the efficiency of patient treatment, reducing the time spent in the doctor's office and promoting the patient in complying with the treatment plan and obtaining reliable information about their disease. In doing so, I have tried to contribute at least in part to solving some of the numerous challenges and problems that today's healthcare system in the Czech Republic faces.

9.2 Evaluation

I have managed to create an application that handles all the key functionalities as they were initially identified and more precisely specified. I consider their proposition, which partly came out of the results of my bachelor's thesis, and the subsequent implementation to be successful and I believe that this has fulfilled the goals of this master's thesis - to find a way to improve the quality and efficiency of the patient's treatment.

The success and potential usefulness of Corposa application is illustrated by the six patient-users and four doctors interviewed. Their evaluation repeatedly pointed to the possibility of improving motivation in the approach to compliance with treatment plans on the side of patients and simplifying and streamlining work on the side of doctors.

9.3 Continuity

For further development of the application, it is proposed to establish cooperation with at least one healthcare facility that will be able to start using the application on a full basis for its patients, i.e. offer them the opportunity to use the application and explain its benefits. It is necessary to agree in advance on the procedure of creating and modifying the generated treatment plans and their individual use by patients. During this preparatory phase, the application development side will improve the application, eliminate any identified shortcomings and provide functional service to the cooperating healthcare facility. After a set period of time, the facility itself can evaluate the actual benefits and impacts on the quality of the health services provided and, on this premise, provide further guidance and recommendations on the direction in which the application can be developed. To that, the application development side will be ready to listen.

According to the recommendations of the individual doctors I interviewed, it is appropriate to choose the path of selection of suitable diseases, which are usually chronic, long-term and for which it makes sense to create treatment plans with an unambiguous possibility of monitoring, instructing the treatment procedure and the necessary education, which will then be used individually by individual patients. The use of these plans must not be complicated and time-consuming for both doctors and patients. A more detailed work with personal and family history, which could be shared with family members, seems to be an interesting possibility for further development. With all aspects of recording and sharing such data, the development of the application needs to be in full compliance with the legal requirements for the protection of such data at all times.

Bibliography

[1] AGENCY, C. P. *Průzkum: Polovina pacientů přiznala nedodržování léčby* [online]. [cit. 18. 5. 2023]. Dostupné z: `https://www.zdravotnickydenik.cz/2019/12/pruzkum-polovina-pacientu-priznala-nedodrzovani-lecby/`.

[2] ANNADANAM, N. *Achieving PostgreSQL Master Slave Replication: 7 Easy Steps* [online]. [cit. 21. 1. 2023]. Dostupné z: `https://hevodata.com/learn/postgresql-master-slave-replication/#intro`.

[3] ANTONIO, G. *Beginning Java EE 7*. Berkeley, CA : Apress, 2013. ISBN 978-1430246268.

[4] ASHWIN. *Playwright Wait for Network Idle* [online]. [cit. 12. 5. 2023]. Dostupné z: `https://www.programsbuzz.com/article/playwright-wait-network-idle`.

[5] BACKER, A. *Playwright: The Future of Test Automation* [online]. [cit. 10. 5. 2023]. Dostupné z: `https://afsalbacker.medium.com/playwright-the-future-of-test-automation-3c82e28c5dc8`.

[6] BISWAS, S. *Networking in Flutter using Dio* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://blog.logrocket.com/networking-flutter-using-dio/`.

[7] CHAUHAN, H. *A Beginner's Guide to Create SPA with React Js* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://dev.to/hiteshtech/a-beginners-guide-to-create-spa-with-react-js-491c`.

[8] CLOUDPLEX-TEAM. *Kubernetes Distributed Performance Testing using Locust* [online]. [cit. 10. 5. 2023]. Dostupné z: `https://thechief.io/c/cloudplex/kubernetes-distributed-performance-testing-using-locust/`.

[9] DANIEL, F. *Complete Guide on Docker Logs [All access methods included* [online]. [cit. 20. 5. 2023]. Dostupné z: `https://signoz.io/blog/docker-logs/`.

[10] DANIEL, F. *Docker Stats | Understand how to monitor Docker Metrics with docker stats* [online]. [cit. 16. 5. 2023]. Dostupné z: `https://signoz.io/blog/docker-stats/`.

[11] ECHOUT, M. *What Is Locust Load Testing?* [online]. [cit. 10. 5. 2023]. Dostupné z: `https://www.blazemeter.com/blog/locust-load-testing`.

[12] FELICE, S. *Comprehensive Guide on Enterprise Application Testing* [online]. [cit. 15. 5. 2023]. Dostupné z: `https://www.browserstack.com/guide/enterprise-application-testing`.

[13] FOWLER, M. *Patterns of Enterprise Application Architecture.* Sebastopol, CA : O'Reilly Media, Inc., 2003. ISBN 9780321127426.

[14] GAMELA, A. – FIGUEIREDO, R. *Podman vs Docker: What are the differences?* [online]. [cit. 20. 1. 2023]. Dostupné z: `https://www.imaginarycloud.com/blog/podman-vs-docker/`.

[15] GILL, S. *PostgreSQL Backup Script: Made Easy 101* [online]. [cit. 12. 5. 2023]. Dostupné z: `https://hevodata.com/learn/postgresql-backup-script/`.

[16] HARRIS, C. *Microservices vs. monolithic architecture* [online]. [cit. 14. 5. 2023]. Dostupné z: `https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith`.

[17] HECKLER, M. *Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Application.* Sebastopol, CA : O'Reilly Media, Inc., 2021. ISBN 978-1492076988.

[18] HáJEK, D. *Web application for better patient care.* Prague : Czech Technical University, 2020.

[19] JANETTA NěMCOVá, P. K. *Nemocnicím chybí přes tisíc sester a stovky lékařů. Podívejte se, jak je na tom ta vaše* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://www.irozhlas.cz/zpravy-domov/nemocnice-kde-chybi-sestry-lekari_2002210600_pek`.

[20] JANEčKOVá, E. *GDPR - Praktická příručka implementace.* Wolters Kluwer, 2018. ISBN 978-80-7552-248-1.

[21] KESKIN, G. *Using Firebase Crashlytics with Flutter* [online]. [cit. 18. 5. 2023]. Dostupné z: `https://dev.to/gulsenkeskin/using-firebase-crashlytics-with-flutter-4a1f`.

[22] MARCOS. *Spring Boot with AWS S3 Bucket from zero to useful* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://medium.com/javarevisited/spring-boot-with-aws-s3-bucket-from-zero-to-useful-c0895ab26aaa`.

[23] MARCOS. *Using Interceptor in a Spring Boot API* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://medium.com/javarevisited/using-interceptor-in-a-spring-boot-api-9d7a0781dd19`.

[24] MATT. *Postman API testing by example* [online]. [cit. 12. 5. 2023]. Dostupné z: `https://testfully.io/blog/postman-api-testing/`.

[25] MAXIME, F. *4 Approaches to write a reactive Widget in Flutter without using StatefulWidget* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://medium.com/flutter-ease/4-approaches-to-write-a-reactive-widget-in-flutter-without-using-statefulwidget-96e9a947b97d`.

[26] MCKENZIE, C. *How to setup an Nginx load balancer example* [online]. [cit. 16. 5. 2023]. Dostupné z: `https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-setup-an-Nginx-load-balancer-example`.

[27] MDN-CONTRIBUTORS. *SPA (Single-page application)* [online]. [cit. 25. 5. 2023]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Glossary/SPA/`.

[28] MEYEN, N. *Keycloak — Open Source Identity and Access Management (IAM)* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://nmeyen.medium.com/keycloak-open-source-identity-and-access-management-iam-fe0f42a9879e`.

[29] MZČR. *Zdravotnictví České republiky ve srovnání se státy OECD* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://www.nzip.cz/clanek/477-zdravotnictvi-ceske-republiky-ve-srovnani-se-staty-oecd`.

[30] DEVELOPER. *OAuth 2.0 and OpenID Connect Overview* [online]. [cit. 20. 5. 2023]. Dostupné z: `https://developer.okta.com/docs/concepts/oauth-openid/`.

[31] ONE.IDENTITY. *What is Identity and Access Management (IAM)?* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://www.onelogin.com/learn/iam`.

[32] OSWAL, V. *Locust - Code construct based Load Testing* [online]. [cit. 25. 5. 2023]. Dostupné z: `https://virendraoswal.com/locust-code-construct-based-load-testing`.

[33] PAYNE, R. *Beginning App Development with Flutter: Create Cross-Platform Mobile Apps*. New York City, NY : Apress Media, LLC, 2019. ISBN 978-1484251805.

[34] PLEVáK, O. *České zdravotnictví trápí nedostatek pracovníků, krizi však zvládlo dobře* [online]. [cit. 23. 5. 2023]. Dostupné z: `https://euractiv.cz/section/ekonomika/news/ceske-zdravotnictvi-trapi-nedostatek-pracovniku-krizi-vsak-zvladlo-dobre/`.

[35] PUBLISHING, O. *Health at a Glance 2019: OECD Indicators* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://doi.org/10.1787/4dd50c09-en`.

[36] SIKHA BAGUI, R. E. *Database Design Using Entity-Relationship Diagrams*. Auerbach Publications, 2011. ISBN 978-1439861769.

[37] SIMPLILEARN-TEAM. *What Is Docker Hub? Explained With Examples [2023 Edition]* [online]. [cit. 16. 5. 2023]. Dostupné z: `https://www.simplilearn.com/tutorials/docker-tutorial/docker-hub`.

[38] SINGH, V. *Liquibase with SpringBoot* [online]. [cit. 20. 5. 2023]. Dostupné z: `https://medium.com/javarevisited/liquibase-with-springboot-d69e08e8bf56`.

[39] SRINIVASAN, K. *Advantages and Disadvantages – JSF* [online]. [cit. 21. 5. 2023]. Dostupné z: `https://javabeat.net/advantages-and-disadvantages-jsf/`.

[40] THOMAS, M. K. *What is New Relic?* [online]. [cit. 16. 5. 2023]. Dostupné z: `https://medium.com/tensult/what-is-new-relic-8106dfde903d`.

[41] TIOBE-SOFTWARE. *TIOBE Index for May 2023* [online]. [cit. 25. 5. 2023]. Dostupné z: `https://www.tiobe.com/tiobe-index/`.

[42] TRIPATHI, D. *How To Dockerize An Application* [online]. [cit. 20. 5. 2023]. Dostupné z: `https://blog.knoldus.com/how-to-dockerize-an-application/`.

[43] VAIBHAV, S. *Auth0 vs Keycloak* [online]. [cit. 20. 5. 2023]. Dostupné z: `https://stackshare.io/stackups/auth0-vs-keycloak`.

[44] VERMEER, B. *Spring dominates the Java ecosystem with 60% using it for their main applications* [online]. [cit. 16. 5. 2023]. Dostupné z: `https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/`.

[45] WASI, W. O. *Setup Nginx Reverse Proxy to access PostgreSQL database remotely* [online]. [cit. 11. 5. 2023]. Dostupné z: `https://wasi0013.com/2021/11/15/setup-nginx-reverse-proxy-to-access-postgresql-database-remotely/`.

[46] web:infogs. SeamlessMD. https://seamless.md/product/product-overview/|, stav k 21. 1. 2023.

[47] WIEGERS, K. *Software Requirements (Developer Best Practices)*. Microsoft Press, 2013. ISBN 978-0735679665.

[48] WIKISKRIPTA. *Telemedicína* [online]. [cit. 20. 5. 2023]. Dostupné z: `https://www.wikiskripta.eu/w/TelemedicÃŋna`.

Chapter 10

List of digital attachments

Core application - Source code of BE and React FE
Mobile application - Source code of mobile application