МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко

підпис

«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Розробка мікросервісного застосунку на основі модульної платформи .Net

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409.22010925

Здобувач _____В. С. Юхненко

підпис

«__» _____ 2024 р.

Керівник ст. викладач _____ К. О. Обухова

підпис

«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

підпис

«__» _____ 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. кафедри
_____Є. О. Давиденко
«___»_____2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи бакалавра

Видано здобувачу групи 409 факультету комп'ютерних наук

_Юхненку Вадиму Сергійовичу_____
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи

_Розробка мікросервісного застосунку на основі модульної платформи .Net___

Затверджена наказом по ЧНУ ім. Петра Могили від «22» грудня 2023 р. № 269.

2. Строк представлення кваліфікаційної роботи «__» _____ 2024 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є застосунок з мікросервісною архітектурою на основі модульної платформи .Net, який буде використовуватися для здійснення пошуку й моніторингу пропозицій, що стосуються питань купівлі-продажу автомобіля

4. Перелік питань, що підлягають розробці
- аналіз предметної області, методології розробки;
- аналіз застосунків-аналогів;
- визначення вимог до необхідного функціоналу;
- проєктування й розробка програмного забезпечення;
- огляд принципу функціонування розроблюваного застосунку;
- здійснення аналізу результатів розробки.

5. Перелік графічних матеріалів
Презентація

6. Завдання до спеціальної частини
Охорона праці в робочих приміщеннях фахівців з інформаційних технологій

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алексєєва А. О.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи ст. викладач Обухова Катерина Олександрівна
(посада, прізвище, ім'я, по батькові)

(підпис)

Завдання прийнято до виконання

Юхненко Вадим Сергійович
(прізвище, ім'я, по батькові здобувача)

(підпис)

Дата видачі завдання «22» грудня 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема кваліфікаційної роботи**:**

Розробка мікросервісного застосунку на основі модульної платформи .Net

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	22.12.2023 р.	23.12.2023 р.	Виконано
2.	Огляд літератури за темою роботи	15.01.2024 р.	22.01.2024 р.	Виконано
3.	Складання календарного плану КРБ	23.01.2024 р.	24.01.2024 р.	Виконано
4.	Аналіз предметної області	25.01.2024 р.	30.01.2024 р.	Виконано
5.	Розробка проєктних рішень	31.01.2024 р.	03.02.2024 р.	Виконано
6.	Моделювання та конструювання ПЗ	04.02.2024 р.	08.02.2024 р.	Виконано
7.	Кодування, тестування розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	09.02.2024 р.	20.05.2024 р.	Виконано
8.	Розробка спеціальної частини зохорони праці	21.05.2024	27.05.2024	Виконано
9.	Оформлення КРБ та презентації	28.05.2024	30.05.2024	Виконано
10.	Відгук керівника КРБ	31.05.2024	02.06.2024	Виконано
11.	Попередній захист	03.06.2024	05.06.2024	Виконано
12.	Завершення оформлення КРБ та презентації	06.06.2024	14.06.2024	Виконано
13.	Рецензування	15.06.2024	18.06.2024	Виконано
14.	Захист кваліфікаційної роботи			

Розробив здобувач ____Юхненко Вадим Сергійович_____ _____
(прізвище, ім'я, по батькові) *(підпис)*

«____» _____ 20__ р.

Керівник роботи ____ст. викладач Обухова К. О._____ _____
(посада, прізвище, ім'я, по батькові) *(підпис)*

«____» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра
«Розробка мікросервісного застосунку
на основі модульної платформи .NET»
Здобувач 409 гр.: Юхненко Вадим Сергійович
Керівник: ст. викладач Обухова Катерина Олександрівна

Робота присвячена розробці мікросервісного застосунку на основі модульної платформи .NET для пошуку й моніторингу вигідних пропозицій пов’язаних із придбанням автомобілів. Вибір теми зумовлений тенденціями розробки сучасних програмних забезпечень і численними перевагами цього архітектурного підходу: висока гнучкість, адаптивність застосунку й підвищена стійкість системи у разі виникнення збоїв у роботі одного або декількох мікросервісів. Використання модульної платформи .NET забезпечить сумісність й інтеграцію різноманітних сервісів, що значно спростить процес розробки, підтримки застосунку й зробить обробку великих обсягів даних, необхідних для пошуку й моніторингу вигідних пропозицій, більш ефективною.

Об’єкт: процес розробки мікросервісного застосунку для стабільної й швидкої роботи при моніторингу або пошуку вигідних пропозицій, що стосуються купівлі транспортного засобу.

Предмет: технології й інструменти, які необхідні для пошуку й моніторингу вигідних пропозицій, пов’язаних із купівлею транспортного засобу.

Мета: розробка мікросервісного застосунку на основі модульної платформи .NET для швидкого пошуку й моніторингу пропозицій щодо придбання автомобіля.

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків та переліку джерел посилання.

У вступі визначається актуальність теми, що приймається за мету, та проводиться невеликий огляд поставленого завдання, предмета та об’єкта дослідження.

У першому розділі наведено порівняння існуючих альтернативних

застосунків, проводиться аналіз їх сильних та слабких сторін, що допоможе у складанні специфікації вимог до програмного забезпечення, що розробляється.

У другому розділі визначаються основні функціональні можливості та ставляться вимоги до майбутнього програмного забезпечення, щоб забезпечити виконання поставлених вимог специфікації застосунку. Детально висвітлюється процес взаємодії мікросервісів для обробки й отримання всіх необхідних даних. Наведено опис вирішення головної поставленої задачі.

У третьому розділі описуються основні мікросервіси програмного забезпечення, надається інформація щодо технологій, мов програмування та бібліотек, розробляються UML-діаграми.

У четвертому розділі проводиться опис користувацьких інтерфейсів, демонструється тестування розробленого програмного забезпечення, виконується аналіз роботи, очікуваних та отриманих даних.

У висновках проводиться аналіз роботи та отриманих результатів.

КРБ викладена на 73 сторінках (без додатків), вона містить 4 розділи, 51 ілюстрацію, 8 таблиць, 27 джерел в переліку посилань, 1 додаток.

Ключові слова: *мікросервісний застосунок, модульна платформа .NET, взаємодія мікросервісів, користувацькі інтерфейси, розробка програмного забезпечення*.

ABSTRACT

of the Bachelor’s Thesis

"Development of a microservice application based on the .NET modular platform"

Student: Yukhnenko Vadym

Supervisor: Senior teacher Obukhova Kateryna

The work is dedicated to the development of a microservices application based on the modular .NET platform for searching and monitoring advantageous offers related to car purchases. The choice of topic is driven by trends in modern software development and the numerous advantages of this architectural approach, including high flexibility, application adaptability, and increased system resilience to failures in one or more microservices. Using the modular .NET platform ensures compatibility and integration of various services, significantly simplifying the development and maintenance processes of the application and making the processing of large volumes of data needed for searching and monitoring advantageous offers more efficient.

Object: the process of developing a microservice application for stable and fast operation when monitoring or searching for favorable offers related to the purchase of a vehicle.

Subject: technologies and tools needed to find and monitor great deals related to the purchase of a vehicle.

Objective: development of a microservice application based on the .NET modular platform for quick search and monitoring of offers for the purchase of a car.

The bachelor’s thesis consists of an introduction, four chapters, conclusions, and a list of references.

The introduction determines the relevance of the topic taken as the goal and provides a brief overview of the task, subject, and object of research.

The first chapter describes a comparison of existing alternative applications, highlighting their strengths and weaknesses. This will help in drawing up a specification of requirements for the software being developed.

The second chapter describes the basic functionality and requirements for the future software to ensure that the requirements of the application specification are met.

The process of interaction of microservices for processing and obtaining all necessary data is covered in detail. A description of the solution to the main task is given.

The third chapter describes the main software microservices, provides information about technologies, programming languages, libraries, and develops UML diagrams.

The fourth chapter describes the user interfaces, demonstrates the testing of the developed software, analyzes the work, and compares the expected and received data.

The conclusions analyze the work and the results obtained.

The bachelor's thesis is presented on 73 pages (without appendices), it contains 4 chapters, 51 figures, 8 tables, 27 references, 1 appendix.

Keywords: microservice application, .NET modular platform, microservices interaction, user interfaces, software development

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних
ПЗ – програмне забезпечення
ПК – персональний комп’ютер
СКБД – система керування базами даних

AMQP – advanced message queuing protocol
API – application programming interface
BFF – backend for frontend
CRUD – create read update delete
gRPC – google remote procedure calls
OIDC – openID connect
SSO – Single Sign-On
UI – user interface
UML – unified modeling language
UX – user experience

ВСТУП

Актуальність теми кваліфікаційної роботи бакалавра зумовлена не лише широким використанням мікросервісної архітектури у сучасній розробці вебзастосунків і сайтів продажу, а й зростанням попиту на придбання автомобілів за вигідною ціною задля особистісного використання або підтримки солдатів, які після широкомасштабного вторгнення потребують якісні транспортні засоби для фронтових задач. Цей архітектурний підхід дозволяє розбити складні системи на невеликі, незалежні сервіси, що спрощують розширення й підтримку застосунку. Така стратегія розробки стає важливою у контексті розвитку платформ-аукціонів, де важливо забезпечити стабільну і швидку роботу системи при великому обсязі даних і транзакцій. На сьогоднішній день важливо мати засоби для аналізу лотів і відстежування найвигідніших пропозицій, що не лише зекономить час і матеріальні ресурси, а також забезпечить доступ до ширшого асортименту автомобілів і можливість отримання кращих умов угоди.

Подібні інструменти користуються популярністю не лише серед фізичних, а й юридичних осіб, оскільки вони допомагають ефективно вирішувати потреби в автотранспорті, пов’язані з логістикою й підвищенням продуктивності бізнесу. Можливість придбання авто за оптимальною ціною через платформи-аукціони стає стратегічним рішенням для них, сприяючи ефективному використанню ресурсів і забезпеченню потреб у транспорті без зайвих витрат фінансів і часу.

Об’єкт: процес розробки мікросервісного застосунку для стабільної й швидкої роботи при моніторингу або пошуку вигідних пропозицій, що стосуються купівлі транспортного засобу.

Предмет: технології й інструменти, які необхідні для пошуку й моніторингу вигідних пропозицій, пов’язаних із купівлею транспортного засобу.

Мета: розробка мікросервісного застосунку на основі модульної платформи .NET для швидкого пошуку й моніторингу пропозицій щодо придбання автомобіля.

Для досягнення поставленої мети необхідно вирішити наступні **завдання**:

– аналіз предметної області, методології розробки;

– аналіз застосунків-аналогів;

– аналіз існуючих технологій розробки програмного забезпечення;

– визначити необхідний функціонал застосунку;

– розробити бекенд частину мікросервісного застосунку із використанням технологій .NET, RabbitMQ, PostgreSql, MongoDB, SignalR;

– розробити фронтенд частину мікросервісного застосунку із використанням технології Next.js.

Практичне значення: мікросервісний застосунок забезпечує стабільну й швидку роботу платформи-аукціону, що дозволяє ефективно шукати й відстежувати вигідні пропозиції для придбання транспортних засобів. Це рішення стане корисним як для фізичних осіб, так і для бізнесів, що потребують оптимального використання ресурсів для задоволення своїх потреб, сприяючи зменшенню витрат часу й коштів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність розробки застосунку для онлайн-аукціонів

Тема кваліфікаційної роботи бакалавра у сфері розробки застосунків для онлайн-аукціонів є актуальною, оскільки відзначається постійним зростанням попиту на такі архітектурні рішення в різних сегментах. Це обумовлено не лише загальною тенденцією до цифровізації у бізнесі й особистому споживанні, але й стрімким розвитком інформаційних технологій у сучасному світі. Платформи спростили спосіб проведення аукціонів, запропонувавши безперебійну й ефективну альтернативу традиційним процесам особистих торгів. Перехід на онлайн-тендери є не просто відповіддю на глобальний поштовх до цифровізації, а й стратегічним кроком для розширення, охоплення ринку, покращення досвіду клієнтів і оптимізації операцій. Розглянемо актуальні аспекти, які підтримують значимість розробки застосунків для онлайн-аукціонів.

Зростання конкуренції на ринку автотранспорту й стрімкий розвиток технологій призводять до того, що користувачі вимагають не лише зручного і швидкого доступу до інформації про лоти, а й можливості аналізу наявних даних для прийняття обґрунтованих рішень. Застосунки для онлайн-аукціонів повинні забезпечувати користувачів можливістю відстеження найвигідніших пропозицій, що сприятиме як економічній вигоді, так й ефективному використанню часу.

Сучасні інформаційні технології дозволяють створювати складні системи, що оптимізують процеси покупки й продажу автомобілів через онлайн-аукціони. У сучасному бізнес-середовищі мікросервісна архітектура зарекомендувала себе як ефективний інструмент для розбиття складних систем на менші, автономні компоненти. Використання цього архітектурного підходу у розробці застосунків для платформ-аукціонів стає ключовим чинником у забезпеченні ефективної роботи застосунку при великому обсязі даних і транзакцій. Розбиття складних систем на невеликі, незалежні сервіси дозволяє спростити їх розширення й

підтримку, що важливо у забезпеченні стабільної і швидкої роботи онлайн-платформ.

Однією з вагомих переваг мікросервісної архітеркутри для бізнес рішень є менш катастрофічні наслідки при збою роботи мікросервісів, ніж у великих системах. Збій в одній частині монолітної програми зазвичай завдає шкоди функціонуванню решти застосунку. Після цього розробник повинен діагностувати проблему і, можливо, випустити іншу версію усієї програми, щоб виправити цю конкретну помилку. Швидше й простіше виправляти помилки в мікросервісній архітектурі. Усунення несправностей буде відбуватися у три кроки – ідентифікація, ізоляція, компенсація. Також розробник зможе швидко створити нову версію несправного мікросервісу, не маючи справу з операційним водоспадом (operational waterfall), який існує в монолітній програмі.

У сфері розподілених систем, особливо в області аукціонів, завдання підтримки узгодженості даних під час керування станом різних служб є першочерговим. Ця складність виникає через необхідність гарантувати, що всі компоненти, що беруть участь, мають уніфіковане уявлення про дані, незважаючи на притаманну затримку і ймовірність збоїв зв’язку. Масштабованість і розширення аукціонної системи залежать від надійної стратегії, яка може впоратися з динамічною природою аукціонів, де ставки постійно розміщуються, оновлюються і відкликаються в режимі реального часу. Із вищезазначеними задачами може впоратися мікросервісна архітертура, де необхідність координувати дії забезпечується без створення тісного зв’язку, що досягається використанням черги повідомлень.

Автомобільні онлайн-аукціони, завдяки прогресу цифрових платформ, підключення до Інтернету й безпечних платіжних систем, зробили революцію в досвіді продажу транспортних засобів. Зручність, доступність, прозорість й охоплення глобального ринку, пропоновані цими платформами, розширили можливості як для покупців, так і для продавців. У результаті індустрія

автомобільних аукціонів стала більш прозорою, ефективною й доступною для ширшого кола осіб.

Отже, актуальність розробки застосунку для онлайн-аукціонів у сфері автомобільного бізнесу визначається зростанням попиту на ефективні інструменти для купівлі-продажу автомобілів, а також необхідністю у забезпеченні швидкої й стабільної роботи платформи при великому обсязі даних і транзакцій. Такі застосунки стають не лише інструментом для фізичних осіб, але й стратегічним рішенням для бізнесу, що дозволяє оптимізувати витрати та забезпечувати потреби у транспорті без зайвих витрат.

1.2 Опис предметного середовища

Об’єкт дослідження – процес розробки мікросервісного застосунку для стабільної і швидкої роботи при моніторингу або пошуку вигідних пропозицій, пов’язаних з купівлею транспортного засобу.

Предмет дослідження – технології й інструменти, необхідні для пошуку й моніторингу вигідних пропозицій на ринку автомобілів, зокрема через платформи-аукціони.

Процес розробки мікросервісного застосунку для моніторингу й пошуку вигідних пропозицій на аукціонах автомобілів виступає об’єктом дослідження. Цей процес включає в себе аналіз, проєктування, розробку і впровадження програмного забезпечення, яке допомагатиме користувачам ефективно вибирати й купувати автомобілі за допомогою онлайн-аукціонів.

Структурні і функціональні особливості об’єкта дослідження визначаються його складовими частинами і функціональними можливостями. Структурні особливості об’єкта дослідження:

1) бекенд: включає в себе серверну частину застосунку, яка відповідає за обробку запитів користувачів, взаємодію з базами даних (БД) і роботу з іншими сервісами. Для розробки цієї частини мікросервісного застосунку заплановане використання технологій .NET, RabbitMQ, PostgreSql, MongoDB і SignalR;

2) фронтенд: включає в себе клієнтську частину застосунку, яка забезпечує відображення інтерфейсу користувача і взаємодію з ним. Планується розробка з використанням технологій Next.js і React, що дозволить створити ефективний і зручний інтерфейс користувача;

3) бази даних: включають в себе реляційну БД PostgreSql для зберігання структурованої інформації про лоти на аукціонах та документи, пов’язані з транспортними засобами, а також NoSQL базу даних MongoDB для зберігання неструктурованої інформації, наприклад, зображення автомобілів.

Функціональні особливості об’єкта дослідження визначаються завданнями і функціями, які необхідно виконати для досягнення мети дослідження. Функціональні особливості об’єкта дослідження:

1) аналіз предметної області методології розробки: дослідження і вибір методів, підходів, що найбільш ефективно використовуються для розробки мікросервісних застосунків для платформ-аукціонів;

2) аналіз застосунків-аналогів: вивчення й аналіз існуючих програмних рішень для моніторингу й пошуку вигідних пропозицій на ринку автомобілів через платформи-аукціони з метою виявлення переваг і недоліків для подальшого використання у власному проєкті;

3) аналіз існуючих технологій розробки програмного забезпечення: вибір технологій та інструментів, що найкраще підходять для реалізації функціональності мікросервісного застосунку;

4) визначення необхідного функціоналу застосунку: встановлення переліку функцій і можливостей, які повинен забезпечити мікросервісний застосунок для ефективного моніторингу й пошуку вигідних пропозицій на аукціонах із продажу автомобілей.

Розробка мікросервісного застосунку для проведення онлайн-аукціонів має на меті підвищити ефективність процесу придбання транспортних засобів, забезпечуючи користувачам швидкий і зручний доступ до актуальної інформації про пропозиції на ринку. Це вимагає детального аналізу технологій та інструментів,

які будуть використовуватися в розробці, а також урахування специфіки потреб користувачів і особливостей ринку автомобілів.

1.3 Огляд існуючих аналогів

Проведення аналізу ринку існуючих альтернатив сприятиме висвітленню їх сильних і слабких сторін. Також огляд функцій застосунків-аналогів є важливим кроком, що спростить процес складання специфікації вимог до розроблюваного програмного забезпечення (ПЗ). Для проведення аналізу альтернатив обрано наступні популярні програмні рішення в сегменті продажу авто: eBay Motors (табл. 1.1), IAAI (табл. 1.2), Copart (табл. 1.3).

1.3.1 eBay Motors

eBay Motors, філія популярної платформи eBay, є онлайн-сайтом автомобільних аукціонів, який пропонує нові і вживані транспортні засоби, а також запчастини, аксесуари для різних типів транспортних засобів, таких як автомобілі, мотоцикли, пікапи й позашляховики. Цей онлайн-аукціон обслуговує приватних продавців й автосалони, надаючи широкий вибір транспортних засобів [1].

Таблиця 1.1 – Опис eBay Motors

Назва	**eBay Motors**
Розробник	Компанія eBay Inc.
Архітектура	Client-server
Мови реалізації	Java, C++, Python – backend JavaScript, HTML, CSS – frontend
Функції	1) Користувачі можуть шукати автомобілі за різними параметрами і застосовувати фільтри для точного вибору; 2) Користувачі можуть спілкуватися з продавцями, задавати питання й узгоджувати умови покупки; 3) Власники автомобілів можуть розміщувати свої оголошення з описом і фотографіями своїх транспортних засобів для продажу; 4) eBay Motors може надавати інструменти для оцінки вартості автомобілів на основі різних факторів; 5) Користувачі можуть брати участь у віртуальних аукціонах та робити ставки на транспортні засоби.

Кінець таблиці 1.1

Переваги	1) Для перегляду доступний широкий вибір категорій і марок транспортних засобів; 2) Повна інтеграція з традиційною платформою eBay, що дозволяє зареєстрованим користувачам легко отримати доступ до eBay Motors; 3) Також пропонуються запчастини й аксесуари для обслуговування або оновлення транспортних засобів.
Недоліки	1) Обмежений щодо ексклюзивних пропозицій порівняно з іншими онлайн-аукціонами; 2) Знайти вигідну пропозицію стало складніше через посилення конкуренції серед учасників торгів, що потенційно може призвести до вищих кінцевих цін.
Вебсайт	www.ebay.com/b/Auto-Parts-and-Vehicles/6000

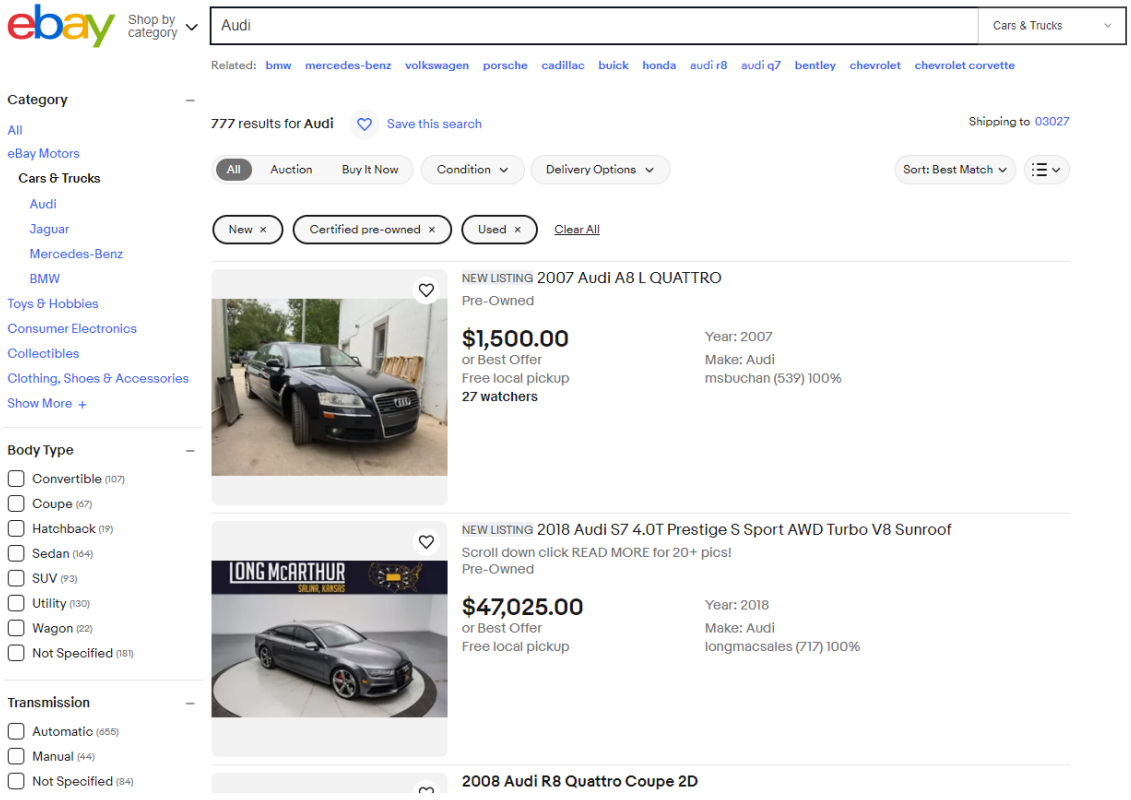

Рисунок 1.1 – Дизайн інтерфейсу застосунку «eBay Motors»

Графічний інтерфейс користувача створено з акцентом на зручність і простоту використання. Основні пункти меню розташовані на верхній і лівій панелях навігації, забезпечуючи легкий доступ до ключових функцій застосунку:

– home (домашня сторінка);

– search (пошук);

– sort by (сортування за критеріями);

– filter options (параметри фільтрування).

Хоча eBay Motors має широкий вибір транспортних засобів і зручний інтерфейс, його недоліки також слід враховувати. Серед них обмежений доступ до ексклюзивних пропозицій порівняно з іншими платформами, а також підвищена конкуренція серед учасників торгів, що зазвичай призводить до надмірного підвищення кінцевих цін. Такі обмеження можуть ускладнити пошук вигідних угод для користувачів.

1.3.2 IAAI

Провідний автомобільний онлайн-аукціон, що спеціалізується на продажу врятованих і пошкоджених автомобілях. Пропонуючи широкий асортимент транспортних засобів, у тому числі автомобілі з незначними або серйозними пошкодженнями або навіть пошкоджені внаслідок повені, IAAI надає покупцям можливість знайти транспортні засоби за потенційно нижчими цінами для різних цілей, таких як ремонт, збір запчастин або перепродаж [2].

Таблиця 1.2 – Опис IAAI

Назва	**IAAI**
Розробник	Insurance Auto Auctions, Inc.
Архітектура	Client Server
Мова реалізації	Java, Python – backend JavaScript, HTML, CSS – frontend
Функції	1) Платформа надає можливість участі в аукціонах для купівлі-продажу автомобілів з пошкодженнями, що не покриває страхування; 2) Користувачі можуть шукати автомобілі за різними критеріями, наприклад, виробник, модель, рік випуску тощо; 3) Власники автомобілів можуть додавати, редагувати і видаляти свої лоти, встановлюючи умови продажу і фіксуючи стан транспортних засобів; 4) Платформа надає інтерфейс страховим компаніям для управління страховими випадками, реалізацією й перереєстрацією автомобілів; 5) Користувачі можуть звертатися до служби підтримки для отримання допомоги, вирішення проблем або отримання відповідей на запитання.

Кінець таблиці 1.2

Переваги	1) Зручний вебсайт автомобільного аукціону з розширеними фільтрами пошуку, які допомагають звузити вибір автомобіля; 2) Для тих, хто не може бути присутнім на аукціоні, доступні параметри живих ставок і проксі-ставки; 3) IAAI забезпечує надання детальної інформацію про кожен автомобіль, що допомагає покупцям приймати обдумані рішення.
Недоліки	1) Дуже нагромаджений інтерфейс на сторінках, що описують характеристики авто; 2) Порівняно з конкурентами має непривабливий інтерфейс сайту.
Вебсайт	www.iaai.com

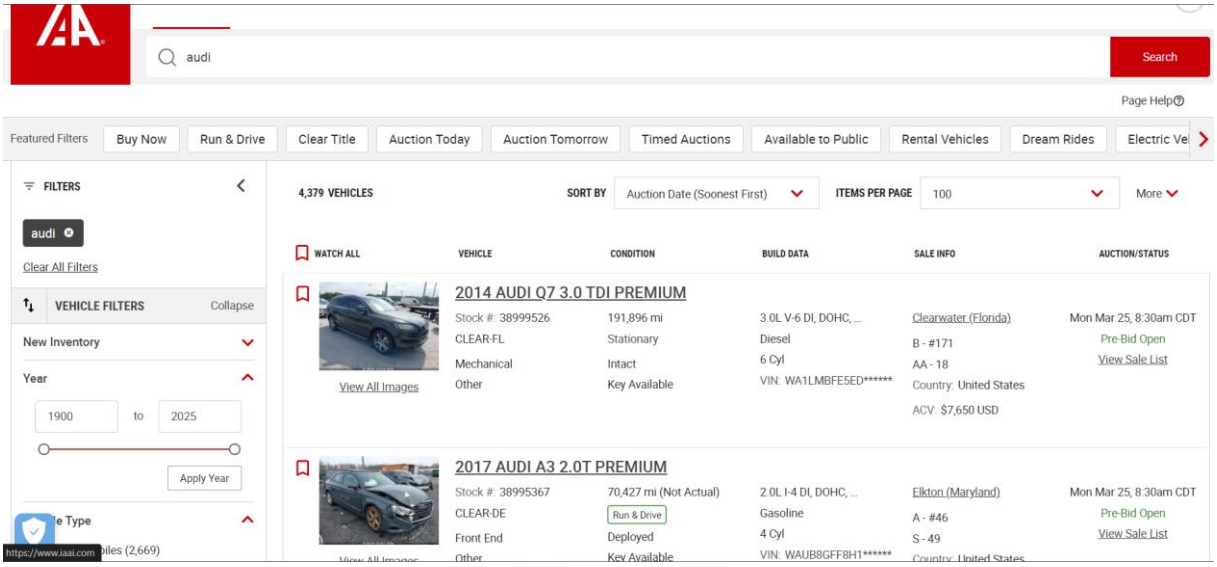

Рисунок 1.2 – Вигляд інтерфейсу застосунку «IAAI»

Інтерфейс користувача є гнучким і налаштовуваним, незважаючи на його застарілий дизайн і нагромадженість, він містить різноманітні інструменти, функції, які допомагають користувачу більш ретельно вибирати транспортний засіб. Наприклад, отримавши список автомобілів, що відповідають пошуковому запиту, можна обрати необхідні позиції:

– від 1 до 999 – це лоти, які заводяться і рухаються;

– від 1000 до 1999 продаються виключно мотоцикли й інша мототехніка;

– від 3000 до 3999 продаються транспортні засоби, які не можуть рухатися;

– від 4000 і далі – спеціалізовані транспортні засоби, а також допоміжна техніка.

Хоча автомобільний онлайн-аукціон IAAI пропонує широкий асортимент автомобілів із різними рівнями пошкоджень за зниженими цінами, його інтерфейс страждає від надмірної нагромадженості і непривабливого дизайну, що може ускладнити користувачам знаходження потрібної інформації і зробить процес покупки менш зручним.

1.3.3 Copart

Copart – це добре відома платформа для онлайн-аукціонів автомобілів, що спеціалізується на продажу й утилізації транспортних засобів із різних джерел, включаючи страхові компанії, автосалони й агентства з оренди [3].

Таблиця 1.3 – Опис Copart

Назва	**Copart**
Розробник	Copart Inc.
Архітектура	Client server
Мови реалізації	PHP– backend JavaScript, HTML, CSS – frontend
Функції	1) Можливість участі у віртуальних аукціонах для придбання автомобілів й інших транспортних засобів; 2) Можливість скористатися послугами незалежних експертів стосовно огляду авто; 3) Можливість продажу транспорних засобів різним категоріям покупців; 4) Можливість здійснювати пошук по фільтрам й сортувати отриманий список товарів.
Переваги	1) Ретельний підхід до реєстрація нових користувачів: перед участю в аукціоні: потрібно завантажити ідентифікаційні документи й будь-які ліцензії для здійснення комерційної діяльності; 2) Можливість скористатися послугами брокера; 3) Доступність на різних платформах.
Недоліки	1) Менш детальний опис товару порівняно з конкурентами; 2) Для новачків процес навігації по сайту і взаємодії з його функціями може бути складним і заплутаним.
Вебсайт	www.copart.com

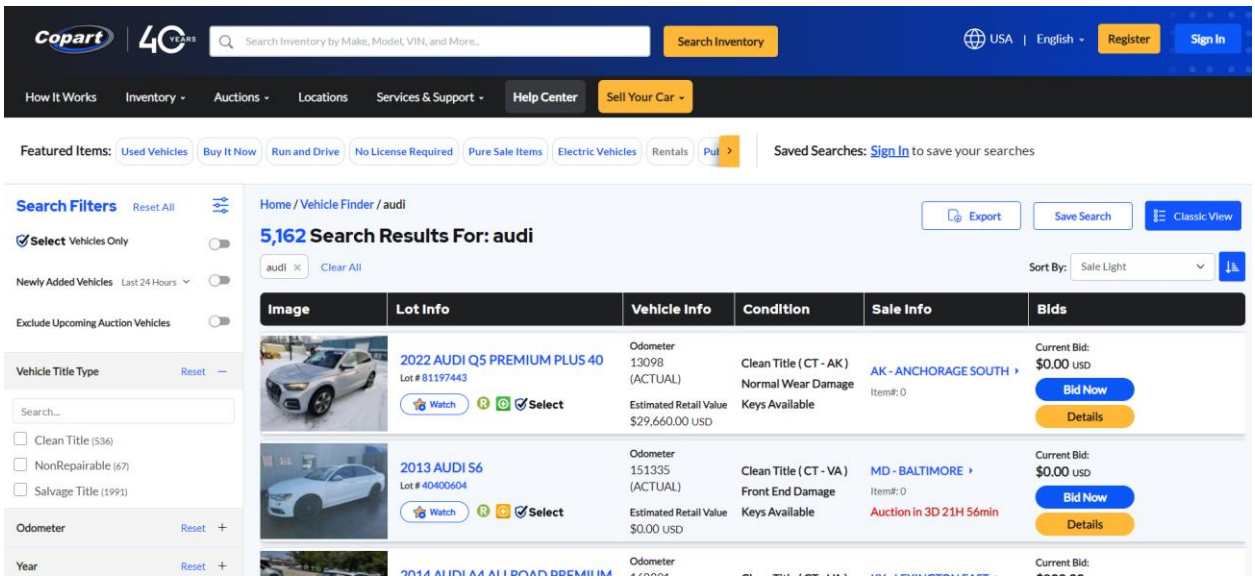

Рисунок 1.3 – Вигляд інтерфейсу застосунку «Copart»

Незважаючи на свої недоліки, складний для навігації інтерфейс і менш детальний опис товарів, дана платформа володіє привабливішим інтерфейсом, а також пропонує широкий спектр функцій для вивчення стану транспортного засобу, який буде куплений, починаючи від фото й закінчуючи висновком служби інспекції.

1.4 Аналіз розроблюваного застосунку

Інструменти для здійснення інтернет-покупок транспортних засобів не настільки популярні як звичайні маркетплейси, але з початком повномасштабного вторгнення питання купівлі машин для потреб фронту за вигідною ціною стало дуже важливим. Їх можна використовувати для швидкого й ефективного пошуку авто, а також для відстеження змін у торгах, щоб за обмежений час встигнути відреагувати на зміни й здійснити успішну покупку.

ПЗ має забезпечувати безпеку для користувачів й повідомляти клієнта про зміни в аукціоні. Існування окремих мікросервісів у застосунку допоможе не тільки при майбутньому масштабуванні системи, а й при додаванні нового функціоналу.

Таблиця 1.4 – Опис системи що розробляється

Основні задачі	1) реєстрація нових користувачів; 2) авторизація й автентифікація; 3) можливість переглядати лоти без входу в систему; 4) пошук транспортного засобу за назвою, кольором, виробником або датою публікації лоту; 5) здійснення CRUD операцій над лотами для авторизованих користувачів; 6) перегляд створених користувачем лотів; 7) перегляд лотів, де користувач переміг.
Користувачі системи	1. користувач; 2. анонімний користувач.
Сценарії роботи	1) користувач авторизується в через вебклієнт і здійснює пошук необхідного автомобіля; 2) неавторизований користувач за допомогою пошукового рядка знаходить необхідні для нього лоти; 3) користувач авторизується через вебклієнт й переглядає лоти, які він створив; 4) користувач після авторизації здійснює CRUD операції відносно створеного ним лоту.
Засоби апаратної та програмної реалізації	1. back-end: .NET, RabbitMQ, SignalR; 2. front-end: React, Next.js; 3. databases: PostgreSQL, MongoDB.

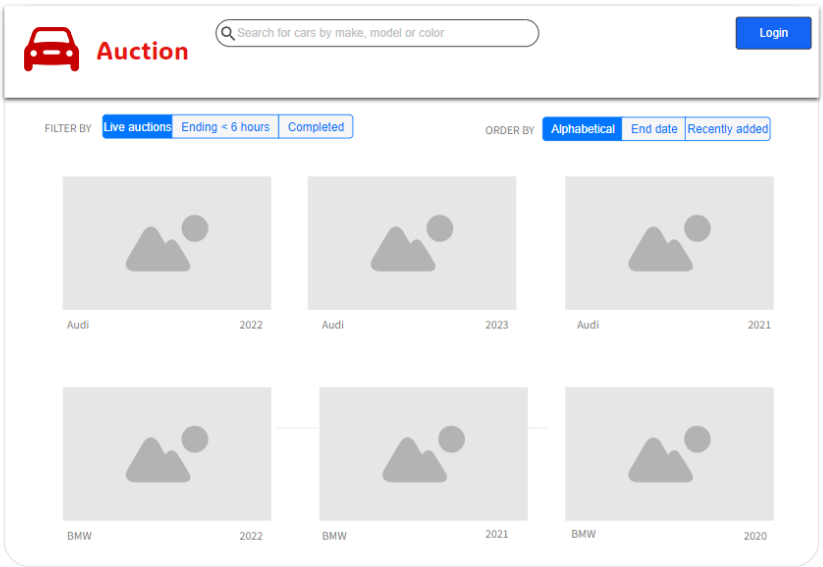

Рисунок 1.4 – Макет інтерфейсу розроблюваного застосунку

Розроблюваний застосунок має кілька переваг над конкурентами, особливо у забезпеченні безпеки для користувачів й моніторингу змін у торгівлі. Крім того, окремі мікросервіси в системі сприятимуть як розширенню застосунку, так і додаванню нового функціоналу, забезпечуючи більшу гнучкість і швидкість реакції на зміни.

1.5 Специфікації вимог до програмного забезпечення

Проєкт розробки програмного забезпечення має на меті створення інноваційного рішення для моніторингу й пошуку вигідних пропозицій щодо купівлі автомобіля.

Призначення й межі проєкту:

1) призначення системи (застосунку), для якої розробляється програмне забезпечення: призначенням застосунку є моніторинг і пошук вигідних пропозицій для купівлі автомобіля;

2) погодження, що ухвалені в програмній документації: погоджено, що для створення ПЗ і його злагодженої роботи будуть використовуватися допоміжні фреймворки і бібліотеки – ASP.NET Core, Next.js, React, Duende Identity Server, RabbitMQ й SignalR;

3) межі проєкту ПЗ: крайня дата завершення роботи над ПЗ – 20.05.2024 р.

Загальний опис:

1) сфера застосування: дане ПЗ не має обмежень у сферах його застосування;

2) характеристики користувачів: основні характеристики користувачів: наявність персонального комп’ютеру (ПК) й доступу до мережі Інтернет;

3) загальна структура й склад системи: основні частини для створення програмного забезпечення – identity, auction, search, bidding, notification мікросервіси, бази даних (PostgreSQL і MongoDB), клієнтська фронтенд частина;

4) загальні обмеження: обмеження для роботи з ПЗ – наявність ПК й підключення до мережі Інтернет.

Функції системи (пошук лотів і відстеження ставок, які робили інші користувачі):

1) опис функції: функція дозволяє авторизованому користувачу здійснювати пошук за параметрами лоту або часу його публікації;

2) вхідна і вихідна інформація: вхідна інформація – запит користувача про наявність лоту за пошуковими параметрами; вихідна інформація – інформація про доступні аукціони й ставки, які робили інші користувачі.

3) функціональні вимоги:

а) реєстрація й авторизація – ця функція дозволяє користувачам створювати обліковий запис на платформі й увійти в систему для доступу до персонального кабінету:

– вхідні дані: для реєстрації: ім’я користувача, електронна пошта, пароль, прізвище й ім’я; для авторизації: електронна пошта, пароль;

– вихідні дані: успішна реєстрація: підтвердження успішного створення облікового запису; успішна авторизація: доступ до участі в аукціонах і створення особистих;

б) пошук аукціонів – ця функція дозволяє користувачам відстежувати пропозиції стосовно купівлі автомобіля й оцінювати конкуренцію між іншими клієнтами:

– вхідні дані: вибір лоту за кольором, назвою або виробником автомобіля;

– вихідні дані: відображення пропозицій, що відповідають параметрам пошуку й відображення ставок інших користувачів;

в) участь в аукціонах –функція дозволяє авторизованим користувачам відстежувати зміни ставок у реальному часі й брати безпосередню участь в аукціоні:

– вхідні дані: вибір суми ставки, яка перевищує ціну визначену продавцем або зазначену останнім авторизованим користувачем;

– вихідні дані: відображення оновленого списку ставок у лоті:

інформація про суму грошей внесену потенційним покупцем.

Вимоги до інформаційного забезпечення:

1) джерела і зміст вхідної інформації (даних): в даному ПЗ джерелом вхідної інформації є користувач;

2) нормативно-довідкова інформація (класифікатори, довідники тощо): немає вимог до пункту;

3) вимоги до способів організації, збереження і ведення інформації: обмін даними відбувається через клієнтський застосунок, google remote procedure calls (gRPC), SignalR і RabbitMQ. В якості БД обрано PostgreSQL і MongoDB.

Вимоги до технічного забезпечення: для розробки програмного забезпечення немає значних технічних обмежень.

Вимоги до програмного забезпечення:

1) архітектура програмної системи: архітектура ПЗ складається з клієнтського застосунку, мікросервісів (identity, auction, search, bidding, notification, gateway), баз даних (PostgreSQL, MongoDB) і брокеру повідомлень (RabbitMQ);

2) системне програмне забезпечення: для написання фронтенд частини обрано бібліотеку React і фреймворк Next.js, для бекенд частини – фреймворк модульної платформи .Net – ASP.NET. Обмін даними має відбуватися з використанням SignalR і RabbitMQ. В якості БД обрано PostgreSQL і MongoDB;

3) мережеве програмне забезпечення: для створення ПЗ використовується ОС Windows, редактор коду Visual Studio Code і будь-який сучасний браузер для використання клієнтського застосунку;

4) програмне забезпечення ведення інформаційної бази: ведення інформаційної бази відбувається за допомогою CRUD-операцій з PostgreSQL і MongoDB;

5) мова і технологія розробки ПЗ: програмне забезпечення має бути розроблене з використанням фреймворку й бібліотеки ASP.NET і React. Мови розробки – C# і TypeScript.

Вимоги до зовнішніх інтерфейсів:

1) інтерфейс користувача: вебклієнт має задовольняти усім вимогам user experience (UX) й user interface (UI), що дозволить користувачу витратити найменше часу на розуміння роботи системи. Шаблон сторінки повинен бути розділений на дві частини, де перша частина – це пошуковий рядок з параметрами для фільтрації результатів пошуку, вона розміщена зверху, друга частина є список аукціонів з таймером його закінчення;

2) апаратний інтерфейс: пристрій користувача (ПК або ноутбук), який він буде використовувати для взаємодії зі сторінками вебклієнту;

3) програмний інтерфейс: ASP.NET – фреймворк з відкритим вихідним кодом для розробки застосунків і сервісів з використанням .NET й C#. React – бібліотека з відкритим вихідним кодом для розробки мобільних і браузерних застосунків з використанням мови програмування TypeScript;

4) комунікаційний протокол: застосунок базується на використанні мережних протоколів WAP – протокол бездротової передачі даних і TCP/IP.

Властивості програмного забезпечення:

1) доступність: застосунок є доступним для всіх користувачів, що мають бажання ним скористатися, за умови наявності ПК й доступу до мережі Інтернет;

2) супроводжуваність: супроводження не передбачається;

3) переносимість: застосунок доступний на будь-якому ПК-пристрої;

4) продуктивність: продуктивність ПЗ залежить від швидкості мережі Інтернет;

5) надійність: користувач отримує доступ виключно до своїх даних лише після авторизації у системі. За безпеку авторизації, автентифікації відповідають протоколи OpenID Connect (OIDC) й OAuth 2.0;

6) безпека: функціонал який дозволяє взаємодіяти з даними має перевіряти токен авторизації й права доступу. Також вищезгаданий протокол OIDC забезпечить стандартизовану ідентифікацію, що додатково підвищить рівень безпеки доступу.

Впровадження даного програмного забезпечення дозволить забезпечити доступність, надійність і безпеку для користувачів. Проєкт має забезпечити доступ користувачам до необхідної інформації для прийняття обґрунтованих рішень щодо купівлі автомобіля.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра було проведено аналіз наявних альтернативних програмних рішень, де розглянуто їх переваги й недоліки. Набуто досвіду у складанні специфікацій вимог до ПЗ, де враховувалися аспекти архітектури, мови реалізації, функціоналу і користувацького інтерфейсу аналогів.

Проведено аналіз системи, що розробляється, шляхом визначення ключових завдань, користувачів, сценаріїв роботи й ресурсів для апаратної і програмної реалізації. Представлено макет головної сторінки з докладним описом розмітки блоків. Розглянуто можливості майбутнього розширення застосунку шляхом модифікації окремих мікросервісів.

Після проведення підсумкового аналізу, визначено важливі етапи розвитку проєкту й зроблено висновок щодо актуальності, перспектив розроблюваного рішення у сегменті купівлі-продажу авто. Розглянуті аспекти аналізу й проєктування є ключовими у формуванні стратегії розробки застосунку для онлайн-аукціонів і виявленні його переваг у конкурентному середовищі.

2 ДОСЛІДЖЕННЯ, МОДЕЛЮВАННЯ І ТЕХНІЧНЕ ПРОЄКТУВАННЯ

2.1 Методи і технології розробки застосунку

Клієнт і мікросервіси можуть спілкуватися між собою за допомогою багатьох різних типів протоколів, кожен з яких орієнтований на різні сценарії й цілі. Спочатку ці типи комунікацій можна розділити на два види:

1) синхронний протокол. HTTP – це протокол, що передбачає послідовний обмін даними між клієнтом і сервером. Клієнт ініціює запит й очікує відповіді від сервера перед тим як продовжити виконання. Незалежно від того чи виконує клієнт код синхронно, чи асинхронно протокол (HTTP/HTTPS) передбачає синхронний характер, тобто виконання коду клієнта може продовжитися лише після отримання відповіді від сервера HTTP;

2) асинхронний протокол. Інші протоколи, такі як advanced message queuing protocol (AMQP), підтримуються багатьма операційними системами й хмарними середовищами і використовують асинхронні повідомлення. Код клієнта або відправник повідомлення зазвичай не чекає на відповідь. Він просто надсилає повідомлення, як і під час надсилання повідомлення до черги RabbitMQ або будь-якого іншого брокера повідомлень [4].

Для проєктування застосунку є необхідним інструмент, що реалізовує протокол AMQP. Даній вимозі відповідає платформа RabbitMQ, яка дозволяє мікросервісам взаємодіяти асинхронно через черги повідомлення. Розглянемо ситуацію, коли надходить post-запит до аукціонного мікросервісу через API. Замість того, щоб сервіс використовував запит HTTP до пошукового мікросервісу, буде використано підхід публікації (publish approach), щоб опублікувати це повідомлення у шину сервісів (service bus). Пошуковий сервіс буде підписаний на шину подій (event bus), отримає це повідомлення, а потім вживе дії, щоб у кінцевому результаті забезпечити консистентність баз даних Mongo і БД Postgres з асинхронним обміном повідомленнями. Аукціонний мікросервіс не знає про службу пошуку для поточних цілей і не цікавиться тим чи оброблено це

повідомлення правильно. Даний приклад також є демонстрацією підходу fire-and-forget, де завдання надсилаються на самостійну обробку, без відстеження початкової програми або очікування результатів цих завдань.[5]

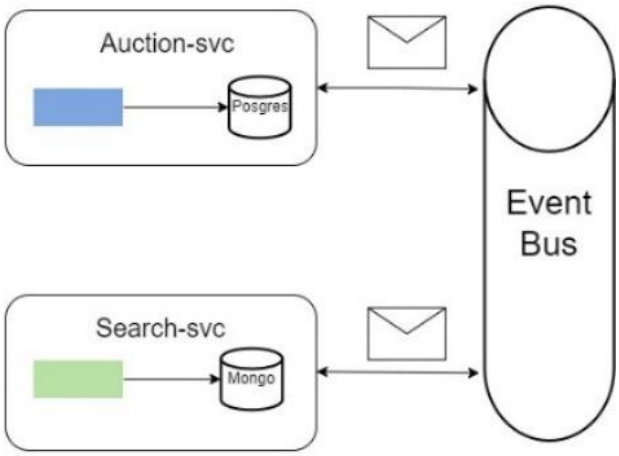

Рисунок 2.1 – Діаграма взаємодії сервісів асинхронно

Зв'язок між клієнтом і сервером у реальному часі є необхідним, оскільки проєкт розрахований на продаж автомобілів за допомогою аукціонів, де дуже цінується час і своєчасно зроблена ставка. При проєктуванні окремих частин мікросервісів буде використаний SignalR, який є хорошим способом досягнення зв’язку в реальному часі і надсилання вмісту клієнтам із внутрішнього сервера. Оскільки спілкування відбувається в режимі реального часу, клієнтські програми відображатимуть зміни майже миттєво. Зазвичай це обробляється за допомогою протоколу, такого як WebSockets, з використанням багатьох підключень WebSockets по одному на клієнта [6]. Спроєктуємо ситуацію, де даний інструмент може бути використаний. Наприклад, ставка надходить до відповідного мікросервісу, після вона потрапить до шини подій. Сервіс сповіщень, який підписаний на цю шину подій, отримає повідомлення і потім він передасть його всім клієнтам, які підключені до хабу.

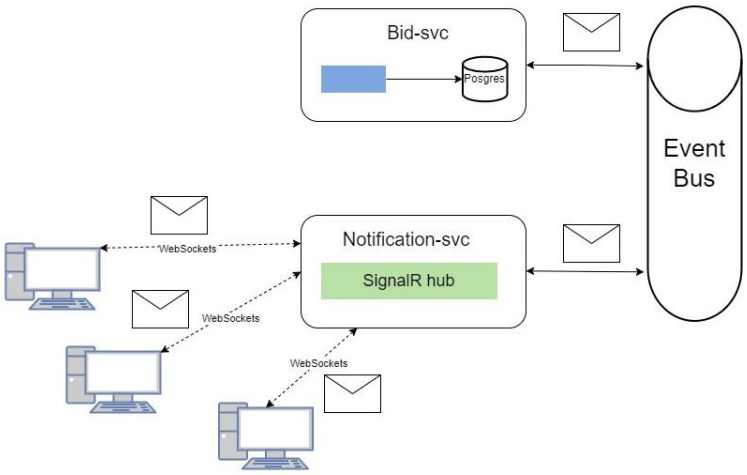

Рисунок 2.2 – Діаграма взаємодії сервісів і клієнтів, що підключені до хабу

Також для забезпечення швидкого виконання запитів й обміну даними між сервером буде використовуватися протокол gRPC. Він призначений для вирішення проблем, пов’язаних із великомасштабними розподіленими системами, що робить його відповідним вибором для сучасних глобальних платіжних систем. gRPC використовує буфери протоколів (protobuf) як мову визначення інтерфейсу (IDL) і формат серіалізації даних, забезпечуючи малі розміри повідомлень і швидший зв’язок порівняно з іншими форматами, такими як JSON.

Серед переваг даного протоколу, що будуть використовуватися, слід виділити продуктивність і взаємодію в реальному часі. Останній аспект представлений у двонаправленій потоковій передачі даних: gRPC дозволяє забезпечити зв’язок між клієнтом і сервером у реальному часі. Це особливо корисно для обробки великих обсягів даних або коли затримка є критичним фактором. Також продуктивність даного протоколу спричинена тим, що gRPC побудовано на HTTP/2, який підтримує мультиплексування, двонаправлене потокове передавання й керування потоком. Це призводить до зменшення затримки й покращенню пропускної здатності, що у свою чергу стане причиною швидшого й ефективнішого зв’язку [7].

Розглянемо область використання даного протоколу на прикладі роботи двох мікросервісів. На стороні клієнта й сервера буде створено протофайл, який

встановлює договір між ними, щоб мати змогу очікувати отримання запиту. Наприклад, користувач робить ставку у відповідному сервісі (bid service), він може не мати аукціону, що міститься в MongoDB, тоді потрібно буде перевірити в аукціонній службі (auction service) наявність запису про аукціон у БД. Якщо аукціон наявний, то він поверне його через HTTP/2 й gRPC до служби ставок. Потім цей аукціон буде збережено у відповідній БД. Тож усі майбутні ставки він зможе отримувати звідти.

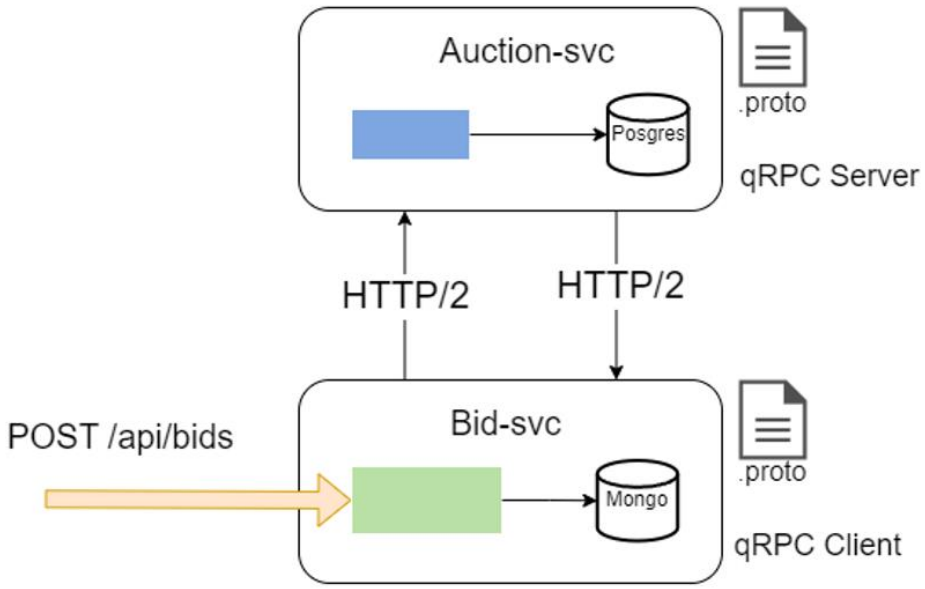

Рисунок 2.3 – Діаграма взаємодії сервісів за допомогою протоколу gRPC

У процесі проєктування, взаємодії між клієнтом і мікросервісами, важливо враховувати різні типи комунікаційних протоколів, оскільки вони дозволяють підвищити продуктивність й ефективність в системі. Не менш важливим аспектом є швидке реагування застосунку на події й зміни, що відбуваються, забезпечуючи зручну й ефективну комунікацію з користувачами.

2.2 Створення діаграми прецедентів

Діаграми прецедентів (або діаграма варіантів використання, англ. use case diagram), описують послідовність дій, які користувач повинен виконати для ефективної взаємодії з системою. Загалом, вони відображають взаємодію між

системою й зовнішніми акторами, демонструючи, як система використовується для досягнення певних цілей або завдань, не занурюючись у технічні аспекти програмної реалізації [8].

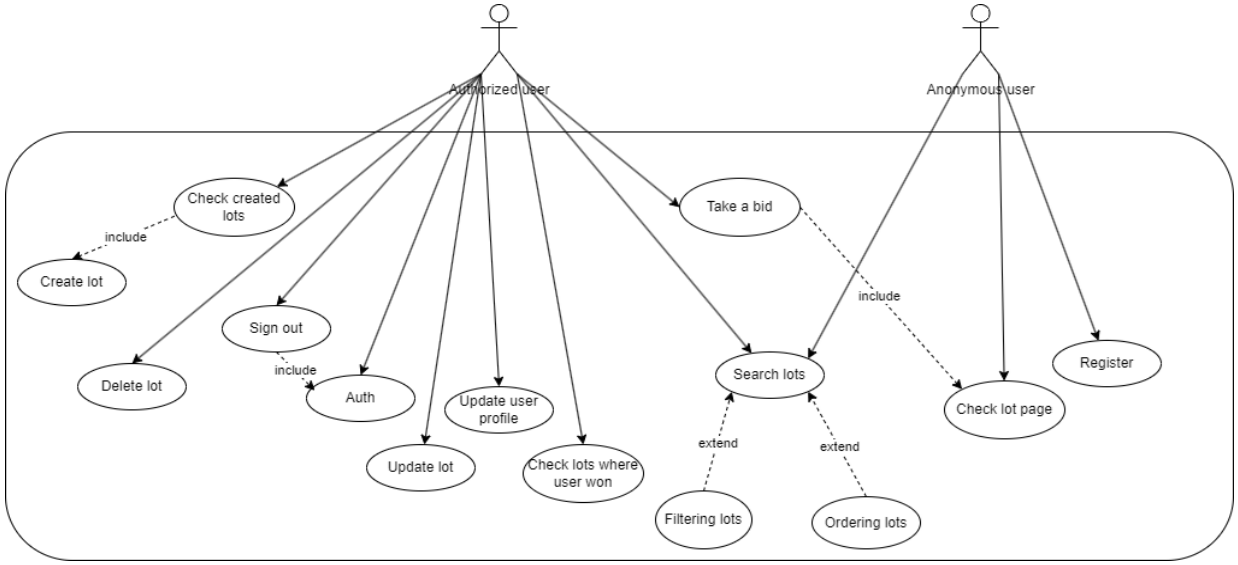

Рисунок 2.4 – Діаграма варіантів використання застосунку для покупки авто

Розробка діаграм прецедентів є ключовою для чіткого визначення ролі користувача в системі й послідовності кроків, необхідних для досягнення мети. Існують три різновиди сценаріїв використання:

1) короткий формат, який полягає в одному абзаці, що описує основний сценарій роботи системи, часто успішний. Цей формат зазвичай використовується на початковому етапі аналізу вимог до майбутньої системи;

2) поверхневий опис, який вільно представляє всі сценарії (основний та альтернативні) одного варіанту використання системи;

3) повний формат, який детально описує всі кроки і дії, включаючи перед і після умови виконання. Цей формат застосовується для надзвичайно важливих сценаріїв використання.

Короткий use case (пошук аукціону)

Користувач авторизується в системі й переходить на головну сторінку з пошуковим рядком і фільтрами для пошуку аукціонів. Користувач обирає фільтр, вводить назву, колір або виробника автомобіля й підтверджує операцію пошуку.

Система відправляє запит на сервер й уразі наявності аукціону, що відповідає вимогам клієнтам, отримує лот або лоти з БД, після чого оновлює UI.

Поверхневий use case (створення лоту)

Головний сценарій (успішний):

Користувач авторизується в системі й переходить на сторінку з формою для створення лоту. Користувач заповнює необхідні поля, натискає кнопку «Додати» для підтвердження операції. Система відправляє запит на сервер й у разі успішного додавання до БД, оновлює UI.

Альтернативні сценарії:

1) токен авторизації недійсний, тому користувач вимушений пройти процедуру автентифікації й авторизації знову;

2) не усі поля форми заповнені, тому при спробі створити лот буде повідомлено про відсутність даних у полях.

Таблиця 2.1 – Повний use case (внесення ставки)

Основний актор	Авторизований користувач
Область застосування	Застосунок для покупки авто
Рівень	Мета користувача
Передумови	Користувач успішно пройшов авторизацію в застосунку
Зацікавлені сторони та інтереси	Користувач зацікавлений у внесені ставки для успішної купівлі транспортного засобу
Основний сценарій успіху	1) Користувач проходить авторизацію через клієнтський застосунок. 2) Клієнт обирає лот, створений іншим користувачем. 3) Користувач під вікном, де відображається історія внесення ставок іншими користувачами, вводить суму більшу за останню ставку зроблену іншим клієнтом. 4) Після повернення на головну сторінку користувач помічає поруч з лотом не лише таймер до його закриття, а й суму останньої внесеної ставки.
Результат	Користувач передав на сервер інформацію про зроблену ставку.
Розширення	1) Користувач обрав лот, який сам і створив, у рядку введення суми виводиться повідомлення про неможливість внесення ставок у власному лоті.

Кінець таблиці 2.1

Розширення	2) Користувач вводить суму, яка менша за останню зроблену ставку або за ціну визначену користувачем, що створив лот. В історії зроблених ставок буде відображено повідомлення, що ставка була відхилення, а на головній сторінці поряд з таймером буде відображено суму останньої успішної ставки. 3) Токен авторизації недійсний, тому при переході на сторінку лоту у рядку для введення суми ставки буде повідомлення про необхідність авторизації. 4) При переході до лоту, що був завершений, користувач у рядку введення суми ставки спостерігатиме повідомлення про неможливість прийняти участь у даному аукціоні.
Спеціальні вимоги	1) Адаптивний інтерфейс. 2) Доступ до мережі Інтернет.
Частота появи	Система має змогу працювати майже безперервно

Сценарії використання повинні бути зрозумілими, концептуально простими й короткими, щоб забезпечити чітке уявлення про застосунок і його взаємодію з середовищем.

2.3 Алгоритмізація логіки застосунку

Для відображення робочого алгоритму застосунку часто використовуються діаграми діяльності. Вони є ефективним інструментом для моделювання різноманітних систем і процесів, включаючи як прості бізнес-процеси, так і складні програмні системи. Діаграми діяльності складаються з набору символів і позначень, які використовуються для відображення різних дій, таких як завдання, рішення й цикли, вони демонструють зв’язки та взаємозалежності між ними [9].

Для застосунку моніторингу трафіку створено три діаграми діяльності які описують наступний ключовий функціонал:

1) пошук лотів (рис. 2.5). Діаграма містить наступні кроки: авторизація користувача, відкриття головної сторінки, введення пошукового запиту, перевірка на наявність введених даних в БД;

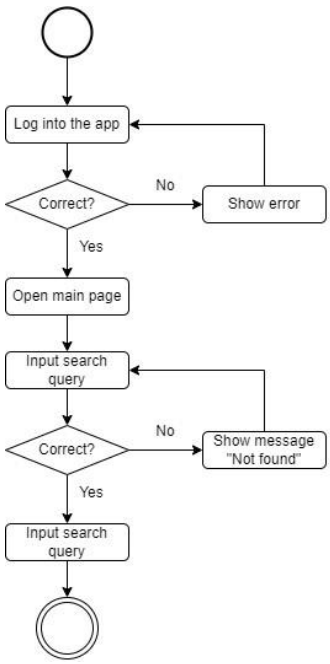

Рисунок 2.5 – Діаграма діяльності здійснення пошукового запиту

2) створення лоту (рис. 2.6). Діаграма містить такі кроки як: введення імені користувача й паролю, перевірка наявності необхідних даних для створення лоту і виконання запиту до мікросервісу ставок;

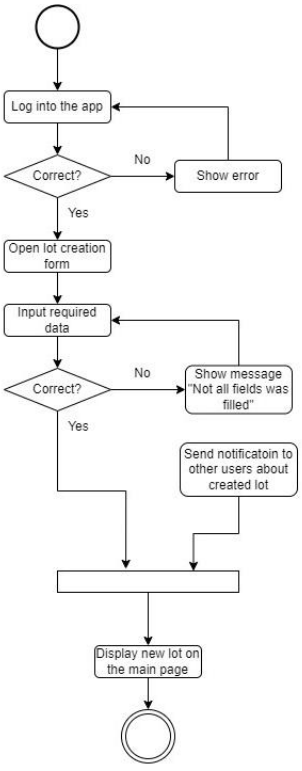

Рисунок 2.6 – Діаграма діяльності створення лоту

3) додавання ставки в лоті (рис. 2.7). Діаграма містить кроки: авторизація користувача, вибір лоту і введення суми ставки.

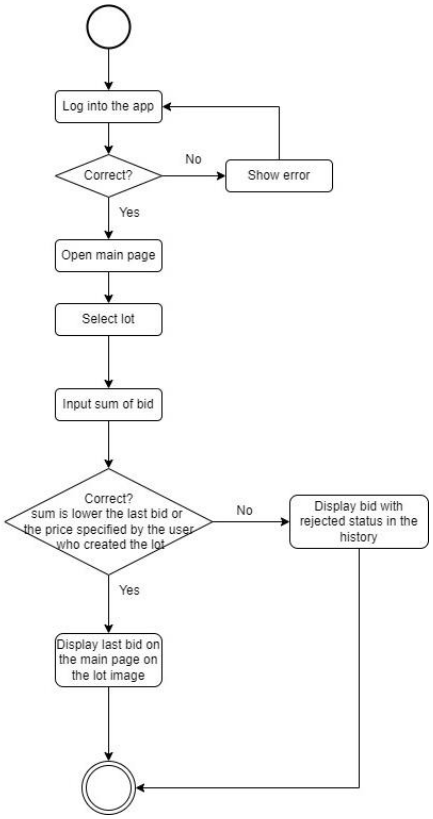

Рисунок 2.7 – Діаграма діяльності для участі в аукціоні

Поглянувши на діаграми діяльності, одразу стає помітною схожість з блок-схемами, але варто зазначити ключову відмінність у тому, що діаграми діяльності дозволяють підтримувати паралельні процеси.

2.4 Розробка діаграм взаємодії

Діаграма послідовності є одним з типів діаграми взаємодії, що демонструє взаємодію між компонентами в застосунку в упорядкованій за часом послідовності. Метою діаграм взаємодії є візуалізація інтерактивної поведінки системи [10].

Таблиця 2.2 – Опис можливих компонентів діаграми взаємодії

Елемент	Опис
Актори	Актори – це сутності, що взаємодіють із системою й зазвичай розташовуються у верхній частині діаграми.
Об’єкти	Об’єкти є екземплярами класів у системі та зображуються прямокутниками на діаграмі.
Лінії життя	Лінії життя ілюструють час існування об’єкта в системі й представлені вертикальними лініями, що починаються від верхньої частини прямокутника об’єкта.
Повідомлення	Повідомлення демонструють взаємодію між об’єктами, зображену стрілками, що з’єднують їхні лінії життя.
Смуги активації	Смуги активації відображають час, протягом якого об’єкт виконує метод, і зображуються горизонтальними прямокутниками на лінії життя.
Обмеження	Обмеження служать для визначення умов або обмежень, які мають бути виконані для здійснення повідомлення або взаємодії. Вони також можуть використовуватися для визначення передумов і післяумов під час роботи системи.

Діаграма послідовності корисна для моделювання й розуміння взаємодії між об’єктами чи компонентами в системі, а також її можна використовувати для розробки та тестування ПЗ.

Для моделювання мікросервісного застосунку для купівлі-продажу автомобілей розроблено 3 діаграми взаємодії:

1) вхід в систему (рис. 2.8). Діаграма представляє собою перехід на головну сторінку вебклієнту, після запиту до Identity service користувача буде перенаправлено на головну сторінку, де згодом відобразяться результати запиту до БД на отримання даних про наявні аукціони.

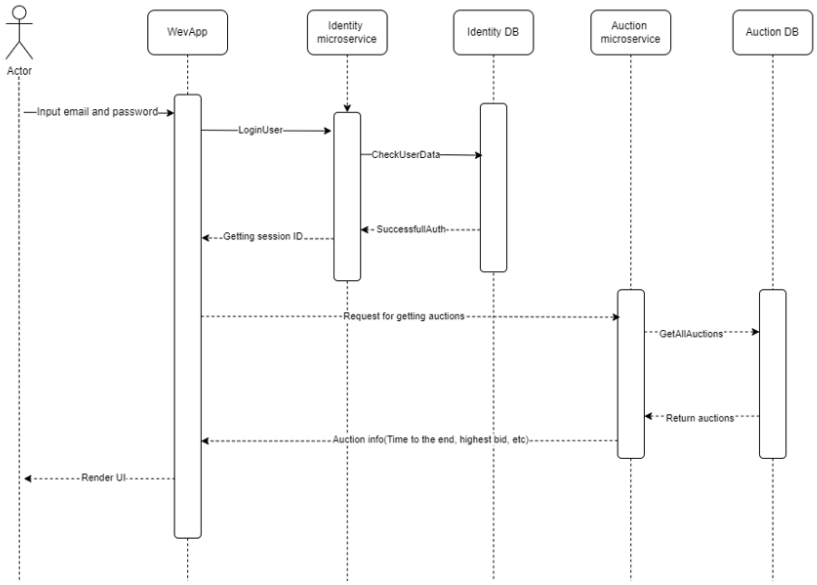

Рисунок 2.8 – Діаграма взаємодії входу в систему перегляду головної сторінки

2) додавання нового аукціону (рис. 2.9). Діаграма представляє собою перехід до вікна створення лоту, здійснення запиту до Auction service, який додає запис про новий товар в БД і повертає оновлений список пропозицій.

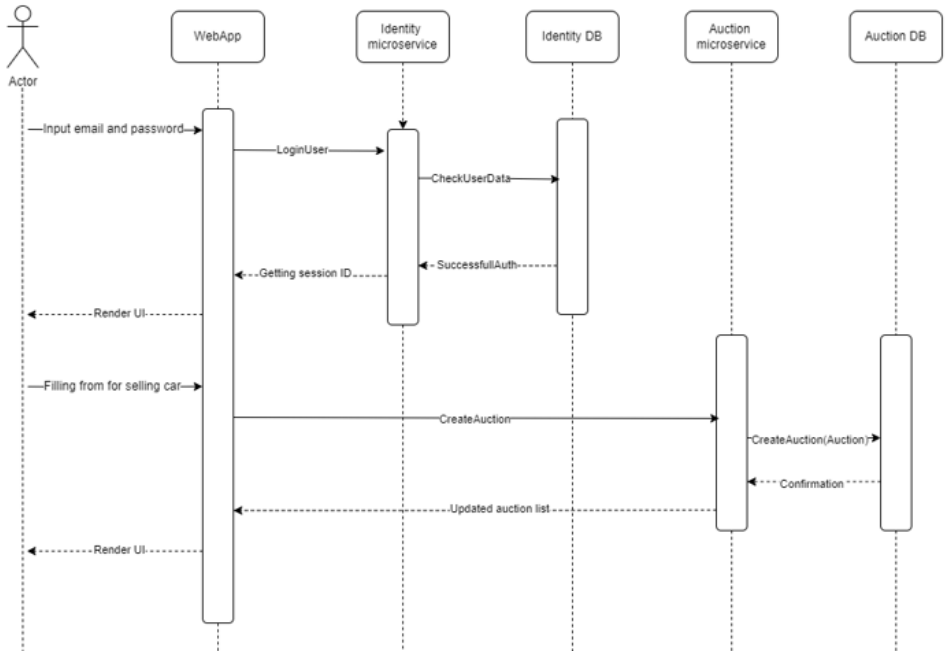

Рисунок 2.9 – Діаграма взаємодії додавання нового лоту

3) пошук лоту за його описом (рис. 2.10). Наступна діаграма описує процес пошуку товару за його певними характеристиками, після введення запиту

в пошуковому рядку здійснюється пошук необхідного товару у базі даних і повертається результат.

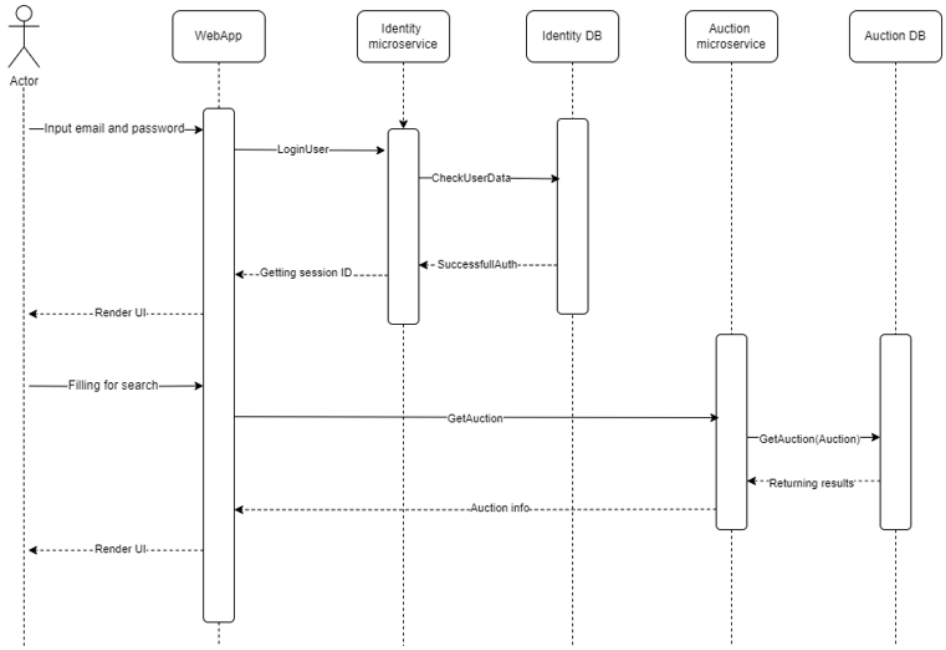

Рисунок 2.10 – Діаграма взаємодії пошук інформації про товар

Зазначені діаграми чітко зображують потік повідомлень та викликів методів між об’єктами й показують їх порядок взаємодії.

2.5 Конструювання й аналіз діаграми розгортання

Діаграми розгортання застосовуються для моделювання фізичного розташування компонентів програмного забезпечення. Вони відображають, як компоненти взаємодіють на апаратних вузлах, наприклад, маршрутизатори й комутатори, оскільки вони спілкуються через мережу.

Ці діаграми є корисними інструментом для архітекторів програмного забезпечення й розробників, оскільки дозволяють зрозуміти структуру розгортання системи і виявити потенційні проблеми з продуктивністю і проблемними місцями. Крім того їх можна використовувати для планування розгортання системи у виробничому середовищі або для швидкого інформування розробників про стратегії розгортання.

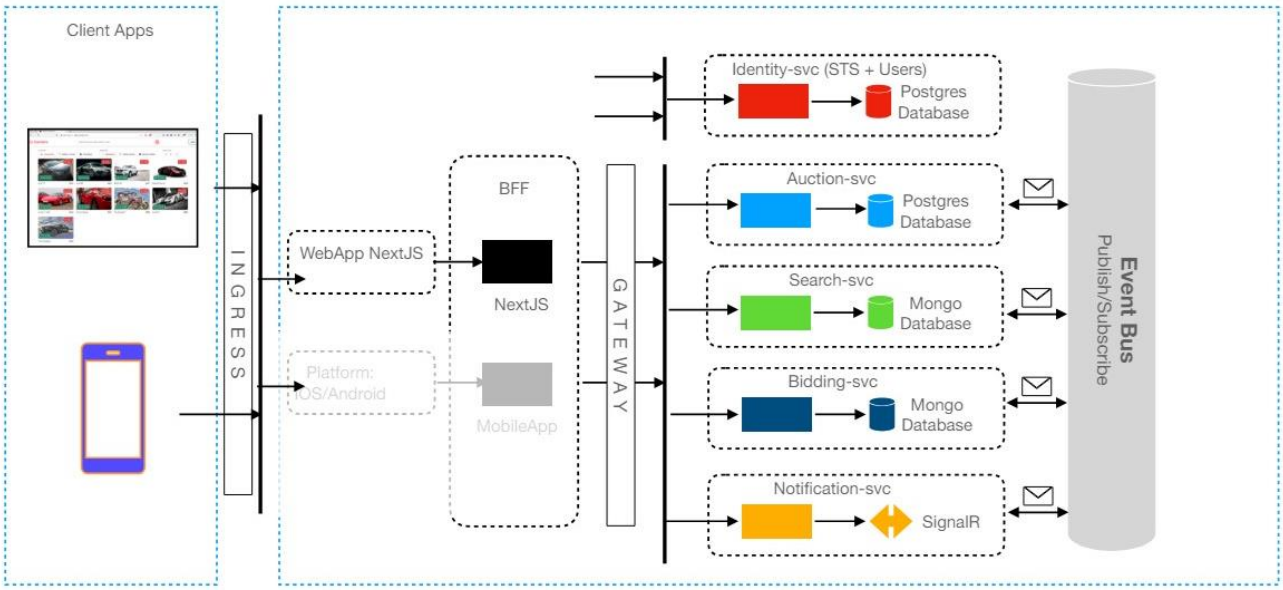

Рисунок 2.11 – Діаграма розгортання застосунку для купівлі автомобіля

Таблиця 2.3 – Опис компонентів діаграми розгортання

Компонент	**Опис**
Client – Web application React, NextJS	Вебзастосунок за допомогою якого можна зареєструватися в системі, створювати лоти, переглядати пропозиції, створені іншими користувачами, здійснювати пошук автомобіля або відстежувати недавно створені лоти.
Server – Identity microservice	Цей сервіс відповідає за автентифікацію й авторизацію користувачів. Також даний сервіс використовує протоколи автентифікації OIDC, OAuth 2.0 й security token service (STS) компонент, що видає й перевіряє токени.
Server – Auction microservice	Мікросервіс відповідає за CRUD-операції пов'язані з лотами. Загалом він відповідальний за отримання, а також за обробку запитів, які звернені до лотів, що містяться в БД.
Server – Search microservice	Використовується для пошуку лоту за його певними харктеристиками (назва, колір, виробник) і відображенням пошуковго запиту на сторінці.

Кінець таблиці 2.3

Server – Bidding microservice	Мікросервіс обробляє процес торгів, включаючи управління ставками, визначення їх статусу (bid accepted, reserve not met, bid was too low) відповідно до стану аукціона, повідомлення користувачам, суми останньої ставки, відображення історії ставок інших користувачів тощо.
Server – Notification microservice	Цей сервіс відповідає за надсилання сповіщень користувачам, наприклад, коли інший користувач розпочав або закінчив аукціон.
Server – PostgreSQL	Зберігає інформацію про користувачів і створені ними пропозиції стосовно купівлі авто.
Server – MongoDB	Зберігає інформацію про результати пошуку й історію ставок, що робили користувачі.
Server – RabbitMQ event bus	Використовується як брокер повідомлень між мікросервісами. Коли сервіс публікує подію, інші сервіси, які підписані на цю подію, можуть отримувати повідомлення.

Розподіл на окремі мікросервіси зменшить навантаження й дозволить працювати системі асинхронно завдяки черзі повідомлень.

Також крім мікросервісів на даній діаграмі розгортання теж присутні важливі елементи, про яких слід згадати. Ingress шлюз дозволяє централізувати управління трафіком і забезпечити безпеку мережевої комунікації, що водночас буде сприяти спрощенню маршрутизації запитів і дозволить здійснювати захист від можливих зовнішніх атак.

Для досягання ефективного й швидкого виконання запитів і комунікації між мікросервісами застосовується Docker. Він дозволяє створити ізольоване середовище для кожного компонента системи, що сприятиме швидкому розгортанню, масштабуванню й управлінню сервісами.

У даному проєкті використовується підхід backend for frontend (BFF). Його суть полягає у тому, що створюється окремий бекенд для кожного клієнтського інтерфейсу, наприклад, вебзастосунку або мобільного застосунку, що у свою чергу дозволяє оптимізувати комунікацію між клієнтським інтерфейсом і

мікросервісами. Також варто зазначити, що BFF дозволяє мінімізувати зайву мережево-пропускну здатність через його особливість агрегувати дані з різних мікросервісів і відсилати їх разом у вигляді одного запиту до клієнтського інтерфейсу. Використання цього півдходу значно підвищує зручність розробки індивідуальних рішень для різних типів клієнтів, забезпечуючи більш цільовий і ефективний підхід до обслуговування кожного користувача. Завдяки BFF буде зменшено кількість запитів між клієнтом і сервером, що сприятиме зменшенню обсягу мережевого трафіку і покращенню продуктивності.

Висновки до розділу 2

В другому розділі кваліфікаційної роботи бакалавра описано моделювання й застосунку для купівлі авто, спроєктовано механізми взаємодії мікросервісів із клієнтами. Приділено особливу увагу побудові діаграм діяльності, прецедентів, послідовності й розгортання. Для вказаних діаграм створено сценарії використання всіх типів (короткий, поверхневий, повний). Проведено огляд стеку використаних технологій фронтенд і бекенд частин системи, описано використані мови програмування, фреймворки, бібліотеки, бази даних і способи обміну даними з серверами.

Проведено аналіз діаграми розгортання, показано способи оптимізації навантаження системи завдяки розподілу на окремі мікросервіси й можливості зменшення навантаження на систему.

Описано шаблони взаємодії мікросервісів між собою і клієнтами. Продемонстровано підходи для консолідації даних між БД, асинхронної взаємодії й обробкою даних у реальному часі. Спроєктовано приклади функціонування частин системи, що реалізовують критерії необхідні для її стабільної роботи й кращого UX при використанні застосунку користувачами.

3 ПРОЄКТУВАННЯ МІКРОСЕРВІСНОГО ЗАСТОСУНКУ НА МОДУЛЬНІЙ ПЛАТФОРМІ .NET

3.1 Конструювання UML-діаграм

UML-діаграми представляють собою набір графічних нотацій [11], які використовуються для представлення різних аспектів програмних систем, таких як структура, поведінка і взаємодії. Вони зазвичай використовуються в розробці ПЗ, щоб допомогти розробникам візуалізувати й передати складні системи та їх дизайн.

Таблиця 3.1 – Опис основних UML-діаграм

Назва	Опис
Діаграма класів	Структурна діаграма, що відображає статичну структуру програмного забезпечення, включаючи класи, інтерфейси, атрибути, методи і зв’язки між ними.
Діаграма використання	Поведінкова діаграма, яка показує функціональні вимоги системи через взаємодію між акторами та випадками використання..
Діаграма послідовності	Поведінкова діаграма, що демонструє взаємодію між об’єктами або компонентами системи протягом певного часу, показуючи порядок обміну повідомленнями між ними.
Діаграма діяльності	Поведінкова діаграма, що представляє потік діяльності в системі, включаючи точки прийняття рішень, паралельні дії та шляхи розгалуження.
Діаграма компонентів	Структурна діаграма, яка показує компоненти, інтерфейси й залежності між ними, демонструючи організацію компонентів системи, їхню взаємодію.
Діаграма розгортання	Структурна діаграма, яка зображує фізичне розгортання компонентів і вузлів у системі, відображаючи розподіл компонентів в апаратній інфраструктурі.
Діаграма пакетів	Структурна діаграма, що відображає організацію пакетів та їхні зв’язки в системі, показуючи як елементи системи згруповані в пакети, а також як вони залежать один від одного.
Діаграма об’єктів	Структурна діаграма, яка відображає стан системи в певний момент часу, показуючи об’єкти та їхні зв’язки.

Під час моделювання мікросервісного застосунку в другому розділі кваліфікаційної роботи бакалавра показано діаграми прецедентів, взаємодії, діяльності й розгортання. На етапі проєктування у цьому розділі буде розроблено діаграми класів, компонентів, пакетів, станів і переходів.

3.1.1 Діаграма класів серверної частини застосунку

Діаграма класів є фундаментом для об’єктно-орієнтованого моделювання [12], оскільки саме вона визначає основні сутності і зв’язки між ними. На діаграмі класів класи прийнято зображувати у вигляді прямокутників із назвою класу зверху. Створена діаграма має 4 основних сутностей і 2 допоміжну.

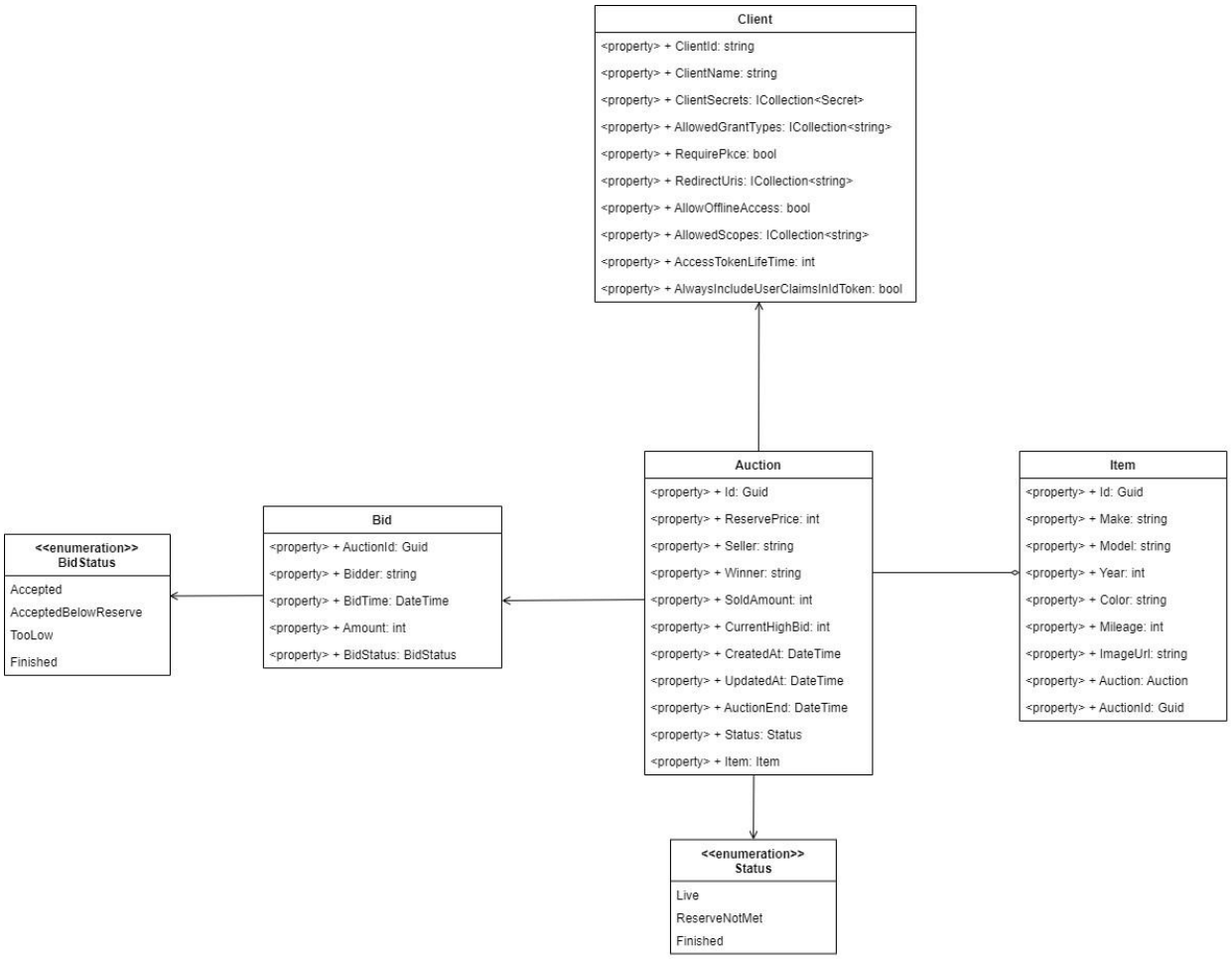

Рисунок 3.1 – Діаграма класів серверної частини

Варто зауважити, що представлені класи не містять методів і використовуються здебільшого для взаємодії з таблицями БД.

3.1.2 Діаграма компонентів серверної частини застосунку

Діаграми компонентів використовуються у розробці програмного забезпечення для візуалізації компонентів, інтерфейсів і залежностей системи. Це графічне представлення показує як різні частини програмної системи взаємопов’язані, як вони співпрацюють для реалізації функціональних можливостей застосунку.

Компонент на діаграмі відображає модульну одиницю системи, яка може бути програмним модулем, бібліотекою, підсистемою або будь-якою іншою логічною групою коду. На рисунку 3.2 зображена діаграма компонентів застосунку для покупки автомобілів, яка складається з вісьмох частин.

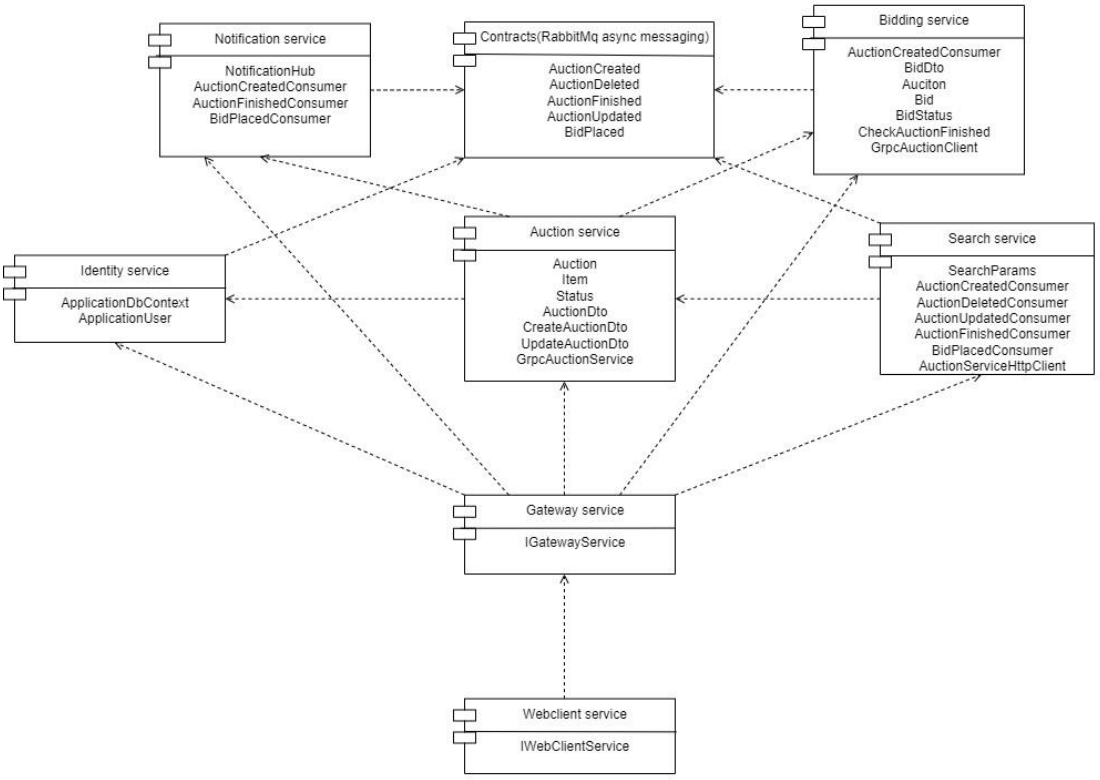

Рисунок 3.2 – Діаграма компонентів застосунку

Варто зауважити, що компонент може складатися з інших менших компонентів і може надавати або вимагати певний інтерфейс для зв’язку.

3.1.3 Діаграма пакетів

Діаграми пакетів використовуються у розробці програмного забезпечення для моделювання організації взаємозв'язків між пакетами, які представлені логічними групами пов'язаних класів, інтерфейсів й інших елементів системи. Це допомагає візуалізувати загальну структуру системи, зрозуміти взаємодію між її частинами [13].

На рисунку 3.3 представлена діаграма пакетів застосунку для онлайн-аукціону, яка включає 8 елементів.

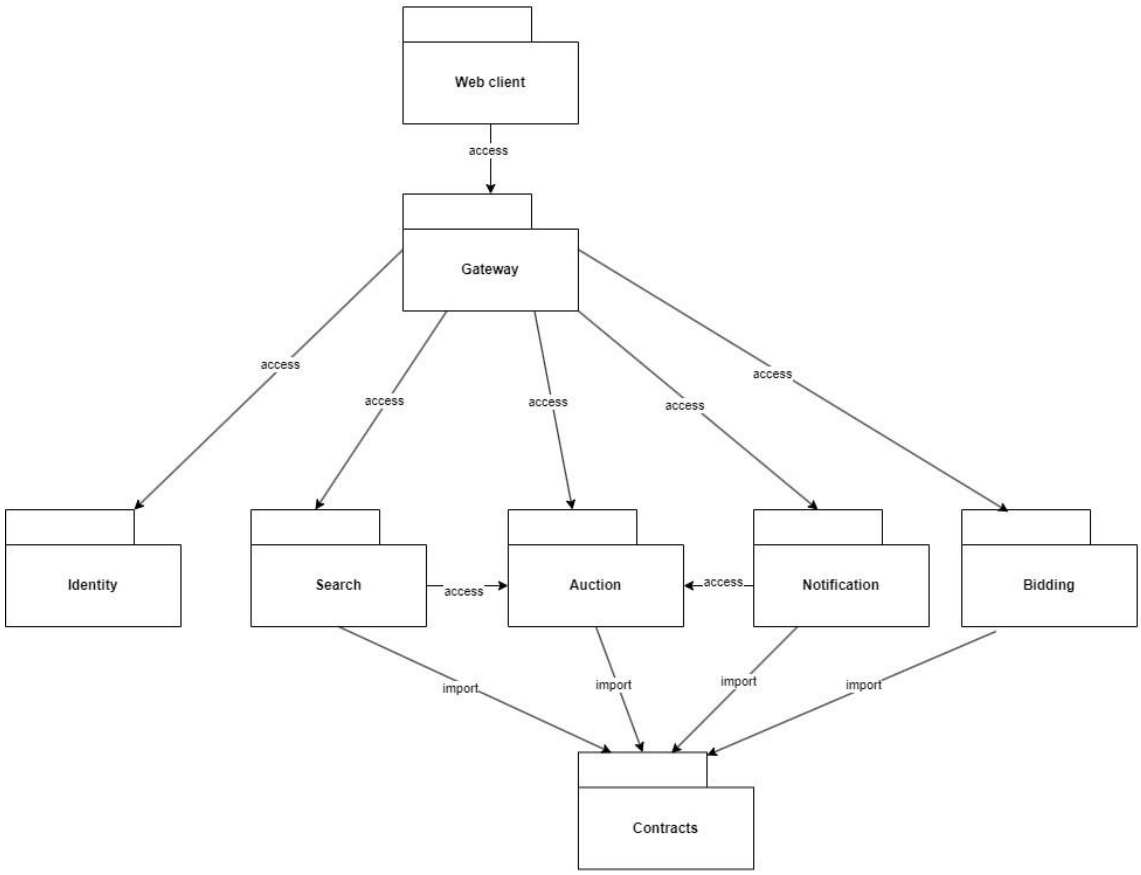

Рисунок 3.3 – Діаграма пакетів

Подібні діаграми особливо корисні для архітекторів ПЗ й розробників, щоб зрозуміти загальну структуру системи при групуванні або збірці її частин в логічні взаємозв'язані групи.

3.1.4 Діаграма станів і переходів

Діаграми станів і переходів показують різні стани, в яких може перебувати об’єкт або система, а також переходи між цими станами на основі різних подій або умов. Це є корисним для моделювання поведінки складних систем, таких як програмні застосунки або вбудовані системи, де поведінка системи залежить від її поточного стану і вхідних даних, які вона отримує.

Для майбутньої системи розроблено 3 діаграми станів і переходів:

1) загальна діаграма, демонструє можливості авторизованих користувачів (рис. 3.4);

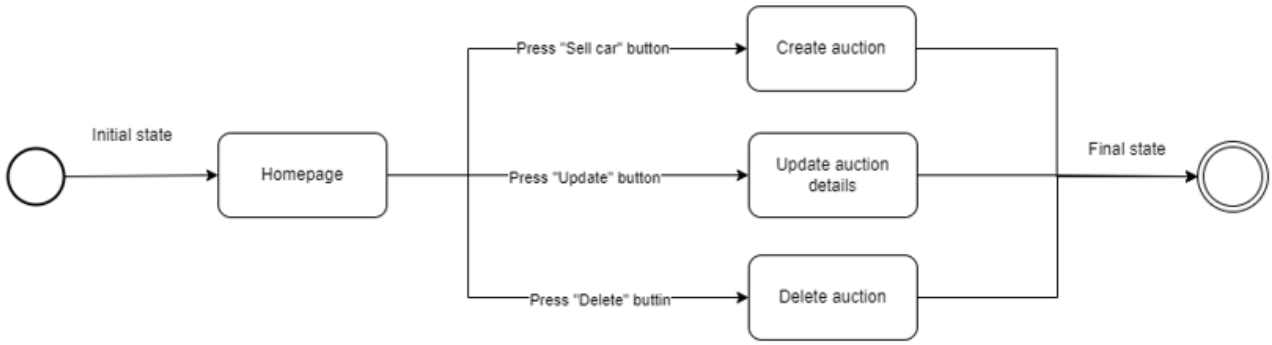

Рисунок 3.4 – Загальна діаграма станів і переходів

2) здійснення операції створення аукціону (рис. 3.5);

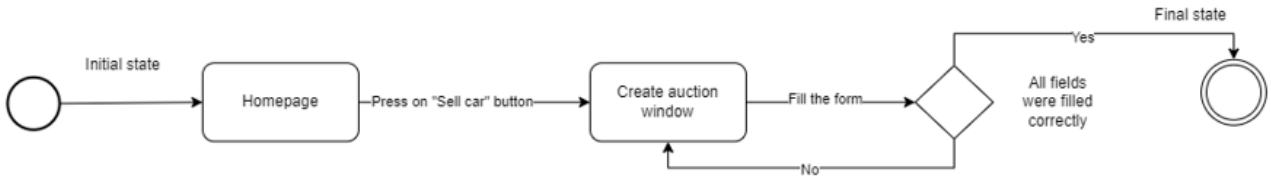

Рисунок 3.5 – Діаграма станів та переходів створення IP-фільтру

3) виконання команд на сторінці створеного аукціону (рис. 3.6).

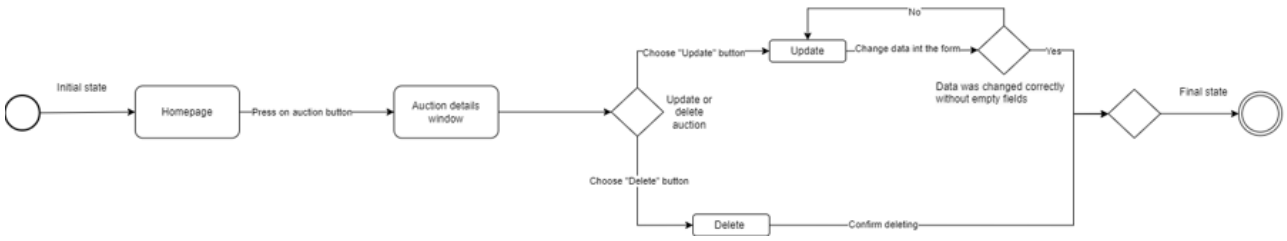

Рисунок 3.6 – Діаграма станів при зміні або видаленні аукіону

Діаграми станів є важливим інструментом для моделювання й аналізу розроблюваного застосунку на усіх етапах його реалізації. Також даний підхід може суттєво допомогти у покращенні створеної системи або її подальшої оптимізації.

3.2 Огляд стеку технологій

Стек технологій є ключовою складовою кожного застосунку, оскільки він напряму впливає на його продуктивність, безпеку і вимоги до апаратного забезпечення користувача. Для створення застосунку онлайн-аукціонів були використані такі технології (див. рис. 3.7):

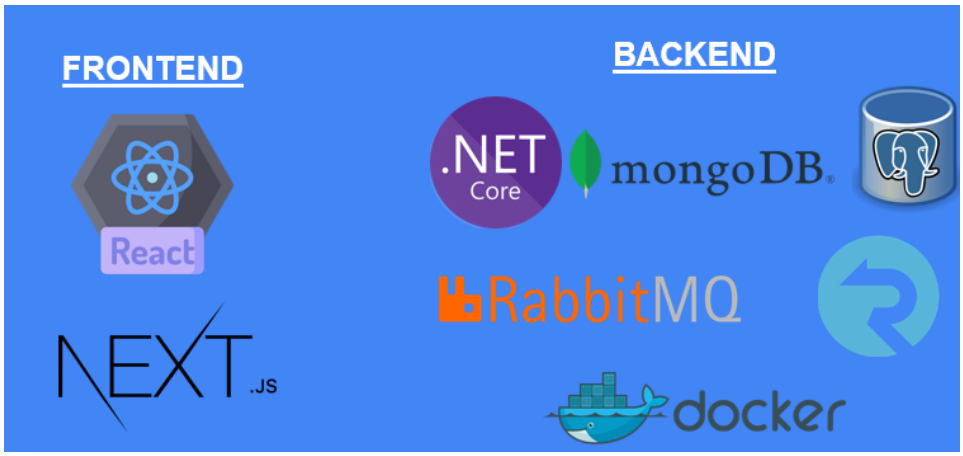

Рисунок 3.7 – Стек технологій застосунку онлайн-аукціонів

Під час вибору стеку технологій важливо усвідомлювати, що деякі мови програмування і фреймворки спеціалізуються на певних завданнях, можуть забезпечувати їх виконання з більшою швидкістю й ефективністю.

3.2.1 Використані мови програмування

C# є сучасною об'єктно-орієнтованою мовою програмування, створеною компанією Microsoft для її .NET платформи. Вона підтримує різноманітні парадигми програмування, включаючи імперативну, об'єктно-орієнтовану і функціональну. C# також має широкий набір вбудованих функцій і бібліотек, таких як LINQ, яка дозволяє розробникам здійснювати операції з колекціями даних зручним синтаксисом, схожим на SQL. Однією з ключових особливостей C# є підтримка асинхронного програмування, що дозволяє виконувати кілька завдань одночасно.

TypeScript представляє собою мову програмування, яка базується на JavaScript, доповнюючи його функціоналом і синтаксисом. Основна мета створення TypeScript полягає в розв'язанні проблем, що виникають при роботі з JavaScript, особливо у великих проєктах. Однією з ключових особливостей TypeScript є підтримка статичної типізації, що дозволяє визначати типи даних змінних і параметрів функцій під час компіляції, що полегшує виявлення помилок і підтримку коду. TypeScript компілюється в JavaScript, що дозволяє використовувати його на будь-яких платформах, що підтримують JavaScript. Останнім часом TypeScript набув популярності, особливо в області веброзробки, де він полегшує процес створення й доповнення великих програм.

C# і TypeScript добре поєднуються, тому що вони мають схожі риси. Вони обидві були розроблені Андерсом Хейлсбергом і багато в чому TypeScript виглядає як результат додавання частин C# до JavaScript. Вони мають схожі функції й синтаксис, тому їх легко використовувати разом в рамках одного проєкту чи команди без зайвих роздумів. Найважливіше те, що схожість C# і TypeScript дозволяє розробникам писати код подібним чином.

3.2.2 Технології розробки фронтенду

До основних інструментів розробки вебклієнту можна віднести React, Next.js. React є однією з найпопулярніших JavaScript бібліотек, яка використовується для створення інтерактивних і динамічних інтерфейсів. Частково успіх React полягає в тому, що він відносно байдужий до інших аспектів створення програм. Це призвело до процвітання екосистеми інструментів і рішень сторонніх розробників, включаючи Next.js. React надає корисні функції для створення інтерфейсу користувача, але залишає розробнику вибір, де використовувати ці функції у своїй програмі. Основні переваги використання React включають:

– складний віртуальний DOM, який дозволяє підтримувати високу продуктивність застосунків навіть при великій кількості компонентів;

– компонентну архітектуру, що спрощує розробку, тестування й підтримку коду;

– широкі можливості для створення компонентів і бібліотек, що можуть бути використані декілька разів;

– гнучку система маршрутизації, що дозволяє легко керувати шляхами й кінцевими точками в застосунках;

– широкий спектр сторонніх бібліотек і розширень для різних потреб розробників.

Next.js є фреймворком для розробки вебзастосунків на основі React, який надає додаткові можливості для побудови серверних і статичних застосунків. Деякі з ключових переваг використання Next.js включають:

– вбудована підтримка рендерингу на стороні сервера, що поліпшує SEO і швидкість завантаження сторінок;

– автоматичне код-розщеплення, що дозволяє завантажувати лише потрібні частини коду при переході між сторінками, що підвищує продуктивність застосунку;

– проста налаштовувана система маршрутизації та підтримка динамічних шляхів.

Parameters	React	Next JS
Definition	It is an open-source front-end JavaScript library.	It is an open-source web development framework.
Server-side Rendering (SSR)	It offers no support for SSR out-of-the-box.	It supports different types of SSR and incremental Static Generation.
Learning Curve	It is easier to learn.	Prior knowledge of React JS is a must to learn Next JS.
Configurability	Basic adjustments are needed for configurations.	Everything can be configured with ease.
Performance	Apps built with React display slow loading times.	Apps built with Next JS are faster as compared to React and Create React App (CRA).
Documentation	Solid and Up to date.	Solid and Up to date.
Community	Large and growing.	Small and evolving.
Developers	Finding experienced React JS developers is easy.	Finding experienced Next JS developers is relatively harder.
Use Case	It can be used to build UIs, SPAs, etc.	It can be used to build static websites, applications, etc.

Рисунок 3.8 – Особливості використаної бібліотеки і фреймворку

Для покращення UX були використані наступні допоміжні бібліотеки:

– Next Auth забезпечує автентифікацію для застосунків на TypeScript і JavaScript, підтримуючи різні провайдери (OAuth, Email, Credentials) з мінімальними налаштуваннями [14];

– React-countdown відображає таймер зворотнього відліку в React-застосунках, підтримуючи різні конфігурації і події про завершення відліку [15];

– Zustand – просте невелике сховище стану для React-застосунків з мінімалістичним API [16];

– React-hot-toast забезпечує легке управління спливаючих повідомлень для відображення коротких і ненав'язливих повідомлень користувачеві [17].

Окрему увагу слід приділити безпеці користувача при використанні застосунку. Це включає кілька ключових аспектів, зокрема шифрування даних. Шифрування даних є основним елементом для захисту конфіденційної інформації користувачів під час передачі і зберігання інформації, тому для забезпечення захищеності користувацьких даних було використано бібліотеку Next Auth, що програмно реалізовує протокол криптографічного шифрування SIGMA-I (рис. 3.9).

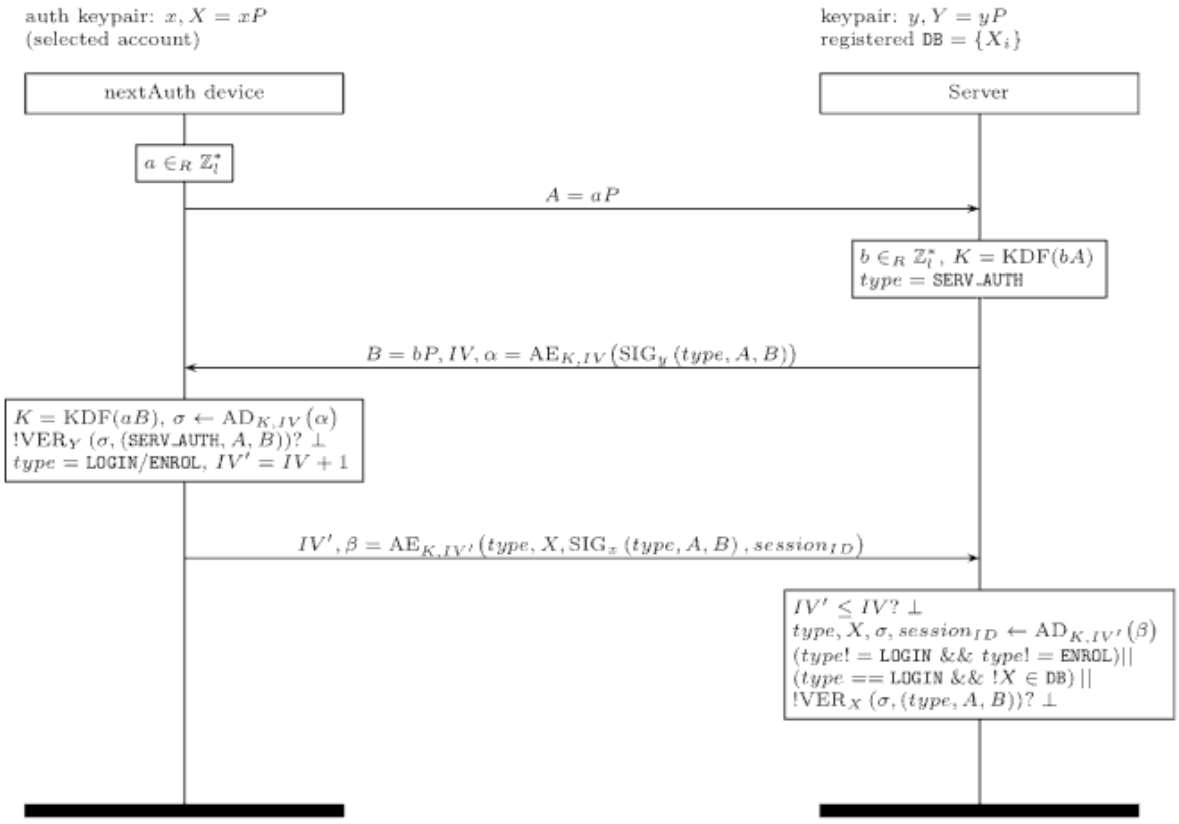

Рисунок 3.9 – Опис процесу шифрування SIGMA-I

Протокол SIGMA-I забезпечує високоефективну взаємну автентифікацію з узгодженням ключа з додатковою перевагою, що ПК може відкласти надсилання своєї ідентифікаційної інформації, поки не підтвердить ідентифікацію сервера. Спільний ключ K генерується за допомогою функції похідного ключа (KDF) на основі неавтентифікованої угоди ключа Діффі-Хеллмана [18]. Коли кожна сторона підписує (SIG і VER) обмінені значення Діффі-Хеллмана, підтверджується походження повідомлень.

Для підвищення ефективності протоколу використовується режим автентифікованого шифрування (AE та AD). Цей протокол застосовується для різних цілей, таких як вхід і реєстрація, з метою, закодованою у типі, який є частиною підписаного повідомлення. Оскільки ПК вже знає ідентифікатор сервера, це значення можна виключити з протоколу. Для ідентифікації ПК використовується його відкритий ключ. Перед тим як прийняти ПК, сервер також перевіряє, чи зареєстровані надані облікові дані в його базі даних. Ідентифікатор сеансу дозволяє серверу пов’язати свій сеанс nextAuth із сеансом інтерфейсу й сервера.

Для збереження станів було обрано бібліотеку Zustand, хоча його аналог Redux більше інтегрований з React. Для розуміння цього вибору слід дослідити особливості функціонування кожної бібліотеки. Redux – це класична бібліотека для керування станом, яка інтегрується з React і є найбільш популярною для збереження станів. Розглянемо архітектуру Redux, щоб зрозуміти його роботу.

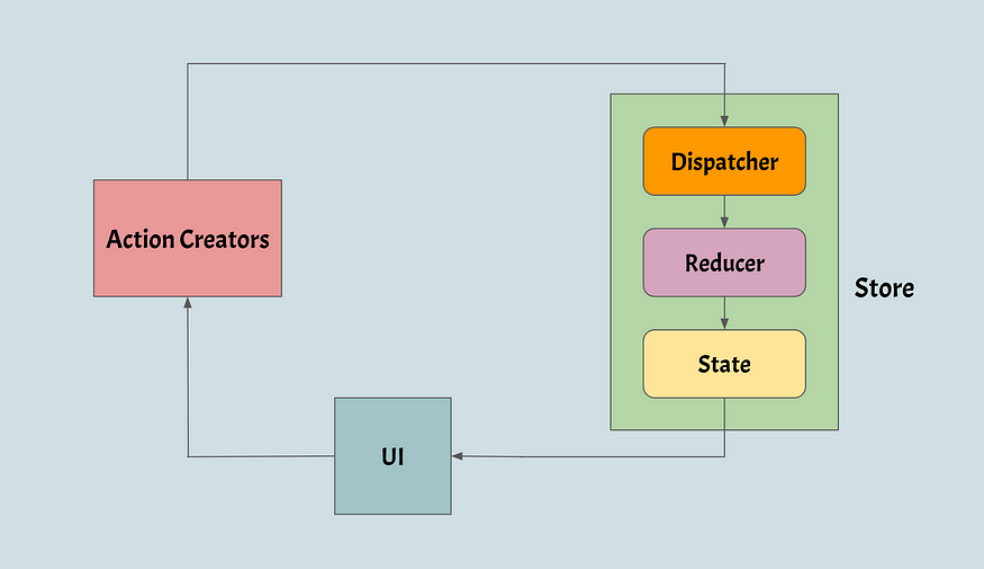

Рисунок 3.10 – Опис роботи бібліотеки Redux

Творці дій (action creators) забезпечують запуск відповідної дії для кожної взаємодії користувача. Дії можна розглядати як події, що описують те, що сталося в додатку, наприклад, натискання кнопки або виконання пошуку. Ці дії відправляються в сховище через диспетчери (dispatchers). Редуктори (reducers)

потім визначають, як змінити стан додатку. Функція редуктора бере поточний стан, об’єкт дії і на основі цього повертає новий стан. З оновленням стану відповідні зміни відображаються в інтерфейсі користувача.

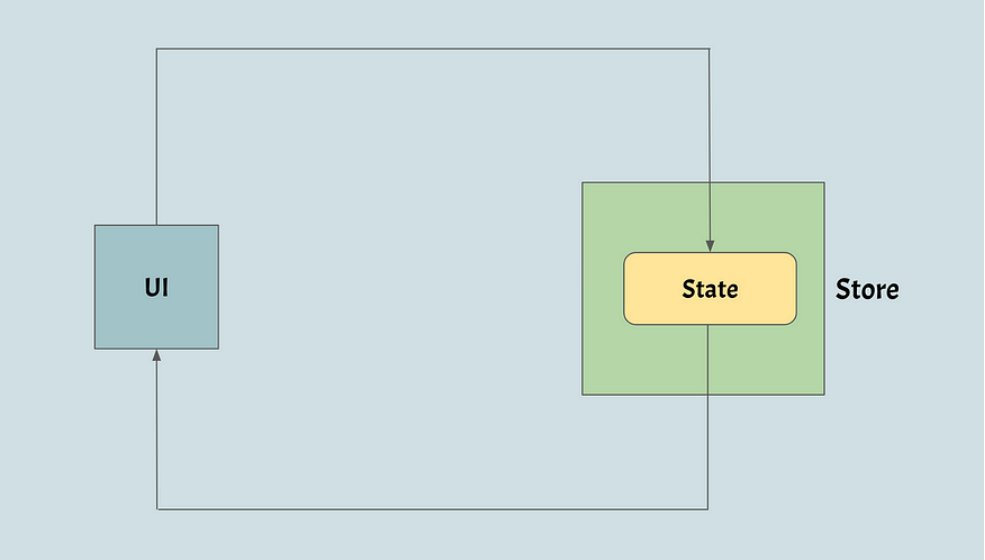

Рисунок 3.11 – Опис роботи бібліотеки Zustand

Тепер розглянемо, як працює Zustand. На схемі такод присутній компонент інтерфейсу користувача. Коли надходить запит на зміну, він направляється до сховища. Воно визначає, як оновити стан, і інтерфейс користувача відображається з оновленими даними, щойно сховище повертає новий стан. Відсутність творців дій, диспетчерів і редукторів у цій схемі робить Zustand більш спрощеним. Натомість він використовує функцію підписки на зміни стану, що дозволяє інтерфейсу користувача залишатися синхронізованим з даними.

Під час розробки буде використовуватися React для створення інтерфейсу користувача, а потім поступово застосовуватися функції Next.js для вирішення типових вимог застосунку, таких як маршрутизація і вибірка даних.

3.2.3 Технології розробки бекенду

Для створення великої бекенд частини проєкту використано безкоштовний фреймворк C# з відкритим вихідним кодом, відомий як ASP.NET. Цей фреймворк є незалежним від платформи, що означає можливість його використання для

розробки вебзастосунків, які працюватимуть на різних операційних системах, включаючи Windows, Linux і macOS. Серед основних переваг ASP.NET можна виділити швидкий розвиток, масштабованість і безпеку.

Рисунок 3.12 – Логотип ASP.NET Core

Для реалізації мікросервісу Identity, що відповідає за вхід у систему й реєстрацію нових користувачів, був використаний фреймворк для ASP .Net Core – Duende Identity Server. Duende IdentityServer – це проміжне програмне забезпечення, яке додає кінцеві точки OpenID Connect і OAuth 2.0, сумісні зі специфікаціями, до довільного хосту ASP.NET Core [19]. Як правило, створюється або повторно використовується програма, яка містить сторінки входу, виходу і додається проміжне програмне забезпечення IdentityServer до цієї частини застосунку (рис. 3.13).

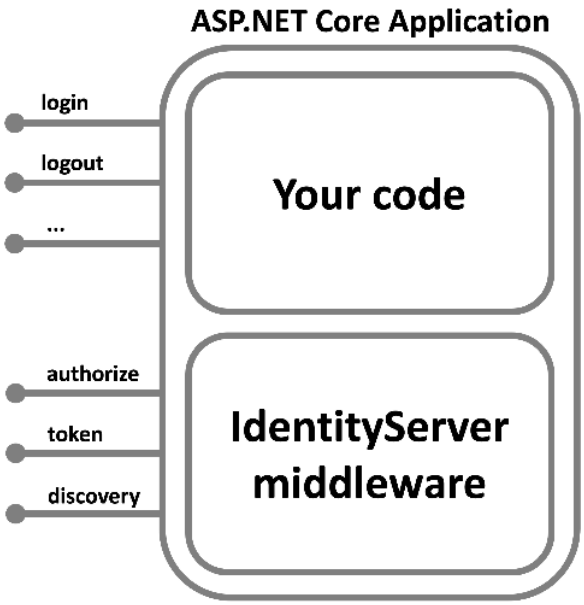

Рисунок 3.13 – Механізм роботи Duende IdentityServer

Проміжне програмне забезпечення додає до програми необхідні заголовки протоколів, щоб клієнти могли спілкуватися з нею за допомогою цих стандартних протоколів [20]. Також цей інструмент, окрім реалізації механізмів автентифікації й авторизації, пропонує готові рішення для фронтенду: вікна для входу, реєстрації, перегляду даних користувача, що увійшов в систему й інше.

Для розуміння нагальності використання наступної технології потрібно уважно дослідити частину розроблюваного застосунку. Призначення аукціонного сервісу полягає у виконанні базових CRUD операцій над сутностями Auction і Item. Задачею пошукового сервісу є здійснення пошуку товару, сортування і фільтрування результатів. Деякі підходи у розробці спрямовані на комунікації мікросервісів без і сервісам з БД для отримання даних, а вже потім повернення результатів у потрібному для користувачів вигляді.

У даному випадку ця стратегія не буде ефективною. Якщо звернути увагу, що для онлайн-аукціонів використовується реляційна БД, то можна зробити висновок, що дві сутності зберігатимуться в двох різних таблицях, тобто у такому випадку потрібно буде звернутися до Postgres, об’єднати дві сутності в одну, після чого повернути їх до пошукового мікросервісу задля обробки. Зрозуміло, що це вплине на продуктивність системи. Використання в MongoDb у Search сервісі дозволить:

– завдяки перевагам NoSql зберігати дані двох сутностей в одному документі;

– здійснювати пошук й брати участь в аукціонах навіть при проблемах із Auction мікросервісом.

У вищезазначеному другому пункті була частково піднята проблема відмовостійкості системи. Для забезпечення функціонування застосунку попри відмову в роботі того чи іншого мікросервісу використовується RabbitMQ.

RabbitMQ – це легкий і масштабований брокер повідомлень на основі AMQP [21]. RabbitMQ діє як посередник між різними сервісами. Він використовується для зменшення завантаження і часу доставки вебзастосунків

сервера шляхом делегування завдань, які зазвичай потребують багато часу або ресурсів. Робота RabbitMQ полягає в тому, щоб переконатися, що повідомлення від publisher(видавців) надходять до потрібних consumers(споживачів) [22].

Для зберігання й обробки даних використовано такі сховища:

1) PostgreSQL – це система керування базами даних (СКБД), яка відома своєю надійністю і високою продуктивністю. Вона є системою з відкритим вихідним кодом, що підтримує широкий спектр SQL-запитів і часто використовується для застосунків, які потребують складного моделювання даних і транзакцій;

2) MongoDB – це документо-орієнтована база даних, яка розроблена для гнучкості й масштабованості. Часто використовується для вебзастосунків та аналізу великих обсягів даних, оскільки вона легко обробляє великі обсяги неструктурованої інформації.

	Postgres	MongoDB
License	Postgres License (MIT alike)	SSPL
Data Model	Tabular, relational	Document
JSON Support	Capable and integrated with SQL	Built-in schema validator
Performance	Optimized for complex query	Optimized for de-normalized data
Reliability	Full ACID transaction support	Transaction + built-in auto failover
Scalability	Scale-up	Scale-out
Usability	Rigorous and powerful query capability	Relaxed enforcement and easy to start
Operability	Optimized for single node and a wide range of hosting providers	Optimized for multi-node and a polished hosting service
Ecosystem	Community driven, decentralized, vibrant	Business driven, centralized, integral

Рисунок 3.14 – Особливості використаних СКБД

PostgreSQL відзначається своєю високою надійністю, що стане у нагоді при зберіганні даних користувачів й аукціонів. MongoDB відома своєю гнучкістю й

масштабованістю, що робить її ідеальним вибором для вебзастосунків й обробки великих обсягів неструктурованої інформації в аналітичних задачах. Саме bid microservice, який буде відповідати за обробку ставок користувачів, буде зберігати стан аукціону в цій СКБД.

Висновки до розділу 3

В третьому розділі кваліфікаційної роботи бакалавра описано проєктування мікросервісного застосунку для здійснення покупок автомобіля й проведено огляд стеку використаних технологій для фронтенду й бекенду застосунку. Було побудовано діаграми класів, компонентів, пакетів, станів і взаємодій.

Описано використані мови програмування, фреймворки, бібліотеки, бази даних й особливості їх використання в залежності від поставленої задачі. Для фронтенду були обрані сучасні інструменти, що підтримують створення інтерактивних і динамічних інтерфейсів, а для бекенду обрані фреймворки, відомі своєю швидкодією, масштабованістю й безпекою. Досліджено впровадження рішень, що забезпечують високу продуктивність і надійність.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ МІКРОСЕРВІСНОГО ЗАСТОСУНКУ

Мікросервісний застосунок для купівлі автомобілів Auction містить окрім основних ще 3 сторінки (перегляд створених лотів, форма для продажі автомобіля, сторінка входу в систему тощо) для авторизованого користувача. Після етапу реалізації систему можна розгорнути як у локальній мережі, так і в Інтернеті, що дозволить надавати доступ користувачам до розроблюваного рішення у сегменті онлайн-аукціонів. Вебзастосунок створено для 3-х типів користувачів: продавець, покупець, анонімний користувач. Різниця між ролями продавця й покупця майже відсутня, оскільки їм обом доступний функціонал один одного за умови входу систему. Відповідно до сформованих вимог користувач має можливість виконувати CRUD-операції над сутностями власних автомобілів виставлених на продаж, оновлення їх даних, перегляду перебігу аукціону.

4.1 Огляд дизайну вебзастосунку

Для проєктування дизайну клієнтської частини застосунку використовувався онлайн-інструмент для створення mock-ups, «Canva». Для не авторизованого в системі користувача (або як вище згадувалося анонімного) доступна головна сторінка з переліком усіх лотів (рис. 4.1) з можливістю переглядати історію ставок, а також кнопки для входу в систему (рис. 4.2) й подальшої реєстрації (рис. 4.3).

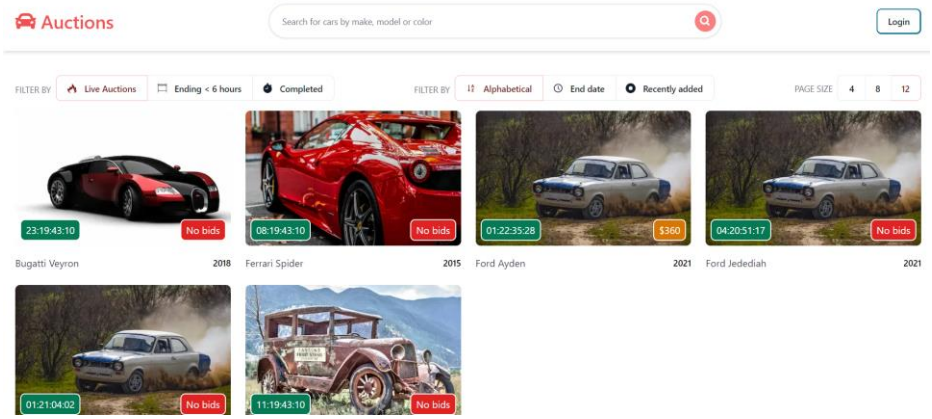

Рисунок 4.1 – Головна сторінка клієнтської частини мікросервісного застосунку

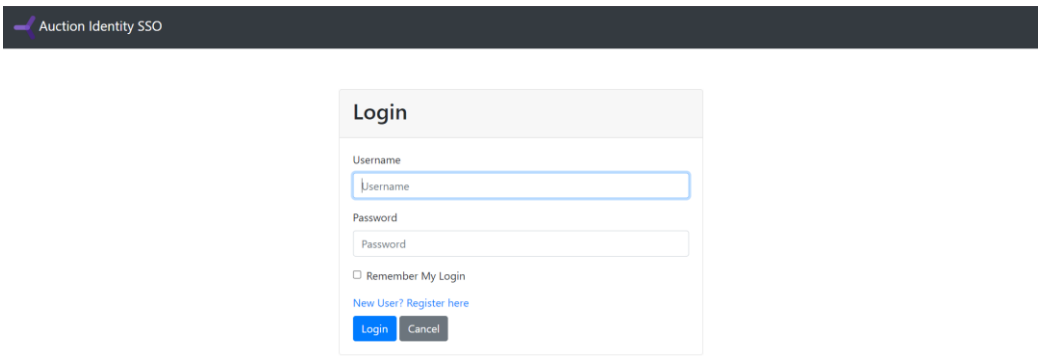

Рисунок 4.2 – Вікно входу в застосунок

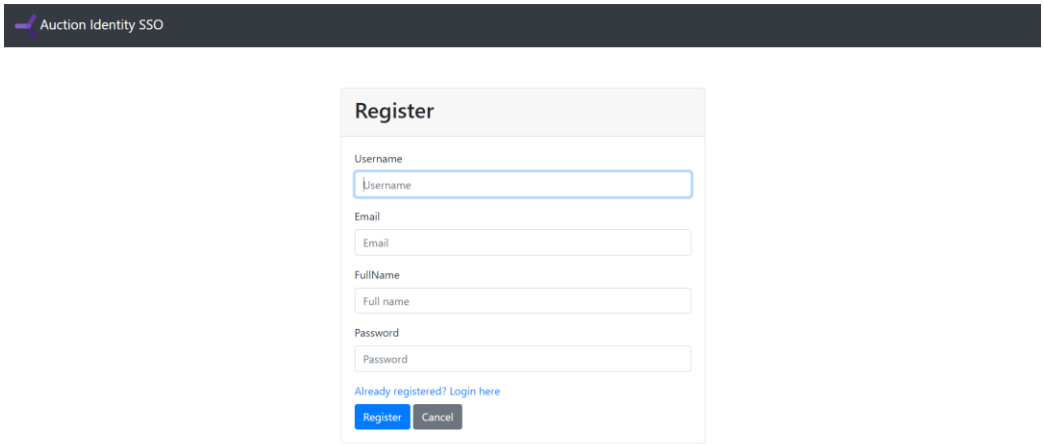

Рисунок 4.3 – Вікно реєстрації нового користувача

На сторінці реєстрації користувачу необхідно ввести особисті дані: username, електронну пошту, ім'я, прізвище й пароль. Крім того, є можливість отримувати сповіщення при створенні іншим користувачем лоту. Також на головній сторінці й інших вікнах розміщені назва й логотип застосунку, який одночасно є кнопкою, що веде на головну сторінку або відміняє обрані фільтри, якщо користувач при пошуку їх попередньо обрав.

Для авторизованого користувача доступно 3 додаткові сторінки в меню:

– My auctions – перегляд створених користувачем лотів (рис. 4.4);

– Auction won – список аукціонів, де користувач здобув перемогу (рис. 4.5);

– Sell my car – сторінка з формою, де зареєстрований користувач може створити аукціон, вказавши необхідні дані (рис. 4.6).

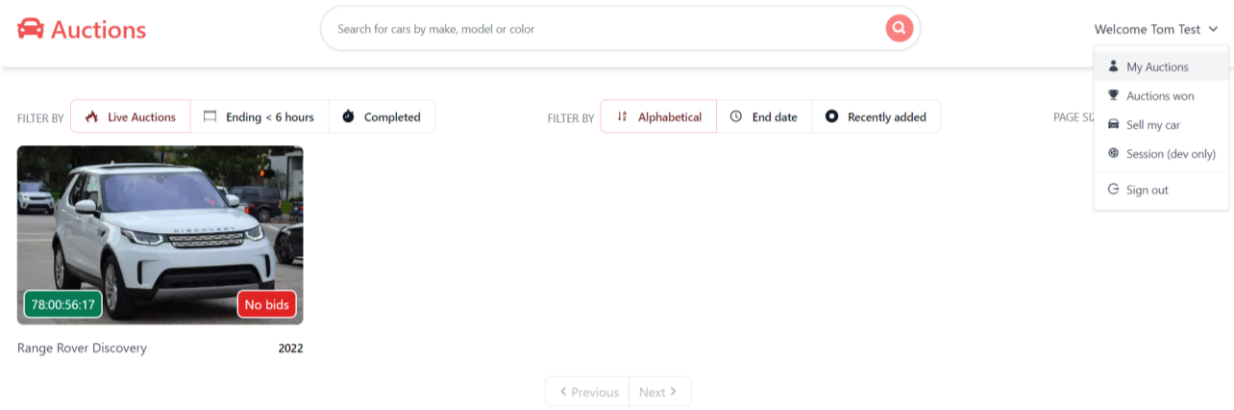

Рисунок 4.4 – Вікно зі списком створених користувачем лотів

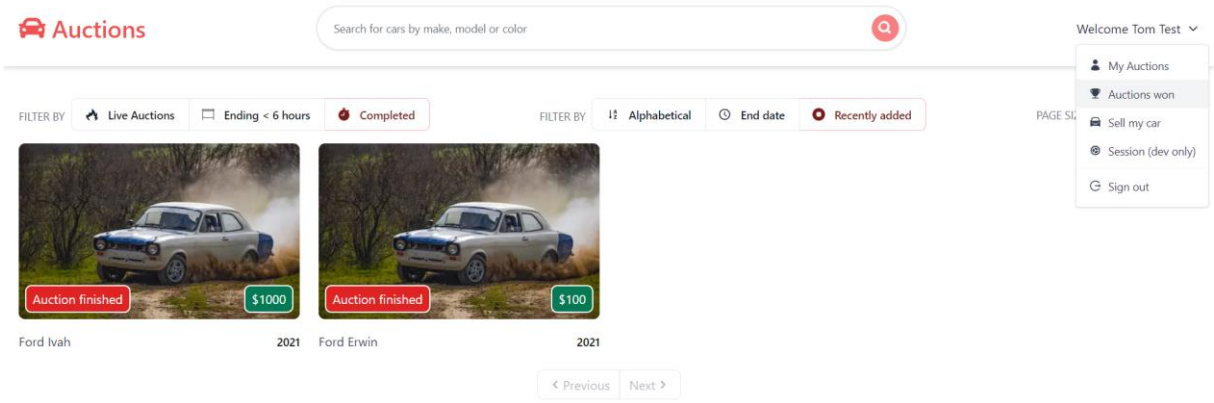

Рисунок 4.5 – Вікно з переліком аукціонів, де користувач переміг

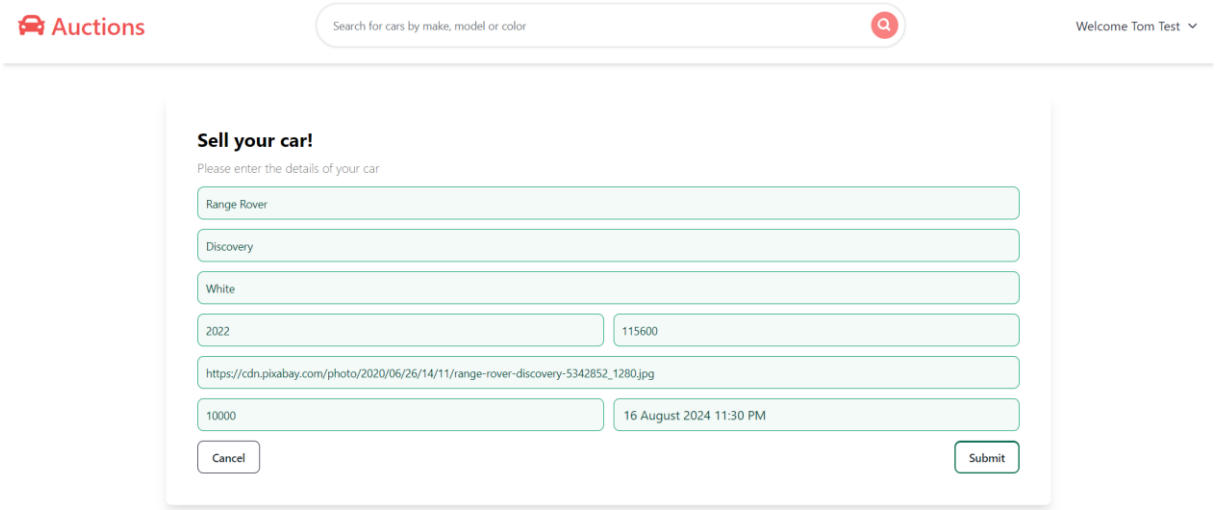

Рисунок 4.6 – Вікно для створення нового лоту

Також на головній сторінці для усіх ролей користувачів у застосунку

доступні функції для фільтрування аукціонів. Наприклад, можна фільтрувати пропозиції за часом:

– були недавно створені;

– опубліковані за останні шість годин;

– ті, що закінчилися.

Також присутня можливість сортування лотів за алфавітним порядком, датою їх закінчення або за категорією нещодавно доданих, також можна регулювати кількість відображуваних пропозицій на сторінці (рис 4.7).

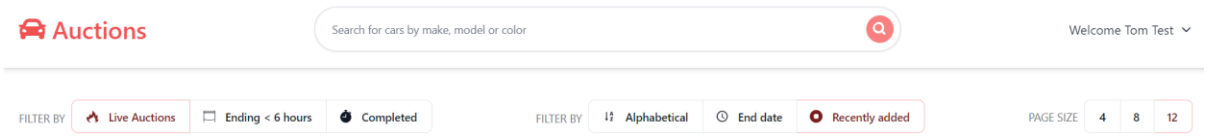

Рисунок 4.7 – Кнопки для вибору фільтрів

При створенні клієнтської частини застосунку і розробці дизайну основним завданням було створення функціонального й інтуїтивно зрозумілого середовища для створення лотів або участі в аукціонах.

4.2 Реалізація програмних компонентів бекенду мікросервісного застосунку для купівлі-продажу автомобілів

4.2.1 Реєстрація, авторизація і автентифікація користувачів

Розглянемо процеси, що будуть відбуватися у застосунку при вході до системи. Спочатку користувач натискає кнопку, його перенаправляють до сторінки входу на сервері аутентифікації, де перевіряється, чи він вже увійшов до системи. Після входу сервер аутентифікації надасть клієнтському застосунку ключ (рис 4.9). Клієнтський застосунок від імені користувача зможе використовувати функціональність, яка потребує аутентифікації.

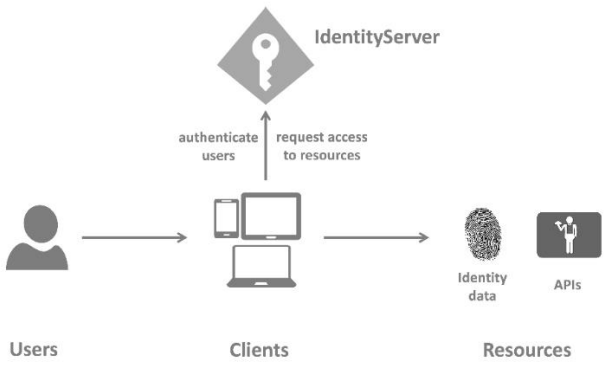

Рисунок 4.8 – Опис процесу входу до системи

Також не менш важливий підхід, який застосовується у даному мікросервісі, називається Single Sign-on (SSO). SSO – це метод контролю доступу, який просить користувача ввійти один раз і дозволяє йому отримати доступ до кількох ресурсів і служб після успішного входу без запиту на повторний вхід. Таким чином, підхід SSO дозволяє користувачам пройти автентифікацію лише один раз, а потім отримати легкий безпечний доступ до інших програм [23]. Загалом мікросервісий застосунок представляє собою окремі застосунки, які взаємодіють між собою і виконують певні функції, тому даний підхід конче потрібний для покращення UX.

Раніше було зазначено, що клієнтський застосунок від імені користувача може здійснювати певні операції. Для впровадження даного функціоналу у файлі *Config.cs* було внесені наступні зміни (рис 4.9):

```
20  public static IEnumerable<Client> Clients =>
21      new Client[]
22      {
23          new Client
24          {
25              ClientId = "postman",
26              ClientName = "Postman",
27              AllowedScopes = {"openid", "profile", "auctionApp"},
28              RedirectUris = {"https://www.getpostman.com/oauth2/callback"},
29              ClientSecrets = new[] {new Secret("NotASecret".Sha256())},
30              AllowedGrantTypes = {GrantType.ResourceOwnerPassword}
31          },
32          new Client
33          {
34              ClientId = "nextApp",
35              ClientName = "nextApp",
36              ClientSecrets = {new Secret("secret".Sha256())},
37              AllowedGrantTypes = GrantTypes.CodeAndClientCredentials,
38              RequirePkce = false,
39              RedirectUris = {"http://localhost:3000/api/auth/callback/id-server"},
40              AllowOfflineAccess = true,
41              AllowedScopes = {"openid", "profile", "auctionApp"},
42              AccessTokenLifetime = 3600*24*30,
43              AlwaysIncludeUserClaimsInIdToken = true
44          }
45      };
```

Рисунок 4.9 – Код класу *Config.cs*

Серед полів класу *Client* фреймворку Duende IdentityService слід виділити наступні:

– *ClientId* вказує унікальний ідентифікатор для клієнта, що дозволяє серверу ідентифікувати цей конкретний клієнт;

– *ClientName* – ім'я клієнта, яке може відображатися користувачам;

– *ClientSecrets* задає зашрифоване значення, завдяки якому клієнт буде використовувати аутентифікацію. В цьому випадку, слово secret хешується за допомогою алгоритму SHA-256;

– *AllowedGrantTypes* вказує, що клієнт може використовувати код авторизації і клієнтські облікові дані для отримання токенів;

– *RedirectUris* – адреса, куди сервер повинен перенаправити користувача після успішної авторизації;

– *AllowOfflineAccess* дозволяє клієнту отримувати токени оновлення, які можна використовувати для отримання нових токенів доступу без присутності користувача;

– *AlwaysIncludeUserClaimsInIdToken* вказує, що клейми користувача повинні завжди включатися в ID токен замість того, щоб клієнт запитував їх через endpoint користувача.

Тепер у кожному мікросервісі у файлі *Program.cs* потрібно зазначити, що буде використовуватися автентифікація. Також у классах-контроллерах будуть використані атрибути, які нададуть потужний метод зв’язування метаданих або декларативної інформації з кодом (збірками, підказками, методами, властивостями тощо) [25].

На наступному прикладі буде продемонстровано код, який дозволить використовувати функціонал вищеописаного фреймворку в аукціонному мікросервісі (рис 4.11).

```
40  builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
41      .AddJwtBearer(options =>
42      {
43          options.Authority = builder.Configuration["IdentityServiceUrl"];
44          options.RequireHttpsMetadata = false;
45          options.TokenValidationParameters.ValidateAudience = false;
46          options.TokenValidationParameters.NameClaimType = "username";
47      });
48
```

Рисунок 4.10 – Код класу *Program.cs*

Тепер Identity Server можна використовувати в різних мікросервісах. Він інтегрований з API і дозволяє виконувати аутентифікацію через OpenID Connect, OAuth 2.

4.2.2 CRUD-операції для аукціонів і їх зв’язок з брокером повідомлень

За CRUD операції для аукціонів відповідає два мікросервіси – Auction і Search. Кожен з цих мікросервісів використовує різні бази даних для проведення операцій з даними у першому сервісі СКБД Postgres використовує зв'язані таблиці для управління даними, тоді як другому сервісі MongoDb є сховищем документів, що зберігає аукціони або елементи як документи. Основні сутності, які використовуються для зв’язку з БД Postgres – це Auction і Item (рис 4.12–4.13).

```
25 references
public class Auction
{
    18 references
    public Guid Id { get; set; }
    13 references
    public int ReservePrice { get; set; } = 0;
    14 references
    public string Seller { get; set; }
    1 reference
    public string Winner { get; set; }
    2 references
    public int? SoldAmount { get; set; }
    3 references
    public int? CurrentHighBid { get; set; }
    0 references
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    2 references
    public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;
    11 references
    public DateTime AuctionEnd { get; set; }
    11 references
    public Status Status { get; set; }
    28 references
    public Item Item { get; set; }

    2 references
    public bool HasReservePrice() => ReservePrice > 0;
}
```

Рисунок 4.11 – Код класу Auction.cs

```
[Table("Items")]
15 references
public class Item
{
    0 references
    public Guid Id { get; set; }
    14 references
    public string Make { get; set; }
    12 references
    public string Model { get; set; }
    12 references
    public int Year { get; set; }
    12 references
    public string Color { get; set; }
    12 references
    public int Mileage { get; set; }
    10 references
    public string ImageUrl { get; set; }


    //nav properties
    [JsonIgnore]
    0 references
    public Auction Auction { get; set; }
    0 references
    public Guid AuctionId { get; set; }
}
```

Рисунок 4.12 – Код класу Item.cs

Уявімо ситуацію, що сервіс для пошуку вийшов з ладу, проте користувач виконав запит на створення аукціону. Зміни були внесені до бази даних Postgre, але не було внесено до Mongo, у такому випадку Auction service надсилає запит до непрацюючого Search service через брокер повідомлень і потрапляє до черги. RabbitMQ зберігає його доки сервіс не зможе виконати обробку, що гарантує виконання даного запиту. Коли ж робота сервісу відновлена, запит буде переданий і виконаний. Описаний механізм можна спостерігати у методі *CreateAuction* контролеру Auction service (рис. 4.13).

```
[Authorize]
[HttpPost]
0 references
public async Task<ActionResult<AuctionDto>> CreateAuction(CreateAuctionDto auctionDto)
{
    var auction = _mapper.Map<Auction>(auctionDto);

    auction.Seller = User.Identity.Name;

    _context.Auctions.Add(auction);

    var newAuction = _mapper.Map<AuctionDto>(auction);

    await _publishEndpoint.Publish(_mapper.Map<AuctionCreated>(newAuction));

    var result = await _context.SaveChangesAsync() > 0;

    if (!result) return BadRequest("Could not save changes to the DB");

    return CreatedAtAction(nameof(GetAuctionById),
        new {auction.Id}, newAuction);
}
```

Рисунок 4.13 – Код класу *AuctionsController.cs*

Цей підхід і послідовність виконання запитів присутня у всіх методах контролерів Auction і Search мікросервісів.

Також до сервісів, що виконують обробку аукціонів, належить Bidding service. У другому розділі згадувалося, що даний мікросервіс використовуватиме gRPC для швидкого двухстороннього зв'язку між клієнтом і сервером. Для використання даного протоколу в проєкті було створено файли .proto [25], що описують структуру користувацьких типів за допомогою спеціальної мови, яка містить стандартні типи даних (рис. 4.14).

```
1   syntax = "proto3";
2
3   option csharp_namespace = "AuctionService";
4
5   service GrpcAuction {
6       rpc GetAuction (GetAuctionRequest) returns (GrpcAuctionResponse);
7   }
8
9   message GetAuctionRequest {
10      string id = 1;
11  }
12
13  message GrpcAuctionModel {
14      string id = 1;
15      string seller = 2;
16      string auctionEnd = 3;
17      int32 reservePrice = 4;
18  }
19
20  message GrpcAuctionResponse {
21      GrpcAuctionModel auction = 1;
22  }
```

Рисунок 4.14 – Приклад .proto файлу авторизації

Для реалізації .proto файлу в C# і перевизначення методів потрібно створити новий C# клас, який успадковуватиме згенерований клас із .proto файлу й перевизначатиме методи базового класу, використовуючи ключове слово override.

```
1 rererence
public class GrpcAuctionService : GrpcAuction.GrpcAuctionBase
{
    2 references
    private readonly AuctionDbContext _dbContext;

    0 references
    public GrpcAuctionService(AuctionDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    3 references
    public override async Task<GrpcAuctionResponse> GetAuction(GetAuctionRequest request,
        ServerCallContext context)
    {
        Console.WriteLine("==> Received Grpc request for auction");

        var auction = await _dbContext.Auctions.FindAsync(Guid.Parse(request.Id))
            ?? throw new RpcException(new Status(StatusCode.NotFound, "Not found"));

        var response = new GrpcAuctionResponse
        {
            Auction = new GrpcAuctionModel
            {
                AuctionEnd = auction.AuctionEnd.ToString(),
                Id = auction.Id.ToString(),
                ReservePrice = auction.ReservePrice,
                Seller = auction.Seller
            }
        };

        return response;
    }
}
```

Рисунок 4.15 – Приклад реалізації .proto файлу для отримання списку аукціонів

Зазвичай подібний підхід використовується для визначення API та міжпроцесного зв’язку в розподілених системах.

4.2.3 Механізми сповіщення користувачів і маршрутизація запитів

Gateway service – це мікросервіс, що відповідає за переадресацію запитів від клієнта до бекенду застосунку. Також даний компонент системи забезпечить зручність користування завдяки Reverse proxy. У мікросервісах зазвичай використовують реверсивний проксі, оскільки вони надають єдину точку доступу для запитів. Клієнт повинен знати лише одну URL-адресу, щоб отримати доступ до бекенду – адресу шлюзу(gateway). Йому не потрібно знати IP-адреси сервісу аукціону чи сервісу пошуку. Йому потрібна лише адреса шлюзу, а не IP-адреси сервісу аукціону чи сервісу пошуку. Для роботи з Reverse proxy було використано бібліотеку YARP, створену компанією Microsoft [26]. Усі адреси для маршрутизації визначені у JSON файлі *appsettingsDevelopment* (рис. 4.15).

```
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Information"
    }
  },
  "IdentityServiceUrl": "http://localhost:5000",
  "ReverseProxy": {
    "Clusters": {
      "auctions": {
        "Destinations": {
          "auctionApi": {
            "Address": "http://localhost:7001"
          }
        }
      },
      "search": {
        "Destinations": {
          "searchApi": {
            "Address": "http://localhost:7002"
          }
        }
      },
      "bids": {
        "Destinations": {
          "bidApi": {
            "Address": "http://localhost:7003"
          }
        }
      },
      "notifications": {
        "Destinations": {
          "notifyApi": {
            "Address": "http://localhost:7004"
          }
        }
      }
    }
```

Рисунок 4.15 – Файл конфігурації з маршрутами для обробки запитів мікросервісами

У другому розділі розглядався сценарій можливої взаємодії між Bid і Notification сервісами. Для нагальної демонстрації роботи надсилання сповіщень при створені іншим користувачем аукціону треба звернутися до коду *NotificationHub.cs* і *AuctionCreatedConsumer.cs* (рис 4.16–4.17).

```
using Microsoft.AspNetCore.SignalR;

namespace NotificationService;

7 references
public class NotificationHub: Hub
{

}
```

Рисунок 4.16 – Код класу *NotificationHub.cs*

```
2 references
public class AuctionCreatedConsumer : IConsumer<AuctionCreated>
{
    2 references
    private readonly IHubContext<NotificationHub> _hubContext;

    0 references
    public AuctionCreatedConsumer(IHubContext<NotificationHub> hubContext)
    {
        _hubContext = hubContext;
    }

    0 references
    public async Task Consume(ConsumeContext<AuctionCreated> context)
    {
        Console.WriteLine("--> auction created message received");

        await _hubContext.Clients.All.SendAsync("AuctionCreated", context.Message);
    }
}
```

Рисунок 4.17 – Код класу *AuctionCreatedConsumer.cs*

У результаті виконання методу *Consume* класу *AuctionCreatedConsumer* буде надіслано повідомлення про публікацію нового аукціону. У наступних розділах буде описано процес сповіщення користувача на клієнтській частині застосунку.

4.3 Реалізація програмних компонентів frontend частини мікросервісного застосунку для купівлі-продажу автомобілів

У попередньому розділі було описано функції Gateway service, тому надсилання запитів до бекенду значно полегшується. Уся логіка виконання запитів поміщена в fetch wrapper [27] із назвою *auctionActions.ts* (рис 4.18), де зображено частину методів що звертаються до backend. Усі CRUD операції, що можуть виконуватися над сутностями, звертаються за певною адресою зазначеною в Gateway service.

```
export async function getData(query: string): Promise<PagedResult<Auction>> {

    return await fetchWrapper.get(`search/${query}`)
}

export async function updateAuctionTest() {

    const data = {

        mileage: Math.floor(Math.random() * 100000) + 1

    }
    return await fetchWrapper.put('auctions/afbee524-5972-4075-8800-7d1f9d7b0a0c', data);
}

export async function createAuction(data: FieldValues) {

    return await fetchWrapper.post('auctions', data);

}

export async function getDetailedViewData(id: string): Promise<Auction> {

    return await fetchWrapper.get(`auctions/${id}`);

}
```

Рисунок 4.18 – Код класу *auctionActions.ts*

Також під час створення фронтенду використовувалися React-компоненти, такі як custom hooks. Наприклад, один із них виконує функцію відображення перебігу аукціону (рис 4.19). За допомогою його функціоналу можна приймати нові ставки додаючи їх до масиву і в подальшому відображувати на екрані історії ставок.

```
import { Bid } from "../types"
import { create } from "zustand"

type State = {
    bids: Bid[]
    open: boolean
}

type Actions = {
    setBids: (bids: Bid[]) => void
    addBid: (bid: Bid) => void
    setOpen: (value: boolean) => void
}

export const useBidStore = create<State & Actions>((set) => ({
    bids: [],
    open: true,

    setBids: (bids: Bid[]) => {
        set(() => ({
            bids
        }))
    },

    addBid: (bid: Bid) => {
        set((state) => ({
            bids: !state.bids.find(x => x.id === bid.id) ? [bid, ...state.bids] : [...state.bids]
        }))
    },

    setOpen: (value: boolean) => {
        set(() => ({
            open: value
        }))
    }
}))
```

Рисунок 4.19 – Код класу *useBidStore.ts*

У третьому розділі згадувалися додаткові бібліотеки для покращення роботи UI. Вони були обрані через ряд переваг. Наприклад, у бібліотеці React-countdown є можливість визначення подій завершення відліку, що дозволяє створювати інтерактивні функції, такі як сповіщення користувача або автоматичне виконання дій. У свою чергу React-hot-toast пропонує широкі можливості кастомізації, дозволяючи налаштувати вигляд і поведінку повідомлень відповідно до власних потреб.

```tsx
'use client'

import { useBidStore } from '../hooks/useBidStore';
import { usePathname } from 'next/navigation';
import React from 'react'
import Countdown, { zeroPad } from 'react-countdown';

type Props = {
  auctionEnd: string;
}

const renderer = ({ days, hours, minutes, seconds, completed }:
  { days: number, hours: number, minutes: number, seconds: number, completed: boolean }) => {
  return (
    <div className={`
                  border-2
                  border-white
                  text-white py-1 px-2
                  rounded-lg flex justify-center
                  ${completed ?
        'bg-red-600' : (days === 0 && hours < 10)
          ? 'bg-amber-600' : 'bg-green-600'}
            `}>
      {completed ? (
        <span>Auction finished</span>
      ) : (
        <span suppressHydrationWarning={true}>
          {zeroPad(days)}:{zeroPad(hours)}:{zeroPad(minutes)}:{zeroPad(seconds)}
        </span>
      )}
    </div>
  )
};
```

Рисунок 4.20 – Код класу *CountDownTimer.tsx* із використанням бібліотеки react-countdown

```tsx
import { placeBidForAuction } from '@/app/actions/auctionActions';
import { numberWithCommas } from '@/app/lib/numberWithComma';
import { useBidStore } from '../../../hooks/useBidStore';
import React from 'react'
import { FieldValues, useForm } from 'react-hook-form';
import { toast } from 'react-hot-toast';

export default function BidForm({ auctionId, highBid }: Props) {
    const {register, handleSubmit, reset, formState: {errors}} = useForm();
    const addBid = useBidStore(state => state.addBid);

    function onSubmit(data: FieldValues) {
        if (data.amount <= highBid) {
            reset();
            return toast.error('Bid must be at least $' + numberWithCommas(highBid + 1))
        }

        placeBidForAuction(auctionId, +data.amount).then(bid => {
            if (bid.error) throw bid.error;
            addBid(bid);
            reset();
        }).catch(err => toast.error(err.message));
    }

    return (
        <form onSubmit={handleSubmit(onSubmit)} className='flex items-center border-2 rounded-lg py-2'>
            <input
                type="number"
                {...register('amount')}
                className='input-custom text-sm text-gray-600'
                placeholder={`Enter your bid (minimum bid is $${numberWithCommas(highBid + 1)})`}
            />
        </form>
    )
}
```

Рисунок 4.21 – Код класу *BidForm.tsx* із використанням бібліотеки react-hot-toast

Ці бібліотеки допомагають ефективно створювати і покращувати React вебзастосунки, надаючи широкий спектр інструментів для реалізації основних функціональних можливостей, таких як аутентифікація, керування станом, відображення інтерфейсу користувача та інші, що спрощує розробку та покращує користувацький досвід.

Висновки до розділу 4

В четвертому розділі кваліфікаційної роботи бакалавра наведено програмну реалізацію ключового функціоналу мікросервісного застосунку купівлі-продажу автомобілей. Проведено огляд дизайну вебклієнту, функціоналу, інструментів і рішень для усунення можливих проблем.

Розглянуто бекенд застосунку для моніторингу пропозицій для купілві-продажу авто на мові програмування C# і детально описано способи зберігання даних в подібних системах. Закріплено навички написання запитів до реляційних і постреляційних БД. Розглянуто синтаксис написання .proto контрактів і продемонстровано їх реалізацію на мові програмування C#.

Описано бібліотеки і підходи для створення зручного вебклієнту і покращення UX шляхом впровадження мінімалістичного інтерфейсу, забезпечення швидкого зв’язку з компонентами системи, що знаходяться на бекенді.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра автоматизовано процес покупки автомобілів за рахунок створення відповідного програмного забезпечення.

Для досягнення поставленої мети було виконано наступні завдання:

– було проаналізовано предметну область, методологію та існуючі розробки;

– було проаналізовано аналогічні застосунки;

– було проаналізовано існуючі технології розробки програмного забезпечення;

– було визначено необхідний функціонал застосунку;

– було розроблено бекенд частину мікросервісного застосунку із використанням технологій .NET, RabbitMQ, PostgreSql, MongoDB, SignalR;

– було розроблено фронтенд частину мікросервісного застосунку із використанням технологій Next.js, React.

Проведено аналіз існуючих альтернативних додатків, висвітлено їх переваги й недоліки, розглянуто архітектуру, мови програмування, функціональні можливості, користувацький інтерфейс. Визначено вимоги до програмного забезпечення та здійснено моделювання додатку за допомогою різних UML-діаграм.

Описано способи оптимізації відмовостійкості системи завдяки розподілу на окремі мікросервіси з використанням брокера повідомлень, розібрано переваги і недоліки різних підходів. Продемонстровано архітектуру застосунку і приділено особливу увагу процесам, пов’язаним з покращенням UX.

Проведено вибір стеку технологій на основі вимог до ПЗ і розроблено всі частини застосунку. При розробці застосунку враховано можливі проблеми, пов’язані з відмовостійкістю або швидкістю роботи.

Розробка мікросервісної архітектури дозволила забезпечити гнучкість і масштабованість застосунку. Кожен мікросервіс відповідає за окрему бізнес-

логіку, що в майбутньому спростить процес подальшого оновлення застосунку або додавання нового функціоналу. Використання RabbitMQ дозволило ефективно керувати обміном повідомлень між мікросервісами, забезпечуючи їхню взаємодію та синхронізацію. Завдяки застосуванню PostgreSQL і MongoDB для зберігання різних типів даних було досягнуто високої продуктивності, надійності системи при виникненні проблем у роботі одного або декількох сервісів. SignalR забезпечив оновлення даних у реальному часі на вебклієнті, що покращило користувацький досвід. Вибір технологій .NET для бекенду і React з Next.js для фронтенду дозволив створити сучасний і зручний інтерфейс для користувачів, зацікавлених у купівлі-продажу авто за вигідною ціною.

Розробка мікросервісів забезпечила модульність системи, що дозволить легко оновлювати окремі компоненти без впливу на всю систему. У майбутньому для покращення роботи застосунку може бути використана контейнеризація, що спростить розгортання і застосунку, забезпечуючи швидкий запуск нових сервісів. Також це архітектурне рішення полегшує впровадження автоматизованих CI/CD процесів, що дозволять швидко інтегрувати нові функції та зміни. Застосування мікросервісів дозволило легко адаптувати систему до змін ринкових умов і вимог користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Analog application: eBay Motors Car marketplace. URL: www.ebay.com/b/Auto-Parts-and-Vehicles/6000 (Last accessed: 05.05.2024).

2. Analog application: IAAI marketplace. URL: www.iaai.com (Last accessed: 05.05.2024).

3. Analog application: Copart Car marketplace. URL: www.copart.com (Last accessed: 05.05.2024).

4. Communication in a microservice architecture. URL: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture com (Last accessed: 05.05.2024).

5. Warazuhudien A. Fire and Forget. URL: https://medium.com/mandiri-engineering/fire-and-forget-e59b745c9f97 (Last accessed: 05.05.2024).

6. Introduction to SignalR. URL: https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr (Last accessed: 05.05.2024).

7. Muditya R. The Future of Payment Systems: Transitioning from REST to gRPC for Improved Efficiency and Security. *The Future of Payment Systems*. 2023. № 1. P. 21–27. DOI: 10.5281/zenodo.7765404.

8. What is Use Case Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (Last accessed: 10.05.2024).

9. What is Activity Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/ (Last accessed: 10.05.2024).

10. UML - Interaction Diagrams. URL: https://www.tutorialspoint.com/uml/uml_interaction_diagram.htm (Last accessed: 10.05.2024).

11. Blokdyk G. UML Tool. A Complete Guide – 2020 Edition. Canada, Toronto : 5STARCooks, 2021. 303 p. ISBN: 978-1867406563.

12. Noback M. Object Design Style Guide. USA, New York : Manning, 2019. 288 p. ISBN: 978-1638350194.

13. Package Diagram Introduction, Elements, Use Cases and Benefits. URL: https://www.geeksforgeeks.org/package-diagram-introduction-elements-use-cases-and-benefits/ (Last accessed: 10.05.2024).

14. Introduction about Next Auth.js. URL: https://next-auth.js.org/getting-started/introduction (Last accessed: 17.05.2024).

15. React <Countdown />. URL: https://www.npmjs.com/package/react-countdown (Last accessed: 21.05.2024).

16. Introduction. How to use Zustand. URL: https://docs.pmnd.rs/zustand/getting-started/introduction (Last accessed: 21.05.2024).

17. What’s new in react-hot-toast 2.0. URL: https://react-hot-toast.com/docs/version-2 (Last accessed: 21.05.2024).

18. Him I. Securing the Digital Realm: Harnessing the Efficiency of Diffie-Hellman Key Exchange for Internet Security. 2024. P. 1–10. URL: https://www.researchgate.net/publication/379651565_Securing_the_Digital_Realm_Harnessing_the_Efficiency_of_Diffie-Hellman_Key_Exchange_for_Internet_Security (Last accessed: 21.05.2024).

19. Moghaddam M. T., Pedersen A., Bolding W., Worm T. A Performant and Secure Single Sign-On System Using Microservices. *The 38th ACM/SIGAPP Symposium On Applied Computing (SAC23)*. Tallinn, Estonia, Mar. 27–31, 2023. P. 1516–1519. DOI: 10.1145/3555776.3577869.

20. The big Picture. URL: https://docs.duendesoftware.com/identityserver/v7/overview/big_picture/ (Last accessed: 21.05.2024).

21. Bhat P., Priya D. Modern Messaging Queues – RabbitMQ, NATS and NATS Streaming. *International Journal of Recent Technology and Engineering (IJRTE)*. 2020. Vol. 9, Is. 2. P. 402–408. DOI: 10.35940/ijrte.B3551.079220.

22. Ćatović A., Buzađija N., Lemeš S. Microservice development using RabbitMQ message broker. *Science, Engineering and Technology*. 2022. Vol. 2, No. 1. P 30–37. DOI: 10.54327/set2022/v2.i1.19.

23. Westers M., Wich T., Jannett L., Mladenov V., Mainka Ch., Mayer A. SSO-Monitor: Fully-Automatic Large-Scale Landscape, Security, and Privacy Analyses of Single Sign-On in the Wild. Feb. 2023 (Preprint). DOI: 10.48550/arXiv.2302.01024.

24. Braga L., Lima Ph., Guerra E., Meirelles P. Attribute Sniffer: Collecting Attribute Metrics for C# Code. *X Conferência Brasileira de Software: Teoria e Prática (CBSoft 2019)*. Salvador, Brasil. Sept. 25, 2019. P. 96–101. DOI: 201996-101. 10.5753/cbsoft_estendido.2019.7664.

25. Clendon S. Protobuf. What is it, why you should care, and when should you use it? URL: adaptiv.nz/protobuf-what-is-it-why-you-should-care-and-when-should-you-use- it (Last accessed: 28.05.2024).

26. Getting Started with YARP. URL: https://microsoft.github.io/reverse-proxy/articles/getting-started.html (Last accessed: 28.05.2024).

27. Fetch Wrapper. URL: https://demirels-organization.gitbook.io/javascript-tutorial/fetch-wrapper (Last accessed: 28.05.2024).

ДОДАТОК А

Лістинг коду застосунку

Код класу *AuctionController.cs*:

```
using AuctionService.Data;
using AuctionService.DTOs;
using AuctionService.Entities;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using MassTransit;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Contracts;
using Microsoft.AspNetCore.Authorization;

namespace AuctionService.Controllers;

[ApiController]
[Route("api/auctions")]
public class AuctionsController : ControllerBase
{
    private readonly AuctionDbContext _context;
    private readonly IMapper _mapper;

    private readonly IPublishEndpoint _publishEndpoint;

    public AuctionsController(AuctionDbContext context, IMapper mapper,
        IPublishEndpoint publishEndpoint)
    {
        _context = context;
        _mapper = mapper;
        _publishEndpoint = publishEndpoint;
    }

    [HttpGet]
    public async Task<ActionResult<List<AuctionDto>>> GetAllAuctions(string date)
    {
        var query = _context.Auctions.OrderBy(x => x.Item.Make).AsQueryable();

        if (!string.IsNullOrEmpty(date))
        {
                                                query       =       query.Where(x       =>
x.UpdatedAt.CompareTo(DateTime.Parse(date).ToUniversalTime()) > 0);
        }

                                                                    return              await
query.ProjectTo<AuctionDto>(_mapper.ConfigurationProvider).ToListAsync();
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<AuctionDto>> GetAuctionById(Guid id)
    {
        var auction = await _context.Auctions
            .Include(x => x.Item)
            .FirstOrDefaultAsync(x => x.Id == id);

        if (auction == null) return NotFound();
```

```
        return _mapper.Map<AuctionDto>(auction);
    }

    [Authorize]
    [HttpPost]
    public async Task<ActionResult<AuctionDto>> CreateAuction(CreateAuctionDto auctionDto)
    {
        var auction = _mapper.Map<Auction>(auctionDto);

        auction.Seller = User.Identity.Name;

        _context.Auctions.Add(auction);

        var newAuction = _mapper.Map<AuctionDto>(auction);

        await _publishEndpoint.Publish(_mapper.Map<AuctionCreated>(newAuction));

        var result = await _context.SaveChangesAsync() > 0;

        if (!result) return BadRequest("Could not save changes to the DB");

        return CreatedAtAction(nameof(GetAuctionById),
            new {auction.Id}, newAuction);
    }

    [Authorize]
    [HttpPut("{id}")]
        public   async   Task<ActionResult>   UpdateAuction(Guid   id,   UpdateAuctionDto
updateAuctionDto)
    {
        var auction = await _context.Auctions.Include(x => x.Item)
            .FirstOrDefaultAsync(x => x.Id == id);

        if (auction == null) return NotFound();

        if(auction.Seller != User.Identity.Name) return Forbid();

        auction.Item.Make = updateAuctionDto.Make ?? auction.Item.Make;
        auction.Item.Model = updateAuctionDto.Model ?? auction.Item.Model;
        auction.Item.Color = updateAuctionDto.Color ?? auction.Item.Color;
        auction.Item.Mileage = updateAuctionDto.Mileage ?? auction.Item.Mileage;
        auction.Item.Year = updateAuctionDto.Year ?? auction.Item.Year;

        await _publishEndpoint.Publish(_mapper.Map<AuctionUpdated>(auction));

        var result = await _context.SaveChangesAsync() > 0;

        if (result) return Ok();

        return BadRequest("Problem saving changes");
    }

    [Authorize]
    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteAuction(Guid id)
    {
        var auction = await _context.Auctions.FindAsync(id);

        if (auction == null) return NotFound();
```

```
        if(auction.Seller != User.Identity.Name) return Forbid();

        _context.Auctions.Remove(auction);

        await _publishEndpoint.Publish<AuctionDeleted>(new { Id = auction.Id.ToString() });

        var result = await _context.SaveChangesAsync() > 0;

        if (!result) return BadRequest("Could not update DB");

        return Ok();
    }

}
```

Код класу Search*Controller.cs*:

```
using Microsoft.AspNetCore.Mvc;
using MongoDB.Entities;

namespace SearchService;

[ApiController]
[Route("api/search")]
public class SearchController : ControllerBase
{
    [HttpGet]
    public async Task<ActionResult<List<Item>>> SearchItems([FromQuery] SearchParams
searchParams)
    {
        var query = DB.PagedSearch<Item, Item>();

        if (!string.IsNullOrEmpty(searchParams.SearchTerm))
        {
            query.Match(Search.Full, searchParams.SearchTerm).SortByTextScore();
        }

        query = searchParams.OrderBy switch
        {
            "make" => query.Sort(x => x.Ascending(a => a.Make))
                .Sort(x => x.Ascending(a => a.Model)),
            "new" => query.Sort(x => x.Descending(a => a.CreatedAt)),
            _ => query.Sort(x => x.Ascending(a => a.AuctionEnd))
        };

        query = searchParams.FilterBy switch
        {
            "finished" => query.Match(x => x.AuctionEnd < DateTime.UtcNow),
            "endingSoon" => query.Match(x => x.AuctionEnd < DateTime.UtcNow.AddHours(6)
                && x.AuctionEnd > DateTime.UtcNow),
            _ => query.Match(x => x.AuctionEnd > DateTime.UtcNow)
        };

        if (!string.IsNullOrEmpty(searchParams.Seller))
        {
            query.Match(x => x.Seller == searchParams.Seller);
        }

        if (!string.IsNullOrEmpty(searchParams.Winner))
```

```
        {
            query.Match(x => x.Winner == searchParams.Winner);
        }

        query.PageNumber(searchParams.PageNumber);
        query.PageSize(searchParams.PageSize);

        var result = await query.ExecuteAsync();

        return Ok(new
        {
            results = result.Results,
            pageCount = result.PageCount,
            totalCount = result.TotalCount
        });
    }
}
```

Код класу *BidsController.cs*:

```
using AutoMapper;
using Contracts;
using MassTransit;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using MongoDB.Entities;

namespace BiddingService;

[ApiController]
[Route("api/[controller]")]
public class BidsController : ControllerBase
{
    private readonly IMapper _mapper;
    private readonly IPublishEndpoint _publishEndpoint;
    private readonly GrpcAuctionClient _grpcClient;

    public BidsController(IMapper mapper, IPublishEndpoint publishEndpoint,
        GrpcAuctionClient grpcClient)
    {
        _mapper = mapper;
        _publishEndpoint = publishEndpoint;
        _grpcClient = grpcClient;
    }

    [Authorize]
    [HttpPost]
    public async Task<ActionResult<BidDto>> PlaceBid(string auctionId, int amount)
    {
        var auction = await DB.Find<Auction>().OneAsync(auctionId);

        if (auction == null)
        {
            auction = _grpcClient.GetAuction(auctionId);

            if (auction == null) return BadRequest("Cannot accept bids on this auction at
this time");
        }

        if (auction.Seller == User.Identity.Name)
```

```
        {
            return BadRequest("You cannot bid on your own auction");
        }

        var bid = new Bid
        {
            Amount = amount,
            AuctionId = auctionId,
            Bidder = User.Identity.Name
        };

        if (auction.AuctionEnd < DateTime.UtcNow)
        {
            bid.BidStatus = BidStatus.Finished;
        }
        else
        {
            var highBid = await DB.Find<Bid>()
                        .Match(a => a.AuctionId == auctionId)
                        .Sort(b => b.Descending(x => x.Amount))
                        .ExecuteFirstAsync();

            if (highBid != null && amount > highBid.Amount || highBid == null)
            {
                bid.BidStatus = amount > auction.ReservePrice
                    ? BidStatus.Accepted
                    : BidStatus.AcceptedBelowReserve;
            }

            if (highBid != null && bid.Amount <= highBid.Amount)
            {
                bid.BidStatus = BidStatus.TooLow;
            }
        }

        await DB.SaveAsync(bid);

        await _publishEndpoint.Publish(_mapper.Map<BidPlaced>(bid));

        return Ok(_mapper.Map<BidDto>(bid));
    }

    [HttpGet("{auctionId}")]
    public async Task<ActionResult<List<BidDto>>> GetBidsForAuction(string auctionId)
    {
        var bids = await DB.Find<Bid>()
            .Match(a => a.AuctionId == auctionId)
            .Sort(b => b.Descending(a => a.BidTime))
            .ExecuteAsync();

        return bids.Select(_mapper.Map<BidDto>).ToList();
    }
}
```

Код класу *Config.cs* мікросервісу Identity:

```
using Duende.IdentityServer.Models;

namespace IdentityService;
```

```
public static class Config
{
    public static IEnumerable<IdentityResource> IdentityResources =>
        new IdentityResource[]
        {
            new IdentityResources.OpenId(),
            new IdentityResources.Profile(),
        };

    public static IEnumerable<ApiScope> ApiScopes =>
        new ApiScope[]
        {
            new ApiScope("auctionApp", "Auction app full access"),
        };

    public static IEnumerable<Client> Clients =>
        new Client[]
        {
            new Client
            {
                ClientId = "postman",
                ClientName = "Postman",
                AllowedScopes = {"openid", "profile", "auctionApp"},
                RedirectUris = {"https://www.getpostman.com/oauth2/callback"},
                ClientSecrets = new[] {new Secret("NotASecret".Sha256())},
                AllowedGrantTypes = {GrantType.ResourceOwnerPassword}
            },
            new Client
            {
                ClientId = "nextApp",
                ClientName = "nextApp",
                ClientSecrets = {new Secret("secret".Sha256())},
                AllowedGrantTypes = GrantTypes.CodeAndClientCredentials,
                RequirePkce = false,
                RedirectUris = {"http://localhost:3000/api/auth/callback/id-server"},
                AllowOfflineAccess = true,
                AllowedScopes = {"openid", "profile", "auctionApp"},
                AccessTokenLifetime = 3600*24*30,
                AlwaysIncludeUserClaimsInIdToken = true
            }
        };
}
```

Код класу *auctionActions.ts*:

```
'use server'

import { Auction, Bid, PagedResult } from "../types";
import { getTokenWorkaround } from "./authActions";
import { fetchWrapper } from "@/lib/fetchWrapper";
import { FieldValues } from "react-hook-form";
import { revalidatePath } from "next/cache";
export async function getData(query: string): Promise<PagedResult<Auction>> {
    return await fetchWrapper.get(`search/${query}`)
}
export async function updateAuctionTest() {
    const data = {
        mileage: Math.floor(Math.random() * 100000) + 1
    }
```

```
    return await fetchWrapper.put('auctions/afbee524-5972-4075-8800-7d1f9d7b0a0c', data);
}

export async function createAuction(data: FieldValues) {
    return await fetchWrapper.post('auctions', data);
}

export async function getDetailedViewData(id: string): Promise<Auction> {
    return await fetchWrapper.get(`auctions/${id}`);
}

export async function updateAuction(data: FieldValues, id: string) {
    const res = await fetchWrapper.put(`auctions/${id}`, data);
    revalidatePath(`/auctions/${id}`);
    return res;
}

export async function deleteAuction(id: string) {
    return await fetchWrapper.del(`auctions/${id}`);
}

export async function getBidsForAuction(id: string): Promise<Bid[]>{
    return await fetchWrapper.get(`bids/${id}`);
}

export async function placeBidForAuction(auctionId: string, amount: number) {
    return await fetchWrapper.post(`bids?auctionId=${auctionId}&amount=${amount}`, {})
}
```

Код класу *authActions.ts*:

```
import { getServerSession } from "next-auth";
import { authOptions } from "../api/auth/[...nextauth]/route";
import { getToken } from "next-auth/jwt";
import {cookies, headers} from 'next/headers';
import { NextApiRequest } from "next";

export async function getSession() {
    return await getServerSession(authOptions);
}

export async function getCurrentUser() {
    try {
        const session = await getSession();

        if (!session) return null;

        return session.user

    } catch (error) {
        return null;
    }
}

export async function getTokenWorkaround() {
    const req = {
        headers: Object.fromEntries(headers() as Headers),
        cookies: Object.fromEntries(
```

```
            cookies()
                .getAll()
                .map(c => [c.name, c.value])
        )
    } as NextApiRequest;

    return await getToken({req});
}
```

Код класу *useAuctionStore.ts*:

```
import { Auction, PagedResult } from "../types"
import { create } from "zustand"

type State = {
    auctions: Auction[]
    totalCount: number
    pageCount: number
}

type Actions = {
    setData: (data: PagedResult<Auction>) => void
    setCurrentPrice: (auctionId: string, amount: number) => void
}

const initialState: State = {
    auctions: [],
    pageCount: 0,
    totalCount: 0
}

export const useAuctionStore = create<State & Actions>((set) => ({
    ...initialState,

    setData: (data: PagedResult<Auction>) => {
        set(() => ({
            auctions: data.results,
            totalCount: data.totalCount,
            pageCount: data.pageCount
        }))
    },

    setCurrentPrice: (auctionId: string, amount: number) => {
        set((state) => ({
            auctions: state.auctions.map((auction) => auction.id === auctionId
                ? {...auction, currentHighBid: amount} : auction)
        }))
    }
}))
```