VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING

Information Technologies 4th year Final Bachelor Thesis

Development of an Online Auction Platform

Internetinės aukcionų platformos kūrimas

Done by:
Andrej Jegorov signature

Supervisor:
dr. Joana Katina

Vilnius
2024

Contents

Preface

I would like to express my gratitude to those who contributed to my passion for back-end development. Special thanks go to Bohdan Borysej, who showed me the enthralling nature of back-end development. I also appreciate Gediminas Rimša for providing additional lectures to help me gain hands-on experience in applying good software development practices. I would like to express my gratitude to my friends and coursemates who supported me during difficult times. Finally, I would like to express my gratitude to Vilnius University for providing me with high-quality education, particularly as part of the 'Problem-Based Learning' branch, and for the unforgettable experience I had as an exchange student. This opportunity allowed me to gain new knowledge that was previously unknown to me, which is highly advantageous for my professional development.

Abstract

Nowadays, collaboration with other people, platforms and organisations is one of the most reliable way to survive on the market. Connecting users with each other is the thing that benefited both users and companies. Thus the goal of this project is to create an online auction platform that will allow people to auction their items and enable easy collaboration with other platforms. This is estimated to achieve by making system as modular as possible with help of microservice architecture design pattern and REST API, which stands for *Representational State Transfer Application Programming Interface* architectural style.

Keywords: microservices, online commerce, web application

Santrauka

Internetinės aukcionų platformos kūrimas

Šio projekto tikslas yra sukurti modulinę internetinę aukcionų sistemą, kurios pagrindinė idėja yra suteikti galimybę naudotojams pirkti ir parduoti prekes tiek aukciono, tiek įprasto pardavimo būdu. Siekiant atsižvelgti į jau egzistuojančius užsienio sprendimus ir pirminius reikalavimus, numatoma, kad sistema turės papildomų funkcionalumų. Šie papildomi funkcionalumai leis naudotojams panaudoti internetinę aukcionų platformą kaip elektroninę parduotuvę, prijungti savo grafinę sąsają ir integruoti į jau esamus projektus.

Po panašių sistemų analizės išaiškėjo, kad Lietuvoje neegzistuoja platforma, kuri apjungtų skirtingų tipų aukcionus ir duotų savo naudotojams pardavimo būdo pasirinkimo laisvės (tai yra, ar prekė būtų parduodama aukcione arba kaip įprasta prekė e-parduotuvėje). Kita vertus, jau egzistuojančios gigantų platformos, tokios kaip Ebay arba Ebid, neturi lokalizacijos Lietuvai, o Vinted apsiriboja tik naudotais daiktais ir 7 kategorijomis.

Pagrindinis šio projekto dėmesys buvo skirtas detaliam sistemos projektavimui, išsiskaldymui į atskirus komponentus ir modeliavimui. Tai buvo padaryta tam, kad pasibaigus planavimo etapui būtų galima visas jėgas ir dėmesį skirti praktinės dalies įgyvendinimui be vėlesnių grįžimų prie tokių fundamentalių dalykų, kaip pasirinktos technologijos, esybių-ryšių modelis (*angl. entity-relationship model*) ir jų keitimo jau pradėjus programinį platformos įgyvendinimą.

Siekiant padidinti įvairių sistemos elementų savarankiškumą ir nepriklausomybę, jie kruopščiai sukurti kaip atskiri komponentai. Šis projektavimo metodas užtikrina, kad kiekvienas komponentas veiktų nepriklausomai ir būtų kuo mažiau priklausomas nuo kitų sistemos elementų. Sukūrus savarankiškus komponentus, sistema tampa labiau modulinė, todėl ją lengviau prižiūrėti, plėsti ir atskirus komponentus galima atnaujinti ar pakeisti be didelės įtakos bendram sistemos funkcionalumui. Ši projektavimo strategija skatina lankstumą ir palengvina vientisą įvairių modulių integraciją, taip prisidedant prie patikimesnės ir lengviau pritaikomos sistemos architektūros.

Galutinis projekto rezultatas - išsami sistema, kuri palengvina naudotojų registraciją ir autentiškumo nustatymą, taip pat sklandų daiktų ir aukcionų skelbimą ir kūrimą platformoje. Nors tam tikriems veiksmams atlikti reikia naudoti HTTP užklausas, kad būtų galima programiškai sąveikauti su sistema, pagrindinis sistemos bruožas yra patogi interneto sąsaja, kuri suteikia naudotojams patogų ir intuityvų būdą pasiekti platformos funkcijas ir jomis naudotis. Šis hibridinis metodas užtikrina, kad tiek techniški naudotojai, tiek mažiau su HTTP užklausomis susipažinę asmenys galėtų visapusiškai naudotis sistema, taip padidinant jos prieinamumą ir patogumą įvairiems naudotojams.

Introduction

The trend of digitizing commerce has become predominant, particularly during the global Covid-19 pandemic. This trend is expected to continue in the coming years. Online auctions are a significant factor in the growing popularity of the electronic marketplace. According to a report from eBay.com, the revenue for the third quarter of 2022 was $2.4 billion. The global active buyer base was 135 million, and the seller community included 20 million users worldwide. The increasing accessibility of the Internet is contributing to the growth of online consumerism.

In Lithuania, there is currently no online platform that combines auctions from different niches, allowing users to create their own auctions for selling personal items without being limited by platform-imposed categories. To address this issue, an initiative has been launched to develop a system that meets individuals' needs to quickly sell items at a fixed price or organize auctions to get the highest bids possible. An additional cardinal aim is to ensure the system's seamless interoperability with extant systems.

The aim of this paper is to describe the design of the system and to rationalise the logical solutions developed. This is achieved through a comparative analysis of similar systems, a review of existing solutions and recommendations in the scholarly literature, and the articulation of functional requirements. The investigation includes a review of systems similar to the proposed one, taking into account the insights and recommendations provided by academics from Griffith University [4]. The system's architecture follows the standards of the Unified Modeling Language (UML), and the database schema is created using the Entity Relationship Model (ER Modeling). Moreover, a meticulous selection of optimal tools and technologies has been conducted.

The paper is structured as follows: Section 1 provides an analysis of systems similar to the proposed one, while section 2 explains the system requirements and proposed solutions. Section 3 details insights related to system development, such as tool and technology selection or database schema. The findings are discussed in the section 4.3. Finally, the paper outlines possible avenues for future research in the section 4.3.

The impetus behind the decision to embark upon the development of an online auction application stems from the functionality, which offers an opportunity to gain in-depth knowledge of the intricacies inherent in modern website development and the array of tools employed for this purpose. Furthermore, there is an intent to implement the microservice architecture design pattern, which is novel to my experience. This endeavor also presents an avenue to apply the knowledge accrued during my professional internship, with the aim of delving further into the practical application and utility of such architectural paradigms within the domain of software engineering. Considering all factors, the **work objective** is to develop an online auction platform that is scalable, easy to maintain, and composed of independent components. Changes made to one module should not affect the functionality of other modules. Having that in mind, the following **tasks** are set:

1. **Decomposition:** Analyse the functionality and identify specific modules or features that can be extracted as independent microservices. Define the API contract for these microservices and establish their boundaries.

2. **Single Responsibility:** Ensure that each module or component within the system has only one responsibility. Focus on a single, well-defined responsibility for each component, preparing it for migration into a microservice.

3. **Competitive Analysis** Conduct a comprehensive analysis of existing systems with similar features, focusing on their architectures, functionalities, and best practices.

4. **Architecture Design** Create a detailed design plan for the system, including its architecture, key components, and functionality specifications, taking into account user requirements and industry standards.

5. **Implementation of Functionality** Programatically implement the system's designed functionalities, ensuring adherence to the specified architecture and feature set, and addressing any technical challenges that may arise during development.

1 Analysis of Similar Systems

1.1 eBay.com

Advantages:

- **Wide category choice** - 12 main categories with 22 subcategories form over a hundred categories to choose from.
- **Search by popular brands/manufacturer, etc.** - an user can see all related to specific manufacturer or brand in one place.
- **Personal recommendations** - after an user has spent some time on eBay, personal advertisements are displayed.
- **Advanced filtering options** - an user can operate in 10 different filter categories to search for more specific goods.
- **Online marketplaces, auctions and private sellers** - on eBay all these selling options can be found. User gets all possible purchase options - from individual trader up to online stores.
- **Seller information** - lot seller information is displayed next to the offer. Seller statistics such as name, positive feedback percentage and number of sold items are shown. To see even more details seller profile page is open to be checked.
- **Automatic bid system** - an user can set the maximum price he/she is ready to pay for an item. The system will automatically increment current offer step by step instead of automatically setting maximum price user had set before. Since the price can vary greatly, the increment step depends on the current price.

Disadvantages:

- **No localisation for Lithuania** - as eBay has no localisation for Lithuania, users from Lithuania are forced to search for the necessary goods among goods from all over the world. That demotivates Lithuanian sellers to use eBay platform in case they want to sell something locally without a need for international shipping.
- **Pay to post** - sellers must pay a fee for listing their items through the eBay system. The commission is taken regardless of the success of the lot.

General thoughts:

The eBay website serves as an excellent illustration of an internet auction system. With its user-friendly interface, the platform exemplifies how online marketplaces, auctions, and individual sellers can adapt to the contemporary era and coexist harmoniously.

1.2 aukcionas123.lt

Advantages:

- **Automatic bid system** - A user can establish the maximum price they are willing to pay for an item. Instead of automatically setting the maximum price the user had defined earlier, the system incrementally increases the current offer step by step. The increment step is determined based on the current price, recognizing the potential for significant price variations.
- **Trustworthy seller** - the auction system physically acquires an item before featuring it on the website. This ensures that the buyer can have confidence in receiving the purchased goods.
- **Market price** - item market price is displayed next to auction.

Disadvantages:

- **Poor choice** - as the system is designed only for Lithuania there are not so many people who want to put their goods into auction.
- **Poor search** - A user is limited to utilizing only keyword search and selecting from 16 categories to find goods, with no option to employ additional filters.
- **Long auction** - since all items are auctioned individually and sequentially, participants might find themselves obligated to remain for hours until the item of their interest comes up for bidding.

General thoughts:

While the system may appear dated, featuring the sale of all goods the auction host possesses one by one in real-time, it has the potential to appeal to individuals seeking an auction experience akin to what is often portrayed in films. The unique approach of selling items individually does result in a lengthier process, especially when dealing with a substantial quantity of goods, potentially numbering in the hundreds.

1.3 eBid.net

Advantages:

- **Automatic bids** - eBid will bid on users behalf by increasing user bid by the current bid increment up to specified maximum.
- **Seller information** - item seller information is displayed next to the offer. Statistics such as name, location and registration date are shown. To see even more details seller profile page is open to be checked.
- **Ask seller** - system provides an option to email seller to ask for more details regarding the listed item.
- **Wide choice** - as eBid is an international platform it provides 31 categories with their own subcategories.
- **Advanced search** - price range, location, condition as well as some other options can be applied to filter search results.

Disadvantages:

- **Lagging pictures** - in list of goods picture of an item usually is not displayed even if item has attached images.
- **No seller rating** - eBid system has no rating system for sellers. Moreover, account status (*silver/gold/platinum*) might be misleading for new system users.
- **Double commission** - the seller must purchase a account subscription in order to post offers. In addition to paying for the ability to sell, a commission is charged for each sale.

General thoughts:

eBid system has low fees for listings, a lot of users and is international, thus it is tempting choice for sellers. Buyers can conveniently search for items they are interested in using advanced search and variety of categories.

1.4 Vinted

Advantages:

- **Buyer Protection** - shoppers can shop confidently on the platform. If their item fails to arrive, arrives damaged, or significantly differs from the description, they can request a refund within 2 days from being notified by Vinted about the expected delivery or potential loss of the order. Failing to do so within the specified time frame or selecting the "Everything is OK" button will result in the completion of the order, and payment will be released to the seller automatically.
- **Guaranteed Payments** - The platform serves as a go-between for purchasers and vendors, promoting their transactions without them having to interact directly. This ensures that the money is transferred to the seller's account as soon as the item has been delivered to the buyer undamaged and does not differ significantly from the description provided.
- **Prepaid Vinted-Generated Shipping Label** - Vinted has seamless integration with three different shipping providers. Upon selection of one such provider by the buyer, a generated shipping label is automatically received by the seller. This label needs only to be affixed onto the parcel. Then parcel is ready to be sent.
- **Collaboration Wiht Poland** - Vinted is available not only in Lithuania, but in Poland as well.

Disadvantages:

- **Restricted Commercial Activity** - Restricting commercial activity on such platforms may prevent small businesses or individual entrepreneurs from expanding their reach, which might otherwise be possible. This move could discourage capable sellers who can offer diverse and inimitable products, thereby, declining the choices offered on the platform.

General thoughts:

Vinted is a well-designed and well-designed platform for the sale of second-hand goods. If the platform is solely utilized to vend pre-owned items, it is hard to see any disadvantages. However, as soon as one attempts to utilise it for business purposes, it appears impracticable due to the platform's regulations. Furthermore, interviews with active platform users indicate a lack of complaints, suggesting that the system can be regarded as a positive example.

1.5 AliExpress

Advantages:

- **Wide Category Selection** - AliExpress has wide category selection with each having own sub-categories.
- **Flexible Item Description** - Item description is flexible, so every seller can describe an item for sale as they want.
- **Global Shipping** - AliExpress facilitates international shipping, making products accessible to a global customer base.
- **Flash Deals and Discounts** - Regular flash deals and discounts provide opportunities for significant savings on various products.

Disadvantages:

- **Shipping Times** - Long shipping times, particularly for standard or free shipping, can be a drawback, and express shipping options may be more expensive.
- **Description Mismatch** - Size mismatch is an example of description mismatch and a common issue when buying clothes.
- **Cluttered Design** - Welcoming page is cluttered with an quick deals, special offers and recommendations.

General thoughts:

AliExpress is a widely used online marketplace that connects buyers with sellers offering a wide range of products at different price points. The wide range of products, competitive pricing and buyer protection policy is frequently mentioned as a positive feature. On the negative side, some users may experience longer delivery times, potential quality concerns or difficulties in communicating with the sellers.

1.6 Comparison of Bid Item Elements And Seller Information

This section presents a detailed comparative analysis of how different online marketplaces present information about auction items and their sellers. This analysis is presented through two distinct tables, each of which dissects and contrasts the specifics of the data provided by these platforms.

Rating aspect	**eBay.com**	**aukcionas123.lt**	**eBid.net**
Bid types	Best offer / Auction / Buy now	Auction / Buy now	Auction / Buy now
Rating	5 star / None	-	-
Item title	+	+	+
Item price	+	+	+
Watchers	+	-	-
Quantity	Selected and available	1/1	1/1
Condition	+	+	+
Condition comment	+	-	-
Time left	+ / unrestricted	2 times/week	+ / unrestricted
No of bid items	+ / N/A	-	-
No of placed bids	+	+	+
Estimated delivery time	+	-	-
Shipping options	1+	2	1+
Shipping cost	+	+	+
Payment options	up to 4	4	up to 3
Add to favourites	+ (Watchlist)	-	+
Market price	-	+	-
Views	-	+	+

Table 1. Comparison of bid informational item elements

Table 1 presents the bid informational item elements available to potential bidders on eBay.com, aukcionas123.lt, and eBid.net. The table includes various rating aspects, such as bid types, item condition, shipping details, and payment options, among others. It also indicates the availability of features like watchers, quantity, and views, which can significantly influence a buyer's decision-making process.

Seller information point	eBay.com	eBid.net	Vinted	AliExpress
Name	+	+	+	+
Rating	Positive feedback percentage for past 12 months; Feedback rating badges	Difference between positive and negative comments	5-star rating by users with comments and automatic reviews	Based on 5-star rating system reviews from users
Date of Registration	+	+	-	+
Recent Activity	-	Date + time	Time since last activity	-
Location	Country level	City level	City level	Can be found in business license information
Self-Description	+	+	+	-
Listings Running	+ (list of active listings)	+ (amount)	+ (list of active listings)	+ (personal store page)
Stores Run by User	-	+	N/A	-
Followers	+	-	+	+
Number of sold items	+	-	-	+

Table 2. Comparison of seller information

These tables offer an overview of the user experience in online marketplaces, assessing the accessibility and transparency of information. This affects both buying and selling behaviours.

1.7 Summary

Common features among the mentioned auction systems include automatically generated bids and information on sellers. In addition to the number of registered users, important features consist of advanced search functions, effective filtering options, and a wide variety of item categories. Buyers need to be mindful of the likelihood of non-delivery from the seller, which is a primary concern when acquiring goods online, including through auction sites. To reduce this risk, employing a seller rating system (*similar to that used on eBay.com or eBid.net*) or a platform being intermediary that facilitates transactions between buyers and sellers (*similar to Vinted*) could be effective solutions.

2 System Design

This chapter serves as a comprehensive blueprint for the envisioned software system, encapsulating the foundational aspects of its development journey. Within this chapter, the requirements and specifications of the system are meticulously defined, ensuring a clear understanding of the system's purpose and objectives. It delves into the intricacies of user needs, business goals, and functional requirements, establishing a solid foundation upon which the system will be built.

Furthermore, this chapter uses UML use-case and activity diagrams to provide a high-level understanding of the system's expected behaviour and interactions. These visual representations offer a structured view of how users will interact with the system and illustrate the dynamic processes and workflows that will be at the core of its operation. The chapter provides a comprehensive roadmap for translating concepts into a robust and functional software system.

2.1 Authentication Microservice Requirements

Functional Requirements

- An authentication microservice user should be able to register with the system so that their credentials can be used for authentication purposes.
- An authentication microservice user should be able to log in to the system to obtain a token to access other microservices.

Non-Functional Requirements

- The microservice needs a dedicated database to store user credentials, which will only be used by the authentication microservice.
- Password encryption should always be used to ensure that sensitive user data, such as passwords, is not stored in plain text in the database.
- The microservice should not be exposed to the external network. If it needs to communicate with other services, it must use a secure protocol such as HTTPS.
- Requests must be handled within 2 seconds or less, provided a stable internet connection with a download and upload speed of 300 Mbps.
- The microservice should be compatible with and run on Vilnius University's infrastructure.

2.2 Use Cases and Interaction Flow of Authentication Microservice

This section presents a thorough overview of the main functionalities and interaction flows of the Authentication Microservice. The microservice's primary operations and scenarios, including user registration, login, token generation, validation, and token refresh, are elucidated through the use of a UML use-case diagram. Additionally, this section includes two UML activity diagrams that clearly illustrate the registration and login processes. The diagrams, along with the detailed explanation, provide a comprehensive understanding of how the Authentication Microservice functions to ensure secure and seamless user authentication and authorization within the microservices architecture.

2.2.1 Use Cases

This UML use-case diagram illustrates the interactions between users and the microservice. It outlines the key functionalities of user registration, login, token generation and validation, and token refresh. Each use case represents a specific aspect of user interaction, providing a clear and concise visual representation of the microservice's role in securely managing user access and authentication within a broader system.

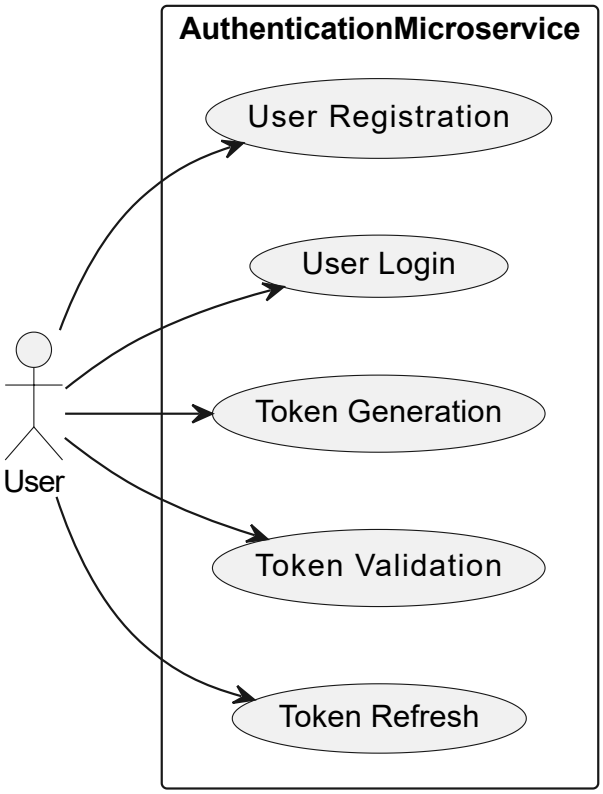

Figure 1. Interaction with Authentication microservice use cases

2.2.2 Interaction Flow

Registration is the first step in the Authentication microservice. It allows users to create accounts and establish their identity within the system. The UML activity diagram 2 illustrates the sequential activities, decision points, and interactions involved in this process. It includes actions such as user input of registration data, validation, and storage of credentials. This diagram elucidates the registration process, establishing it as a fundamental component in constructing a secure and user-friendly authentication system within the microservices framework.

Login feature is a crucial aspect of the Authentication microservice, acting as the gateway for users to securely access other microservices. The figure 3 outlines the steps involved in this operation, including user credential validation, password verification, and the generation of JWT tokens upon successful authentication. This diagram illustrates the process by which users obtain authenticated access, highlighting the crucial role of the Authentication microservice in guaranteeing secure and controlled access to other microservices.

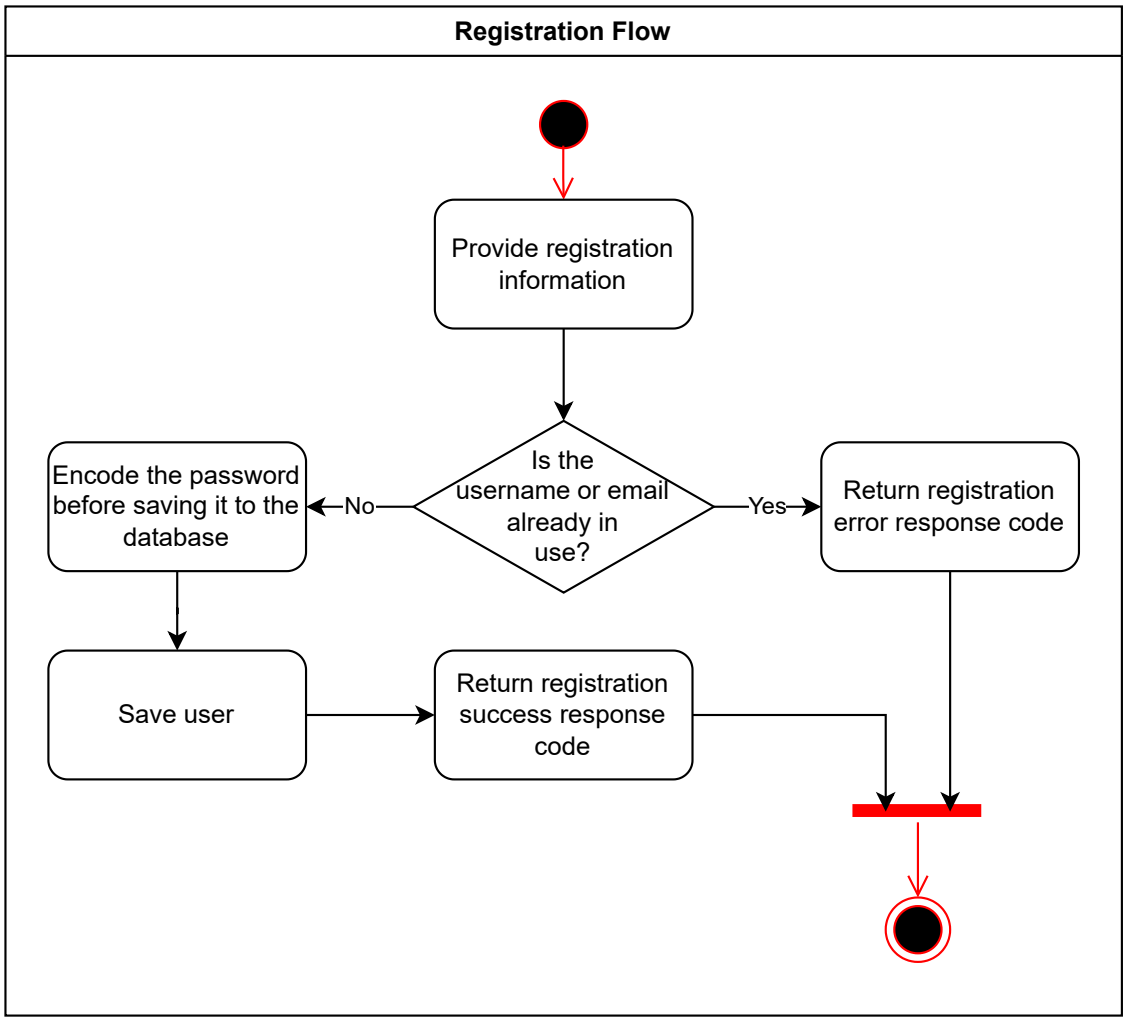

Figure 2. UML activity diagram representing registration flow

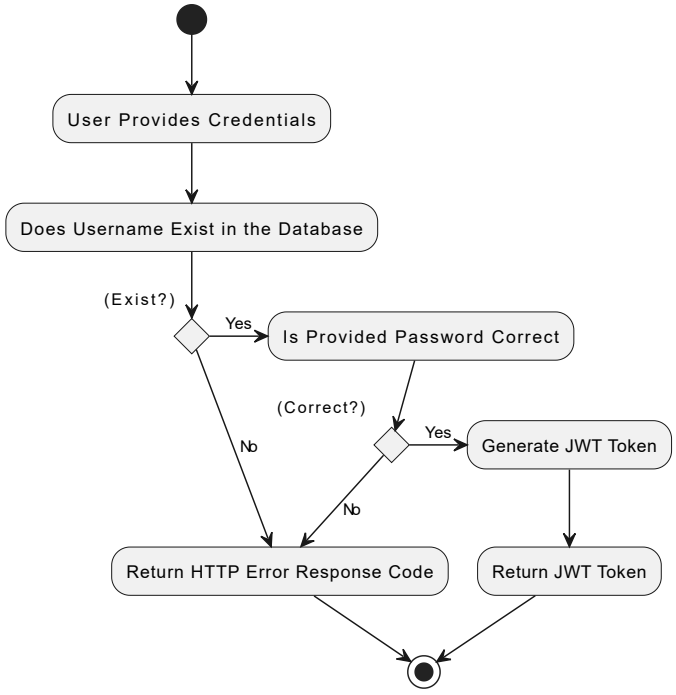

Figure 3. UML activity diagram representing login flow

2.3 Product and Auction Listing Microservice Requirements

Functional Requirements

- **Browse Functionality:** The microservice shall enable users to browse through all current listings and auctions to identify potential items for purchase.
- **Item Description Access:** The microservice must provide users with detailed descriptions of items, enabling them to make informed purchase decisions.
- **Auction Viewing Capability:** The microservice shall allow users to view all ongoing and upcoming auctions to assess the availability of items for bidding.
- **Auction Creation Feature:** The microservice shall provide users with the ability to create auctions for selling items within a set timeframe.
- **Auction Detail Editing:** The microservice shall allow users to edit the details of their auctions, including updating item descriptions.
- **Auction Cancellation Option:** The microservice must offer the functionality for users to cancel their auctions, thus retaining possession of the item.
- **Item Listing for Sale:** The microservice shall enable users to list items for sale, facilitating the opportunity for monetary transactions.
- **Item Detail Editing Function:** The microservice shall provide an option for users to edit the details of their listed items to ensure the provision of relevant and accurate information.
- **Item Sale Withdrawal:** The microservice must allow users to remove their items from sale, providing them with the discretion to retain the item if desired.
- **Detailed Item Description Viewing:** The microservice shall present users with detailed descriptions of items to support informed decision-making in the purchasing process.

Non-Functional Requirements

- The microservice uses a dedicated database to store item listings and auctions, which will only be accessed by the Product and Auction Listing microservice.
- The microservice uses authentication microservice described in section 2.1 to authenticate users and link them to items and auctions created by them.
- Requests must be handled within 2 seconds or less, provided a stable internet connection with a download and upload speed of 300 Mbps.
- The microservice should be compatible with and run on Vilnius University's infrastructure.

2.4 Use Cases of Product and Auction Listing Microservice

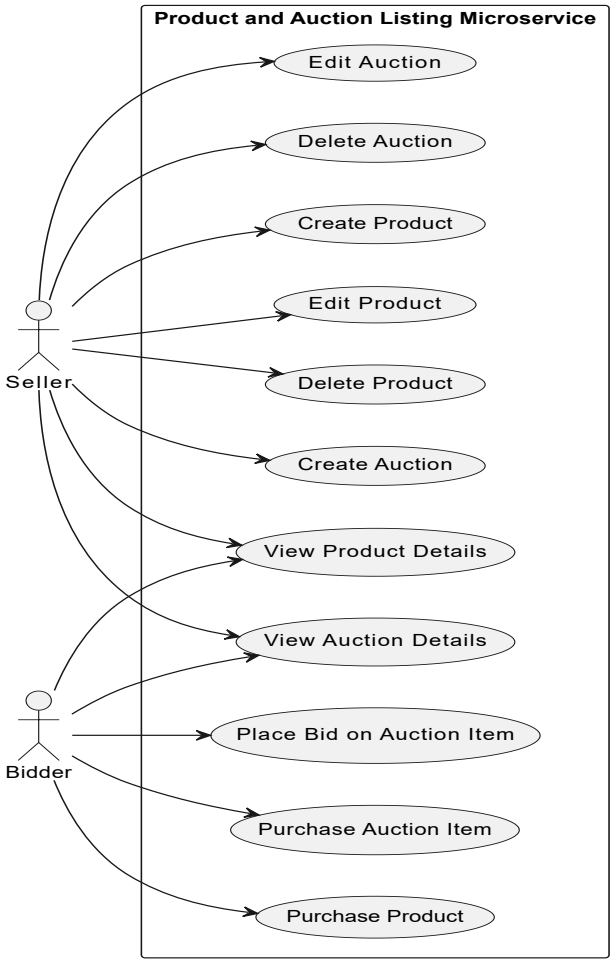

Figure 4. Interaction with Product and Auction Listing microservice use cases

The UML use case diagram 4 illustrates the interactions and functionalities within the "Product and Auction Listing Microservice". This microservice serves as a dynamic marketplace where sellers and bidders play different roles in a variety of activities.

Actors:

1. **Seller**: Represents users who create, manage and monitor product listings and auctions.
2. **Bidders**: Represents users who browse, buy products and participate in auctions by placing bids.

Use Cases:

- **Create Product:** Sellers can create new product listings by specifying details of items they wish to sell.
- **Edit Product:** Sellers have the ability to modify and update product listings, ensuring accurate and up-to-date information.
- **Delete Product:** Sellers can remove product listings that are no longer available or relevant.
- **Create Auction:** Sellers can create auctions and set parameters for bidding.
- **Edit Auction:** Sellers can make adjustments to ongoing auctions.
- **Delete Auction:** Sellers can close auctions early if necessary.
- **View Product Details:** Both sellers and bidders can access detailed information about the products listed.
- **View Auction Details:** Both sellers and bidders can view full details of current auctions.
- **Buy Product:** Bidders can purchase products listed for sale.
- **Place a bid on an auction item:** Bidders can participate in auctions by placing bids on auction items.

Interactions:

- Sellers initiate and manage product listings and auctions.
- Both sellers and bidders can view product and auction details.
- Bidders can buy products.
- Bidders can actively participate in auctions by placing bids.

Overall, use case diagram 4 encapsulates the essential functionality of the Product and Auction Listing Microservice within a dynamic marketplace, describing the roles and interactions of sellers and bidders as they engage with various product listings and auctions. It highlights the core activities and features that contribute to the functionality of this microservice and enhance the user experience for both sellers and buyers.

2.5 Front-end Microservice Requirements

Functional Requirements

- **Registration and Authorization Forms:** The system shall include forms to facilitate user registration and authorization, integrating with the Authentication microservice.
- **Product and Auction Creation Forms:** The system shall provide forms enabling users to create new product listings and auctions, interfacing with the Product and Auction Listing microservice.
- **Comprehensive Product and Auction Views:** The system shall offer views for users to browse all products and auctions, including detailed descriptions of each item.

Non-Functional Requirements

- **Performance Requirement for Page Load Time:** Any web page within the system must load within 2 seconds under the condition of a stable internet connection with a minimum download and upload speed of 300 Mbps. This requirement ensures efficient and timely access to the system's functionalities for users.
- **Availability and User Notification Requirement:** In the event of any microservice becoming inaccessible, the system must promptly notify the user of this unavailability. Furthermore, access to functionalities directly dependent on the inaccessible microservice should be restricted or disabled. This requirement is critical for maintaining transparency in service availability and ensuring a reliable user experience.
- **Compatibility Requirement with Vilnius University's Infrastructure:** The microservice must be fully compatible with, and function seamlessly on, Vilnius University's existing technological infrastructure. This requirement involves adhering to compatibility standards and system configurations specific to Vilnius University.
- **Database Usage and Access Requirement:** If a dedicated database is utilized by the microservice, it shall be exclusively accessed and manipulated by the microservice itself. This requirement is essential for maintaining data integrity, security, and ensuring that the database's operations are optimized for the microservice's specific needs.

3 System Development

Microservices architecture has become popular in modern system development due to its ability to break down complex applications into smaller, more manageable components. Each microservice is responsible for specific features or functionalities, making it easier to develop, deploy, and scale different parts of the system. However, the technology selection for each microservice may have both similarities and differences due to their unique purposes. This section discusses the concept of microservices within a system and emphasises the importance of using distinct technology stacks for each microservice.

3.1 Authentication Microservice

The Authentication Microservice is responsible for verifying user identity and controlling access within the system. Its main objective is to ensure the security and privacy of user data and interactions. This microservice manages user registration, login, and session management, implementing industry-standard authentication protocols such as OAuth 2.0 and JWT. The Authentication Microservice enforces user authentication and authorization to ensure that only authorized individuals can access protected resources and functionalities. This plays a crucial role in maintaining the system's integrity and trustworthiness by safeguarding user accounts and permissions.

3.1.1 Tools and Technologies

Java The selection of the Java programming language as the cornerstone of microservice development is substantiated by a constellation of compelling factors. Java's enviable reputation for platform independence, buttressed by its venerable "write once, run anywhere" paradigm, accords developers the ability to cultivate microservices capable of deployment across diverse platforms and operating systems. This attribute of portability is particularly advantageous within the microservices architectural paradigm, where services may traverse heterogeneous deployment environments.

Furthermore, Java's adherence to strong typing principles augments the software development process by facilitating the early detection of errors at compile-time, thereby mitigating the incidence of runtime anomalies. The presence of a prodigious ecosystem of libraries, frameworks, and tools further elevates Java's stature, offering developers an expansive toolkit that expedites microservice construction. Of notable significance is the Spring framework, exemplified by Spring Boot, which extends a superlative suite of utilities tailored to microservices development. Spring's venerable features, such as Dependency Injection (DI) and Inversion of Control (IoC), afford means of managing dependencies with alacrity, promoting loose coupling and enhancing the veracity of unit tests.

Finally, Java also has features that allow it to run efficiently on certain processors, further optimising the performance of Java programs. One such feature is Just-In-Time (JIT) compilation. Java source code is compiled into an intermediate form called bytecode, which is platform-independent. When a Java application is run, the bytecode is interpreted by the Java Virtual Machine (JVM). However, modern JVMs often use JIT compilation, which translates the bytecode into native machine code at runtime. This native machine code is optimised for the specific processor architecture on which the program is running, taking full advantage of the underlying hardware. This dynamic compilation approach allows Java programs to achieve impressive execution

speeds on different hardware architectures, while retaining the benefits of portability and platform independence.

Spring Boot framework is the most popular module of the Spring framework [5] that is uniquely tailored to microservices development, furnishing practitioners with a constellation of microservices-centric features, inclusive of embedded web servers, auto-configuration, and centralized configuration management. These capabilities imbue developers with the capacity to establish and administer application infrastructure with unparalleled expediency.

Moreover, Spring's intrinsic capacity for securing microservices is integral to its adoption in this context. Spring Security, a component thereof, provides robust mechanisms for authentication and authorization, engendering an environment where user credential management can be entrusted to the framework.

Complementing this technology stack is the PostgreSQL relational database. PostgreSQL is celebrated for its unwavering reliability and steadfast commitment to data integrity, making it an ideal choice for storing critical user credential data.

PostgreSQL In the field of database management, PostgreSQL strictly adheres to the principles of ACID (*Atomicity, Consistency, Isolation, Durability*) compliance. This ensures data consistency and reliability, even under conditions of high concurrency, which are common in microservices environments.

Furthermore, PostgreSQL's flexible architecture makes it suitable for scalability, allowing for strategies such as replication and sharding. Agility is crucial as it caters to the scalability demands associated with the proliferation of microservices.

PostgreSQL excels in the domain of user credential storage, providing a robust set of security features, including role-based access control, encryption, and authentication mechanisms. The user credential store is fortified by a range of protective measures, which instils confidence in the security of the microservices ecosystem.

The selection of the Java programming language, the Java Spring framework, and PostgreSQL for the development of microservices is substantiated by their collective robustness, versatility, and security. This strategic combination enables developers to create and maintain microservices with confidence, resulting in scalability, reliability, and ease of development.

Hibernate ORM is a widely adopted Object-Relational Mapping (ORM) framework for Java. It simplifies database interaction within applications by abstracting the complexities of translating Java objects into relational database tables and SQL queries. This significantly reduces the amount of boilerplate code required, making development more efficient and maintainable. Hibernate facilitates seamless data retrieval and persistence by mapping Java classes to database tables and Java fields to table columns.

Additionally, it promotes database portability by allowing database-agnostic code, enabling us to switch between different database management systems (DBMS) without extensive code modifications. This ensures flexibility and adaptability as our application evolves. Hibernate's automatic generation of SQL statements optimises database access, improving performance and reducing the risk of SQL injection attacks.

Additionally, Hibernate supports caching, which enhances application speed by storing frequently accessed data in memory. This feature minimises database round-trips and is especially

advantageous for applications with high read-to-write ratios. Overall, Hibernate ORM enables to concentrate on business logic rather than complex database interactions, resulting in more maintainable, efficient, and portable code and faster quality development.

JSON Web Tokens (JWT) have been chosen as a key component in the authentication strategy due to their robust security features and efficiency. JWT is a compact and self-contained method of securely transmitting information between parties as a JSON object. This choice aligns with modern authentication requirements, as JWTs provide a stateless and scalable mechanism for managing user identity and access control.

Several academic studies and research papers have highlighted the security and effectiveness of JWT in real-world applications. For instance, Bradley et al. [2] discuss the design principles and security considerations of JWT, demonstrating its suitability for secure data exchange. Furthermore, Jones et al. [1] and Lodderstedt et al. [3] explore OAuth 2.0 and OpenID Connect, which are two authentication and authorization frameworks closely related to JWT. They emphasize the pivotal roles of these frameworks in modern web security.

In practice, JWT provides a tamper-evident and verifiable method of token-based authentication, which enhances the trustworthiness of our authentication microservice. By embedding user claims within the token and signing it cryptographically, JSON Web Tokens (JWT) ensure data integrity and enable users to trust the information they carry. This aligns seamlessly with our commitment to secure and efficient user authentication.

io.jsonwebtoken library is an essential tool chosen for the development of the Authentication Microservice, primarily for its exceptional capabilities in managing JSON Web Tokens (JWTs). This library proves invaluable in simplifying the complex processes associated with JWT management.

The 'io.jsonwebtoken' library offers a user-friendly API that simplifies the creation, parsing, and signing of JWTs. This functionality reduces the development effort required to implement authentication and authorization mechanisms. Additionally, the library excels in extracting and verifying claims from JWTs, making user data easily accessible within the Authentication Microservice.

The decision to leverage the "io.jsonwebtoken" library is firmly grounded in its ability to enhance both the efficiency and security of the Authentication Microservice. By providing a convenient and reliable means of handling JWTs, the library empowers the microservice to enforce stringent authentication protocols, safeguarding user identities and access privileges effectively. Consequently, the adoption of this library serves as a prudent choice, strengthening the foundation of the Authentication Microservice's security mechanisms.

3.1.2 Communication With Other Microservices

The Authentication microservice uses JSON Web Tokens (JWTs) to securely exchange data with other microservices in the architecture. Each JWT contains essential information, such as user data, along with a cryptographic signature. When a user authenticates, the Authentication microservice generates a JWT and issues it to the user. Subsequently, the user's client includes the JWT in requests to other microservices. Upon receipt, the recipient microservice validates the token's signature, expiration, and issuer to ensure its authenticity. The JWT is then used to extract user data, enabling secure and authorized data exchange between microservices. This approach improves

security, scalability, and performance across the architecture while simplifying user authentication and authorization.

As there is no need to include all user known details as a part of the token, the "UserDetails" class described below on listing 1 is used to map required user data and store it inside signed JWT token.

```
class UserDetails {
    private long userId;
    private String username;
}
```

Listing 1. Structure of UserDetails class

The JWT's final structure is described in listing 2. The token is utilised to authenticate a user when they attempt to perform an action that requires confirmation of their authenticity or when their details, such as their ID or username, need to be used at a later time.

```
{
  "header": { "alg": "HS256", "typ": "JWT"},
  "payload": {
  // default claims (Issuer, Subject, Issued At, Expiration Time
      , Not Before, Token ID)
    "user": {
      "userId": 123456,
      "username": "exampleUsername"
    }
  },
  "signature": "HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    secretKey
  )"
}
```

Listing 2. Structure of issued JWT token to be used by other microservices

3.1.3 Database Schema

The implementation of a microservice architecture has facilitated the development of a database schema that is both small and simple. This architectural approach is modular and decentralized, enabling individual microservices to operate independently and focus on specific functionalities. As a result, each microservice requires only a minimal and self-contained database schema, leading to reduced complexity and improved maintainability.

Figure 5 shows that the database schema of the Authentication service is small and simple. This is achieved by moving complex relations required for other services' functionality to be implemented and managed by other themselves and potentially more suitable database management systems for the desired functionality.

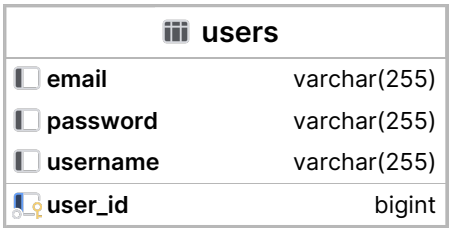

Figure 5. Database schema of the Authentication microservice

3.2 Product and Auction Listing Microservice

In the forthcoming section, the focus will be on the 'Product and Auction Listing Microservice,' a pivotal component within the framework of our microservices-based ecosystem. This microservice operates as a dynamic marketplace, affording users the seamless capability to list their items for sale and orchestrate captivating auction events. A comprehensive exploration of the tools and technologies employed in its implementation will ensue, coupled with an elucidation of the architectural decisions that underlie its core functionalities. The primary objective of this section is to impart a lucid comprehension of how the microservice facilitates users in showcasing their products, instigating auction processes, and fostering interactive engagements within our virtual marketplace, thereby fostering a vibrant community of both buyers and sellers.

3.2.1 Tools and Technologies

Java and Spring Boot The 'Product and Auction Listing Microservice' is implemented using Java and the Spring Boot framework. This decision is based on the reasons explained in previous sections, particularly sections 3.1.1 and 3.1.1. Java is known for its robustness, scalability, and versatility, while Spring Boot offers rapid development capabilities and extensive ecosystem support. These technologies enable the microservice to provide a robust and flexible platform, ensuring efficient product and auction listing functions while seamlessly integrating with our wider microservices-based architecture.

MongoDB The selection of MongoDB as the Database Management System (DBMS) for our platform was a strategic decision driven by its inherent flexibility and agility, particularly in catering to the diverse range of product listings within our marketplace. MongoDB, often heralded as a NoSQL database, breaks away from the rigid, tabular structures of traditional relational databases. Instead, it adopts a schema-less, document-oriented approach, allowing us to store and manage product data with unparalleled versatility.

One of MongoDB's standout features lies in its ability to effortlessly handle dynamic, ever-evolving data structures. In our dynamic marketplace, where product listings encompass a wide spectrum of attributes and characteristics, MongoDB's schema-less design is a clear advantage. This design allows us to adapt and extend product attributes dynamically as our marketplace grows, accommodating the vast array of choices available when describing items for sale. Unlike traditional databases, there's no need for extensive schema modifications or migrations, providing us with the agility to respond swiftly to changing market dynamics.

Furthermore, MongoDB's scalability and horizontal scaling capabilities align harmoniously with our vision for a highly responsive and scalable platform. As our user base and product listings expand, MongoDB's distributed architecture ensures that we can seamlessly accommodate increased data volumes and user interactions.

In conclusion, MongoDB was a natural choice for our DBMS, primarily due to its unparalleled flexibility in managing product listings with a myriad of attribute options. This strategic decision empowers us to build a dynamic and responsive marketplace that adapts effortlessly to the ever-changing preferences of our users, while ensuring scalability and robust performance as our platform continues to grow.

Spring Data is a crucial component of the Spring ecosystem. It was strategically chosen to improve our interaction with MongoDB due to its many advantages. Spring Data simplifies data access for various data stores, including MongoDB, by providing a higher-level, consistent programming model. This abstraction significantly reduces the complexity of data access code, facilitating cleaner and more maintainable data interactions.

For our MongoDB-based product and auction listings microservice, we found Spring Data MongoDB to be particularly beneficial due to its robust support for MongoDB's document-oriented data model. Spring Data MongoDB allows for effortless mapping between Java objects and MongoDB documents, enabling us to work with data in its natural JSON-like format. This fosters agility and clarity in our application code. Spring Data provides repository support that simplifies data access by offering a consistent API for common database operations, reducing development effort and potential sources of errors.

Additionally, Spring Data's integration with the broader Spring ecosystem aligns seamlessly with our microservices architecture, ensuring cohesive and efficient data interactions across our entire platform. The combination of Spring Data and MongoDB enables us to utilize the capabilities of a NoSQL database while preserving the consistency and dependability of the Spring framework. This ultimately leads to a scalable, high-performance, and developer-friendly solution for our product and auction listing microservice.

3.2.2 Database Model

MongoDB is a NoSQL database, meaning it does not follow a rigid, predefined database schema like traditional relational databases. Instead, it uses flexible database models to represent how data is stored. These models allow for dynamic and schema-less data, accommodating changes and evolving data structures, making it well-suited for applications with varying data requirements. Data is stored in BSON (Binary JSON) format. This format allows for documents within collections to have different structures, enabling developers to efficiently adapt and scale their data models to meet the needs of their applications.

This subsection outlines two distinct database data models: 'Product' and 'Auction.' These models define the composition and organization of data entities within a MongoDB NoSQL database. The 'Product' model serves as the foundational schema, while the 'Auction' model, an extension of 'Product,' introduces additional fields. Documenting these data models provides a comprehensive overview of the database's data structure, aiding in its effective utilization and management.

```
1    {
2  "_id": ObjectId("5f9ff31"), // MongoDB will automatically generate
       this unique identifier
```

```
3   "id": "product123",                          // String
4   "title": "Smartphone X",                     // String
5   "description": "High-quality smartphone",    // String
6   "condition": "NEW_WITH_TAGS",   // Enumeration "Conditions"
7   "price": 599.99,                // BigDecimal price
8   "optionalParameters": {
9     "parameter1": "value1",       // Map<String, Object>
          optionalParameters
10    "parameter2": 42,
11    "parameter3": true
12  },
13  "creator": {
14    "userId": 12345,          // UserDetails creator (inner class)
15    "username": "user123"
16  }
17 }
```

Listing 3. Structure of documents in 'Product' collection

The 'Product' data model, outlined in Listing 3, is a fundamental schema in the MongoDB NoSQL database. It includes essential attributes such as '**id**,' '**title**,' '**description**,' '**condition**,' '**price**,' '**optionalParameters**,' and '**creator**.' These fields define the structure and content of each 'Product' document stored in the associated MongoDB collection. The '**id**' field provides a unique identifier for each product, while '**title**' and '**description**' offer descriptive information. '**Condition**' represents the product's condition according to predefined categories, and '**price**' denotes its monetary value. The '**optionalParameters**' field allows for flexible and customizable additional data, and '**creator**' identifies the user responsible for creating the product entry. This data model is highly flexible and adaptable, making it ideal for managing and retrieving data efficiently. It is particularly well-suited for applications with diverse data requirements.

The '**Conditions**' enumeration, which includes values such as '**NEW_WITH_TAGS**', '**NEW_WITHOUT_TAGS**', '**VERY_GOOD**', '**GOOD**' and '**SATISFACTORY**', has been adopted from Vinted [6], a popular online marketplace. These values, originally used by Vinted, serve as standardised representations of product conditions and are used within the data model to categorise and describe the condition or quality of products. By using these values, the data model aligns with established industry standards, improving clarity and consistency in the representation of product conditions within the MongoDB database.

Finally, the 'UserDetails' inner class encapsulates essential information about the user associated with a product. It consists of two fields: 'userId', a unique identifier for the user, and 'username', the user's identifying name or handle. This inner class is an integral part of the 'product' data model, facilitating the association of products with their respective creators or owners within the MongoDB database.

```
1    {
2   "_id": ObjectId("5f9f01a902d71627dc8eef32"), // MongoDB will
       automatically generate this unique identifier
3   "id": "auction456",                        // (inherited from Product)
4   "title": "Artwork Auction",                 // (inherited from Product)
5   "description": "Original artwork", // (inherited from Product)
6   "condition": "VERY_GOOD",                   // (inherited from Product)
7   "price": 0,                                 // (inherited from Product)
8   "optionalParameters": {
```

```
 9     "parameter1": "value1",                    // (inherited from Product)
10     "parameter2": 42,
11     "parameter3": true
12   },
13   "creator": {
14     "userId": 56789,                           // (inherited from Product)
15     "username": "user456"
16   },
17   "startDate": ISODate("2024-01-15T08:00:00Z"), // Date (new field for
       Auction)
18   "endDate": ISODate("2024-01-20T20:00:00Z"),   // Date (new field for
       Auction)
19   "durationInMs": 432000000,                    // 5 days in milisecinds;
        long durationInMs (new field for Auction)
20   "instantPurchasePrice": 299.99                // BigDecimal (new field
       for Auction)
21 }
```

Listing 4. Structure of documents in 'Auction' collection

The Auction model extends the Product data model by introducing additional attributes specific to auctions. In addition to the fields inherited from the Product model, the Auction model includes the following attributes

1. **StartDate**: This attribute represents the date and time when the auction starts, indicating the moment when bidding becomes active.
2. **EndDate**: This attribute indicates the date and time when the auction closes, i.e. when bids are no longer accepted.
3. **DurationInMs**: This field represents the duration of the auction in milliseconds, calculated as the time difference between the '**startDate**' and the '**endDate**'. It provides a numeric value that indicates the length of the auction.
4. **InstantPurchasePrice**: 'instantPurchasePrice' is a BigDecimal type attribute that specifies the fixed price at which a bidder can immediately purchase the auctioned item, effectively ending the auction when the bid is accepted.

These additional attributes address the specific needs of auctions and allow for the representation of time-sensitive bidding and instant purchase options. Together with the fields inherited from the Product model, the Auction model provides a comprehensive representation of auction items in the MongoDB database, supporting various e-commerce and bidding scenarios.

3.3 Front-End Microservice

The Front-End Microservice acts as an API gateway, enabling communication between client applications, such as web browsers, and other microservices. This architectural approach improves the separation of concerns, allowing the Front-End Microservice to concentrate on handling API requests and responses while delegating complex processing to pure back-end microservices. To facilitate effective communication between microservices, the Front-End Microservice utilises a message broker, simplifying messaging between itself and the back-end microservices. This decoupled design promotes modularity, scalability, and maintainability within the microservices ecosystem, ultimately contributing to a robust and efficient system architecture.

3.3.1 Tools and Technologies

Java and Spring Boot The front-end microservice uses the Java programming language and the Spring Boot framework to enable rapid development and efficient functionality. This choice of technology stack is driven by the inherent benefits it offers. The benefits of using Java and Spring Boot have been described in detail in sections 3.1.1 and 3.1.1. This combination allows developers to implement features and enhancements quickly, ensuring that the front-end microservice remains responsive and adaptable to evolving user requirements.

Thymeleaf is a Java-based templating engine used in web applications for server-side rendering. Its integration with Java web frameworks, notably Spring Boot, serves to streamline the development of dynamic web pages.

The ascendancy of Thymeleaf can be ascribed to two key attributes. Firstly, its adoption of an HTML-like syntax presents an intuitive environment for template creation and modification, significantly reducing the learning curve. Secondly, Thymeleaf excels in the domain of server-side rendering, a process whereby HTML content is dynamically generated on the server and subsequently dispatched to the client. This methodology confers several advantages, including streamlined search engine optimisation (SEO), heightened security measures, and the guarantee of uniform rendering across diverse devices and browsers.

Moreover, Thymeleaf exhibits remarkable compatibility with the Spring Boot framework. This compatibility enables developers to utilise the full potential of Spring Boot's extensive capabilities, which include facets such as dependency injection and robust security mechanisms, in conjunction with Thymeleaf's templating posibilities. Notably, Thymeleaf includes a powerful expression language (Thymeleaf EL) that allows developers to establish seamless connections between templates and backend data. This functionality facilitates the dynamic generation of content predicated on user inputs, session data, or queries directed at the underlying database.

Spring Session is an extension to the Spring Framework that addresses the challenges of managing sessions in web applications, making it an excellent choice for front-end microservices. At its core, Spring Session provides a more robust and scalable solution than traditional in-memory session handling by externalising the management and storage of HTTP sessions. This is especially advantageous in the context of front-end microservices because it allows for the use of stateless architecture and horizontal scalability, which are essential.

Of particular benefit is the ability to centralise session data storage. This ensures that session data is stored in a shared, externalized storage system such as Redis or a relational database, even when multiple instances of the microservice run concurrently to handle incoming requests. Centralization simplifies session management and allows all microservice instances to access session data consistently. This fosters a stateless architecture, improving both reliability and scalability.

Spring Session is compatible with the broader Spring ecosystem and offers significant benefits. Spring Session was chosen for use in front-end microservices to provide robust session management, enable a stateless architecture, and integrate seamlessly with the Spring ecosystem to help build scalable and reliable microservice.

3.3.2 Communication With Other Microservices

The Frontend Microservice plays a pivotal role as an API gateway in the microservices architecture. This strategic position allows it to act as an intermediary between client applications and other microservices. It facilitates communication through a dual mechanism: the use of a message broker for asynchronous messaging and HTTP requests for synchronous interactions. The Frontend Microservice enhances the efficiency and cohesion of the microservices ecosystem by effectively managing the flow of requests and responses.

3.4 Deployment

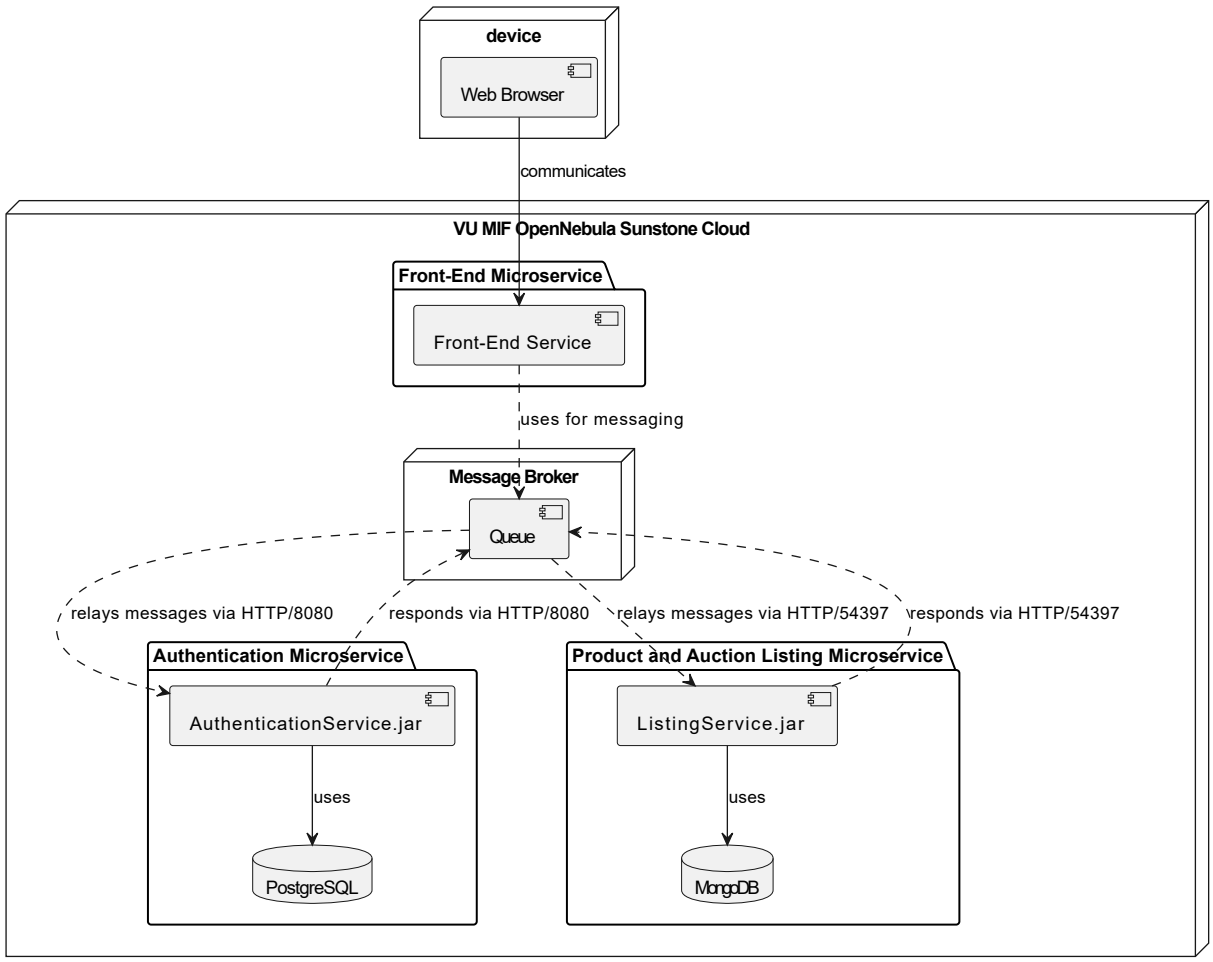

Figure 6. UML deployment diagram

The diagram 6 is a UML (Unified Modeling Language) Deployment Diagram, which is used to visualize the physical deployment of artifacts on nodes. Here's a description based on the visible elements in the diagram:

1. **VU MIF OpenNebula Sunstone Cloud:** This is the overall environment where the deployment is occurring. It uses the OpenNebula cloud management platform for providing infrastructure services.

2. **Front-End Microservice:**

- **Front-End Service** This is the service that handles the user interface and interaction.
- **Uses for messaging** Indicates that this service uses a messaging system, which is used to communicate with other services.

3. **Message Broker:**
   - **Queue** It's part of the messaging system that temporarily holds messages before they are processed by the consuming services.
   - **Relays messages via HTTP/8080** Indicates that the Message Broker receives messages over HTTP using port 8080.
   - **Responds via HTTP/8080** It also sends out messages or responses over the same protocol and port.
4. **Authentication Microservice:**
   - **AuthenticationService.jar:** This is a JAR (Java archive) file that contains the service used for authentication purposes.
   - **PostgreSQL:** This is a database used by the authentication service to store user credentials information.
5. **Product and Auction Listing Microservice:**
   - **ListingService.jar:** Similar to the authentication service, this is a Java archive file that contains the service for listing products and auctions.
   - **MongoDB:** A NoSQL database used by the listing service to store product and auction data.
6. **Device:** Represents the client device with a web browser that communicates with the front-end service.
7. **Web Browser:** The software application used for accessing information on the World Wide Web, which interacts with the front-end service.

The dashed lines with labeled arrows indicate the interactions between different components, such as the communication between services and databases, as well as the messaging protocol used.

This diagram provides a high-level overview of how the services are deployed and interact within the cloud environment, focusing on a microservice architecture with a message broker pattern for handling communications.

4 Faced Challenges

This section delves into the various challenges encountered during the development of three distinct microservices within our system: the Authentication Microservice, the Product and Auction Listing Microservice, and the Front-End Microservice. Each of these microservices presented unique obstacles and complexities that needed to be addressed to ensure the successful implementation and seamless integration into the overall system architecture. This chapter aims to provide an insightful overview of the specific challenges faced, shedding light on the intricacies of designing, building, and coordinating these essential components within our system's ecosystem.

4.1 Authentication Microservice

During the development of the Authentication Microservice, we faced several challenges, especially in comprehending and efficiently implementing JSON Web Tokens (JWTs). The challenges involved understanding the complex structure and functionality of JWTs, and finding ways to use them effectively in the authentication process while maintaining their security and validity.

One of the main challenges was understanding the complexity of JWTs. These tokens are composed of three distinct parts: the header, payload, and signature, each serving a unique purpose in the authentication process. It was crucial to understand how these components interrelated and contributed to the overall integrity of the token. Considerable effort was dedicated to studying the JWT specification and analyzing sample tokens to obtain a thorough comprehension of their structure.

After understanding the basics, the next challenge was to seamlessly integrate JWTs into the authentication flow. The challenge was to create a system that would not only produce a token after successful authentication, but also validate it effectively.

In addition, we encountered the challenge of creating efficient token validation routines to verify their authenticity and claims without compromising performance. We designed comprehensive token validation logic, utilizing cryptographic libraries and best practices to ensure the tokens remained secure against any form of tampering.

In addressing these challenges, we developed a comprehensive understanding of JWTs and their potential to enhance the security and functionality of our Authentication Microservice. This experience enhanced understanding of authentication mechanisms.

4.2 Product and Auction Listing Microservice

Embracing MongoDB as our database technology for the Product and Auction Listing Microservice introduced a significant learning curve and posed challenges, particularly during the initial stages of development. Connecting the microservice application with MongoDB was a noteworthy challenge that required a thorough understanding of both the technology and its integration nuances. MongoDB's document-oriented structure, schema flexibility, and the absence of rigid table structures differed markedly from traditional relational databases, demanding a shift in the approach to data storage and retrieval.

Establishing a seamless connection between the application and MongoDB took effort and diligence. I had to navigate the intricacies of MongoDB's drivers and libraries, understand its query language, and ensure that data interactions were efficient and reliable. Overcoming this challenge was instrumental in harnessing the full potential of MongoDB's scalability and adaptability for

the microservice. It was a valuable learning experience that underscored the importance of robust database integration in building a high-performance and dynamic microservice.

4.3 Front-End Microservice

Connecting the Front-End Microservice to a message broker and configuring seamless communication with other microservices was a notably challenging task. The intricacies of setting up this communication channel involved numerous hurdles, including selecting the right message broker technology, ensuring message reliability, handling message serialization and deserialization, and managing message routing and subscriptions. Additionally, understanding the nuances of the microservices' APIs and data formats required meticulous attention to detail. Coordinating the timing and sequencing of messages to maintain data consistency across services proved to be a significant challenge. Overcoming these complexities required a deep understanding of both the Front-End Microservice and the broader system architecture, as well as a commitment to thorough testing and troubleshooting to ensure reliable and efficient communication among the microservices.

Conclusions and Recommendations

Throughout the project's development phase, a rigorous competitive analysis was conducted, proving to be instrumental in shaping the system's requirements and selecting the most advantageous elements from existing solutions. This comprehensive examination of competitors allowed to gain valuable insights and identify best practices that greatly influenced development approach. As a result, we successfully crafted three key microservices. The first enables users to effortlessly create auctions and publish items for sale, providing a user-friendly platform for e-commerce activities. The second microservice facilitates user registration and later authorization, employing JWT tokens for enhanced security and user management. Finally, the third microservice seamlessly integrates with the previously described components, ensuring a cohesive and fully functional system that meets the demands of the users.

Working on this project has significantly deepened my understanding of microservice architecture, providing invaluable hands-on experience that will undoubtedly prove to be an asset in future endeavors. The complexities and challenges encountered during the development process allowed me to gain a profound insight into the intricacies of designing, implementing, and orchestrating microservices within a larger system. This newfound expertise will serve as a solid foundation for future projects, enabling to make informed decisions, optimize performance, and enhance scalability while leveraging the advantages of microservices to build robust and adaptable software systems.

Recommendations:

- Consider creating a dedicated gateway microservice within the system architecture. Introducing such a gateway service can facilitate seamless integration with external systems. This gateway serves as a unified entry point, streamlining communication and data exchange between microservices and external entities. Centralizing authentication, authorization, and request routing would enhance the system's flexibility, security, and scalability. This approach would also position us well to adapt to future changes and expand our system's capabilities, making it a valuable asset for our organization's long-term success.

- Consider creation of comprehensive test suites and implementing automated deployment processes in the development workflow. This strategic move can significantly boost development speed and, in turn, have a positive impact on the overall quality of the software. Extensive test suites ensure the reliability and robustness of the code and help catch issues early in the development cycle, reducing the time and effort spent on debugging and maintenance. The automation of deployment processes can streamline the release pipeline, resulting in faster and more consistent delivery of updates and new features. Adopting these practices can accelerate development cycles, increase productivity, and eventually deliver a more stable, higher quality product to the users.

- Consider optimizing certain microservices' performance by transitioning both the formulation and execution of database requests from Hibernate ORM to database procedures. This strategic shift can yield significant improvements in functioning speed and efficiency. Database procedures have the advantage of reducing the overhead associated with ORM frameworks, such as Hibernate, by executing database operations directly within the database

engine. This can result in faster response times and reduced resource utilization. Additionally, stored procedures often provide better control over query execution plans and can lead to more efficient database interactions. By implementing this change, the overall performance of microservices can be enhanced, resulting in a more responsive and scalable system for the users.

Future Work

This section outlines plans for future work and the to-do items for each of the key microservices in the system. This section will detail the enhancements and optimizations for the Authentication Microservice, the Product and Auction Listing Microservice, and the Front-End Microservice. These plans aim to refine and expand the system's functionality, addressing any identified shortcomings and evolving to meet the changing needs of users and the ever-evolving technology landscape. By laying out these tasks and objectives, a clear roadmap for the continued development and improvement of the microservices is ensured, ultimately delivering a more robust and user-friendly platform.

General Plans

Gateway Microservice The development of a gateway microservice is an anticipated and strategic step to ensure seamless integration with various backend systems. This gateway microservice will serve as a central point of contact, enabling the system to communicate effectively with external backend systems, APIs, or services. By implementing this gateway, the integration process can be streamlined, authentication and authorization can be managed, and data exchange can be orchestrated with minimal complexity. Such a solution not only enhances interoperability but also promotes flexibility, allowing our system to adapt and connect with a wide array of backend systems, thereby future-proofing our architecture and expanding our capabilities to meet the evolving needs of our organization.

In addition to facilitating integration, the development of a gateway microservice will also enhance the system's security measures. By establishing a single access point through which all microservices must pass, robust security protocols, including authentication and authorization checks, firewall protections, and rate limiting can be implemented . This centralized approach ensures a controlled and secure environment, reducing the potential attack surface and vulnerabilities. By using the gateway as a gatekeeper, microservices architecture can be strengthened and thus protect system from unauthorised access and potential security threats.

Certificates Considering the adoption of certificates to enable the use of the HTTPS protocol for microservices is a strategic move that can greatly enhance the flexibility and security of the system. Implementing HTTPS through certificates ensures secure data transmission, safeguarding sensitive information from potential eavesdropping and man-in-the-middle attacks. Furthermore, HTTPS enables us to securely distribute microservices across various machines or execution environments. This is because HTTPS provides encryption and authentication, which are crucial for secure communication in a distributed architecture. This approach aligns with best practices in network security and data protection, which are paramount in today's interconnected and data-driven landscape. It reinforces the integrity of the microservices.

Authentication Microservice

The following functional requirements can be expected to be implemented next:

1. An authentication microservice user should be able to update their user profile details to change their credentials and secure their account from unauthorised access.

2. An authentication microservice user should be able to renew their access token to extend their session time without having to provide their credentials again.

3. An authentication microservice user must have the ability to invalidate their access token to ensure that the token cannot be used to access the session the user wishes to terminate.

Bidding Microservice

Moving the bidding functionality into a dedicated microservice is a strategic decision that can yield substantial benefits in terms of performance and load distribution. By isolating bidding into its own microservice, a specialized component is created that is optimized for handling the intricate logic and real-time processing associated with auctions. This focused approach enhances performance by reducing the complexity of other microservices and ensuring efficient resource allocation. Furthermore, it enables better load distribution, allowing us to scale the bidding microservice independently to meet fluctuating demand. This architectural refinement improves our system's efficiency and ensures a more responsive and scalable platform, capable of delivering a seamless auction experience to our users even during peak loads and high traffic periods.

Bidding Microservice Requirements

Functional Requirements

- **High Bid Retrieval:** The microservice shall provide the functionality for users to receive information on the three highest active bids in any given auction, enabling them to assess the feasibility of placing a higher bid.

- **Bid Placement Feature:** The microservice shall enable users to place bids on specific auctions, offering them the opportunity to purchase items at competitive prices.

- **Bid Cancellation Option:** The microservice must offer users the ability to cancel their bids, thereby freeing their funds for use in other auctions.

Low-Priority Functional Requirements

1. **User Bid Monitoring:** The microservice shall allow users to view all of their bids across various auctions, facilitating the tracking of their competitive standing in ongoing bids.

2. **Auction Participation History:** The microservice shall provide users with access to a comprehensive view of all auctions they have participated in, including information on auctions where they were outbid.

3. **Winning Bid Display:** The microservice shall display to the users all auctions where their bid was the highest, indicating successful acquisition of the item.

Non-Functional Requirements

- The microservice receives requests through a message broker. The Point-to-Point (P2P) pattern is used.

- The microservice should be compatible with and run on Vilnius University's infrastructure.
- The microservice utilises a dedicated database to store bid information. This database will be exclusively accessed by the Bidding microservice.

Bidding Logic

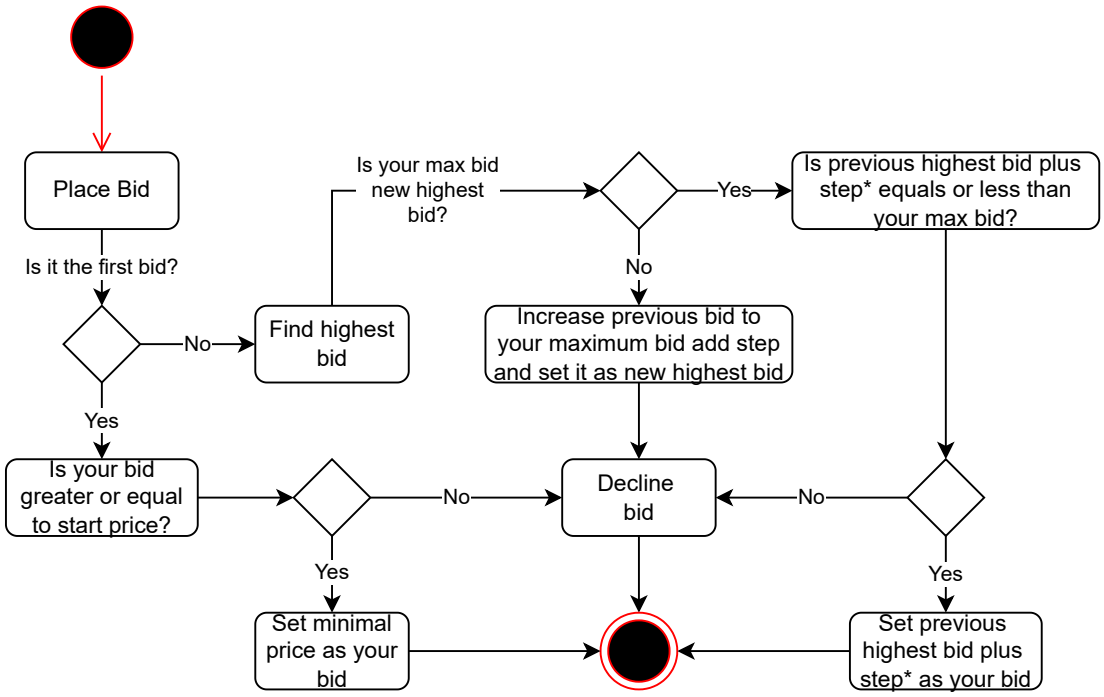

Figure 7. Bidding logic

The section on the bidding process explains how auctions are conducted within the microservices architecture. The discussion centres around UML activity diagram 7, which visually represents the underlying bidding logic. It provides an understanding of the sequential steps and decision points that govern the bidding process, ensuring transparency and clarity.

Additionally, this section presents table 3, which is a useful reference resource that outlines the variable step sizes used in the bidding mechanism. These step sizes are adjusted dynamically based on the current price, resulting in a more nuanced and responsive bidding experience. By including this table, the section improves readers' understanding of the complex pricing strategy used in the auction system, promoting a greater appreciation for the technical details that support the microservice's functionality.

Current price	Step size
0.01 Eur - 0.99 Eur	0.05 Eur
1.00 Eur - 4.99 Eur	0.25 Eur
5.00 Eur - 24.99 Eur	0.50 Eur
25.00 Eur - 99.99 Eur	1.00 Eur
100.00 Eur - 249.99 Eur	2.50 Eur
250.00 Eur - 499.99 Eur	5.00 Eur
500.00 Eur - 999.99 Eur	10.00 Eur
1000.00 Eur - 2499.99 Eur	25.00 Eur
2500.00 Eur - 4999.99 Eur	50.00 Eur
5000.00 Eur and up	100.00 Eur

Table 3. Step size dependence on current price

Social Interaction Microservice

The creation of a specialised Social Interactions Microservice presents a valuable opportunity to improve the overall value and attractiveness of the platform. This microservice can be instrumental in promoting user engagement and interaction, making the platform more dynamic and engaging for its users. By integrating functions such as user comments, likes, shares, and messaging, a vibrant social ecosystem can be created within the platform, encouraging users to connect and collaborate. Furthermore, the Social Interactions Microservice can be used as the basis for a comprehensive feedback and rating system. This system allows users to provide valuable insights and ratings on products, sellers, or services. The system enhances the user experience and provides valuable data for business intelligence and continuous improvement. Ultimately, this makes the platform more competitive and user-centric.

Functional Requirements

- An user of the Social Interaction microservice should be able to comment on an item, so their feedback can be shared with others.
- An user of the Social Interaction microservice should be able to react to comments, so they can help with filtering out irrelevant or improper feedback from other users.
- An user of the Social Interaction microservice should be able to send a message to another user, so they can ask for more details about an item.

Non-Functional Requirements

1. **Performance Requirement for Page Load Time:** Any request within the system must be handled within 2 seconds under the condition of a stable internet connection with a minimum download and upload speed of 300 Mbps. This requirement ensures efficient and timely access to the system's functionalities for users.
2. **Compatibility Requirement with Vilnius University's Infrastructure:** The microservice must be fully compatible with, and function seamlessly on, Vilnius University's existing

technological infrastructure. This requirement involves adhering to compatibility standards and system configurations specific to Vilnius University.

3. **Database Usage and Access Requirement:** A dedicated database utilized by the microservice shall be exclusively accessed and manipulated by the microservice itself. This requirement is essential for maintaining data integrity, security, and ensuring that the database's operations are optimized for the microservice's specific needs.

Front-End Microservice

The following functional requirements can be expected to be implemented next:

1. **Auction Bidding Form:** The system shall include a form enabling users to place bids on auctions.

2. **Automatic Token Renewal:** The system must automatically renew access tokens, maintaining seamless authentication via the Authentication microservice.

3. **Active and Outbid Bid View:** The system must provide a view where users can see their active and outbid bids, along with relevant information about each bid.

4. **Feedback and Comment Form:** The system shall include a form that allows users to write comments for feedback, react to posted comments, and provide an edit feature for modifying their own comments.

References

[1] Dick Hardt. The oauth 2.0 authorization framework. Technical report, 2012.

[2] Michael Jones, John Bradley, and Nat Sakimura. Rfc 7519: Json web token (jwt), 2015.

[3] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. Oauth 2.0 threat model and security considerations. Technical report, 2013.

[4] Nazia Majadi, Jarrod Trevathan, and Neil Bergmann. uauction: Analysis, design, and implementation of a secure online auction system. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 278--285, 2016.

[5] Simon Maple. JVM Ecosystem report 2018 - About your Platform and Application, 11 2021.

[6] Vinted. Choosing item condition, 11 2023. Accessed on 27th November 2023.