UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Kert Männik

From Legacy to Microservices: A Case Study on Automatic Investment at LHV

Master's Thesis (30 ECTS)

Supervisor: Fredrik Payman Milani, PhD

Tartu 2023

From Legacy to Microservices: A Case Study on Automatic Investment at LHV

Abstract:

Microservices are a popular choice as an enterprise application architecture because they offer many benefits when compared to monolithic applications. Monolithic application is a good choice when an enterprise is small, but as the engineering department expands, the monolithic architecture drawbacks become even more visible. Then comes a moment when a monolithic application needs to be split to make use of the microservices architecture. This study is a case study in a large banking enterprise where a part of monolithic application was migrated to a microservice.

Keywords: Microservices, monolithic architecture, software architecture, system transformation, application migration

CERCS: P170 Computer science, numerical analysis, systems; P175 Informatics, systems theory; T120 Systems engineering, computer technology

Pärandrakenduselt mikroteenustele: automaatse investeerimise juhtumianalüüs LHV-s

Lühikokkuvõte:

Mikroteenused on viimastel aastal populaarsemaks muutunud, kuna pakuvad võrreldes monoliitsete rakenduste ees erinevaid eeliseid. Monoliitrakendused on hea viis ettevõtte alustamisel, kuna monoliitset süsteemi on lihtsam arendada. Ettevõtte kasvades aga tulevad välja antud arhitektuuri puudused ning tekib vajadus monoliidset rakendust väiksemateks rakendusteks tükeldada ning kasutada mikroteenuste arhitektuuri. Antud uuring sisaldab juhtumianalüüsi panganduse suurettevõttes, kus migreeriti osa monoliidist mikroteenusesse ning valideeriti uue lahenduse võimekust ning parendusi.

Võtmesõnad: Mikroteenused, monoliit, tarkvara arhitektuur, süsteemi ümberkujundamine, rakenduste migratsioon

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine; P175 Informaatika, süsteemiteooria; T120 Süsteemitehnoloogia, arvutitehnoloogia

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Banking has become significantly more digitalized and automated in recent decades, with advancements in technology leading to a range of benefits for both customers and banks [1]. Online and mobile banking platforms now allow customers to access their accounts, transfer funds, and make payments from the comfort of their own homes. Automation has also streamlined many banking processes, reducing the need for manual labour and improving efficiency [2]. For example, automatic bill payments, digital account opening, and online loan applications are just a few examples of the ways in which banking has become more automated. Furthermore, the use of AI and machine learning has enabled banks to better understand customer behaviour and preferences, leading to personalized products and services [3]. Overall, the trend towards digitalization and automation is likely to continue, as banks strive to meet the evolving needs and expectations of their customers.

LHV started in the 90's as an investment union and that is also the time period when the first software applications were developed to support back-office operations. The initial choice for the programming language was ColdFusion and some part of the bank still runs on that language. Most of the code was written by selected developers and the code has become hard to maintain. The written solutions also do not support the increased system load which has been created by the increasing number of customers. Most of the solutions use basic for-loops to process data, but that way of manipulating data is time consuming and only supports linear processing. That has created performance problems and made back-office operations time consuming [4].

Growth Account (GA) is an investment product at LHV, and it was released to customers at the end of January 2001 [5]. Being over 20 years old, the fundamental principal is still the same [6]: offer automated dollar-cost averaging fractional trading to customers. The process automatically buys instruments on behalf of the customer and customer's only actions are to sign an agreement and keep transferring money to the bank account. Like most of the legacy solutions at LHV, GA purchase process written over 20 years ago and it is showing its age: as the number of customers has risen significantly in that time period, the software solution is performing badly and needs to be rewritten and redesigned.

Monolithic and microservices architecture are two approaches to building software applications. In a monolithic architecture, the entire application is built as a single, interconnected unit. All the components, such as the user interface, business logic, and data storage, are tightly coupled and run on a single server or platform [7]. On the other hand, in a microservices architecture, the application is divided into a collection of small, independent services, each of which is responsible for a specific function [8]. These services communicate with each other through well-defined APIs and can be developed, deployed, and scaled independently [9].

The main difference between these two architectures is the level of modularity and flexibility they offer [10]. Monolithic architecture is simpler to build and deploy, but it can be challenging to maintain and scale as the application grows. Microservices architecture, on the other hand, offers greater flexibility and scalability, as services can be added, removed, or replaced without affecting the entire system. However, microservices architecture can be more complex to build and manage, as it requires additional infrastructure and communication mechanisms to ensure service availability and reliability. Ultimately, the choice of architecture depends on the specific needs and requirements of the application.

1.1 Research Objective

The existing GA purchase process is time-consuming and cannot handle the increasing number of customers: the account processing keeps growing and is negatively affecting the back-office workflows. The existing process has been maintained and improve several times, but because the core logic is dated, the future improvements cannot deliver any acceptable benefits. Another major reason for change is that the existing process is a part of the legacy system and major changes are forbidden by the LHV IT-department. That rule conflicts with LHV business need to offer customers high-quality products and services. A minor shortcoming of the existing process is that it lacks the needed transparency from the software developer perspective to debug or analyse the process if an exception happens.

This thesis addresses the following research question: **RQ**. *How to migrate a part of the legacy monolith application to a microservice?* To answer the research question, we followed the design science approach. At first, we set the problem statement to understand the system we want to improve. This was done in cooperation with a product owner. Secondly, we identified the requirements for the new artifact. Next, the new artifact was developed based on the requirements. After that, the new artifact was evaluated according to the requirements. Lastly, a set of insights and learnings were proposed based on the findings how to migrate an existing part of a legacy system to a microservice. This contribution will be useful to those involved in the software engineering and architecture, for example, software engineers, team leads, software architects, product owners, etc. The contribution of this thesis is a set of insights and learnings from migrating a part of the legacy system to an independent microservice.

The rest of the thesis is structured as follows. Section 2 describes the background of this thesis, briefly outlining LHV and its products, different software architectures, Omnibus account logic, and fractional shares. Section 3 goes into detail how the research was performed. Section 4 is about the artifact and everything related to that: explanation, requirements, technological choices, process overview, migration, and evaluation. Section 5 provides a discussion of the findings and Section 6 is a conclusion about the thesis.

2 Background

This section introduces the key concepts and notions used in this thesis, such as LHV Group and its products, software architectures, and omnibus account.

2.1 LHV Group AS

LHV Group AS is a holding company and LHV is known for its focus on digital innovation and customer-centric approach. Founded in 1999, the bank offers a range of services, including banking, investment, and pension products. LHV has been at the forefront of Estonia's e-government initiatives and has developed several digital solutions, such as the mobile payment app, which has gained widespread popularity. With a strong emphasis on innovation and sustainability, LHV is a leading player in Estonia's financial sector [11].

LHV has been growing rapidly in Estonia and has around 530 000 customers by the end of January 2023 [4]. The bank has offices in Tallinn, Tartu, and Pärnu. In the last years, LHV Group has been expanding to UK and founded a filial LHV UK Limited in October 2021.

2.1.1 Growth Account

Growth Account (GA) is a LHV investment product that enables customers to start investing with minimum amount worth of 1 EUR on their account [12]. Customer can choose up to 10 allocations per account and after transferring money to the account, those allocations are purchased automatically on every Wednesday. GA offers fractional share trading so customer can use the total amount of money available on the account. The currency conversions are automatically performed for the customer.

GA automates the maintenance of the investment portfolio and after the customer has created their portfolio, GA utilizes its proprietary trading system to automate all their trading activity. Customers never have to input or execute any buy trades manually. GA automatically uses the available money to buy new shares proportional to the ratios. GA does not offer portfolio rebalancing.

For the customer, the only manual part of this product is the sell orders, which must be entered manually. Customers cannot manually enter buy orders; those are only created by the automatic investment process. Over 50 000 customers use GA to invest in fractional shares.

GA offers different exchange traded funds and individual stocks for the customers. By the end of January 2023, the 3 most popular allocations are iShares Core S&P 500 UCITS, iShares NASDAQ-100 UCITS and Vanguard FTSE All-World UCITS ETFs [13].

2.1.2 Pension Investment Account

Pension Investment Account (PIA) is a separate bank account, which customers can use to grow their II Pillar pension assets, while making the investment decisions independently [14]. Customer can direct their regular pension payments prescribed for II Pillar members to their PIA. Customer cannot make any additional payments to their PIA. LHV PIA has a feature to automatically invest customer allocations each Wednesday [15]. The setup and business logic are the same as GA automatic investment process.

Customer can also withdraw the funds from PIA. When withdrawing money before retirement age, they must pay income tax of 20% on the withdrawn amount. After withdrawing, they can rejoin the pension system (including opening a new PIA) after 10 years.

The main difference between PIA and GA automatic investment feature is that PIA is part of the Estonian pension system and customer cannot transfer additional money to the account, only Estonian Funded Pension Registry can make payments to the account. GA allows customer to transfer additional money to the account.

There is around 4500 active PIA at LHV, almost 1000 of those use automatic investing. By the end of November 2022, the 3 most popular fund allocations are iShares Core S&P 500 UCITS, iShares NASDAQ-100 UCITS and Vanguard FTSE All-World UCITS ETFs [16].

2.1.3 Automatic Investment

Automatic Investment is the name of the process which manages the purchase process of GA and PIA automatic investment orders at LHV. For the customer, the process is fully automated. Their only obligations are to choose allocations, sign the agreement and transfer some money to the account. The purchase process is manually started by a broker in the Investment Administrator Platform around 11 AM Estonian time each Wednesday.

LHV uses an omnibus account logic to hold fractional investment positions on behalf of the customer. Automatic Investment does not have any customer user interface, the only interface is for brokers in the Investment Administrator Platform. For customer, this process is hidden behind GA and PIA fractional orders.

Automatic Investment process finds all the active automatic investment accounts and reserves their money for fractional orders. The process aggregates all the customers' orders and executes them during a single trading window. Based on the aggregated order executions, an average purchase price is found for each order and that information is synched to customer fractional orders.

2.2 Software Architectures

2.2.1 Monolithic Architecture

Monolithic software architecture is a traditional approach to building software applications where the entire application is built as a single, interconnected unit. In a monolithic architecture, all the components of the application, such as the user interface, business logic, and data storage, are tightly coupled and run on a single server or platform. This architecture has been widely used in the past, and it can be relatively simple to build and deploy. However, it can also be challenging to maintain and scale as the application grows, since changes to one component can affect the entire system [7]. Despite its limitations, many applications still use monolithic architecture due to its simplicity and familiarity.

2.2.2 Microservices Architecture

Microservices software architecture is a modern approach to building software applications that emphasizes modularity and scalability. In a microservices architecture, the application is divided into a collection of small, independent services, each of which is responsible for a specific function [8]. These services communicate with each other through well-defined APIs and can be developed, deployed, and scaled independently. This architecture allows for greater flexibility, as services can be added, removed, or replaced without affecting the entire system. It also enables teams to work on different services simultaneously, which can lead to faster development cycles [9]. However, microservices architecture can be more complex to build and manage, as it requires additional infrastructure and communication mechanisms to ensure service availability and reliability.

2.3 Omnibus Account

An omnibus account is a type of investment account that is held by a financial institution on behalf of multiple clients [17]. In this arrangement, the financial institution acts as the account holder and is responsible for managing the account, while individual client holdings are not visible. Instead, each client's assets are aggregated and held together with those of other clients in the same account. This can be advantageous for institutions, such as broker-dealers or investment banks, as it allows them to aggregate trades and transactions across multiple clients, resulting in lower costs and more efficient execution. However, omnibus accounts can also pose certain risks, such as increased complexity and reduced transparency, as individual client transactions may not be visible or auditable [18].

2.4 Fractional Share

A fractional share is when a customer owns less than one whole share of a company [19]. Fractional shares allow investors to invest in instruments based on a fiat currency amount, so they may end up with a fraction of a share. The main benefit for the customer is that they can start investing with smaller amounts of money thus lowering the barrier for investing [20]. For example, if share is priced at 100 € and customer has 10 €, then they can purchase 0.1 of that share.

3 Methodology

This section details the research process and describes how the input required for artifact development was collected and analysed.

This thesis aim is to understand how to migrate a part of the legacy system to a microservice. The research objective, formulated as RQ: *How to migrate a part of the legacy monolith application to a microservice?* can be answer differently and has many acceptable answers. To achieve this research objective, we use the design-science research guidelines [21]. This approach is a good fit for us, because our goal is to develop and evaluate the created artifact. We need to find a way how scale up the old artifact, separate it from the legacy system and move it to the Investment domain. The old artifact cannot keep up with the increasing number of accounts and bottlenecks other back-office processes. There is a real need for the new artifact as there are over 50 000 active GA accounts and almost 1000 PIA automatic investment accounts.

By following the guidelines, our first step is to understand the problem statement. We approach this by analysing the existing application and find its limitations and shortcomings. We use the problem statement analysis result to come up with different requirements for the artifact development. When the artifact is developed, we will evaluate it according to the requirements created in the previous step. The research process is presented in Figure 1.

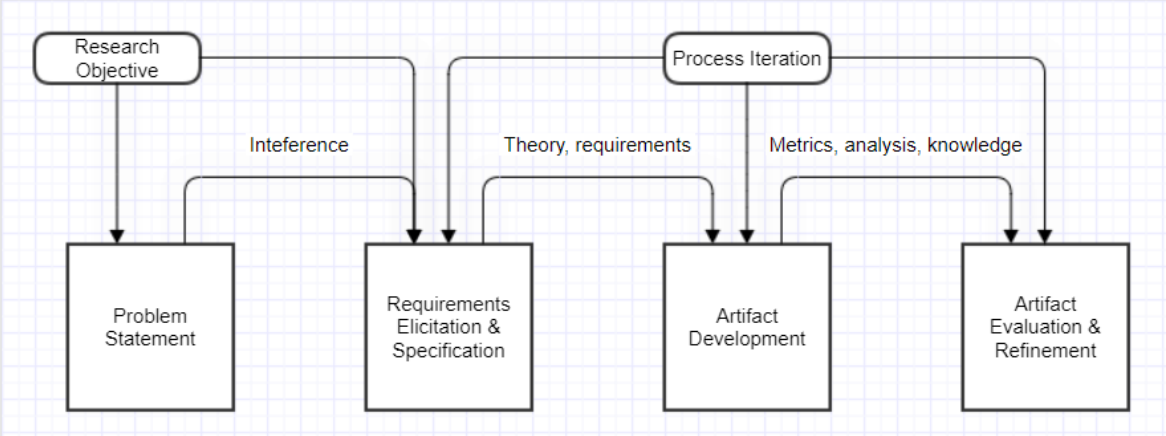


Figure 1. Research process

3.1 Problem Statement

In this design-science research guideline step [21], we define the scope of the problem that the microservice will solve and identify the specific features or functionalities that will be migrated. This will help to ensure that the microservice is focused on solving a specific problem and is not overly complex or difficult to maintain.

In the problem statement phase, we will also have to determine the appropriate architecture and design for the microservice. This should include defining the APIs and communication protocols that the microservice will use to interact with the legacy system and other microservices.

We also have to identify the data that will be migrated to the microservice and determine how it will be stored, managed, and accessed. This should include defining the data schema, data storage mechanisms, and data access patterns.

3.2 Requirements Elicitation & Specification

In this design-science research guideline step, we gather and manage the requirements for the artifact development [21]. We will identify and gather functional and non-functional requirements for the new artifact. This should involve gathering input from different stakeholders, including end-users, business analysts, and product owner.

The initial requirement gathering will be conducted by a product owner who is responsible for the Automatic Investment process. They will perform informal interviews with different stakeholders. Another method will be reviewing in-house documents about the existing process to fully understand and replicate its main functionality.

After the initial requirement gathering, we will analysis those requirements and prioritize them according to the feasibility and impact. We should identify any potential conflicts and dependencies between requirements and prioritize them based on the importance on the overall system. Also, we will have to ensure that the requirements are complete, accurate, and consistent. This should involve reviewing the requirements with relevant stakeholders and many event prototyping the requirements to validate their feasibility.

3.3 Artifact Development

In this design-science research guideline step, we will develop the artifact based on the requirements set in the Requirements Elicitation & Specification [21]. This involves coming up with design, implementing the design, integration with other applications and testing the outcome.

We will design the microservice based on the requirements and scope definition, this includes designing the architecture, components, and interfaces of the artifact. During the design phase, we must think about other relevant systems or services, which the new artifact must communicate with using different communication protocols and APIs. After that, we will implement the microservice according to the design specifications.

The artifact will be developed by the thesis author who is a developer at LHV, because it is more cost-effective than outsourcing the development or buying an existing solution from a third-party company. One of the arguments to support in-house development was that the final solution must be fully integrated to existing platform at LHV. That integration and development knowledge would be hard to find outside of the LHV IT-department.

The artifact will be developed using different Agile software development techniques. The main method will be Scrum which is a framework to define the guidelines with rules, roles, artifacts and events and describe an iterative and incremental approach to software development process [22]. The amount of work in the system will be visualized on a Kanban board and the work will be categorized into different statuses [23].

3.4 Artifact Evaluation & Refinement

In this design-science research guideline step, we will evaluate the artifact created in the guideline step Artifact Development [21]. We will identify the criteria that measures the success of the microservice. These may include factors such as performance, scalability, reliability, maintainability, etc.

The main focus will be the account processing step in the automatic investment event. The reason is that the old artifact cannot handle the increasing number of accounts to process, and that processing has become a major bottleneck in the old artifact.

We will interpret the results of the evaluation and draw conclusions about the effectiveness of the microservice. This should involve identifying any strengths and weaknesses of the microservice. After the evaluation, we will make recommendations for further development or improvement of the microservice.

There are many qualitative and quantitative metrics to evaluate the new artifact. Architecture, parallel processing, decoupled services, transparency, and documentation & knowledge transfer are the qualitative metrics. The migration from the legacy system and moving the process to a microservice is one of the main areas of this thesis so it makes architecture evaluation relevant. Because the account processing step must process thousands of accounts in a short time period, the parallel processing qualitative metric is also relevant to this artifact evaluation. The old artifact uses several database procedures to perform some actions to the accounts so that makes decoupled services metric important to us. The process transparency is another key metric because that makes the process easy to track and debug if an exception happens. The last qualitative metric is documentation and knowledge transfer. That is relevant to this thesis because the old artifact was developed over 20 years ago and none of the Investment domain software developers were part of that original software development team.

To perform the quantitative evaluation, the data will be extracted from the application log files. The script will be written in a log parser application which is used at LHV to display and analysis logs. The log parses applications reads and parses raw log files to return a structured format. The entire quantitative evaluation is built upon application logs, for example, to evaluate how long did one specific account currency conversion took, then two log lines were written: “Starting account XXX currency conversion step.” and “Finished account XXX currency conversion step.” Because each log was a timestamp when we subtract the finishing timestamp from the starting timestamp to get the currency conversion duration. Because the account processing step consist of several sub steps, then the entire account processing can be thoroughly analysed.

4 Artifact

This section describes the development of the artifact.

4.1 Explanation

The need for a new solution arose due to several reasons. Firstly, the existing process was written in a legacy system that had become difficult to maintain. Additionally, it was written in a monolithic software system using practices that were no longer acceptable under the latest software development guidelines at LHV. Furthermore, the existing process was unable to cope with the growing number of customers, with processing time increasing linearly over the past few years. In addition, the existing data model did not meet the product manager's requirements, and the solution was written in ColdFusion programming language, which is no longer the primary programming language used at LHV. The existing solution also had an outdated user interface and was not intuitive nor user-friendly, with no data evaluation or transparency, making it impossible to track the progress of the process.

4.2 Requirements

The following requirements were elicited from a product owner and the main focus are the non-functional requirements rather the functional requirements:

- It should be able to process up to 100,000 accounts per event.
- It should be able to process up to 250,000 fractional orders per event.
- It should be able to process up to 500 trade orders per event.
- The solution needs to process all the accounts within 30 minutes.
- It must adhere to the latest software development standards set by the LHV IT department.
- The solution should support GA and PIA automatic investment accounts.
- It needs to work with all major stock exchanges worldwide.
- It should require minimal input from users.
- The solution should have a user interface that allows brokers to easily track the progress of the process and interact with it.
- The solution must be able to partially fill orders if a trade order was not fully executed on the stock exchange.

4.3 Technological Choices

This section gives a brief overview of different programming and software development technologies chosen for the new artifact.

4.3.1 Back-end programming language

Java is a popular and widely used programming language for several reasons [24]. First and foremost, it is platform-independent, meaning that code written in Java can run on any device or operating system with a Java Virtual Machine (JVM) installed. This makes it highly versatile and accessible. Additionally, Java is an object-oriented language, making it easier to write and maintain complex applications. It is also a secure language, with built-in features for handling security issues like memory leaks and buffer overflow errors. The language has a large and active developer community, providing resources and support for those learning and working with Java. Finally, Java is well-suited for enterprise-level applications due to its scalability, reliability, and performance. Overall, Java is a solid choice for programmers looking for a flexible, powerful, and secure language for their projects.

Java programming language version 11 was chosen for the solution because it has a wide community support, does static type checking, is object-oriented. The main reason is that Java is the main programming language at LHV and has passed different validations and security tests performed by the IT-department. Java also supports threads which allows an application to operate more efficiently by processing multiple threads at the same time. This is a useful feature to process each individual account in parallel [25].

4.3.2 Back-end framework

Spring Boot is a popular framework for several reasons [26]. First, it simplifies the process of building and deploying web applications, providing a range of pre-built components and templates that speed up development time. Spring Boot also uses a convention-over-configuration approach, which reduces the need for extensive configuration, making it easier for developers to focus on building the application's core functionality. Additionally, Spring Boot integrates well with other frameworks and libraries, providing a high degree of flexibility and customization. The framework also has a large and active community, with extensive documentation and support available. Finally, Spring Boot is scalable and reliable, making it well-suited for large and complex enterprise-level applications. Overall, Spring Boot is a powerful and versatile framework that can significantly streamline the process of developing web applications.

Spring Boot framework version 2.7 was chosen because it is one of the popular Java frameworks, is easy to use and reduces the development time. Spring Boot also different abstractions to implement different solutions more easily. For example, Spring provides a JMS integration framework that simplifies the use of the message brokers [27].

4.3.3 Front-end framework

Angular is a widely used framework for building web applications, and it is known for its versatility and power [28]. One of the main advantages of Angular is that it is a fully featured framework that offers a comprehensive suite of tools and features for building complex and scalable applications [29]. Angular is also based on TypeScript, which adds strong typing to JavaScript, making it easier to build and maintain large-scale applications. The framework includes powerful templating and data binding capabilities, which help developers to build rich and dynamic user interfaces. Angular also offers a wide range of third-party libraries and plugins, making it easy to add additional functionality and features to your application. The framework has a large and active developer community, providing plenty of resources and support for those learning and working with Angular. Finally, Angular is well-suited for building cross-platform applications, with support for mobile platforms and progressive web apps. Overall, Angular is a solid choice for developers looking for a powerful and flexible framework for building modern web applications.

Angular was chosen for the solution because it is the main standard at LHV Bank at the moment.

4.3.4 Database server

Microsoft SQL Server is a powerful and widely used relational database management system [30]. It offers a comprehensive suite of tools for data storage, management, and analysis, and supports a wide range of programming languages, making it a versatile tool for developers and data analysts. SQL Server is designed to manage large-scale data processing tasks. With its robust security features and advanced data management capabilities, SQL Server is a top choice for enterprise-level applications.

Microsoft SQL Server 2019 was chosen for the solution because it is the main standard at LHV Bank at the moment.

4.3.5 Message broker

Apache ActiveMQ is a popular open-source message broker that facilitates communication between different applications and systems using messaging protocols [31]. It provides a reliable and scalable messaging system that supports a wide range of messaging patterns and protocols, including JMS, AMQP, and STOMP. ActiveMQ allows for the decoupling of different components in a distributed system, making it easier to build and maintain complex systems. It also offers advanced features like message filtering, message persistence, and message prioritization, making it a powerful tool for building high-performance messaging systems. Overall, Apache ActiveMQ is a robust and flexible message broker that is widely used in enterprise-level applications for reliable message delivery and management. Message broker is needed for the solution to process the accounts in parallel.

ActiveMQ 5.17 was chosen for the solution because it offers JMS API and has Spring Support so it can be easily embedded into Spring applications.

4.4 Process Overview

The process consists of several steps which are described in this section. The high-level overview is presented in Figure 2. The new artifact architectural overview is presented in Figure 3.

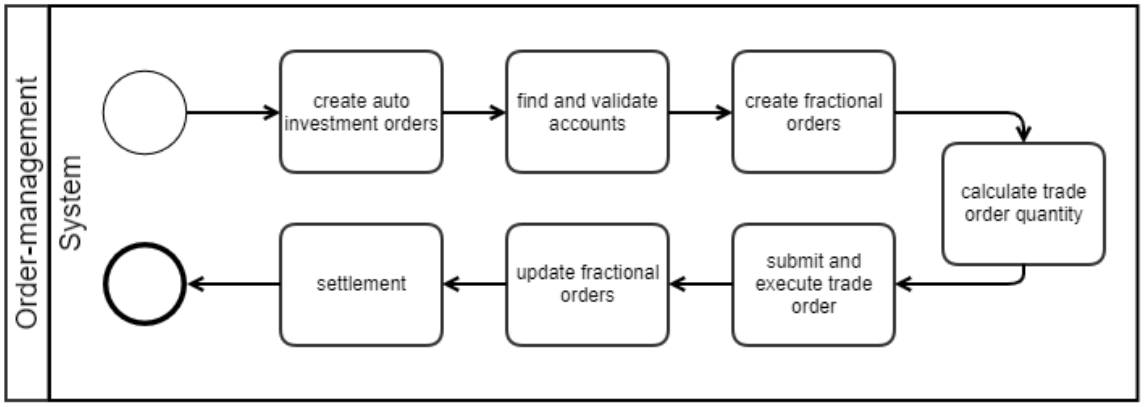


Figure 2. The high-level overview of the process

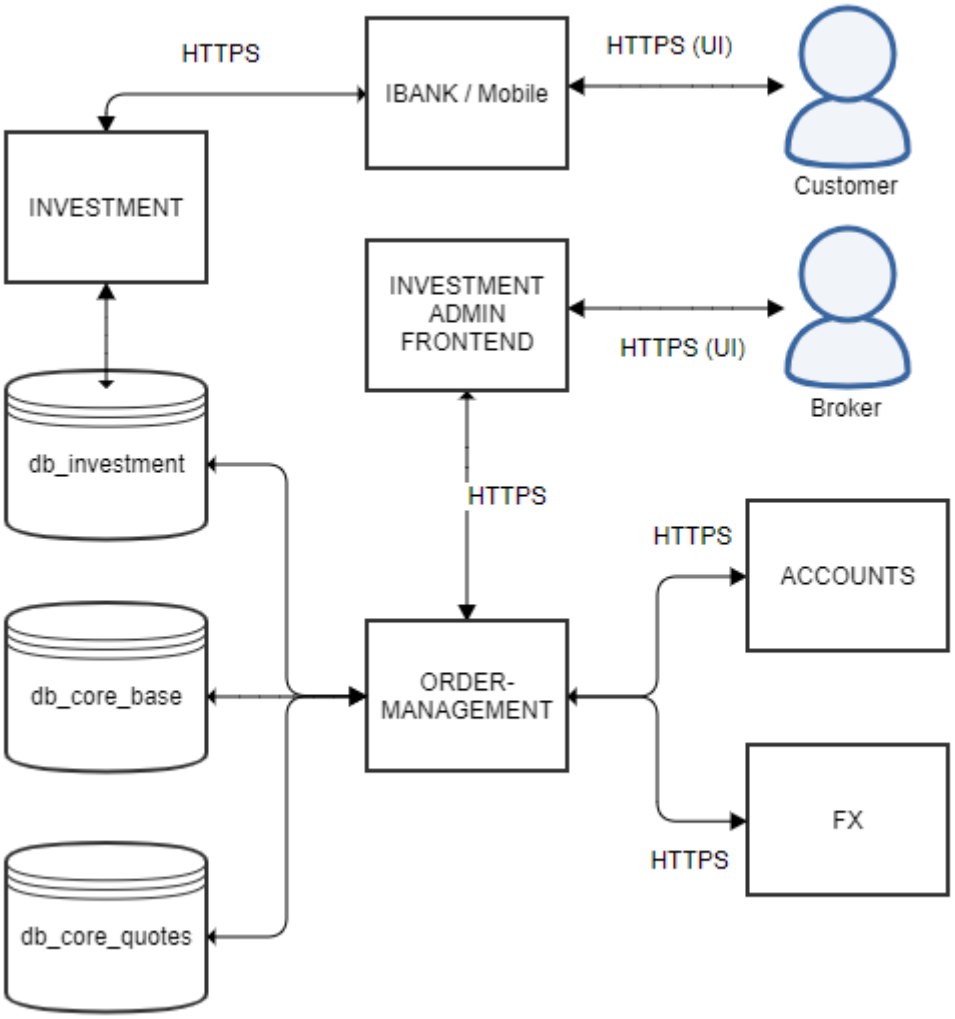


Figure 3. The new artifact architectural overview

4.4.1 Preparation

The process is started in the Investment domain administrator platform by a broker. The landing page can be seen in Figure 4.

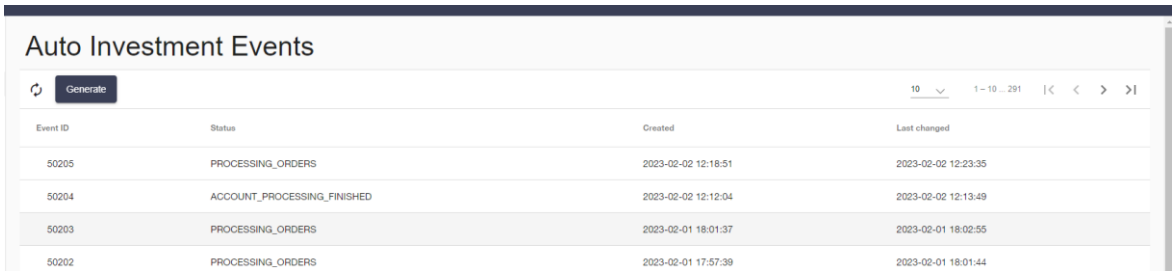


Figure 4. The new artifact landing page

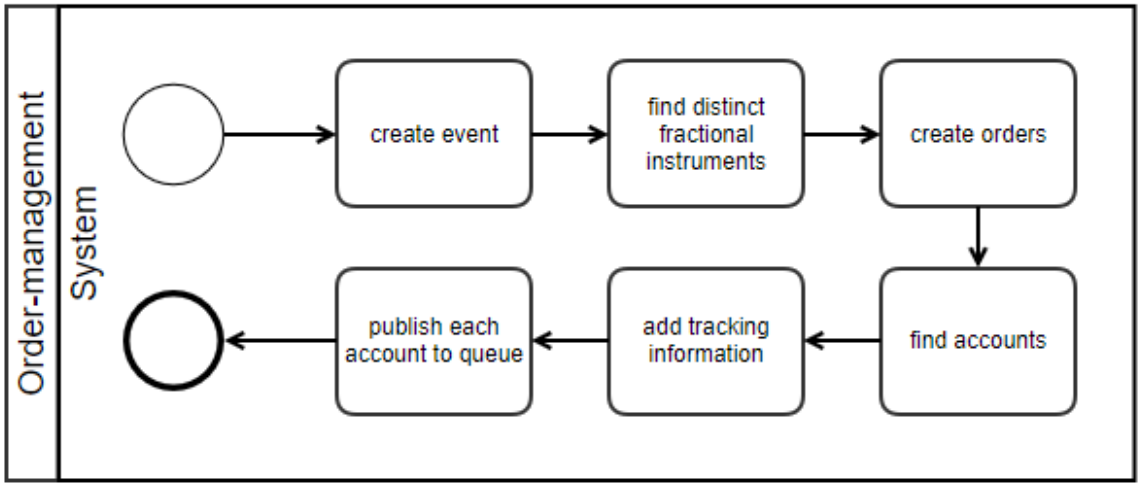


Figure 5. The preparation step overview

As shown in Figure 5, the process starts by creating a new automatic investment event. The event status is PREPARATION_STARTED. After the event creation, the process finds each unique instrument what customers have chosen for their portfolio. That is done by fetching customers' allocations from the database and finding the instruments. Because brokers can change instruments and their statuses then each fetched instrument is validated so it must be active and can be bought. For each instrument, we create an automatic investment order which is linked to the event. Each automatic investment order status is PENDING.

After the automatic investment order creation, we start finding the accounts for the process. The process fetches the PIA and GA accounts separately, but the results are merged. For each PIA, the user must be active, must have active PIA agreement and has enabled automatic investment for their account. For each GA, the user has similar validations, but the validated agreement is GA agreement. All of those validations are done by database queries.

After all the accounts are found for the event, the event status is changed to PROCESSING_ACCOUNTS and the accounts are produced to the queue by ActiveMQ publisher.

4.4.2 Account processing

As shown in Figure 6, each account processing starts by fetching customer information from the database. That information is needed to perform validation for the account and customer. If the validation is negative, then the account processing is terminated.

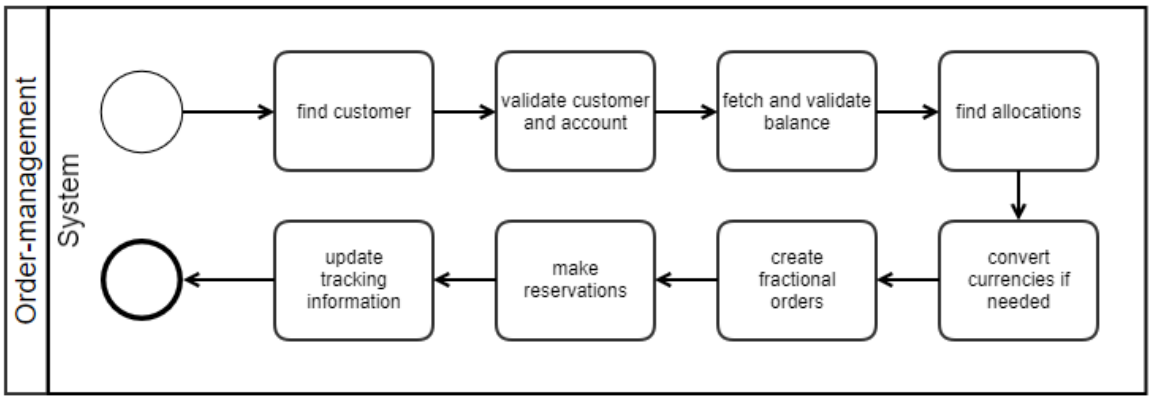


Figure 6. The account processing overview

For the positive validation result, the process carries on by requesting the account balance from an external system. There is a business requirement that to process the account, the account balance must be worth at least 1 EUR. If there is not enough balance, then that account processing is terminated.

In case of a positive validation result, the process continues and fetches account allocations from the database. Each allocation is also validated because it can be inactive and cannot be used. If that is true, then that allocation ratio is equally distributed between the remaining allocations. If all account allocations are invalid, then the account processing is terminated.

If there is at least one valid allocation, then the allocations are used to calculate the necessary amount of each currency. That is done by calculating the entire portfolio's value in EUR and multiplying it by the allocation ratio. The result is converted to the allocation currency. That calculation is done for each allocation and the results are summed by each currency. The result of this logic is called the necessary currencies.

The necessary currencies are then subtracted by the available balance already present on the account. If the currency is not present in the account balance, then the necessary currency amount remains unchanged.

After that, we have two results:

- a list of necessary currency amounts
- a list of available currency amounts

Those lists are combined into currency conversion pairs and those pairs are sent to an external system which performs the currency conversions. If all the conversions are finished, then the process moves on to create fractional orders for the account. Each fractional order is linked to its automatic investment order. The link is created if they have the same instrument. The fractional order status is PENDING. At the moment, each fractional order only has monetary value. For each fractional order, a reservation is created to lock the money so it cannot be spent in the meantime. The reservation requests are sent to an external system. Each reservation also contains the fractional order purchase fee.

After the reservation requests, the account processing is finished.

If all the accounts are processed, then the event status is changed to ACCOUNT_PROCESSING_FINISHED which signals the broker that they can move on the next step of the process.

4.4.3 Aggregation and routing

Now, we have many automatic investment orders which have fractional orders.

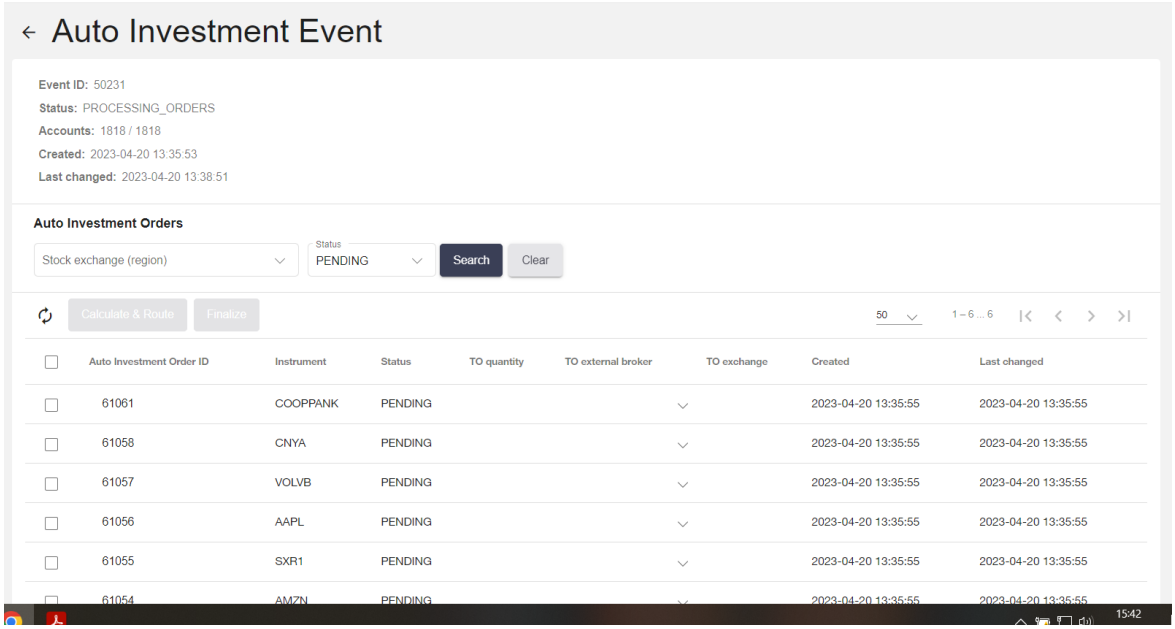


Figure 7. The automatic investment order aggregation and routing view

For each automatic investment order, the broker must start its aggregation. The automatic investment order list is presented in Figure 7. The aggregation contains of summing up the monetary value of its fractional orders. Then the sum is divided by the latest market price. The Quotient is always rounded up to the next integer. The result is the necessary quantity which LHV must buy from the stock exchange. That information is used to create a trade order. The automatic investment order status is changed to TRADE_ORDER_CREATED. The entire trade order creating process is summarized in Figure 8.

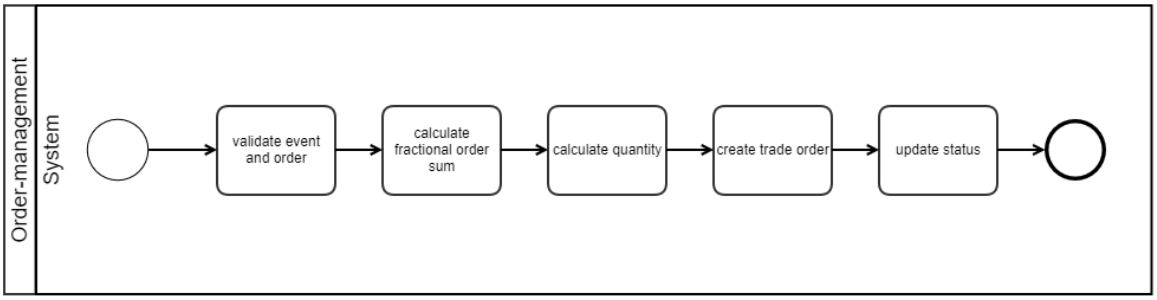


Figure 8. Trade order creating overview

The trade order is routed to the stock exchange. Each automatic investment order has one trade order. The automatic investment order nor the fractional orders are not sent to the stock exchange. The automatic investment order status is changed to ROUTED. That result can be seen in Figure 9.

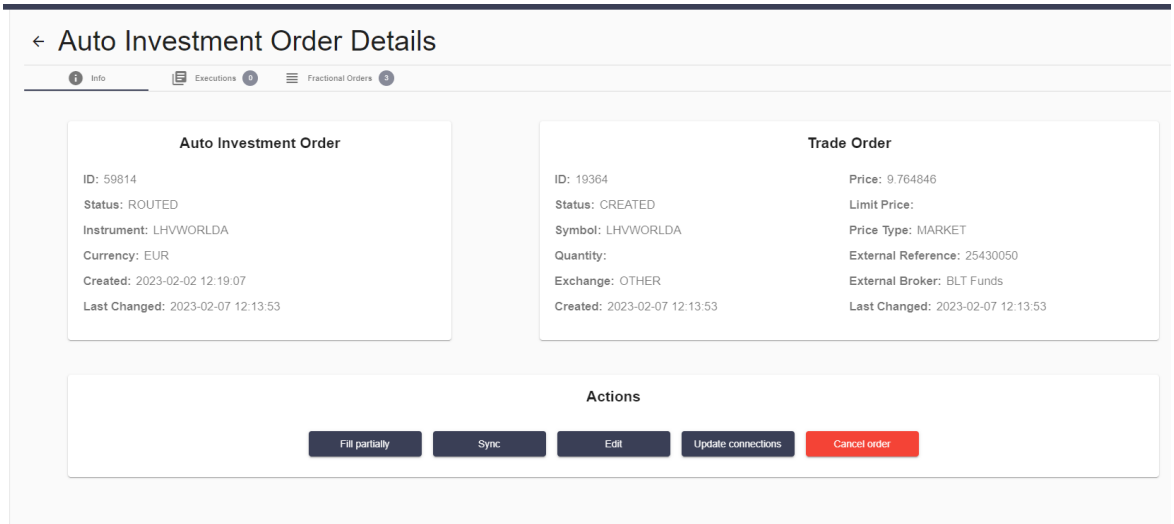


Figure 9. The routed trade order detail page

If this is the first automatic investment order aggregated and routed for that event, then the event status is changed to PROCESSING_ORDERS. The entire routing process is presented in Figure 10.

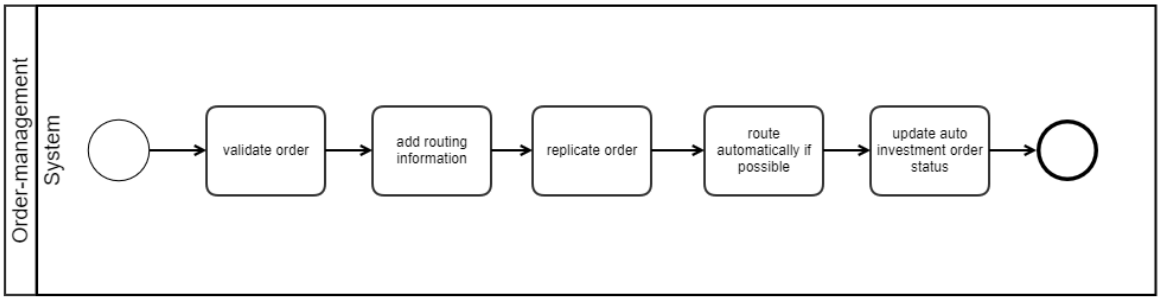
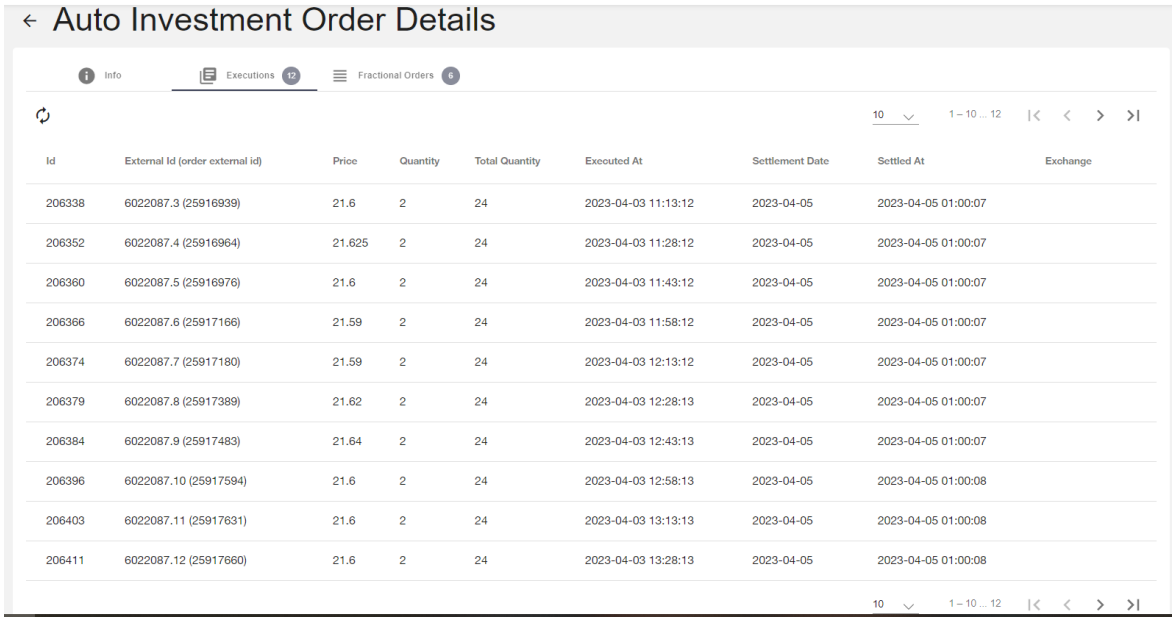


Figure 10. The trade order routing overview

4.4.4 Finalization

If the trade order is fully filled on the market, then the process can start finalizing that automatic investment order.



Id	External Id (order external id)	Price	Quantity	Total Quantity	Executed At	Settlement Date	Settled At	Exchange
206338	6022087.3 (25916939)	21.6	2	24	2023-04-03 11:13:12	2023-04-05	2023-04-05 01:00:07	
206352	6022087.4 (25916964)	21.625	2	24	2023-04-03 11:28:12	2023-04-05	2023-04-05 01:00:07	
206360	6022087.5 (25916976)	21.6	2	24	2023-04-03 11:43:12	2023-04-05	2023-04-05 01:00:07	
206366	6022087.6 (25917166)	21.59	2	24	2023-04-03 11:58:12	2023-04-05	2023-04-05 01:00:07	
206374	6022087.7 (25917180)	21.59	2	24	2023-04-03 12:13:12	2023-04-05	2023-04-05 01:00:07	
206379	6022087.8 (25917389)	21.62	2	24	2023-04-03 12:28:13	2023-04-05	2023-04-05 01:00:07	
206384	6022087.9 (25917483)	21.64	2	24	2023-04-03 12:43:13	2023-04-05	2023-04-05 01:00:07	
206396	6022087.10 (25917594)	21.6	2	24	2023-04-03 12:58:13	2023-04-05	2023-04-05 01:00:08	
206403	6022087.11 (25917631)	21.6	2	24	2023-04-03 13:13:13	2023-04-05	2023-04-05 01:00:08	
206411	6022087.12 (25917660)	21.6	2	24	2023-04-03 13:28:13	2023-04-05	2023-04-05 01:00:08	

Figure 11. The trade order executions

The trade order can have one or more executions, as shown in Figure 11, so to calculate the average purchase price, the total execution monetary amount is divided by quantity. The average purchase price is necessary to update fractional orders.

Fractional orders are processed one at a time. When processing a fractional order, several steps are taken to ensure that the order is executed correctly. First, the average purchase price is added to the fractional order. Then, the quantity of the order is calculated by dividing the fractional order's monetary amount by the average purchase price. The fractional order reservation is then cancelled to free up the necessary funds for the transaction. The transaction for the fractional order quantity and the accompanying fee is performed, and the new purchase quantity is added to the account instrument ledger. Finally, the fractional order status is updated to COMPLETED to reflect the successful execution of the order. These steps ensure that fractional orders are processed accurately and efficiently, allowing for optimal trading outcomes.

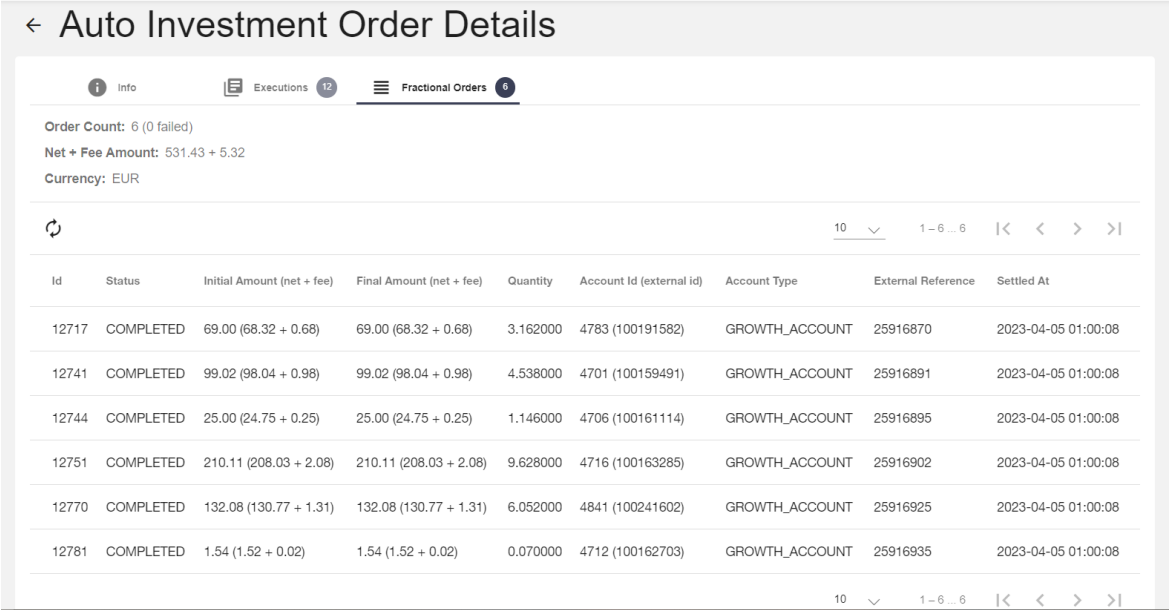


Figure 12. The finalized fractional orders overview

If automatic investment order fractional orders are finalized, as presented in Figure 12, then the automatic investment order status is changed to FINALIZED.

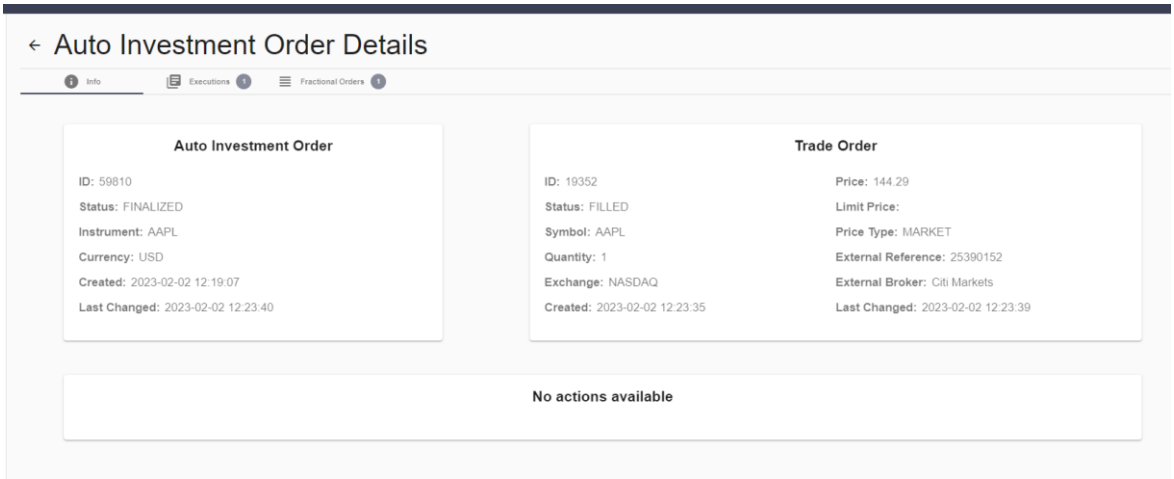


Figure 13. The finalized automatic investment order overview

If all the automatic investment orders are finalized, then the event status is changed to FINISHED. That result is displayed in Figure 13. The entire finalization process is displayed in Figure 14.

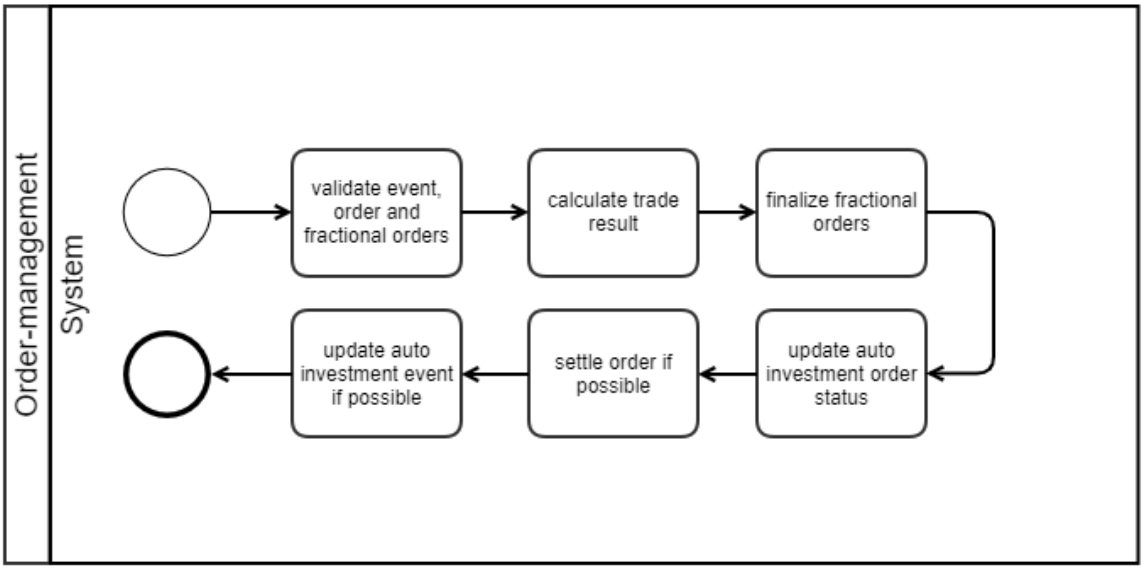


Figure 14. The finalization step overview

4.4.5 Settlement

The settlement step is separate from the automatic investment process [32]. Securities settlement is done according to the standard rules and is usually performed 2 workdays after the purchase. The settlement step is the final step in the transfer of securities ownership. Settlement involves securities delivery to the beneficiary, usually against payment of money, to fulfil contractual obligations, such as those arising under securities trades.

4.5 Migration

Migration from the old solution to the new solution took place around one month time period. The migration was done incrementally to validate the artifact reliability and system load. The main reason for this migration logic was to avoid big bang release and minimize the problem impact area if an exception happened or a development bug was found.

The first migration step was performed with selected accounts which belonged to the employees of the investment domain. This step helped to validate the logic in production environment. The sample size was five accounts.

The next migrations step took place a week after the first one and this event selected all the accounts from the beta group users. This sample size was 121 accounts.

After the first two migration steps, all GA and PIA automatic investment accounts were split into 10 different groups of the same size and the groups were migrated over incrementally. The size of each group was around 5000 accounts. This migration logic helped to analyse the new artifact workload and total processing time.

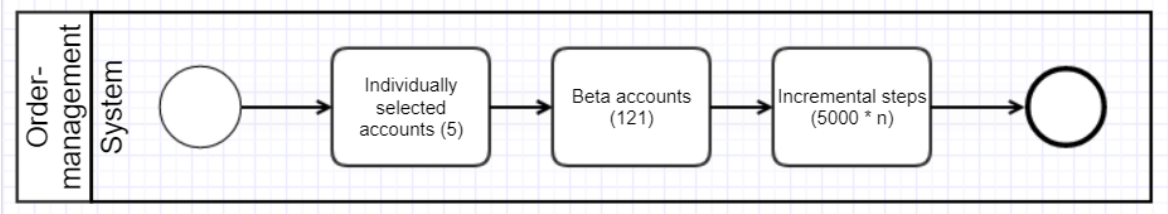


Figure 15. Migration and its steps

During the migration, the old and new artifacts were used in parallel, but they worked on different sample groups. For example, if the new artifact processed the beta group accounts, then those accounts were excluded in the old artifact. This exclusion was needed to avoid processing same account multiple times in one day.

4.6 Evaluation

4.6.1 Qualitative evaluation

4.6.1.1 Architecture

A traditional platform-based architecture generally contains all functionality in one codebase. This kind of architecture is easy to set up; however, scaling, customizing, and maintaining it can quickly become costly and complex. This means that platform-based architecture is not agile enough to meet the constantly changing needs of companies. All of this is solved with the new artifact because the entire logic was moved to a new microservice in the Investment domain. Additionally, the new artifact has its own database schema and tables which makes it easy to understand and easier to improve in the future.

4.6.1.2 Parallel processing

The old solution fetched suitable accounts from the database and started to process them linearly in a for-loop. The new solution also fetches the accounts from the database, but after that step produces one message broker event per account and that event is consumed by a message broker consumer. This solution is event-driven and can be more easily scaled up if needed. At the moment, there are 30 consumers processing the accounts, but that number can be easily increased if the number of accounts keeps growing.

In the legacy system, scaling was limited to vertical scaling due to its monolithic architecture, which made it difficult to distribute resources across multiple instances. This meant that the system's capacity was limited by the resources of a single server. The ability to horizontally scale the microservices architecture resulted in increased flexibility and better resource utilization compared to the legacy system. This allowed for better performance under high traffic conditions and improved the overall resilience of the system. The migration from the legacy system to the microservices architecture demonstrated the advantages of horizontal scaling, which can help meet the increasing demands of modern applications [33].

4.6.1.3 Decoupled services

The old solution was a part of the legacy system and used database procedures to perform certain activities to the account, for example transfer money, convert currencies etc. The new solution is part of the Investments domain and is written according to latest software development standards at LHV. If a certain activity needs to be perform, then external services are used for that. This solution creates a decoupled architecture which is easier to develop and maintain than using a monolith architecture.

Another key area which was improved in the new solutions is the database. The old solutions used legacy database which uses pessimist locking and it creates potential deadlock and increases the database operation times. The new solution mainly uses the Investment domain database which uses optimistic locking which is better suited for parallel processing and speeds up the overall process.

4.6.1.4 Transparency

The old solution had an out-of-date user interface design and had data model which was difficult to manage. The new solution uses an entirely new data model which is easier to understand and offers possible automation improvements for the future. The event has several states and that helped to create a user interface for the brokers which in easy to understand and use.

The old solution also did not have any logging, which made debugging complicated and developers had to speculate about the root cause of an exception if that happened. The new solution has happy-path and not-so-happy-path logging which makes the process easy to follow and debug if needed.

4.6.1.5 Documentation and knowledge transfer

The existing legacy solution was transferred to the Investment development team when the team was created around 5 years ago. Investment team was not part of the software development team which created the old artifact. Thanks to the new artifact, the knowledge is present in the team and process is thoroughly document.

4.6.2 Quantitative evaluation

The focus of the quantitative evaluation is on the account processing step, because that step in the process was not able to handle the number of accounts in the process.

It is difficult to compare old solution to the new one, because old solution did not have any metrics or any other way to track the data processing. The new solution has sufficient logging to track each action and its duration. For example, each account can be independently analysed to find shortcomings and potential bottlenecks.

4.6.2.1 Data extraction

Data extraction is done thanks to the log parser application at LHV. It aggregates different logs from the applications and enables to display and analyse the logged information.

It was possible to extract total processing time and the total number of accounts with fractional orders from the old artifact. We used application HTTP request and response logging information to calculate the total processing time. The number of accounts with fractions orders and the number of fractional orders were fetched from the database, because that information was not present in the logging information.

The old artifact was analysed from the 2nd of March to 20 of April of 2022. The old artifact extracted data is presented in Table 1.

Table 1. The old artifact data

Date[1]	Processing time in seconds	Number of accounts with orders	Total amount of orders	Average processing time for account with orders in milliseconds
02.03	3244	13 687	47 942	237
09.03	4092	15 744	54 534	259
16.03	4165	16 200	56 006	257
23.03	3018	13 909	48 315	216
30.03	4388	14 203	49 857	308
06.04	5588	18 945	64 816	294
13.04	4308	17 301	59 574	249
20.04	3233	13 138	45 455	246

To evaluate the new artifact account processing step, a lot of logging was added to the process to extract timestamps for each account processing sub step. Those timestamps were then used to calculate each step duration. The entire new artifact data extraction was done in the log parser application, additional database fetching was not needed, because all the needed information was present in logs. A log parser application script was written to extract the new artifact account processing step information. An example of the script is displayed in Figure 16.

[1] All of the dates are in year 2022.

untitled.sx | autoinvest_analysis.sx

```
@only_timestamps=
SELECT
start_ts as start_ts
, finish_ts as finish_ts
, ts as customer_start_ts
, customer_finish_ts as customer_finish_ts
, validation_start_ts as validation_start_ts
, validation_finish_ts as validation_finish_ts
, balance_start_ts as balance_start_ts
, balance_finish_ts as balance_finish_ts
, allocation_start_ts as allocation_start_ts
, allocation_finish_ts as allocation_finish_ts
, fx_start_ts as fx_start_ts
, fx_prepare_ts as fx_prepare_ts
, fx_finish_ts as fx_finish_ts
, fo_create_start_ts as fo_create_start_ts
, fo_create_finish_ts as fo_create_finish_ts
, fo_save_start_ts as fo_save_start_ts
, fo_save_finish_ts as fo_save_finish_ts
, reservation_start_ts as reservation_start_ts
, reservation_finish_ts as reservation_finish_ts
, total_time_ms as total_time_ms
, request_id as request_id
FROM @accounts_with_balance as sub_result
JOIN @customer_start as customer_start ON request_id = customer_start.request_id
JOIN @customer_finish as customer_finish ON request_id = customer_finish.request_id
JOIN @validation_start as validation_start ON request_id = validation_start.request_id
JOIN @validation_finish as validation_finish ON request_id = validation_finish.request_id
JOIN @balance_start as balance_start ON request_id = balance_start.request_id
```

Figure 16. The log extraction script to extract step timestamps

The result of the log parser application script is a table of information. It consists of columns which represent sub steps in the account processing step.

The following sub steps were analysed:

- Customer preparation.
- Customer and account validation.
- Balance and its validation.
- Allocation preparation.
- Currency conversion preparation.
- Currency conversion.
- Fractional order creation.
- Fractional order storage.
- Reservation.
- Conclusion.

	A	B	C	D	E	F	G	H	I	J	K	L
1	customer_	validation_tim	balance_time_m	allocation_tim	fx_prepare	fx_convers	fo_create_	fo_save_ti	reservatio	conclusion	fo_amoun	total_time_ms
11	3	7	32	41	0	0	25	43	193	26	3	370
12	2	5	31	36	0	0	117	25	498	21	6	736
13	10	20	35	262	1	1472	53	32	341	25	6	2252
14	2	8	25	18	0	0	44	20	310	25	5	452
15	3	5	30	6	0	1	8	8	75	45	1	181
16	1	10	26	35	0	0	28	20	222	27	3	371
17	2	6	38	9	0	0	20	26	101	24	2	226
18	3	28	25	46	1	0	52	38	328	42	5	564
19	3	13	28	17	0	0	36	17	169	23	3	307
20	2	6	40	18	1	0	34	13	279	35	4	428
21	9	17	21	55	0	0	105	60	309	77	2	654
22	2	4	27	10	1	0	28	29	195	30	3	326
23	7	13	19	5	0	0	21	8	61	24	1	158
24	2	5	19	17	0	0	43	15	1400	26	6	1527
25	3	10	20	32	3	1128	51	16	360	23	6	1646
26	2	6	22	11	0	0	63	24	204	25	3	357
27	1	4	30	15	0	0	23	11	218	23	3	325
28	10	14	32	234	0	1341	80	20	571	30	6	2332
29	4	5	19	39	1	1348	36	15	263	25	4	1755
30	10	22	24	120	0	0	99	69	1224	22	3	1590
31	2	5	21	15	0	0	26	13	269	44	3	395

Figure 17. An example of the log parsing script execution result

The account processing step was analysed from the 1st of March to 19 of April of 2023. By that time, all the accounts were migrated to the new artifact. One example of the extracted data is presented in Figure 17.

In Table 2, 19th of April 2023 account processing steps timestamps are displayed. There can be seen that currency conversion and reservation steps take the longest. Those findings are consistent in all events. That is the reason why in the result table in Table 3, only those account processing steps are displayed and not all of them.

Table 2. 19th of April 2023 account processing steps timestamps

Step	Median time in milliseconds	Average time in milliseconds
Customer preparation	2	3.07
Customer and account validation	4	6.95
Balance and its validation	28	31.52
Allocation preparation	14	18.72
Currency conversion preparation	0	0.19
Currency conversion	6873	6367.92
Fractional order creation	31	37.66
Fractional order storage	12	15.89
Reservation	236	251.79
Conclusion	18	20.63

Table 3. The new artifact data[2]

	01.03	**08.03**	**15.03**	**22.03**	**29.03**	**05.04**	**12.04**	**19.04**
Number of accounts	51 948	52 069	52 193	52 263	52 381	52 483	52 645	52 746
Number of accounts with money	13 771	15 085	17 518	17 131	14 357	21 509	18 374	14 973
Number of orders	49 590	59 944	62 388	61 755	52 407	76 666	65 828	54 545
Number of currency conversions	1929	2577	2582	7829	2559	13 670	4077	2795
Processing time in seconds	621	682	743	1898	749	5369	1616	1093
Median account processing time in milliseconds	393	400	350	552	372	6073	405	399
Average account processing time in milliseconds	916	862	954	4026	1124	7169	2073	1614
Median currency conversions processing time per account in milliseconds	1753	1995	2944	6679	3285	7681	7503	6873
Average currency conversions processing time per account in milliseconds	3460	2695	3851	8000	4036	10 193	7365	6367
Median reservations processing time per account in milliseconds	219	239	216	229	224	232	232	236
Average currency conversions processing time per account in milliseconds	227	253	228	246	236	247	249	251

4.6.2.2 Data analysis findings

The old artifact was able to process one account around 250-300 milliseconds. It is 3-4 times faster than the new artifact processing time, which was around 900 milliseconds, when the

[2] All of the dates are in year 2023.

number of currency conversion stayed around 2000-3000. Those findings cannot be directly compared, because the old artifact processed accounts in a for-loop which means that one account is processed at a time and the old process cannot move forward with another account as long it is processing the previous account. That is not relevant to the new artifact, because the new artifact adopted an event-drive approach and can process many accounts at a time. At the moment, the number of consumers is 30 and that means 30 accounts are processed in parallel at the same time. Doing it this way, the total processing time is shorter and eliminates the main problem of the old artifact which was the accounts processing step duration. Another side note is that new artifact one account median processing time is 2-3 times faster than its equivalent average processing time. This can be explained by the bottlenecks in the process which become worse when the system load increases.

When both artifacts' total account processing time is compared with each other, then it can be concluded that when the number of accounts with fractional orders is about the same, then the new artifact is able to process the accounts 5-6 times faster.

After analysing the new artifact account processing data, two potential bottlenecks were found: currency conversion and reservation steps. Currency conversion step finds what currencies and how much is needed to carry on with the process. If a currency is missing or its value is not enough, then a currency conversion is performed using a redundant currency present on the account. The conversion is done using an external service in the Foreign Exchange microservice. Those HTTP requests take a lot of time, because an external service provider cannot service the requests fast enough. The median processing time increasing is noticeable if the number of currency conversions is higher than usual. For example, on a normal week the total number of conversions is around 2000 to 3000, but once in a quarter, the number rises to around 13 000, because of the dividends paid by ETFs. When that happens, then the processing time increases, and the total processing times takes a lot longer. The second bottleneck is the currency reservation step which reserves money for each fractional orders, so customers can not spend their money which is allocated to their orders. That service uses the legacy database which has pessimistic locking and long query times. The reservation is done using an external service in the Accounts microservice.

5 Discussion

The design-science research guidelines were used to achieve the research objective, formulated as RQ: *How to migrate a part of the legacy monolith application to a microservice?*.

The research begins with understanding the old artifact and its limitations. This thesis focuses on a legacy system and its process which does not perform as needed, because the number of accounts in the process has increased significantly and the total time to process all the accounts in not acceptable anymore. To overcome this limitation, the solution was a total rewrite of the existing process and separate it from the legacy system.

The next step was requirements elicitation and specification. Informal interviews were conducted by a product owner with different stakeholders and end-users to come up with functional and non-functional requirements. The old process was used as an example to drive the requirement elicitation, because the old process contained all the necessary steps to offer automatic fractional trading to customers. The most important requirement was that the new artifact has to be able to process 100 000 accounts in under 30 minutes. That ruled out any linear processing flows because that would have meant the one account has to be processed in 18 milliseconds. That is not possible by LHV IT infrastructure. The only choice was to figure out how to process the account in parallel. LHV IT-department uses a message broker, and it offers producing events to a queue and consuming it by one or many consumers. To achieve the main requirement set by the product owner, this seemed the only way how to do it.

All of the technological choices were driven by the LHV limitation that there exists some technologies which have become standard at the company and can be used without extra validation or approval process. That made the technological choices easy to choose. There choices were Java and Spring Boot framework for back-end, Angular for front-end, Microsoft SQL for database, and ActiveMQ for message broker.

The new artifact also improved different qualitative parameters of the product. The new artifact uses microservices architecture which is easier to maintain and scale. It also makes use of message broker features to process all the accounts in parallel. The new artifact is decoupled from other services in the domain so the services do not impact each other and can be developed independently. Another qualitative benefit is the improved transparency: the new artifact has decent application logging to track the process. Documentation and knowledge transfer was achieved during the new artifact development because its features and functionality was thoroughly analysed and documented.

There were different approaches how to migrate accounts from the old artifact to the new artifact. The best risk mitigated solution was to slowly increase the number of accounts in the new artifact and analyse each increase. This allowed to validate the system load and artifact. Another plus of this approach was that if an exception happed, then the fault can be discovered sooner, and the consequences can be kept to a minimum.

The old artifact used database procedures to convert currencies and reserve money. Both of those were replaced with requests to external microservices. Those changes helped to highlight a fact that external systems can become bottlenecks in a process. On average, the requests to Accounts application last around 200-300 milliseconds, but the main problem area is the Foreign Exchange application which cannot tolerate the load originating from the new artifact. Because the new artifact processes the accounts in 30 message broker consumers in parallel, this creates a high load on the Foreign Exchange system which they cannot handle so many requests at a time. The third-party API used by the Foreign Exchange system to convert currencies is not meant to be used in parallel and each request is processed slowly,

and the response times can last up to 30 seconds. This creates a noticeable delay in the overall process and the total processing times of the accounts increases.

It can be said that the new artifact satisfies the requirements specified by the product owner. The only major drawback is the long duration of requests to the Foreign Exchange application, but this fault is not caused by the new artifact and is only be tolerated as long as the number of currency conversions stays around 2000-3000. If the number is bigger than the average response time grows significantly and can grow up to 30 seconds, which is not acceptable and causes delays in the process. It was not possible to predict this problem, because the new artifact was the first to use that Foreign Exchange service.

Here are some learnings and suggestions based on this thesis for similar software development projects:

- Understand the existing infrastructure and its limitations.
- Incremental migration.
- Benefits of application logging.
- Unexpected bottlenecks.

The idea behind understanding the existing infrastructure and its limitations is that for the new artifact development, the technological choices were driven by the technologies present in the LHV IT-infrastructure. Different technologies have different features and enable to use logics which may not be present in some other technology. For example, while processing the accounts, ActiveMQ does not have an option to automatically start a new programming logic when all the events in a queue are successfully consumed. Because of that limitation, an additional account process tracking logic was added to the process.

Incremental migration is a suggestion to minimize potential problem escalation and keep aftermath as small as possible. This is especially crucial in production environment where customer money and data can be affected. Thanks to the incremental migration, it was possible to analyse each migration step and its impact on the systems. If a programming exception happened, it was possible to detect it sooner and keep the aftermath under control.

Application logging is a useful way to add transparency to the process. Is also offers better debugging information to find a mistake in the programming logic. Application happy-path logging is also beneficial to track that all the steps were performed to guarantee meeting the business requirements.

The bottlenecks caused by external services highlighted the importance of thoroughly evaluating and optimizing all external services before integrating them into a microservices architecture. Failure to do so can result in performance issues and slow down the entire system. Despite the unexpected bottlenecks caused by external services, the migration to a microservices architecture ultimately resulted in a more scalable and flexible system. However, it was important to address these bottlenecks to ensure that the benefits of the microservices architecture could be fully realized.

5.1 Limitations

Firstly, there is the contextual limitation. The effectiveness of this new artifact is limited to the specific context which the artifact was developed for. If a microservice is developed for one specific goal, then it may not be suitable for use in other systems due to difference in system requirements and business needs.

Another limitation was the available technologies to develop the new artifact. That limitation is relevant because LHV has pre-approved technologies which can be used to develop

software. Technologies influence the software application design because different technologies offer different feature. At the moment, those technologies were available to develop the new artifact and that is the reason they were chosen.

The old artifact missed application logging and any metrics to analyse its process. That made the old artifact data extraction difficult and limited, and the extracted logging information is not as good as the information from the new artifact. The consequence is it is not possible to analyse the old artifact as thoroughly as the new one, for example, each account processing sub step can be individually analysed in the new artifact, but that is not possible in the old artifact because there is no data on that. The only logging information present was the HTTP logs which were used to calculate the total processing time of all accounts. Because that information was not enough to evaluate the new artifact, additional database queries were done to fetch the total number of accounts and fractional orders.

6 Conclusion

This thesis aimed to migrate a part of the legacy monolith application to a microservice. A design-science research methodology was employed, and the new artifact was designed and developed considering the requirements from a product owner.

Firstly, to design and develop the new artifact, the software development process started with the problem statement to understand the old artifact and its shortcomings. The old artifact was not able to handle the increasing number of accounts in the process and the total processing time kept rising. Secondly, requirements elicitation and specification were done to describe the new artifact so it satisfies the set requirements and can handle the system load and scale up if needed. Thirdly, the technologies were chosen, and the new artifact was designed and developed. Finally, the evaluation results demonstrated that the new artifact achieved the primary goal of improving performance and scalability compared to the old monolithic legacy artifact. The secondary goals of parallel processing, decoupled services, transparency, and documentation & knowledge transfer were also achieved. However, some limitations were identified.

In conclusion, this thesis proves that design-science research methodology is an acceptable approach to address complex software development problem and can lead to the development of artifacts that solve real-life needs. The new artifact developed in this study can serve as a model for future research and software development. The artifact can be applied to other contexts with similar requirements, and further research could be conducted to address the identified limitations and refine the artifact.

For future work, there are several improvement opportunities. It is possible to scale the artifact vertically so the number of account processing message consumer is increased, and more accounts can be processed in parallel. Another option is to decrease the individual account processing duration so each account is processed more quickly. To achieve that, the artifact can be optimized internally and externally. At the moment, the internal optimization gain would be minimal, because the external bottlenecks are too limiting. The external services should solve their shortcomings so that the artifact internal optimization benefit would be noticeable.

At the moment, the main focus was to speed up the account processing and allow process to scale up if needed. Because of that focus, all the resources were allocated to make that part of the process as good as possible. The consequence of that focus is that there are other process steps which can be improved. In the finalization step for example, the fractional orders are processed one at a time and that may become a bottleneck in the future.

References

[1] J.D. Power. U.S. Retail Banks Nail Transition to Digital during Pandemic, J.D. Power Finds. https://www.jdpower.com/sites/default/files/file/2021-04/2021037%20U.S.%20Retail%20Banking%20Satisfaction.pdf (22.04.2023)

[2] Oliver S. Qonto Blog. The perks of modern banking. https://qonto.com/en/blog/business-management/banking/traditional-vs-modern-banking (23.04.2023)

[3] Deutsche Bank. How AI is changing banking. https://www.db.com/what-next/digital-disruption/better-than-humans/how-artificial-intelligence-is-changing-banking (22.04.2023)

[4] LHV. Ettevõttest. https://www.lhv.ee/et/ettevottest(23.04.2023)

[5] Lõhmus R. Finantsportaal. Täringumäng ja kasvukonto. https://fp.lhv.ee/news/newsView?locale=et&newsId=3969607 (23.04.2023)

[6] Pikkel S. Finantsportaal. Kasvukonto - pikaajaline investeerimine. https://fp.lhv.ee/forum/invest/130191?postId=3027951 (22.04.2023)

[7] Lewis J., Fowler M.. Microservices. www.martinfowler.com/articles/microservices.html (30.04.2023)

[8] Chen L. Microservices: Architecting for Continuous Delivery and DevOps. IEEE International Conference on Software Architecture (ICSA 2018), pp. 1-8

[9] Richardson C. Microservice Architecture. Pattern: Microservice Architecture. https://microservices.io/patterns/microservices.html (30.04.2023)

[10] Harris C. Atlassian. Microservices vs. monolithic architecture. https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (23.04.2023)

[11] Morningstar. LHV Group AS. https://www.morningstar.com/stocks/xtal/lhv1t/quote (23.04.2023)

[12] LHV. Kasvukonto. https://www.lhv.ee/et/kasvukonto (22.04.2023)

[13] Tõnisson R. Finantsportaal. Kuhu investeeriti LHV Kasvukontoga IV kvartalis 2022?. https://fp.lhv.ee/news/newsView?newsId=5649492 (22.04.2023)

[14] Pensionikeskus. Pensioni investeerimiskonto (PIK). https://www.pensionikeskus.ee/ii-sammas/pensioni-investeerimiskonto-pik (22.04.2023)

[15] LHV. LHV PIK ehk pensioni investeerimiskonto. https://www.lhv.ee/et/pik (23.04.2023)

[16] Tõnisson R. LHV PIKi kasutajad ootavad turgude languse jätkumist. https://fp.lhv.ee/news/newsView?locale=et&newsId=5643863 (22.04.2023)

[17] Nasdaq. Omnibus account. https://www.investopedia.com/terms/o/omnibusaccount.asp (23.04.2023)

[18] Chen J. Investopedia. What Is an Omnibus Account? How It's Managed. https://www.nasdaq.com/glossary/o/omnibus-account (23.04.2023)

[19] Majaski C. Investopedia. Fractional Share: Definition, Examples, How To Buy & Sell. https://www.investopedia.com/terms/f/fractionalshare.asp (02.05.2023)

[20] Marquit M. How Do Fractional Shares Work?. https://www.forbes.com/advisor/investing/fractional-shares/ (02.05.2023)

[21] Gregor, S., and Hevner, A. R. Positioning and presenting design science research for maximum impact. MIS quarterly (2013), 337–355.

[22] Boehm B., A View of 20th and 21st Century Software Engineering, *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 12-29.

[23] Kniberg, H., Skarin, M., Kanban and Scrum: Making the Most of Both. United States of America: C4Media. 2010.

[24] Java. What is Java technology and why do I need it?. https://www.java.com/en/download/help/whatis_java.html (23.04.2023)
[25] Java Documentation. Processes and Threads. https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html (23.04.2023)
[26] Spring Framework Reference Documentation. Part I. Overview of Spring Framework. https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/spring-introduction.html (23.04.2023)
[27] Spring Framework Reference Documentation. 21. JMS (Java Message Service) Part VI. Integration. https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/jms.html (23.04.2023)
[28] Angular. What is Angular?. https://angular.io/guide/what-is-angular (23.04.2023)
[29] Angular. Angular Features. https://angular.io/features (23.04.2023)
[30] Microsoft. Introducing SQL Server 2019. https://www.microsoft.com/en-us/sql-server/sql-server-2019 (23.04.2023)
[31] Apache ActiveMQ. ActiveMQ Classic. https://activemq.apache.org/components/classic/ (22.04.2023)
[32] Committee on Payment and Settlement Systems A glossary of terms used in payments and settlement systems (2003), 47–48.
[33] Section. Scaling Horizontally vs. Scaling Vertically. https://www.section.io/blog/scaling-horizontally-vs-vertically/ (29.04.2023)

I. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Kert Männik**,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

From Legacy to Microservices: A Case Study on Automatic Investment at LHV, supervised by Fredrik Payman Milani,

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in points 1 and 2.
4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kert Männik

05.05.2023