МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЮСТИМЕНКО ЄВГЕНІЙ АНАТОЛІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

РОЗРОБКА БЕКЕНД-ЧАСТИНИ ДЕСКТОПНОГО ДОДАТКУ ДЛЯ БЛАГОДІЙНИХ АУКЦІОНІВ: АРХІТЕКТУРА, БАЗА ДАНИХ І БЕЗПЕКА

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
О.В Зелінська, доцент кафедри
інформаційних технологій,
к. т. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2025

АНОТАЦІЯ

Юстименко Є.А. Розробка бекенд-частини десктопного додатку для благодійних аукціонів: архітектура, база даних, безпека. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі розглядається процес розробки бекенд-частини десктоп-додатку для проведення благодійних аукціонів. Основну увагу приділено архітектурі програмної системи, моделюванню бази даних та забезпеченню інформаційної безпеки. У ході дослідження проаналізовано особливості предметної області та функціональні вимоги до системи, запропоновано серверну архітектуру. Результатом роботи є прототип надійної та масштабованої серверної частини десктоп-додатку, який може бути використаний як основа для повноцінної системи онлайн-аукціонів із соціальним спрямуванням. За допомогою таких технологій як: C#, WPF, ASP.NET Core, SignalR, MySQL, EntityFramework був розроблений десктоп-додаток, основною метою якого є проведення благодійних аукціонів.

Ключові слова: десктоп-додаток, клієнт-сервер, C#, ASP.NET Core, SignalR, MySQL, аукціони, благодійність.

80 ст. 50 рис., 2 табл., 2 дод., 43 джерел.

ABSTRACT

Yustymenko E.A. Development of the Backend Part of a Desktop Application for Charity Auctions: Architecture, Database, Security. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2025.

The qualification (bachelor's) thesis examines the process of developing the backend part of a desktop application for conducting charitable auctions. The main focus is on the system architecture, database modeling, and ensuring information security. The study analyzes the specifics of the subject area and the functional requirements of the system, and proposes a server architecture. The result of the work is a prototype of a reliable and scalable server-side desktop application, which can serve as the foundation for a full-fledged online auction system with a social focus. Using technologies such as C#, WPF, ASP.NET Core, SignalR, MySQL, and Entity Framework, an application was developed with the primary goal of facilitating charitable auctions.

Keywords: desktop application, client-server, C#, ASP.NET Core, SignalR, MySQL, auctions, charity.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База даних

СУБД – Система управління базами даних

МП – Мова програмування

WPF – Windows Presentation Foundation

TCP – Transfer Control Protocol

IP – Internet Protocol

HTTP- Hypertext Transfer Protocol

HTTPS – HTTP Secure

SSL – Secure Sockets Level

TLS – Transport Level Security

REST API – Representational State Transfer

SOAP – Simple Object Access Protocol

XML – Extensible Markup Language

JSON – JavaScript Object Notation

JWT – JSON Web Tokens

CSRF - Cross-Site Request Forgery

XSS - Cross-Site Scripting

SQL – Structured Query Language

BSON – Binary JSON

ORM – Object Relational Mapping

EF – Entity Framework

ВСТУП

Інформаційні технології все більше і більше наповнюють реальність сьогодення. Основна маса різних процесів спрощується та автоматизується з допомогою інформаційних систем та запрограмованих алгоритмів. Технології нашого часу полегшують наше життя, роблячи його комфортнішим та продуктивнішим. Вони надають доступ до будь-яких новин, контакту з людьми в будь-якій частині світу, швидких обчислень. Збір, аналітика та представлення непобутових обсягів інформації все частіше здійснюється виключно за допомогою комп'ютерних програм, витісняючи участь людини та одночасно мінімізуючи шанс на помилку.

У теперішній час волонтерство, підтримка та допомога один одному, благодійність також перейшли у інформаційний простір. Збори коштів, волонтерські акції, благодійні аукціони організовують та проводять в соціальних мережах. Соціальні мережі є ефективним інструментом для масового залучення учасників у благодійні ініціативи, значно перевершуючи за охопленням традиційні форми комунікації. Створення окремого додатку для благодійності дозволить набрати та збільшувати аудиторію користувачів-благодійників, що значно збільшить активність допомоги.

Предметом досліджень в даній бакалаврській роботі є методологія розробки бекенд-частини десктоп додатку, створення баз даних, серверів для обробки запитів, налаштування безпеки їх взаємодії.

Об'єктом досліджень є десктоп додаток, який надає можливість брати участь в благодійних-онлайн аукціонах та організовувати їх виключно в межах додатку.

Метою даної бакалаврської роботи є систематизація і закріплення знань, щодо розробки додатків, створенні та взаємодії з базою даних, створенні клієнт-серверних систем та реалізації альтернативи теперішнім методам проведення благодійних аукціонів.

В першому розділі роботи завданням було проаналізувати теоретичні аспекти предметної області, покращити розуміння всіх понять та принципів відносно теми бакалаврської роботи, провести дослідження щодо поточних пропозицій, які надають вже існуючи платформи конкуренти, довести актуальність теми роботи.

Завданням другого розділу даної роботи було проведення глибокого аналізу сучасних технологій, що використовуються у сфері розробки програмного забезпечення, з метою обґрунтованого вибору найбільш відповідних інструментів, фреймворків і мов програмування. У процесі відбору враховувалися як загальноприйняті вимоги до сучасного ПЗ — масштабованість, безпека, продуктивність, зручність розгортання, — так і специфічні потреби проєктованого застосунку, включаючи підтримку клієнт-серверної архітектури, інтеграцію з базою даних та можливість реалізації функціоналу в реальному часі.

У третьому розділі роботи детально розглянуто процес фактичної реалізації серверної частини десктоп та моделювання бази даних. Описано архітектуру серверного застосунку, реалізацію основних компонентів, взаємодію між ними та підключення до бази даних. Наведено приклади створення RESTful-ендпоінтів, реалізації механізмів автентифікації та авторизації користувачів, обробки запитів, а також описано логіку обміну повідомленнями в реальному часі за допомогою SignalR. Окрема увага приділена розробці структури бази даних, визначенню таблиць, зв’язків між ними та ініціалізації початкових даних.

Апробація результатів дослідження доповідалась на конференціях(II Міжнародна науково-практична конференція «Прикладні аспекти сучасних міждисциплінарних досліджень» та III Всеукраїнська науково-практична конференція «Комп’ютерні технології обробки даних») у вигляді тез.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБГРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ РОБОТИ

1.1 Загальні відомості та актуальність благодійних аукціонів

Аукціо́н чи авкціо́н – спеціально організований і періодично дієвий ринок продажу товарів, майна з публічного торгу покупцеві, який запропонував найвищу ціну. Може проводитись як у вигляді зібрання продавців і покупців в одному місці, так і в інтернеті, що називається онлайн-аукціоном [1].

Аукціони також не обійшли стороною благодійність та збори різного роду. Благодійні аукціони подібні до звичайних аукціонів, але відрізняються тим, що виручені кошти передаються на благодійні потреби, в основному тим, хто і надав певну річ або послугу в якості лоту аукціону. Їх перевагою перед звичайними зборами є те, що люди, які вирішують допомогти, отримують винагороду – лот аукціону. Це дає змогу певною мірою мотивувати людей до участі в таких акціях підтримки, тим самим збільшити обсяг допомоги та закритих волонтерських потреб.

В Україні благодійні аукціони стали популярними після повномасштабного вторгнення в 2022 році, завдяки волонтерам, які спочатку пропонували можливість отримати певні «винагороди» кожному учаснику їх збору, незалежно від обсягу їх підтримки, а вже потім перейшли до традиційних форматів аукціону – винагороду отримає той, хто залишив найбільшу ставку. Такі аукціони є важливим інструментом підтримки нашої армії, благодійних фондів та організацій, конкретних осіб, які потребують нашої допомоги.

В основному аукціони проводяться в двох форматах:

- Благодійні вечори або заходи(концерти, зустрічі тощо);
- Аукціони в онлайн-форматі.

Зрозуміло, що розвиток та вплив цифрових технологій привели до значної різниці в популярності офлайн та онлайн аукціонів. Теперішні технології значно

розширили можливості та посилили потенціал онлайн-аукціонів, надавши їм більшу важливих кількість переваг.

Розглянемо ключові переваги онлайн-аукціонів, порівнюючи їх з офлайн-заходами:

- Прозорість проведення аукціонів – учасники можуть відслідковувати процес аукціону в реальному часі;
- Доступність – взяти участь у аукціоні можна у будь-який час з будь-якого місці, зручного для учасника;
- Автоматизація процесу – весь процес проведення аукціону автоматизований, відсутня можливість похибки людини або час очікування її відповіді;
- Широке охоплення аудиторії – за рахунок доступності та простоти в участі значно більша кількість людей зможуть взяти участь в благодійному аукціоні;
- Можливості поширення – онлайн аукціон можна легко поширювати в соціальних мережах та на інших платформах, що значно полегшує просування таких аукціонів;

Розглядаючи ключові переваги онлайн-аукціонів також необхідно враховувати їх недоліки. Одним із найбільших та ключових є первинна недовіра до бренду онлайн-платформи проведення таких аукціонів. В порівнянні з офлайн заходами, де кожен учасник знає ініціаторів та організаторів заздалегідь, до початку аукціону, онлайн-аукціони не можуть надати такої можливості новим користувачам. Також учасники онлайн-аукціонів не можуть перевірити лот аукціону, його автентичність, весь перегляд здійснюється за рахунок фотографій лоту або відео-обзору.

В нинішніх умовах життя в нашій країні, враховуючи трансформацію всіх верств суспільства, суспільну свідомість, благодійність, в тому числі благодійні аукціони, набувають особливої актуальності, як інструмент економічної та соціальної підтримки об’єктів суспільства, які потребують цю підтримку.

Актуальність організації та проведення таких аукціонів можна визначити за рахунок декількох чинників.

По-перше, зростаюча та більш свідома громадянська активність та позиція. Наше суспільство все більше і більше усвідомлює для себе важливість всілякої допомоги один-одному або внесків в спільну справу. Благодійні аукціони, як форма участі, дають змогу допомогти та долучитись до спільної справи у простий спосіб, але в той час емоційно насичений спосіб.

По-друге, в умовах складної економічної ситуації в межах країни, саме громадські ініціативи та акції стають основним джерелом підтримки для тих, кому вона справді потрібна. Благодійні онлайн-аукціони дозволяють залучати в підтримку таких ініціатив всі частини населення країни: приватних осіб, бізнеси підприємства, а також меценатів з міжнародної спільноти і збільшити популярність благодійності за рахунок особливої взаємодії – пожертви у вигляді символічного обміну.

По-третє, актуальність аукціонів зростає в умовах інформаційного суспільства. Поширення соціальних мереж, онлайн-платформ та цифрових технологій створює нові можливості для організації, масштабування та популяризації благодійних ініціатив. Завдяки цьому, благодійні аукціони перетворюються на інструмент не лише збору коштів, а й формування нової культури доброчинності, яка базується на відкритості, креативності та персональній відповідальності.

Також необхідно враховувати, що благодійні онлайн-аукціони надають можливість об’єднувати між собою такі сектори як мистецтво, бізнес, публічну діяльність і соціальну відповідальність. Створюваний міжсекторний підхід сприяє налагодженню нових форм співпраці між митцями, громадськими організаціями, підприємствами та волонтерами. У результаті створюється позитивний інформаційний фонд, який стимулює розвиток культури благодійності, особливо серед молоді.

У підсумку можна сказати, що війна в Україні потребує гнучких, ефективних та швидких рішень щодо підтримки. Благодійні аукціони можуть стати прозорим, емоційним, відносно простим та перевіреним рішенням, напрямленим на конкретний результат.

1.2 Платформи-конкуренти, переваги та недоліки

При створені інформаційних систем даного типу необхідно орієнтуватись на поточні пропозиції цієї категорії, так як потрібно створити такий продукт, переваги якого зможуть зацікавити необхідну аудиторію, яка готова створювати благодійні-аукціони для закриття власних потреб, використовуючи платформу, а також аудиторію благодійників, готових допомагати отримуючи певну «матеріальну нагороду» після проведення аукціону.

Українські платформи, які є прямими або непрямими конкурентами, мають значний вплив на нішу ринку таких продуктів.

Розглянемо основні українських платформ-конкурентів:

- Monobank
- Prozorro.Sale
- Reibert. Info
- Благодійні аукціони на базах магазинів

Monobank – український банк, який працює онлайн, без фізичних відділень, виключно через мобільний додаток[2]. Хоч Monobank не спеціалізується на проведені аукціонів будь якої форми, він зміг реалізувати механізм «банки» – онлайн-збору зазначеної суми коштів на визначені потреби. Monobank дозволяє визначити людину, яка пожертвувала найбільшу кількість коштів, або випадкового серед всіх благодійників. «Банки» стали простим але дуже ефективним механізмом, який став основою українського волонтерства. Також функціонал на основі платформи мобільного банку надає широкі можливості зі сторони управління фінансами та переказами, з якими важко конкурувати звичайним платформам.

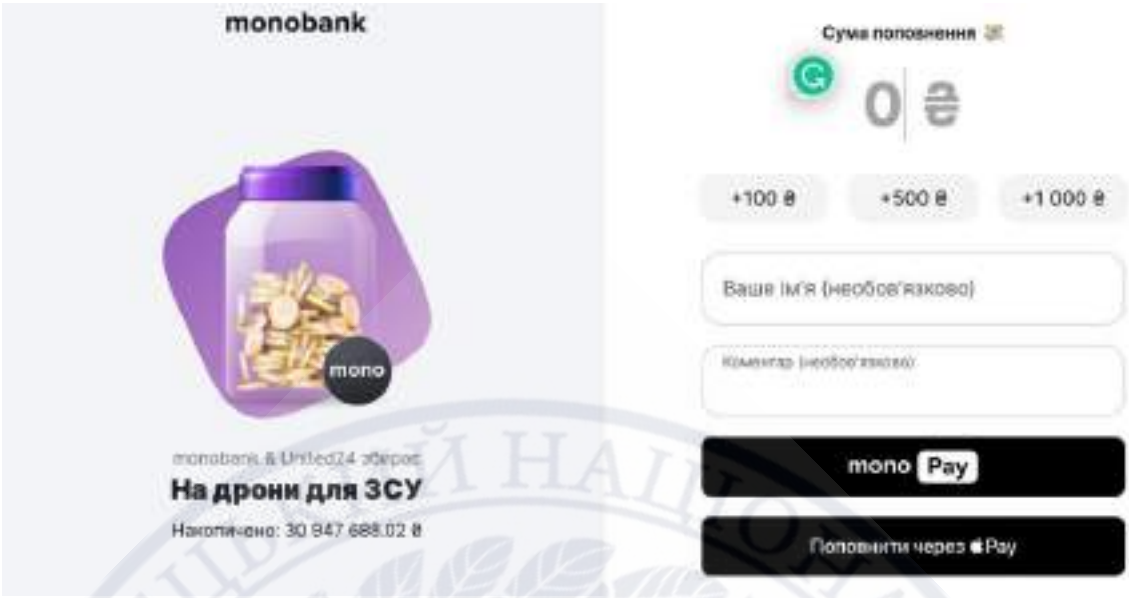

Рисунок 1.1 «Сторінка механізму зборів Monobank»

Prozorro.Sale – українська державна електронна торгова система, яка використовується для продажу державного майна і нерухомості у вигляді аукціонів та оренди подібного майна[3]. Також на платформі реалізований функціонал проведення благодійних аукціонів. Система має надійний та максимально прозорий механізм проведення аукціонів, але в загальному не націлена на благодійність. Платформа надає можливість створення аукціонів як юридичних осіб та організацій, так і для фізичних осіб.

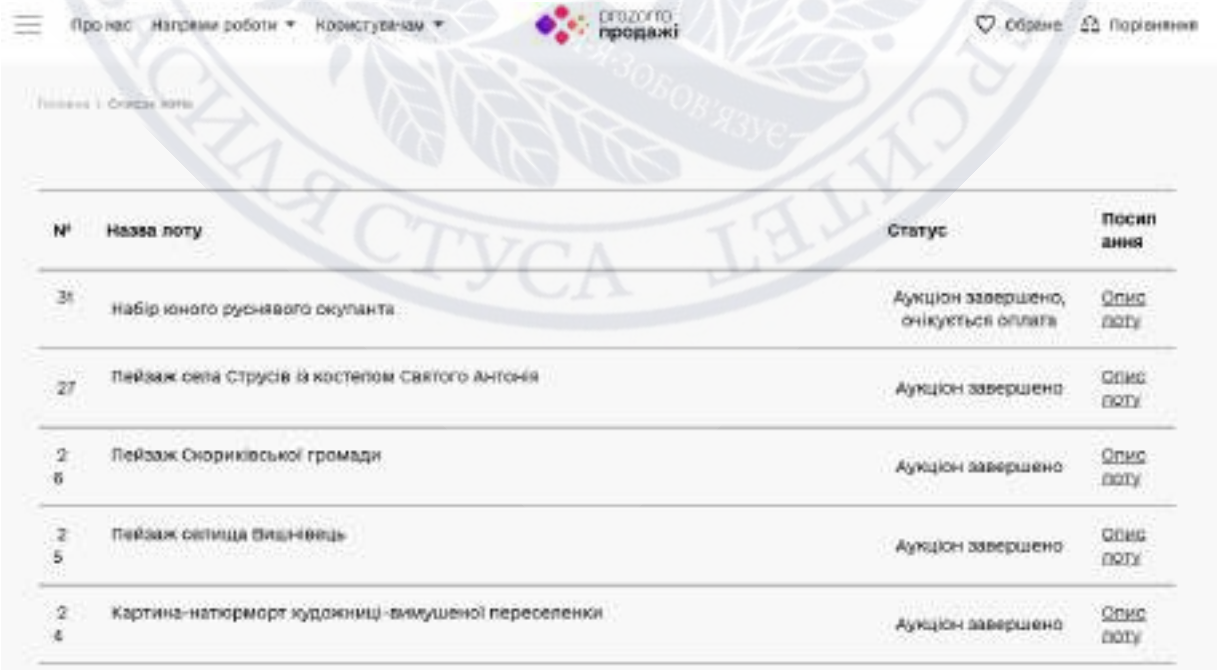

Рисунок 1.2 «Сторінка благодійних аукціонів Prozorro.Sale»

Reibert.Info – це український військово-інформаційний форум та портал, який існує ще з початку 2000-х і є однією з найстаріших платформ в Україні для обговорення військової тематики, зброї, історії бойових дій, техніки, уніформи, а також сучасної війни й волонтерської діяльності[4]. Цей форум має спеціальні розділи для проведення продажів та обмінів між користувачами. В основному вони користуються популярністю серед військових та деяких колекціонерів. Також форум має розділи проведення аукціонів, в тому числі благодійних, які може створити будь-який користувач. Участь в таких аукціонах також необмежена. До переваг форуму можна віднести перевірену та надійну систему, яка виконує всі необхідні функції для проведення таких аукціонів та продажів. Основними недоліками є відсутність власної платіжної системи, що потребує переказу коштів напряму між користувачами. Також форум має обмежену популярність через його специфіку, що значно зменшує кількість учасників аукціонів.

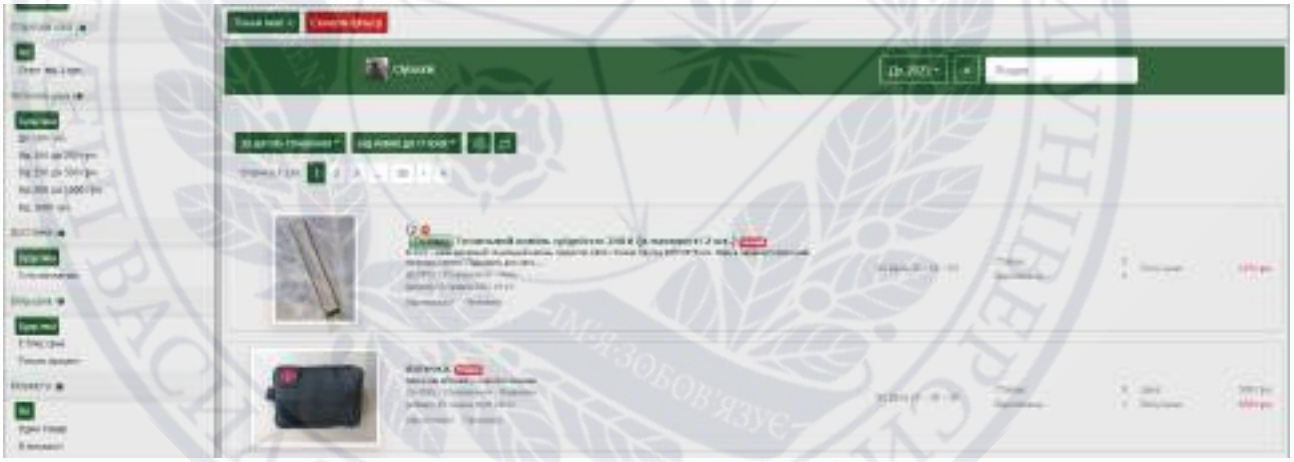

Рисунок 1.3 «Сторінка аукціонів Reibert.Info»

Благодійні аукціони також проводяться локально в межах інтернет-магазинів. Такі платформи не можна віднести до суттєвих конкурентів, так як вони в основному націлені лише на свою клієнтську аудиторію та зазвичай мають специфічні товари, які зацікавлюють менші групи благодійників. Основними прикладами таких платформ є інтернет-магазини картин та інших видів мистецтва, книгарні та інші спеціалізовані магазини.

Отже, основними конкурентами на переваги та недоліки яких варто орієнтуватися під час створення цифрового продукту є Monobank та Prozorro.Sale, так як вони мають найбільшу популярність в сфері благодійних-аукціонів.

Розглядаючи іноземні платформи-конкуренти можна виділити наступні платформи:

- eBay for Charity
- BiddingForGood
- Charity Auctions Today

eBay for Charity – американська благодійна платформа створена на основі всесвітнього сервісу eBay, який спеціалізується на аукціонах[5]. Платформа має зручний, інтуїтивно зрозумілий інтерфейс, перевірену систему та створює прозорі аукціони, з можливістю відстежити, в яку благодійну організацію підуть кошти, проте виключає можливість збирати кошти особам, які не є частиною благодійних організацій. Також недоліком є залежність від системи PayPal, через яку здійснюються транзакції.

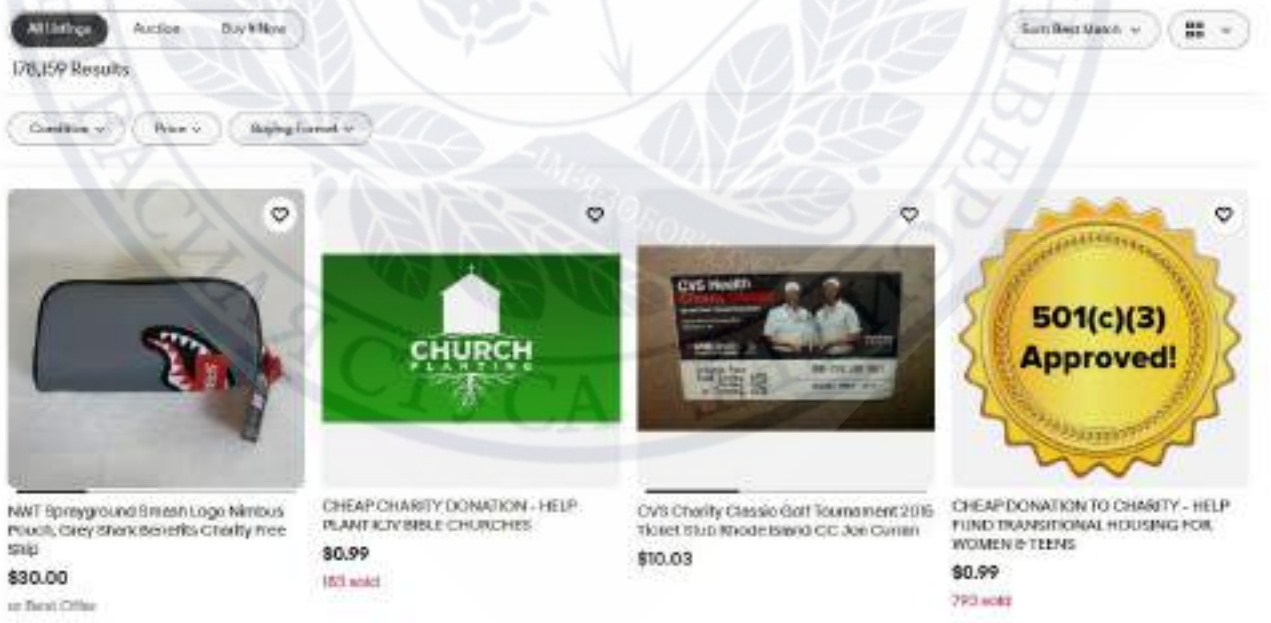

Рисунок 1.4 «Сторінка благодійних аукціонів eBay For Charity»

BiddingForGood – американська благодійна платформа, яка в основному збирає кошти для невеликих організацій та користувачів [6]. До переваг

платформи можна віднести можливість автоматичних ставок до вибраного аукціону, адаптивність до мобільних пристроїв. Недоліками є відсутність можливості участі у аукціонах благодійникам не з США, через обмеження платіжної системи, відносно великі комісії. Платформа вимагає наявність підписки для публікації лотів на аукціон. Також великим недоліком є візуально застарілий інтерфейс, який не приваблює користувачів.

Рисунок 1.5 «Сторінка благодійних аукціонів BiddingForGood»

Charity Auctions Today – американська благодійна платформа, орієнтована виключно на аудиторію межах США [7]. Перевагами цієї платформи є хороша адаптація до мобільних пристроїв, простий в використанні інтерфейс. До недоліків системи можна віднести високу комісію та підтримку від платформи тільки в межах США, що не дає змогу брати участь з інших країн світу. Також платформа вимагає багатокрокової реєстрації навіть для перегляду існуючих аукціонів.

Рисунок 1.6 «Сторінка благодійних аукціонів CharityAuctionToday»

Розглянувши платформи конкуренти, можна сказати, що поточні платформи-конкуренти залишають вільну нішу на ринку, для спеціалізованого десктоп додатку, який зміг би задовільнити всі потреби користувачів, який зможе поєднати в собі зручність використання, прозорість процесів та підтримку благодійних ініціатив.

1.3 Постановка завдання

Основною метою цієї бакалаврської роботи є розробка бекенд-частини десктоп додатку, який слугуватиме платформою проведення благодійних аукціонів. Ця платформа повинна забезпечити простоту у проведенні торгів, можливість просто організувати аукціон, керувати створеними аукціонами. Також необхідно реалізувати захист персональної інформації користувачів, їх банківських даних, організувати надійне збереження всіх даних пов’язаних з роботою десктопного додатку.

Додаток матиме чітко структуровану трьохрівневу архітектуру, яка включатиме клієнтську частину додатку, сервер для обробки запитів від користувачів та реляційну базу даних. Також в додатку необхідно реалізувати логіку взаємодії за принципом розділення відповідальностей. Система передбачає дві ролі користувачів.

- Адміністратор;
- Користувач.

Користувачі мають необмежені можливості щодо участі у аукціонах та можуть самостійно створювати аукціони, додаючи до них лоти. Роль адміністратора передбачає роботу щодо модерації роботи платформи, зокрема тимчасове або перманентне блокування користувачів, контроль над створеними аукціонами, вирішення проблем та питань користувачів. Такий підхід сприяє децентралізованому формуванню контенту, підвищує активність у системі та знижує адміністративне навантаження, зберігаючи при цьому контроль над безпекою й порядком на платформі.

Наступним важливим елементом розробки платформи є проектування реляційної бази даних, яка зберігатиме всю необхідну інформацію про користувачів, адміністраторів, аукціони, ставки, результаті аукціонів, інформацію про проведені платежі. База даних має відповідати всім сучасним вимогам, враховувати принципи нормалізації баз даних, щоб уникати лишнього дублювання інформації, виключити можливість помилок при роботі з базою даних та оптимізувати швидкість проведення запитів до бази даних. Обов'язковою частиною є передбачення можливості резервного копіювання з можливістю швидкого відновлення даних, шифрування конфіденційної та персональної інформації користувачів, включаючи фінансові дані, та історію всіх аукціонів.

Особливу увагу необхідно приділити під час реалізації механізму торгів. Платформа має забезпечувати стабільну обробку всіх ставок у реальному часі, оновлюючи інформацію для всіх користувачів, враховувати час кожної ставки, фіксувати та повідомляти переможця та організатора після торгів. Необхідно реалізувати систему точного таймеру, яка забезпечить точний час аукціону та зважену систему продовження часу аукціону, якщо активність ставок збільшилась в останні хвилини аукціону. Також кожен аукціон має зберігати повну історію ставок, доступну для адміністраторів та звичайних користувачів, що збільшить прозорість проведення аукціонів.

Окремо слід виділити питання безпеки. Оскільки система передбачає обробку персональних та потенційно платіжних даних користувачів, необхідно

реалізувати захищену автентифікацію, механізми валідації введених даних, обмеження доступу до критичних частин системи. Також необхідно розглянути можливість впровадження системи сповіщень про підозрілі дії, як-от багаторазові невдалі спроби входу або незвичну активність.

Результатом виконання завдання має стати повноцінний десктоп додаток, який можна використовувати для проведення реальних благодійних аукціонів. Програмне забезпечення має забезпечувати високу продуктивність, відповідати вимогам безпеки, бути стабільним у роботі та мати можливість подальшого масштабування. Такий додаток сприятиме популяризації онлайн-благодійності, надаючи зручний і зрозумілий інструмент для проведення благодійних аукціонів.

РОЗДІЛ 2

ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ДЕСКТОП ДОДАТКУ БЛАГОДІЙНИХ АУКЦІОНІВ

2.1 Порівняння та вибір мови програмування для розробки десктоп додатків

Теперішній процес розробки програмного забезпечення охоплює велику кількість платформ, мов програмування та технологій, які спрощують роботу з цими мовами. Перед розробкою будь-якої програми, незалежно від її складності, необхідно вибрати мову програмування, на якій буде реалізований алгоритм програми. Розробка десктопного додатку є набагато складнішим процесом, тому потребує більш ретельного підходу до вибору мови програмування. Етап вибору є критично важливим, який визначає не тільки технічні особливості розробки продукту, а і можливості його подальшої підтримки, можливості простого масштабування, визначає його загальну продуктивність. Мова програмування визначає архітектуру додатку, сумісність з різними операційними системами. Також, незалежно від того, що в роботі буде розглянута розробка бекенд-частини, варто забезпечити зручну та ефективну розробку фронтенд-частини.

Розглянемо найпопулярніші мови програмування, які використовуються для створення десктоп додатків:

- C#
- C++
- Python
- Java
- JavaScript
- TypeScript

C# – це об’єктно-орієнтована мова програмування, створена компанією Microsoft спеціально для платформи .NET [7]. Її дизайн орієнтований на

простоту, потужність і зручність використання, що робить її особливо привабливою для розробників. Основною перевагою є тісна інтеграція з Windows, адже саме ця операційна система є головною для корпоративного програмного забезпечення. Завдяки підтримці фреймворків Windows Forms, WPF (Windows Presentation Foundation) та MAUI (для кросплатформенних рішень), розробники отримують гнучке середовище для створення інтерфейсів користувача будь-якої складності.

C# має сильну типізацію, підтримує сучасні концепції програмування, включаючи LINQ, async/await, обробку винятків та делегати. У поєднанні з Visual Studio, однією з найкращих IDE, мова дозволяє ефективно розробляти, тестувати та налагоджувати програми. C# також підтримує нові можливості .NET Core/.NET 5-8, що зробило можливим запуск додатків не лише на Windows, а й на Linux та macOS, перетворивши мову на кросплатформенну. Особливо зручним є створення багатофункціональних інтерфейсів з використанням XAML.

Хоча C# – це мова, яка традиційно орієнтована на корпоративний сектор, вона все більше використовується і в освітніх, комерційних, наукових проектах. Її гнучкість дозволяє створювати як класичні додатки для взаємодії з БД, так і графічні редактори, офісні утиліти чи навіть ігри (використовуючи Unity). Все це робить її універсальним інструментом для розробки програмного забезпечення.

Python – мова програмування високого рівня, яка здобула популярність завдяки простому та читабельному синтаксису[8]. Вона ідеально підходить для швидкого прототипування, автоматизації завдань, обробки даних та наукових розрахунків. Що стосується розробки додатків, Python також має низку інструментів, таких як Tkinter, PyQt, wxPython і Kivy. Вони дозволяють створювати графічний інтерфейс, хоча цей процес вимагає більше зусиль для досягнення рівня нативності та естетики, що надає, наприклад, C#.

Основною перевагою Python є його екосистема, побудована розробниками, які його використовують. Створено дуже велику кількість бібліотек, які дозволяють швидко підключити функціонал до програми: забезпечення доступу до файлової системи, робота з мережевими ресурсами, інтеграція з базами даних

та виконання операцій з графічною інформацією. Але продуктивність Python, як інтерпретованої мови, є відносно нижчою. Для складних та ресурсоємних програм це може стати серйозним обмеженням. Крім того, Python-додатки зазвичай вимагають встановлення додаткових середовищ (наприклад, Python-інтерпретатора або віртуального середовища), що ускладнює поширення програм.

Для створення легких утиліт або внутрішніх інструментів Python може бути чудовим вибором. Але якщо передбачається створення багатофункціонального додатку з вимогами до швидкодії, ресурсоємності та естетичного інтерфейсу – Python не завжди виправдовує очікування. Його краще використовувати у якості допоміжного інструменту або для створення прототипів.

TypeScript – була представлена як надбудова до мови програмування JavaScript, що додає статичну типізацію та об’єктно-орієнтовані можливості[9]. Вона була створена для того, щоб полегшити підтримку великих кодових баз, і набула популярності серед розробників веб-інтерфейсів. TypeScript використовується переважно разом із Electron – платформою, яка дозволяє створювати кросплатформені додатки на базі веб-технологій. Таким чином, TypeScript фактично дозволяє створювати нативні застосунки за допомогою вебінтерфейсів.

Перевага TypeScript у тому, що він уможливлює створення складних інтерфейсів з розділенням компонентів, перевіркою типів на етапі компіляції та зручною інтеграцією з сучасними фреймворками, такими як React або Vue. Розробка відбувається швидко, оскільки більшість інструментів автоматизує багато аспектів – від збірки до оновлень. Але при цьому продуктивність таких програм часто є нижчою, ніж у нативних застосунків на C# або C++.

Попри це, TypeScript стає популярним вибором для інтерфейсно-орієнтованих, мультиплатформенних програм – таких як чат-клієнти, панелі керування, редактори контенту. Але слід враховувати, що розробка на TypeScript

передбачає використання вебтехнологій, що може створювати додаткову залежність від веб-двигунів і збільшувати використання оперативної пам’яті.

JavaScript, як і TypeScript, в основному асоціюється з веб-розробкою, але варто зазначити, що і в контексті десктопного програмування ця мова теж знайшла своє місце завдяки платформі Electron, яка дозволяє створювати кросплатформені застосунки з використанням HTML, CSS та JavaScript[10]. Відомі приклади таких додатків – Visual Studio Code, Slack, Discord. Вони працюють на всіх основних операційних системах і демонструють, що JavaScript може бути ефективним навіть в будь-якому операційному середовищі.

Основною перевагою JavaScript є величезна екосистема та підтримка інструментів, які прискорюють розробку. Наявність npm-пакетів, фреймворків і великої кількості документації дозволяє швидко стартувати і реалізувати широкий спектр функцій. Але, як і у випадку з TypeScript, продуктивність додатків, створених з Electron, зазвичай нижча в порівнянні з нативними програмами. Вони споживають більше ресурсів системи, мають вищий поріг мінімальних системних вимог і повільніший старт.

Для команд, які вже мають досвід у веб-розробці, JavaScript може бути логічним вибором, адже знижує час на навчання новим інструментам. Але з погляду довготривалої підтримки, оптимізації продуктивності та глибокої інтеграції з ОС – JavaScript програє таким мовам, як C# чи C++.

Java є однією з найстаріших і найстабільніших мов програмування, яка досі активно використовується у корпоративному середовищі. Однією з ключових особливостей Java є незалежність від платформи: програми, написані на цій мові, можуть працювати на будь-якій операційній системі, де встановлена Java Virtual Machine (JVM). Для розробки додатків існують фреймворки, як-от JavaFX та Swing. Вони дозволяють створювати графічний інтерфейс, хоча з точки зору дизайну та інтуїтивності Java-програми часто поступаються сучасним рішенням на C#.

Потужність Java полягає в її стабільності, безпеці та зручності підтримки великих проєктів. Розробники мають змогу створювати масштабовані рішення з

чіткою структурою, що особливо важливо в середовищах, де необхідна багатопотокова обробка даних або інтеграція з великою кількістю сервісів. Java має сильну спільноту та багато готових бібліотек, однак недоліком може бути відносно громіздкий синтаксис та більш складне налагодження графічного інтерфейсу в порівнянні з C#.

У контексті розробки застосунків Java залишається стабільним вибором, особливо для програм, які потребують мультиплатформенності. Проте вона рідко є першим вибором для створення сучасних застосунків через менш зручний UX/UI, відносно високі вимоги до пам’яті JVM і повільний час запуску.

C++ – це потужна мова програмування низького рівня, яка надає розробнику прямий контроль над пам’яттю, продуктивністю та системними ресурсами. Завдяки цьому C++ часто використовується для створення високопродуктивних програм, включаючи ігрові рушії, графічні редактори, CAD-системи, а також системне ПЗ. Що стосується десктопних додатків, C++ підтримує бібліотеки для створення інтерфейсів, такі як Qt, wxWidgets або MFC. Вони дозволяють створювати кросплатформенні рішення з привабливим і гнучким GUI.

Основною перевагою C++ є надзвичайно висока продуктивність. Якщо додаток вимагає обробки великої кількості даних, обчислень у реальному часі або тісної інтеграції з обладнанням, C++ буде найкращим вибором. Водночас, ця мова вимагає високого рівня технічних знань: управління пам’яттю вручну, робота з покажчиками та шаблонами часто створює додаткові труднощі для менш досвідчених розробників.

Крім того, вартість розробки на C++ може бути вищою, адже процес створення графічного інтерфейсу менш інтуїтивний. Тому, хоча C++ ідеально підходить для програм, де важлива максимальна ефективність, він не є найзручнішим вибором для стандартних застосунків, які не потребують критичної оптимізації.

Під час порівняння мов програмування також розглянемо та врахуємо результати опитування розробників програмного забезпечення щодо їх вибору

мов програмування для розробки додатків. Популярність мови програмування визначає підтримку від фахівців-девелоперів, які її створили. Також більшість фреймворків та бібліотек, які явно спрощують підтримку та розробку продуктів на певній мові програмування створюються спільнотою, яка її використовує, що є важливим фактором при виборі.

Переглянемо результати опитування за 2024 [22] та 2025 [23], які провела платформа DOU.UA:

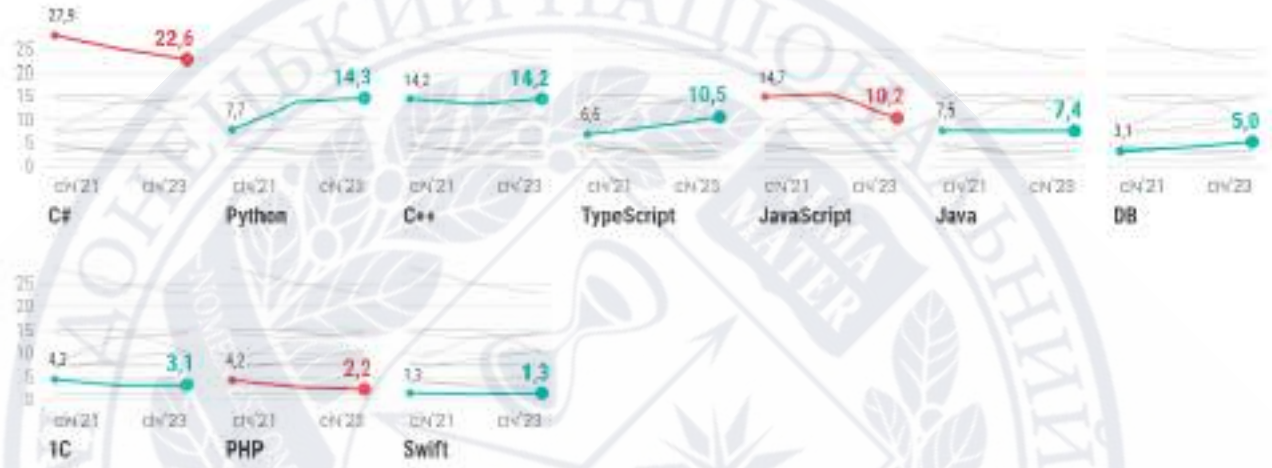

Рисунок 2.1 «Результати опитування популярності мов програмування за 2024»

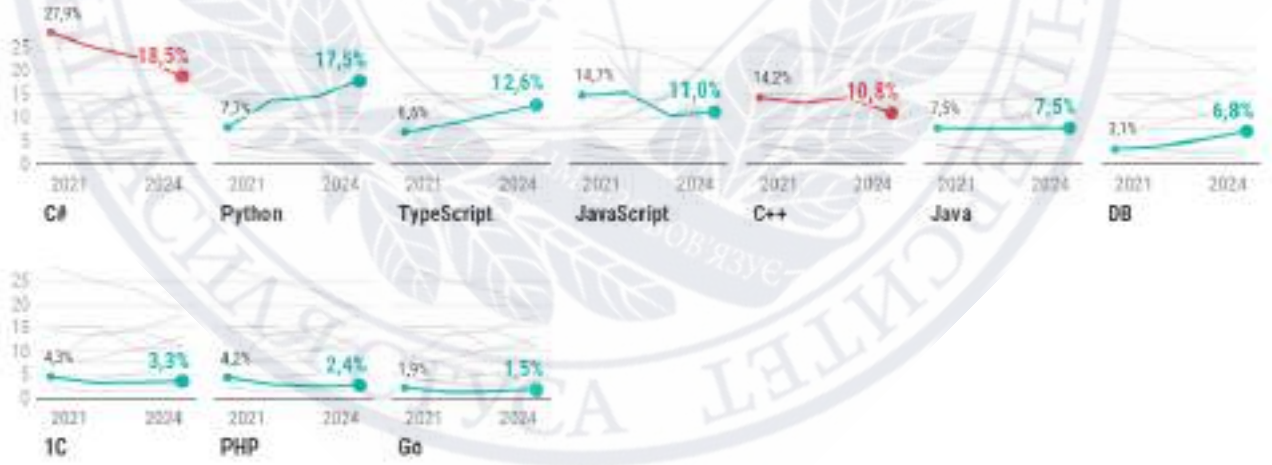

Рисунок 2.2 «Результати опитування популярності мов програмування за 2025»

Отже, ми можемо побачити, що за останні два роки лідируючі позиції займає мова C#. Розглядаючи динаміку зміни популярності необхідно врахувати, що популярність мови C# спадає, на відміну від мов програмування Python, TypeScript, JavaScript. Таким чином, хоча C# і продовжує залишатися однією з

ключових мов для розробки програмного забезпечення, особливо в корпоративному середовищі, загальні тенденції свідчать про зростання інтересу до більш гнучких, легких у вивченні та ширше застосовуваних мов, таких як Python та TypeScript. Водночас C# активно трансформується – завдяки переходу на відкритий код, запуску .NET 6/7/8, підтримці нових парадигм програмування, що дозволяє йому залишатися на гребені хвилі технологічного розвитку.

Представимо результати порівняння мов програмування у вигляді зведеної таблиці (див. табл 2.1):

Табл. 2.1 «Переваги та недоліки мов програмування»

МП	Переваги	Недоліки
C#	Прекрасна інтеграція з Windows; Великий вибір бібліотек для UI (WPF, WinForms); Підтримка .NET; Зручний синтаксис.	Обмежена кросплатформеність для UI (за виключенням.NET MAUI); Вища складність налаштування під Linux чи macOS.
Python	Проста у вивченні; Величезна кількість бібліотек; Гарні можливості для швидкого прототипування.	Низька швидкодія; GUI-фреймворки виглядають застаріло (Tkinter, PyQt);
Java	Кросплатформеність через JVM; Добре масштабування та стабільність; Підтримка Swing/JavaFX.	Високий поріг входу в UI; Застарілий вигляд інтерфейсів; Запуск через JVM знижує швидкодію.
C++	Висока продуктивність і контроль; Добра підтримка GUI через Qt; Підходить для складних застосунків.	Складний синтаксис; Високий ризик помилок через ручне управління пам’яттю; Тривала розробка та дебагінг.
JavaScript	Миттєвий старт для веб-розробників Потужна екосистема Electron дозволяє створювати кросплатформенні додатки.	Важкий і повільний запуск; Високе споживання пам’яті; Низька продуктивність порівняно з нативними мовами.
TypeScript	Краща структура і типізація у порівнянні з JavaScript; Electron дозволяє створювати кросплатформенні додатки.	Високе споживання ресурсів; Менш нативний вигляд та поведінка; Залежність від Node.js та Chromium.

Враховуючи наведені вище аспекти кожної з мов програмування, для виконання бакалаврської роботи виберемо мову програмування C#, так як вона повністю підходить за вимогами до потреб створюваного додатку.

2.2 Технології клієнт-серверних додатків

Паралельно до розвитку інформаційних технологій зростає складність інформаційних систем, що зумовлює збільшення необхідної кількості обчислювальних потужностей. Клієнт-серверні системи є основою сучасних додатків та ефективним засобом розподілення навантажень між ресурсами. Під час створення додатку проведення благодійних аукціонів також реалізуємо трьохрівневу клієнт-серверну систему.

Основними компонентами клієнт-серверної системи є:

- Один або декілька серверів;
- Клієнти;
- Мережа;

Клієнт – програмний компонент, який виконує мінімальні апаратні обчислення і в основному тільки надсилає запити до серверу[24].

Сервер – комп'ютер з програмним забезпеченням, який приймає запити від клієнтів, обробляє їх, виконує більшість апаратних обчислень та взаємодіє з базою даних, або іншим рівнем системи.

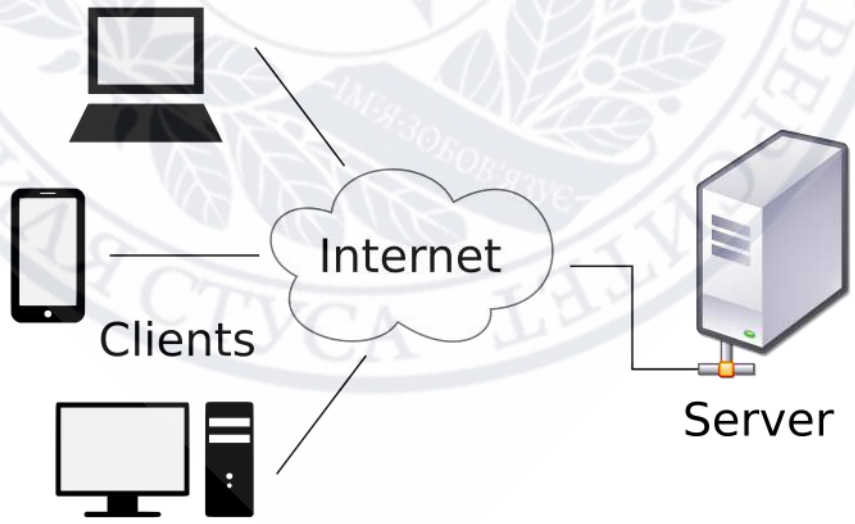

Рисунок 2.3 «Клієнт-серверна взаємодія»

В клієнт-серверній архітектурі виділяють два основних види взаємодії клієнта з сервером:

- Синхронна взаємодія;

– Асинхронна взаємодія;

Використання синхронної взаємодії передбачає, що клієнт буде очікувати відповідь від сервера, після відправлення запиту, перш ніж продовжити виконання подальших дій або запитів. В такому випадку обробка зупиняється на момент на момент виконання запиту, що зумовлює простоту розробки та передбачуваність порядку виконання операцій. Варто зазначити, що синхронна взаємодія може призвести до затримки у роботі додатку, особливо якщо час запиту, обробка даних на сервері займає тривалий проміжок часу, що є частим випадком. Найкращим прикладом є стандартні HTTP-запити, де відповідь обов'язково очікується до завершення обробки запиту.

На відміну від синхронної взаємодії, асинхронна взаємодія надає можливість клієнту відправити запит до серверу і відразу продовжити виконання інших операцій та обчислень без очікування відповіді. Коли сервер відправить відповідь до клієнту, вона буде оброблена окремо за допомогою механізмів, таких як callback-функції або асинхронні об'єкти.

Callback-функція – функція зворотного виклику, яка передається як аргумент до коду та викликається після завершення його виконання [25].

Розглядаючи мову програмування C#, для реалізація асинхронної взаємодії використовуються ключові слова async та await, які дозволяють створювати зручний для читання і підтримки код. Використання асинхронного виду взаємодії значно підвищує продуктивність і швидкість відгуку додатків, особливо в умовах великої кількості запитів або проведення тривалих обчислень та операцій. Варто зазначити, що використання асинхронної взаємодії вимагає більш ретельного проектування процесів додатку.

Клієнт-серверна архітектура базується на основному принципі розділення відповідальностей між двома частинами системи – клієнтською частиною та серверною частиною.

Клієнтська частина займається безпосередньою взаємодією з користувачем: отримує введені дані, ініціює запити до сервера та відображає отримані результати. Серверна частина, своєю чергою, виконує обробку запитів,

реалізує бізнес-логіку застосунку, забезпечує роботу з базою даних і здійснює управління ресурсами. Такий розподіл(див. табл 2.2.) дозволяє кожному компоненту зосередитися на своїй спеціалізованій функції, що спрощує процес розробки, підвищує масштабованість та полегшує підтримку системи у майбутньому.

Табл. 2.2 «Функції компонентів в клієнт-серверній архітектурі»

Компонент	Основні функції
Клієнт	Взаємодія з користувачем; Збір та передача даних серверу; Обробка та відображення результатів; Виклик API та отримання відповідей.
Сервер	Обробка запитів від клієнта; Реалізація логіки системи; Взаємодія з базою даних та іншими компонентами системи; Забезпечення безпеки при взаємодії компонентів системи.

Також необхідно зазначити, що однією з ключових вимог в клієнт-серверній архітектурі є забезпечення незалежності компонентів системи між собою. Клієнтську та серверну частину необхідно визначати як автономні модулі, що взаємодіють між собою тільки через визначені механізми, з використанням протоколів та інтерфейсів. Така незалежність компонентів означає, що будь-які зміни у внутрішній структурі одного з компонентів системи не повинні мати вплив на інший компонент, якщо механізми взаємодії залишаються незмінними. Це надає можливість розробникам паралельно працювати з різними частинами системи, використовувати різні технології та платформи для створення клієнтської та серверної частини, легко оновлювати та масштабувати окремі компоненти системи. Ще одною перевагою такої незалежності є підвищення надійності функціонування системи – у випадку помилок або збоїв в роботі одного з компонентів, інші компоненти можуть продовжувати виконувати свої функції без суттєвих проблем. Зрозуміло, що при наявності лише двох компонентів, клієнта та серверу, при неполадках одного, система працювати не

буде. Таким чином, дотримання принципів розподілу обов’язків та незалежності компонентів є основою побудови ефективних, масштабованих і стійких клієнт-серверних систем.

Розробка додатків з клієнт-серверною архітектурою вимагає реалізацію взаємодії між окремими компонентами системи. Моделі цієї взаємодії відрізняються між собою за кількістю рівнів обробки даних, способом розподілу функціональності між клієнтом, сервером та іншими підсистемами. До найпоширеніших моделей взаємодії в клієнт-серверній архітектурі можна віднести наступні три типи моделей [28]:

- Дворівневі моделі;
- Трирівневі моделі;
- Багаторівневі моделі;

Дворівневі моделі є найпростішою формою реалізації клієнт-серверної архітектури. В цій моделі взаємодії клієнт напряму з’єднується з сервером бази даних або з сервером додатків. Клієнтська частина не тільки реалізує інтерфейс користувача, але й виконує частину обчислювальних операцій та логіку додатку. Серверна частина, в основному, обробляє запити клієнтів, взаємодіє з базою даних, яка зазвичай розгорнута на ньому, і повертає результати клієнту.

Головною перевагою дворівневої моделі архітектури є простота в її реалізації та найменші витрати на розгортання системи. Варто врахувати, що при зростанні кількості клієнтів, що зумовить збільшення навантаження на сервер, буде знижена продуктивність та з’явиться потреба в горизонтальному масштабуванні.

Горизонтальне масштабування передбачає розділення серверу з базою даних на декілька сегментів для розподілення навантаження на них.

Розглянемо схематичну візуалізацію дворівневої архітектури [28].

Рисунок 2.4 «Дворівнева модель клієнт-серверної архітектури»

Трирівнева модель архітектури передбачає реалізації проміжного рівня – сервера додатків, який виступатиме посередником між клієнтською частиною та сервером бази даних. Клієнтська частина буде взаємодіяти виключно з сервером додатків, а сервер додатків здійснюватиме обробку логіки, перевірку даних, обробку запитів та доступ до бази даних.

Реалізації такої моделі забезпечує ряд суттєвих переваг – зменшення навантаження на клієнтську частину додатку, що значно збільшить його продуктивність, відокремлене управління логікою додатку, покращення безпеки за рахунок обмеження прямого доступу до баз даних, кращу можливість для оновлення та масштабування за рахунок розділення компонентів.

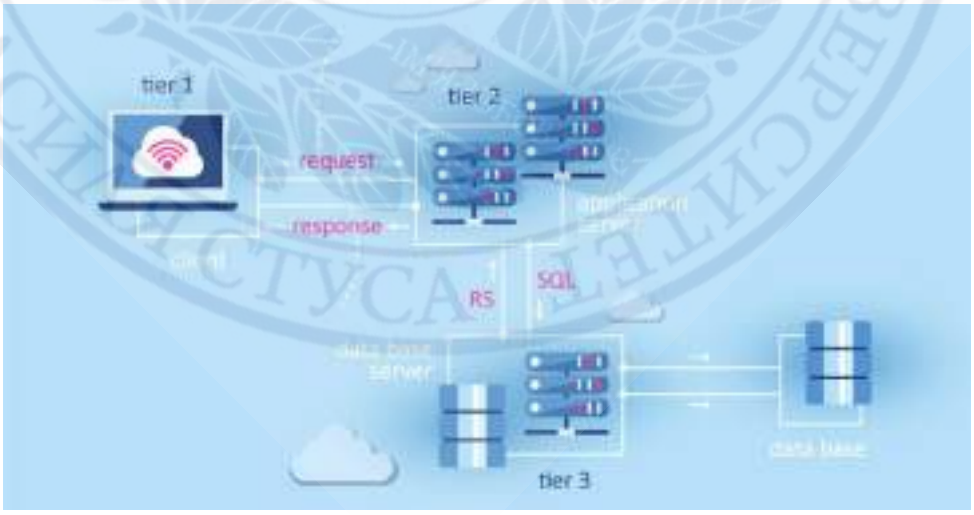

Рисунок 2.5 «Трирівнева модель клієнт-серверної архітектури»

Багаторівнева архітектура є логічним продовженням трирівневої моделі. У цій моделі система розділяється на декілька окремих логічних рівнів, кожен з яких виконує чітко визначені функції. Додатковими рівнями можуть бути сервіси

аутентифікації, модулі кешування, аналітичні сервіси, окремі мікросервіси для специфічних завдань. Кількість додаткових рівнів не є обмеженою та залежить виключно від потреб системи.

Головна ідея багаторівневого підходу – забезпечити максимальну гнучкість, модульність і масштабованість системи. Кожен рівень може масштабуватися окремо, оптимізуватися відповідно до навантаження або вимог безпеки.

Розглянемо схематичну візуалізацію багаторівневої архітектури [28].

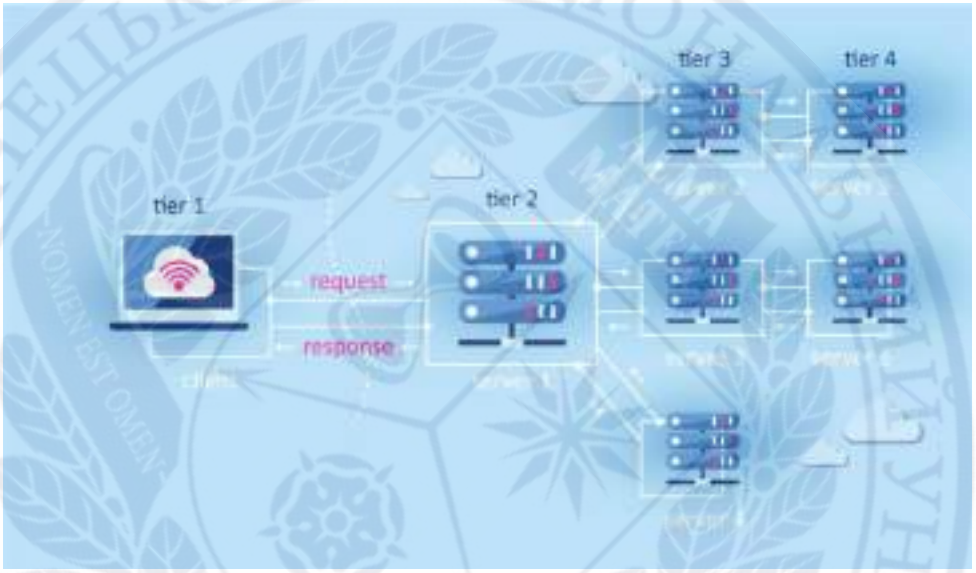

Рисунок 2.6 «Багаторівнева модель клієнт-серверної архітектури»

Клієнт-серверна архітектура вимагає надійного та ефективного обміну даними між клієнтом і сервером. Для цього використовуються різні мережеві протоколи, методи передачі інформації з різними формати обміну даними. Вибір конкретних технологій залежить від вимог до швидкості роботи системи, обсягу даних, які потрібно передавати, необхідного рівня безпеки та особливостей інтеграції з іншими сервісами.

Мережевий протокол – це комплекс установок, завдяки яким визначається і регулюється процес інформаційного обміну між комп’ютерами, підключеними до інтернету[39]. Правильний вибір протоколу напряму впливає на швидкість, безпеку та ефективність роботи застосунку.

Одним із базових стеків протоколів є TCP/IP. Це набір мережевих протоколів, який організовує зв'язок між клієнтами і серверами через Інтернет

або локальні мережі. TCP/IP гарантує надійну доставку даних і підтримує встановлення стабільних з'єднань між учасниками обміну. Протокол TCP забезпечує зв’язок між двома пристроями, передачу інформації та отримання підтвердження про успішність передачі. Протокол IP забезпечує доставку пакетів між двома пристроями.

Протокол HTTP це текстовий протокол, який використовується для передачі запитів і відповідей між клієнтом і сервером, особливо у веб-застосунках. HTTP працює за принципом «запит - відповідь», що найкраще підходить для більшості класичних взаємодій. Щоб забезпечити безпечний обмін даними, використовується HTTPS. Він будується на базі HTTP, але доповнюється протоколом TLS/SSL, який шифрує трафік між клієнтом і сервером. Завдяки цьому унеможливлюється перехоплення або модифікація даних третіми особами.

У випадках, коли необхідна постійна двостороння комунікація в режимі реального часу (наприклад, у чатах, онлайн-іграх та інших real-time системах), застосовується WebSocket. Це протокол, який дозволяє встановити постійне з'єднання між клієнтом і сервером без необхідності постійного повторного відкриття з'єднань, як у випадку з HTTP.

Таким чином, мережеві протоколи відіграють критично важливу роль у побудові клієнт-серверних додатків, визначаючи, як саме буде здійснюватися обмін даними, наскільки швидкою і безпечною буде ця взаємодія.

Передача даних між клієнтом і сервером є основним елементом функціонування будь-якого клієнт-серверного додатку. Серед сучасних технологій найчастіше застосовують три підходи:

- REST API;
- gRPC;
- SOAP.

REST API є найбільш поширеним підходом до побудови веб-сервісів завдяки своїй простоті та масштабованості [29]. Він базується на використанні стандартних HTTP-методів (GET, POST, PUT, DELETE) для взаємодії із ресурсами, які представляються у вигляді унікальних URL-адрес. Клієнти

відправляють запити на сервер, а сервер повертає відповідь, зазвичай у форматі JSON або XML. REST орієнтований на статичність та простоту інтерфейсів, що робить його надзвичайно популярним у веб-розробці.

gRPC – це сучасна технологія, створена компанією Google, яка реалізує ідею віддаленого виклику процедур [31]. Вона використовує HTTP як транспортний рівень і Protocol Buffers як формат серіалізації даних, що забезпечує високу продуктивність, малий розмір повідомлень та можливість організації двостороннього стрімінгу даних між клієнтом і сервером. Завдяки цим перевагам gRPC особливо активно використовується в мікросервісних архітектурах та системах з високими вимогами до швидкості обміну.

SOAP є старішим, але досі актуальним протоколом для обміну структурованими повідомленнями. Він забезпечує строгі стандарти передачі даних та підтримує складні механізми безпеки, що робить його популярним у банківських, фінансових та урядових системах. SOAP-сервіси, на відміну від REST і gRPC, часто передають повідомлення у форматі XML та використовують власний набір правил для побудови запитів і відповідей.

Формат обміну даними між клієнтом і сервером визначає, яким чином буде структуровано та передано інформацію у запитах та відповідях. Найпопулярнішими є формати JSON та XML.

JSON сьогодні є фактично стандартом у більшості веб-додатків. Його основна перевага полягає у легкості синтаксису, зрозумілості для людини та ефективності обробки на боці програми. Завдяки своїй простій структурі у вигляді пар "ключ-значення" JSON дозволяє легко серіалізувати об'єкти та передавати їх через мережу з мінімальними накладними витратами.

XML, хоча і має більший об’єм через надлишкову кількість тегів, забезпечує більшу строгість у структурі даних. XML активно застосовується у системах, де важливо мати жорстке визначення форматів даних і можливість валідації їх через XSD-схеми.

Перейдемо до питання безпеки клієнт-серверної системи. Забезпечення безпеки є одним із ключових завдань при розробці клієнт-серверних додатків.

Ненадійний захист може призвести до втрати даних, витоку конфіденційної інформації або навіть компрометації всієї системи. Основними напрямками безпеки є:

- Аутентифікація та авторизація користувачів;
- Шифрування переданих даних;
- Захист від поширених видів атак.

Суть аутентифікації полягає в перевірці особистості користувача. Вона забезпечує впевненість у тому, що саме той, хто намагається отримати доступ до певного ресурсу, є тим, за кого себе видає. Однією з найпопулярніших технологій для аутентифікації є OAuth 2.0.

OAuth 2.0 - це простий протокол авторизації, заснований на HTTP, що дає можливість застосовувати його практично на будь-якій платформі [32]. Він має хорошу документацію, і більшість великих майданчиків його підтримують. Ця технологія надає можливість користувачам авторизуватися через сторонні сервіси(Google, Facebook) без використання своїх паролів. У процесі авторизації користувачу буде наданий доступ до визначеної кількості ресурсів, відповідно до наданих дозволів.

Іншою важливою технологією, яку варто враховувати та використати є JWT. JSON Web Token (JWT) – це відкритий стандарт для безпечної передачі інформації між сторонами у вигляді об'єкта JSON [33]. Він часто використовується для автентифікації користувачів та безпечної передачі інформації між клієнтом та сервером. Токен містить в собі зашифровану інформацію про користувача та надані йому права доступу. Це надає можливість серверну швидко та без зайвих перевірок ідентифікувати користувача та перевірити, чи має він право виконувати певні операції

Ще одною важливою складовою безпеки клієнт-серверних систем є забезпечення захисту переданих даних за допомогою шифрування. Враховуючи сучасний розвиток мережевих технологій найпопулярнішим інструментом для цього є протокол TLS. TLS протокол забезпечує конфіденційність, цілісність та автентичність даних під час їх передавання між клієнтом і сервером.

Шифрування, створене на основі TLS, є основою протоколу HTTPS, який часто використовують для захисту веб-додатків і API-інтерфейсів. Використання такого протоколу гарантує, що навіть у випадку перехоплення трафіку, доступ до перехопленої інформації отримати неможливо, без відповідного ключа дешифрування.

Використання шифрування є не лише необхідною умовою захисту даних у мережі, а ще важливим елементом довіри користувача до системи, особливо у контексті благодійних аукціонів. Сертифікати безпеки, що підтверджують справжність сервера, видаються довіреними центрами сертифікації, і є важливою частиною механізму аутентифікації. Окрім передавання інформації, шифрування широко застосовують для збереженої інформації – дані у базах даних можуть бути зашифровані, щоб запобігти отримання доступу навіть у разі компрометації фізичного середовища.

Разом із шифруванням особливу увагу приділяють захисту від поширених мережевих атак, які можуть спричинити несанкціоноване виконання дій, компрометацію даних або порушення цілісності системи. Захист від таких атак необхідно проектувати на основі архітектури та коду.

Розглянемо основні з них:

- Cross-Site Request Forgery(CSRF);
- Cross-Site Scripting(XSS);
- SQL Injection.

CSRF – тип атаки, під час якої зловмисник змушує авторизованого користувача виконати небажану дію в системі. Для захисту від атак такого типу необхідно використовувати токени CSRF(токени захисту), які додаються до кожного запиту та перевіряються на сервері [34].

XSS – тип атаки, під час якої зловмисник впроваджує власний шкідливий скрип у веб-сторінку, яку використовують інші користувачі. Захист від атак такого типу полягає в правильному екрануванні введених даних, валідації введення на стороні серверу.

SQL Injection – тип атаки, під час якої зловмисник, використовуючи поля для введення даних, передає шкідливі SQL запити до серверу. Для протидії атакам такого типу необхідно використовувати параметризовані запити та уникати прямої вставки даних з полів введення у SQL запити.

Мова програмування C# з її екосистемою DotNET має широкий спектр бібліотек та технологій, які надають можливості для створення різного програмного забезпечення, зокрема клієнт-серверних додатків. Вибір конкретних технологій та інструмент залежить від вимог до інтерфейсу користувача, рівня інтерактивності, продуктивності, типу взаємодії та платформи, на яку буде націлений додаток. Розглянемо основні технологій та найпоширеніші фреймворки для створенні клієнтської та серверної частин в C#.

Технології клієнтської частини:

– WPF;
– WinForms.

Процес розробки клієнтської частини системи традиційно здійснюється з використанням бібліотек WinForms та WPF.

WinForms є однією з найстаріших технологій для створення графічного інтерфейсу користувача на платформ і [36] .NET. Її основною перевагою є просте використання, що зумовлює низький вхідний поріг. WinForms надає можливості створювати інтерфейс за допомогою вбудованого візуального редактора, з можливістю використання широкого набору стандартних елементів керування – кнопок, полів вводу, списків, меню, діалогів тощо. Архітектура WinForms реалізована на основі принципів обробки подій, що значно спрощує можливість реагування на дії користувача, обробляти логіку клієнта та взаємодіяти з серверною частиною. Враховуючи певні обмеження в дизайні та графічних можливостях, WinForms все ще активно застосовують при реалізації внутрішніх бізнес-додатків, систем автоматизації та інших додатків, де інтерфейс користувача виконує виключно роль панелі керування, та не потребує гарного інтерфейсу.

WPF, в порівнянні з WinForms є більш сучасною платформою для розробки користувацького інтерфейсу, з значно ширшими можливостями [36]. Ця платформа дозволяє створювати складні, масштабовані та естетично привабливі інтерфейси з підтримкою різного роду анімацій та шаблонів, надає можливість прив'язки даних та стилізації. Однією з головних особливостей WPF є використання декларативної мови XAML для опису інтерфейсу, що дозволяє відокремити логіку додатку від презентаційного рівня. Це надає можливості отримати високу гнучкість при проектуванні інтерфейсу та підтримує сучасні шаблони проектування, спрощує масштабування й супровід додатків.

Окрім побудови користувацького інтерфейсу, обидві технології дозволяють інтегруватися з мережею, обробляти запити серверу, працювати з локальними файлами та базами даних, а також реалізувати частину логіки додатку без потреби у використанні додаткового серверного компоненту, що може забезпечити функціональність інтерактивного додатку.

Технології серверної частини:

- ASP.NET Core;
- SignalR.

Для розробки серверної частини у C# найпопулярнішим фреймворком є ASP.NET Core. Це кросплатформенна та високооптимізована веб-платформа з відкритим кодом, призначена для створення веб-додатків, API, мікросервісів та серверної логіки[8]. ASP.NET Core забезпечує основу для побудови REST-API сервісів, що дозволяє клієнтським частинам додатку надсилати запити до сервера й отримувати відповіді у стандартизованому форматі. Основною перевагою ASP.NET Core є гнучкість конфігурації. Платформа надає можливість розробити власний обробник запитів, як послідовність пов'язаних компонентів, що значно спрощує управління аутентифікацією, обробкою помилок тощо. Крім того, в рамках фреймворку ASP.NET Core реалізуються ключові механізми безпеки – аутентифікація, авторизація, підтримка HTTPS та інші.

Також варто враховувати можливість використання SignalR – бібліотеки для ASP.NET Core, яка дозволяє розробити двосторонню комунікацію в режимі

реального часу між клієнтською та серверною частинами. На відміну від звичайних HTTP-запитів, в яких ініціатором виступає клієнтська частина, SignalR надає можливість реалізувати механізм, завдяки якому сервер може надсилати інформацію клієнту без запиту з їхнього боку. Це має особливу актуальність у випадках розробки систем моніторингу, багатокористувацьких додатків та ігор та інших додатків, які потребують оновлення в режимі real-time. SignalR виключає втручання розробка в деталі протоколів та сам автоматично вибирає найкращий спосіб для передавання даних, залежно від можливостей клієнта та сервера.

Завдяки тому, що SignalR є бібліотекою для ASP.NET Core, вони мають чудову інтеграцію між собою та іншими інструментами .Net, що дозволяє створювати продуктивні та безпечні серверні рішення, адаптовані до вимог будь-якого проекту, та матимуть можливості будь-якого масштабування.

І ASP.NET Core, і SignalR чудово інтегруються між собою і з іншими інструментами екосистеми .NET, що дозволяє створювати масштабовані, продуктивні та безпечні серверні рішення, адаптовані до потреб конкретного проекту.

В результаті можна сказати, що поєднання WPF з ASP.NET Core і SignalR на клієнтській та серверній частинах надасть змогу реалізувати повноцінну клієнт-серверну архітектуру, виконуючи вимоги сучасних стандартів безпеки, створюючи можливість масштабування та маючи зручність підтримки та використання.

2.3 Технології і фреймворки для роботи з базами даних

Основою будь-якого сучасного додатку, який пов’язаний з обробкою значних обсягів інформації є база даних. При розробці такого програмного забезпечення варто вибрати відповідні технології та фреймворки для взаємодії з базами даних. Розглянемо найбільш популярні рішення та підходи до роботи з базами даних в контексті розробки бекенд-частини додатку з використанням мови програмування C#.

База даних – це організована сукупність взаємопов’язаних між собою даних, що зберігаються в електронному вигляді та готові до ефективної обробки та взаємодії за допомогою програмного забезпечення – систем управління базами даних. Дані в БД можуть бути структуровані у вигляді таблиць, списків, об’єктів, документів, графів, залежно від моделі даних, яку підтримує СУБД. Основною метою БД є забезпечення надійного зберігання, зручного доступу з можливістю оновлення та багатокористувацькою взаємодією з даними.

Серед усіх моделей зберігання даних самою поширеною є реляційна модель баз даних. Реляційні БД зберігають інформацію у вигляді таблиць, де кожен рядок таблиці відповідає певному запису, а стовпці атрибутам. Усі зв’язки між таблицями реалізуються за допомогою первинних та зовнішніх ключів, що дозволяє уникати дублювання інформації, забезпечити цілісність даних. До найголовніших переваг реляційної моделі зберігання даних можна віднести формальну структуру даних, можливість використання мови SQL для взаємодії з БД, широка підтримка від розробників програмного забезпечення та спільноти.

Також варто брати до уваги об’єктно-орієнтовані бази даних, що стабільно набирають популярність. Об’єктно-орієнтовані бази даних – це БД, в яких дані і їх зв’язки зберігаються у вигляді об’єктів, без стовпців і рядків, що робить їх більш зручними для взаємодії з кодом. Графічно ці бази даних можна зобразити у вигляді дерева [40].

При проектуванні баз даних також необхідно враховувати всі вимоги до них – уникати надлишковості даних, їх дублювання та можливості виникнення аномалій баз даних. Для цього застосовується процес нормалізації баз даних. Нормалізація баз даних – структурне упорядкування таблиць в баз даних відносно форм нормалізації. Більшість сучасних систем вимагають доведення бази даних до третьої нормальної форми.

Розглянемо форми нормалізації баз даних:

- Перша нормальна форма;
- Друга нормальна форма;
- Третя нормальна форма.

Перша нормальна форма вимагає від таблиць баз даних структуру, яка передбачає: визначення основного ключа, виключення можливості дублювання рядків, нероздільність рядків.

Друга нормальна форма передбачає збереження вимог першої нормальної форми та винесення даних, що повторюються в декількох рядках таблиці в іншу таблицю та передачі первинного ключа з нової таблиці.

Третя нормальна форма вимагає збереження вимог другої нормальної форми та вимагає винесення полів, що залежать від первинного ключа та будь-яких інших полів, в іншу таблицю.

Перед початком створення будь-якої бази даних, необхідно визначитись з вибором СУБД, щоб ефективно взаємодіяти з нею. Розглянемо самі популярні СУБД, що активно застосовуються в сучасному програмному забезпеченні:

- MySQL;
- SQLite;
- Microsoft SQL Server;
- MongoDB;

MySQL – одна з найпопулярніших реляційних СУБД з відкритим кодом, що створена корпорацією Oracle. Вона підтримує SQL для створення запитів, забезпечує надійність і високу продуктивність. СУБД MySQL є кросплатформеною, також дуже легко інтегрується з більшістю популярних мов програмування з використанням бібліотек, має можливість масштабування, підтримує реплікацію, має велику документаційну базу. Більшість популярних фреймворків підтримує взаємодію з цією СУБД. Розробники використовують MySQL для створення різних додатків та у веб-розробці, як централізована база даних.

SQLite – вбудована СУБД, що означає можливість розміщення бази даних на одному пристрої разом з додатком, вона зберігається у одному файлі на диску. Завдяки простоті та невеликим розмірам вона є певним стандартом для мобільних додатків, та для невеликих програм без складної логіки та великих обсягів даних для збереження. Враховуючи обмеженість повноцінного

багатокористувацького доступу та обробки складних операцій з базою даних, ця СУБД підходить для особистих проектів, тестування, автономних програм та прототипів.

Microsoft SQL Server – реляційна СУБД розроблена компанією Microsoft, з орієнтацією на корпоративний сегмент. Порівнюючи з конкуруючими системами, вона пропонує ширші можливості аналітики, безпеки, управління транзакціями та інтеграції з іншими продуктами Microsoft. Також одною з переваг є підтримка Transact-SQL, розширення базової SQL з можливістю використання змінних та циклів. Така СУБД підходить для програмного забезпечення з надвеликими обсягами даних, які потребують надійності та безперебійної роботи, можливості масштабування та чіткого контролю доступу.

MongoDB – документно-орієнтована СУБД, яка не використовує мову запитів SQL, натомість пропонуючи власну мову запитів. Система не використовує таблиці для збереження, замість них використовуються документи формату BSON – бінарний формат JSON. Такі властивості дозволяють легко працювати у разі складної або непередбачуваної структури даних. MongoDB має перевагу використання в системах, де структура даних може змінюватись з моменту початкової розробки, наприклад в соціальних мережах. Головною перевагою є можливість автоматичного горизонтального масштабування, яке дозволяє розподілити навантаження на декілька серверів з БД, зберігаючи продуктивність.

Враховуючи всі потреби додатку для проведення благодійних аукціонів, використаємо СУБД MySQL, яка в взаємодії з фреймворком стане зручним інструментом для зберігання та взаємодії з даними, створюючи гарантії безпеки та цілісності даних.

Наступним кроком після вибору СУБД є вибір фреймворку, за допомогою якого і буде проводитися взаємодія застосунку з БД. В сучасному програмному забезпеченні, розробленому на мові C# можна виділити декілька найпопулярніших технологій:

– Entity Framework;

- ADO.NET;
- Dapper.

Entity Framework – найпопулярніший фреймворк для .NET. Він надає можливість взаємодіяти з базою даних, як з об’єктною моделлю, виключаючи потребу написання SQL запитів. Замість цього розробнику використовую об’єкти C#. Entity Framework забезпечує зручну навігацію між об’єктами, зв’язками, підтримує можливість написання запитів за допомогою коду C#, має механізм міграцій для оновлення схеми бази даних.

Також до переваг фреймворку можна віднести найзручнішу можливість використання підходів до реалізації баз даних Code First та DataBase First, в порівнянні з іншими фреймворками C#.

Підхід CodeFirst передбачає створення моделей у коді, після чого за допомогою EF генерацію бази даних на основі цього коду з моделями. Це дозволяє отримати повний контроль над структурою даних та зв’язками між ними та зручно при проектуванні нових систем з нуля.

Підхід Database First передбачає генерацію моделей на основі структури бази даних, яка вже існує. Цей підхід може бути зручнішим для інтеграції в нові системи, які вже мають власні бази даних.

ADO.NET – це набір низькорівневих бібліотек для роботи з базами даних. Технологія передбачає ручне встановлення з’єднання з базою даних, написання SQL запитів, роботу з командними об’єктами. Перевагою фреймворку є отримання повного контролю на обробкою даних, що може стати перевагою в задачах, де необхідно отримати найвищі показники продуктивності. Зрозуміло, що такий підхід потребує кращого розуміння механізмів баз даних.

Dapper – ORM - фреймворк створений компанією Stack Overflow, який поєднує переваги ADO.NET з автоматичним зпівставленням таблиць та об’єктів. Він також дозволяє виконувати SQL запити з мінімальним кодом, отримуючи результати у вигляді C# об’єктів. Фреймворк не містить складних функцій, таких як, наприклад механізм міграцій, тому за рахунок простоти роботи ідеально

підходить для високонавантажених систем і дозволяє отримати найвищу продуктивність.

В результаті порівняння, для розробки додатку використаємо Entity Framework, як основний фреймворк для взаємодії з базою даних, оскільки цей фреймворк поєднує зручність та хорошу інтеграцію з екосистемою DotNET.

2.4 Архітектура додатку благодійних аукціонів

Отже, для реалізації додатку благодійних аукціонів, використаємо трьохрівневу клієнт-серверну архітектуру, що включатиме в себе:

- Клієнтську частину, як презентаційний рівень;
- Сервер, як логічний рівень;
- Базу даних, як рівень зберігання даних.

Клієнтська частина повинна виконувати функцію збору даних, передаючи їх до серверу використовуючи ASP.Net Core. Користувач повинен мати функції реєстрації та авторизації, входу за допомогою OAuth 2.0, використовуючи акаунт Google, переглядати та брати участь в аукціонах, створювати їх, здійснювати операції з власним балансом коштів. Також необхідно реалізувати валідацію даних, для забезпечення їх коректності.

Серверна частина буде створена за допомогою ASP.NET Core, та буде обробляти запити від клієнтської частини, виконуючи логіку додатку. Сервер оброблятиме реєстрації та авторизації користувачів, взаємодію з аукціонами. Для забезпечення взаємодії в реальному часі буде використаний SignalR, так як він надає можливість передавати дані клієнтам без запиту з їх сторони, що необхідно для реалізації аукціону. Також необхідно реалізувати механізми автентифікації, перевірку прав доступу, збереження необхідної інформації до бази даних.

Для збереження даних використовуватиметься реляційна база даних на основі MySQL, взаємодія з якою відбуватиметься через ORM Entity Framework. База даних повинна відповідати третьому нормальному рівню, зберігаючи всі

необхідні дані для коректної роботи системи. Також важливо передбачити механізми резервного копіювання та шифрування інформації.

Рисунок 2.7 «Візуалізація клієнт-серверної архітектури»

РОЗДІЛ 3

ПРОГРАМНА РЕАЛІЗАЦІЯ БЕКЕНД ЧАСТИНИ ДОДАТКУ

3.1 Проектування та розробки бази даних

Правильно спроектована та розроблена база даних є основою стабільного функціонування додатків та надійного зберігання даних. Розподілимо процес проектування та розробки на 3 частини:

- Створення схеми БД для наглядної візуалізації структури та зв’язків між таблицями;
- Розробка бази даних MySQL з використанням утиліти MySQL Workbench;
- Розгортання фреймворку EntityFramework на сервері та налагодження взаємозв’язку з БД з використанням підходу DatabaseFirst.

Також після закінчення розробки бази даних проведемо мінімальне мануальне тестування, для перевірки збереження інформації в БД, задаючи значення напряму через код.

Для створення візуальної схеми бази даних використаємо сервіс DrawSQL, який надає зручні інструменти для таких завдань.

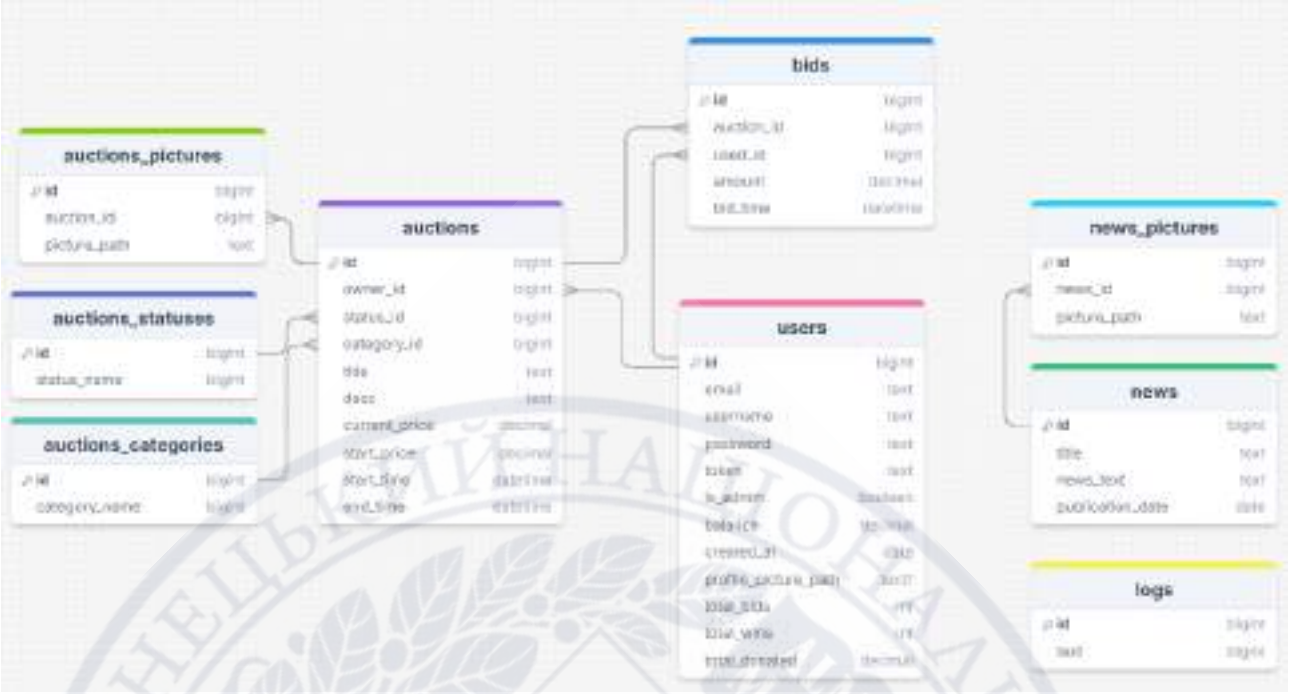

Рисунок 3.1 «Схематична візуалізація бази даних»

Отже, в результаті маємо 9 наступних таблиць, необхідних для правильного функціонування додатку:

- Таблиця користувачів;
- Таблиця аукціонів;
- Таблиця категорій аукціону;
- Таблиця статусів аукціону;
- Таблиця з картинками до аукціонів;
- Таблиця ставок;
- Таблиця новин;
- Таблиця картинок для новин;
- Таблиця логів;

Розглянемо стовпці кожної таблиці, що буде реалізована в базі даних. Таблиця користувачів містить стовпці:

- Id - ідентифікатор користувача, первинний ключ кожного рядку в таблиці;
- Email користувача;
- Username користувача;

- Password – хеш пароля користувача;
- Refresh-token – для продовження авторизованої сесії;
- Is_admin – булева змінна наявності прав адміністратора;
- Balance – баланс користувача;
- Created_at - дата та час створення аккаунту користувача;
- Profile_picture_path – шлях, по якому зберігається зображення аккаунту користувача. Може набувати нульового значення, якщо зображення не встановлено.
- Total_bids – кількість ставок користувача за весь час, для збереження простої статистики.
- Total_wins – кількість виграних аукціонів, також для збереження статистики та відображення в профілі користувача.
- Total_donated – загальна сума донатів користувача, для статистики та відображення.

Таблиця аукціонів має наступну структуру:

- Id – ідентифікатор аукціону, первинний ключ.
- Owner_id – ідентифікатор користувача, який створив аукціон;
- Status_id – ідентифікатор статусу аукціону;
- Category_id – ідентифікатор категорії аукціону;
- Title – заголовок аукціону;
- Desc – опис аукціону;
- Current_price – поточна ціна аукціону.
- Start_price – стартова ставка аукціону;
- Start_time, end_time – час та дата початку і закінчення аукціону.

Таблиця з картинок з аукціонами містить три стовпці: ідентифікатор рядку в таблиці, ідентифікатор аукціону, до якого відноситься ця картинка, шлях до картинки;

Таблиця статусів аукціону має лише два стовпці: ідентифікатор статусу та його текстове значення(назву).

Таблиця категорії аукціонів, по аналогії, також має стовпці: ідентифікатор категорії та її текстове значення(назву).

Таблиця ставок містить наступні стовпці:

- Id – ідентифікатор ставки;
- Auction_Id – ідентифікатор аукціону, на якому було цю зроблено ставку;
- User_Id – ідентифікатор користувача, який зробив ставку на аукціон;
- Amount – значення ставки;
- Bid_time – дата та час ставки.

Таблиця новин має наступну структуру:

- Id – ідентифікатор новини;
- Title – заголовок новини;
- News_text – повний текст новини;
- Publication_date – дата публікації новини.

Таблиця з картинками для новин містить три стовпці: ідентифікатор рядка в таблиці, ідентифікатор новини, до якої відноситься картинка, та шлях до картинки.

Таблиця логів також містить лише стовпці: ідентифікатор рядку(одного запису в логах), та повний текст запису.

Розглянемо зв’язки між таблицями в базі даних. Структура бази даних передбачає лише один тип зв’язку – один до багатьох. Такий зв’язок міститься між полями:

- Ідентифікатор в таблиці користувачів та owner_id в таблиці аукціонів, оскільки один користувач може створити багато аукціонів;
- Ідентифікатор в таблиці користувачів та user_id в таблиці ставок, так як один користувач може зробити багато ставок;
- Ідентифікатор в таблиці аукціонів та auction_id в таблиці ставок, оскільки на один аукціон можна зробити багато ставок;

- Ідентифікатор в таблиці категорій аукціонів, та category_id в таблиці аукціонів, оскільки до одної категорії відноситься багато аукціонів;
- Ідентифікатор в таблиці статусів аукціонів, та status_id в таблиці аукціонів, так як до одного статусу може відноситись багато аукціонів;
- Ідентифікатор в таблиці картинок аукціону та ідентифікатор в таблиці аукціонів, оскільки до одного аукціону може відноситись декілька картинок;
- Ідентифікатор таблиці новин з news_id в таблиці картинок новин, оскільки декілька картинко може відноситись до одної новини.

Перейдемо до створення бази даних використовуючи програмне забезпечення MySQL WorkBench.

Для початку створимо таблицю для логів. Вона не має зв'язків з іншими таблицями, та буде зберігати кожну дію та помилку, під час роботи всієї платформи.

```
1  CREATE TABLE `auction_db`.`logs` (
2    `id_logs` INT NOT NULL AUTO_INCREMENT,
3    `text` VARCHAR(1000) NOT NULL,
4    PRIMARY KEY (`id_logs`));
```

Рисунок 3.2 «Запит створення таблиці логів»

Наступним кроком буде створення таблиці новин, та таблиці картинок новин, ці таблиці мають лише один зв'язко між собою. Додамо значення по замовчуванню для дати публікації, оскільки воно відповідатиме даті запису інформації в таблицю.

```
1  CREATE TABLE `auction_db`.`news` (
2    `id_news` INT NOT NULL AUTO_INCREMENT,
3    `title` VARCHAR(100) NOT NULL,
4    `text` VARCHAR(1000) NOT NULL,
5    `publication_date` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
6    PRIMARY KEY (`id_news`));
```

Рисунок 3.3 «Запит створення таблиці новин»

```
CREATE TABLE `auction_db`.`news_pictures` (
  `id_news_pictures` INT NOT NULL AUTO_INCREMENT,
  `id_news` INT NULL,
  `path` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`id_news_pictures`),
  INDEX `fk_np_to_n_idx` (`id_news` ASC) VISIBLE,
  CONSTRAINT `fk_np_to_n`
    FOREIGN KEY (`id_news`)
    REFERENCES `auction_db`.`news` (`id_news`)
    ON DELETE SET NULL
    ON UPDATE CASCADE);
```

Рисунок 3.4 «Запит створення таблиці зображень новин»

Наступним кроком буде створення таблиці для зберігання інформації про користувачів платформи. Поле дати створення аккаунту користувача також будемо встановлювати поточним часом по замовчуванню.

```
CREATE TABLE `auction_db`.`users` (
  `id_users` INT NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(45) NOT NULL,
  `password` VARCHAR(100) NOT NULL,
  `refresh_token` VARCHAR(100) NOT NULL,
  `is_admin` TINYINT NOT NULL DEFAULT 0,
  `balance` DOUBLE NOT NULL DEFAULT 0.0,
  `created_at` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `profile_picture_path` VARCHAR(100) NULL,
  PRIMARY KEY (`id_users`));

ALTER TABLE `auction_db`.`users`
ADD COLUMN `total_bids` INT NULL DEFAULT 0 AFTER `profile_picture_path`,
ADD COLUMN `total_wins` VARCHAR(45) NULL DEFAULT 0 AFTER `total_bids`;
```

Рисунок 3.5 «Запит створення таблиці користувачів»

Створимо таблиці для статусів та категорій аукціонів, які в подальшому заповнимо вручну, задавши можливі статуси та категорії для аукціонів.

```
CREATE TABLE `auction_db`.`auction_statuses` (
  `id_auction_status` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id_auction_status`));

CREATE TABLE `auction_db`.`auction_categories` (
  `id_auction_category` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id_auction_category`));
```

Рисунок 3.6 «Запит створення таблиць статусів та категорій»

Таблицю для збереження інформації про аукціони створимо після таблиць, від яких вона залежить. Також задамо значення по замовчуванню для часу початку. Час закінчення буде розраховуватись кодом, або трігером в базі даних, додаючи необхідний час від моменту початку аукціону, що відповідатиме логіці додатку. Додамо зв’язки до таблиць з статусами та категоріями, таблиці користувачів.

Рисунок 3.7 «Запит створення таблиці аукціонів»

Таблиця з картинками буде виконувати функції, аналогічні таблиці картинок новин, проте вони будуть розподілені для простішої взаємодії з ними. Додамо зв’язок з таблицею аукціонів, з відношенням багатьох картинок до одного аукціону.

```
CREATE TABLE `auction_db`.`auction_pictures` (
  `id_auction_picture` INT NOT NULL AUTO_INCREMENT,
  `id_auction` INT NOT NULL,
  `path` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`id_auction_picture`),
  INDEX `fk_ap_to_a_idx` (`id_auction` ASC) VISIBLE,
  CONSTRAINT `fk_ap_to_a`
    FOREIGN KEY (`id_auction`)
    REFERENCES `auction_db`.`auctions` (`id_auction`)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

Рисунок 3.8 «Запит створення таблиці аукціонів»

Останньою створеною таблицею є таблиця ставок, яка має зв'язки з таблицями аукціонів та користувачів. Стовпець з часом ставки також встановлюватимемо за замовчуванням, оскільки він відповідатиме часу створення запису.

```
CREATE TABLE `auction_db`.`bids` (
  `id_bid` INT NOT NULL AUTO_INCREMENT,
  `id_auction` INT NOT NULL,
  `id_users` INT NOT NULL,
  `amount` DOUBLE NOT NULL,
  `bid_time` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id_bid`),
  INDEX `fk_b_to_a_idx` (`id_auction` ASC) VISIBLE,
  INDEX `fk_b_to_u_idx` (`id_users` ASC) VISIBLE,
  CONSTRAINT `fk_b_to_a`
    FOREIGN KEY (`id_auction`)
    REFERENCES `auction_db`.`auctions` (`id_auction`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_b_to_u`
    FOREIGN KEY (`id_users`)
    REFERENCES `auction_db`.`users` (`id_users`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE);
```

Рисунок 3.9 «Запит створення таблиці аукціонів»

В результаті виконання запитів буде створена база даних, яка відповідатиме 3-й нормальній формі. На основі створеної бази даних, використовуючи засоби MySQL Workbench, створимо ER-модель(див. рис. 3.10), задля візуалізації та можливості простої перевірки створеної БД.

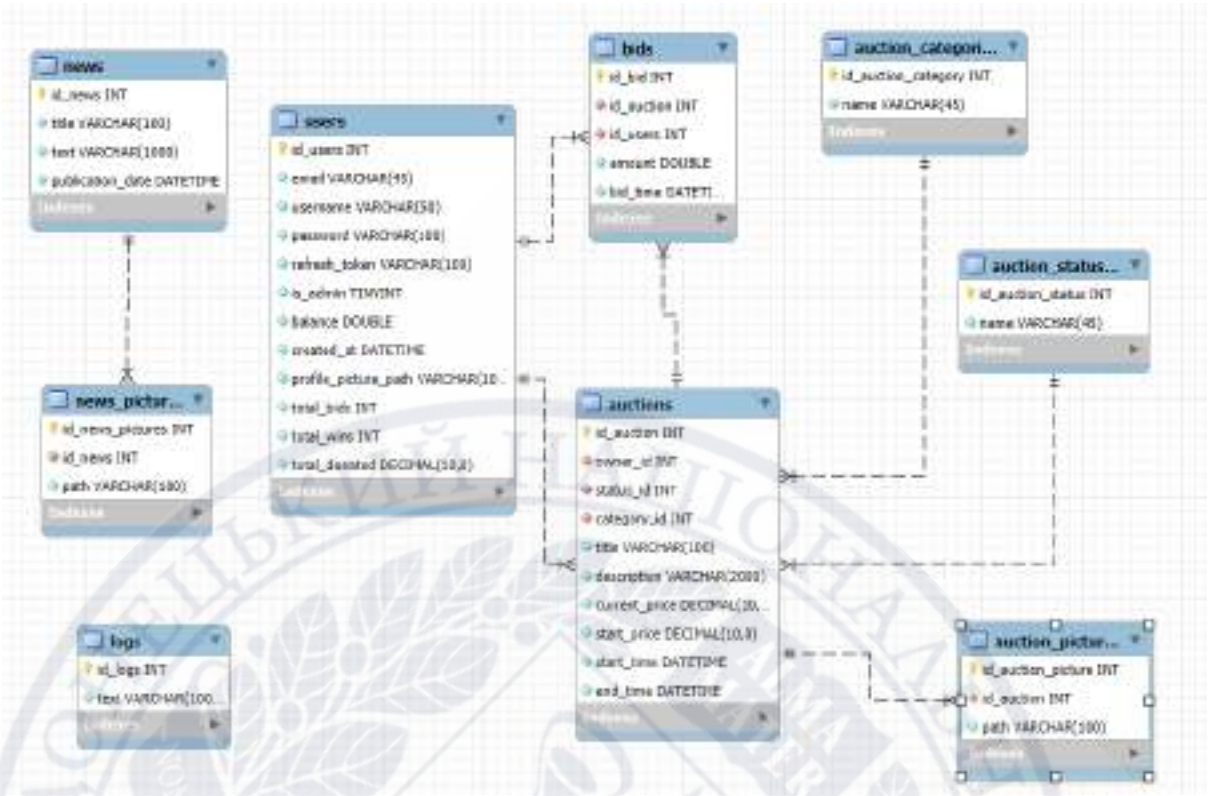

Рисунок 3.10 «Запит створення таблиці аукціонів»

Також необхідно додати дані до таблиць статусів та категорій аукціонів. Це можна здійснити в два способи: створити seed-метод або клас, який за допомогою коду додасть необхідні значення до таблиць при запуску програми, або вручну через SQL запит. Скористаємось варіантом з запитами вручну, задавши необхідні нам категорії аукціонів, згідно логіки додатку:

```
1  INSERT INTO `auction_db`.`auction_categories` (`name`) VALUES ('Сувеніри');
2  INSERT INTO `auction_db`.`auction_categories` (`name`) VALUES ('Монети');
3  INSERT INTO `auction_db`.`auction_categories` (`name`) VALUES ('Мистецтво');
4  INSERT INTO `auction_db`.`auction_categories` (`name`) VALUES ('Одяг/аксесуари');
5  INSERT INTO `auction_db`.`auction_categories` (`name`) VALUES ('Шкільна збруя');
6  INSERT INTO `auction_db`.`auction_categories` (`name`) VALUES ('Предмети харчування');
```

Рисунок 3.11 «Запит додавання даних до таблиці категорій»

Повторимо операцію для таблиці статусів аукціонів, додавши необхідні дані:

```
`auction_db`.`auction_statuses` (`id_auction_status`, `name`) VALUES ('1', 'Активний');
`auction_db`.`auction_statuses` (`id_auction_status`, `name`) VALUES ('2', 'Завершений');
`auction_db`.`auction_statuses` (`id_auction_status`, `name`) VALUES ('3', 'Видалений');
```

Рисунок 3.12 «Запит додавання даних до таблиці статусів»

Підключення бази даних до EntityFramework та налаштування взаємодії з нею буде розглянуто в межах розділу розробки серверної частини, оскільки цей розділ присвячений виключно проектуванню та розробки бази даних.

3.2 Проектування та розробка серверної частини

Серверна частина додатку реалізована з використанням фреймворку ASP.NET Core. Для початку спроектуємо сервер, розподіливши функції.

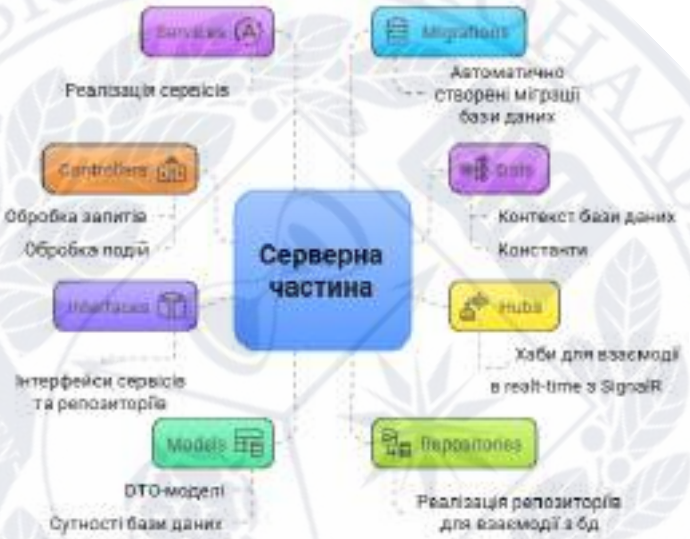

Рисунок 3.13 «Структура серверної частини»

Розглянемо загальну структуру серверної частини, розділеної на компоненти:

- Controllers – містить три основні контроллери, для обробки запитів користувача, внутрішньої обробки аукціонів та обробки взаємодії користувачів з ними, обробки запитів, що містять в собі зображення.
- Interfaces – містить інтерфейси, що визначають структуру та набір властивостей класів, що реалізовані в серверній частині.
- Services – містить класи для виконання допоміжних операцій: відправка email – листів, background-перевірка закінчення аукціонів, мапінг об’єктів, перетворення об’єктів між собою.
- Hubs – містить класи, що реалізують SignalR хаби, для взаємодії клієнтів з сервером в реальному часі.

- Data – містить клас-контекст бази даних, який використовує EntityFramework для визначення сутностей в базі даних, та конфігурації зв’язків між ними. Також каталог містить в собі клас з константами(ключі, строго визначенні заголовки та повідомлення тощо).
- Models – містить сутності бази даних, згенеровані з використанням EntityFramework та DTO – Data Transfer Object, які використовуються для передачі визначеної інформації між клієнтом і сервером.
- Repositories – реалізують шаблон репозиторіїв для взаємодії з базою даних, щоб виключити пряму взаємодію з контроллерів, хабів або сервісів з контекстом бази даних.
- Migrations – містить міграції, які генерує EntityFramework, що дозволяються змінювати структуру бази даних та її моделі, відповідно до реалізації нового функціоналу додатку, не втрачаючи вже існуючі дані.

Оскільки в процесі розробки серверної частини був використаний підхід Database First, що передбачає створення бази даних до написання коду серверної логіки, необхідно на основні бази даних згенерувати моделі сутностей EF та створити початкову міграцію. Це дозволить автоматично створити класи, які відображатимуть структуру таблиць та зв’язки між ними, за допомогою навігаційних елементів. Для генерації моделей використовується команда спеціальна команда Scaffold-DBContext, що виконується в межах каталогу застосунку. Параметрами команди є адреса серверу бази даних, логін та пароль, та назва бази даних.

В результат виконання було згенеровані моделі(див. рис. 3.14) на основі всіх таблиць, що є в базі даних:

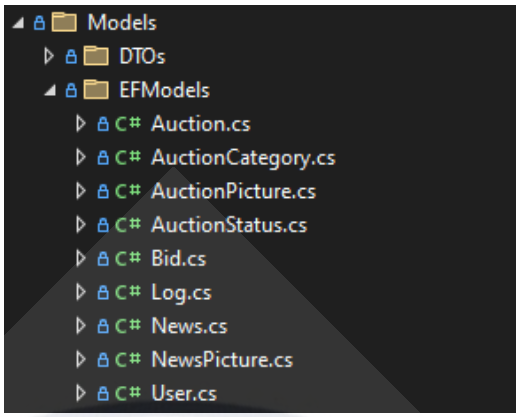

Рисунок 3.14 «Згенеровані класи»

Переглянемо клас аукціону, для прикладу, щоб перевірити правильність генерації на основі таблиці:

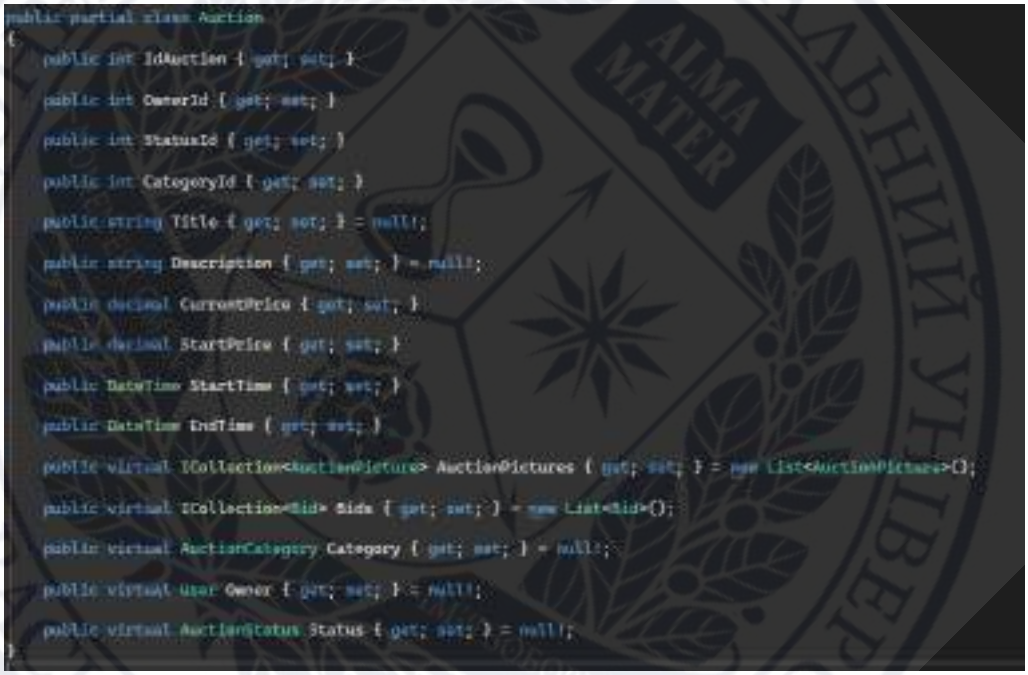

Рисунок 3.15 «Модель Auction»

Клас містить змінні на основі кожного рядку таблиці, та навігаційні елементи з іншими моделями, які мають зв’язки з поточною моделлю.

Після генерації моделей створимо початкову міграцію(див. рис. 3.16), хоч при підході DatabaseFirst це не є обов’язковим, оскільки всі моделі і так згенеровані з бази даних, та мають повну відповідність. Це дозволить в майбутньому вносити зміни до моделей та синхронізувати їх з базою даних, перейшовши до підходу CodeFirst.

Рисунок 3.16 «Результат створення міграції»

Наступним кроком роботи буде створення інтерфейсів репозиторіїв та їх реалізації, для взаємодії з базою даних. Для правильної роботи програми необхідно створити три репозиторії:

– UserRepository – що відповідатиме за взаємодію таблицею користувачів;
– LogRepository – для взаємодії з таблицею логів;
– AuctionRepository – що реалізовуватиме взаємодію з аукціонами, картинками аукціонів та ставками.

AuctionRepository матиме наступну структуру, визначену інтерфейсом(див. рис. 3.17):

– Метод отримання аукціону за Id, та списку всіх аукціонів;
– Методи створення, оновлення та видалення аукціону;
– Метод отримання ставок за аукціоном, їх додавання та видалення;
– Методи створення та отримання картинок до аукціону;

```
public interface IAuctionRepository
{
    Task<Auction?> GetAuctionByIdAsync(int id);
    Task<IEnumerable<Auction?>> GetAllAuctionsAsync();
    Task AddAuctionAsync(Auction auction);
    Task UpdateAuctionAsync(Auction auction);
    Task DeleteAuctionAsync(int id);

    Task<IEnumerable<Bid?>> GetAllBidsByAuctionIdAsync(int auctionId);
    Task AddBidToAuctionAsync(Bid bid);
    Task DeleteBidAsync(int actionId, int bidId);

    Task AddAuctionPicture(AuctionPicture picture);
    Task<IEnumerable<AuctionPicture?>> GetAllPicturesByAuctionIdAsync(int auctionId);

    Task SaveChangesAsync();
}
```

Рисунок 3.17 «Структура репозиторію аукціонів»

UserRepository матиме наступну структуру, визначену інтерфейсом(див. рис. 3.18):

– Методи отримання користувачів;
– Методи створення, оновлення та видалення користувачів;
– Метод оновлення балансу користувача.

```
public interface IUserRepository
{
    Task<User?> GetUserByEmailAsync(string email);
    Task<IEnumerable<User?>> GetAllUsersAsync();
    Task AddUserAsync(User user);
    Task UpdateUserAsync(User user);
    Task DeleteUserAsync(string email);
    Task SaveChangesAsync();
    Task UpdateUserBalanceAsync(int userId, double amount);
}
```

Рисунок 3.18 «Структура репозиторію користувачів»

Репозиторії логів міститиме лише два методи – метод створення нового логу та метод збереження даних. Реалізацію репозиторіїв представимо в додатках(див. додаток А).

Наступним кроком розробки стане реалізація DTO-класів, та сервісу для перетворення DTO в моделі EntityFramework. Для правильної роботи серверу необхідно створити DTO користувача, аукціону та ставки, так як лише вони будуть передаватися до клієнта. DTO матимуть подібну структуру до моделей EntityFramework(див. додаток А). Створимо інтерфейс(див. рис. 3.19) та його реалізацію для сервісу ObjectMapperService. Сервіс виконуватиме перетворення EF моделей аукціону, ставки та користувача в DTO та навпаки.

```
public interface IObjectMapperService
{
    UserDTO getDTOByUser(User user);
    User getUserByDTO(UserDTO userDTO);

    AuctionDTO getDTOByAuction(Auction auction);
    Auction getAuctionByDTO(AuctionDTO auctionDTO);

    BidDTO getDTOByBid(Bid bid);
    Bid getBidByDTO(BidDTO bidDTO);
}
```

Рисунок 3.19 «Структура сервісу перетворення об’єктів»

Реалізуємо сервіс на основі інтерфейсу(див. додаток А). Для прикладу розглянемо метод перетворення моделі аукціону на його DTO:

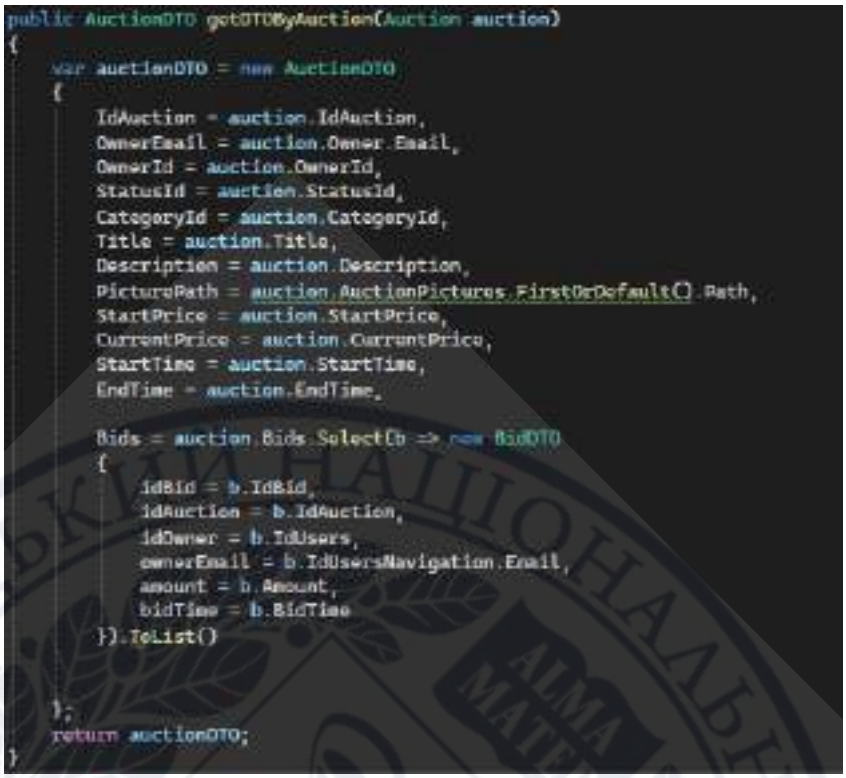

Рисунок 3.20 «Метод перетворення моделі аукціону»

Наступним сервісом для реалізації стане EmailService, задачами якого буде відправка листів користувачам, а саме:

– Код підтвердження email-адреси при реєстрації;
– Повідомлення користувачу про виграш аукціону;
– Повідомлення власнику аукціону про закінчення його аукціону з переможцем або без ставок.

Створимо інтерфейс сервісу(див рис. 3.21) та його реалізацію(див. додаток А):

Рисунок 3.21 «Інтерфейс сервісу відправки email-листів»

Також для збереження констант(таких як ключі доступу до smtp серверу, заголовки повідомлень з назвою відправника та інші) створимо клас Consts, в каталозі Data.

Наступним важливим етапом розробки серверної частини є створення контроллера(див. додаток А), який відповідатиме за реєстрацію та авторизацію

клієнтів, обробляючи HTTP-запити, що надходять від клієнта. Контроллер виконуватиме роль посередника між користувачем та серверною логікою до моменту входу в аккаунт, оскільки після цього від буде під'єднаний до SignalR хабу, і вся взаємодія проходитиме через хаб. Створений клас контроллер успадковується від класу ControllerBase та позначений атрибутом [ApiController], що вказує на те, що він працює в контексті REST API. В контроллері реалізовані endpoint-и, що відповідають за окрему дію:

- Auth/isExist – перевіряє наявність користувача за email в системі, перед реєстрацією або входом;
- Auth/loginByPass – авторизує користувача за паролем;
- Auth/sendCode – відправляє код підтвердження при реєстрації, використовуючи EmailService.
- Auth/registerUser – реєструє користувача за початковими даними;

Контроллер взаємодіє з базою даних за допомогою UserRepository та з EmailService. В результаті логіна клієнта буде підключено до SignalR хабу MainUpdateHub.

MainUpdateHub(див. додаток А) передбачає обробку користувачів, які підключені до серверу та міститиме наступні методи:

- OnConnectedAsync, що виконуватиме передачу даних(повної інформацію про користувача, список активних аукціонів, список історії аукціонів) користувачу, що авторизувався, та додавати його до ConnectionMapperService, щоб мати можливість отримати відношення email – connectionID, при потребі.
- UpdateUserAsync, що виконуватиме оновлення користувача, використовуючи ConnectionMapperService, задля визначення його connectionID.
- CheckUserBalance – метод необхідний для верифікації балансу користувача перед ставкою, задля уникнення несанкціонованих змін на стороні клієнта.

– CreateAuctionAsync – метод необхідний для обробки аукціону, який був створений на стороні клієнта.
– PlaceBidAsync – метод для додавання ставки зі сторони клієнта

Також розглянемо структуру ConnectionMapperService, що використовується для мапінгу користувача та connectionID клієнта хабу:

```
public interface IConnectionMapperService
{
    public void addConnection(string connectionId, string email);
    public void removeConnection(string connectionId);
    public string getEmail(string connectionId);
    public string getConnection(string email);
    public IReadOnlyDictionary<string, string> getAllConnections();
}
```

Рисунок 3.22 «Інтерфейс сервісу мапінгу підключень»

Наступним етапом розробки є створення контроллера для обробки аукціонів(див. додаток А). Цей контроллер виконує наступні функції:

– Завантаження списку активних аукціонів з бази даних, при включенні сервера, перевірка поточної ціни відповідно ставкам, статусу аукціону відносно часу його завершення;
– Отримання списку активних аукціонів для передачі користувачу;
– Додавання нового аукціону до списку активних(після отримання з хабу SignalR).
– Оновлення існуючого аукціону, та ініціалізація передачі нової інформації про аукціон по підключеним клієнтам;
– Додавання нової ставки до аукціону та ініціалізація передачі нової інформації про ставки по підключеним клієнтам;
– Отримання списку історії активності в аукціонах для певного користувача.
– Метод перевірки закінчення аукціону, який викликається з background сервісу, та подальше сповіщення користувачів, пов’язаних з завершенням аукціону, зміну балансу власника.

Розглянемо створення background сервісу, який викликає перевірку статусів аукціонів, що ініціалізує завершення. Варто зазначити, що на клієнті

реалізована перевірка часу закінчення аукціону, що унеможливлює здійснення ставки на закінчений аукціон, навіть якщо здійснити спробу в 10-ти секундний проміжок між оновленнями.

```
public class AuctionCheckerService : BackgroundService
{
    private readonly IAuctionController _auctionController;

    public AuctionCheckerService(IAuctionController auctionController)
    {
        _auctionController = auctionController;
    }

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            await _auctionController.startAuctionsStatusVerify();
            await Task.Delay(TimeSpan.FromSeconds(10), stoppingToken);
        }
    }
}
```

Рисунок 3.23 «Backgroud сервіс завершення аукціонів»

Останнім реалізованим контроллером є ImageController(див. додаток А). Цей контроллер реалізовуватиме endpoint, по якому будуть завантажуватись зображення аукціонів, користувачів, новин. В результаті контроллер повертає шлях до зображення, яке зберігатиметься на сервері. Цей шлях згодом може бути використаний клієнтською частиною для відображення на візуальних елементах. Для забезпечення можливості збереження статичних файлів, та доступу до них, необхідно ввімкнути обробку статичних файлів в налаштуваннях сервері. Це дозволяє сервісу не лише приймати і зберігати файли, а й повертати їх при запиті безпосередньо через URL, що значно спрощує роботу з мультимедійним контентом у межах застосунку. Структура контроллера передбачає можливість сортування файлів по каталогам, та подальшого масштабування.

```
[HttpPost("upload")]
public async Task<IActionResult> UploadPicture(IFormFile file, [FromQuery] string type)
{
    if (file == null || file.Length == 0) return BadRequest("nofile");

    switch (type)
    {
        case "users":
            type = "users";
            break;
        case "auctions":
            type = "auctions";
            break;
        case "news":
            type = "news";
            break;
        default:
            return BadRequest("badtype");
    }

    var imagesPath = Path.Combine(_env.WebRootPath, "images", type);

    if (!Directory.Exists(imagesPath))
        Directory.CreateDirectory(imagesPath);

    var uniqueFileName = Guid.NewGuid().ToString() + Path.GetExtension(file.FileName);
    var filePath = Path.Combine(imagesPath, uniqueFileName);

    using (var stream = new FileStream(filePath, FileMode.Create))
    {
        await file.CopyToAsync(stream);
    }

    var relativePath = $"/images/{type}/{uniqueFileName}";

    Console.WriteLine($"Loaded new image in {type}.");
    return Ok(new { path = relativePath });
}
```

Рисунок 3.24 «Метод завантаження зображень»

Перед початком застосування серверної частини необхідно налаштувати Dependency Injection, що дозволить автоматично створювати та надавати об’єкти там де це потрібно, в зазначеній кількості.

```
12      builder.Services.AddControllers();
13
14      builder.Services.AddHostedService<AuctionCheckerService>();
15
16      builder.Services.AddScoped<IUserRepository, UserRepository>();
17      builder.Services.AddScoped<IAuctionRepository, AuctionRepository>();
18
19      builder.Services.AddSingleton<IEmailService, EmailService>();
20      builder.Services.AddSingleton<IConnectionMapperService, ConnectionMapperService>();
21      builder.Services.AddSingleton<IAuctionController, AuctionController>();
22      builder.Services.AddSingleton<IObjectMapperService, ObjectMapperService>();
23
24      builder.Services.AddSignalR();
25
26      builder.Services.AddDbContext<AuctionDbContext>(options =>
27          options.UseMySql("server=localhost;user=root;password=;database=auctions_db",
28              new MySqlServerVersion(new Version(8, 0, 42))));
```

Рисунок 3.25 «Налаштування залежностей до серверу»

3.3 Реалізація взаємодії серверної та клієнтської сторони десктоп-додатку

В межах розділу була налагоджена ефективна взаємодія між серверною та клієнтською частинами додатку. Основний обмін даними був реалізований за допомогою SignalR, що забезпечує можливість realtime оновлення. Це дозволило досягти інтерактивності застосунку – користувачі в реальному часі отримують нові аукціони, ставки на цих аукціонів, завершення аукціонів, тощо. HTTP запити були використані лише для базових операцій, таких як реєстрація та авторизація в додатку та завантаження зображень на сервер, оскільки ці операції на потребують постійного з’єднання.

В попередньому розділі вже розглядалось створення Data Transfer Objects – необхідних для передачі інформації між клієнтом та сервером. DTO на клієнті відповідатимуть серверним моделям, але матимуть додаткові змінні для відображення на клієнті. Розглянемо їх для кожного DTO.

AuctionDTO містить наступні додаткові змінні:

- FullPicturePath, що містить адресу серверу та посилання на картинку, обраховується автоматично при створенні;
- TimeLeft, що містить час до закінчення аукціону в HH:MM:SS, для відображення таймеру на візуальній частині.

BidDTO містить лише одну додаткову змінну - displayString, яка обраховується на основі дня часу ставки, адреси користувача, який її зробив, та суми ставки. Ця змінна необхідна для візуалізації.

Для початку необхідно реалізувати AuthController(див. додаток Б) - контроллер для авторизації. Він містить в собі наступні методи:

- Окремі методи GetAsync та PostAsync для здійснення GET та POST запитів;
- Метод валідації email перед відправленням запиту;
- Метод відправлення запиту для авторизації користувача;
- Метод відправлення запиту для реєстрації користувача;

- Метод для перевірки наявності email в списку зареєстрованих;
- Метод для перевірки блокування аккаунта користувача при авторизації;

Наступним етапом реалізації буде створення контроллеру UserSessionController(див. додаток Б). Його функціонал полягає в збереженні інформації про авторизованого користувача, для подальшої візуалізації в додатку. Також контроллер містить метод для первинного підключення до SignalR хабу, яке буде ініціалізовано після авторизації.

Після реалізації контроллерів для авторизації та зберігання інформації реалізуємо SignalRService(див. додаток Б), що буде здійснювати обмін інформацією в реальному часі з сервером. В класі необхідно реалізувати методи, що викликатимуться при відправці інформації з сервера, а саме:

- RecieveUserDTO, для отримання інформації про користувача при авторизації, та подальшого оновлення(наприклад при редактуванні профіля);
- RecieveAuctions, для отримання списку активних аукціонів після авторизації;
- RecieveHistoryAuctions, для отримання списку історії аукціонів, в яких користувач брав участь, або створював;
- AuctionAdded, для отримання активного аукціону, який був створений під час сеансу користувача;
- AuctionUpdated, для оновлення активного аукціону;
- UpdateBalance, для оновлення балансу користувача(поповнення балансу, повернення суми ставки, якщо вона була перебита).

Також необхідно реалізувати методи, що викликатимуться зі сторони клієнта, та передаватимуть інформацію на сервер:

- CreateAuction, для створення нового аукціона;

– PlaceBid, для створення ставки на активний аукціон, з попередньою перевіркою, чи не є аукціон створеним користувачем, який хоче зробити ставку, та чи не є найвища ставка, поставлена користувачем;
– CheckBalance, для перевірки балансу користувача перед ставкою, щоб уникнути несанкціонованих змін на стороні клієнта;
– Disconect, для закриття підключення з сервером, при закритті десктоп-додатку.

Після реалізації контроллерів, що підтримують зв'язок з серверном, оновлюють інформацію про користувача та аукціони, необхідно створити контроллер аукціонів на клієнті AuctionStorageController(див. додаток Б), що після отримання списку активних аукціонів, надаватиме можливості їх фільтрації, буде перевіряти інформацію при спробі зробити ставку чи створити аукціон. Також аукціон реалізує таймер, що оновлюватиме значення часу до кінця аукціону, яке прив'язане до візуалізації.

3.4 Тестування додатку

Після закінчення розробки бази даних, серверної частини, клієнтської частини та взаємодії всіх частин необхідно протестувати роботу системи. Оскільки серверна частина не має візуалізації всіх процесів, лише вивід повідомлень в консоль, проведемо мануальне тестування зі сторони клієнта, та результати записів у базі даних. Для початку запустимо сервер:

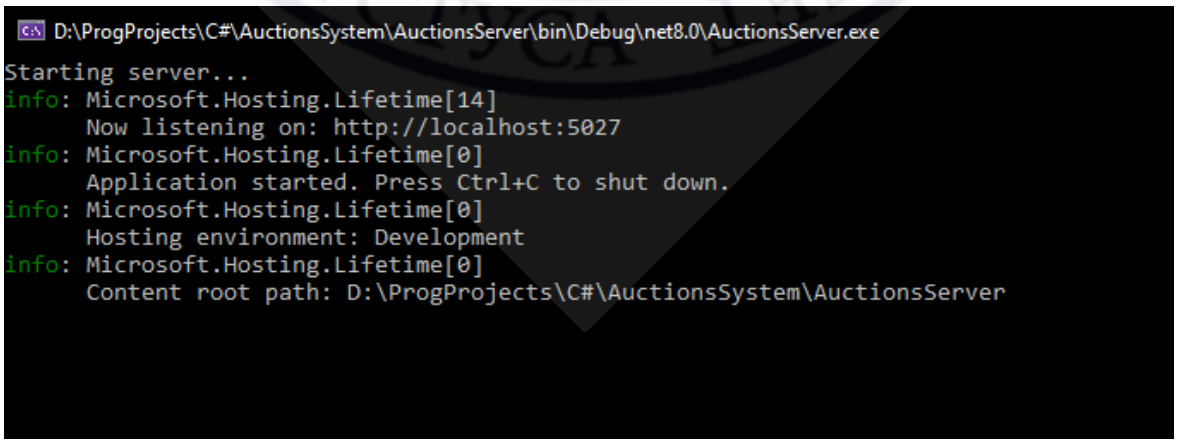

Рисунок 3.26 «Запущене вікно серверу»

Після чого запустимо сервер, та перейдемо до вікна реєстрації:

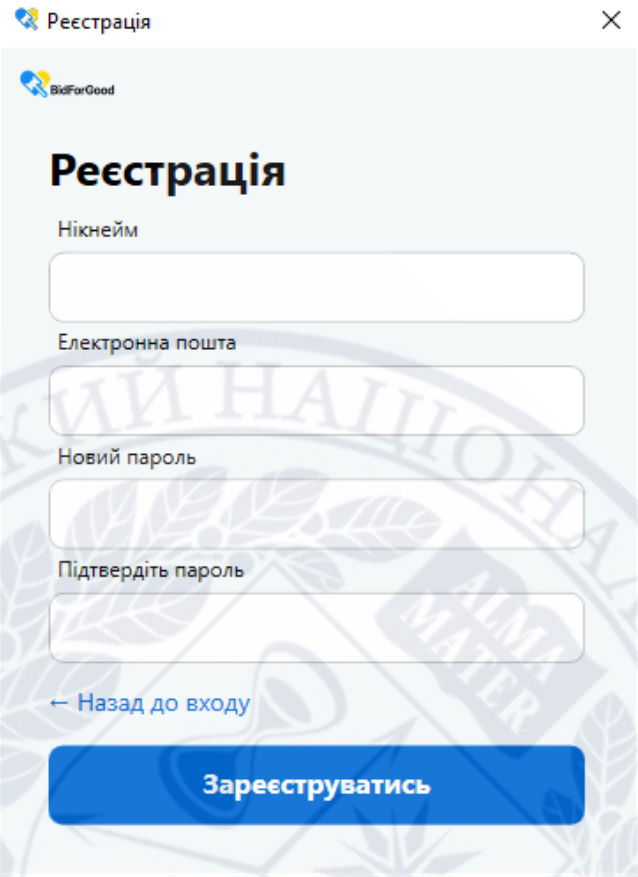

Рисунок 3.27 «Вигляд вікна реєстрації»

Перед початком реєстрації спробуємо ввести вже зареєстровану адресу, яку додамо в базу даних вручну.

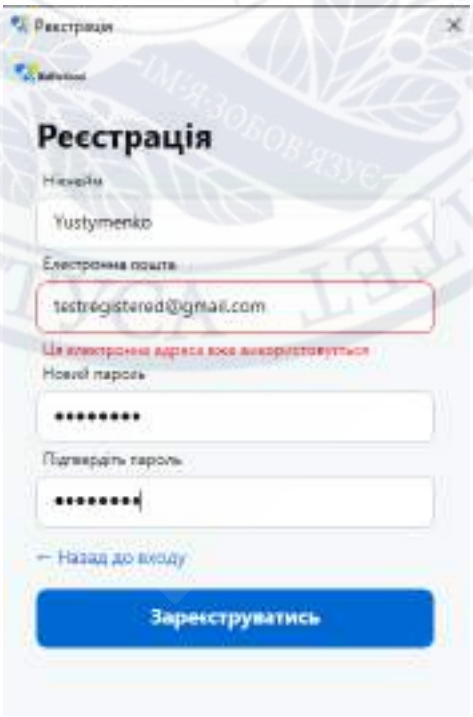

Рисунок 3.28 «Вигляд вікна реєстрації»

В результаті побачимо помилку, що ця адреса вже зареєстрована. Спробуємо зареєструватись з новою адресою:

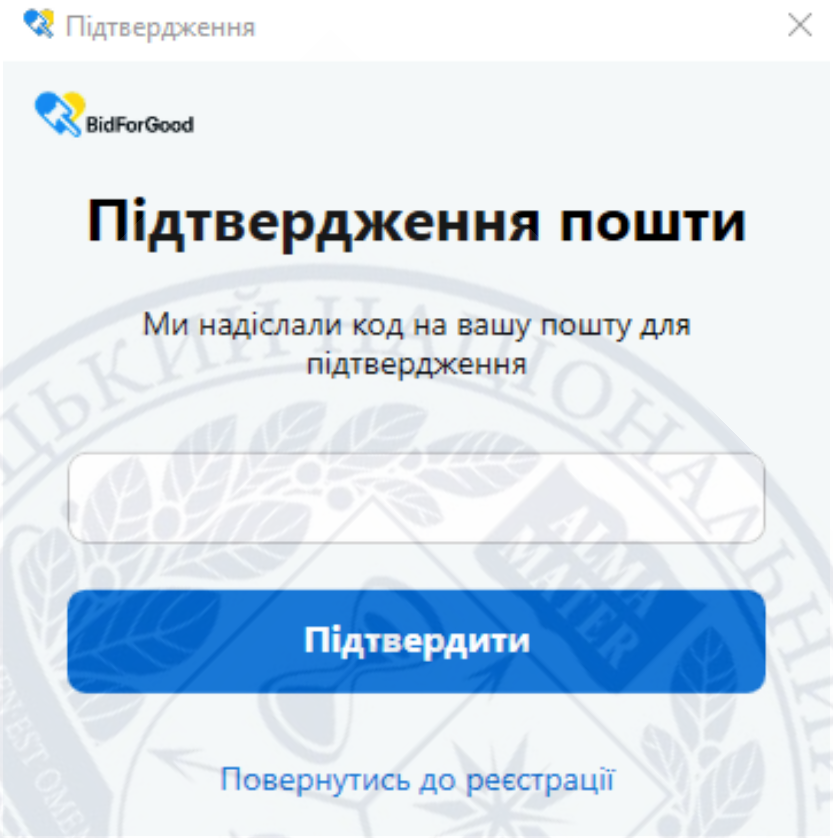

Рисунок 3.29 «Вигляд вікна підтвердження ел. адреси»

В результаті відкриється вікно підтвердження ел. адреси. Перевіримо вказану пошту, на наявність листа підтвердження:

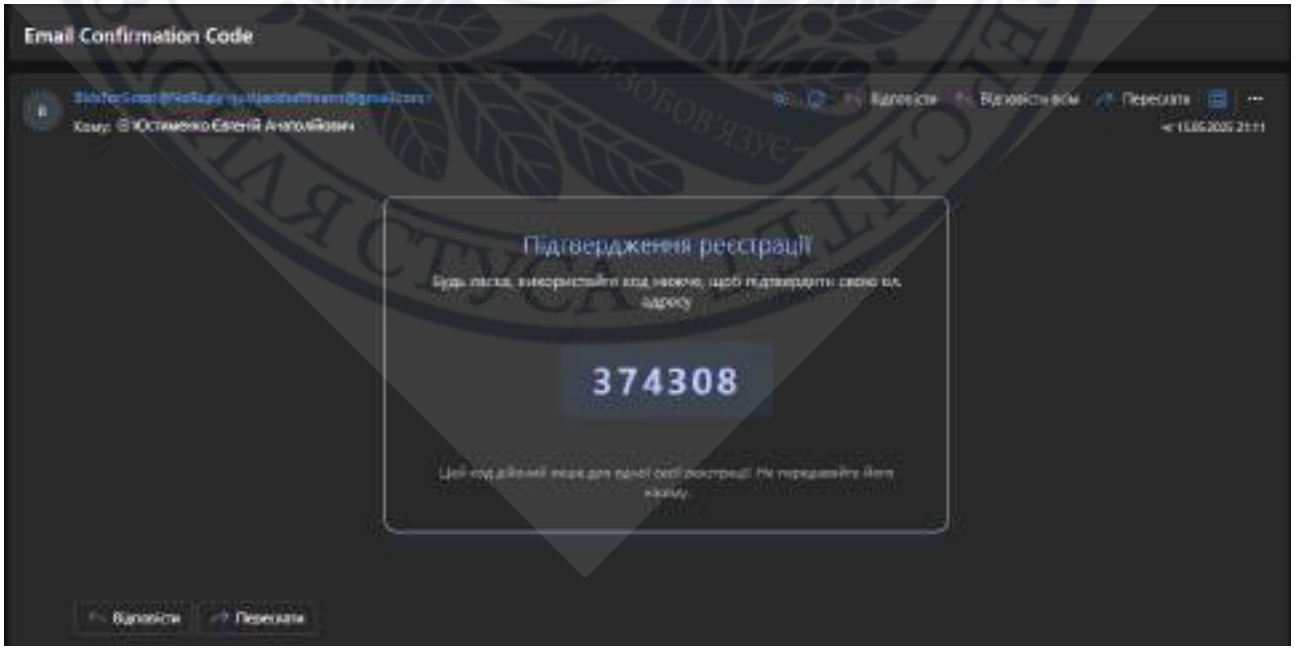

Рисунок 3.30 «Вигляд листа підтвердження ел. адреси»

Введемо код з пошти, та побачимо підтвердження реєстрації:

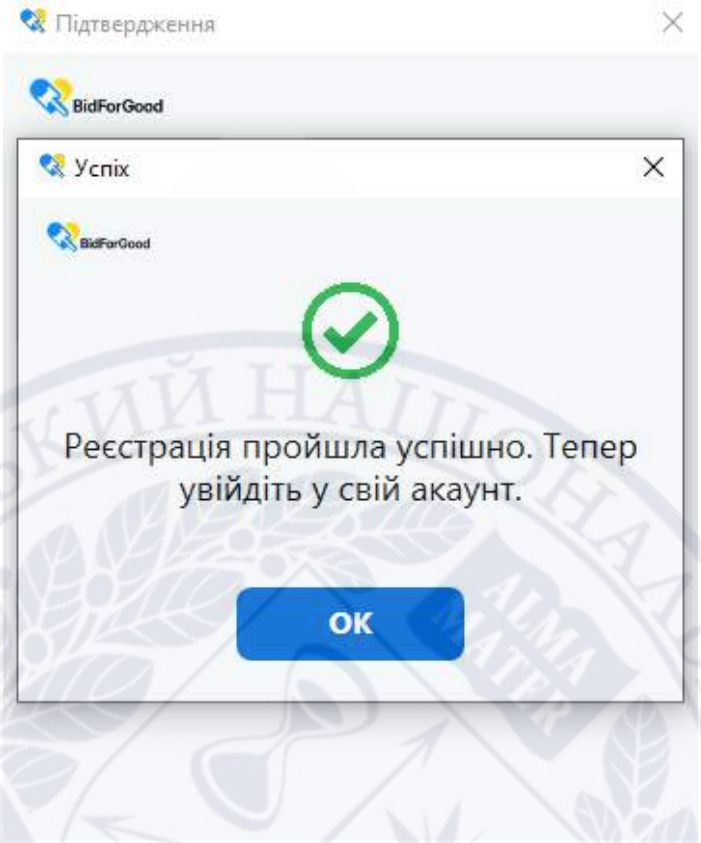

Рисунок 3.30 «Вигляд листа підтвердження реєстрації»

Авторизуємось, використавши вказану адресу та пароль, після чого перевіримо наявність запису в базі даних:

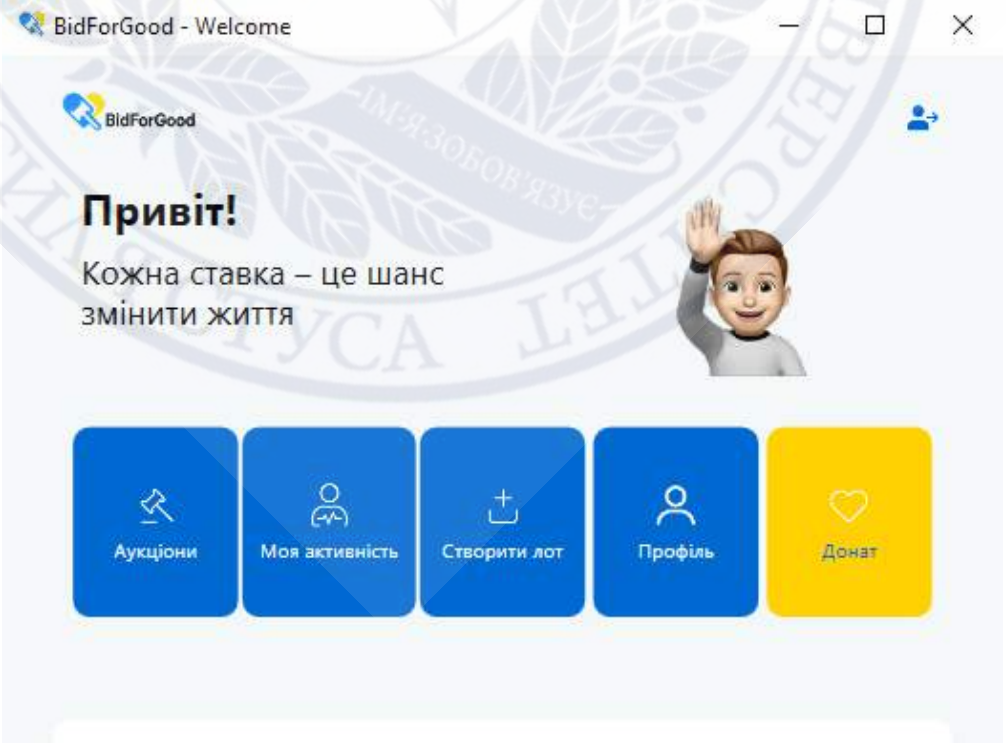

Рисунок 3.31 «Вигляд головного вікна додатку»

Також можна побачити результат запису в базі даних, що показує успішність передачі даних на сервер та до бази даних.

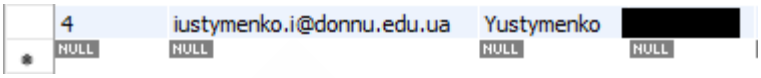

Рисунок 3.32 «Запис в таблиці користувачів»

Оскільки декілька аукціонів уже були створені в процесі розробки, перевіримо їх відображення. Для прикладу використаємо категорію «Предмети харчування» вибравши її в стрічці фільтрів:

Рисунок 3.32 «Вигляд каталогу аукціонів»

Перевіримо створення аукціонів, для цього перейдемо до вкладки «Створити лот». Також відкриємо додаток ще раз, авторизуємось через іншу адресу, щоб перевірити відображення на інших клієнтах, та можливість ставок на аукціон.

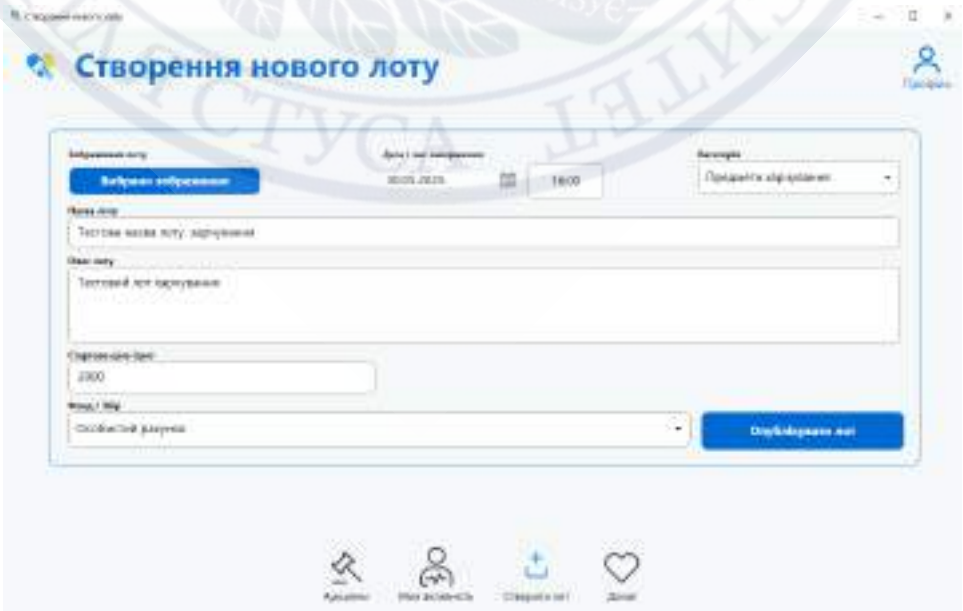

Рисунок 3.33 «Вигляд вікна створення аукціонів»

На іншому аукціоні можна спостерігати новий аукціон, в відповідній категорії:

Рисунок 3.34 «Новий аукціон в вікні каталогу аукціонів»

Відкриємо аукціон, та спробуємо зробити нову ставку, попередньо оновивши баланс користувача з тестовим аккаунтом:

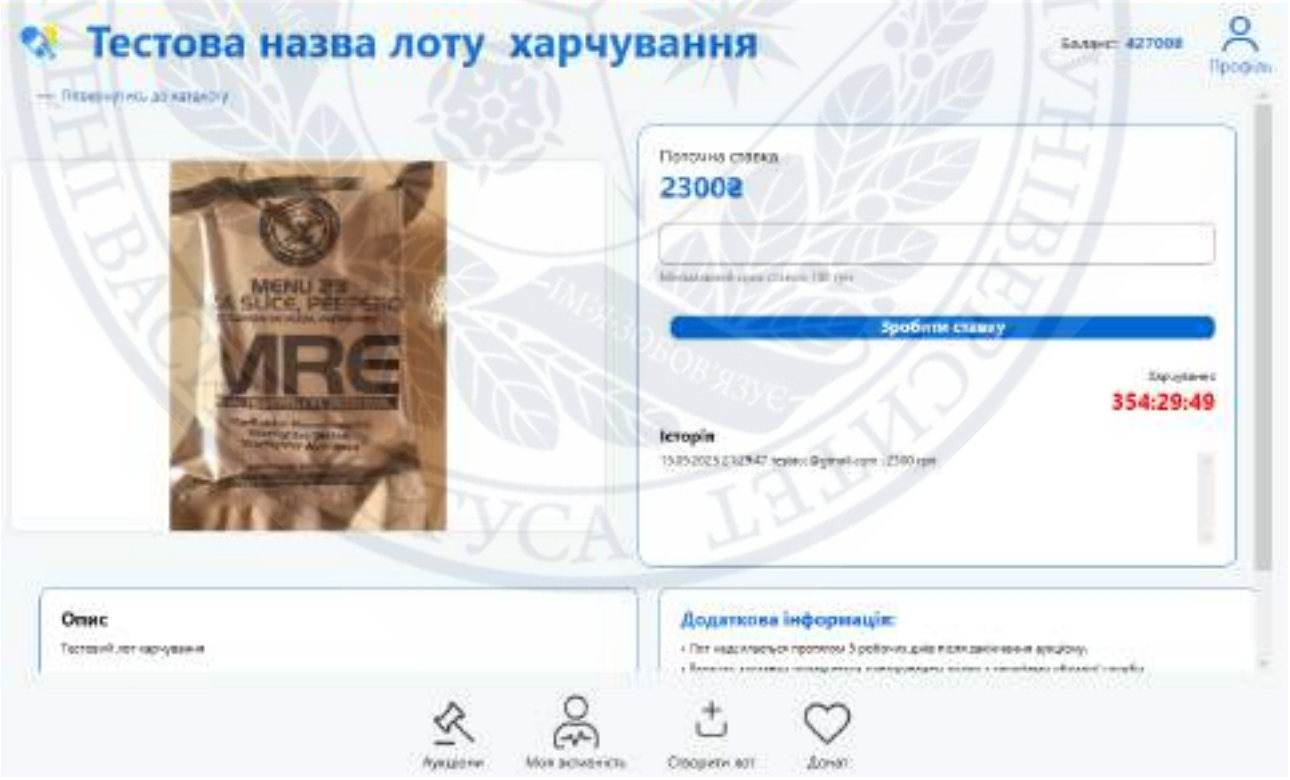

Рисунок 3.35 «Вікно деталей аукціонів з новою ставкою»

В результаті можна побачити нову ставку, ціну, яка змінилась відносно ставки, та оновлення балансу користувача.

Останнім кроком буде перевірка закінчення аукціону. Для цього змінимо час аукціонів, а адресу тестового аккаунта на реальну адресу, щоб перевірити листи власника аукціону та переможця аукціону.

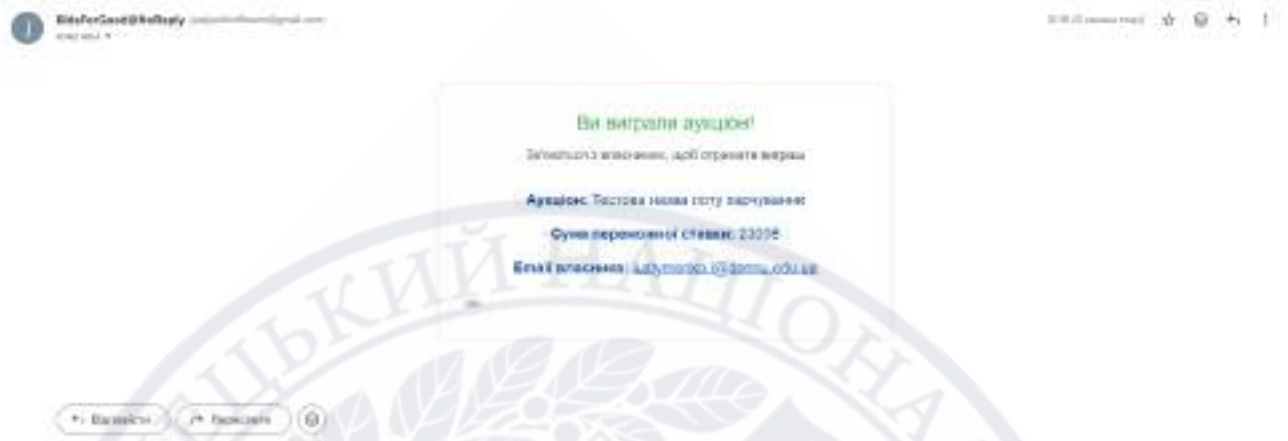

Рисунок 3.36 «Лист переможця аукціона»

Також можна побачити, що власнику аукціона був надісланий лист про завершення аукціону:

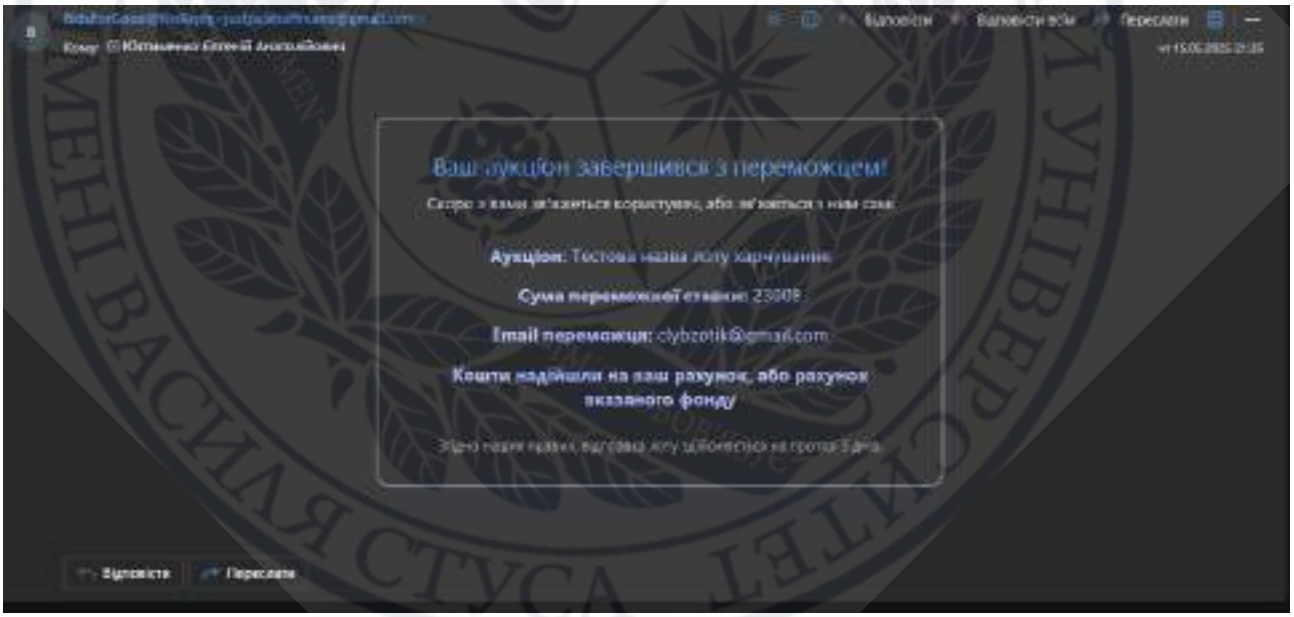

Рисунок 3.37 «Лист власника аукціона»

В результаті тестування можна побачити, що весь функціонал проведення аукціонів працює правильно, реалізована реєстрація та авторизація, створення та відображення аукціонів, їх правильне завершення.

ВИСНОВКИ

У процесі виконання бакалаврської роботи було досліджено, спроектовано та реалізовано серверну частину десктопного додатку, що має на меті спростити процес організації та проведення благодійних онлайн-аукціонів.

На основі аналізу предметної області було встановлено, що соціальні мережі та онлайн платформи відіграють ключову роль у популяризації волонтерських та благодійних ініціатив, але спеціалізованих інструментів, орієнтованих виключно на благодійність, бракує, що залишає вільну нішу для створення такого продукту. Це підтвердило актуальність створення окремого додатку, який забезпечить надійне та зручне середовище, для організаторів та учасників таких ініціатив.

У хоті роботи було здійснено аналіз сучасних мов програмування та технологій для них, що дало змогу обгрунтовано обрати найбільш придатні інструменти та архітектурні рішення. Було реалізовано клієнт-серверну модель взаємодії за допомгою REST API, забезпечено авторизацію користувачів, реалізовано обробку запитів у реальному часі, за допомогою технологій SignalR, а також спроектовано та реалізовано реляційну базу даних, з чітко визначеною структурою таблиць та зв’язків між ними.

Результатом виконання роботи є прототип додатку онлайн-аукціонів, що відповідає сучасним вимогам до якості програмного забезпечення – масштабованості, гнучкості, продуктивності. Реалізоване рішення може стати достойною основою для подальшого розвитку повноцінної платформи, спрямованої на підтримку благодійної діяльності в теперішньому світі.

Отриманні знання і практичні навички з розробки клієнт серверних систем, роботи з базами даних стали підтвердженням здобутих у процесі навчання компетентностей.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Monobank – банк у телефоні. URL: https://monobank.ua/ (дата звернення 11.03.2025)
2. Prozorro.Sale – електронна торгівля. URL: https://prozorro.sale/ (дата звернення 11.03.2025)
3. Військовий аукціон Reibert. URL: https://reibert.info/ (дата звернення 11.03.2025)
4. eBay For Charity. URL: https://charity.ebay.com/ (дата звернення 14.03.2025)
5. Charity Fundraising Auctions for Schools & Nonprofits. URL: https://www.biddingforgood.com/ (дата звернення 14.03.2025)
6. Fundraising Software for Nonprofits. URL: https://www.charityauctionstoday.com/ (дата звернення 14.03.2025)
7. Ceder N. The Quick Python Book, Fourth Edition. – Maning Co., 2025, 584 с.
8. Farrell J. Java Programming. – 9-те вид. - Cengage Learning, 2020. 1024 с.
9. Freeman A. Essential TypeScript 5, Third Edition – Maning Co., 2023, 568 с.
10. Griffiths I. Programming C# 12: Build Cloud, Web, and Desktop Applications. - O'Reilly Media, 2024. 873 с.
11. Price M. C# 13 and .NET 9 – Modern Cross-Platform Development Fundamentals. – Packt Publishing, 2024. 828 с.
12. AltexSoft. The Good and the Bad of C# Programming. URL: https://www.altexsoft.com/blog/c-sharp-pros-and-cons/ (дата звернення 24.03.2025)
13. Code Quickly. Learn C++ Quickly: A Complete Beginner's Guide to Learning C++, Even If You're New to Programming. - Independently published, 2020. 175 с.
14. Pangea.ai. A Comprehensive Guide to C++: Advantages and Disadvantages. URL: https://pangea.ai/resources/a-comprehensive-guide-to-c-advantages-and-disadvantages (дата звернення 24.03.2025)
15. Advantages and Disadvantages of C++ . URL: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-cpp/ (дата звернення 24.03.2025)
16. A Programmers Guide to Python: Advantages & Disadvantages | Linode Docs https://www.linode.com/docs/guides/pros-and-cons-of-python/ (дата звернення 25.03.2025)
17. Pros and Cons of Java. URL: https://data-flair.training/blogs/pros-and-cons-of-java/ (дата звернення 25.03.2025)
18. TypeScript vs JavaScript. URL: https://www.altexsoft.com/blog/typescript-pros-and-cons/ (дата звернення 25.03.2025)
19. Advantages of JavaScript. URL: https://softjourn.com/insights/the-advantages-and-disadvantages-of-javascript (дата звернення 25.03.2025)
20. Відмінності мов програмування. URL: https://foxminded.ua/vidminnosti-mov-prohramuvannia/ (дата звернення 26.03.2025)

21. Рейтинг мов програмування в 2024. DOU.UA. URL: https://dou.ua/lenta/articles/language-rating-2024/(дата звернення. 28.03.25)
22. Рейтинг мов програмування в 2025. DOU.UA. URL: https://dou.ua/lenta/articles/language-rating-2025/ (дата звернення. 28.03.25)
23. Розуміння клієнт-серверної архітектури. DOU.UA URL: https://dou.ua/forums/topic/44636/ (дата звернення. 29.03.25)
24. Колбек-функції. URL: https://uk.javascript.info/callbacks (дата звернення. 29.03.25)
25. Сучасні клієнт-серверні технології та їх застосування при вивченні систем управління базами даних. URL: https://sj.udu.edu.ua/index.php/kosn/article/view/620/581 (дата звернення. 30.03.25)
26. Burns S. Hands-On Network Programming with C# and .NET Core. – Packt Publishing, 2018. – 398 с.
27. Клієнт-серверна архітектура. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення. 30.03.25)
28. Що таке REST API?. URL: https://foxminded.ua/shcho-take-rest-api/ (дата звернення 04.04.25)
29. Masse M. REST API Design Rulebook. - O'Reilly Media, - 112c.
30. Documentation gRPC. URL: https://grpc.io/docs/ (дата звернення 04.04.25)
31. Базове розуміння OAuth 2.0. URL: https://stfalcon.com/uk/blog/post/oauth-2.0 (дата звернення 05.04.25)
32. JWT Introduction. URL: https://jwt.io/introduction (дата звернення 06.04.25)
33. Chakraborty R. Securing the Web: Machine Learning for CSRF Vulnerability Detection. LAP LAMBERT Academic Publishing, 2022. 76 p.
34. Lock A. ASP.NET Core in Action. 3rd Edition. Manning Publications, 2024. 650 p.
35. WPF vs WinForms. URL: https://www.bytehide.com/blog/wpf-vs-winforms (дата звернення 08.04.25)
36. Ben-Gan I. T-SQL Fundamentals. 4th Edition. Microsoft Press, 2023. 464 p.
37. SignalR Introduction. URL: https://binary-studio.com/blog/signalr-introduction/ (дата звернення. 18.04.25)
38. Мережеві протоколи та їх призначення. URL: https://deltahost.ua/ua/tipi-merezhevix-protokoliv-i-ih-priznachennya-http-ip-ssh-ftp-pop3-mac.html (дата звернення. 18.04.25)
39. Юстименко Є.А., Труханська В.О, Зелінська О.В. Проектування об’єктно-орієнтовних баз даних. Прикладні інформаційні технології, 2022.
40. Зелінська О.В, Коновалюк І.Л. Забезпечення безпеки баз даних.- Комп'ютерні технології обробки даних, 2024. С.34-38
41. Database Scheme Diagrams. URL: https://drawsql.app/ (дата звернення 29.04.2025)

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАСИЛЯ СТУСА
ALMA MATER
NOMEN EST OMEN
ІМ'Я ЗОБОВ'ЯЗУЄ

ДОДАТКИ

ДОДАТОК А

Лістинги класів серверної частини

```
namespace AuctionsServer.Repositories
{
   public class AuctionRepository : IAuctionRepository
   {
      private readonly AuctionDbContext _context;
      public AuctionRepository(AuctionDbContext context)
      {
         _context = context;
      }
      public async Task AddAuctionAsync(Auction auction)
      {
         await _context.Auctions.AddAsync(auction);
      }
      public async Task AddAuctionPicture(AuctionPicture picture)
      {
         await _context.AuctionPictures.AddAsync(picture);
      }
      public async Task AddBidToAuctionAsync(Bid bid)
      {
         await _context.Bids.AddAsync(bid);
      }
      public async Task DeleteAuctionAsync(int id)
      {
         await _context.Auctions.Where(a => a.IdAuction ==
id).ExecuteDeleteAsync();
      }
```

```
    public async Task DeleteBidAsync(int actionId, int bidId)
    {
        await _context.Bids.Where(b => b.IdAuction == actionId && b.IdBid ==
bidId).ExecuteDeleteAsync();
    }
    public async Task<IEnumerable<Auction?>> GetAllAuctionsAsync()
    {
        var auctionList = await _context.Auctions.Include(a => a.Owner)
                                        .Include(a => a.AuctionPictures)
                                        .Include(a => a.Bids).ThenInclude(b =>
b.IdUsersNavigation).
                                        ToListAsync();
        return auctionList;
    }
    public async Task<IEnumerable<Bid?>> GetAllBidsByAuctionIdAsync(int
auctionId)
    {
        return await _context.Bids.Where(b => b.IdAuction ==
auctionId).ToListAsync();
    }
    public async Task<IEnumerable<AuctionPicture?>>
GetAllPicturesByAuctionIdAsync(int auctionId)
    {
        return await _context.AuctionPictures.Where(p => p.IdAuction ==
auctionId).ToListAsync();
    }
    public async Task<Auction?> GetAuctionByIdAsync(int id)
    {
        return await _context.Auctions.Include(a => a.Owner)
                                .Include(a => a.AuctionPictures)
```

```
                .Include(a => a.Bids).ThenInclude(b =>
b.IdUsersNavigation).
                FirstOrDefaultAsync(a => a.IdAuction == id);
        }
        public async Task UpdateAuctionAsync(Auction auction)
        {
            await _context.Auctions.Where(a => a.IdAuction ==
auction.IdAuction).ExecuteUpdateAsync(a => a
                .SetProperty(a => a.OwnerId, auction.OwnerId)
                .SetProperty(a => a.StatusId, auction.StatusId)
                .SetProperty(a => a.CategoryId, auction.CategoryId)
                .SetProperty(a => a.Title, auction.Title)
                .SetProperty(a => a.Description, auction.Description)
                .SetProperty(a => a.StartPrice, auction.StartPrice)
                .SetProperty(a => a.StartTime, auction.StartTime)
                .SetProperty(a => a.EndTime, auction.EndTime)
            );
        }
        public async Task SaveChangesAsync()
        {
            await _context.SaveChangesAsync();
        }
    }
}
namespace AuctionsServer.Repositories
{
    public class LogRepository : ILogRepository
    {
        private readonly AuctionDbContext _context;
```

ДОДАТОК Б

Лістинги класів клієнтської частини

```
namespace AuctionsDesktopApp.Controllers
{
   public class AuthController
   {


      public async Task<bool> loginWithPassAsync(string email, string password)
      {
         string result = await
GetAsync($"auth/loginByPass?email={Uri.EscapeDataString(email)}&password={U
ri.EscapeDataString(password)}");


         if(result == "Loged by pass") return true;
         else return false;
      }


      public async Task registerUser(string email, string password, string username)
      {
         await
PostAsync($"auth/registerUser?email={Uri.EscapeDataString(email)}&password={
Uri.EscapeDataString(password)}&username={Uri.EscapeDataString(username)}");
         return;
      }


      public async Task<bool> isEmailExist(string email)
      {
         string result = await
GetAsync($"auth/isExist?email={Uri.EscapeDataString(email)}");
```

```
        try { return Convert.ToBoolean(result); }
        catch { return false; }
    }

    public bool isEmailBanned(string email)
    {
        //todo check banned
        if (email == "banned@gmail.com") return true;
        else return false;
    }

    public bool isCoolDown()
    {
        //todo check kd
        return false;
    }

    public async Task sendEmailCode(string email, string code)
    {
        await
PostAsync($"auth/sendCode?email={Uri.EscapeDataString(email)}&code={Uri.Esc
apeDataString(code)}");
        return;
    }

    public bool isValidEmail(string email)
    {
        try
        {
```