TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Richard Õnnis 201421IVSM

Readiness of Estonia’s digital appointment system to health crisis: Lessons from the COVID-19 pandemic

Master’s thesis

Supervisors:	Juri Belikov
	PhD
	Maksim Boiko
	MSc
	Maksim Zhukov
	BSc

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Richard Õnnis 201421IVSM

Eesti Üleriigilise Digiregistratuuri valmidus tervisekriisiks: COVID-19 pandeemiast saadud õppetunnid

Magistritöö

Juhendajad:	Juri Belikov
	PhD
	Maksim Boiko
	MSc
	Maksim Zhukov
	BSc

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis and this thesis has not been presented for examination or submitted for defence anywhere else. All used materials, references to the literature and work of others have been cited.

Author: Richard Õnnis

09.05.2022

Abstract

The COVID-19 pandemic exposed many problems in the architectual implementation of Estonia’s digital appointment system and created demand for a solution where patients and healthcare workers could create and manage appointments for vaccinations. The architectual shortcomings revolved around three main categories: the scalability and performance of the application, the maintainability of the software, and the generation and processing of documents used in health information exchange across institutions. A custom solution for creating vaccination appointments was required due to the unique nature of the vaccination process, where the tracking of patients’ current immunisation status was essential. In this research we show how the identified architectual problems were resolved by restructuring the digital appointment system’s monolithic design to a mircoservice-based architecture and how a solution for the creation and management of vaccination appointments was realised within the portal. The results showed that the architectual restructuring reduced the average available appointment time slots request processing time by 26.42% and the application maintained very good performance when running during higher demand. Approximately 725 000 new vaccination appointments were created by patients and healthcare workers in Estonia with the help of the vaccination appointment creation system realised throughout this thesis. We anticipate this research to be a starting point for further restructuring of monolithic designs in the digital appointment system and the healthcare domain across Estonia. Furthermore, the adaptive vaccination appointment module is expected to guide the realisation of similar functionality and significantly increase Estonia’s healthcare network’s readiness for any upcoming pandemics or health crises in general. The thesis is written in English and contains 88 pages of text, 9 chapters, 44 figures and 6 tables.

Annotatsioon

COVID-19 pandeemia paljastas mitmed probleemid Eesti Üleriigilise Digiregistratuuri arhitektuuris ning lõi nõudluse lahendusele, kus patsiendid ja tervishoiutöötajad saaksid luua ja hallata broneeringuid vaktsineerimistele. Puudused arhitektuuris seisnesid kolmes põhilises kategoorias: rakenduse skaleeritavus ja jõudlus, tarkvara hooldatavus ja tervishoiuasutuste vahelises suhtluses kasutatavate dokumentide loomine ja töötlus. Vaktsineerimisprotsessi iseärasuse tõttu, kus patsientide immuniseerimise oleku jälgimine on hädavajalik, tekkis vajadus uuele, kohandatud lahendusele, kus kasutajad saaksid luua vaktsineerimiste broneeringuid. Selles töös näitame, kuidas tuvastatud probleemid lahendati monoliitse arhitektuuri ümbertegemisega mikroteenusteks ning kuidas digiregistratuuri portaalis realiseeriti vaktsineerimiste broneeringute loomise ja haldamise lahendus. Tulemused näitasid, et endise lahenduse asendamine mikroteenustega lühendas vabade aegade päringu töötlemise aega 26.42% võrra ning rakendus saavutas väga häid tulemusi kõrgema nõudluse puhul. Töös realiseeritud lahenduse abiga lõid patsiendid ja tervishoiutöötajad ligikaudu 725 000 uut broneeringut vaktsineerimisele. Eeldame, et tööst saab edukas alguspunkt teiste monoliitsete lahenduste ümbertegemisele nii digiregistratuuris kui ka muudes tervishoiusüsteemides üle Eesti. Veelgi enam, patsiendi immuniseeritusele kohanduv vaktsineerimiste broneeringute loomise moodul juhib tulevikus sarnaste lahenduste arendust ning suurendab märkimisväärselt Eesti tervishoiu valmidust tulevasteks pandeemiateks ja tervisekriisideks. Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 88-l leheküljel, 9 peatükki, 44 joonist ja 6 tabelit.

List of abbreviations and terms

üDR	Estonia’s digital appointment system
Tehik	Health and Welfare Information Systems Centre
HL7	Health Level Seven
V3	Version 3
XML	Extensible Markup Language
TIS	Estonia’s central health information system
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
JAXB	Java Architecture for XML Binding
CRAB	Common Resources Access Bank
SPONGE	Service for Properties Organisation, Normalisation, Gathering and Exchange
SPRUT	Slots Processing and Requesting UniT
ORCA	Operator for Reserving and Cancelling Appointments
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
TTO	Healthcare service provider
NARVAL	Narrowing And Restricting Vaccination Access List
RSD	Relative standard deviation
FHIR	Fast Healthcare Interoperability Resources
REST	Representational state transfer

Table of contents

List of figures

List of tables

1. Introduction

On March 11, 2020, the World Health Organisation (WHO) declared the novel coronavirus (COVID-19) to be a global outbreak [1]. A day later, local transmission of the virus was found in Estonia [2], followed by the state of emergency announced in the country [3].

The aggressive COVID-19 [4] pandemic has posed many challenges to healthcare information systems worldwide. Being overwhelmed by extraordinary demand for patient admissions and treating a large number of COVID-19 cases, healthcare institutions and workers were required to adapt amidst scarcity of resources [5]. Additionally, the COVID-19 pandemic has exposed the weaknesses of existing public health systems. The specific nature of the COVID-19 pandemic has required a strong coordination of connected data, people, and systems [6]. To alleviate the pressure on healthcare institutions as well as tackle problems with existing information systems, it has been crucial to invest in health system infrastructures as even the smallest investments can yield massive gains before, during and after a crisis [7].

Estonia's digital appointment system Üleriigiline Digiregistratuur or üDR [8] is a web portal developed by the Health and Welfare Information Systems Centre TEHIK [9]. It is a backbone of the Estonian society that enables citizens to view and manage their referrals and appointments in all the interfaced healthcare institutions. Most healthcare institutions have their individual digital appointment systems, however üDR makes managing multiple appointments and/or referrals with different healthcare providers more convenient. Currently around 200 healthcare providers all over Estonia are connected to üDR. Through the portal patients are able to view, change and cancel their future appointments, search for available appointment times, book new appointments, manage their active referrals and browse their appointments history.

With the COVID-19 pandemic, üDR became the main channel for Estonian citizens to book appointments for their vaccinations, which brought a lot of pressure on the application, as well as opened many improvement opportunities, due to the solution not initially being designed to handle such large-scale traffic.

1.1 Motivation

As stable delivery of high-quality digital health services is of crucial importance, especially during a global health crisis, it has been essential that the Estonia's digital appointment system answers to the highest standards. Therefore the elimination of any possible bottlenecks and flaws within the application should be of the highest priority, to enable stable and quick performance and good maintainability.

Additionally, effective distribution of all the available vaccination appointment slots is vital during a pandemic. For that reason, to enable a smooth and efficient immunisation of the general population of Estonia, üDR must become an easy to use, reliable tool for creating and managing vaccination appointments

1.2 Goal and research questions

One of the primary goals of this thesis will be the improvement of üDR's general architecture by restructuring its legacy monolithic architecture to microservices. The maintenance and development of monolithic software systems is a difficult task and service-oriented architectures yield more maintenance and less complexity in developing large-scale software applications [10].

Another key benefit of microservices is scalability. Each microservice can be deployed on different machines, each one with different levels of performance, and can be written in the more appropriate programming language. If there is a bottleneck on a specific microservice, it can be containerised and executed across multiple hosts that run in parallel, without the need to deploy the system on a new and more powerful machine [11].

Furthermore, when handling large traffic and a big number of requests, microservices have been proven to perform more efficiently when compared to monolithic software solutions [12]. Altogether, the microservice architecture has many advantages for building high quality software that is easy to scale, more reliable and in long term more convenient to maintain.

In addition to the architectural restructuring, the author aims to implement a custom vaccination appointment creation solution, where patients will be able to see their COVID-19 immunisation status, their vaccination history and book and manage appointments for vaccinations. The following research questions were formulated to guide this research:

- **RQ1:** How can Estonia's digital appointment system be restructured to a microservice-based architecture?
- **RQ2:** What benefits will the restructuring of Estonia's digital appointment system to a microservice-based architecture yield?
- **RQ3:** How should the creation and management of vaccination appointments be implemented in Estonia's digital appointment system?

1.3 Contributions

Throughout this thesis the author has thoroughly researched the current architecture of the digital appointment system, central health information system and the data transfer of health-related documents in Estonia. The current implementations were re-evaluated, and numerous shortcomings were identified. New microservice-based architecture consisting of four microservices was designed to replace the underperforming elements in the present implementation, containing novel solutions for healthcare document processing, custom circuit breakers and healthcare providers' properties management. In general, the scalability, maintainability and performance of the application was improved significantly.

With the help of the World Health Organisation, Estonia's Ministry of Social Affairs and the Health and Welfare Information Systems Centre, the author has contributed to the design of the vaccination rules and guidelines within Estonia. A new, adaptive solution for creating and managing vaccination appointments within Estonia's digital appointment system was implemented. The information regarding the patients immunisation was gathered and stored by the Health and Welfare Information Systems Centre, while the algorithms responsible for choosing appropriate vaccine types for the next vaccination and the design of the user interface were realised by the author.

The architectural changes yielded an average of 26.42% improvement in the time during which the requests are processed, the maintaniablity of the solution was considerably improved and the scalability of the implementation ensured that the solution will function properly in times of higher demand. At the time of writing this thesis, 398 449 unique patients have created 724 756 vaccination appointments with the help of the custom solution implemented throughout this thesis. Multiple further improvement opportunities were explored at the end of the research.

1.4 Outline of the thesis

At the start of this thesis, the overall structure of Estonia’s current digital appointment system, central health information system and health document transfer is explored. In the next chapter the shortcomings of the current implementation are highlighted and analysed. Chapter 4 introduces and thoroughly reviews the new microservice architecture and the structure of all the individual microservices. After that the custom vaccination appointment creation and management solution is presented and the vaccination algorithms are discussed. The next chapter analyses the performance and scalability of the presented architectural restructuring, as well as overviews the statistics related to the created vaccination appointment solution. Chapter 7 covers the dicussion and future improvements, where any further possibilites for enchancing the solution are explored. Finally the work done throughout the thesis and the results achieved are concluded.

2. General structure and work of üDR

In this chapter we discuss the technical and architectural structure of üDR, describe what actions users can perform through the portal and review how üDR communicates with the interfaced healthcare providers.

2.1 General description

üDR allows healthcare providers to offer their appointment times for different services through a single portal. By doing that it takes stress off of individual healthcare information systems, as well as improves the visibility of different available appointments by making them accessible through a unified web application.

üDR is a Spring Boot [13] application that is written on Java 1.8 [14]. It uses a monolithic architecture, meaning that it is built as a single unit, is self-contained and is independent from other computing applications. The solution uses external configuration files for setup and flexible change of different parameters, as well as a PostgreSQL [15] database that contains all the data relevant to the work of the program. üDR uses Maven [16] as its build automation tool.

The main goal of the solution is to save patients' time by providing a clear, quick overview of all the possible appointment times all over Estonia [17]. Instead of searching for the most suitable appointment through all the individual healthcare provider information systems, the patients can perform all the appointment searching and management activities through a single national portal.

In 2021, üDR had a total of 2 415 193 unique visits, or an average of 201 266 visits every month and 6617 visits every day. The users spent 361 seconds or approximately 6 minutes and made 4.7 actions in the portal during a single visit on average. Figure 1 shows how throughout the year of 2021, üDR had the highest activity during the earlier hours of the day, from 9.00 to 13.00 local time. The lowest activity was expectedly registered at night. Additionally the chart illustrates how the amount of visits to the üDR portal increased five times from year 2020 to 2021.

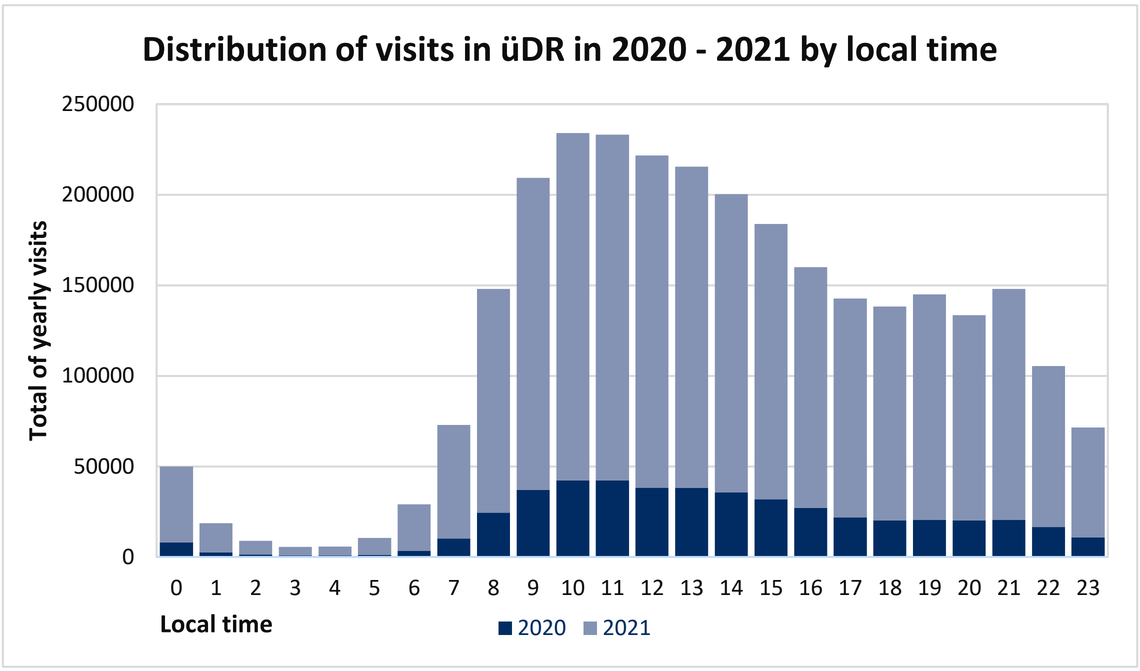


Figure 1. Distribution of visits in üDR in 2020 and 2021 by local time

2.2 Activities patients can perform through üDR

There are three main views in üDR through which patients can perform activities: the bookings view, the open referrals view and the find a time view.

In the bookings view patients can view and manage their currently active, upcoming appointments. The page displays a list of patients' active appointments describing the most relevant information: service, provider, doctor, appointment time, patient and important notes from the provider. From there the users can open a more detailed view of the appointment, where they can see more information and can cancel or change the appointment.

In the open referrals view patients can see all their available referrals, which can be used to book appointments for services specified on the referral. Similarly to the appointments view, the referrals view displays a table of referrals with their service, provider, doctor, the validity and duration of the referral, patient and a reason why the patient was referred to the service. From this view the patient can choose to search for an available appointment fitting the description provided in the referral and utilising that referral if a suitable time is booked as a result.

Both the referrals view and the bookings view (when changing an appointment) can lead to the find a time view i.e. the search appointment view. The view can also be accessed separately when creating a new appointment without a referral. In this view, the patient can specify the service, the counties and the healthcare providers in which they want to search for available appointments. Alternatively, when redirected from the referrals view or the bookings view, these fields are filled in automatically. Additionally, the patient has to specify the dates between which they wish to search for appointment times. After finding a suitable appointment, the patient can book it, and if the booking is successful, the patient's new appointment will be displayed under the bookings view.

In addition to the three main views, there are also the referrals and bookings view, the history view and the help view. The referrals and bookings view simply shows the patients referrals and bookings on a single page and is considered as the primary view where the patients are initially redirected. It does not have any additional functionality and contains functions of both the bookings and open referrals views. The history view allows the patients to see the history of their past appointments and the help view contains news related to the application, a user manual and information about the interfaced healthcare providers.

2.3 Communication with healthcare providers

There are four main events when üDR needs to communicate with healthcare providers: search for available appointment time slots, creation of an appointment, changing of an appointment and cancellation of an appointment

2.3.1 Search time slots

When a user creates a request for available time slots in the find a time view, a request is sent to all the healthcare providers that the user has specified in their search. Subsequently, after receiving all the responses, üDR displays all the found time slots in the selected healthcare providers. After that the user can choose a suitable time slot and proceed to create an appointment.

2.3.2 Creation, changing and cancellation of appointments

When creating an appointment, the request is sent to a single healthcare provider, where the appointment takes place. After receiving the request, the healthcare provider's information system responds whether the creation was successful or failed, and the user of üDR is notified accordingly.

Similarly to the creation of an appointment, üDR needs to communicate with the healthcare provider when one of its appointments is cancelled or changed. When the user decides to cancel an appointment, a single request is sent to which the provider can also respond with a success or a failure.

However, when changing an appointment to an appointment in a different provider, üDR needs to communicate with two different healthcare information systems. The new provider, where the user wants to change their appointment to, is sent an appointment creation request. After a successful response, the appointment in the previous healthcare provider is cancelled. Next, the format and channels through which üDR communicates with the healthcare providers is discussed.

2.3.3 X-Road, HL7 and TIS

The transfer of data between üDR and healthcare providers is done through the X-Road. X-Road is an open-source software and ecosystem solution that provides unified and secure data exchange between organisations. The basic idea of X-Road is that members of an ecosystem exchange data through access points called Security Servers [18] that implement the same technical specifications.

The information sent through the X-Road in üDR follows the HL7 (Health Level Seven) standards [19]. The HL7 framework contains a set of standards for the exchange, integration, sharing, and retrieval of electronic health information. These standards define how information is packaged and communicated from one party to another, setting the language, structure and data types required for seamless integration between systems. Currently the Estonian healthcare information network uses the HL7 V3 (Version 3) suite of specifications [19].

Before sending any of the aforementioned requests to a healthcare provider, üDR generates a standardised HL7 V3 XML (Extensible Markup Language) [20] request document that contains all the relevant information the healthcare provider needs to process the query and send a response. Below is an example of a standard for the HL7 V3 appointment cancellation request XML document (Figure 2).

```
<?xml version="1.0" encoding="UTF-8"?>
<PRSC_IN010601UV01 ITSVersion="XML_1.0"
xsi:schemaLocation="urn:hl7-org:v3 http://pub.e-
tervis.ee/standards2/Schema/V3/HL7-ORG-V3-2007-
01/processable/multicacheschemas/PRSC_IN010601UV01.xsd"
xmlns="urn:hl7-org:v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <templateId root="1.3.6.1.4.1.28284.6.1.1"
                extension="1.3.6.1.4.1.28284.6.1.1.91.2"/>
    <id root="1.3.6.1.4.1.28284.6.2.4.14" extension="DR.04066"/>
    <creationTime value="20170403224513"/>
    <versionCode code="V3-2007-01"/>
    <interactionId root="2.16.840.1.113883"
                   extension="PRSC_IN010601UV01"/>
    <processingCode code="P"/>
    <processingModeCode code="T"/>
    <acceptAckCode code="AL"/>
    <receiver typeCode="RCV">
     <device classCode="DEV" determinerCode="INSTANCE">
       <id root="1.3.6.1.4.1.28284.6.2.2.38.1"
           extension="X-tee-andmekogu-ID"/>
     </device>
    </receiver>
    <sender typeCode="SND">
     <device classCode="DEV" determinerCode="INSTANCE">
       <id root="1.3.6.1.4.1.28284.6.2.2.18.6" extension="DR"/>
     </device>
    </sender>
    <controlActProcess moodCode="RQO">
     <authorOrPerformer typeCode="AUT">
       <assignedPerson>
         <id root="1.3.6.1.4.1.28284.6.2.2.1"
             extension="45002280288"/>
         <assignedPerson classCode="PSN"
                     determinerCode="INSTANCE">
           <name>
             <family>Cuusk</family>
             <given>Katy</given>
           </name>
         </assignedPerson>
       </assignedPerson>
     </authorOrPerformer>
     <subject xsi:nil="false">
       <actAppointment>
         <id root="1.3.6.1.4.1.28284.1.3.2.20"
             extension="PERH.00323"/>
       </actAppointment>
     </subject>
     <reasonOf typeCode="RSON">
```

```
        <detectedIssueEvent classCode="ALRT" moodCode="EVN">
          <id root="1.3.6.1.4.1.28284.1.3.2.20"
              extension="PERH.49302"/>
          <code code="CHG"/>
        </detectedIssueEvent>
      </reasonOf>
    </controlActProcess>
</PRSC_IN010601UV01>
```

Figure 2. Appointment cancellation request HL7 V3 XML standard used in üDR

All the standards used in the health information exchange connected to the functionality of üDR are available through TEHIK's publication portal [21].

Before being sent to the healthcare provider, the request travels through Estonia's central health information system Tervise Infosüsteem or TIS, which stores and processes all the data collected in the course of the provision of Estonia's health services not only connected to the work of üDR [22]. E-services created on the basis of the TIS health information system allow the delivery of an improved health service and provide a good overview of treatment processes and health data to both healthcare professionals and patients across Estonia. The Ministry of Social Affairs is responsible for its operation and TEHIK for its maintenance and development. TIS is implemented as a WebMethods [23] integration server.

TIS performs all the necessary validation for the requests, stores the relevant information in the central database and sends the HL7 V3 request to the healthcare provider. After the healthcare provider responds, TIS validates the response, the information concerning the result of the query is stored and finally, the response is sent back to üDR. While all healthcare providers have their separate databases that store the information concerning its patients' appointments, the information about all the appointments of all the interfaced healthcare providers is also stored and processed through TIS's centralised database.

3. Shortcomings of the current implementation

In the beginning of March 2021, higher batches of COVID-19 vaccines started to arrive to Estonia [24]. To allocate the 6000 available vaccination times Estonia integrated üDR to support the creation of appointments for vaccinations. During this period, the target group for vaccinations were people above the age of 65, which set the pool of potential users to 300 000. On the launch of the service in üDR, the system was overwhelmed. This helped to identify and highlight multiple shortcomings of the current implementation related to three primary categories:

- Performance and scalability of the application
- Maintainability of the solution
- XML document generation process

The identified problems are dicussed in more detail in the following chapters.

3.1 Performance and scalability

One of the main problems with üDR was its architectural solution's impact on the performance and scalability of the application.

3.1.1 Problems with the monolithic architecture

As all the data communicated with the healthcare providers travels and is processed through TIS, üDR is completely dependent on its stable performance. Similarly, when üDR sends an unusually large amount of traffic through TIS, it can impact the performance of the whole system. If there is global downtime in TIS, üDR will stop functioning as the communication with healthcare providers will not be possible.

Furthermore, it is impossible to allocate more resources to a dedicated service in the TIS webMethods integration server. When higher load is expected, the whole system needs to be upscaled. In addition, the upscaling and downscaling of the whole integration server is difficult, it can be extremely costly as the distribution of resources on different packages and services is poorly optimised, which results in higher resource requirements.

3.1.2 X-Road message duplicity

As all the HL7 XML requests and responses to and from the healthcare providers need to be processed through TIS, the same messages are sent twice. The request that üDR sends to TIS is exactly the same request that is later sent to the hospital. This means that TIS does not play a role in modifying or decorating the request, but simply performs validation and saves information and statistics to the database.

The same applies to responses from healthcare providers. The XML messages can be very large and contain a lot of irrelevant text to the data itself, for example different wrappers, headers, structure for the message etc. The duplicity of every message sent to and from the healthcare providers can significantly slow down the speed at which the requests are processed.

3.2 Maintenance and updates

With all the requests and responses being processed through the central webMethods integration server, the development, updates and debugging of the new and existing features concerning the appointment processing has to be done in TIS. Containing hundreds of packages, each responsible for tens, and in some cases even hundreds of services written in visual programming, the management activities for the system can become inefficient and overwhelming.

3.2.1 WebMethods integration server visual programming in TIS

Visual code can be a powerful tool, serving as a common language between developers, administrators and project managers. However, the toolkit provided by the webMethods integration server is somewhat unintuitive and can be very difficult to understand when defining and implementing algorithms or complex programming constructs that are present in TIS.

In some cases the references for some of the services can exceed the depth of ten layers and keeping track of which services are causing the problem or where the new implementations are expected can be cumbersome. In addition to that, the TIS implementation has a lot of switch and if statements that process the requests differently based on many parameters. With so many branches it is easy to leave an important code sequence unnoticed, which renders the solution prone to errors.

In addition to that, in the current implementation of TIS, a generic object called requestRelated is passed through most of the services and requests that contains the template for all the possible request variants. This makes tracking of input and output of different services very difficult. For example, when passing input parameters to the appointment cancellation service by sending relevant data to the process concerning the cancellation of an appointment, like the appointment ID and the cancellation status, the requestRelated object that contains these details within is passed instead.

Finally, as webMethods was created over 20 years ago and its popularity is diminishing, the lack of developers specialised on the software is having an impact. Onboarding and tutoring of new developers to use the solution takes considerable resources and time, due to the unique architecture of the application.

3.2.2 Locking of the services for development

TIS integration server does not have the equivalent of branch management or pushing changes to a remote repository as in, for example, the Git version control system [25]. Instead all the developers are connected directly to the test server through a virtual machine. To make changes, developers need to lock the services they are working with to avoid any conflicts with other developers who may be working on the same services. Therefore, concurrent development of updates to a service across different projects and partners is inconvenient. Whoever got the lock on the service first is able to create changes until the lock is removed.

The TIS microservice contains multiple services that are used across several systems and work with multiple different requests. As mentioned before, based on the type of the request TIS has many switch statements in services. When one of those switch blocks is updated, the service needs to be locked, rendering the creation of updates to other branches in this service impossible.

3.3 Generation and processing of XML documents

The current generation of XML requests to healthcare providers in üDR is done completely manually and from scratch. This means that the generation of specific XML requests has no limitations or boundaries, it is completely dependent on the code. Due to that the generation of the requests is error prone, as without a strict template it is possible to construct and send an XML document with any random content.

Without the structuring and limitations to the generation of the requests, the implementation heavily depends on extensive documentation and careful introduction of updates. Furthermore, without a strict structure to follow it is often difficult to detect problems. Similarly to the generation of XML requests, the processing of XML responses from healthcare providers is also extremely unreliable.

4. Reworks and the microservice architecture

To solve the aforementioned problems and remove the interdependence between TIS and üDR, a fully scalable, JVM-based (Java Virtual Machine) architecture [26] written on Micronaut [27] microservices was proposed. The main problems this solution fixes are as follows:

- **Reduce the interdependence between TIS and üDR** - all the validation checks, data operations and communication with healthcare providers are removed from TIS and implemented through microservices
- **Improve the performance of üDR and TIS** - üDR has separate, fully scalable microservices responsible for its operations and TIS does not have to process requests sent form üDR
- **Send half the amount of XML messages through the X-Road** - üDR communicates with the microservices using JSON (JavaScript Object Notation) [28] requests, microservices are responsible for the generation of XML requests and communicate directly with the healthcare providers
- **Improve the generation of XML requests** - generation of JAXB (Java Architecture for XML Binding) [29] XML schemas is implemented through the microservices to improve the reliability, readability and performance of the creation of XML requests and the processing of the responses
- **Improve code maintainability** - TIS WebMethods visual code is rewritten to Java 16 [30] based microservices

Subsequently, four different microservices are implemented, which are responsible for the following tasks:

- **DR-Commons (CRAB)** - generation of JAXB Java objects from XML schemas and provisions of common resources across all microservices
- **DR-Admin (SPONGE)** - provision of healthcare providers' X-Road communication details, enabling the data transfer between microservices and healthcare providers

- **DR-Slots (SPRUT)** - microservice responsible for all the operations related to available time slots requests, generation of XML requests, validation checks, communication with the healthcare providers, handling and parsing of the XML responses
- **DR-Appointments (ORCA)** - microservice responsible for all the operations related to creation, changing and cancellation of appointments, generation of XML requests, validation checks, communication with healthcare providers, handling and parsing of XML responses

In the following chapters we will discuss the architecture and general work of all the aforementioned microservices.

4.1 Common resources provisioner DR-Commons CRAB

CRAB or Common Resources Access Bank provisions common resources to both SPRUT and ORCA microservices. CRAB is written in Micronaut using Java 16 and consists of three main subprojects: DR-Schemas, DR-Xroad and DR-Commons-Http-Client.

4.1.1 XML schema generation with DR-Schemas

All the requests and responses sent to and from the healthcare providers have a unique format, consisting of fields relevant to the data necessary for the exchange of information. The format and data contained in all the requests and responses through which üDR communicates with the providers is discussed in more detail in the following chapters describing the SPRUT and ORCA microservices. In this chapter we will review the technology DR-Schemas uses to generate Java objects from XML schemas.

DR-Schemas uses a Java Architecture for XML Binding or JAXB framework that maps XML elements and attributes to Java fields and properties using Java annotations. It provides a fast and convenient way to marshal Java objects into XML and unmarshal XML into objects. To generate Java objects, JAXB takes .xsd XML schemas as input.

All the standards and .xsd schemas relevant to health data transfer in Estonia are available through TEHIK’s publication centre. Necessary standards are imported to the DR-Schemas microservice, specified in the Java object generation script and converted to

Java objects. Generated sources are published to the TEHIK artifact repository manager Artifactory [31], from where they can be imported to the SPRUT and ORCA microservices.

In addition to the generation of the Java objects for reliable creation of XML documents, another subproject DR-Commons-Jaxb provides SPRUT and ORCA with marshalling and unmarshalling tools for the conversion of the created Java request objects to XML request documents and the conversion of healthcare provider XML response documents to Java response objects.

4.1.2 X-Road message utilities with DR-Xroad

To send requests and responses through the X-Road Security Servers, all the XML documents need to be decorated with X-Road headers using the Simple Object Access Protocol or SOAP [32]. In the context of Estonia's e-health information exchange, these headers usually contain information about the sender and the recipient of the document, their respective IDs and addresses, version of the currently used protocols and other minor details.

DR-Xroad microservice provides a framework for the assembly and decoration of the SOAP messages and methods for the sending and receiving of the SOAP requests and responses though the X-Road security servers. The microservice is also responsible for the validation of the SOAP headers and the handling of X-Road security server related errors.

4.1.3 Base HTTP client with DR-Commons-Http-Client

The communication between üDR and the SPRUT and ORCA microservices will be done through HTTP (Hypertext Transfer Protocol) [33] clients using JSON requests and responses to convey relevant information. Therefore, the DR-Commons-Http-Client microservice is implemented to provide a common resource for building, sending, retrieving and parsing the different HTTP requests and responses and the decoration of necessary HTTP headers.

4.2 X-Road properties microservice DR-Admin SPONGE

As mentioned in one of the previous chapters, to send XML documents across X-Road security servers, the IDs and addresses of the recipients need to be specified. There are five fields that need to be defined when sending the request to a recipient: their memberCode, memberClass, subsystemCode, xmlns and xroadInstance.

Every healthcare provider in Estonia has a registry code, using which the providers can be identified. All the X-Road provider details for every provider are currently stored in a tis-xroad.properties file. In the current implementation, for every request sent to a healthcare provider, the recipient details are read from the tis-xroad.properties file. This is highly inefficient as loading properties from a file using an input stream for every request operation can be very costly, especially when working with available time slots requests where the same request can be sent to multiple providers simultaneously (more on that in the SPRUT chapter).

The SPONGE or Service for Properties Organisation, Normalisation, Gathering and Exchange alleviates the issue as the provider details are requested only every configurable time period and saved to a key-value set of cached provider details in the SPRUT and ORCA microservices. When the cache expires, SPONGE is called again and the properties are loaded anew. The cache expiration time period is configured because properties are subject to change and the cache is refreshed every once in a while.

4.3 Available time slots processing with DR-Slots SPRUT

One of the main functionalities of üDR, the search of available time slots, is implemented through the SPRUT or Slots Processing and Requesting UniT. SPRUT will be responsible for the following processes:

- Retrieval of the available time slots JSON requests
- Generation of XML request documents
- Sending and receival of available time slots requests and responses from healthcare providers
- Processing of the available time slots responses
- Saving of the requests and responses to the database for statistics
- Generation and sending of the available time slots JSON responses to üDR

The sequence diagram below (Figure 3) illustrates the overall flow of an available time slots request in SPRUT, however, all the processes relevant to the request are described in more detail in the following paragraphs.

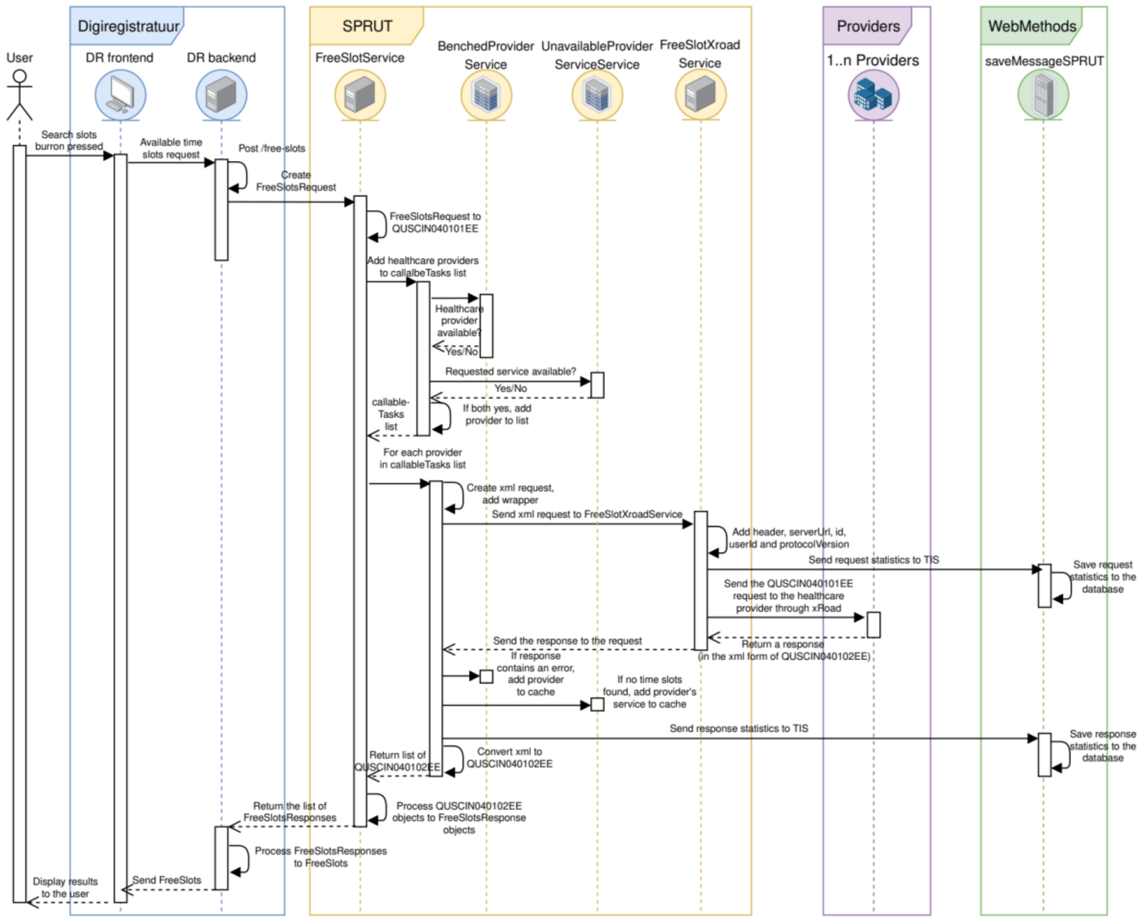

Digiregistratuur
SPRUT
Providers
WebMethods
User
DR frontend
DR backend
FreeSlotService
BenchedProvider Service
UnavailableProvider ServiceService
FreeSlotXroad Service
1..n Providers
saveMessageSPRUT
Search slots burron pressed
Available time slots request
Post /free-slots
Create FreeSlotsRequest
FreeSlotsRequest to QUSCIN040101EE
Add healthcare providers to callalbeTasks list
Healthcare provider available?
Yes/No
Requested service available?
Yes/No
If both yes, add provider to list
callable-Tasks list
For each provider in callableTasks list
Create xml request, add wrapper
Send xml request to FreeSlotXroadService
Add header, serverUrl, id, userId and protocolVersion
Send request statistics to TIS
Save request statistics to the database
Send the QUSCIN040101EE request to the healthcare provider through xRoad
Return a response (in the xml form of QUSCIN040102EE)
Send the response to the request
If response contains an error, add provider to cache
If no time slots found, add provider's service to cache
Send response statistics to TIS
Save response statistics to the database
Convert xml to QUSCIN040102EE
Return list of QUSCIN040102EE
Process QUSCIN040102EE objects to FreeSlotsResponse objects
Return the list of FreeSlotsResponses
Process FreeSlotsResponses to FreeSlots
Send FreeSlots
Display results to the user

Figure 3. Sequence diagram illustrating the processing of available time slots requests with SPRUT

4.3.1 Receival of the free time slots requests

Instead of composing an XML request document and sending it to TIS, üDR will now create a FreeSlotsRequest class JSON request, containing all the data relevant to the retrieval of available time slots. The FreeSlotsRequest class contains information displayed on Figure 4.

```
public class FreeSlotsRequest {
  private String requestIdRoot;
  private String requestIdExtension;

  private String requestAuthorIdRoot;
  private String requestAuthorIdExtension;
  private String requestAuthorGivenName;
  private String requestAuthorFamilyName;
  private String representedOrganizationIdRoot;
  private String representedOrganizationIdExtension;
  private String representedOrganizationName;

  private String patientIdRoot;
  private String patientIdExtension;
  private String patientGivenName;
  private String patientFamilyName;

  private String queryIdRoot;
  private String queryIdExtension;

  private LocalDate dateFrom;
  private LocalDate dateTo;

  private String clinicalServiceCodeSystem;
  private String clinicalServiceCodeSystemName;
  private String clinicalServiceCode;
  private String clinicalServiceName;

  private List<XroadProviderDetails> xroadProvidersDetails;

  private String payerCode;
  private String initialQuantity;
  private String priorityCode;
}
```

Figure 4. FreeSlotsRequest class used for communication between üDR and SPRUT

The first nine fields of the request from requestIdRoot to representedOrganizationName are mostly administrative and represent data concerning the ID, the author of the request and the organisation where the request was made from. As in üDR the request for available time slots can be done not only by the patient himself, but also by his custodians, like parents or careholders, the request must contain information about the requests author and about the patient separately. In case the request was created by a doctor, information relevant to the organisation where the request was sent from is added as well.

Next, the request contains information about the patient the request is made for, followed by the queryId fields. The difference between the requestId and queryId is that the requestId-s represent unique identifiers concerning any type of requests sent from üDR application. QueryId is a more specific identifier, specifying the unique FreeSlotsRequest query.

In addition to that, all of the ID fields later assembled into HL7 XML request documents contain two values: the root of the ID and the extension. Root of the IDs identify what type of the ID is currently represented, for example all the roots representing the ID of a patient are 1.3.6.1.4.1.28284.6.2.2.1, while all the roots representing the ID of a queryID are 1.3.6.1.4.1.28284.9.6.3. The extension of the ID is the actual unique identifier.

Following we have the filters, describing the time period within which we want to search for available time slots, fields identifying the service that we are searching an appointment for and a list of healthcare providers in whose available time slots we are interested in. For the available time slots request, we can concurrently search for slots in multiple healthcare providers, but only for one specific service, like a visit to a gynaecologist or a cardiologist. Below is the XroadProviderDetails class, containing data about one of the providers the request is made to (Figure 5).

```
public class XroadProviderDetails {
  private String providerRegCode;
  private String memberCode;
  private String memberClass;
  private String subsystemCode;
  private String xmlns;
  private String xroadInstance;
}
```

Figure 5. XroadProviderDetails class containing healthcare provider X-Road details

As the SPONGE microservice will be responsible for the decoration of the X-Road details described in the XroadProviderDetails class, when sending the FreeSlotsRequest to the SPRUT from üDR, only the providerRegCode field is filled, identifying the registry code of one of the providers the request must be made to.

Finally we have the payerCode, identifying whether we want to search for available time slots where payment is covered by Estonia's Health Insurance Fund, slots the patient has to pay by himself, or both alternatives. InitialQuantity identifies the maximum number of available time slots we want to query from every healthcare provider and the priorityCode shows whether the priority of finding a time slot is urgent. The distinction between different priorityCodes is used only by some of the healthcare providers that keep a reserve of available time slots unavailable to the public, for cases when the request is made by emergency services. After receiving the FreeSlotsRequest, SPRUT will start the generation of the HL7 XML request document, described in the following chapter.

4.3.2 Generation of the available time slots XML request document

The unique codename for the HL7 free slots XML request document is QUSC_IN040101EE. It follows a similar structure to the HL7 appointment cancellation document we discussed in the X-Road, HL7 and TIS chapter (Figure 2), but instead of the to-be cancelled appointment ID and reason for the cancellation, it contains data relevant to the search of the available time slots. All of the data sent with the FreeSlotsRequest is used to create the QUSC_IN040101EE document.

For reliable generation, we import the generated Java classes from the CRAB subproject DR-Schemas. The generated QUSCIN040101EE Java class has the identical inner structure to the document it represents, and after the decoration and filling of all the necessary fields, we can seamlessly convert the QUSCIN040101EE Java object to the QUSC_IN040101EE HL7 XML document with the help of DR-Commons-Jaxb. Here is a small snippet of the code responsible for the generation of the base QUSCIN040101EE Java object and its ID and creationTime fields (Figure 6).

```
QUSCIN040101EE request = new QUSCIN040101EE();
request.setITSVersion("XML_1.0");

//Id
II id = new II();
id.setRoot(freeSlotsRequest.getRequestIdRoot());
id.setExtension(freeSlotsRequest.getRequestIdExtension());
request.setId(id);

//Creation Time
TS creationTime = new TS();
DateFormat df = new SimpleDateFormat("yyyyMMddHHmmss");
df.setTimeZone(TimeZone.getTimeZone("Europe/Tallinn"));
creationTime.setValue(df.format(new Date()));
request.setCreationTime(creationTime);
```

Figure 6. Generation of HL7 V3 free slots request document ID and creationTIme fields within SPRUT

All the other fields relevant to the QUSCIN040101EE request are mapped in a similar fashion, and after finalising the whole document structure, the QUSCIN040101EE java object is converted to a XML document using the JAXB marshaller from the DR-Commons-Jaxb project. Below is an example of the beginning of a generated QUSC_IN040101EE XML document (Figure 7).

```
<?xml version="1.0" encoding="UTF-8"?>
<QUSC_IN040101EE ITSVersion="XML_1.0"
xsi:schemaLocation="urn:hl7-org:v3
http://pub.e-tervis.ee/standards2/Schema/V3/HL7-ORG-V3-2007-01-
EE-DR-Ext-
V2/processable/multicacheschemas/QUSC_IN040101_EE03.xsd"
xmlns="urn:hl7-
org:v3"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <id root="1.3.6.1.4.1.28284.6.2.4.14" extension="DR.04034"/>
    <creationTime value="20220326132021"/>
    . . .
```

Figure 7. First lines of the standard HL7 V3 free slots request XML document

As all the requests sent to the different healthcare providers are the same, the generic request is generated only once, and before sending the request to a specific provider, the requests' recipients ID and addresses are added.

4.3.3 Communication with healthcare providers

As discussed previously, the FreeSlotsRequest’s XroadProviderDetails list only contains the registry codes of the healthcare providers we want to query. At this stage we use the help of the SPONGE microservice to find the providerDetails of all the requested healthcare providers.

After the provider details are added to the request, it is decorated with the HL7 SOAP wrapper and sent to the provider with the help of DR-Xroad service. The sending of free slots requests to different healthcare providers is asynchronous, meaning that multiple requests are sent at the same time.

After sending the request through the X-Road security server to a healthcare provider, there are four main cases for responses we can receive:

- **X-Road exception** - an X-Road exception may occur for example when the provider fails to send a response within a set timeout period, when the request contained a mistake or another error within the X-Road, unrelated to the request itself
- **Provider exception** - an exception sent from the provider’s healthcare system, specifying the details of what went wrong with the request. For example when requesting available time slots for a service the healthcare institution does not provide.

- **Not found** - a response indicating that no time slots were available for the specified service in the requested period of time
- **OK** - a response containing a list of available time slots

In the scenario where a healthcare provider responds to a free slot request with a timeout exception, or another error, a circuit breaker is implemented to stop the application from overburdening the healthcare providers who are experiencing problems. Requests to such providers are suspended for a configurable period of time. When the healthcare provider responds with a NF or not found response, requests for this specific service within the requested time period are also halted. The custom circuit breakers are implemented with the use of expirable caches. Caches are always checked before sending requests to healthcare providers.

4.3.4 Processing of the free time slots responses

The HL7 standard document for the available time slot responses is QUSC_IN040102EE. In structure the response document follows a very similar structure to all the other standardised HL7 documents: it contains information relevant to the request itself like the IDs, sender and recipient, and other details. The core of the response consists of resourceSlot fields that are grouped into subjects.

The response with an OK status can contain one or more subjects. From the business logic side of view, subjects are considered as slot groups. Slot groups are, as the name suggests, groups of slots that share the same exact address (a healthcare provider can have the same service available across several facilities), share the same practitioner and share the same payment details (whether covered by the health insurance fund or are paid by the patient and have the same price). Below a slightly simplified version of a subject element in the free slots response is displayed (Figure 8).

```
<subject typeCode="SUBJ">
    <schedule moodCode="SLOT">
        <!-- Address of the time slots within the slot group -->
        <indirectTarget xsi:nil="false">
            <identifiedEntity classCode="IDENT">
                <identifiedPlace classCode="PLC">
                    <id root="1.3.6.1.4.1.28284.4"
                      extension="90006399"/>
                    <code codeSystem="1.3.6.1.4.1.28284.6.2.3.2"
                          codeSystemName="EHAK"
```

```
                         code="0784" displayName="Tallinn, J.
                                             Sütiste tee 19"/>
                    <name>Põhja-Eesti Regionaalhaigla</name>
                  </identifiedPlace>
              </identifiedEntity>
           </indirectTarget>
           <!-- List of the appointment slots -->
           <resourceSlot moodCode="SLOT"> ... </resourceSlot>
           <resourceSlot moodCode="SLOT"> ... </resourceSlot>
       </schedule>
       <billableClinicalService.code>
           <!-- Service of the appointment slots -->
          <value codeSystem="1.3.6.1.4.1.28284.6.2.1.293.4"
                 codeSystemName="Ambulatoorsed vastuvõtud"
                 code="AOF001"
                 displayName="silmaarsti vastuvõtt">
              <originalText>
                  Selgitav tekst
               </originalText>
              <qualifier>
                  <!-- Category of the service -->
                  <name code="E630"
                    codeSystem="1.3.6.1.4.1.28284.6.2.1.4.5"
                        codeSystemName="Eriala"
                        displayName="oftalmoloogia -
                                    silmakirurgia"/>
              </qualifier>
          </value>
           <cost value="50" currency="EUR"/>
       </billableClinicalService.code>
       <payor.code>
           <value code="PAT"/>
       </payor.code>
   </subject>
```

Figure 8. Standard subject element of the HL7 V3 free slots response document

When a healthcare provider has available time slots for a requested service across two different practitioners, and one of them has available time slots across two different facilities, but all the time slots share the same payment details, the response will contain three different subjects or slot groups:

- Practitioner A at address A with payment A
- Practitioner B at address A with payment A
- Practitioner B at address B with payment A

The actual time slots are contained within the available subjects, and contain information on the appointment type, time and the practitioner (Figure 9).

```
<resourceSlot moodCode="SLOT">
    <!-- Slot ID -->
    <id root="1.3.6.1.4.1.28284.1.3" extension="A039286"/>
    <!-- Appointment type -->
    <code code="1" codeSystem="1.3.6.1.4.1.28284.6.2.1.30.2"
          codeSystemName="Visiidi liik" displayName="esmane"/>
    <!-- Appointment status -->
    <statusCode code="new"/>
    <!-- Appointment time -->
    <effectiveTime value="201904311500"/>
    <!-- Appointment order nr -->
    <repeatNumber value="1"/>
    <directTarget xsi:nil="false">
        <identifiedEntity>
           <identifiedPerson xsi:nil="false">
                <!-- Practitioner's code -->
               <id root="1.3.6.1.4.1.28284.6.2.4.9"
                   extension="D12345"/>
               <name>
                   <!-- Practitioner's family name -->
                   <given>Kuusk</given>
                   <!-- Practitioner's given name -->
                   <family>Mari</family>
               </name>
           </identifiedPerson>
        </identifiedEntity>
    </directTarget>
</resourceSlot>
```

Figure 9. Standard resourceSlot element of the HL7 V3 free slots response document

Information relevant to the location, service and payment details for the appointment slot are available from the subject fields, as all these details are shared across all the time slots in a slot group. Similarly to how the generated QUSCIN040101EE Java request object was marshalled into an XML document with the help of DR-Commons-Jaxb, the available time slots response XML document QUSC_IN040102EE is unmarshalled into the respective QUSCIN040102EE Java object.

4.3.5 Free slot request and response statistics

All the available time slot requests and responses are saved to the TIS database table QUSC_FREE_SLOT, to enable the gathering of statistics and tracking of erroneous situations. In addition to the whole XML document, the table tracks the ID, status, type, provider, X-Road ID and the creation time of the message. If the available time slot request failed or the response contained an error, the error message is also saved. To save the information, SPRUT sends the request or response details to the TIS WebMethods service saveMessageSPRUT, responsible for writing data to the QUSC_FREE_SLOT database (Figure 10).

```
public void sendStatistics(FreeSlotStatistics
            freeSlotStatistics) throws ServiceException {
  Values stats = new Values();
  stats.put("QUERY_ID", freeSlotStatistics.getQueryId());
  stats.put("QUERY_ID_OID", freeSlotStatistics.getQueryIdOid());
  stats.put("STATUS_CODE", freeSlotStatistics.getStatusCode());
  stats.put("ACT_TYPE_CODE",
             freeSlotStatistics.getActTypeCode());
  stats.put("ERROR_MESSAGE",
             freeSlotStatistics.getErrorMessage());
  stats.put("ORGANIZATION_ID",
             freeSlotStatistics.getOrganizationId());
  stats.put("ORGANIZATION_ID_OID",
             freeSlotStatistics.getOrganizationIdOid());
  stats.put("XROAD_ID", freeSlotStatistics.getXroadId());
  stats.put("XML_CONTENT", freeSlotStatistics.getXmlContent());
  stats.put("INSERT_TIME", freeSlotStatistics.getInsertTime());

  wmContext.invoke("DHR_QUSC_FreeSlot.work.utils",
                   "saveMessageSPRUT", stats);
}
```

Figure 10. Saving of SPRUT request and response statistics through TIS

4.3.6 Generation and sending of free slot responses to üDR

After all the requested providers have returned a response or a configured timeout has expired, all the QUSC_IN040102EE XML responses are gathered and processed into FreeSlotsResponse Java objects. Every response is parsed individually and saved into a singular FreeSlotsResponse. A list of FreeSlotsResponses is subsequently sent in the JSON format back to üDR. The general structure of the HL7 response document is also preserved in the Java format, meaning that a FreeSlotsResponse (Figure 11) object contains a list of subjects or SlotGroups (Figure 12), that each contain one or many time slots or FreeSlots (Figure 14).

```
public class FreeSlotsResponse {
    private String queryIdRoot;
    private String queryIdExtension;
    private String queryResponseCode;
    private List<SlotGroup> slotGroups;
}
```

Figure 11. FreeSlotsResponse class returned by SPRUT to üDR

Responses that did not contain a single available time slot or that contained an error are only saved to the QUSC_FREE_SLOT table and not returned to üDR in the form of a FreeSlotsResponse. Therefore, if time slots from three providers were requested, and only two of them had the response status OK, a list of two FreeSlotsResponses is returned to üDR.

```
public class SlotGroup {
    private String providerRegistryIdRoot;
    private String providerRegistryIdExtension;

    private String locationCodeSystem;
    private String locationCodeSystemName;
    private String locationCode;
    private String locationDisplayName;
    private String locationName;

    private String serviceCodeSystem;
    private String serviceCodeSystemName;
    private String serviceCode;
    private String serviceDisplayName;

    private String qualifierCode;
    private String qualifierCodeSystem;
    private String qualifierCodeSystemName;
    private String qualifierDisplayName;

    private String servicePrice;
    private String serviceCurrency;
    private String payerCode;
    private Map<String, String> originalText;

    private List<TimeSlot> availableTimeSlots;
}
```

Figure 12. SlotGroup class of SPRUT's FreeSlotsResponse

All the relevant fields of every response with status OK that were previously unmarshalled into QUSCIN040102EE Java objects are mapped into FreeSlotsResponse objects. Below is a small example of how the location of the QUSCIN040102EE Subject is converted into respective fields in a SlotGroup (Figure 13).

```
PRSCMT050101UV01Place place =
         subject.getSchedule().getIndirectTarget().get(0)
        .getIdentifiedEntity().getIdentifiedPlace().getValue();

slotGroup.setProviderRegistryIdRoot(place.getId().get(0)
                    .getRoot());
slotGroup.setProviderRegistryIdExtension(place.getId().get(0)
                                         .getExtension());

slotGroup.setLocationName(place.getName().get(0).getContent() ==
null ? ""
     : place.getName().get(0).getContent().get(0).toString());
slotGroup.setLocationCodeSystem(place.getCode()
                                .getCodeSystem());
slotGroup.setLocationCodeSystemName(place.getCode()
                    .getCodeSystemName());
slotGroup.setLocationCode(place.getCode().getCode());
slotGroup.setLocationDisplayName(place.getCode()
                 .getDisplayName());
```

Figure 13. Mapping of the HL7 V3 Subject address XML element into SPRUT SlotGroup fields

Here the retrieval of the PRSCMT050101UV01Place object illustrates how the unmarshalled QUSCIN040102EE class replicates the structure of the identifiedPlace element in the available time slots response document shown in Figure 8. Finally, below is the structure of a TimeSlot Java class, lists of which are contained within SlotGroups.

```
public class TimeSlot {
  private String slotIdRoot;
  private String slotIdExtension;

  private String appointmentTypeCode;
  private String appointmentTypeCodeSystem;
  private String appointmentTypeCodeSystemName;
  private String appointmentTypeDisplayName;
  private String appointmentStatusCode;
  private String effectiveTime;
  private Integer repeatNumber;

  private String physicianIdRoot;
  private String physicianIdExtension;
  private String physicianGivenName;
  private String physicianFamilyName;
}
```

Figure 14. TimeSlot class of a SlotGroup contained in SPRUT's FreeSlotResponse

After the processing of all the QUSCIN040102EE responses with status OK into FreeSlotsResponses, the list of responses is returned back to üDR. With all these steps the workflow of a single available time slots request is complete. Next we will overview the work of the DR-Appointments microservice ORCA.

4.4 Appointments creation with DR-Appointments ORCA

ORCA or Operator for Reserving and Cancelling Appointments is the microservice responsible for two request types: appointment creation and cancellation request.

Contrary to querying data with the SPRUT’s available time slots request, ORCA will create and cancel new and existing appointments and will require connection with several data tables that oversee the states of appointments in TIS. While the general structure for the creation of XML documents, marshalling and unmarshalling of requests and responses and the communication with healthcare providers will remain similar to the one used within SPRUT’s architecture, creation of some additional documents and functions will be necessary.

The process of the creation of an appointment is the most data intensive out of the two requests managed by ORCA. Here is an overview of all the steps the microservice performs in the process of the creation of an appointment:

- Retrieval and validation of the üDR appointment creation request
- Generation of the appointment creation request XML document
- Sending and receival of appointment create request and response from the provider
- Processing of the appointment creation response and the publication of related documents and database entries if the creation of an appointment was successful
- Saving of request and response messages in the database
- Generation and sending of the new appointment creation JSON response to üDR

Below you can see a comprehensive sequence diagram, illustrating the flow of an appointment creation request (Figure 15).

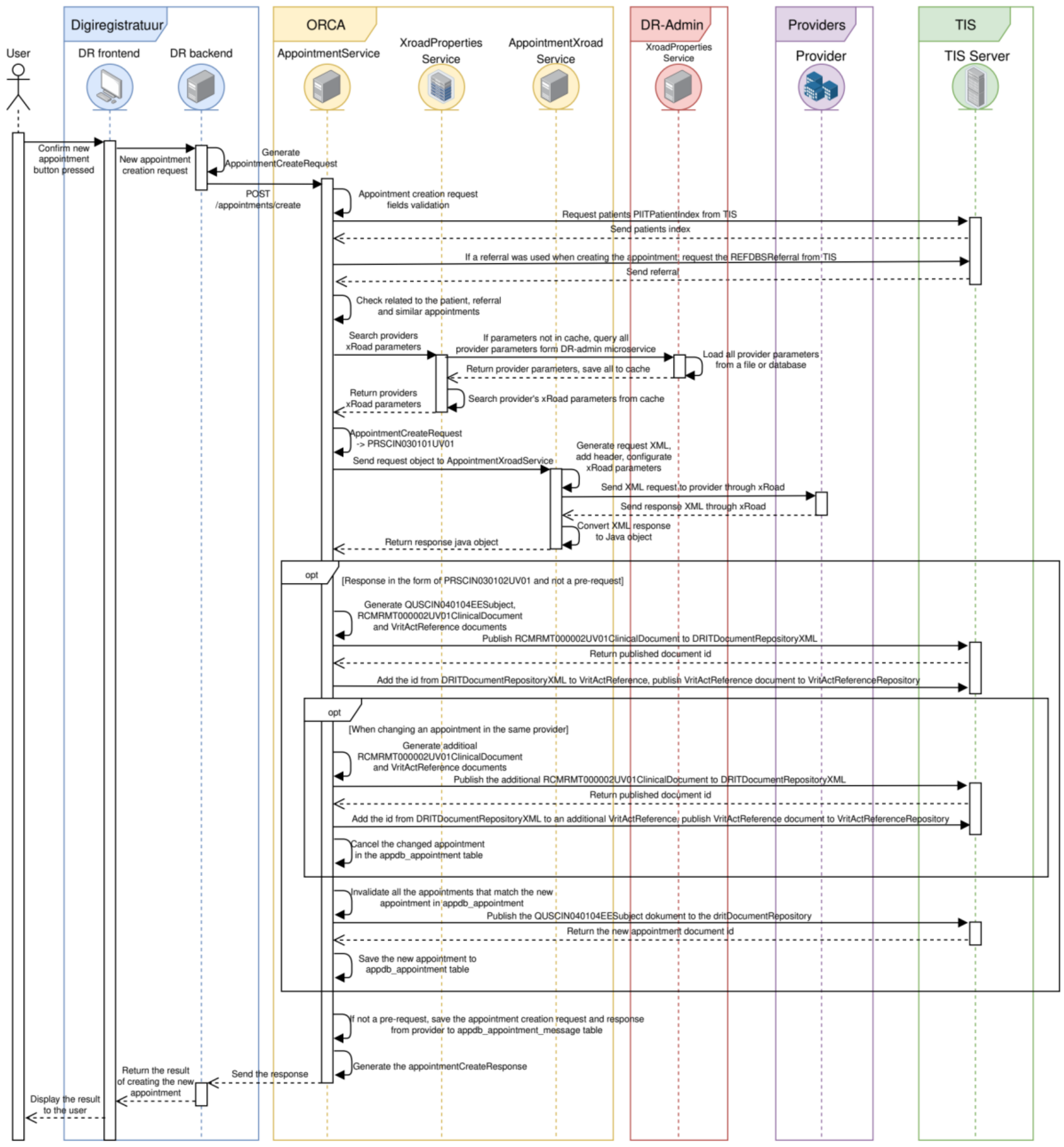

Digiregistratuur
ORCA
DR-Admin
Providers
TIS
User
DR frontend
DR backend
AppointmentService
XroadProperties Service
AppointmentXroad Service
XroadProperties Service
Provider
TIS Server
Confirm new appointment button pressed
New appointment creation request
Generate AppointmentCreateRequest
POST /appointments/create
Appointment creation request fields validation
Request patients PIITPatientIndex from TIS
Send patients index
If a referral was used when creating the appointment, request the REFDBSReferral from TIS
Send referral
Check related to the patient, referral and similar appointments
Search providers xRoad parameters
If parameters not in cache, query all provider parameters form DR-admin microservice
Load all provider parameters from a file or database
Return provider parameters, save all to cache
Return providers xRoad parameters
Search provider's xRoad parameters from cache
AppointmentCreateRequest -> PRSCIN030101UV01
Send request object to AppointmentXroadService
Generate request XML, add header, configurate xRoad parameters
Send XML request to provider through xRoad
Send response XML through xRoad
Convert XML response to Java object
Return response java object
opt
[Response in the form of PRSCIN030102UV01 and not a pre-request]
Generate QUSCIN040104EESubject, RCMRMT000002UV01ClinicalDocument and VritActReference documents
Publish RCMRMT000002UV01ClinicalDocument to DRITDocumentRepositoryXML
Return published document id
Add the id from DRITDocumentRepositoryXML to VritActReference, publish VritActReference document to VritActReferenceRepository
opt
[When changing an appointment in the same provider]
Generate additioal RCMRMT000002UV01ClinicalDocument and VritActReference documents
Publish the additional RCMRMT000002UV01ClinicalDocument to DRITDocumentRepositoryXML
Return published document id
Add the id from DRITDocumentRepositoryXML to an additional VritActReference, publish VritActReference document to VritActReferenceRepository
Cancel the changed appointment in the appdb_appointment table
Invalidate all the appointments that match the new appointment in appdb_appointment
Publish the QUSCIN040104EESubject dokument to the dritDocumentRepository
Return the new appointment document id
Save the new appointment to appdb_appointment table
If not a pre-request, save the appointment creation request and response from provider to appdb_appointment_message table
Generate the appointmentCreateResponse
Send the response
Return the result of creating the new appointment
Display the result to the user

Figure 15. Sequence diagram illustrating the appointment creation process in ORCA

4.4.1 Retrieval and validation of the appointment creation request

Analogously to the FreeSlotsRequest, instead of the generation of an appointment creation XML document, üDR will send a JSON request to the ORCA microservice. The appointment creation request, or AppointmentCreateRequest (Figure 16) consists of data relevant to the author, organisation, patient, service, payment details, provider, appointment slot and the doctor of the to-be created appointment.

```
public class AppointmentCreateRequest {

  private String queryIdExtension;
  private String creationTime;

  private String assignedPersonIdRoot;
  private String assignedPersonIdExtension;
  private String assignedPersonFamilyName;
  private String assignedPersonGivenName;

  private String representedOrganizationIdRoot;
  private String representedOrganizationIdExtension;
  private String representedOrganizationName;

  private String changedAppointmentIdRoot;
  private String changedAppointmentIdExtension;

  private String statusCode;
  private String priorityCode;

  private String patientIdRoot;
  private String patientIdExtension;
  private String patientEmail;
  private String patientPhoneNumber;
  private String patientMobileNumber;
  private String patientFamilyName;
  private String patientGivenName;
  private String patientSuffixName;

  private String serviceCodeSystem;
  private String serviceCodeSystemName;
  private String serviceCode;
  private String serviceDisplayName;
  private Boolean serviceRequiresReferral;

  private String referralIdRoot;
  private String referralIdExtension;

  private String payorCode;
  private String payorIdRoot;
  private String payorIdExtension;
  private String payorEmail;
  private String payorName;

  private String providerIdRoot;
  private String providerIdExtension;
  private String providerCodeSystem;
  private String providerCodeSystemName;
```

```
   private String providerCode;
   private String providerDisplayName;
   private String providerName;

   private String slotIdRoot;
   private String slotIdExtension;
   private String slotEffectiveTime;

   private String doctorIdRoot;
   private String doctorIdExtension;
   private String doctorFamilyName;
   private String doctorGivenName;
}
```

Figure 16. AppointmentCreateRequest class used for communication between üDR and ORCA

With most of the fields self-explanatory and similar to those used within the FreeSlotsRequest, there are some that require some context. The changedAppointmentId root and extension fields are used when we are changing an appointment within the same healthcare provider. Usually, when changing an existing appointment through üDR, a new appointment creation request is sent, and after a successful response, the old appointment is cancelled. However, when we are changing the appointment to a new one within the same healthcare provider, the sending of two separate requests is unnecessary, as the HL7 standards support the special version of the appointment creation request that contains the cancellation info of the previous appointment.

Further, there are the referralId root and extension fields that are specified when creating an appointment by using a referral. üDR supports the creation of appointments with patient’s referrals when a patient searches for appointments through their Open Referrals view. After the creation of an appointment with a referral, the referral will be marked as utilised and unavailable, unless the appointment is changed or cancelled.

Before generating the request document, two pieces of additional information is searched for in the TIS databases and several validation checks are performed. Firstly, we search TIS for the patient’s index using the patient's personal code. This index is used across TIS databases when persisting data connected to patients. Secondly, if referralId root and extension fields were specified in the request, additional data about the referral is requested. Four main validation check types are performed:

- Existence of all the required AppointmentCreateRequest fields is checked
- When cheangedAppointmentId fields are specified, we check that both changed appointment and to-be-created appointment are in the same provider
- Multiple validation checks connected to the referral, for example whether the referral with the specified ID exists and if the patient connected to the referral is the same for who we are creating an appointment
- Checks for similar appointments, whether patient does not already have two or more appointments for the same service without a referral, where Estonia's Health Insurance Fund is the payer

After all the validation checks are passed, the generation of the appointment creation request HL7 document can begin.

4.4.2 Generation of the appointment creation XML request document

The HL7 standard for the appointment creation request document is PRSC_IN030101UV01. The generation process for the PRSC_IN030101UV01 XML document is exactly the same as the generation of the available time slots request: the AppointmentCreateRequest fields are mapped to the PRSCIN030101UV01 class generated by DR-Scehmas, and DR-Commons-Jaxb marshals the object into the HL7 request document. The main differences between the different requests are their content and size, as the appointment creation request contains significantly more data than the available time slots request.

4.4.3 Communication with the healthcare provider

Contrary to the available time slots request, the appointment creation request is sent to only one healthcare provider. Due to that we don’t have to worry about the asynchronisation of several messages within the same appointment creation request. Aside from that, we follow the same process as we did before: the provider details are loaded with the help of DR-Admin microservice, the X-Road SOAP headers are decorated and the request is sent to the healthcare provider by using the DR-Xroad microservice.

The response returned by the healthcare provider however is slightly different. There are two different valid document types that a provider can send as an appointment creation response:

- PRSC_IN030102UV01 - Appointment creation request succeeded
- PRSC_IN030103UV01 - Appointment creation request failed

The PRSC_IN030103UV01, or creation failure document, follows a simple structure and its main content is within the reasonOf (Figure 17) block that contains the detectedIssueEvent, which describes why the appointment creation was unsuccessful.

```
<reasonOf xsi:nil="false">
  <detectedIssueEvent>
    <code codeSystem="1.3.6.1.4.1.28284.6.2.2.33.1.1"
          code="TAKEN"/>
    <text> Soovitud vastuvõtu aeg broneeriti kellegi teise poolt
           01.04.2017 kell 13:25:29. Palun valida mõni teine
           vaba aeg.
    </text>
  </detectedIssueEvent>
</reasonOf>
```

Figure 17. ReasonOf element of the HL7 V3 appointment creation failure response

The example above describes that the requested appointment slot was taken by another patient, and asks the user to choose another available time. Next we will outline the processing of the PRSC_IN030102UV01 success response and the publishing of related documents.

4.4.4 Processing and publication of appointment creation documents

When the creation of an appointment is successful, multiple different documents and entries are published to several TIS databases. Firstly, we describe the queryDocument and the vritActReference documents.

QueryDocument follows the HL7 RCMR_MT000002UV01.ClinicalDocument standard and contains information relevant to the appointment creation request. It contains information about the patient, author, representedOrganisation, effectiveTime and appointment creation request IDs, linking it to the original request. Additionally, Figure 18 displays how codes indicating that the document is connected to the appointment creation request are added.

```
//RelatedDocument -> ParentDocument -> Id
II parentDocumentId = new II();
parentDocumentId.setRoot(request.getId().getRoot());
parentDocumentId.setExtension(request.getId().getExtension());
parentDocument.getId().add(parentDocumentId);

//RelatedDocument -> ParentDocument -> Code
CD parentDocumentCode = new CD();
parentDocumentCode.setCode("appointment_new_request");
parentDocumentCode.setCodeSystem(
                   "1.3.6.1.4.1.28284.6.2.2.32.11");
parentDocumentCode.setCodeSystemName(
                   "Viidaregistri kande tüüp");
parentDocumentCode.setDisplayName("Uue broneeringu nõue");
parentDocument.setCode(parentDocumentCode);
```

Figure 18. Mapping of ID and code fields of the QueryDocument publication document in ORCA

After the RCMRMT000002UV01ClinicalDocument Java object is generated, it is converted to the XML format with our familiar DR-Commons-Jaxb service and the document is published to the DRDB_XML_DOCUMENT database, through the TIS service DRITDocumentRepositoryXML.service.insertXmlMetadataOrContent. After saving the document, TIS returns the created drit_documentId, which we will use when saving the vritActReference.

For the VritActReference, no XML documents are created and fields related to the object are saved directly into the columns of the database in TIS. The VritActReference relevant to the appointment creation request contains information shown below (Figure 19).

```
VritActReference actReference = new VritActReference();
actReference.setOriginalId(request.getId().getExtension());
actReference.setOriginalIdOid(request.getId().getRoot());
actReference.setClinicalDocumentCode("appointment_new_request");
actReference.setClinicalDocumentCodeOID(
                 "1.3.6.1.4.1.28284.6.2.2.32.11");

actReference.setDrdbReferenceId(dritDocumentId);
actReference.setDrdbDocumentId(dritDocumentId);

actReference.setActTypeCode("query");
actReference.setActTypeCodeOID("1.3.6.1.4.1.28284.6.2.2.32.1");
actReference.setTemplateId(request.getTemplateId().get(0)
                  .getExtension());
actReference.setTemplateIdOID(request.getTemplateId().get(0)
                     .getRoot());
```

```
actReference.setEffectiveTime(request.getCreationTime()
                    .getValue());
actReference.setStatusCode("completed");
actReference.setAuthorId(appointmentRequest
                   .getAssignedPersonIdExtension());
actReference.setAuthorIdOID(appointmentRequest
                   .getAssignedPersonIdRoot());
actReference.setOrganizationId(appointmentRequest
                .getRepresentedOrganizationIdExtension());
actReference.setOrganizationIdOID(appointmentRequest
                   .getRepresentedOrganizationIdRoot());
actReference.setPatientId(patientIndex);
actReference.setPatientIdOID(personIdentifierOidVrit);
```

Figure 19. Mapping of the VritActReference fields in ORCA's appointment creation process

In addition to the information about the author, patient, representedOrganisation, effectiveTime and ID of the appointment request, VritActReference contains the reference to the just saved queryDocument. VritActReference is saved to the VRDB_ACT_REFERENCE table, through the TIS service vritActReferenceRepository.service.insertActReference.

In case the AppointmentCancelRequest contains the changedAppointmentId fields, an additional pair of queryDocument and vritActReference are created, registering the cancellation of an appointment. For that, the already created queryDocument and vritActReference documents are slightly modified and published to the TIS repositories (Figure 20).

```
actReference.setClinicalDocumentCode(
      "appointment_nullified_request");
actReference.setClinicalDocumentIdOID(
      appointmentRequest.getChangedAppointmentIdRoot());
actReference.setOriginalId(
      appointmentRequest.getChangedAppointmentIdExtension());

actReference.setDrdbReferenceId(nullifiedDritDocumentId);
actReference.setDrdbDocumentId(nullifiedDritDocumentId);
```

Figure 20. Modifying of an appointment creation VritActReference to a nullification VritActReference

When all the documents related to the request have been published, the microservice starts persisting data on the newly created appointment itself.

For every newly created appointment, an HL7 XML document of type QUSCIN040104EESubject is saved to the DRDB_XML_DOCUMENT table in the TIS database. The new appointment document encompasses information about the appointments ID, code, status, time, its patient and request author information, location of the appointment and its service, and all the details relevant to the payment process. After the mapping of all the fields is finished, the XML document is published to the database with the help of the TIS service dritDocumentRepository.service.insertDocument that returns the drit_documentId.

The main data table, through which TIS and üDR manages appointments, is APPDB_APPOINTMENT. After the publication of all the necessary XML documents, all the relevant data to the appointment is saved there. APPDB_APPOINTMENT persists data related to ten main variables explained in Table 1.

Table 1. Data persisted within the APPDB_APPOINTMENT table

Field name	**Field desciption**
Appointment ID	ID of the created appointment returned by the provider
Appointment Status	New, nullified or error
Patient ID	Person who will go to the appointment
EffectiveTime	When the appointment takes place
Referral ID	If a referral was used when creating the appointment
New appointment document ID	QUSCIN040104EESubject document ID
Provider ID	Organisation where the appointment happens
Insert and update times	Insert and update times of this entry
Service code	For which service is the appointment
Payor type	Who pays for the appointment (patient or insurance fund)

Before saving the new entry, however, all the upcoming appointments in the APPDB_APPOINTMENT database that have the same Appointment ID as the newly created appointment and have the status “new” are marked with status “error”. As the database is used by many third-party services like TIS and healthcare providers asynchronously, it is possible that more than one appointment with the same ID can be created at the same time. To avoid these situations, the most recently created appointment will be marked as the correct appointment and all the others are marked as erroneous.

Additionally, if the appointment creation request contained the changedAppointmentId fields, meaning that an appointment was being changed in the same healthcare provider, the appointment with the Appointment ID matching the changedAppointmentId will be marked as “nullified” or cancelled in the APPDB_APPOINTMENT table.

Finally, the new appointment entry is added to the database with status “new”. Let’s now take a quick look at how appointment creation request and response statistics are gathered.

4.4.5 Appointment creation request and response statistics

The TIS database table responsible for the gathering of appointment creation requests and responses is APPDB_APPOINTMENT_MESSAGE. The purpose of the table is to save the exact XML appointment creation request and response documents sent to and received from the providers. Fields that are persisted through the table are shown in Table 2.

Table 2. Data persisted within the APPDB_APPOINTMENT_MESSAGE table

Field name	Field description
Appointment ID	ID of the created appointment returned by the provider
Related appointment ID	changedAppointmentId, if it was present
X-Road ID	ID of the X-Road request or response message
XML content	Request or response document
Document type code	Appointment_tto_new_request or Appointment_tto_new_response
Error message	Error message, if the response contained one
Insert and update times	Insert and update times of this entry

The saving of the statistics messages is very similar to the gathering of available time slots request statistics. For every appointment creation request, two entries are created, one with the status “Appointment_tto_new_request” and the request document and one with the status “Appointment_tto_new_response” and the response document. After all the information connected to the creation of the appointment has been saved, ORCA sends the appointment information back to üDR.

4.4.6 Generation and sending of appointment information to üDR

ORCA always returns the AppointmentCreateResponse (Figure 21) after the creation of the new appointment is finished. If the creation of the appointment failed, the AppointmentCreateResponse fields success, rejectedByTTO (rejected by healthcare service provider), rejectionCode and rejectionReason indicate what went wrong with the request.

```
public class AppointmentCreateResponse {
  private boolean success;
  private boolean rejectedByTTO;

  private String serviceDisplayName;
  private String providerCode;
  private String providerDisplayName;
  private String doctorName;

  private LocalDateTime dateTime;
  private LocalDateTime firstAllowedDateTime;

  private Patient patient;

  private Map<String, String> description;
  private Map<String, String> descriptionHtml;
  private Boolean isFirstVisit;
  private Boolean enabled;

  private String referenceNum;
  private String referenceNumOid;

  private Address address;
  private String telecom;

  private String price;
  private Boolean freeOfCharge;
  private Boolean servicePaidByEHK;

  private String referralInfo;
  private String referralId;
  private String referralOid;

  private Map<String, String> indebtedness;
  private Map<String, String> paymentInstructions;
  private String paymentReceiver;
  private String paymentReferenceNum;
  private String paymentAccountInfo;
  private Map<String, String> paymentAccountIbanInfo;
  private String paymentAmount;
  private String paymentCurrency;
  private Boolean paid;
  private String paymentOrderNum;
  private String paymentOrderDescription;
  private LocalDateTime paymentDateTime;
  private String payerAccountNum;
  private String payerName;
  private String invoiceRows;

  private String rejectionCode;
  private String rejectionReason;

  private String statusCode;

  private List<ProviderBank> providerBanks;
  private Node appointmentNode;

  private String queryId;
  private String templateId;
  private String creationTime;
}
```

Figure 21. AppointmentCreateResponse class sent from ORCA to üDR

If the creation of the appointment was successful, ORCA returns all the information regarding the newly created appointment. In addition to details about the provider, appointment time, service and payment details, AppointmentCreateResponse contains details about the Address (Figure 22), list of banks through which the provider accepts payments (Figure 23) and the Patient (Figure 24).

```
public class Address {
  private String country;
  private String state;
  private String county;
  private String city;
  private String streetName;
  private String unitId;
  private String postalCode;
  private String streetAddressLine;
  private String additionalLocator;
  private String houseNumber;
  private String houseNumberNumeric;
  private Map<String, String> addressDirection;
}
```

Figure 22. Address class of the AppointmentCreateResponse in ORCA

```
public class ProviderBank {
  private String bankCode;
  private String vkSndId;
}
```

Figure 23. ProviderBank class of the AppointmentCreateResponse in ORCA

```
public class Patient {
  private String personalCode;
  private String firstName;
  private String lastName;
  private String suffix;
  private String mobile;
  private String phone;
  private String email;
  private Set<String> vkTypes = new HashSet<>();
  private String ttoRegCode;
}
```

Figure 24. Patient class of the AppointmentCreateResponse in ORCA

After all the response fields have been mapped, the JSON response is returned back to üDR through DR-Commons-Http-Client. In the following paragraphs we will discuss the AppointmentCancellationRequest.

4.5 Appointment cancellation with DR-Appointments ORCA

ORCAs appointment cancellation request is very similar to the appointment creation request, but requires less data and actions, as creating an appointment in TIS is more complex than cancelling one. Here is the step-by-step description of the appointment cancellation process:

- Retrieval and validation of the üDR appointment cancellation request
- Generation of the appointment cancellation request XML document
- Sending and receival of the appointment cancellation request and response from the provider
- Processing of the appointment cancellation response and the publication of related documents and database entries if the cancellation was successful
- Saving of request and response messages in the database
- Generation and sending of the appointment cancellation JSON response to üDR

Below you can see a detailed sequence diagram depicting the flow of cancelling an appointment through the ORCA microservice (Figure 25).

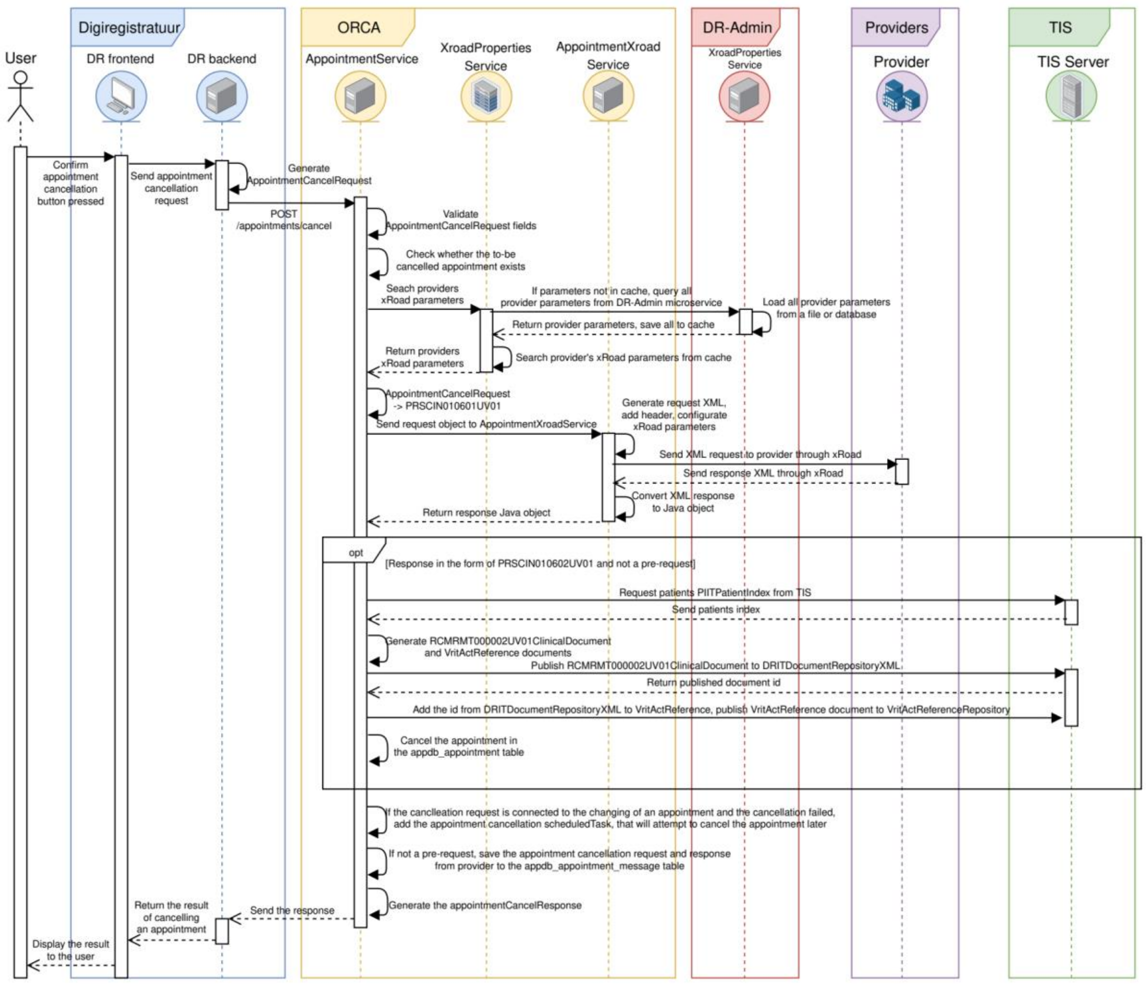

Digiregistratuur
ORCA
DR-Admin
Providers
TIS
User
DR frontend
DR backend
AppointmentService
XroadProperties Service
AppointmentXroad Service
XroadProperties Service
Provider
TIS Server
Confirm appointment cancellation button pressed
Send appointment cancellation request
Generate AppointmentCancelRequest
POST /appointments/cancel
Validate AppointmentCancelRequest fields
Check whether the to-be cancelled appointment exists
Seach providers xRoad parameters
If parameters not in cache, query all provider parameters from DR-Admin microservice
Load all provider parameters from a file or database
Return provider parameters, save all to cache
Search provider's xRoad parameters from cache
Return providers xRoad parameters
AppointmentCancelRequest -> PRSCIN010601UV01
Send request object to AppointmentXroadService
Generate request XML, add header, configurate xRoad parameters
Send XML request to provider through xRoad
Send response XML through xRoad
Convert XML response to Java object
Return response Java object
opt
[Response in the form of PRSCIN010602UV01 and not a pre-request]
Request patients PIITPatientIndex from TIS
Send patients index
Generate RCMRMT000002UV01ClinicalDocument and VritActReference documents
Publish RCMRMT000002UV01ClinicalDocument to DRITDocumentRepositoryXML
Return published document id
Add the id from DRITDocumentRepositoryXML to VritActReference, publish VritActReference document to VritActReferenceRepository
Cancel the appointment in the appdb_appointment table
If the canclleation request is connected to the changing of an appointment and the cancellation failed, add the appointment cancellation scheduledTask, that will attempt to cancel the appointment later
If not a pre-request, save the appointment cancellation request and response from provider to the appdb_appointment_message table
Generate the appointmentCancelResponse
Send the response
Return the result of cancelling an appointment
Display the result to the user

Figure 25. Sequence diagram illustarting the appointment cancellation process in ORCA

4.5.1 Retrieval and validation of the appointment cancellation request

The AppointmentCancelRequest (Figure 26) sent from üDR to the ORCA microservice is a lot less data-heavy compared to the AppointmentCreateRequest and contains information about the request author, their organisation, the identification of the to-be cancelled appointment, IDs for the new appointment (if the cancellation request is part of an appointment changing process in a different provider), reason for the cancellation (simple cancellation or changing of an appointment) and the registry code for the provider where the appointment is registered.

```
public class AppointmentCancelRequest {
  private String queryIdExtension;
  private String creationTime;

  private String assignedPersonIRoot;
  private String assignedPersonIdExtension;
  private String assignedPersonFamilyName;
  private String assignedPersonGivenName;

  private String representedOrganizationIdRoot;
  private String representedOrganizationIdExtension;
  private String representedOrganizationName;

  private String appointmentIdRoot;
  private String appointmentIdExtension;

  private String newAppointmentIdRoot;
  private String newAppointmentIdExtension;

  private String reasonCode;
  private String providerRegCode;
}
```

Figure 26. AppointmentCancelRequest class used for communication between üDR and ORCA

The microservice checks the existence of all the necessary fields in the request and additionally checks if an active appointment with the specified identifiers is present in the database. After the validation checks have passed, the patient index is loaded from the TIS database and the microservice can begin the generation of the appointment cancellation HL7 document.

4.5.2 Generation of the appointment cancellation XML request document

Appointment cancellation request HL7 standard is PRSC_IN010601UV01 and you can see an example of the document in the chapter X-Road, HL7 and TIS. All the necessary data to fill the document fields is presented in the AppointmentCancelRequest (Figure 26) and the request is generated, as per usual, with the help of the DR-Schemas and DR-Commons-Jaxb projects.

4.5.3 Communication with the healthcare provider

After the provider details are loaded through the DR-Admin microservice and the request sent to the provider through DR-Xroad, similarly to the appointment creation request, the provider can answer with two possible document types:

- PRSC_IN010602UV01 - Appointment cancellation successful
- PRSC_IN010603UV01 - Appointment cancellation failed

If the appointment cancellation was successful, the provider returns the IDs of the cancelled appointment, indicating that the appointment was cancelled. However, if the cancellation failed, in addition to the requested appointment ID, the cancellation response will contain a reason code and text, explaining why the cancellation was unsuccessful. Below are some of the reasons, why the appointment cancellation request could be rejected by the provider (Table 3).

Table 3. Possible reasons for the rejection of an appointment cancellation request

Rejection code	Rejection description
CANCELLED	The appointment has already been cancelled
LATE	It is too late to cancel the appointment (some providers decline cancellations of appointments if the appointment is happening soon)
UNKNOWN	Unknown reason, more info from the healthcare provider
OCCURRED	The appointment has already occurred

4.5.4 Publication of appointment cancellation documents and statistics

Every successful cancellation of an appointment requires the publication of a queryDocument and a vritActReference. These entries contain details about the appointment cancellation requests author, organisation, patient and the cancelled appointments IDs. As the process of the generation and publishing of these documents is exactly the same as when creating a new appointment, the details regarding this operation are specified in the chapter “Processing and publication of appointment creation documents”.

When the documents have been published, the appointment entry of the TIS database table APPDB_APPOINTMENT needs to be updated to represent its cancelled status. Therefore, the microservice updates the status of the appointment to “nullified” and sets a new update time.

All the successful as well as failed requests and responses are saved to the APPDB_APPOINTMENT_MESSAGE table. Once again, statistics saved related to the cancellation of an appointment are similar to those persisted when a new appointment is created. The main difference is that the Document Type Codes are now Appointment_tto_nullified_request or Appointment_tto_nullified_response.

4.5.5 Appointment cancellation scheduler

If an appointment is cancelled individually by the request of a user in üDR, in case of a failure the user is notified and they can attempt the cancellation again later. However, if the cancellation is part of the process of changing an appointment to one in a different provider, it is performed automatically after the new appointment is created for the user.

For situations where the cancellation of the appointment has the reason "CHG", meaning that its part of an appointment change process, an automatic appointment cancellation scheduler is implemented in ORCA, to attempt to cancel the appointments that failed initially. The scheduler has configurable parameters like the maximum number of cancellation attempts and the time period before ORCA attempts to cancel the appointment again.

4.5.6 Generation and sending of appointment cancel information to üDR

The AppointmentCancelResponse (Figure 27) returned to üDR after the processing of an appointment cancellation request has been finished contains data indicating the success or the failure of the request, reasons for the failure if it occurred, IDs and creation time of the appointment cancellation request sent to the healthcare provider.

```
public class AppointmentCancelResponse {
  private boolean success;
  private boolean rejectedByTTO;

  private String rejectionCode;
  private String rejectionReason;

  private String referenceNum;
  private String referenceNumOid;

  private String queryId;
  private String templateId;
  private String creationTime;
}
```

Figure 27. AppointmentCancelResponse class returned by ORCA to üDR

With this, both requests processed by the ORCA microservice have been almost entirely reviewed. We will now cover the final aspect of both of these requests - the pre-booking and pre-cancellation of appointments.

4.5.7 Appointment pre-booking and pre-cancellation requests

Both appointment creation and cancellation requests can respectively have a special type of pre-creation or pre-cancellation. These kinds of requests are indicated by AppointmentCreateRequest’s statusCode “HELD” or AppointmentCancelRequest’s reasonCode “PRE” and are used in two primary scenarios:

1. When the user attempts to create a new appointment, he selects an available free slot from the list of all the available appointments and is redirected to the appointment’s detailed view, where they can confirm their registration. The pre-booking request is created before they are redirected to the detailed view and it:
    a. Checks whether the free slot is still available and the appointment could be created
    b. The healthcare provider puts a lock on this specific free slot, so that it is not available to other users for a set period of time

   In the process of this request no appointment documents, entries or statistics are saved to the database, the pre-booking request is simply sent to the provider and the response is returned to üDR.
2. When changing an appointment to one in a different provider, before creating a new appointment üDR sends a pre-cancellation request for the currently active request to check whether the cancellation of the appointment is possible. The appointment is not actually cancelled, but üDR can decide based on the provider's response whether the changing of this appointment should be enabled to the user. No document or statistics entries are published in this scenario as well.

If we wanted to send our appointment cancellation request shown in the paragraph X-Road, HL7 and TIS as a pre-cancellation request, the reasonOf block would contain information displayed on Figure 28.

```
<reasonOf typeCode="RSON">
    <detectedIssueEvent classCode="ALRT" moodCode="EVN">
        <id root="1.3.6.1.4.1.28284.1.3.2.20"
            extension="PERH.49302"/>
        <code code="PRE"/>
    </detectedIssueEvent>
</reasonOf>
```

Figure 28. HL7 V3 Appointment pre-cancellation request’s reasonOf element

At this point we have covered all the four microservices CRAB, SPONGE, SPRUT and ORCA planned for the restructuring of the applications general architecture. In the following chapters we will discuss the specific solution regarding the vaccination of patients through üDR.

5. Vaccinations in üDR

When the COVID-19 pandemic started, people in Estonia were able to register for a vaccination mainly by calling their family doctor, a specific healthcare provider or through the family doctor advice line 1220. This put a lot of pressure on the phone line operators and the providers in general, as the manual registration of every patient for a vaccination used a lot of resources.

For this reason it was decided to automate the vaccination registration process through üDR, so that patients could create and manage their vaccination appointments by themselves and healthcare providers could manage and offer vaccinations as regular appointments, or available time slots.

5.1 Limitations of managing appointments in üDR

Although the vaccinations could be offered to patients as regular time slots, there are problems due to the unique nature of the COVID-19 vaccination process:

- The vaccinations have multiple types of vaccines, all having unique vaccination rules
- With most vaccine types, the vaccination process had to include two separate vaccinations, each a certain time period apart
- After finishing one vaccination cycle or being fully vaccinated, the patients should be able to register for a new vaccination cycle called a revaccination after a longer period of time
- Vaccination of patients of different age groups have to be done with different vaccines, as not all the vaccines are suitable for all the patients
- The registration for vaccines that patients are not supposed to be vaccinated with or on periods of time when they have just been recently vaccinated should be restricted
- Special immunisation rules apply if the patient has been infected with COVID-19

All these unique parameters meant that a custom solution for the vaccination process had to be built on top of the current appointment management implementation.

5.2 Tracking of patients’ vaccination status

When the patient is searching for a vaccination time slot, üDR needs to know their current vaccination status to be able to offer only the times that suit that patient. There are two limitations üDR can impose on the patient’s vaccination time slot search:

- The vaccine types the search is made for
- The time period between which the times are searched

To track the vaccination status of all the patients in Estonia, two tables were created in the TIS database: VACCINATION_DATA and PERSON_COVID19_IMMUNISATION. If a patient is vaccinated, or they show a positive test for the COVID-19 infection, a special immunisation notice is created, based on which the information in the tables is updated.

The VACCINATION_DATA table contains a list of all the vaccination events for all patients in Estonia. In the context of üDR, the primary function of this data table is to give an overview of a patient's history of vaccinations.

The PERSON_COVID19_IMMUNISATION contains only a single entry for every patient in Estonia. Patients' most recent immunisation event is always stored in the table, and this table is used by üDR to make decisions on which vaccines are suitable for the patient's next vaccination and when the next vaccination should happen. The PERSON_COVID19_IMMUNISATION table contains information shown in Table 4.

Table 4. Data stored within the PERSON_COVID19_IMMUNISATION table

Field name	**Field description**
personal_identification_code	Patient’s unique identification code
last_vaccination_date	Date of the patient’s last vaccination
last_vaccination_status	Patient’s last vaccination status (Completed or InProgress), for vaccines that require two vaccinations
last_vaccine_service_code	Patient’s last vaccination vaccine service code, identifying the type of the vaccine used
next_vaccination_date	Next earliest date when the patient should be allowed to be vaccinated again

5.3 Vaccination adaptations in üDR and the microservice NARVAL

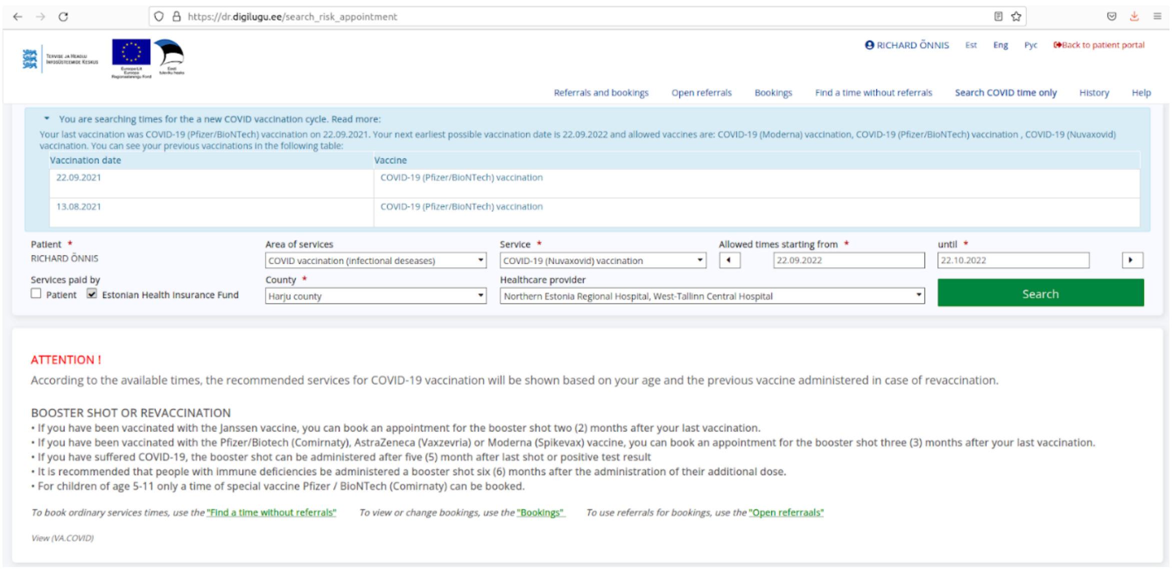
With the COVID-19 vaccination process requiring custom solutions, a new “Search COVID time only” page was created in üDR, dedicated only to COVID-19 vaccination time slot search. This page has a custom backend architecture, built on top of the base “Find a time” view, with the addition of the suitable next vaccine and vaccination period finding algorithms and the support for the patient’s vaccination history widget.

5.3.1 Vaccination history widget

Patient’s current vaccination status, as well as their vaccination history is displayed from the new vaccination history widget. The widget will give an overview of the following information:

- Patient’s vaccination status, displayed by what kind of a vaccination is the patient’s next vaccination, for example “You are searching times for a new COVID vaccination cycle”
- The vaccine they were last vaccinated with, as in “Your last vaccination was COVID-19 (Pfizer/BionTech) vaccination”
- The date when they were previously vaccinated: “vaccination on 22.09.2021”
- Patient’s next earliest possible vaccination date: “Your next earliest vaccination date is 22.09.2022”
- Vaccines that the patient is allowed to vaccinate with on their next vaccination: “allowed vaccines are Moderna, Pfizer, Nuvaxovid”
- A list of patient’s previous vaccinations, with vaccination dates and vaccine types

A “Search COVID time only” page depicting the widget described in the examples above can be seen on Figure 29.


https://dr.digilugu.ee/search_risk_appointment

RICHARD ÕNNIS Est Eng Рус Back to patient portal

Referrals and bookings | Open referrals | Bookings | Find a time without referrals | Search COVID time only | History | Help

You are searching times for the a new COVID vaccination cycle. Read more:

Your last vaccination was COVID-19 (Pfizer/BioNTech) vaccination on 22.09.2021. Your next earliest possible vaccination date is 22.09.2022 and allowed vaccines are: COVID-19 (Moderna) vaccination, COVID-19 (Pfizer/BioNTech) vaccination , COVID-19 (Nuvaxovid) vaccination. You can see your previous vaccinations in the following table:

Vaccination date	Vaccine
22.09.2021	COVID-19 (Pfizer/BioNTech) vaccination
13.08.2021	COVID-19 (Pfizer/BioNTech) vaccination

Patient * RICHARD ÕNNIS
Area of services: COVID vaccination (infectional deseases)
Service *: COVID-19 (Nuvaxovid) vaccination
Allowed times starting from *: 22.09.2022
until *: 22.10.2022
Services paid by: ☐ Patient ☑ Estonian Health Insurance Fund
County *: Harju county
Healthcare provider: Northern Estonia Regional Hospital, West-Tallinn Central Hospital

ATTENTION !

According to the available times, the recommended services for COVID-19 vaccination will be shown based on your age and the previous vaccine administered in case of revaccination.

BOOSTER SHOT OR REVACCINATION

- If you have been vaccinated with the Janssen vaccine, you can book an appointment for the booster shot two (2) months after your last vaccination.
- If you have been vaccinated with the Pfizer/Biotech (Comirnaty), AstraZeneca (Vaxzevria) or Moderna (Spikevax) vaccine, you can book an appointment for the booster shot three (3) months after your last vaccination.
- If you have suffered COVID-19, the booster shot can be administered after five (5) month after last shot or positive test result
- It is recommended that people with immune deficiencies be administered a booster shot six (6) months after the administration of their additional dose.
- For children of age 5-11 only a time of special vaccine Pfizer / BioNTech (Comirnaty) can be booked.

To book ordinary services times, use the "Find a time without referrals" *To view or change bookings, use the "Bookings"* *To use referrals for bookings, use the "Open referraals"*

View (VA.COVID)

Figure 29. Search COVID time only page in üDR portal depicting patient’s vaccinations widget

Patient’s vaccination history, as well as their latest vaccination information in üDR is requested with the DR-Vaccination NARVAL microservice.

5.3.2 Vaccinations microservice DR-Vaccinations NARVAL

NARVAL or Narrowing And Restricting Vaccinations Access List is the microservice responsible for querying data from the VACCINATION_DATA and PERSON_COVID19_IMMUNISATION data tables. Respectively, the microservice has two endpoints: /vaccinations and /vaccinations/latest. Both endpoints require a single query parameter: the patient’s personal identification code to query information about their vaccination history or data on their latest immunisation event.

The history request returns a list of VaccinationData (Figure 30) objects, while the latest request returns a single VaccinationData entry. The VaccinationData class contains all the fields relevant to the patient’s vaccination information.

```
public class VaccinationData {
  private Long id;
  private String patientCode;
  private LocalDateTime vaccinationDate;
  private String vaccinationStatus;
  private String serviceCode;
  private LocalDate nextVaccinationDate;
}
```

Figure 30. VaccinationData class of the NARVAL microservice

When the patient is redirected to the “Search COVID time only” view in üDR, both the history and the latest vaccination requests are made with the NARVAL microservice, the patients vaccination history widget is drawn and algorithms detailed in the following paragraphs are launched to determine patient’s next allowed vaccinations.

5.3.3 Specifying patient’s next vaccination

Based on the VaccinationData retrieved from the PERSON_COVID19_IMMUNISATION table, a vaccination algorithm finds a suitable vaccine for the patient’s next vaccination. Before reviewing the algorithm, there are two important configurable parameters in üDR that influence the choice of the next suitable vaccines: the age restrictions on different vaccinations and vaccines allowed for revaccinations.

At the time of writing this thesis, there were five different vaccine types available for vaccination through üDR. Each of these vaccines has configurable age restrictions, depicting the age of patients that can be vaccinated with these vaccines. The table below (Table 5) illustrates all the available vaccines and age restrictions associated with them.

Table 5. Vaccine types and their age restrictions in üDR

Vaccine type	Vaccine age restriction
Pfizer	12+
Moderna	12+
Jannsen	18+
AstraZeneca	50+
Nuvaxovid	18+
Pfizer (5-11)	5-12

As derived from the table, patients of ages between 0 and 4 are not allowed to be vaccinated.

When searching times for a revaccination, the list of available vaccine types is narrowed even further. To specify the list of vaccines allowed for revaccinations, the parameter reVaccinationVaccineCodes is implemented. At the time of writing this thesis, vaccines allowed for revaccinations were Pfizer, Moderna and Nuvaxovid.

This section descibes the decision making process of the vaccination algorithm in üDR, i.e. choosing the patient's next suitable vaccinations based on data from the PERSON_COVID19_IMMUNISATION table. The vaccination time search limitations are applied as follows:

- If the patient's VaccinationData is empty, it means that no latest vaccination entries were found in the database and that the patient has never been previously vaccinated. Patient's vaccination time search is not limited any further, aside from the limitations imposed by the age restrictions. Their earliest allowed vaccination date is the current date.
- If the patient's VaccinationData's vaccinationStatus is "Completed", the patient is considered as fully vaccinated. Patients earliest next vaccination date is taken from the VaccinationData's nextVaccinationDate and the patient's next vaccines are the ones specified in the reVaccinationVaccineCodes list.
- If the patient's VaccinationData's vaccinationStatus is "InProgress", the patient is considered as partially vaccinated. Patients earliest next vaccination date is taken from the VaccinationData's nextVaccinationDate and the patient's next vaccine is his previous vaccine specified in the VaccinationData's serviceCode.
- If the patient has a VaccinationData entry, but its vaccinationStatus is empty, the patient is considered as unvaccinated and infected by the COVID-19 virus. Patients earliest next vaccination date is taken from the VaccinationData's nextVaccinationDate and the patient's next vaccines are the ones specified in the reVaccinationVaccineCodes list.

Note that if a patient is infected with the virus while his current vaccinationStatus is "Completed" or "InProgress", no new entry is added to the PERSON_COVID19_IMMUNISATION table. Instead, the patient's next earliest vaccination date is sent back by a configurable period of time. So, for example, if the patient's status is "InProgress" and their next vaccination date is on 14.06.2022, today is 13.06.2022, and the patients infection with the COVID-19 virus is registered, he is not allowed to vaccinate for two additional weeks since the infection, meaning that their next vaccination date will be 28.06.2022. Their status and next vaccine remain unchanged.

This algorithm applies to most of the vaccination services in üDR, however, some exceptions were applied with the introduction of a new vaccine Nuvaxovid. Based on the immunoprophylaxis expert commissions decision, it is not recommended to revaccinate with the Nuvaxovid vaccine, if both previous vaccinations were done with the Nuvaxovid vaccine. For this reason, if the patients VaccinationData’s vaccinationStatus is “Completed” and the serviceCode is Nuvaxovid’s service code, the patients vaccination history is checked to find out whether both the previous vaccinations were Nuvaxovid vaccinations. If it is the case, the patient's next vaccines are the ones specified in the reVaccinationVaccineCodes list, except the Nuvaxovid vaccination.

5.3.4 Vaccinations with referrals

The aforementioned algorithm is applied only when the patients are looking for possible vaccination times themselves. If the vaccination time is searched through the patient’s “Open referrals” view, meaning the practitioner has specified a referral for the patient's vaccination, all the filters regarding the patient's next allowed vaccination are taken from the referral.

A specialist's opinion is always considered superior to the work of the base algorithm, therefore if a patient has specific needs due to their unique health condition, he can always consult a doctor who can in turn write them a referral for any of the previously described services at any given time.

6. Validation and results

In this chapter we will measure the impact the architectural restructuring had on the performance of the application in general and discuss the statistics on the vaccination appointments created during the crisis with the help of the vaccination module implemented throughout this thesis.

6.1 Performance of the microservice architecture

To measure the impact microservices had on the performance of the application, several test scenarios were created to compare the available time slots request processing speed before and after the restructuring. A total of six different scenarios were chosen for the testing purposes and all the scenarios were run with and without the new implementation for 50 times. The available time slots request processing speed was measured for every request.

For the improved accuracy of the results, the tests were carried out in the development environment, where the load on the healthcare institutions and their request processing times are minimal. Request processing times before the architectural restructuring are represented under the label TIS and after under the label SPRUT.

To calculate the difference between both architectures' request processing speeds, a mean value was calculated for all the 50 requests in every scenario, and the mean values of both TIS and SPRUT architectures were compared. Equation (1) displays the formula for calculating the mean of time measurements.

$$Mean = \frac{\sum_{i=1}^{N} X_i}{N} \tag{1}$$

Here X is the set of measurements and N is the number of these measurements. A percentile difference of the request processing speed between the architectures for a single scenario was calculated with the formula displayed in Equation (2).

$$Percentage\ difference = (1 - \frac{MSPRUT}{MTIS}) \times 100\% \tag{2}$$

Here $MSPRUT$ is the mean value for the SPRUT architecture measurements and $MTIS$ is the mean value for the TIS architecture measurements. To measure the differences in stability of request processing times, the coefficients of variations or relative standard deviations (RSD) of request processing speeds sample are compared between the two different architectures for every scenario. To calculate the relative standard deviation, the sample standard deviation is divided with the mean value of the sample. You can see the formula for these calculations below on Equation (3).

$$RSD\ difference = \frac{\sqrt{\frac{\sum_{i=1}^{N}(X_i - MTIS)^2}{N-1}}}{MTIS} - \frac{\sqrt{\frac{\sum_{i=1}^{N}(Y_i - MSPRUT)^2}{N-1}}}{MSPRUT} \tag{3}$$

Here X is the set of measurements for the TIS architecture and Y the set of measurements for the SPRUT achitecture. A positive RSD difference value shows that SPRUT had a more stable performance on a specific scenario than TIS, while a negative value shows the opposite.

6.1.1 Scenario 1: Ophthalmologist in all providers

In the first scenario, the parameters for the available time slots search were as follows:

- Service: Visit to an ophthalmologist
- Counties: All counties
- Healthcare providers: Linnamõisa Perearstikeskus OÜ, HEDA Dev, Pärnu haigla, Põhja-Eesti Regionaalhaigla, Põlva haigla, Reveron Baltic OÜ, Sihtasutus Narva haigla and Tartu Ülikooli Kliinikum SA (A total of eight providers)
- Payer type: Insurance fund or patient
- Search dates: 27.04.2022 - 26.07.2022

The testing results for Scenario 1 are shown on Figure 31.

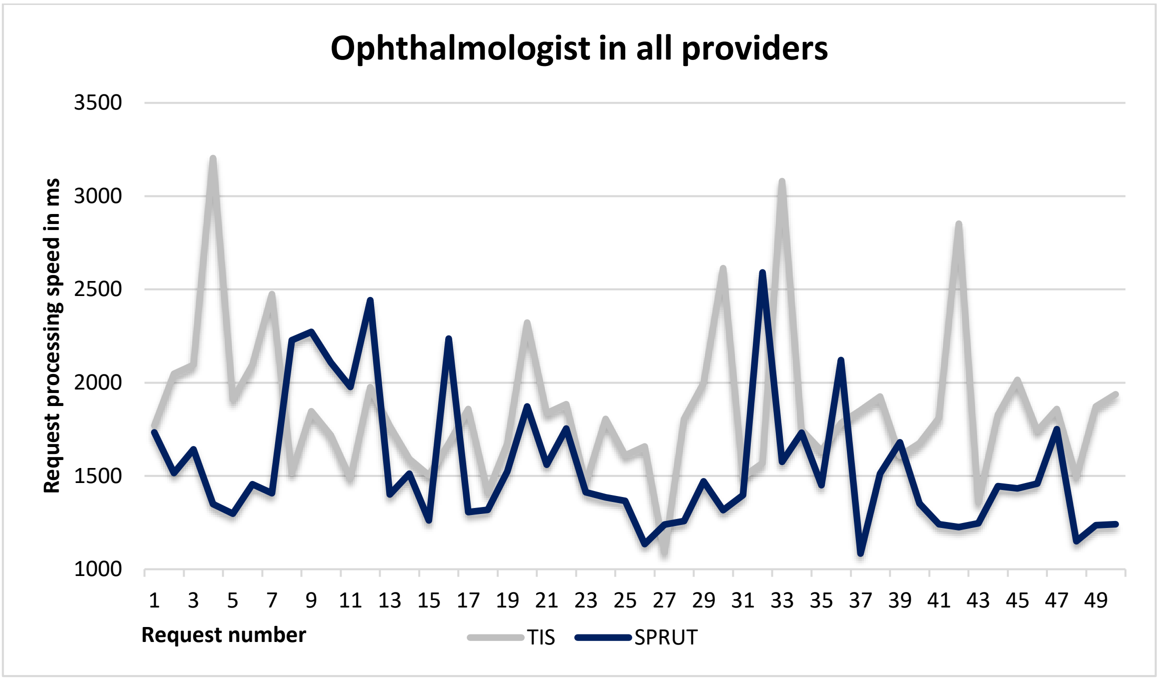


Figure 31. SPRUT and TIS performance for Ophthalmologist in all providers

The requests were made to eight different providers and 60 available time slots from two healthcare providers (Põhja-Eesti Regionaalhaigla and Pärnu haigla) and two doctors were found on every request. The mean request processing time for TIS was 1856.84 milliseconds, while the mean time for SPRUT was 1553.75 milliseconds. Therefore, on average, for Scenario 1 the new microservice architecture was able to process the available time slots requests 16.32% faster than the previous solution.

The sample standard deviation for the TIS architecture in Scenario 1 was 405.863. When divided with the mean, the RSD value of 0.219 or 21.9% was found. The SPRUT architecture had a sample standard deviation of 356.993 and the RSD of 0.23 or 23%. The calculation of the difference between relative standard devitations of TIS compared to SPRUT yielded the negative result of 0.219 - 0.23 = -0.011 or -1,1% RSD difference, meaning that the SPRUT requests had slightly less stable relative request processing speeds for Scenario 1.

6.1.2 Scenario 2: Ophthalmologist in Pärnu Haigla

In the second scenario, the parameters for the available time slots search were as follows:

- Service: Visit to an ophthalmologist
- Counties: Pärnu county
- Healthcare providers: Pärnu haigla

- Payer type: Insurance fund or patient
- Search dates: 27.04.2022 - 26.07.2022

The testing results for Scenario 2 are shown on Figure 32.

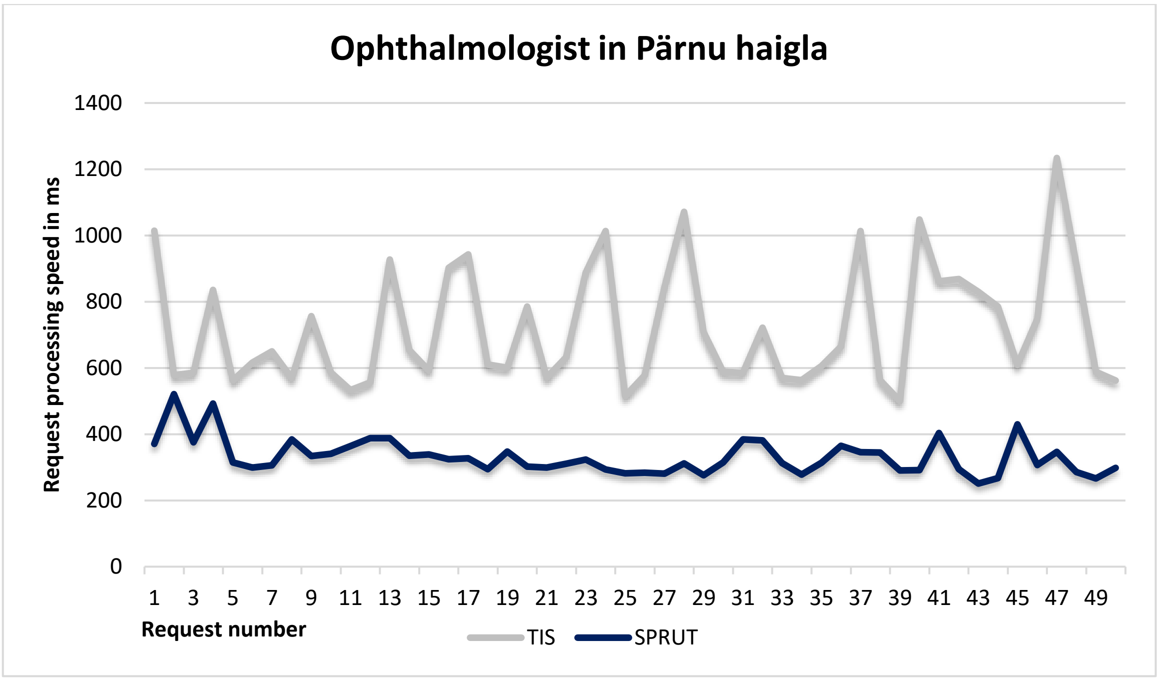


Figure 32. SPRUT and TIS performance for Ophthalmologist in Pärnu haigla

The requests were made to a single provider Pärnu haigla and 20 available time slots for one doctor were found on every request. The average request processing time for TIS was 721.96 milliseconds, while the average time for SPRUT was 331.68 milliseconds. Therefore, on average, for Scenario 2 the new microservice architecture was able to process the available time slots requests 54.06% faster than the previous solution.

The sample standard deviation for the TIS architecture in Scenario 2 was 179.815. When divided with the mean, the RSD value of 0.249 or 24.9% was found. The SPRUT architecture had a sample standard deviation of 54.075 and the RSD of 0.163 or 16.3%. The calculation of the difference between relative standard devitations of TIS compared to SPRUT yielded the positive result of 0.249 - 0.163 = 0.086 or 8,6% RSD difference, meaning that the SPRUT requests had slightly more stable relative request processing speeds for Scenario 2.

6.1.3 Scenario 3: Cardiologist in all providers

In the third scenario, the parameters for the available time slots search were as follows:

- Service: Visit to a cardiologist
- Counties: All counties
- Healthcare providers: HEDA Dev, Pärnu haigla, Põhja-Eesti Regionaalhaigla, Põlva haigla, Medicum, Sihtasutus Narva haigla and Tartu Ülikooli Kliinikum SA (A total of seven providers)
- Payer type: Patient
- Search dates: 27.04.2022 - 26.07.2022

The testing results for Scenario 3 are shown on Figure 33.

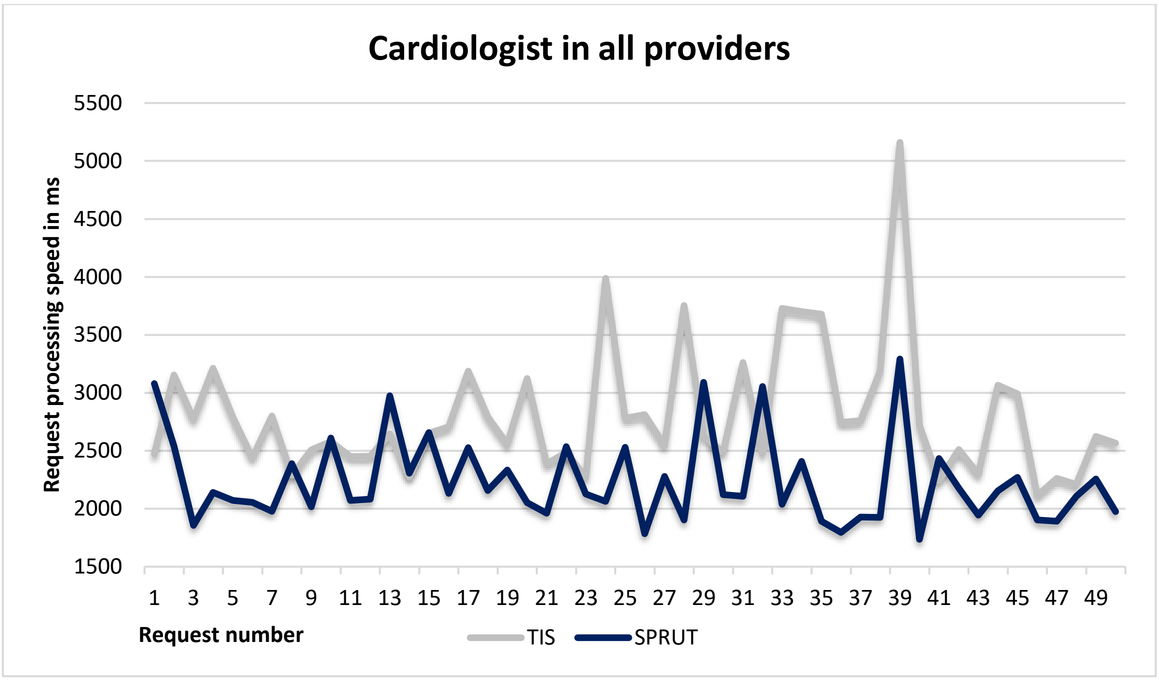


Figure 33. SPRUT and TIS performance for Cardiologist in all providers

The requests were made to seven different providers and 80 available time slots from two healthcare providers (Medicum and Tartu Ülikooli Kliinikum SA) and two doctors were found on every request. The average request processing time for TIS was 2806.08 milliseconds, while the average time for SPRUT was 2234.64 milliseconds. Therefore, on average, for Scenario 3 the new microservice architecture was able to process the available time slots requests 20.36% faster than the previous solution.

The sample standard deviation for the TIS architecture in Scenario 3 was 559.306. When divided with the mean, the RSD value of 0.199 or 19.9% was found. The SPRUT architecture had a sample standard deviation of 368.202 and the RSD of 0.165 or 16.5%. The calculation of the difference between relative standard devitations of TIS compared to SPRUT yielded the positive result of 0.199 - 0.165 = 0.034 or 3,4% RSD difference, meaning that the SPRUT requests had slightly more stable relative request processing speeds for Scenario 3.

6.1.4 Scenario 4: Cardiologist in Medicum

In the fourth scenario, the parameters for the available time slots search were as follows:

- Service: Visit to a cardiologist
- Counties: Harju county
- Healthcare providers: Medicum
- Payer type: Patient
- Search dates: 27.04.2022 - 26.07.2022

The testing results for Scenario 4 are shown on Figure 34.

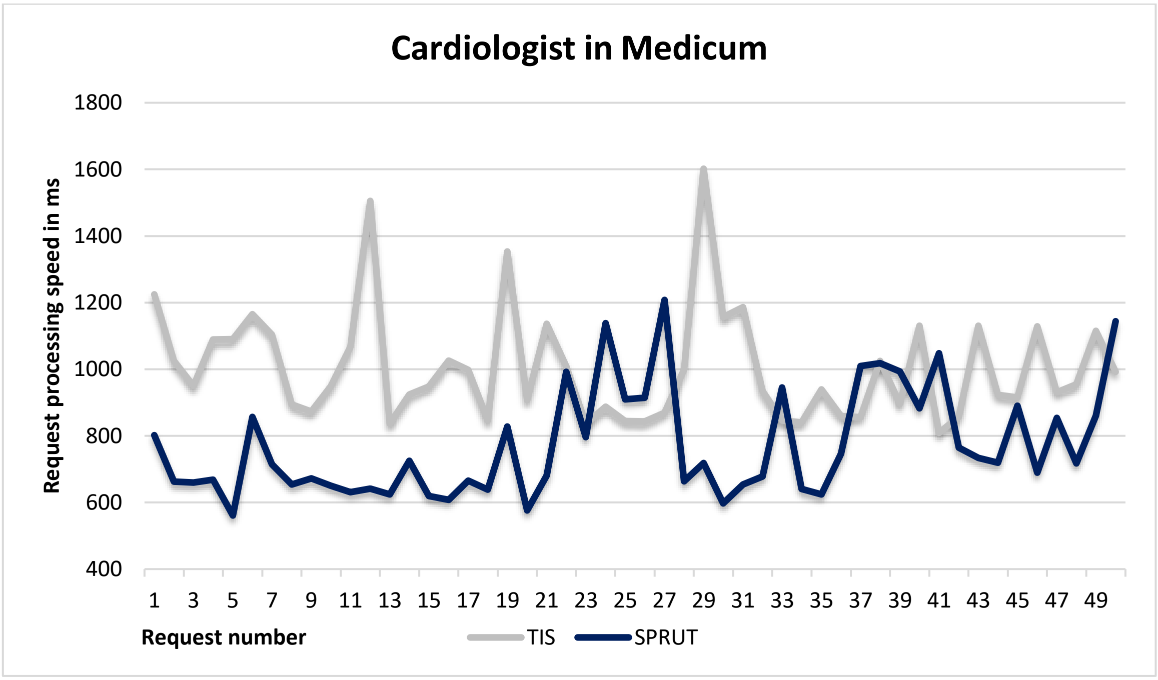


Figure 34. SPRUT and TIS performance for Cardiologist in Medicum

The requests were made to a single provider Medicum and 40 available time slots for one doctor were found on every request. The average request processing time for TIS was 1004.64 milliseconds, while the average time for SPRUT was 773.84 milliseconds. Therefore, on average, for Scenario 4 the new microservice architecture was able to process the available time slots requests 22.97% faster than the previous solution.

The sample standard deviation for the TIS architecture in Scenario 4 was 168.131. When divided with the mean, the RSD value of 0.167 or 16.7% was found. The SPRUT architecture had a sample standard deviation of 163.202 and the RSD of 0.211 or 21.1%. The calculation of the difference between relative standard devitations of TIS compared to SPRUT yielded the negative result of 0.167 - 0.211 = -0.044 or -4,4% RSD difference, meaning that the SPRUT requests had slightly less stable relative request processing speeds for Scenario 4.

6.1.5 Scenario 5: Dermatologist in all providers

In the fifth scenario, the parameters for the available time slots search were as follows:

- Service: Visit to a dermatologist
- Counties: All counties
- Healthcare providers: HEDA Dev, Pärnu haigla, Põhja-Eesti Regionaalhaigla, Medicum and Tartu Ülikooli Kliinikum SA (A total of five providers)
- Payer type: Insurance fund
- Search dates: 27.04.2022 - 26.07.2022

The testing results for Scenario 5 are shown on Figure 35.

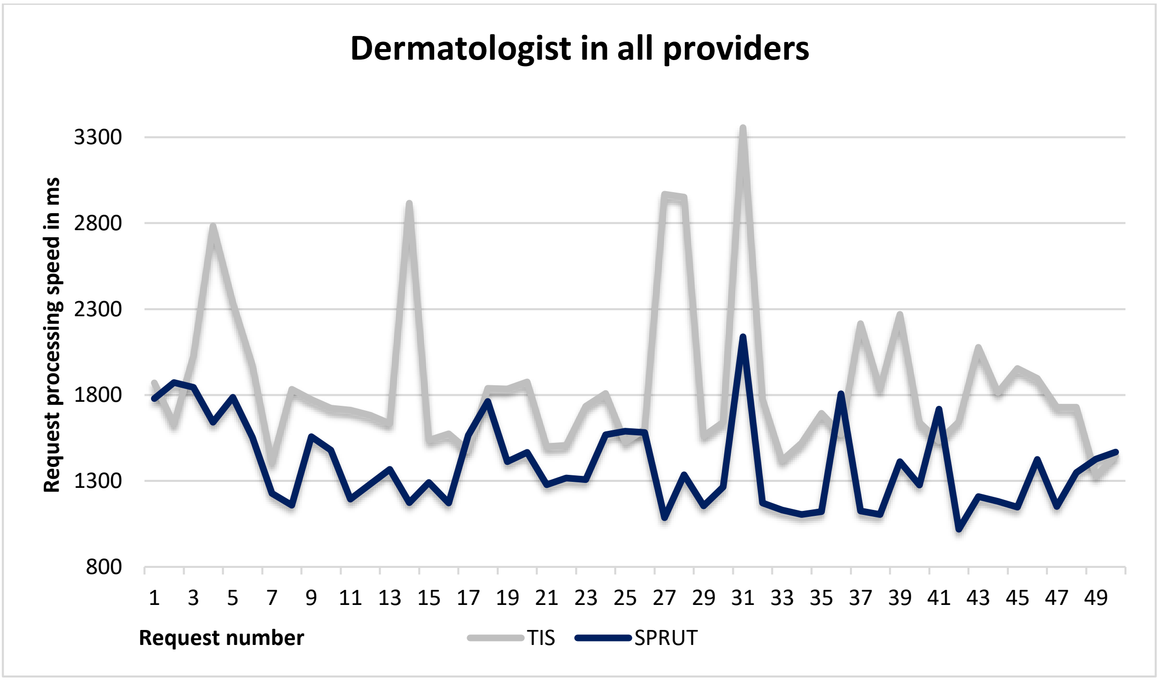


Figure 35. SPRUT and TIS performance for Dermatologist in all providers

The requests were made to five different and 80 available time slots for two healthcare providers (Põhja-Eesti Regionaalhaigla and Tartu Ülikooli Kliinikum SA) and two doctors were found on every request. The average request processing time for TIS was 1853.94 milliseconds, while the average time for SPRUT was 1391.24 milliseconds. Therefore, on average, for Scenario 5 the new microservice architecture was able to process the available time slots requests 24.96% faster than the previous solution.

The sample standard deviation for the TIS architecture in Scenario 5 was 444.358. When divided with the mean, the RSD value of 0.24 or 24% was found. The SPRUT architecture had a sample standard deviation of 255.497 and the RSD of 0.184 or 18.4%. The calculation of the difference between relative standard devitations of TIS compared to SPRUT yielded the positive result of 0.24 - 0.184 = 0.056 or 5,6% RSD difference, meaning that the SPRUT requests had slightly more stable relative request processing speeds for Scenario 5.

6.1.6 Scenario 6: Dermatologist in Tartu Ülikooli Kliinikum SA

In the sixth scenario, the parameters for the available time slots search were as follows:

- Service: Visit to a dermatologist
- Counties: Tartu county
- Healthcare providers: Tartu Ülikooli Kliinikum SA
- Payer type: Insurance fund
- Search dates: 27.04.2022 - 26.07.2022f

The testing results for Scenario 6 are shown on Figure 36.

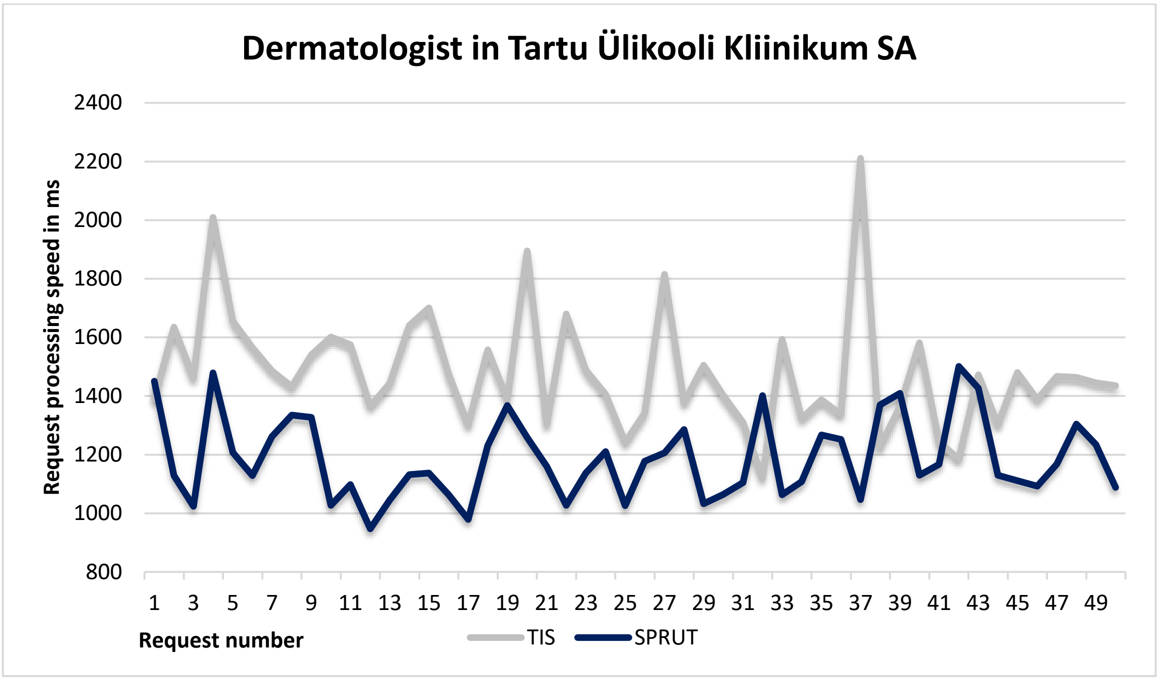


Figure 36. SPRUT and TIS performance for Dermatologist in Tartu Ülikooli Kliinikum SA

The requests were made to a single provider Tartu Ülikooli Kliinikum SA and 40 available time slots for one doctor were found on every request. The average request processing time for TIS was 1480.3 milliseconds, while the average time for SPRUT was 1186.58 milliseconds. Therefore, on average, for Scenario 6 the new microservice architecture was able to process the available time slots requests 19.84% faster than the previous solution.

The sample standard deviation for the TIS architecture in Scenario 6 was 202.342. When divided with the mean, the RSD value of 0.137 or 13.7% was found. The SPRUT architecture had a sample standard deviation of 139.022 and the RSD of 0.117 or 11.7%. The calculation of the difference between relative standard devitations of TIS compared to SPRUT yielded the positive result of 0.137 - 0.117 = 0.02 or 2% RSD difference, meaning that the SPRUT requests had slightly more stable relative request processing speeds for Scenario 6.

6.1.7 Analysis of the results

Altogether, a total of 600 available time slots request were made and results registered during the testing process. For every scenario we saw a significant performance improvement when using the new SPRUT microservice architecture (Table 6).

Table 6. Available time slots request processing speed and stability changes when using microservices

Name of the scenario	Decrease in request processing speed with SPRUT	Difference in RSD between TIS and SPRUT
Scenario 1: Ophthalmologist in all providers	16.32%	-1.1%
Scenario 2: Ophthalmologist in Pärnu Haigla	54.06%	8.6%
Scenario 3: Cardiologist in all providers	20.36%	3.4%
Scenario 4: Cardiologist in Medicum	22.97%	-4.4%
Scenario 5: Dermatologist in all providers	24.96%	5.6%
Scenario 6: Dermatologist in Tartu Ülikooli Kliinikum SA	19.84%	2%
Average	26.42%	2.35%

When we combine the results from all the test scenarios by calculating the mean, we get that the SPRUT microservice processed available time slot requests 26.42% faster than the TIS architecture on average. The stability of request processing speeds was approximately the same for SPRUT and TIS, with SPRUT having a 2.35% smaller average relative standard deviation.

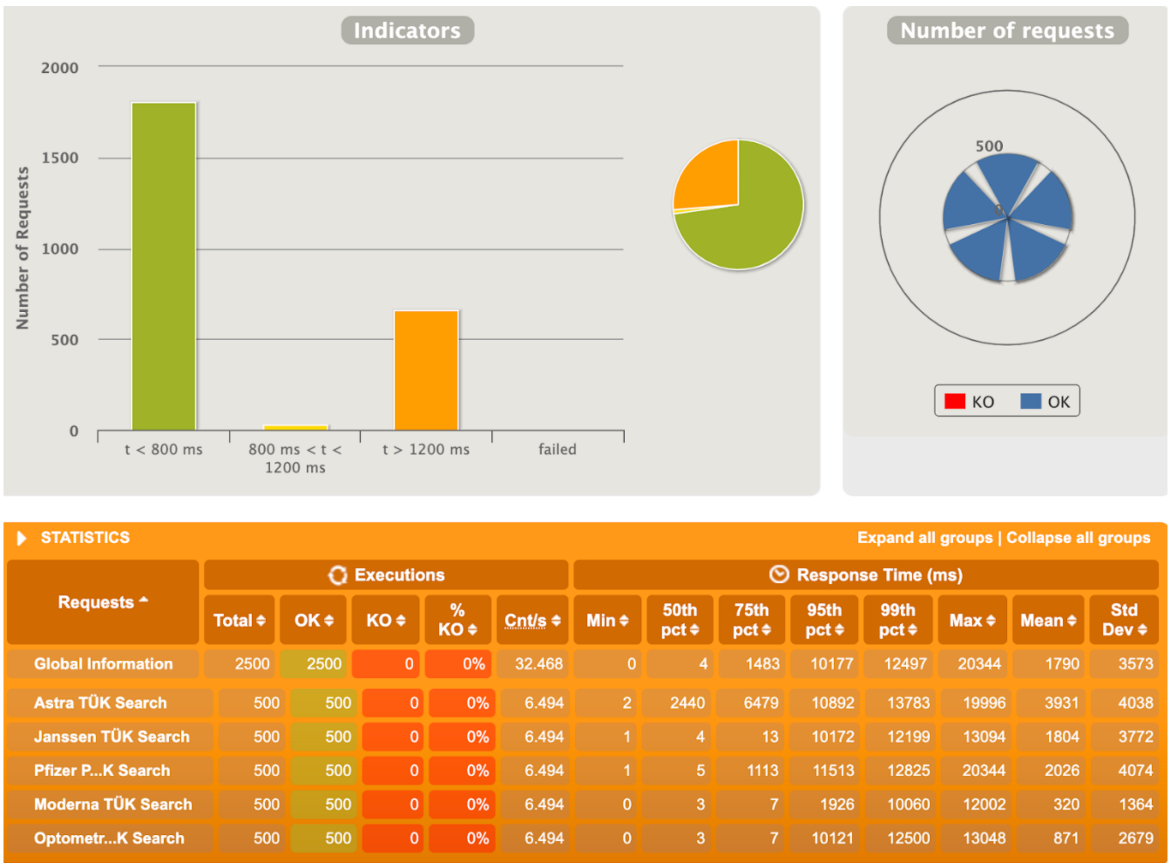
6.2 Scalability of the microservice architecture

To measure the scalability of the SPRUT application and its reaction to higher load, several load tests were conducted on a MacBook Pro 16Gb RAM, 2.3GHz 8-core i9 local machine through the Gatling framework [34]. All the load tests shared the same testing scenario where requests were made to a single provider Tartu Ülikooli Kliinikum SA for five services:

- AstraZeneca vaccination
- Jannsen vaccination
- Pfizer vaccination
- Moderna vaccination
- Visit to an ophthalmologist

Three separate load tests were conducted, with sequentially increased load requirements. The first test had 500 users making 2500 available time slot requests within one minute, the second test was twice as intense compared to the first test, having 1000 users making

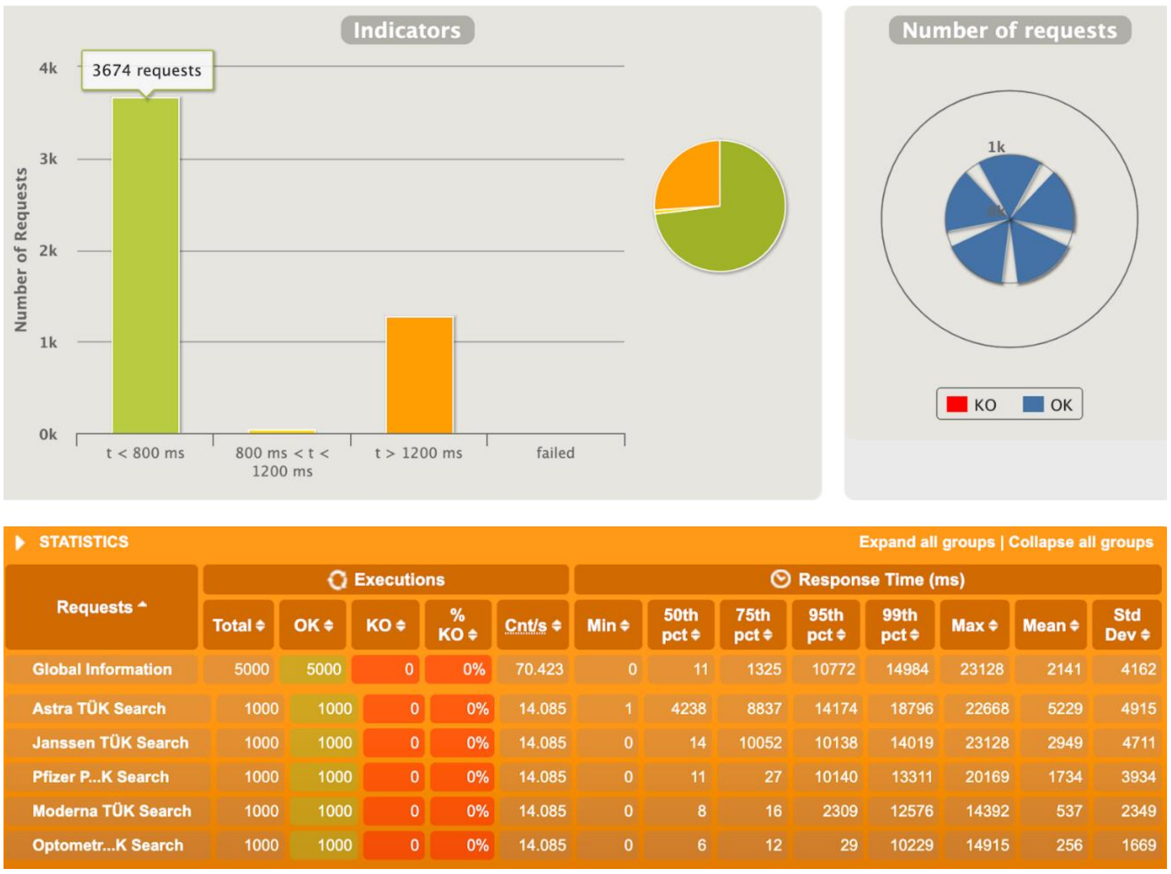
5000 requests within one minute and the final test had 3000 users making 15000 requests within the same period of time. Below, in FiguresFigure 37,Figure 38 andFigure 39 you can see the results of the load testing.



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	2500	2500	0	0%	32.468	0	4	1483	10177	12497	20344	1790	3573
Astra TÜK Search	500	500	0	0%	6.494	2	2440	6479	10892	13783	19996	3931	4038
Janssen TÜK Search	500	500	0	0%	6.494	1	4	13	10172	12199	13094	1804	3772
Pfizer P...K Search	500	500	0	0%	6.494	1	5	1113	11513	12825	20344	2026	4074
Moderna TÜK Search	500	500	0	0%	6.494	0	3	7	1926	10060	12002	320	1364
Optometr...K Search	500	500	0	0%	6.494	0	3	7	10121	12500	13048	871	2679

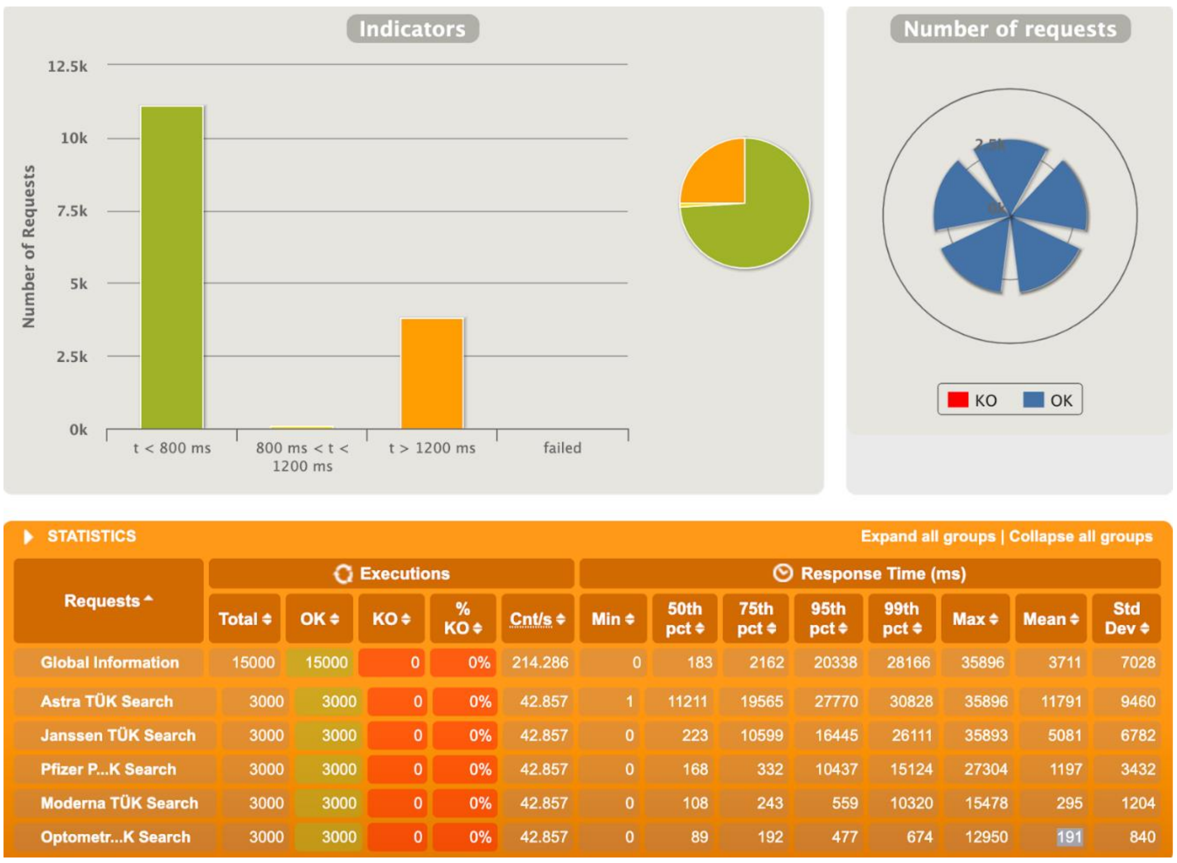
Figure 37. SPRUT microservice load testing results with 500 users and 2500 requests



STATISTICS Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	5000	5000	0	0%	70.423	0	11	1325	10772	14984	23128	2141	4162
Astra TÜK Search	1000	1000	0	0%	14.085	1	4238	8837	14174	18796	22668	5229	4915
Janssen TÜK Search	1000	1000	0	0%	14.085	0	14	10052	10138	14019	23128	2949	4711
Pfizer P...K Search	1000	1000	0	0%	14.085	0	11	27	10140	13311	20169	1734	3934
Moderna TÜK Search	1000	1000	0	0%	14.085	0	8	16	2309	12576	14392	537	2349
Optometr...K Search	1000	1000	0	0%	14.085	0	6	12	29	10229	14915	256	1669

Figure 38. SPRUT microservice load testing results with 1000 users and 5000 requests



STATISTICS Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	15000	15000	0	0%	214.286	0	183	2162	20338	28166	35896	3711	7028
Astra TÜK Search	3000	3000	0	0%	42.857	1	11211	19565	27770	30828	35896	11791	9460
Janssen TÜK Search	3000	3000	0	0%	42.857	0	223	10599	16445	26111	35893	5081	6782
Pfizer P...K Search	3000	3000	0	0%	42.857	0	168	332	10437	15124	27304	1197	3432
Moderna TÜK Search	3000	3000	0	0%	42.857	0	108	243	559	10320	15478	295	1204
Optometr...K Search	3000	3000	0	0%	42.857	0	89	192	477	674	12950	191	840

Figure 39. SPRUT microservice load testing results with 3000 users and 15000 requests

The load testing showed very good results, as the microservice had 0 failures throughout the total of 22500 requests and was capable of reacting quickly to the requests during high demand. From the graphs displayed above we can see that the average request processing times remained similar for the first load test with lower demand and the last test with the highest demand.

6.3 Vaccination statistics

In this chapter we will review the data on how many vaccination appointments were created in Estonia with the help of functionality implemented throughout this thesis. All the data discussed in this chapter was retrieved from the APPDB_APPOINTMENT table in the TIS database on May 4th, 2022.

A total of 724 756 vaccination appointments have been created, with the first vaccination appointment registered on April 30th, 2021. Out of these appointments, 131 974 have been changed or cancelled, therefore a total number of 592 782 vaccination appointments were assigned and carried through.

The appointments were created for 398 449 unique patients in 93 unique healthcare institutions. A vast majority, or 97.05% of vaccination appointments were marked with the payerType ORG, meaning that for 703 992 registered appointments the payment was covered by the health insurance fund.

Overall, as of May 6th, 2022, 860 750 patients in Estonia have received at least one vaccination dose and 845 042 have finished their first vaccination cycle, or have been fully vaccinated. This means that a total of 64.7% of Estonian citizens (including children) have received at least one dose of the COVID-19 vaccine [35].

7. Discussion and future improvements

We have now covered both the architectural changes of üDR, as well as the custom solution for the vaccination of patients in Estonia. All the changes implemented throughout this thesis will serve as a strong base for any upcoming pandemics or situations where similar stress on the üDR's appointment management system can be expected. In the next chapter, some of the possible future improvements that could be made to even further improve the work of üDR and Estonia's e-health in general are outlined.

7.1 Microservices for additional requests

One of the main steps for moving forward would be to remove all the other requests made through üDR and processed in TIS. In this thesis we have covered only the main and most important requests: the available time slot request, the appointment cancellation request and appointment creation request. The other requests that could be restructured into microservices are as follows:

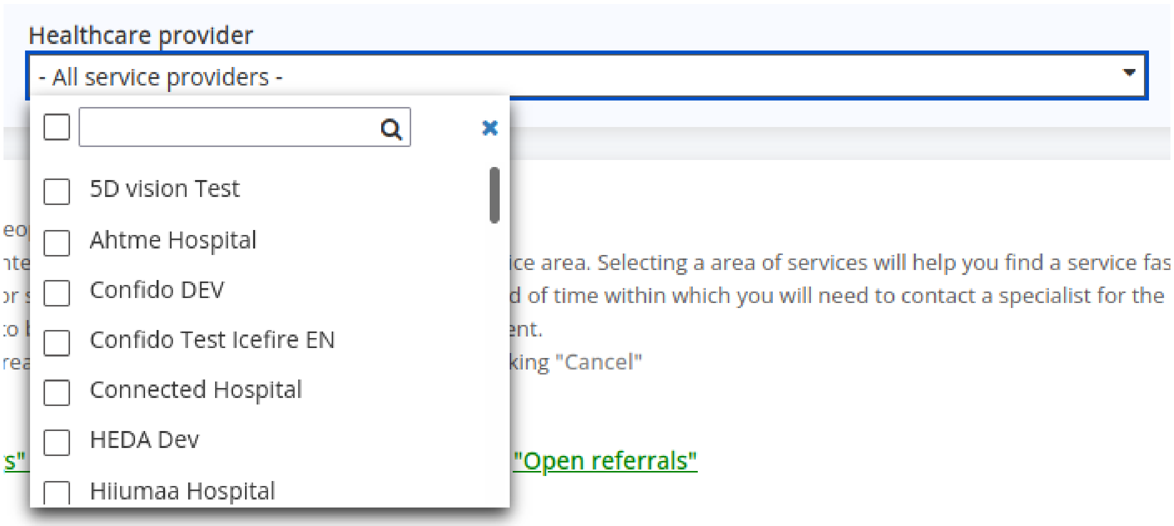
- **Appointment list request** - the request of the patient's currently active appointments
- **History request** - the request of the patient's appointment history
- **Referrals request** - the request of the patient's available referrals
- **Payment confirmation request** - the request notifying the healthcare provider if the payment for an appointment was done

A separate microservice or microservices could be created for processing of all the aforementioned requests or they could be integrated into the ORCA microservice developed in this thesis, as they are all connected to appointments in some way. Regardless, the base for the requests processing developed throughout this paper like DR-Schemas and DR-Admin would facilitate the creation of new functionality.

7.2 Frontend redesign in üDR

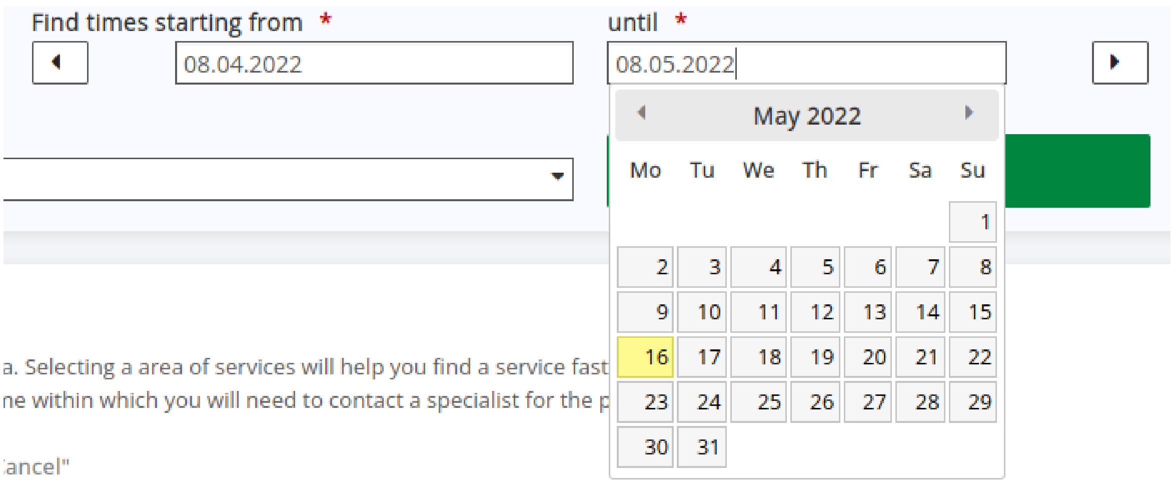
The general restructuring and redesign of the whole üDR's web application would be beneficial for both the visual and performance perspectives. Currently üDR's design feels old-fashioned and at times unintuitive. From the process of specifying available time slot search filters to the selection and confirmation of a suitable appointment, the visuals of the solution could be modernised.

In the figures below you can see some of the downsides of the current implementation, for example how the filters, texts and dropdown menus are simplistic and unproportionate. For instance, the provider selection dropdown button (Figure 40) is unreasonably wide, the arrows on the date picker menu (Figure 41) that push the date further by a month are unintuitive and the found appointment slots selection's (Figure 42) provider, location and doctor columns take up most of the space on the screen.



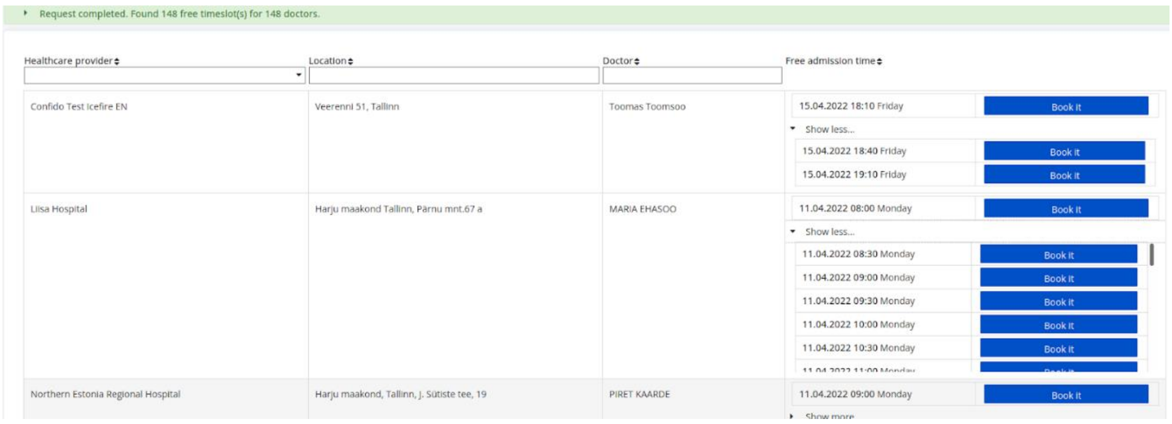
ice area. Selecting a area of services will help you find a service fas
d of time within which you will need to contact a specialist for the
ent.
king "Cancel"

Figure 40. Provider selection dropdown menu in the search appointment view in üDR



a. Selecting a area of services will help you find a service fast
ne within which you will need to contact a specialist for the p
ancel"

Figure 41. Date picker menu in the search appointment view in üDR



Request completed. Found 148 free timeslot(s) for 148 doctors.

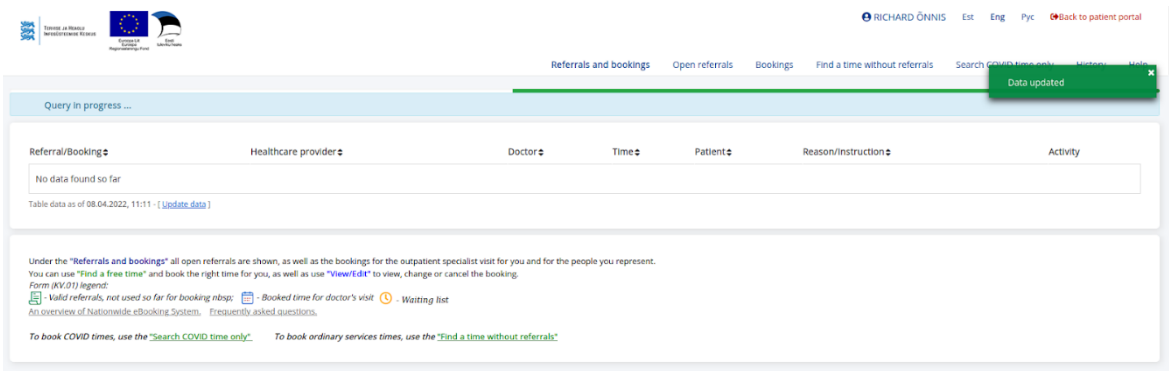
Healthcare provider	Location	Doctor	Free admission time	
Confido Test Icefire EN	Veerenni 51, Tallinn	Toomas Toomsoo	15.04.2022 18:10 Friday	Book it
			Show less...	
			15.04.2022 18:40 Friday	Book it
			15.04.2022 19:10 Friday	Book it
Liisa Hospital	Harju maakond Tallinn, Pärnu mnt.67 a	MARIA EHASOO	11.04.2022 08:00 Monday	Book it
			Show less...	
			11.04.2022 08:30 Monday	Book it
			11.04.2022 09:00 Monday	Book it
			11.04.2022 09:30 Monday	Book it
			11.04.2022 10:00 Monday	Book it
			11.04.2022 10:30 Monday	Book it
Northern Estonia Regional Hospital	Harju maakond, Tallinn, J. Sütiste tee, 19	PIRET KAARDE	11.04.2022 09:00 Monday	Book it
			Show more	

Figure 42. Found appointment slots selection menu in the search appointment view in üDR

Additionally, on the Search COVID time only page displayed on Figure 29, the highlight of the current tab in the upper menu that specifies on which view the user is currently at is barely visible.

While all these imperfections are definitely very minor, they can all add up to an unpleasant experience. More importantly, there are multiple performance related issues with the current structure of üDR-s frontend, which result in infinite queries, resetting of filters and other buggy behaviour.

In the following figure (Figure 43) we can see as if the application notifies us that the data has been updated However, the query is still in progress, and in this particular situation it is in progress indefinitely or until the page is refreshed.



Query in progress ...

Referral/Booking	Healthcare provider	Doctor	Time	Patient	Reason/Instruction	Activity
No data found so far						

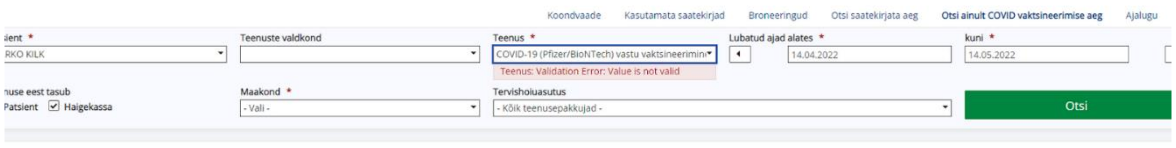
Table data as of 08.04.2022, 11:11 - [Update data]

Under the "Referrals and bookings" all open referrals are shown, as well as the bookings for the outpatient specialist visit for you and for the people you represent.
You can use "Find a free time" and book the right time for you, as well as use "View/Edit" to view, change or cancel the booking.
Form (KV.01) legend:
- Valid referrals, not used so far for booking nbsp; *- Booked time for doctor's visit* *- Waiting list*
An overview of Nationwide eBooking System. Frequently asked questions.

To book COVID times, use the "Search COVID time only" *To book ordinary services times, use the "Find a time without referrals"*

Figure 43. Infinite query on the Referrals and bookings view in üDR

As displayed in Figure 44, after selecting the service from the dropdown menu the application can sometimes show a value is not valid error. On some occasions this issue occurs on all filters simultaneously, being resolved with a refresh of the page.



ient * RKO KILK
Teenuste valdkond
Teenus * COVID-19 (Pfizer/BioNTech) vastu vaktsineerimin
Teenus: Validation Error: Value is not valid
Lubatud ajad alates * 14.04.2022
kuni * 14.05.2022
nuse eest tasub: Patsient [✓ Haigekassa]
Maakond * - Vali -
Tervishoiuasutus - Kõik teenusepakkujad -

Figure 44. Value is not valid in the service selection of the search appointment view in üDR

By rewriting the current implementation of üDR’s web application by using a modern framework like, for example, Angular, we could resolve the optimisation and performance issues as well as give the solution a modern, fresh and user-friendly outlook.

7.3 FHIR standard

One of the bigger steps for the modernisation of Estonia’s health-related information exchange would be the transition from the current HL7 V3 message-oriented standards to the new HL7 approved FHIR or Fast Healthcare Interoperability Resources [36] standards that use a RESTful (Representational state transfer) [37] architectual paradigm. FHIR reduces the implementation complexity of the HL7 V3 standard,while preserving information integrity, and moreover, it combines the advantages of currently used and widely accepted standards and overcomes their limitations [38]. The use of FHIR may improve the performance as well as the maintainability of the microservices significantly, as the generation of JSON-based documents is much more simple and efficient that the generation of XML documents.

FHIR standards are planned to be taken into use for the communication between all healthcare providers across Estonia by Autumn of 2023. This means that the generation of current XML documents used for the communication between üDR and healthcare providers will have to be rewritten into the generation of JSON-based FHIR documents.

8. Conclusion

The COVID-19 pandemic exposed many shortcomings in the architectual implementation of Estonia’s digital appointment system and created demand for a solution where patients and healthcare workers could create and manage appointments for vaccinations. The main goals of this thesis were to resolve the identified problems by restructuring the digital appointment systems monolithic design to a mircoservice-based architecture, to measure the impact of the architectual restructuring and to implement the creation and management of vaccination appointments through the portal.

As a result, four microservices were created to replace the current implementations of searching available appointment slots and the creation, cancellation and changing of patients’ appointments. Interdependence between the central health information system and the digital appointment system was reduced, performance and scalability of the application was increased, the amount of request and response documents transferred during the communication with healthcare providers was halved, the effectiveness and reliability of the generation and processing of the request and response documents was improved and the maintainability of the solution was enchanced.

A custom adaptive vaccination appointment creation and management page was implemented where patients could view their vaccination status and history, create new appointments for vaccinations with suitable vaccines based on their age and previous vaccinations and change or cancel their vaccination appointments.

The architectual restructuring reduced the average available appointment time slots request processing time by 26.42% and showed very good performance when running during higher demand. Approximately 725 000 new vaccination appointments were created by patients and healthcare workers in Estonia with the help of the vaccination appointment creation system realised throughout this thesis.

Similar architectual restructuring of more functionality within the digital appointment system, the front-end redesign and the use of Fast Healthcare Interoperability Resources standard for document exchange were discussed as some of the potential future improvements within the application.

9. References

[1] D. Cucinotta and M. Vanelli, ‘WHO Declares COVID-19 a Pandemic.’, *Acta Biomed*, vol. 91, no. 1, pp. 157–160, Mar. 2020, doi: 10.23750/abm.v91i1.9397.

[2] ERR, ‘Terviseamet: Eestis on kinnitatud 27 koroonajuhtu ja kohalik levik’, 2020. https://www.err.ee/1063204/terviseamet-eestis-on-kinnitatud-27-koroonajuhtu-ja-kohalik-levik (accessed May 05, 2022).

[3] ERR, ‘Valitsus kuulutas välja eriolukorra’, 2020. https://www.err.ee/1063213/valitsus-kuulutas-valja-eriolukorra (accessed May 05, 2022).

[4] World Health Organization, ‘Coronavirus disease (COVID-19)’. https://www.who.int/health-topics/coronavirus (accessed May 01, 2022).

[5] Y. W. Lin, C. H. Lin, and M. H. Lin, ‘Vaccination distribution by community pharmacists under the COVID-19 vaccine appointment system in Taiwan’, *Cost Effectiveness and Resource Allocation*, vol. 19, no. 1, p. 76, Nov. 2021, doi: 10.1186/s12962-021-00331-2.

[6] W. He, Z. (Justin) Zhang, and W. Li, ‘Information technology solutions, challenges, and suggestions for tackling the COVID-19 pandemic’, *International Journal of Information Management*, vol. 57, p. 102287, 2021, doi: https://doi.org/10.1016/j.ijinfomgt.2020.102287.

[7] A. Lal, H. C. Ashworth, S. Dada, L. Hoemeke, and E. Tambo, ‘Optimizing Pandemic Preparedness and Response Through Health Information Systems: Lessons Learned From Ebola to COVID-19’, *Disaster Medicine and Public Health Preparedness*, pp. 1–8, 2020, doi: 10.1017/dmp.2020.361.

[8] *Estonia’s Digital Appointment System Web Portal*. Health and Welfare Information Systems Centre. Accessed: May 02, 2022. [Online]. Available: https://www.digiregistratuur.ee

[9] ‘TEHIK Home Page’. https://www.tehik.ee/ (accessed Apr. 27, 2022).

[10] D. Kuryazov, D. Jabborov, and B. Khujamuratov, ‘Towards Decomposing Monolithic Applications into Microservices’, in *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*, 2020, pp. 1–4. doi: 10.1109/AICT50176.2020.9368571.

[11] L. De Lauretis, ‘From Monolithic Architecture to Microservices Architecture’, in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 93–96. doi: 10.1109/ISSREW.2019.00050.

[12] K. Gos and W. Zabierowski, ‘The Comparison of Microservice and Monolithic Architecture’, in *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, 2020, pp. 150–153. doi: 10.1109/MEMSTECH49584.2020.9109514.

[13] VMware, ‘Spring Boot Documentation’. https://spring.io/projects/spring-boot (accessed Apr. 27, 2022).

[14] Oracle, ‘Java 1.8 Documentation’. https://docs.oracle.com/javase/8/docs/ (accessed May 01, 2022).

[15] PostgreSQL Global Development Group, ‘PostgreSQL About Page’. https://www.postgresql.org/about/ (accessed Apr. 27, 2022).

[16] Apache Software Foundation, ‘Apache Maven Home Page’. https://maven.apache.org/ (accessed May 01, 2022).

[17] Health and Welfare Information Systems Centre, ‘Estonia’s Digital Appointment system information page’. https://www.tehik.ee/uleriigiline-digiregistratuur (accessed May 02, 2022).

[18] Nordic Institute for Interoperability Resources, ‘X-Road Architecture’. https://x-road.global/architecture (accessed Apr. 24, 2022).

[19] Health Level Seven International, ‘HL7 Version 3 Product Suite’. https://www.hl7.org/implement/standards/product_brief.cfm?product_id=186 (accessed Apr. 29, 2022).

[20] ‘Introduction to XML’. https://www.w3schools.com/xml/xml_whatis.asp (accessed Apr. 29, 2022).

[21] TEHIK, ‘TEHIK’s Health Document Standards Publication Portal’. https://pub.etervis.ee/standards2/Standards/DR (accessed Apr. 27, 2022).

[22] TEHIK, ‘Tervise Infosüsteem Information Page’. https://www.tehik.ee/tervise-infosusteem (accessed Apr. 27, 2022).

[23] Software AG, ‘WebMethods Documentation’. https://docs.webmethods.io/integration/overview/basics/#gsc.tab=0 (accessed Apr. 24, 2022).

[24] Eesti Haigekassa, ‘65-aastaseid ja vanemaid inimesi oodatakse digiregistratuuri kaudu vaktsineerimisele’, Apr. 11, 2021. https://vaktsineeri.ee/uudised/65-aastaseid-ja-vanemaid-inimesi-oodatakse-digiregistratuuri-kaudu-vaktsineerimisele/ (accessed May 01, 2022).

[25] Software Freedom Conservancy, ‘Git Version Control’. https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (accessed May 02, 2022).

[26] javatpoint, ‘JVM (Java Virtual Machine) Architecture’. https://www.javatpoint.com/jvm-java-virtual-machine (accessed Apr. 27, 2022).

[27] Micronaut Foundation, ‘Micronaut Framework’. https://micronaut.io/ (accessed Apr. 27, 2022).

[28] ‘Introducing JSON’. https://www.json.org/json-en.html (accessed Apr. 29, 2022).

[29] ‘Java Architecture for XML Binding’. https://javaee.github.io/jaxb-v2/ (accessed May 01, 2022).

[30] Oracle, ‘Java 16 Documentation’. https://docs.oracle.com/en/java/javase/16/ (accessed May 01, 2022).

[31] ‘JFROG Artifactory’. https://jfrog.com/artifactory/ (accessed Apr. 29, 2022).

[32] T. Nolle, S. Moriah, and D. Linthicum, ‘SOAP (Simple Object Access Protocol)’. https://www.techtarget.com/searchapparchitecture/definition/SOAP-Simple-Object-Access-Protocol (accessed Apr. 27, 2022).

[33] Mozilla Corporation, ‘An overview of HTTP’. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (accessed May 01, 2022).

[34] *Gatling Load Testing Framework*. Gatling Corp. Accessed: May 02, 2022. [Online]. Available: https://gatling.io/

[35] Ministry of Social Affairs, ‘COVID-19 vaktsineerimine Eestis - ametlik info - statistika’. https://vaktsineeri.ee/covid-19/statistika/ (accessed May 06, 2022).

[36] ‘HL7 FHIR Standard Documentation’, Nov. 01, 2019. http://hl7.org/fhir/

[37] L. Gupta, ‘What is REST’, Apr. 07, 2022. https://restfulapi.net/ (accessed Apr. 24, 2022).

[38] M. Ayaz, M. Pasha, M. Alzahrani, R. Budiarto, and D. Stiawan, ‘The Fast Health Interoperability Resources (FHIR) Standard: Systematic Literature Review of Implementations, Applications, Challenges and Opportunities’, Sep. 2021, doi: 10.2196/21929.

Appendix 1 - Non-exclusive licence for reproduction and publication of a graduation thesis

I Richard Õnnis

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Readiness of Estonia’s digital appointment system to health crisis: Lessons from the COVID-19 pandemic", supervised by Juri Belikov, Maksim Boiko, Maksim Zhukov
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the nonexclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

09.05.2022