МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЧЕРНИШЕНКО ЯРОСЛАВ АНАТОЛІЙОВИЧ

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

РОЗРОБКА ФРОНТЕНД – ЧАСТИНИ ДЕСКТОПНОГО ДОДАТКУ ДЛЯ БЛАГОДІЙНИХ АУКЦІОНІВ: UI/UX ТА ІНТЕРАКТИВНІ ФУНКЦІЇ

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

Чернишенко Я.А. Розробка фронтенд-частини десктоп-додатку для благодійних аукціонів: UI/UX та інтерактивні функції. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі досліджено процес розробки інтерфейсу десктоп-додатку для проведення благодійних аукціонів. Визначено функціональні та нефункціональні вимоги до front-end частини, сформовано макети інтерфейсів користувача з урахуванням принципів UI/UX-дизайну. Розробка здійснювалась із використанням Figma для проектування макетів, а також технологій WPF відповідно до обраної архітектури.

Ключові слова: десктоп-додаток, благодійний аукціон, front-end, інтерфейс користувача, UI/UX, Figma, WPF.

69 ст. 27 рис., 4 табл., 1 дод., 30 джерел.

ABSTRACT

Chernyshenko Y. Development of the Front-End Part of a Desktop Application for Charity Auctions: UI/UX and Interactive Functions. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2025.

The bachelor's qualification thesis explores the process of developing the user interface of a desktop application for conducting charity auctions. Functional and non-functional requirements for the front-end part were defined, and user interface mockups were created in accordance with UI/UX design principles. The development was carried out using Figma for UI prototyping and WPF technologies according to the chosen architecture.

Keywords: desktop application, charity auction, front-end, user interface, UI/UX, Figma, WPF.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – інтерфейс прикладного програмування; набір протоколів і засобів для створення взаємодії між клієнтом і сервером.

DTO (Data Transfer Object) – об'єкт передачі даних; клас, що використовується для обміну інформацією між системними шарами.

БД (База Даних) – організоване сховище структурованої інформації, яку можна легко зберігати, змінювати та отримувати за допомогою запитів. У роботі використано СУБД MySQL.

Figma – хмарний сервіс для проектування макетів UI/UX інтерфейсів і спільної роботи дизайнерів та розробників.

Front-end – клієнтська частина додатку; інтерфейс користувача, з яким безпосередньо взаємодіє користувач.

MVVM (Model–View–ViewModel) – архітектурний шаблон програмного забезпечення, що розділяє логіку, дані і представлення.

SignalR – бібліотека для реалізації обміну даними в реальному часі між сервером і клієнтом.

WPF (Windows Presentation Foundation) – технологія Microsoft для створення графічних інтерфейсів у десктопних застосунках на платформі .NET.

XAML (Extensible Application Markup Language) – мова розмітки, яка використовується в WPF для опису інтерфейсу користувача.

UI (User Interface) – інтерфейс користувача; сукупність візуальних елементів програми, з якими працює користувач.

UX (User Experience) – користувацький досвід; сприйняття та реакція користувача під час взаємодії з інтерфейсом.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту; використовується для обміну даними між клієнтом і сервером.

JSON (JavaScript Object Notation) – текстовий формат обміну даними між клієнтом і сервером.

REST (Representational State Transfer) – архітектурний стиль API для взаємодії з ресурсами через HTTP-запити.

ObservableCollection – тип колекції в .NET, що сповіщає інтерфейс про зміни у даних (додавання, видалення, оновлення).

Back-end – серверна частина додатку, що відповідає за обробку даних, бізнес-логіку, збереження в БД та API-запити.

ВСТУП

У сучасних реаліях цифрового світу благодійність набуває нових форм. Допомога військовим, волонтерам, медичним установам та іншим соціальним ініціативам активно здійснюється через онлайн-платформи, соціальні мережі та спеціалізовані десктоп-додатки. Одним з перспективних напрямів є використання благодійних аукціонів - форматів, у яких користувачі змагаються за унікальні лоти, а зібрані кошти направляються на підтримку суспільно важливих потреб.

Попри наявність численних вебресурсів, які дозволяють збирати кошти, більшість із них мають обмежену функціональність або не пристосовані для проведення повноцінних інтерактивних аукціонів. Також спостерігається відсутність акценту на естетику, інтуїтивність та зручність інтерфейсів, що безпосередньо впливає на користувацький досвід і рівень залучення потенційних учасників.

Тому виникає потреба у створенні сучасного та зрозумілого десктоп-додатку з ефективним інтерфейсом, що дозволяє користувачам брати участь в онлайн-аукціонах, взаємодіяти із благодійними організаціями та легко орієнтуватися в системі.

Об’єктом дослідження є процес розробки та функціонування програмного додатку для проведення благодійних онлайн-аукціонів. Це включає сукупність технічних, функціональних та користувацьких аспектів створення системи, яка дозволяє користувачам брати участь в аукціонах, здійснювати ставки, переглядати лоти, а також сприяти збору коштів на підтримку волонтерських ініціатив.

Предметом дослідження є проектування та реалізація front-end частини десктоп-додатку, що охоплює створення макетів інтерфейсу, побудову UI/UX логіки, розробку інтерактивних елементів управління, адаптацію візуального оформлення під різні сценарії використання та забезпечення ефективної взаємодії клієнтської частини з серверною. Особлива увага приділяється

зручності користування, візуальній привабливості інтерфейсу та відповідності сучасним принципам проектування інтерфейсів.

Метою роботи є створення ефективного, привабливого та зручного для користувача інтерфейсу благодійного десктоп-додатку, що сприятиме залученню більшої кількості учасників і підвищенню активності в аукціонах.

Завдання дослідження: проаналізувати сучасні підходи до розробки UI/UX у схожих системах; дослідити вимоги до фронтенд – додатків з точки зору зручності та доступності; розробити прототип інтерфейсу в Figma; сформувати дизайн – систему для візуальної цілісності; створити макети основних вікон програми; забезпечити технічну можливість інтеграції з front-end.

Теоретичне та практичне значення отриманих результатів полягає в можливості використання розробленої інтерфейсної частини в реальних умовах – як прикладного додатку для організації благодійних заходів або як шаблону для майбутніх аукціонних платформ. Крім того, напрацьовані шаблони та логіка UX можуть бути адаптовані для інших соціальних проєктів.

Апробація результатів дослідження представлялись на конференції VI Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених

Структура роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. У першому розділі подано аналіз предметної області та вимог до front-end. У другому – описано інструменти та технології розробки, архітектуру та UX–рішення. Третій розділ присвячено реалізації та тестуванні інтерфейсу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО СИСТЕМИ

1.1 Огляд та аналіз існуючих онлайн-аукціонів

У межах розробки десктоп-додатку для благодійних аукціонів, першочерговим завданням стало вивчення існуючих онлайн-рішень, які реалізують механізми торгів, ставок та збору коштів через цифрові платформи. Оскільки мій проєкт має на меті поєднати зручний інтерфейс, повну автоматизацію і водночас гуманітарну ціль – фінансування волонтерських ініціатив та підтримку ЗСУ, – необхідно було проаналізувати найпоширеніші підходи до побудови аукціонних систем у комерційному, державному та благодійному сегментах.

Такий аналіз дозволяє не лише зрозуміти, які функціональні рішення вже існують, але й виявити їх обмеження в контексті благодійності: відсутність емоційної залученості, складність інтерфейсів, низька адаптованість до українських реалій. Саме тому було розглянуто декілька найвідоміших платформ, які реалізують різні типи аукціонів: від класичних комерційних до соціальних, некомерційних та волонтерських. Цей огляд є базисом для формування вимог до майбутнього продукту – сучасного, адаптивного та зрозумілого десктоп-додатку для проведення благодійних аукціонів.

Онлайн-аукціони стали невід’ємною частиною сучасної цифрової економіки. Вони забезпечують динамічний спосіб продажу товарів або послуг через конкурентні ставки. На відміну від звичайної купівлі-продажу за фіксованою ціною, у аукціонах кожен учасник має шанс виграти лот, запропонувавши найвищу ціну за обмежений час. Це створює елемент гри, мотивації та змагального духу, який робить аукціони популярними серед користувачів різного віку [1].

Сьогодні існує велика кількість онлайн-аукціонів, які відрізняються як за цільовою аудиторією, так і за функціональністю. Залежно від призначення,

інтерфейсу та доступності можна виділити кілька основних типів онлайн-аукціонів:

- **Класичні комерційні аукціони** – спрямовані на продаж товарів між фізичними особами або компаніями (eBay, OLX–аукціони);
- **Державні аукціони** – проводяться через офіційні платформи для відкритих торгів з майном чи правами (Prozorro.Продажі, СЕТАМ);
- **Благодійні аукціони** – організовуються для збору коштів на гуманітарні або соціальні цілі (United24, Charitybuzz, Auction for Ukraine).

eBay – лідер класичного аукціону

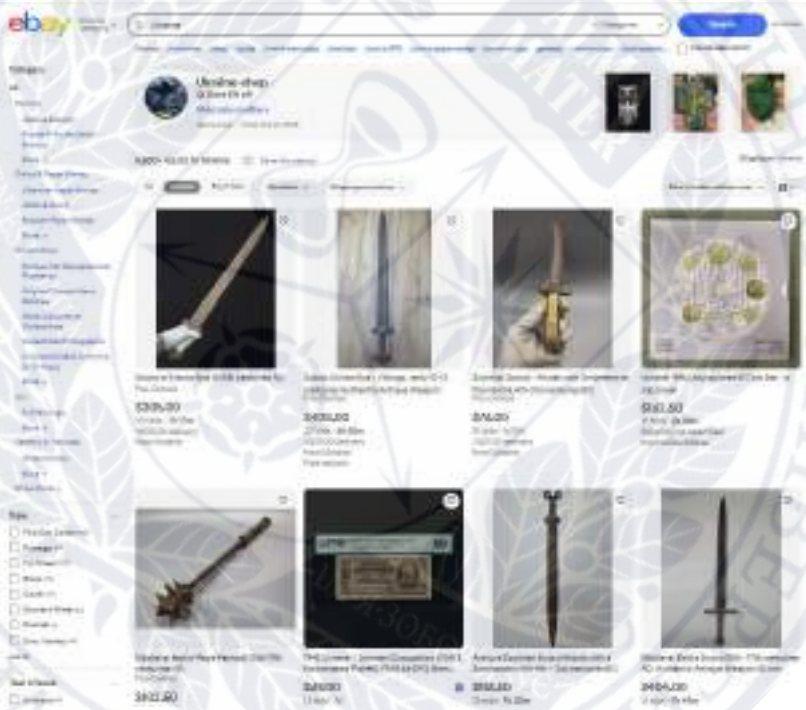


Рисунок. 1.1 – Сторінка активних лотів на eBay

eBay є однією з найвідоміших і наймасштабніших онлайн-платформ, яка поєднує функції інтернет-магазину та аукціону, що працює з 1995 року [2]. На цій платформі користувачі можуть не лише купувати товари за фіксованими цінами, але й брати участь в аукціонних торгах. eBay підтримує зручний, інтуїтивний інтерфейс, багатомовність, а також мобільну версію для користувачів смартфонів.

Відмінною рисою платформи є її високий рівень автоматизації – ставки обробляються миттєво, користувачі отримують сповіщення про зміну ціни, а оплата здійснюється через інтегровані платіжні системи (PayPal, банківські картки). Платформа має розвинену систему зворотного зв’язку, де продавець і покупець можуть залишати оцінки, що формує довіру у спільноті.

Проте eBay є суто комерційним проєктом, і його механізми не призначені для благодійних кампаній. Крім того, користування платформою для українських продавців обмежується мовними бар'єрами, платними сервісами, податковими складнощами та високими комісіями.

Інтерфейс eBay вважається зручним, але класичним і перевантаженим. Дизайн орієнтований на практичність: велика кількість елементів на екрані, фокус на інформаційне наповнення, а не візуальну естетику. З точки зору UX – користувач отримує всі функції одразу, але це ускладнює навігацію, особливо для новачків. Сторінки побудовані за принципом комерційного каталогу, тому відсутня будь-яка емоційна залученість. Брендинг мінімальний, візуальні асоціації з місією чи соціальною цінністю відсутні.

Prozorro.Продажі – державна платформа України

Рисунок. 1.2 – Приклад торгів на Prozorro.Продажі

Prozorro.Продажі – це офіційна державна платформа, яка функціонує в Україні і є частиною більшої системи відкритих закупівель [3]. Вона створена для реалізації державного та комунального майна, ліцензій, дозволів тощо. Основна концепція платформи – «всі бачать все». Це означає, що інформація про лоти, учасників та результати торгів є повністю відкритою.

Система Prozorro забезпечує високий рівень звітності, інтеграцію з Єдиним державним реєстром, підтримку API для зовнішніх електронних майданчиків. Процес участі в торгах строго регламентований, що мінімізує корупційні ризики.

Однак інтерфейс Prozorro орієнтований переважно на юридичних осіб і потребує спеціальних знань для участі. Платформа не адаптована під звичайного користувача або волонтера, а також не підтримує кастомізацію під благодійні формати чи донат - орієнтовані кампанії.

Дизайн Prozorro строго функціональний, з домінуванням текстових блоків, таблиць, технічних фільтрів. Інтерфейс не пристосований для кінцевого споживача – він радше нагадує внутрішню CRM або портал для юристів. Кольорова схема нейтральна, однак не відповідає сучасним принципам UI-дизайну – бракує візуальних акцентів, ієрархії блоків, анімацій чи підказок. UX побудовано на лінійній логіці кроків, без урахування потреб користувача, який хоче швидко зробити дію. У контексті благодійності – цей підхід занадто «сухий» і формальний.

Charitybuzz – американська платформа, яка спеціалізується на благодійних аукціонах, де лотами виступають не стільки предмети, як унікальні «досвіди» [4]. Це можуть бути вечеря з відомою особистістю, участь у зйомках, доступ на ексклюзивні події тощо. Усі зібрані кошти спрямовуються на потреби затверджених некомерційних організацій.

Сайт Charitybuzz має сучасний, естетично привабливий інтерфейс, продуману навігацію, захищені механізми платежів і чіткий процес модерації. Участь у торгах можуть брати користувачі з усього світу, що забезпечує широкий охоплення та збільшує потенціал залучення коштів.

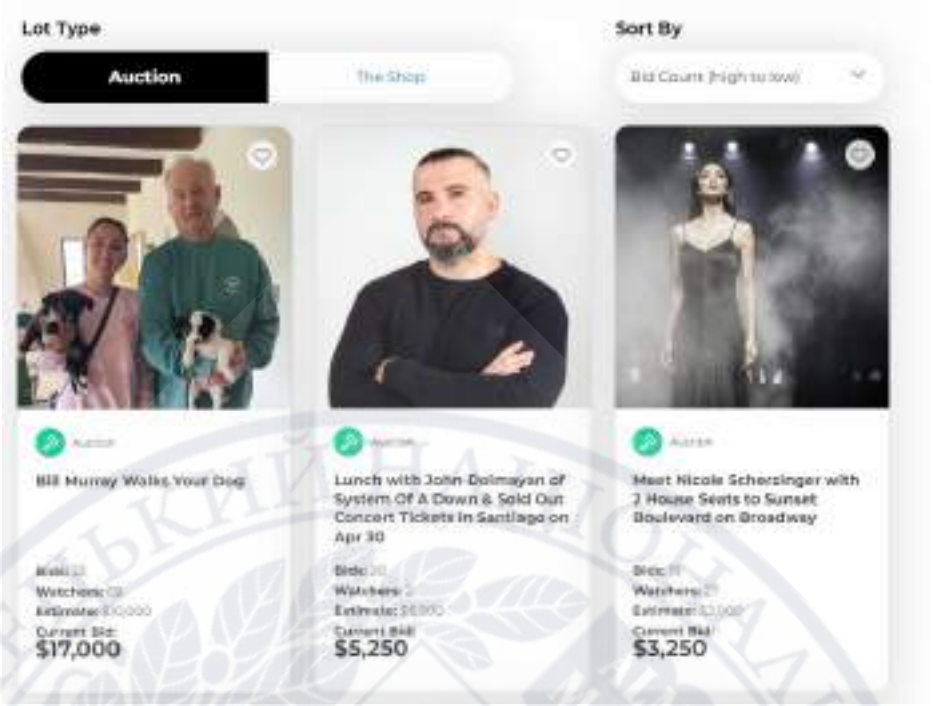


Рисунок 1.3 – Сторінка аукціонних пропозицій на Charitybuzz

Серед обмежень варто зазначити високу вартість входу для організаторів, складну систему перевірки благодійних фондів, а також англомовну орієнтацію. Платформа не має підтримки локалізації або можливості адаптації під локальні контексти, як-от волонтерські ініціативи в Україні.

Charitybuzz має продуманий, преміальний візуальний стиль. Основні акценти зроблені на великих фото, привабливих типографічних заголовках і просторій сітці, що створює відчуття ексклюзивності. UI мінімалістичний, але не бідний: є плавні переходи, ефекти наведення, спливаючі вікна з деталями. UX побудовано на емоції – користувача супроводжує легка анімація, проста структура і персоналізовані звернення. Основна вада – вся логіка спрямована на англомовну аудиторію з певним рівнем достатку. Інтерфейс складно адаптувати під локальні, українські потреби або проєкти з невеликим бюджетом.

Platfor.ma + dobro.ua – Instagram–аукціон для п’яти благодійних фондів

Рисунок 1.4 – Аукціонна кампанія Platfor.ma + dobro.ua у мережі Instagram

У лютому 2023 року українська платформа інноваційних і соціальних проєктів **Platfor.ma** у співпраці з благодійним фондом **dobro.ua** організувала тематичний благодійний аукціон у форматі Instagram–публікації [5]. Лотом став ексклюзивний светр «Волинь» із лімітованої серії «Україна». Аукціон був проведений із метою збору коштів для п’яти перевірених благодійних організацій [6].

Зібрані кошти передавались одразу п’яти фондам, які займаються допомогою дітям, літнім людям, тваринам, а також розмінуванням територій

Попри відсутність спеціалізованої платформи, сам аукціон мав високу ефективність завдяки простоті, прозорості та залученню довірених медіа. Соціальна мережа Instagram стала не просто каналом комунікації, а майданчиком для активної взаємодії, що дозволило залучити широку аудиторію та зібрати значні кошти у дуже стислі терміни.

У цьому випадку інтерфейс формально не створювався – UI/UX цілком залежав від можливостей Instagram. Проте це й є головною перевагою: знайоме, нативне середовище, де користувачі вже звикли взаємодіяти з постами, коментувати й реагувати на сторіз.

United24 – державна благодійна платформа

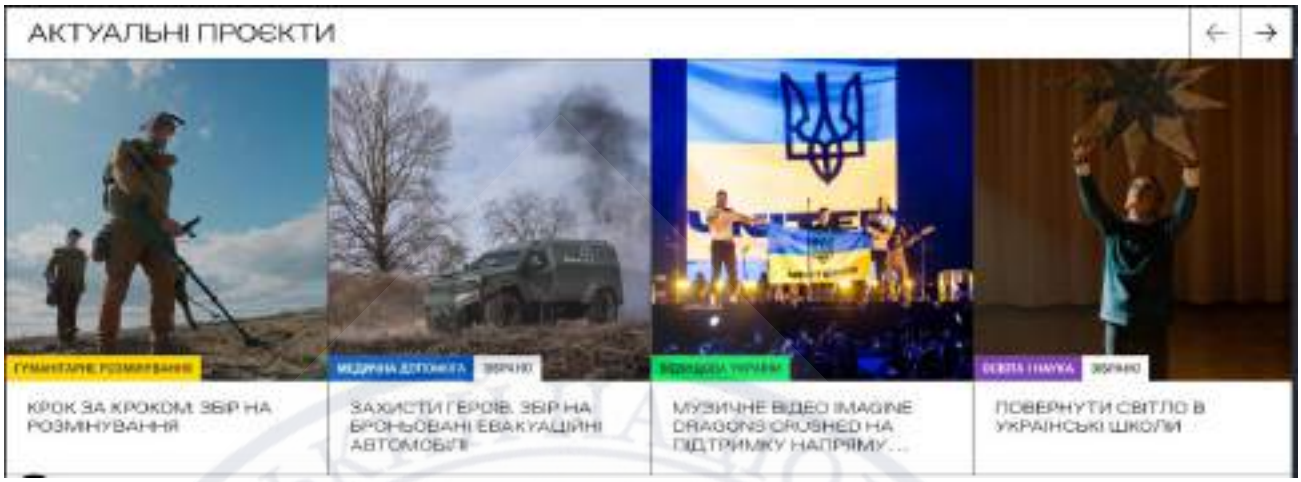


Рисунок 1.5 - Актуальні проєкти на платформі United24

United24 – це офіційна ініціатива, започаткована Президентом України, яка об'єднує всі напрями допомоги у межах єдиної платформи: Збройні Сили, медицина, відбудова [7] . У рамках діяльності ініціативи періодично проводяться благодійні аукціони. Наприклад, у лотах пропонувалися особисті речі відомих людей, ексклюзивні артефакти, унікальні сертифікати.

Завдяки державній підтримці, United24 має високу репутацію, масове охоплення аудиторії та співпрацює з великими брендами. Усі надходження фіксуються офіційно, а результати аукціонів широко висвітлюються в медіа.

Проте аукціони United24 не інтегровані в постійну платформу. Вони проводяться вручну або через партнерські ресурси, що не забезпечує автоматизації процесу. Це знижує ефективність для регулярного проведення торгів і унеможливлює масштабування в реальному часі.

United24 має сучасний, дизайн, витриманий у стриманих тонах, що викликає довіру та відчуття офіційності. Усі елементи інтерфейсу – чітко структуровані: зрозумілі кнопки, блоки зі статистикою, фокус на прозорості. Проте візуальна мова – скоріше інформативна, ніж емоційна. Платформа не використовує виразних візуальних образів благодійності чи соціального залучення – відсутня гейміфікація, взаємодія, динамічні елементи. UX

прямолінійний і більше підходить для одноразового внеску, а не для залучення в інтерактивні процеси, як-от участь у ставках.

Табл. 1.1 Порівняння аукціонних платформ

Платформа	Тип аукціону	Основна ціль	Унікальність	Недоліки
eBay	Комерційна	Продаж товарів	Автоматизовані ставки, глобальна аудиторія, захист транзакцій	Відсутність благодійного спрямування, складність для новачків
Prozorro.Продажі	Державна	Продаж держмайна	Антикорупційна модель, відкритий API, повна прозорість	Складний інтерфейс, розраховано на юросіб, не підходить для волонтерства
Charitybuzz	Благодійна	Збір коштів через «емоційні» лоти	Якісний UX, інтернаціональна аудиторія, ексклюзивні лоти	Високий поріг входу, англомовність, складна модерація
dobro.ua / Platfor.ma	Благодійна (Instagram)	Збір коштів для кількох фондів	Мобільність, простота, охоплення через соцмережі	Відсутність автоматизації, повністю ручна модерація
United24	Держ. благодійна	Національні гуманітарні ініціативи	Державна підтримка, публічні кампанії, довіра	Фрагментованість, ручне адміністрування, відсутність додатку

Попри різноманітність аукціонних платформ, жодна з них не забезпечує повноцінного продукту з благодійним фокусом, зручним інтерфейсом і автоматизацією процесів. В існуючих рішеннях спостерігаються:

- обмежена інтерактивність або взагалі її відсутність;
- складність для нових організаторів або пересічних користувачів;
- залежність від соцмереж, ручного модераторства або зовнішніх сервісів;
- фрагментація системи ставок, повідомлень, безпеки.

Це формує чіткий запит на створення нового продукту – аукціонного додатку, який:

- забезпечить простий і красивий інтерфейс;

- дозволить повністю автоматизувати аукціонну логіку;
- буде адаптований до українських реалій і соціальних ініціатив;
- поєднає UX, чітке та логічне розуміння, прозорість і довіру.

Таким чином, проект розробки front-end частини благодійного десктоп-додатку є не лише актуальним з технічної точки зору, а й суспільно важливим внеском у розвиток волонтерської та цифрової інфраструктури України.

1.2 Формулювання функціональних та нефункціональних вимог до front-end частини

Формулювання вимог до користувацького інтерфейсу є одним із ключових етапів під час створення будь-якої програмної системи, особливо у випадках, коли успіх взаємодії залежить від залученості користувачів, як у випадку благодійних платформ. Інтерфейс несе в собі не лише функціональне навантаження, а й емоційний контакт із людиною. Він має бути простим, зрозумілим і таким, що надихає до дії. Якщо користувач почувається розгублено або перевантажено через складну навігацію, неочевидні кнопки чи перенасиченість екрану, він скоріше за все закриє додаток. Особливо це критично для благодійного аукціону, де кінцева мета – не лише участь в торгах, а й добровільна допомога. Тому від якості інтерфейсу залежить не просто задоволеність користувача, а й те, чи наважиться він зробити внесок [8].

Під час проектування інтерфейсу надзвичайно важливо чітко розуміти, для кого створюється система. У випадку благодійного аукціону маємо справу із трьома основними групами користувачів – звичайні учасники, організатори та адміністратори. Кожна з цих категорій має власні потреби, рівень технічної обізнаності та специфіку користування. Звичайні учасники хочуть максимально просто знайти лот, зробити ставку та відчути емоцію перемоги. Їм важливо бачити зручні картки, таймери, візуальні підтвердження дій, прозору історію ставок. Організаторам, в свою чергу, потрібен інструмент для створення та керування аукціонами: зручна форма заповнення, попередній перегляд,

можливість редагування. Адміністратори ж потребують повного контролю: перегляд системної активності, фільтрація користувачів, модерація контенту. Врахування цих відмінностей дозволяє зробити гнучкий і адаптивний інтерфейс.

Щодо функціональності інтерфейсу, він повинен покривати всі ключові сценарії взаємодії. Перш за все, це механізм реєстрації та авторизації. Додаток має підтримувати створення нового облікового запису з перевіркою пошти, а також можливість входу за допомогою соціальних мереж. Після успішної авторизації користувач повинен бачити лише ті функції, які відповідають його ролі. Для учасника – перегляд лотів і участь у торгах, для організатора – управління лотами, для адміністратора – панель контролю.

Центральною функцією є перегляд та участь в аукціонах. На головній сторінці має бути реалізована скролювана стрічка активних лотів, оформлена у вигляді інтерактивних карток з фотографією, назвою, ціною та таймером. Передбачено також можливість фільтрації і сортування – за категорією, актуальністю, часом завершення. Коли користувач натискає на лот, відкривається сторінка з деталізованим описом, великим зображенням, таймером, полем для ставки та кнопкою підтвердження. Інтерфейс повинен перевіряти коректність введеної суми, нагадувати про мінімальний крок і запобігати відправленню помилкових даних. Всі ці дії повинні супроводжуватись системою сповіщень – як на екрані, так і на пошту, щоб користувач не пропустив важливий момент, наприклад, коли його ставку перебили.

Окрему увагу заслуговує особистий кабінет. Це місце, де користувач бачить свій вплив: скільки лотів виграв, скільки пожертв зробив, які аукціони підтримав. У ньому реалізовано історію активних ставок, перегляд виграних лотів із можливістю оплати або підтвердження отримання, а також можливість змінити особисті дані. Такий блок не лише інформує, а й мотивує: коли користувач бачить результати своїх дій, у нього виникає емоційний зв’язок із платформою.

Організатори, зі свого боку, повинні мати розділ для створення і редагування лотів. У формі заповнення передбачено всі необхідні поля: назва, опис, стартова ціна, крок ставки, дата завершення, категорія. Фото лоту можна завантажити безпосередньо з комп'ютера. Також організатор повинен мати змогу відслідковувати кількість ставок, бачити лідерів торгів і вручну завершувати аукціон за потреби. Така автономність дозволяє волонтерам оперативно реагувати на ситуації без залучення адміністраторів.

Адміністративна панель – це інструмент контролю. Вона включає перегляд нових лотів для затвердження, контроль за поведінкою користувачів, блокування у разі порушень, аналітичні графіки щодо кількості ставок, зібраних коштів, активності в системі. Панель також має лог подій – для відстеження хто і коли змінював або видаляв інформацію. Усе це необхідно для забезпечення прозорості та довіри до системи.

Окрім функціональних вимог, необхідно чітко визначити нефункціональні – тобто ті, що не стосуються "що робити", а відповідають на питання "як робити". Це – продуктивність, доступність, UX-досвід, безпека, стабільність, масштабованість. Кожен із цих параметрів безпосередньо впливає на успішність системи. Інтерфейс має завантажуватись швидко, працювати плавно, бути інтуїтивно зрозумілим. Навігація повинна бути мінімалістичною – у кілька кліків до основних функцій, а елементи – логічно згруповані. Для користувачів з особливими потребами слід передбачити навігацію з клавіатури, можливість масштабування тексту, альтернативні описи для зображень.

Таблиця 1.2 – Функціональні та нефункціональні вимоги до front-end

Функціональні вимоги	**Нефункціональні вимоги**
Реєстрація, вхід у систему, розмежування ролей (учасник / організатор / адміністратор).	Продуктивність: швидке завантаження UI, асинхронна логіка, кешування даних.
Перегляд лотів: стрічка з плитками, фільтри, пошук, сортування.	UX: інтуїтивна структура, мінімум кліків, мікроанімації, підказки.
Перехід до деталізованої сторінки лоту з таймером і ставкою.	Доступність: масштаб шрифтів, навігація з клавіатури, контрастність.
Участь в торгах: введення ставки, перевірка, підтвердження.	Безпека: HTTPS, JWT, захист від XSS / CSRF / SQL-ін’єкцій, не зберігати токени локально.
Особистий кабінет: історія ставок, виграні лоти, редагування профілю.	Стабільність: офлайн-повідомлення, збереження чернеток, відновлення після збоїв.
Створення і редагування лотів (для організаторів).	Масштабованість: компонентний інтерфейс, легке додавання нових функцій.
Панель адміністратора: модерація, керування користувачами, перегляд статистики.	Адаптивність: підтримка різних розмірів вікон, темний/світлий режим.

Безпека – окремий аспект. Усі дії повинні виконуватись через захищене з’єднання. Персональні дані зберігаються у зашифрованому вигляді, а доступ до особистої інформації обмежується сесією з JWT-токеном [9]. Аби уникнути втрати введених даних, система повинна підтримувати збереження чернеток і відновлення сесії після збою мережі. Якщо інтернет пропав – користувач бачить повідомлення, а всі дії відновлюються автоматично після з’єднання.

Усе це вимагає масштабованої архітектури – як у коді, так і в інтерфейсі. Компонентний підхід дозволяє додавати нові функції – чат підтримки, бонусні

програми, віджети подяк – без необхідності змінювати основу. Інтерфейс має бути адаптивним і відповідати майбутнім змінам. Саме така система вимог – чітко сформульованих, логічно пов’язаних і орієнтованих на користувача – забезпечує успіх розробки платформи благодійних аукціонів.

Надзвичайно важливим аспектом у проєктуванні інтерфейсу є забезпечення продуктивності та швидкодії. У контексті аукціонної системи навіть незначна затримка у завантаженні сторінки або реакції інтерфейсу може вплинути на рішення користувача зробити ставку. Наприклад, при завершенні торгів, коли кожна секунда критична, користувач очікує миттєвого оновлення інформації. Тому завантаження ключових елементів інтерфейсу має здійснюватися асинхронно з використанням локального кешування. Такий підхід дозволяє зменшити навантаження на сервер і забезпечити швидке реагування системи на дії користувача.

Щоб підтримувати надійність і стійкість програми, архітектура інтерфейсної частини має враховувати сценарії відновлення після збоїв. У разі втрати інтернет-з’єднання користувач має отримати відповідне повідомлення, а дані, введені до того моменту, повинні зберігатись у локальній пам’яті. Після відновлення з’єднання система повинна автоматично оновлювати сесію і пропонувати відновлення дій. Такі механізми дозволяють уникнути фрустрації користувача і зменшити ризики втрати цінної інформації, особливо у випадках, коли користувач заповнював форму створення лоту або готував ставку на аукціон.

Не менш важливою вимогою до front-end є підтримка тестованості логіки інтерфейсу. Шаблон проектування MVVM, що використовується у WPF та Avalonia, дає змогу реалізувати структуру додатку таким чином, щоб більшість логіки була зосереджена у ViewModel [10]. Це дозволяє створювати unit-тести для перевірки функціональності без необхідності запуску повного інтерфейсу. Завдяки цьому розробники можуть швидко перевіряти правильність обробки подій, валідації даних і взаємодії з API. Такий підхід підвищує якість розробки та прискорює пошук і виправлення помилок.

Останнім, але не менш важливим чинником є підтримка подальшого розширення системи. Архітектура інтерфейсу повинна бути спроєктована з урахуванням можливості додавання нових розділів, змін у ролях користувачів або підключення зовнішніх сервісів (наприклад, платіжних систем, сервісів верифікації або інтеграції з відкритими API фондів). Компонентний підхід до побудови інтерфейсу дає змогу додавати нові модулі без переписування існуючого коду. У межах благодійної платформи це створює простір для розвитку – наприклад, запуск тематичних кампаній, партнерських зборів або інтерактивних подій. Такий рівень гнучкості дозволяє зберегти актуальність продукту на тривалу перспективу.

1.3 Взаємодія клієнтської та серверної частини системи

У сучасних програмних системах, особливо тих, що реалізують функціональність через мережу, чітке розмежування між клієнтською та серверною частинами є ключовим архітектурним принципом. Це дозволяє досягти високої гнучкості, масштабованості та безпеки при взаємодії користувача із даними та логікою системи.

У контексті створення десктоп-додатку для благодійних аукціонів, таке розділення не лише дозволяє реалізувати незалежну логіку інтерфейсу та обробки даних, а й дає можливість паралельної роботи двох розробників над окремими частинами проєкту: один займається UI, інший – серверною логікою та базами даних.

Однією з основних переваг клієнт-серверної архітектури є чіткий розподіл відповідальностей: клієнтська частина відповідає за зручну візуалізацію, інтерактивність та взаємодію з користувачем, тоді як серверна частина – за обробку запитів, збереження даних та забезпечення бізнес–логіки. Такий підхід не лише підвищує продуктивність розробки, а й спрощує масштабування системи в майбутньому.

Для забезпечення ефективної взаємодії між клієнтською та серверною частинами системи необхідно ретельно підібрати стек технологій, який би одночасно відповідав вимогам функціональності, продуктивності, безпеки та простоти масштабування. У межах даного проєкту було обрано сучасні, перевірені інструменти, які активно використовуються в індустрії розробки програмного забезпечення.

Загальна технічна структура проєкту включає такі компоненти:

- Клієнтська частина (Front-end) – десктоп-додатоку на основі WPF або Avalonia, що дозволяє створювати нативні інтерфейси з високим рівнем кастомізації. Для кросплатформеності може бути використано .NET MAUI[11].
- Серверна частина (Front-end) – створена за допомогою ASP.NET Core Web API, що забезпечує сучасну, швидку та безпечну платформу для побудови RESTful сервісів.
- База даних – використовується PostgreSQL або SQL Server, із підключенням через Entity Framework Core, що дозволяє працювати з даними на рівні об’єктів.
- Авторизація – реалізована за допомогою JWT для захисту API та OAuth2 для соціальної автентифікації.
- Хостинг API – передбачається на Azure, Heroku або VPS з Docker, що забезпечує гнучке розгортання та просте масштабування.

Клієнт–серверна архітектура – це один із найпоширеніших підходів до побудови інформаційних систем, який передбачає розподіл компонентів на дві основні частини: клієнт (користувач) і сервер (логіка + база даних). Такий підхід дозволяє створювати розподілені системи, які легко підтримувати, оновлювати і масштабувати [12].

Клієнт – це програма, яка встановлюється на пристрій користувача. Клієнт надає графічний інтерфейс, відображає дані, взаємодіє з користувачем і надсилає запити на сервер.

Сервер – це окрема програма або сервіс, який знаходиться на віддаленому хості, обробляє запити клієнтів, виконує бізнес–логіку та звертається до бази даних.

База даних (БД) – центральне сховище інформації про лоти, користувачів, ставки, історію, донати тощо. Сервер взаємодіє з базою за допомогою ORM (Object–Relational Mapping), у нашому випадку – Entity Framework Core.

У рамках проєкту використовується саме така архітектура, що дозволяє легко змінювати логіку клієнта без змін на сервері, централізовано оновлювати логіку API без необхідності змін у додатку, забезпечити захист даних через централізовану обробку та авторизацію, масштабувати систему на хмарних платформах.

Більшість сучасних клієнт-серверних додатків побудовані на основі багаторівневої (мультирівневої) архітектури, яка передбачає чіткий поділ за функціональними блоками. У нашому випадку виділяються три основні рівні:

1. Рівень представлення (Presentation Layer)

Це інтерфейс, з яким працює користувач – десктоп-додаток, створений у WPF або Avalonia. Він відповідає за відображення даних, навігацію, збір введеної інформації та передачу її на сервер. Інтерфейс повинен бути простим, інтуїтивним, логічним і зрозумілим [13].

2. Логічний рівень (Business Logic Layer)

Цей рівень реалізований у вигляді API (на ASP.NET Core). Він обробляє всі запити: перевіряє ставки, авторизує користувача, взаємодіє з базою, обраховує переможця аукціону, надсилає повідомлення. Уся логіка функціонування платформи знаходиться саме тут.

3. Рівень даних (Data Access Layer)

Реалізується за допомогою PostgreSQL або SQL Server. Entity Framework Core дозволяє працювати з БД не напряму, а через об’єкти, що спрощує код і підвищує безпеку. Саме на цьому рівні зберігається історія аукціонів, ставки, користувачі, їхні профілі тощо.

Такий підхід дозволяє ізолювати зміни: наприклад, якщо потрібно змінити структуру даних – це не вплине на зовнішній вигляд додатку; якщо змінюється логіка аукціону – інтерфейс залишається незмінним.

Алгоритм взаємодії між клієнтом і сервером:

1. Користувач запускає десктоп-додаток і бачить головну сторінку з доступними лотами. У цей момент клієнт надсилає запит до API для отримання списку актуальних аукціонів.
2. API повертає дані у форматі JSON, які обробляються на стороні клієнта і виводяться у вигляді карток лотів з таймерами, фото, стартовою ціною.
3. Якщо користувач хоче взяти участь у торгах – він повинен авторизуватись. При вході дані надсилаються на API, де перевіряються, і у відповідь клієнт отримує JWT-токен, який зберігається у пам’яті додатку.
4. Усі наступні дії (наприклад, ставки, створення лотів, перегляд історії) надсилаються до серверу з доданим токеном у заголовку. Сервер перевіряє дійсність токена і виконує відповідні операції.
5. Коли користувач виграє аукціон, система повідомляє його через повідомлення в інтерфейсі або e-mail, а також оновлює базу даних. Адміністратор може перевірити інформацію через панель модерації.

Такий цикл взаємодії повторюється кожного разу, коли користувач працює із системою – ілюструючи постійний діалог між UI та API.

Таблиця 1.3 – Переваги та особливості реалізації десктоп-додатку

Категорія	Опис
Переваги	• Краща інтеграція з операційною системою • Можливість локального кешування даних (лоти, аватар тощо) • Плавна і чутлива графіка • Не потребує браузера
Особливості реалізації	• Заборона зберігання чутливих даних на диску (тільки в пам’яті) • Повідомлення про втрату з’єднання та часткова офлайн–функціональність • Асинхронність усіх запитів до API

Для зв’язку з сервером використовується **HttpClient** (або аналогічні механізми), які формують запити, обробляють відповіді, валідують дані.

Взаємодія клієнтської та серверної частин у межах створення благодійного десктоп-додатку є фундаментально важливою. Вона забезпечує не лише ефективну обробку даних і безпеку, а й формує основу довіри користувачів до системи. Обрані технології та архітектурні рішення дозволяють створити продукт, який одночасно є сучасним, надійним, зручним та готовим до розвитку в майбутньому.

РОЗДІЛ 2
ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ FRONT-END ЧАСТИНИ ДЕКСТОП-ДОДАТКУ

2.1 Огляд технологій та інструментів для створення десктоп-додатків

Сучасна розробка інтерфейсної частини програмного забезпечення для систем вимагає не лише знання синтаксису мов програмування, але й глибокого розуміння того, які інструменти, фреймворки та технології будуть найбільш ефективними для реалізації конкретного функціоналу. При створенні фронтенду застосунку для благодійних аукціонів критичним є вибір рішень, що забезпечують не лише швидкодію, але й зручність використання, естетичність інтерфейсу та кросплатформену сумісність.

Перед тим як перейти до реалізації графічного інтерфейсу, було проведено порівняльний аналіз технологій, що використовуються для побудови UI в середовищі .NET. Розглядалися фреймворки Windows Presentation Foundation (WPF), Avalonia UI, .NET MAUI, а також частково WinForms. Оцінювання проводилося за критеріями стабільності, кросплатформеності, підтримки шаблону MVVM, доступності документації, активності спільноти та зручності інтеграції з API. Такий аналіз дозволив визначити найдоцільніші інструменти для реалізації інтерфейсної частини проєкту.

Згідно з аналітичним матеріалом «Desktop App Development Guide 2025» (JhavTech Studios, 2024), фреймворки WPF, Avalonia та .NET MAUI є ключовими рішеннями для розробки десктопних інтерфейсів у найближчі роки [14]. У публікації зазначено, що WPF зберігає лідерство у корпоративному сегменті Windows -додатків завдяки стабільності та зрілості, тоді як Avalonia стрімко набирає популярності завдяки своїй кросплатформенності та схожості з WPF у підході до архітектури та стилізації. .NET MAUI визначається як перспективна, але ще не повністю стабільна платформа, яка більше орієнтована на мобільно–десктопний гібрид, ніж на класичні настільні застосунки.

Одним із ключових факторів при виборі технології для створення інтерфейсу є простота освоєння та швидкість розробки. Особливо це важливо в умовах обмеженого часу, коли кожен етап проєкту має бути ефективним і прогнозованим. Якщо технологія добре документована, має знайому структуру та підтримує зрозумілий шаблон роботи – це суттєво зменшує кількість помилок, пришвидшує реалізацію інтерфейсу та дозволяє більше часу приділити логіці взаємодії з сервером або дизайну. Саме тому WPF і Avalonia, які використовують схожий підхід до розмітки через XAML, є практичним вибором.

Ще один важливий критерій – стабільність і підтримка обраної технології на ринку. У динамічному середовищі IT важливо працювати з тими фреймворками, які постійно оновлюються, мають активну спільноту і не втратили актуальності. WPF залишається стабільним інструментом для Windows - розробки, але Avalonia все більше привертає увагу розробників, оскільки дозволяє створювати інтерфейси одразу для кількох платформ. Цей фреймворк активно оновлюється, має відкритий код і підтримується спільнотою, що є перевагою у довгостроковій перспективі.

Не менш важливим є те, як інтерфейсна частина буде взаємодіяти з сервером. Якщо використовується REST API, потрібно, щоб технологія підтримувала просту роботу з HTTP–запитами, асинхронну обробку відповідей і зручне парсування JSON–даних. У фреймворках WPF і Avalonia такі можливості реалізуються через стандартні засоби платформи .NET, зокрема через HttpClient і підтримку async/await. Це дозволяє забезпечити плавну роботу користувацького інтерфейсу навіть у разі повільного інтернет–з’єднання або помилок на стороні сервера.

Також важливо враховувати гнучкість дизайну та можливість створення привабливого інтерфейсу. У проєкті, пов’язаному з благодійністю, велике значення має перше враження, яке отримує користувач. Технологія повинна дозволяти легко налаштовувати візуальні елементи, застосовувати теми, анімації, адаптувати інтерфейс під різні розміри вікна. Це сприяє емоційному зв’язку з користувачем і стимулює участь в аукціонах.

У сучасному середовищі .NET–розробки існує кілька платформних підходів до створення десктопних застосунків. Найбільш актуальними є фреймворки WPF, Avalonia UI, .NET MAUI, а також частково WinForms та Electron. Кожен із них має свої переваги, обмеження та сценарії ефективного використання.

Windows Presentation Foundation (WPF) – це зріла і стабільна технологія від Microsoft, яка вже понад 15 років використовується для розробки UI у корпоративних Windows–додатках. Вона побудована на XAML–розмітці, підтримує шаблон MVVM і має розвинену систему стилів та анімацій. Незважаючи на свою прив'язаність до Windows, WPF залишається актуальним для проєктів, орієнтованих на локальні десктопні рішення в корпоративному середовищі.

Avalonia UI – це open–source фреймворк, що підтримує створення повноцінних XAML–додатків для Windows, macOS і Linux. Він зберігає логіку WPF, але розроблений для кросплатформеності, з можливістю створення нативних інтерфейсів без участі браузера. Avalonia активно розвивається з 2018 року та вже використовується в таких проєктах, як GitHub Desktop, WalletWasabi, OpenTabletDriver [15].

.NET MAUI (Multi–platform App UI) – це сучасне універсальне рішення від Microsoft, офіційно представлене у 2022 році. Воно базується на Xamarin і дає змогу створювати інтерфейси одразу для Windows, macOS, Android та iOS. Незважаючи на підтримку Microsoft, MAUI поки ще не досяг стабільності WPF або Avalonia у десктопному сегменті. MAUI все ще має обмежену підтримку нативних функцій десктопів, особливо у кастомізації вікон та адаптації до великих екранів [16].

WinForms вважається застарілою технологією, хоча й досі використовується в невеликих внутрішніх корпоративних рішеннях [17]. Вона не підтримує шаблон MVVM і має обмежену гнучкість дизайну. З іншого боку, Electron дозволяє створювати десктопні додатки на базі web – технологій (HTML, CSS, JS), проте є надмірно важким для систем із низькою

продуктивністю та не інтегрується з .NET API так ефективно, як рідні фреймворки.

Для обґрунтованого вибору було сформовано порівняльну таблицю, що демонструє відмінності між ключовими фреймворками:

Таблиця 2.1 –Порівняння фреймворків для розробки десктоп–додатків

Параметр	WPF	Avalonia UI	.NET MAUI
Платформи	Windows	Windows, Linux, macOS	Windows, macOS, Android, iOS
Підтримка XAML	Повна	Повна	Часткова
MVVM	Так	Так	Так, з обмеженнями
Документація	Розширена, офіційна	Ком'юніті, GitHub	Ведеться, ще не повна
Анімації, стилі	Так	Часткова підтримка	Так, обмежено
Стабільність	Висока	Середня	Низька / у процесі стабілізації
Навчальна крива	Низька	Низька (для WPF–розробників)	Середня
Підтримка Microsoft	Так	Ні	Так
Кросплатформеність	Ні	Так	Так

Як видно, для розробки повноцінного UI в умовах стабільної Windows – середовища WPF залишається оптимальним рішенням. Avalonia виграє в умовах потреби кросплатформенності. MAUI привабливий у перспективі, однак поки що не досяг зрілості для великих застосунків.

Велика частина десктопних застосунків у корпоративному середовищі .NET будуються саме на основі WPF. Це підтверджує лідерство фреймворку у фінансовій, державній та аналітичній галузях, де ключову роль відіграє стабільність і точна робота з системними ресурсами. У свою чергу Avalonia обирають розробники, яким потрібна підтримка macOS або Linux, зокрема у стартапах, open – source платформах та громадських ініціативах.

У межах дипломного проєкту було обрано поєднання WPF (як основного інтерфейсу під Windows). Це забезпечує:

- швидку реалізацію стабільного UI для MVP;
- можливість подальшого переходу на кросплатформенну структуру без значної зміни архітектури;
- використання спільної логіки через шаблон MVVM.

Для забезпечення ефективної розробки інтерфейсної частини десктопного застосунку були використані сучасні інструменти, що покривають повний цикл UI – розробки – від дизайну до тестування інтеграції з API. Їх правильне застосування дозволяє значно пришвидшити роботу, забезпечити послідовність у написанні коду, поліпшити командну взаємодію та гарантувати якість кінцевого продукту.

Visual Studio – головне інтегроване середовище розробки (IDE), яке використовується для написання, компіляції, налагодження та тестування front-end коду. Платформа підтримує WPF, Avalonia, MAUI, а також має повну інтеграцію з NuGet, системами контролю версій, багатьма розширеннями для XAML, C# та .NET. Вона також дозволяє створювати власні шаблони проектів, профілі збірки, аналізувати продуктивність додатку та профілювати ресурси. Саме Visual Studio забезпечує найбільш зручне середовище для роботи з MVVM–архітектурою, шаблонами сторінок та прив’язками даних [18].

Figma – веб – додаток для створення макетів інтерфейсу, який дозволяє розробляти дизайн кожного вікна застосунку до початку його програмної реалізації. Figma використовується для побудови wireframes, UI - компонентів, логічних сценаріїв переходу між екранами. Особливо цінним є можливість командної роботи в реальному часі та спільного перегляду макетів. Крім того, дизайн із Figma можна легко адаптувати до XAML-структури, оскільки він базується на блоковому підході до верстки [19].

XAML Styler –це розширення для автоматичного форматування XAML-коду відповідно до заданих правил і стилістичних стандартів. Його використання забезпечує уніфікацію структури коду, підвищує читабельність та зменшує

ризик помилок під час мануального редагування. Інструмент дозволяє впорядковувати атрибути, структуру вкладень і позиціонування елементів, що особливо актуально при роботі з шаблонами і великими файлами інтерфейсу [20].

Git + GitHub – система контролю версій і хмарна платформа для зберігання, відстеження та колективної розробки проєкту. Git забезпечує контроль змін у коді, можливість створення паралельних гілок для тестування нових функцій, а GitHub – надає можливість онлайн–співпраці, ведення історії змін, створення Pull Requests, рев'ю коду та автоматичної інтеграції з іншими інструментами. Це важливо в умовах командної розробки або підготовки до розширення проєкту іншими фахівцями [21].

Правильний вибір технологій для реалізації інтерфейсної частини десктопного-додатку відіграє ключову роль у забезпеченні стабільності, продуктивності та зручності використання системи. Крім того, застосування сучасних інструментів розробки забезпечує гнучкість, швидку інтеграцію з API та високий рівень візуальної якості інтерфейсу.

2.2 Архітектура front-end частини декстоп-додатку

У сучасній розробці програмного забезпечення архітектура інтерфейсної частини (front-end) виступає одним із фундаментальних елементів, що визначає не лише зовнішній вигляд додатку, а й його функціональність, гнучкість та можливості масштабування. Архітектура front-end охоплює структуру програмних компонентів, принципи їх взаємодії, розподіл відповідальностей між модулями та логіку обробки даних. Від того, наскільки грамотно спроєктована ця структура, залежить зручність розробки, тестування, розширення і підтримки застосунку в подальшому.

При побудові десктоп - додатку важливо особливо уважно продумувати архітектуру ще до початку безпосередньої розробки. Це дозволяє уникнути безладу в коді, дублювання функціоналу, ускладнень при додаванні нових можливостей або виправленні помилок. Продумана front-end архітектура

забезпечує ясність у розподілі завдань між розробниками та дозволяє ефективно управляти масштабуванням системи.

Варто також розуміти, що архітектура десктопного-додатку відрізняється від архітектури веб додатку за кількома важливими параметрами. По-перше, десктоп-застосунки зазвичай мають більше можливостей доступу до ресурсів пристрою та можуть працювати офлайн. По-друге, взаємодія між інтерфейсом і бізнес-логікою в десктопних рішеннях здійснюється без залучення браузерного середовища, що дозволяє будувати більш складні сценарії і використовувати нативні елементи керування. По-третє, десктопна архітектура, як правило, вимагає кращої оптимізації продуктивності через обмеження заліза користувача.

У разі відсутності чіткого розмежування між UI, логікою і даними виникають такі проблеми: дублювання коду в інтерфейсі, складність внесення змін без порушення інших частин системи, зростання залежностей і ризик появи помилок при оновленнях. Це призводить до швидкого зниження якості продукту і ускладнює масштабування.

Таким чином, побудова правильної, багаторівневої та масштабованої front-end архітектури є критично важливим етапом розробки десктопного-додатку, особливо у проєктах, орієнтованих на взаємодію користувачів через реальні фінансові операції, як-от у випадку благодійних аукціонів.

У межах розробки десктопного-додатку було обрано трирівневу (three-tier) архітектуру інтерфейсної частини, яка базується на чіткому розподілі системи на три основні рівні: Presentation (UI), Logic (ViewModel) і Data (API/DTO). Такий підхід дозволяє досягти високої модульності додатку, спрощення супроводу, підвищення надійності системи та полегшення подальшого масштабування або модифікацій.

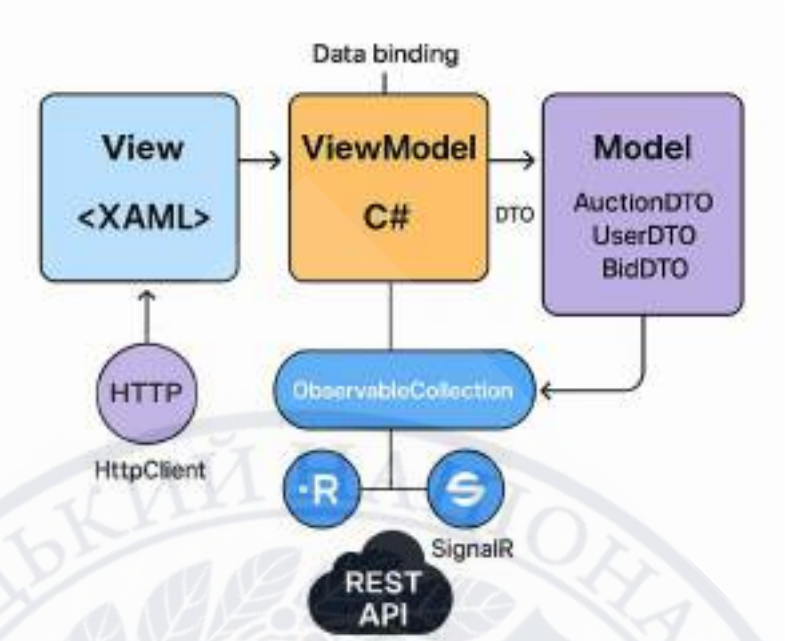


Рис 2.1 – Схема архітектори front-end додатку

Трирівнева архітектура дозволяє чітко розділити завдання кожного компонента. Кожен рівень взаємодіє лише з безпосередньо суміжним шаром, що мінімізує залежності та забезпечує принцип «розділення відповідальностей» (Separation of Concerns). Це дає змогу ізолювати бізнес-логіку від деталей реалізації інтерфейсу або способу доступу до даних, а також спрощує написання модульних тестів для окремих компонентів.

Presentation Layer відповідає за безпосередню взаємодію з користувачем: відображення даних, прийом введення, ініціацію подій та забезпечення зворотного зв'язку. У середовищі WPF реалізація UI здійснюється через мову розмітки XAML (eXtensible Application Markup Language). Важливо, що у WPF/XAML інтерфейс і логіка розділені, що дозволяє дизайнерам працювати над візуальною частиною окремо від розробників бізнес-логіки.

Інтерфейс будується у вигляді ієрархії вікон, сторінок і контролів: кнопок, текстових полів, списків, панелей розташування тощо. Кожен елемент має властивості, які дозволяють йому прив'язуватись до даних за допомогою механізму Binding. Це означає, що зміни даних у ViewModel автоматично оновлюють інтерфейс, без необхідності писати зайвий код вручну.

Приклад:

Кнопка «Зробити ставку» у вікні перегляду лоту має прив'язку до команди у ViewModel. Натискання користувачем на цю кнопку викликає команду, що обробляє логіку перевірки та надсилання ставки на сервер, без прямого втручання у UI.

Таке використання прив'язок спрощує підтримку UI, підвищує читабельність коду та знижує ймовірність помилок при оновленні даних.

ViewModel Layer є центральним рівнем логіки програми. Він виступає посередником між Presentation Layer та Data Layer, координуючи обмін інформацією між інтерфейсом користувача та джерелами даних.

У ViewModel реалізується обробка подій, виконання команд, валідація введених даних, керування станом екранів та логіка навігації. Класична реалізація ViewModel у WPF базується на двох ключових інтерфейсах:

- INotifyPropertyChanged – забезпечує автоматичне оновлення прив’язаних властивостей у інтерфейсі при їх зміні у ViewModel.
- ICommand – описує дії, що можуть бути викликані користувачем через елементи управління (наприклад, натискання кнопок).

ViewModel ізольована від View і не знає, як виглядає інтерфейс користувача – вона працює лише з даними. Завдяки цьому код стає більш тестованим та легко адаптується при зміні вимог до UI.

Приклад:

ViewModel аукціону містить властивість CurrentBid (поточна ставка) та команду PlaceBidCommand. Коли користувач вводить нову ставку і натискає кнопку, PlaceBidCommand перевіряє правильність введених даних і надсилає запит до серверу.

Data Layer відповідає за роботу з зовнішніми сервісами або базами даних. У контексті дипломного проєкту реалізація цього рівня здійснюється через використання HttpClient для виконання REST-запитів до серверної частини, розробленої на ASP.NET Core Web API.

Кожен запит ViewModel до API виконується через окремий сервіс-клас (наприклад, AuctionService або LotService), який відповідає за відправку запиту, обробку відповіді, перехоплення можливих помилок (обробка коду помилок HTTP) та трансформацію отриманих JSON-даних у об'єкти DTO.

DTO (Data Transfer Object) – це спеціалізовані класи, які служать для обміну даними між клієнтом і сервером. Вони містять лише дані без бізнес-логіки.

DTO забезпечують зручність роботи з даними, стандартизують обмін інформацією і мінімізують залежність клієнтської логіки від змін на сервері.

Приклад:

Клас LotDto може мати такі властивості:

- Id (унікальний ідентифікатор лоту),
- Title (назва лоту),
- Description (опис лоту),
- CurrentBid (поточна ставка),
- EndTime (час завершення аукціону).

При отриманні даних через API відповідь парситься у об'єкт LotDto, який потім використовується ViewModel для оновлення відображення інформації у UI.

При розробці десктопних застосунків на базі WPF особливу роль відіграє архітектурний шаблон MVVM (Model-View-ViewModel). Цей підхід було спеціально розроблено для XAML-базованих технологій, таких як WPF, Avalonia або Xamarin, і він дозволяє логічно розділити структуру додатку на три основні компоненти: Model, View та ViewModel.

View (представлення) відповідає лише за візуалізацію даних і реагування на дії користувача. Вона не містить бізнес-логіки і не взаємодіє напряму із джерелами даних. У середовищі WPF View реалізується за допомогою XAML-файлів, де кожен елемент інтерфейсу (кнопки, текстові поля, списки) прив'язується до відповідних властивостей або команд ViewModel через механізм Binding.

Model – це рівень даних та бізнес-логіки. Він представляє сутності предметної області, наприклад, модель «Лот аукціону», а також може включати DTO для обміну даними з API. Модель не має прямого впливу на інтерфейс або ViewModel.

- View – те, що бачить і натискає користувач.
- ViewModel – те, що обробляє дії користувача і управляє даними на екрані.
- Model – реальні дані, які ми відображаємо і обробляємо.

Основна перевага MVVM полягає в тому, що розробник може окремо працювати над логікою додатку (у ViewModel) і окремо над інтерфейсом (у View). Це суттєво спрощує підтримку і тестування програмного забезпечення. Наприклад, можна протестувати ViewModel за допомогою юніт-тестів без потреби запускати інтерфейс користувача.

Крім того, MVVM ідеально підходить для роботи з асинхронними сценаріями. У випадку інтеграції з API або базами даних ViewModel виконує асинхронні запити (async/await) і автоматично оновлює відображення інформації у View через механізм прив'язки.

Типовий сценарій взаємодії виглядає наступним чином:

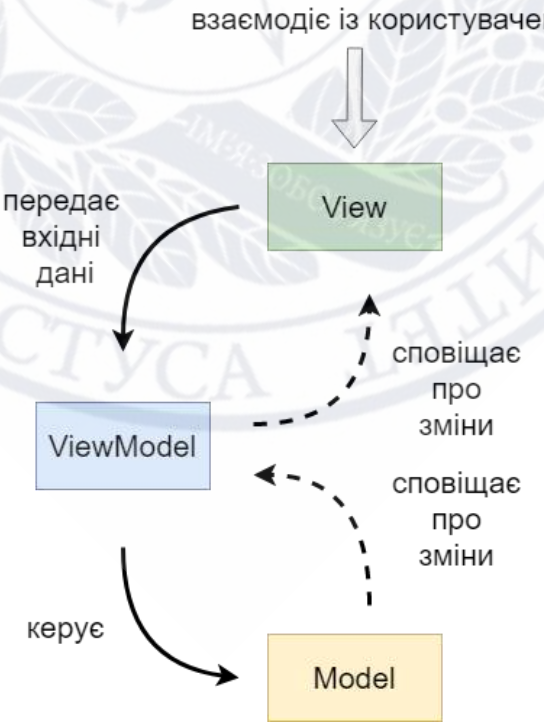


Рисунок 2.2 - Діаграма взаємодії між компонентами шаблону MVVM

1. Користувач натискає кнопку у інтерфейсі View.
2. View за допомогою прив'язки (Command) передає подію у відповідну команду у ViewModel.
3. ViewModel обробляє подію:
    - Виконує валідацію введених даних;
    - Звертається до API через сервіс для отримання або зміни інформації;
    - Оновлює властивості, прив'язані до елементів інтерфейсу.
4. View автоматично оновлюється завдяки механізму повідомлень про зміну властивостей (INotifyPropertyChanged).
5. Model служить джерелом істинних даних: локальних або отриманих із серверу.

Такий механізм дозволяє уникнути надмірних залежностей, полегшує тестування ViewModel без запуску UI та забезпечує чисту структуру коду.

Використання трирівневої архітектури разом із шаблоном MVVM дозволяє створити надійний, масштабований і легко підтримуваний десктопний застосунок.

Це особливо важливо для проєктів, пов'язаних із фінансовими операціями та великою кількістю інтерактивних користувачів, як у випадку благодійних аукціонів.

2.3 Проектування та розробка прототипу інтерфейсу

Перед початком реалізації десктопного-додатку було прийнято рішення створити повноцінний прототип інтерфейсу програми. Це дозволило заздалегідь визначити ключові вікна застосунку, перевірити логіку переходів між ними та візуалізувати загальну структуру взаємодії. На цьому етапі також узгоджувалися вимоги до зручності, доступності та послідовності інтерфейсу.

Прототип слугує основою для подальшої реалізації дизайну в середовищі розробки – зокрема, в інструментах WPF. На його базі розробляється кожне вікно додатку, відповідно до передбачених сценаріїв використання: від реєстрації

користувача до участі в аукціоні та перегляду історії ставок. Такий підхід дозволяє працювати швидше й уникати неузгоджених змін на пізніших етапах розробки.

Крім того, розробка прототипу дає змогу сфокусуватися на користувацькому досвіді ще до початку програмування: оптимізувати кількість кліків, уникнути перевантаженості інтерфейсу, правильно розставити акценти та візуальні пріоритети. Це особливо важливо для благодійного продукту, де простота й емоційна довіра – критичні чинники залучення користувача.

Проектування інтерфейсу користувача базується на дотриманні сучасних принципів UX (User Experience) та UI (User Interface) дизайну, які спрямовані на забезпечення інтуїтивності, доступності та емоційної привабливості додатку. Враховуючи специфіку благодійної платформи, основний акцент буде зроблено на простоту використання, логічність навігації та створення атмосфери довіри [22].

Одним із базових принципів стало прагнення до спрощення інтерфейсу. Кожне вікно додатку орієнтоване на виконання основної дії: реєстрація нового користувача, перегляд списку аукціонних лотів, перегляд деталей окремого лоту, участь у торгах, керування профілем. Це дозволяє уникнути перевантаженості інформацією, що, своєю чергою, підвищує ефективність взаємодії з системою.

Послідовність – ще один ключовий елемент при побудові прототипу. Розташування основних елементів інтерфейсу (кнопок підтвердження, меню переходу, полів введення) має залишитись однаковим у межах усіх сторінок додатку. Це формує у користувача очікуване і передбачуване середовище взаємодії, що скорочує час на освоєння нових функцій та підвищує загальну задоволеність використанням.

Важливу роль відіграє впровадження системного зворотного зв’язку. Кожна дія користувача супроводжується відповідним відгуком інтерфейсу – спливаючими повідомленнями про успішну реєстрацію, підтвердженням участі в торгах, сповіщенням про перебиту ставку або виграш. Такий підхід сприяє

створенню відчуття контролю над процесом і мінімізує ймовірність виникнення непорозумінь.

Особлива увага приділяється питанням візуальної доступності. Усі інтерфейсні елементи розробляються з урахуванням достатнього контрасту між текстом і фоном, великого розміру кнопок, зручності натискання. Це дозволяє забезпечити комфортну взаємодію для широкого кола користувачів.

Дизайн інтерфейсу орієнтується також на емоційне сприйняття. Використання теплих кольорових схем, округлих форм елементів керування, візуальних ефектів легких анімацій має на меті викликати у користувача позитивні емоції, почуття безпеки та підтримки, що особливо важливо у благодійних проєктах.

Усі екрани прототипу проектувалися таким чином, щоб мінімізувати кількість кліків для досягнення цільової дії. Наприклад, від моменту входу на головну сторінку до подання ставки передбачено не більше трьох взаємодій: вибір лоту → перегляд деталей → подання ставки. Це значно скорочує шлях користувача та підвищує ймовірність успішного завершення цільової взаємодії.

Важливим принципом стало також розміщення інформації за пріоритетністю: найважливіші елементи, як-от таймер завершення торгів чи сума поточної ставки, розміщуються на верхній частині екрану або мають найбільший візуальний акцент. Це допомагає швидко зорієнтуватися в інтерфейсі навіть при першому відвідуванні.

При розробці прототипу інтерфейсу для десктопного застосунку було важливо обрати такий інструмент, який дозволяв би швидко створювати інтерактивні макети, працювати з багатошаровими екранами, застосовувати адаптивні сітки, а також за потреби надавати макети для перегляду іншим учасникам команди чи керівнику проєкту.

Аналіз популярних середовищ для проектування інтерфейсів включав такі інструменти:

- Adobe XD – потужна програма для UX/UI дизайну, орієнтована на інтеграцію в екосистему Adobe.
- Figma – вебзастосунок для кросплатформеної розробки інтерфейсів із підтримкою спільної роботи в реальному часі.
- Sketch – професійний редактор інтерфейсів для macOS, який вимагає локальної установки [23].
- Axure RP – інструмент для розробки більш складних прототипів із логікою переходів та інтерактивними діями.

Після аналізу було прийнято рішення використовувати Figma як основний інструмент для проектування інтерфейсу [24]. Ключовими факторами вибору стали:

1. Доступність і простота початку роботи.

Figma є веб-застосунком, що не потребує складної інсталяції. Для роботи достатньо мати сучасний браузер і підключення до Інтернету, що дозволяє працювати навіть на базових пристроях. А також Figma має свій десктопний-додаток для зручного користування.

2. Можливість створення інтерактивних прототипів.

Figma дозволяє не лише малювати статичні макети, а й об'єднувати екрани логікою переходів, імітуючи реальні сценарії використання додатку. Це суттєво допомагає тестувати зручність навігації ще на етапі макетування.

3. Спільна робота та доступність для перевірки.

Усі макети Figma зберігаються у хмарі, і доступ до них можна надавати іншим користувачам за посиланням. Це особливо зручно для демонстрацій керівнику проєкту або консультанту.

4. Підтримка компонентів і стилів.

Figma дозволяє створювати шаблони кнопок, полів введення, заголовків тощо, які можна використовувати на різних екранах без дублювання роботи. Це забезпечує консистентність дизайну.

5. Адаптивність і використання сіток.

Можливість проектувати інтерфейси, які зберігають пропорційність при зміні розмірів вікна, що актуально для десктопних додатків із різними розмірами екранів.

Варто також зазначити, що Figma має безкоштовний тариф для невеликих команд, що робить її доступною для навчальних і некомерційних проєктів, таких як розробка дипломного застосунку.

Порівняно з альтернативами:

- Adobe XD потребує встановлення локального ПЗ і краще підходить для користувачів Adobe Creative Cloud.
- Sketch є чудовим рішенням для macOS, але обмежений у кросплатформеності.
- Axure RP більше орієнтований на розробку складних бізнес-прототипів, що робить його зайвим для завдань базового середовища благодійного аукціону.

Таким чином, Figma була обрана як оптимальне рішення для проектування макетів інтерфейсу завдяки її простоті, швидкості роботи, підтримці інтерактивності та можливості командної взаємодії [25].

Розробка прототипу здійснювалася поетапно, що дозволило системно підходити до створення інтерфейсу, зберігаючи баланс між функціональністю, логікою взаємодії та візуальною привабливістю.

Початковим етапом розробки стало чітке визначення функціональних і нефункціональних вимог до користувацького інтерфейсу, що мали відповідати реальним сценаріям взаємодії з платформою. Основу вимог склали ключові дії користувачів – від реєстрації до участі в аукціоні, перегляду історії ставок та управління особистими даними. Також були враховані загальні принципи зручності інтерфейсу: простота навігації, мінімізація кількості кліків до основної дії, адаптивність і приємна візуальна мова дизайну.

На наступному етапі було створено логічну карту взаємодії користувача з додатком – user flow. Для цього побудовано ієрархічну модель основних сторінок, що охоплюють повний функціонал системи. Структура взаємодії охоплює такі вузли:

Рисунок 2.3 – Структура прототипу

Кожна сторінка відповідає окремому сценарію і дозволяє уникати плутанини, що значно покращує UX у всій системі.

Наступним кроком було створення «чорнових» ескізів кожного вікна. На цьому етапі перевірялася логіка розміщення об’єктів, їх розміри, відстані між ними, читабельність структури. Ескізне проектування дало змогу швидко виявити слабкі місця в логіці інтерфейсу та оптимізувати структуру до створення повноцінного дизайну.

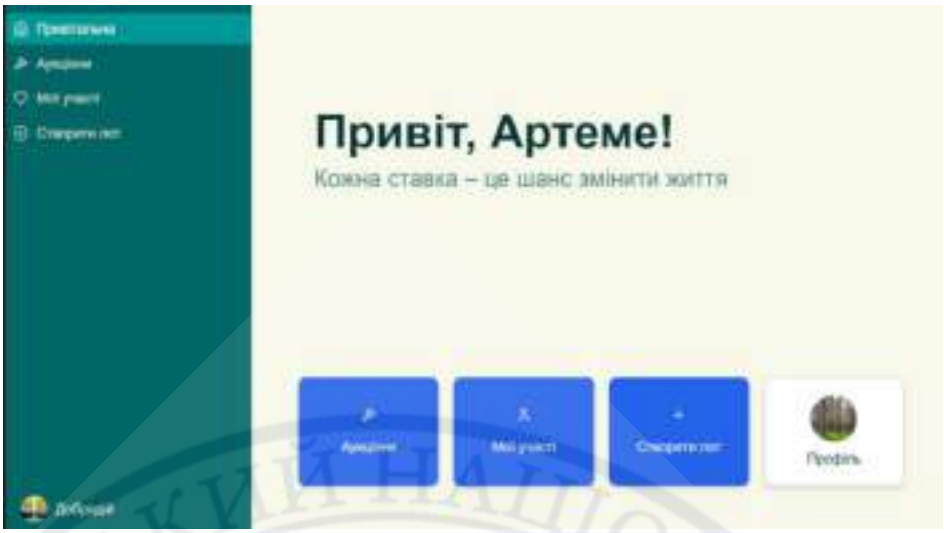


Рисунок 2.4 – Прототип головного екрану додатку

Після узгодження основної логіки переходів та розташування елементів розпочалася побудова макетів деталізації у Figma. На цьому етапі були застосовані основні кольори, шрифти, стилі кнопок та інші ключові елементи, що забезпечують зручність читання і навігації. Було створено такі базові екрани:

Рисунок 2.5 – Головна сторінка програми (привітальна)

Привітальна сторінка – це стартова точка, де користувач бачить персоналізоване звернення та швидкий доступ до ключових розділів: аукціонів, донату, перегляду профілю або створення лоту, перегляд історії активності.

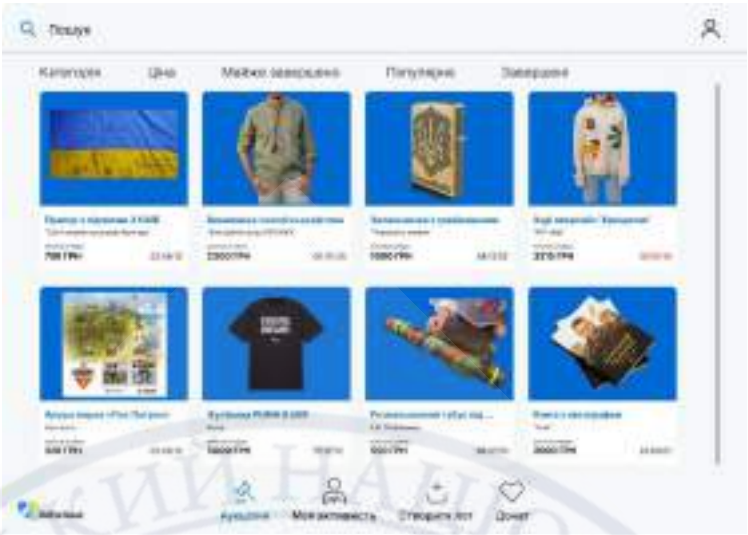


Рисунок 2.6 – Стрічка зі списком лотів

Каталог аукціонів – містить стрічку лотів у вигляді карток із фільтрами та пошуком, що забезпечує швидку навігацію по лотах.

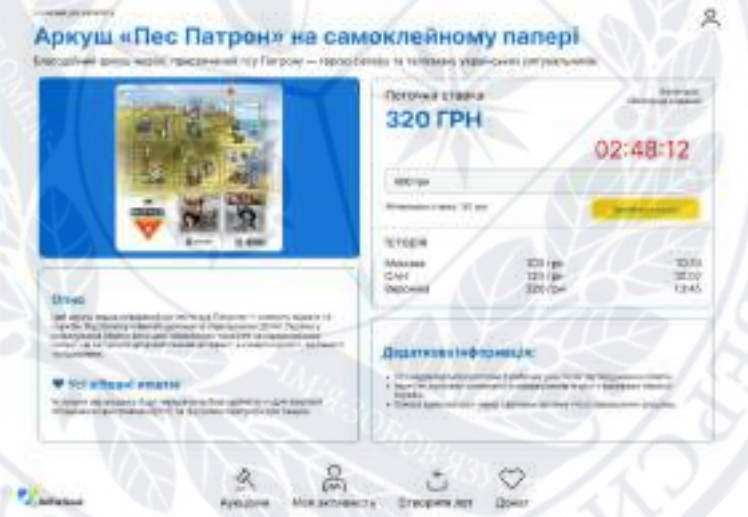


Рисунок 2.7 – Сторінка деталізації лоту

Детальна сторінка лоту – включає повноекранне зображення, таймер, поточну ціну, історію ставок та можливість зробити і підтвердити ставку.

Рисунок 2.8 – Особистий кабінет користувача

Профіль користувача – дає змогу переглядати персональні дані, редагувати профіль, виводити загальну статистику: кількість виграних лотів, суму пожертв.

Рисунок 2.9 – Сторінка «Мої активність» (для учасника)

Сторінка “Мої активність” – реалізує відображення активних ставок, виграних лотів, історії участі та прямих пожертв.

Далі макети були об’єднані за допомогою інтерактивних переходів (linking) у Figma для імітації реальної навігації додатком.

Це дозволило оцінити зручність переходів, логіку інтерфейсу і виявити можливі точки покращення до початку реальної розробки.

Після фінального тестування логіки були створені остаточні версії макетів, використано кольорові теми (основні кольори: світлий фон, контрастні акцентні кнопки), додано шрифтову ієрархію (заголовки, підзаголовки, текстові блоки), застосовано анімації наведених станів для кнопок і форм, підготовлено версії екранів для стандартних розмірів вікна.

Таким чином, прототип остаточно набув вигляду, близького до фінальної версії застосунку.

На фінальному етапі інтерактивний прототип був самостійно протестований з точки зору сценаріїв користувача. Після дрібних коригувань у структурі кнопок і текстів прототип був готовий до подальшої передачі в розробку – або до використання як референс при написанні коду WPF додатку.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ FRONT-END ЧАСТИНИ ДОДАТКУ

3.1 Створення базового шаблону десктоп-додатку

На початковому етапі реалізації фронтенд-частини десктопного додатку було закладено архітектурну структуру, яка слугує фундаментом для всіх наступних етапів розробки. Від якості цього шаблону залежить стабільність функціонування додатку, зручність підтримки, ефективність взаємодії з користувачем і можливість подальшого масштабування системи. Саме тому першим етапом розробки стало створення продуманого базового шаблону – як логічної, так і візуальної структури застосунку.

Для реалізації було обрано технологію Windows Presentation Foundation (WPF) – сучасний фреймворк від Microsoft, що дозволяє створювати гнучкі та функціональні десктопні застосунки для операційної системи Windows. Однією з ключових переваг WPF є підтримка мови XAML, яка дозволяє чітко розділяти візуальну частину (UI) та логіку взаємодії (backing logic). Це значно полегшує розробку, дозволяє зручно працювати в команді (UI/UX-дизайнер окремо, розробник окремо), і забезпечує гнучкість у стилізації та компонуванні інтерфейсу.

Перед початком реалізації інтерфейсу я створив макети всіх основних вікон у Figma, дотримуючись принципів сучасного UI/UX-дизайну та єдиного візуального стилю. Кольорова палітра була обрана не випадково: білий і світло-сірий – це нейтральні відтінки, які не перевантажують очі й забезпечують чистий, повітряний простір, а синій колір (#0069D2) виконує роль акценту. Саме синій часто асоціюється із надійністю, стабільністю та довірою – це якості, що напряму пов’язані з благодійною діяльністю. Мінімалістичний стиль інтерфейсу дозволяє користувачу сконцентруватися на головному – на діях, а не на декоративних елементах, і робить навігацію інтуїтивно зрозумілою.

Ще одним візуальним принципом, який активно застосовувався під час розробки, є ієрархія елементів [26]. Наприклад, усі заголовки мають більший розмір і жирність, ніж звичайний текст; активні кнопки мають яскравіший фон, а вторинні – світліший; важливі повідомлення виділяються іншим кольором або рамкою. Це допомагає користувачеві швидко орієнтуватися у змісті екрана, фокусувати увагу на ключових діях або інформації.

Шрифт Inter обраний не випадково – це сучасний, чіткий та нейтральний шрифт, який забезпечує гарну читабельність як у невеликих елементах (наприклад, підпис кнопок), так і в більших заголовках або описах. Він не відволікає увагу, але водночас надає інтерфейсу професійного вигляду.

Завдяки макетам у Figma я зміг заздалегідь продумати структуру вікон, розташування елементів, шрифти, відступи та логіку переходів, що значно пришвидшило реалізацію у WPF.

Проєкт створено на основі архітектурного патерну MVVM (Model-View-ViewModel). Це класичний підхід для WPF-додатків, який забезпечує чітке розділення відповідальностей між інтерфейсом (View), логікою взаємодії (ViewModel) та моделями даних (Model). Такий підхід підвищує модульність, дозволяє легко тестувати окремі частини логіки та спрощує масштабування проєкту в майбутньому. Наприклад, у ViewModel описується команда PlaceBidCommand, яка реагує на натискання кнопки «Зробити ставку», а в UI лише прив’язується до цієї команди через Binding.

Після ініціалізації WPF-проєкту було створено кілька основних файлів:

- **App.xaml** – файл конфігурації застосунку, який визначає стартове вікно, глобальні ресурси (якщо вони використовуються) та параметри запуску.
- **MainWindow.xaml** – головне вікно додатку, яке слугує основним маршрутизатором або меню для переходу до окремих розділів: перегляду лотів, створення нового лоту, перегляду профілю, історії участі та донатів.
- **App.xaml.cs** – файл з логікою запуску додатку.

```
<Application x:Class="AuctionsDesktopApp.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <!-- Глобальні стилі тут не використовуються -->
    </Application.Resources>
</Application>
```

Рис. 3.1 – Основний вміст App.xaml

Після створення стартової структури проєкту наступним етапом стало формування основного шаблону головного вікна – **MainWindow.xaml**, яке виконує роль навігаційного контейнера для всієї програми. Це вікно є базовим «хабом», з якого користувач переходить до перегляду аукціонів, історії участі, створення лоту, а також до профілю або донату. Грамотна реалізація цього компонента дозволяє забезпечити логічну, зручну та передбачувану структуру навігації в межах усього застосунку.

У вікні MainWindow.xaml використано класичну комбінацію контейнерів компонування: Grid як основа для побудови всієї структури, а також StackPanel і DockPanel для окремих груп елементів (меню, заголовок, кнопки переходу). Для головного вмісту використовується Frame – спеціальний компонент, який дозволяє динамічно змінювати вміст у центральній частині вікна залежно від дій користувача.

Одним із центральних елементів базового шаблону інтерфейсу в MainWindow.xaml є навігаційна панель, яка забезпечує швидкий доступ до основних розділів програми: перегляду аукціонів, перегляду активності, створення лоту, профілю користувача та сторінки донату. Ця панель розташовується в нижній частині вікна і реалізована через контейнер UniformGrid, який автоматично розподіляє елементи в однакові за розміром стовпці.

Для створення зручного й сучасного вигляду кожна кнопка навігації містить іконку та підпис, які згруповано в StackPanel з горизонтальним центруванням. Завдяки цьому інтерфейс виглядає структуровано, а користувач інтуїтивно розуміє, що означає кожна дія.

```
<!-- Navigation -->
<UniformGrid Grid.Row="2" Columns="5" HorizontalAlignment="Center" Margin="0,20,0,0">
    <Button Style="{StaticResource RoundedButtonStyle}" Margin="0,5,0,5" Click="Auctions_Click" Cursor="Hand" Width="100">
        <StackPanel HorizontalAlignment="Center">
            <Image Source="/Icons/Bit.png" Height="30" Margin="0,0,0,5"/>
            <TextBlock Text="Аукціони" TextAlignment="Center"/>
        </StackPanel>
```

Рис. 3.2 – Приклад XAML-коду навігаційної кнопки

Навігаційна панель стилістично узгоджена з рештою інтерфейсу, всі кнопки мають однакову ширину, однакові відступи, фірмовий синій колір у стилі RoundedButtonStyle або унікальний стиль DonateButtonStyle. Усі стилі були задані локально, без використання ResourceDictionary, що дало змогу оперативно адаптувати зовнішній вигляд кожної кнопки без переходу до окремих словників.

При натисканні кожна кнопка викликає метод переходу, що змінює Frame.Content на відповідну сторінку – наприклад, CatalogWindow, ActivityWindow, CreateLotWindow.

Крім цього, у стилях кнопок навігаційної панелі реалізовано анімацію наведення – коли користувач підводить курсор до кнопки, її фон плавно змінює колір, підкреслюючи активність елемента. Така взаємодія створює приємний візуальний ефект і покращує користувацький досвід, роблячи інтерфейс більш «живим» та інтуїтивним.

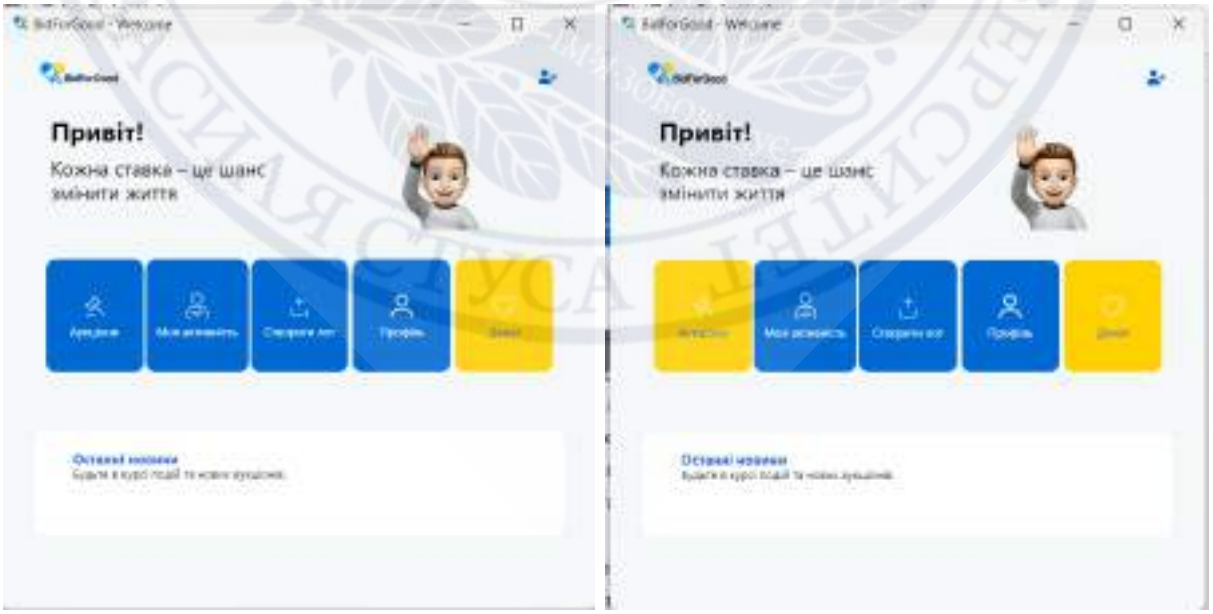


Рис. 3.3 – 3.4 - Реалізоване головне вікно додатку

Важливо зазначити, що у проєкті не використовуються ResourceDictionary – всі стилі, шрифти, кольори задаються безпосередньо в XAML-коді кожного вікна [27]. Такий підхід має як плюси, так і мінуси. З одного боку, він дозволяє швидко змінювати вигляд елементів локально, не вносячи зміни у загальний словник стилів. Це особливо зручно на етапі макетування, експериментів із зовнішнім виглядом або точкових правок. З іншого боку, відсутність централізації стилів призводить до дублювання коду, знижує масштабованість і ускладнює уніфікацію інтерфейсу при подальших зміненнях. Це рішення дозволяє зосередитися на функціоналі та перевірці UX-гіпотез.

Також під час програмування сторінки було враховано можливість масштабування сторінки при розгортанні та згортанні до необхідного розміру вікна користувчу. Елементи вийкна гармонійно адаптуються під розмір вікна.

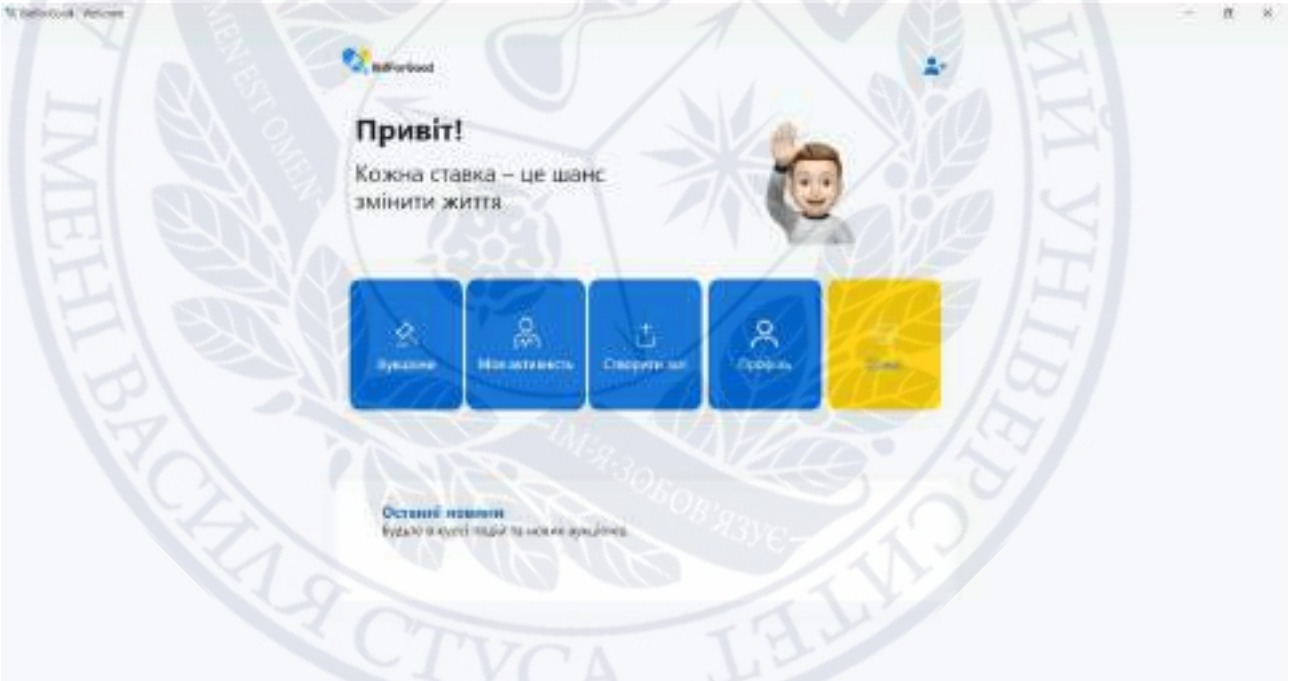


Рис. 3.5 – Адаптація елементів до масштабу сторінки

Таким чином, реалізована навігаційна панель забезпечує зрозумілу логіку переміщення між основними функціями додатку, візуальну єдність і зручність у використанні. Такий підхід підтримує принципи UX-дизайну, де кожна дія користувача має бути передбачуваною, доступною з мінімальною кількістю

кроків і зрозумілою з першого погляду. Це рішення було зручно реалізувати на основі WPF та XAML без складних шаблонів, з акцентом на ефективність, простоту та чистоту інтерфейсу.

3.2 Розробка основних сторінок та компонентів інтерфейсу

Одним із ключових етапів розробки клієнтської частини десктопного додатку є створення основних вікон інтерфейсу користувача. Кожне з цих вікон виконує певну роль у функціональній структурі застосунку та забезпечує доступ користувача до конкретних можливостей – перегляд лотів, участь в аукціонах, створення нових лотів, донати, керування профілем тощо. Усі сторінки створені в рамках єдиної візуальної концепції, заснованої на прототипах, реалізованих у Figma.

Загалом у рамках реалізації фронтенд-частини було створено 18 окремих вікон, кожне з яких має власну структуру, стилістику, логіку обробки подій і взаємодію з сервером. Вони охоплюють увесь спектр дій, які може виконувати користувач у межах додатку – від авторизації та перегляду лотів до управління профілем і отримання повідомлень про перемогу.

Використовуючи архітектуру MVVM, для кожного вікна реалізовано логічно поділену структуру: XAML-файл відповідає за вигляд (View), ViewModel-клас – за обробку подій, а Model (DTO) – за отримання або надсилання даних. Уся логіка роботи – від натискання кнопки до збереження в базі даних – розділена між цими трьома рівнями, що дозволяє зберігати чистоту коду та спрощує підтримку. Це також дало змогу забезпечити високу гнучкість, повторне використання компонентів у різних частинах інтерфейсу та масштабованість застосунку.

LoginWindow і RegisterWindow

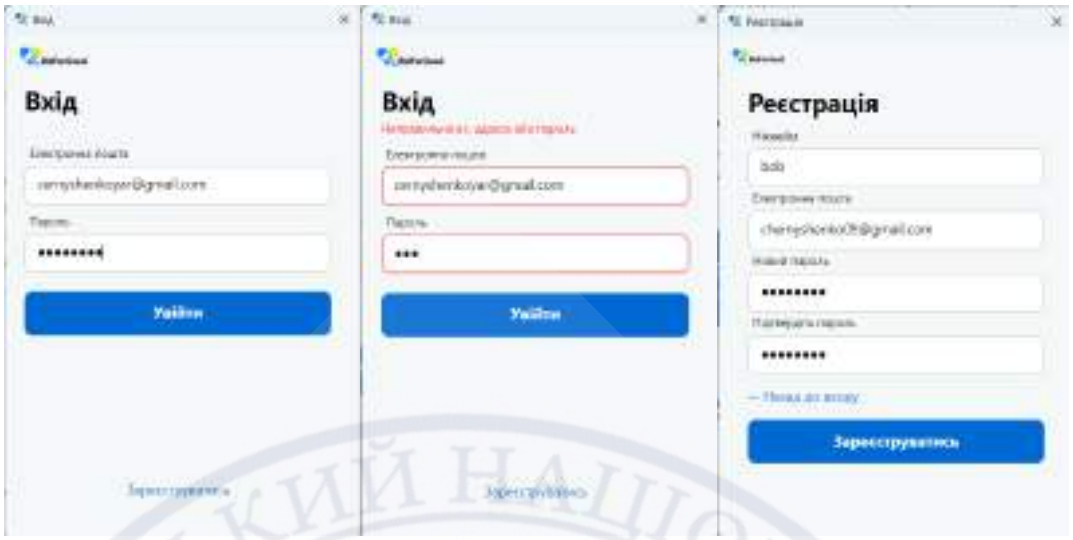


Рис. 3.6 – Демонстрація вікно входу та реєстрації

Ці два вікна відповідають за авторизацію та реєстрацію користувача. У LoginWindow передбачено введення email та пароля. Поля мають валідацію: якщо значення порожнє або некоректне – відображається повідомлення про помилку, поле підсвічується червоним.

Окрім цього, у RegisterWindow реалізовано автоматичну перевірку коректності email і захист від повторної реєстрації з тим самим логіном. При введенні некоректного пароля система не лише видає підказку, а й блокує можливість продовжити, що знижує кількість помилок на стороні сервера.

У RegisterWindow додано поле підтвердження пароля та механізм перевірки його збігу. Після успішної реєстрації відкривається EmailCodeWindow для введення коду з пошти. Запити до API /auth/login та /auth/register надсилаються через HttpClient, після чого користувач отримує токен.

CatalogWindow

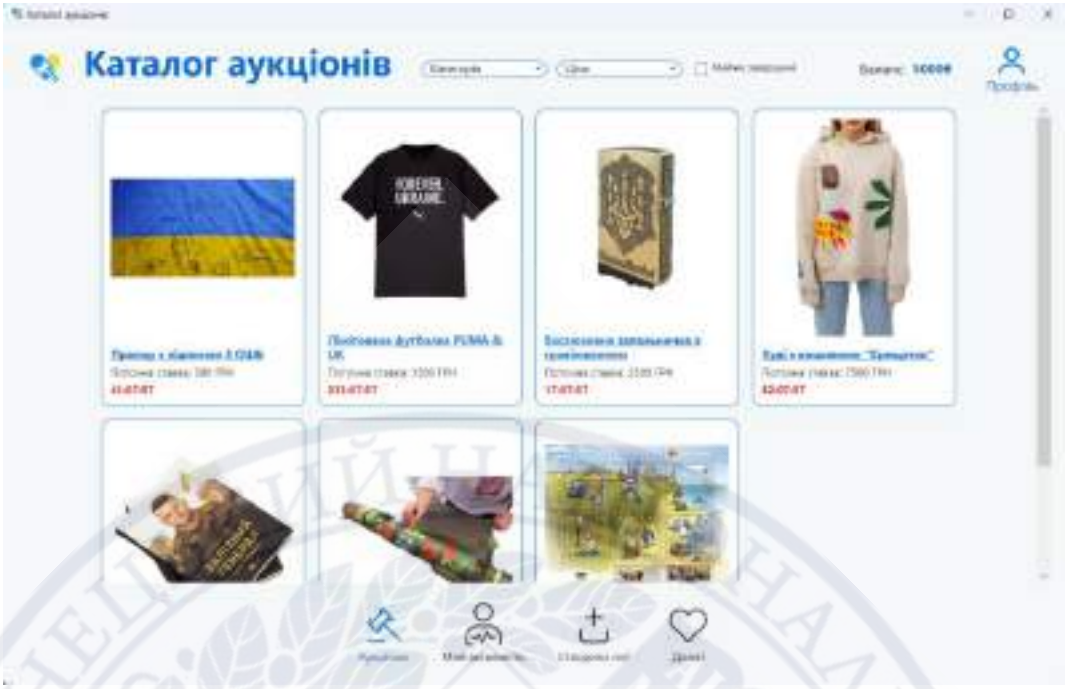


Рис. 3.8 – Каталог активних аукціонів

Це вікно програми, де відображається список активних лотів. Кожен лот представлений у вигляді плитки: фото, назва, стартова ціна, поточна ставка, таймер завершення. Дані отримуються з API-запиту GET /auctions. Результат запиту парситься у список ObservableCollection<AuctionDTO>, прив'язаний до ItemsControl.

Оновлення інформації про лоти відбувається за допомогою SignalR, який отримує повідомлення з сервера, коли:

- по лоту зроблено нову ставку;
- завершено таймер;
- додано новий лот.

Таймер кожного лоту оновлюється щосекунди через DispatcherTimer, що забезпечує відлік до завершення аукціону в режимі реального часу [28].

Користувач може відфільтрувати лоти за категоріями (через ComboBox), або відсортувати за датою завершення/ціною. Запити надсилаються через API /auctions із відповідними параметрами фільтрації.

LotDetailsWindow

Рис. 3.8 – Вікно з детальним описом лоту

В цьому вікно відображається деталізоване представлення окремого аукціону та відкривається при виборі лоту в каталозі. Його основна мета – надати користувачеві повну інформацію про обраний лот і дозволити взяти участь у торгах. Інтерфейс вікна включає фотографію лоту, його назву, повний опис, стартову і поточну ціну, інформацію про благодійний фонд або мету збору, історію ставок, статус аукціону та таймер завершення.

Особливою рисою цього вікна є повна синхронізація з сервером у режимі реального часу. Завдяки використанню SignalR, усі зміни, пов’язані з лотом – нові ставки, оновлення статусу, завершення аукціону – миттєво відображаються в інтерфейсі без необхідності ручного оновлення [29]. Така динамічність створює ефект «живого» аукціону, де користувач постійно залучений у процес.

У випадку помилки – наприклад, якщо введена ставка менша за поточну або некоректна – користувач отримує чітке модальне повідомлення з поясненням проблеми.

LotDetailsWindow забезпечує зручну, інформативну та інтерактивну взаємодію з лотом, дозволяючи користувачам брати участь в аукціоні в реальному часі, з мінімальними затримками і максимальною наочністю.

CreateLotWindow

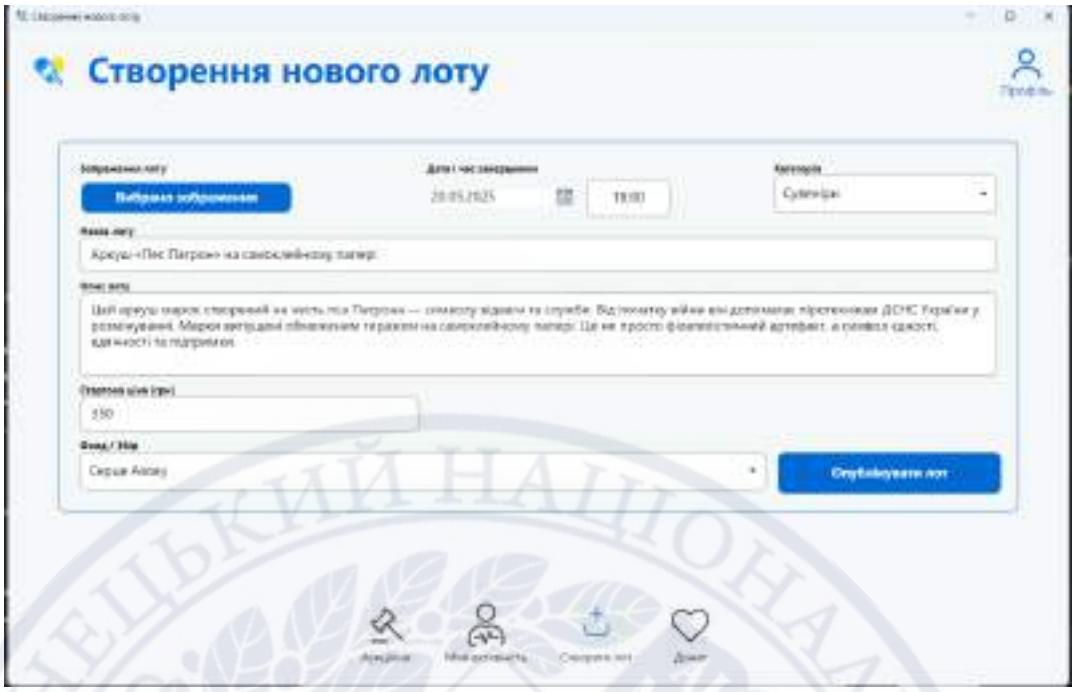


Рис. 3.8 – Форма заповнення інформації створення нового лоту

Вікно CreateLotWindow реалізоване для забезпечення ключової функції платформи – можливості створення нових лотів зареєстрованими користувачами. Це дозволяє кожному охочому виставити на аукціон власний товар або артефакт, кошти з якого будуть спрямовані на підтримку обраного благодійного фонду. Таким чином, вікно виконує важливу роль у залученні користувачів до активної участі в системі.

Зовні це одне з найбільш насичених вікон додатку, проте структура лишається інтуїтивно зрозумілою. Усі поля – назва, опис, стартова ціна, дата завершення, категорія, благодійний фонд та кнопка вибору зображення – логічно розташовані вертикально, мають зрозумілі підписи та плейсхолдери. Валідація реалізована для кожного поля: перевіряється числове значення ціни, дата не може бути в минулому, а зображення обов’язкове для завантаження. Після натискання кнопки “Створити” виконується перевірка всіх введених даних, і лише за її успішного проходження формується об’єкт AuctionDTO, який надсилається на сервер через API.

Після створення нового лоту всі користувачі в каталозі отримують оновлення в режимі реального часу через SignalR. У випадку помилки,

користувач бачить відповідне повідомлення з підказкою, а у разі успіху – підтвердження з інформацією про доданий лот.

ActivityWindow

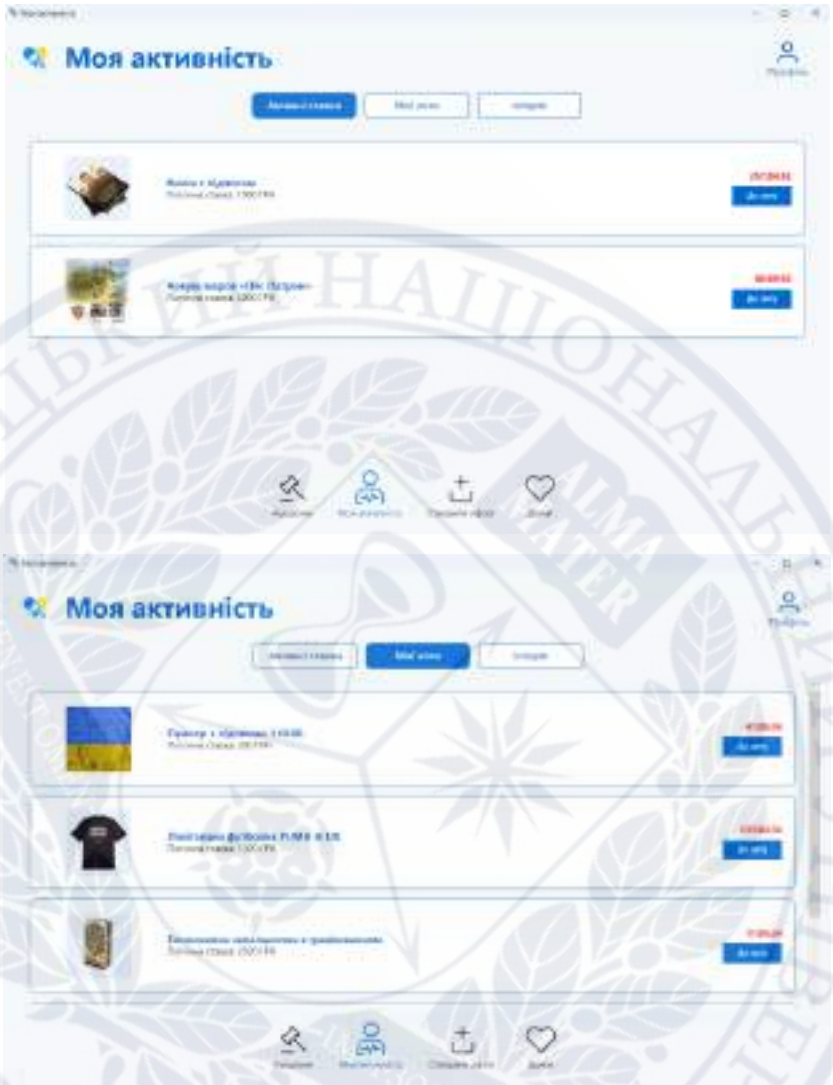


Рис. 3.9 – 3.10 – Вікно активностей користувача

Це вікно дозволяє користувачу переглядати власну активність. Унікальна особливість цього вікна – вкладки, які фільтрують інформацію для користувача:

- Активні ставки;
- Створені лоти;
- Історія участі.

Кожна вкладка формує окремий запит до API, наприклад, GET /user/bids, GET /user/auctions. Навігація реалізована через кнопки з кастомним стилем:

активна вкладка підсвічується синім, інші – сірим. Дані під кожною вкладкою завантажуються лише при її відкритті (lazy loading), що економить ресурси. Таймери працюють локально, але синхронізуються із серверними даними. При натисканні на лот – відкривається LotDetailsWindow.

DonateWindow

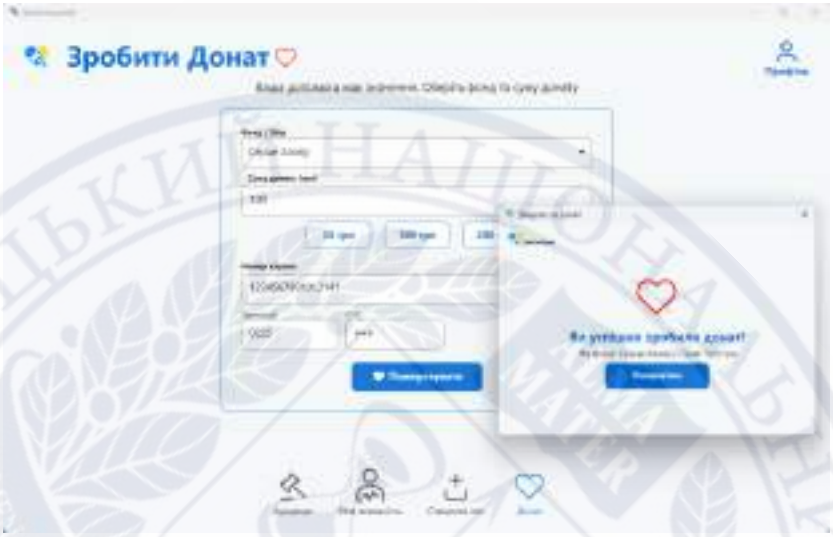


Рис. 3.11 – Успішний благодійний внесок на сторінці «Донат»

Вікно DonateWindow створене з метою забезпечити зручний і швидкий спосіб переказу коштів користувачем на підтримку благодійних організацій, представлених у системі. Основна його функція – реалізація окремого каналу для фінансової підтримки, незалежно від участі в аукціонах. Це дозволяє користувачам долучитися до благодійності навіть без активної участі у торгах.

У цьому вікні користувач може ввести суму пожертви, обрати благодійний фонд зі списку (реалізованого через ComboBox) та підтвердити платіж. Перед надсиланням запиту відбувається валідація введених даних: перевірка на заповненість, правильність числового формату та вибір фонду. Після успішного донату система оновлює дані користувача (баланс, статистику донатів) і відображає відповідне повідомлення. Таким чином, DonateWindow виконує важливу соціальну функцію у структурі застосунку, підвищуючи його ефективність і залученість користувачів у волонтерську діяльність.

ProfileWindow

Рис. 3.12 – Профіль користувача

Вікно ProfileWindow реалізовано як особистий простір користувача, де відображаються ключові персональні дані та статистика його взаємодії з платформою. Основною метою цього вікна є надання зручного доступу до особистої інформації та її редагування, а також забезпечення прозорості у відображенні досягнень користувача – виграних лотів, зроблених донатів, загальної активності.

В інтерфейсі вікна відображаються ім’я користувача, email, баланс, кількість виграних аукціонів, загальна сума пожертв. Усі ці дані завантажуються з бекенду через запит до API /user/profile. За допомогою кнопки «Редагувати» користувач може змінити свій email або пароль. Усі зміни супроводжуються валідацією введених даних – перевіряється коректність email-адреси, мінімальна довжина пароля та відповідність підтвердження. Після успішного збереження користувач отримує спливаюче повідомлення про оновлення даних, що створює зворотний зв’язок і підтвердження дії.

Окрім основних функціональних вікон, у застосунку були реалізовані також допоміжні компоненти, які покращують загальну зручність, безпеку й інформативність взаємодії користувача із системою. Такі вікна доповнюють ключовий функціонал, створюючи цілісну й передбачувану логіку роботи програми. До них належать модальні повідомлення, підтвердження дій,

інформування про помилки, а також спеціальні інтерфейси для підтвердження реєстрації та повідомлення про перемогу в аукціоні.

Зокрема, було реалізовано:

- **ConfirmCodeWindow** – вікно підтвердження коду, яке відкривається після реєстрації. Користувач отримує на пошту шестизначний код і вводить його в окремому полі. Без верифікації доступ до системи заблокований, що підвищує рівень безпеки.
- **ErrorWindow** та **MessageDialogWindow** – використовуються для відображення помилок (некоректні поля, помилка з'єднання, невдала дія). Всі вікна мають однакове оформлення: червона рамка, іконка помилки, кнопка “Добре”.
- **SuccessWindow** – універсальне вікно успіху, яке використовується після успішної дії, наприклад, при оновленні профілю, перемозі в аукціоні чи поповненні балансу. Має відповідну іконку, повідомлення і кнопку закриття.
- **WinNotificationWindow** – окреме вікно, що відкривається одразу після перемоги в аукціоні. Воно інформує користувача, що його ставка стала найвищою, і пропонує подальші інструкції. Інформація також дублюється на електронну пошту.
- **ConfirmationDialog** – модальне вікно підтвердження дії з варіантами “Так” або “Ні”, що використовується для виходу, скасування або видалення.
- **BlockedWindow** – інтерфейс повідомлення про блокування акаунта, що відображається при виявленні порушень або підозрілої активності.
- **CoolDownWindow** – спрацьовує після кількох невдалих спроб входу. Відображає повідомлення з проханням зачекати перед повторною спробою.
- **EditProfileWindow** – окреме вікно для редагування акаунта, з перевіркою введених даних, підсвіткою помилок і підтвердженням змін.
- **TopUpBalanceWindow** – вікно поповнення балансу з полями для введення суми, картки, терміну дії та CVC. Містить валідацію й підтвердження операції.

Кожен із цих інтерфейсів відповідає загальному стилю програми – з використанням фірмових кольорів, заокруглених кнопок, шрифта Inter та інтуїтивної структури. Їхня реалізація дозволила підвищити зручність користування, зменшити кількість помилок і створити довершений користувацький досвід

3.3 Інтеграція з back-end частиною

Після реалізації основних вікон інтерфейсу користувача та налаштування логіки взаємодії на рівні front-end, важливим етапом стало забезпечення повноцінної інтеграції з серверною частиною системи. Саме бекенд відповідає за збереження та обробку даних, автентифікацію, збереження ставок, створення лотів, надсилання листів та оновлення інформації в реальному часі. Для цього була створена окрема серверна частина (back-end), що побудована на основі фреймворку ASP.NET Core Web API [30].

Бекенд застосунку реалізовано у вигляді RESTful API із підключенням SignalR для обміну повідомленнями в реальному часі. Для зберігання даних використовується MySQL, зокрема база даних *auctions_db*, яка містить таблиці користувачів, лотів, ставок, категорій тощо. ORM-прошарок побудовано на Entity Framework Core, що дозволило ефективно працювати з сутностями через C#-класи, не використовуючи сирі SQL-запити [31].

У Program.cs бекенд-застосунку було зареєстровано всі сервіси та залежності, включаючи репозиторії, сервіси відправки пошти, мапінгу об'єктів, SignalR-хаб і контекст бази даних AuctionDbContext, який підключено за допомогою UseMySql. Під час запуску API також ініціалізується перевірка лотів через фоновий сервіс AuctionCheckerService, що автоматично завершує аукціони, коли вичерпується час.

На клієнтській стороні була реалізована спеціальна структура контролерів для обробки запитів до API. Наприклад, AuthController відповідає за логіку авторизації та реєстрації, AuctionStorageController – за обробку та фільтрацію

лотів, а UserStatusController – за збереження сесії користувача та завантаження персональної інформації. Обмін даними з API відбувається за допомогою HttpClient, а у випадку SignalR – через хаб.

При авторизації або реєстрації на стороні клієнта формується GET або POST-запит на відповідний маршрут [32].

Сервер обробляє запит, створює нового користувача в базі, надсилає лист-підтвердження та повертає статус.

Передача даних відбувається за допомогою DTO-об’єктів (Data Transfer Objects). Наприклад, AuctionDTO, UserDTO або BidDTO містять тільки необхідні дані без зайвої логіки, що дозволяє ефективно передавати інформацію між клієнтом і сервером. Ці об’єкти використовуються як для надсилання запитів, так і для відображення відповідей, які клієнт отримує від API [33].

Одним із ключових елементів взаємодії є підтримка реального часу через SignalR [34][35]. Усі клієнти підключаються до SignalR-хабу /mainHub, а сервер через методи цього хабу надсилає повідомлення про нові ставки, завершення аукціонів або появу нових лотів. Це дозволяє не оновлювати сторінку вручну – дані змінюються автоматично.

Наприклад, при створенні нового лоту користувач надсилає об'єкт AuctionDTO, який сервер приймає через SignalR, зберігає у базі, а потім повідомляє про це всіх інших користувачів.

Інформація про авторизованого користувача зберігається у UserStatusController. Там фіксується email, ID, баланс, кількість виграних лотів, загальна сума донатів тощо.

Під час взаємодії з бекендом кожна дія перевіряється на стороні API – наприклад, чи існує користувач, чи правильно введено пошту, чи можна зробити ставку. У випадку помилки сервер повертає повідомлення, яке обробляється клієнтом та відображається через ErrorWindow або MessageDialogWindow.

Ще одним важливим аспектом є надсилання email-повідомлень. На сервері реалізовано IEmailService, який надсилає:

- код підтвердження при реєстрації;

- сповіщення про перемогу у ставці;
- повідомлення власнику лоту про завершення аукціону.

Це реалізовано через звичайні SMTP-виклики, з використанням системної бібліотеки System.Net.Mail [36][37].

Таким чином, клієнтська частина програми інтегрована з бекендом на всіх рівнях: автентифікація, робота з лотами, ставки, донати, баланс, а також обробка подій у реальному часі через SignalR. Серверна логіка побудована на принципах REST + SignalR та дозволяє масштабувати систему, додавати нові функції, не порушуючи вже реалізовану архітектуру. Завдяки такій взаємодії забезпечується надійна синхронізація між користувачами, безпечна авторизація та висока швидкодія інтерфейсу.

3.4 Тестування та оптимізація

Після завершення розробки основних функціональних модулів та інтеграції клієнтської частини з серверною, важливим етапом стало проведення тестування і базової оптимізації десктопного застосунку [38]. Цей етап є ключовим для забезпечення надійної, стабільної та передбачуваної роботи системи, а також для виявлення потенційних помилок і слабких місць у логіці, інтерфейсі або взаємодії з API.

На першому етапі було проведено ручне функціональне тестування кожного з вікон програми. Перевірка охоплювала такі основні сценарії:

- успішна авторизація й обробка помилкових email/паролів;
- реєстрація користувача та підтвердження через код;
- відображення списку лотів, правильна робота фільтрів і таймерів;
- створення нового лоту з усіма варіантами валідації;
- участь в аукціоні, оновлення ставок у реальному часі;
- перемога у лоті та отримання повідомлень через SignalR;
- відправка повідомлень на пошту;

- редагування профілю та вивід оновлених даних;
- поповнення балансу та валідація платіжних полів.

Для виявлення логічних помилок використовувалися breakpoints у ViewModel-класах, а також аналіз відповідей API через інструменти Postman та логування у Visual Studio [39].

Особливу увагу було приділено перевірці роботи додатку у випадках неправильних дій або проблем на стороні сервера. Зокрема, тестувалися ситуації:

- введення неіснуючого email або неправильного коду підтвердження;
- блокування акаунта або перевищення кількості спроб входу (відображення CoolDownWindow або BlockedWindow);
- спроба створити лот без заповнення обов’язкових полів;
- спроба надіслати ставку нижчу за поточну.

У таких випадках програма коректно відображає вікна помилок (ErrorWindow) або повідомлення з поясненням дії. Це дозволяє користувачу не розгубитися і зрозуміти, що саме пішло не так.

Кожне вікно було протестоване на зручність використання та інтуїтивність інтерфейсу. Було оцінено:

- логіку навігації між вікнами (перехід по кнопках у головному меню);
- поведінку елементів при зміні розмірів вікна (масштабування, адаптивність);
- зручність валідації (видимість повідомлень, підсвітка полів, реакція на помилки);
- відповідність до попередніх макетів у Figma.

Також було перевірено всі стилі: кольори, шрифти, розміри кнопок і відступи. Це дозволило виявити й виправити кілька неузгодженостей у стилях між CreateLotWindow і ProfileWindow.

У процесі тестування було виявлено, що таймер у каталозі лотів (CatalogWindow), який оновлювався кожну секунду, при великій кількості лотів

спричиняв навантаження на UI. Для цього була впроваджена оптимізація через DispatcherTimer з перевіркою Visibility плитки – таймери оновлюються лише для видимих лотів.

Тестування дозволило забезпечити високу стабільність додатку, виявити низку логічних та інтерфейсних недоліків, які були успішно усунуті. Після оптимізації продуктивності та перевірки основних сценаріїв взаємодії, система стала повністю готовою до демонстрації й подальшого використання.

У перспективі проєкт має потенціал для подальшого розширення функціональності. Наприклад, можна додати можливість створення аукціонів із захищеними ставками (blind bids), впровадити систему рейтингів користувачів, реалізувати чат між продавцем і покупцем, або інтегрувати сторонню платіжну систему для реальних транзакцій. Також доцільним буде адаптація додатку під мобільні платформи або створення веб-версії для ширшого охоплення аудиторії. З технічного боку, можлива оптимізація SignalR-взаємодії, розширення аналітики у профілі користувача, а також впровадження адмін-панелі для модерації лотів. Усе це дозволить зробити платформу ще більш гнучкою, масштабованою та зручною як для волонтерських організацій, так і для звичайних користувачів.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було досліджено, спроєктовано та реалізовано фронтенд-частину десктоп додатку для благодійних онлайн-аукціонів, з акцентом на UI/UX-інтерфейс та інтерактивні функції. Об'єктом дослідження став сам процес створення клієнтської частини системи, а предметом – практичні аспекти реалізації інтерфейсів користувача.

На основі аналізу предметної області було встановлено, що попри широке використання соціальних мереж та платформ збору коштів, спеціалізовані рішення для проведення благодійних аукціонів зустрічаються рідко. Це підтвердило актуальність теми та визначило напрямки дослідження. Виявлено особливості UI/UX-дизайну, притаманні системам соціального призначення, зокрема: доступність, прозорість, передбачуваність дій та довіра до візуального середовища.

Систематизовано вимоги до інтерфейсної частини на основі сучасних рекомендацій з проектування. Підготовлено візуальні макети сформовано єдину стилістичну систему – кольорова палітра, типографіка, компоненти керування. Було уточнено й реалізовано архітектурний підхід MVVM у середовищі WPF, що дозволило ефективно розмежувати логіку та інтерфейс.

У роботі було реалізовано 19 вікон, що охоплюють повний життєвий цикл взаємодії користувача із системою: від реєстрації та підтвердження email до перегляду профілю, участі в аукціонах, здійснення донатів, створення лотів і отримання повідомлень. Здійснено підключення до серверної частини через REST API та забезпечено обмін подіями в реальному часі. Усі основні сторінки проходили перевірку на працездатність, коректність обробки даних та відгук інтерфейсу.

Результатом виконаної роботи став інтерактивний прототип клієнтської частини, готовий до розгортання або подальшої інтеграції з повноцінною back-end системою. Реалізоване рішення відповідає базовим критеріям якості

програмного забезпечення – зокрема масштабованості, адаптивності, зручності використання.

Отримані знання та навички з проєктування UI/UX, роботи у середовищі WPF, організації взаємодії з API та реалізації клієнтських логік стали результатом професійної підготовку й важливим практичним підґрунтям для майбутньої діяльності в галузі розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Бондаренко С. І. Благодійні онлайн-аукціони як інноваційна форма соціального підприємництва. Вісник ЧДТУ. Серія: Економічні науки. Черкаси: ЧДТУ, 2016. № 4. С. 62–66. URL: https://er.chdtu.edu.ua/bitstream/ChSTU/1648/1/9.pdf (дата звернення: 11.04.2025).
2. eBay For Charity. URL: https://charity.ebay.com/ (дата звернення 11.03.2025)
3. Prozorro.Sale - Електронна Торгова Система URL: https://prozorro.sale (дата звернення: 11.04.2025).
4. Charity Fundraising Auctions for Schools & Nonprofits. URL: https://www.biddingforgood.com/ (дата звернення 11.04.2025)
5. Platfor.ma. URL: https://www.platfor.ma (дата звернення: 11.04.2025).
6. dobro.ua - допомога Україні. URL: https://dobro.ua (дата звернення: 11.04.2025).
7. United24. Офіційна платформа допомоги Україні. URL: https://u24.gov.ua (дата звернення: 11.04.2025).
8. Dhaduk H. The Impact of User Experience on Online Auction Success: Key Features to Include. *Medium*. URL: https://medium.com/@HirenDhaduk1/the-impact-of-user-experience-on-online-auction-success-key-features-to-include-221937999b01 (дата звернення: 12.04.2025).
9. Токени аутентифікації. *Онлайн-курси від компанії QATestLab* URL: https://training.qatestlab.com/blog/technical-articles/authentication-tokens/ (дата звернення: 12.04.2025).
10. Windows Presentation Foundation | WPF & .NET | Visual Studio. *Visual Studio*. URL: https://visualstudio.microsoft.com/vs/features/wpf/ (дата звернення: 12.04.2025).
11. Model-View-ViewModel - .NET. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/uk-ua/dotnet/architecture/maui/mvvm (дата звернення: 12.04.2025).

12. Клієнт-серверна архітектура. *Онлайн-курси від компанії QATestLab* . URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 14.04.2025).
13. ПРИКЛАДНИЙ РІВЕНЬ (APPLICATION LAYER). *Stud*. URL: https://stud.com.ua/94311/informatika/prikladniy_riven_application_layer (дата звернення: 14.04.2025).
14. Studios J. Desktop App Development: A Complete Guide for 2025. *Medium*. URL: https://jhavtech.medium.com/desktop-app-development-a-complete-guide-for-2025-931d4fe354e4 (дата звернення: 16.04.2025).
15. What is Avalonia? | Avalonia Docs. *Avalonia Docs | Avalonia Docs*. URL: https://docs.avaloniaui.net/docs/overview/what-is-avalonia (дата звернення: 16.04.2025).
16. What is .NET MAUI? - .NET MAUI. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/uk-ua/dotnet/maui/what-is-maui?view=net-maui-9.0 (дата звернення: 16.04.2025).
17. What is windows forms - windows forms. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/ (дата звернення: 05.05.2025).
18. Microsoft Visual Studio: що це, для чого це потрібно та як це працює. *Rivenditore software: Microsoft Office, Windows, Adobe | Macrosoft*. URL: https://macrosoft.store/uk/blog/post/29-для-чого-потрібна-microsoft-visual-studio (дата звернення: 16.04.2025).
19. Figma для початківців - огляд популярного інструменту для дизайнерів. *ITSTEP Академія Київ*. URL: https://kiev.itstep.org/blog/figma-is-a-basic-tool-for-designers (дата звернення: 16.04.2025).
20. XAML styles - Windows apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/uk-ua/windows/apps/develop/platform/xaml/xaml-styles?view=azurermps-1.7.0 (дата звернення: 16.04.2025).

21. PNN Soft. *Компанія по розробці програмного забезпечення* URL: https://pnn.com.ua/ua/blog/detail/git-gitlab-and-github-difference-and-peculiarities-of-version-control-systems (дата звернення: 16.04.2025).
22. User interface це про забезпечення користувачам зручності. *FoxmindEd*. URL: https://foxminded.ua/user-interface-tse/ (дата звернення: 21.04.2025).
23. Прототипування за допомогою Figma, Sketch та Adobe XD: порівняльний аналіз та практичні прийоми роботи - Блог Mate academy. URL: https://mate.academy/blog/ui-ux-design/prototyping-tools/ (дата звернення: 21.04.2025).
24. Sketch - найкращий інструмент для дизайну в Mac OS. Підручний посібник для початківців. *Mediacom*. URL: https://mediacom.com.ua/sketch-najkrashij-instrument-dlya-dizajnu-v-mac-os-pidruchnij-posibnik-dlya-pochatkivtsiv/ (дата звернення: 21.04.2025).
25. 10 advantages of Figma for your website designs. *Ouiflow - Nous crÃ©ons des sites Webflow performants*. URL: https://www.ouiflow.io/en/ressources/blog/10-advantages-figma-design-website-webflow (дата звернення: 21.04.2025).
26. Принципи візуальної ієрархії в графічному дизайні | BLOG Dizz Agency ➢ Dizz.in.ua. *DIZZ*. URL: https://dizz.in.ua/uk/princzipi-vizualnoi-ierarhii-v-grafichnomu-dizajni/ (дата звернення: 01.05.2025).
27. How to: Use a ResourceDictionary to Manage Localizable String Resources - WPF. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/uk-ua/dotnet/desktop/wpf/advanced/how-to-use-a-resourcedictionary-to-manage-localizable-string-resources (дата звернення: 01.05.2025).
28. DispatcherTimer Class (System.Windows.Threading). URL: https://learn.microsoft.com/uk-ua/dotnet/api/system.windows.threading.dispatchertimer?view=windowsdesktop-8.0 (date of access: 01.05.2025).
29. Bandara B. WPF to WPF Real-Time Communication Made Easy with SignalR and .NET Core. *LinkedIn: Log In or Sign Up*.

URL: https://www.linkedin.com/pulse/wpf-real-time-communication-made-easy-signalr-net-core-bandara-64htc/ (дата звернення: 02.05.2025).

30. Getachew S. Introduction to ASP.NET Core Web API: A Complete Guide for Developers. *Medium*. URL: https://medium.com/@solomongetachew112/introduction-to-asp-net-core-web-api-a-complete-guide-for-developers-347df16b8ab6 (дата звернення: 02.05.2025).
31. EduGaon R. Connect Frontend To Backend. *Medium*. URL: https://medium.com/@Rinki.EduGaon/connect-frontend-to-backend-b2c56bec8484 (дата звернення: 05.05.2025).
32. Методи передачі даних (GET і POST). *Český PHP manuál*. URL: https://uk.php.brj.cz/metodi-peredaci-danih-get-i-post (дата звернення: 05.05.2025).
33. Про проблему DTO та шляхи її вирішення. *DOU: Спільнота програмістів*. URL: https://dou.ua/forums/topic/43912/ (дата звернення: 05.05.2025).
34. Mapping signalr users to connections. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/aspnet/signalr/overview/guide-to-the-api/mapping-users-to-connections (дата звернення: 05.05.2025).
35. SignalR A complete WPF client using MVVM. *Software Engineering*. URL: https://damienbod.com/2013/11/20/signalr-a-complete-wpf-client-using-mvvm/ (дата звернення: 05.05.2025).
36. What is SMTP? - SMTP Server Explained - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is/smtp/?nc1=h_ls (дата звернення: 05.05.2025).
37. Simple mail transfer protocol (SMTP) | CQR. *CQR*. URL: https://cqr.company/wiki/protocols/simple-mail-transfer-protocol-smtp/ (дата звернення: 05.05.2025).
38. Особливості тестування десктопних додатків: Стаття з блогу IT-школи Hillel. *Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна*

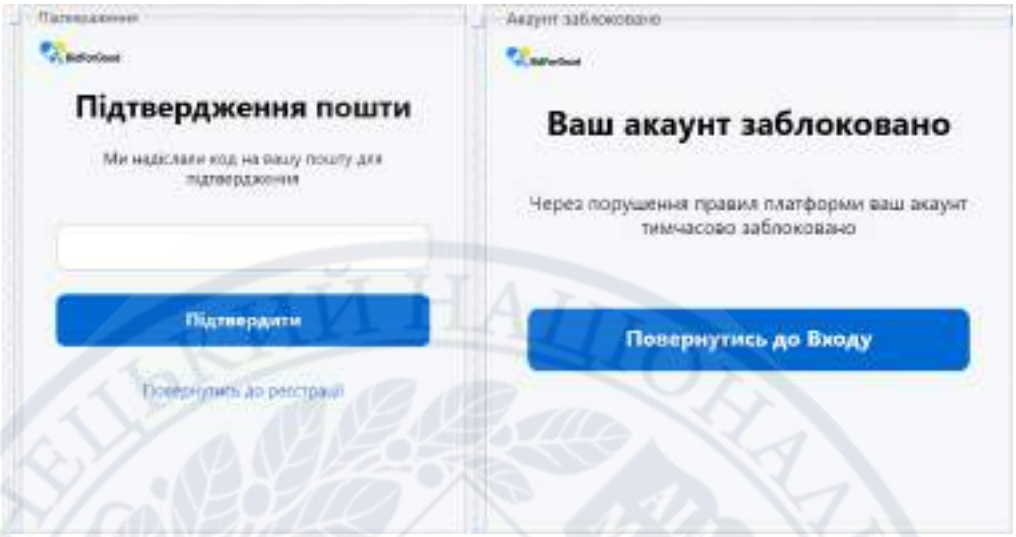
школа Hillel. URL: https://blog.ithillel.ua/articles/testing-desktop-applications (дата звернення: 05.05.2025).

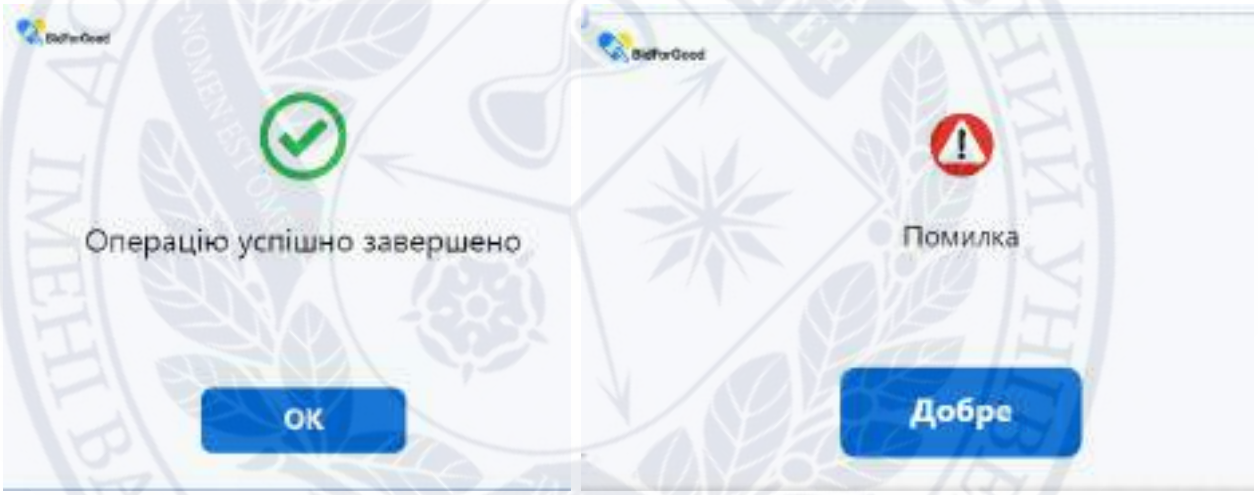
39. Microsoft.VisualStudio.Debugger.Breakpoints namespace. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.debugger.breakpoints?view=visualstudiosdk-2022 (дата звернення: 05.05.2025).
40. Чернишенко Я.А., Зелінська О.В. Застосування шаблону MVVM у реалізації UI логіки десктопних додатків. *Прикладні інформаційні технології VI Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених* : матеріали VI Всеукр. науково-практ. конф. здобувачів, аспірантів та молодих вчен., м. Вінниця. С. 87–89.

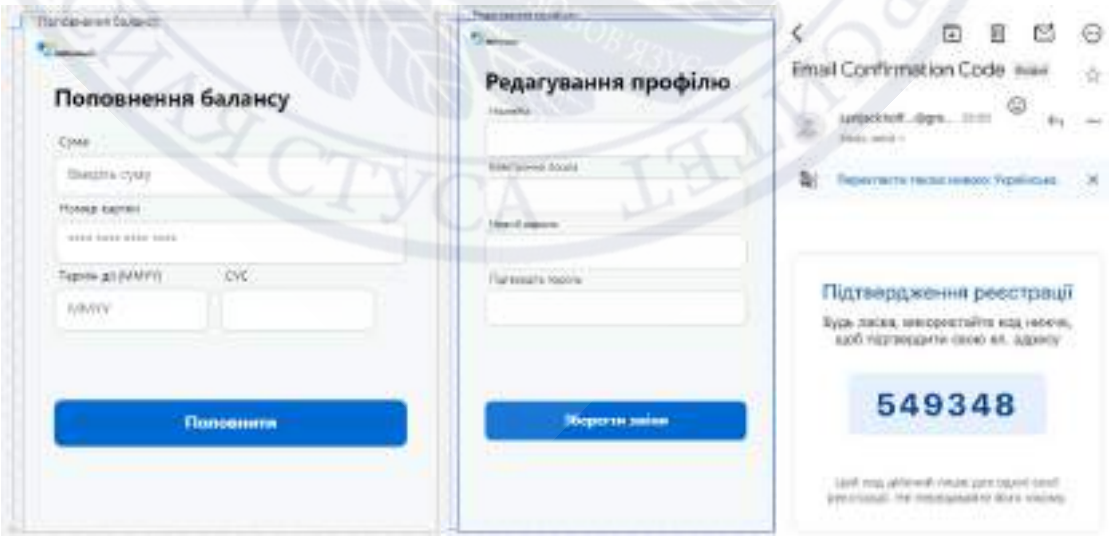
ДОДАТКИ

Додаток А

Додаткові вікна зі сповіщеннями та помилками







Додаток Б

Код розмітки головного меню програми

```
<Window x:Class="AuctionsDesktopApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="BidForGood - Welcome"
        Width="544" Height="544"
        ResizeMode="CanResize"
        WindowStartupLocation="CenterScreen"
        FontFamily="Inter"
        Background="#F4F8F9" Icon="/Icons/icon.png">

    <Window.Resources>
        <!-- Flat icon button -->
        <ControlTemplate x:Key="FlatImageButtonTemplate"
TargetType="Button">
            <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center"/>
        </ControlTemplate>

        <!-- Rounded blue action button -->
        <Style x:Key="RoundedButtonStyle" TargetType="Button">
            <Setter Property="Width" Value="110"/>
            <Setter Property="Height" Value="115"/>
            <Setter Property="FontFamily" Value="Inter"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="Background" Value="#0069D2"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="Button">
                        <Border x:Name="border"
                                Background="{TemplateBinding
Background}"
                                CornerRadius="10"
                                Padding="10">
                            <ContentPresenter
HorizontalAlignment="Center" VerticalAlignment="Center"/>
                        </Border>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsMouseOver"
Value="True">
                                <Trigger.EnterActions>
                                    <BeginStoryboard>
                                        <Storyboard>
                                            <ColorAnimation
Storyboard.TargetName="border"

Storyboard.TargetProperty="(Border.Background).(SolidColorBrush.Co
lor)"
```

```
To="#FFD100"

Duration="0:0:0.2"/>
                                        </Storyboard>
                                    </BeginStoryboard>
                                </Trigger.EnterActions>
                                <Trigger.ExitActions>
                                    <BeginStoryboard>
                                        <Storyboard>
                                            <ColorAnimation
Storyboard.TargetName="border"

Storyboard.TargetProperty="(Border.Background).(SolidColorBrush.Co
lor)"

To="#0069D2"

Duration="0:0:0.2"/>
                                        </Storyboard>
                                    </BeginStoryboard>
                                </Trigger.ExitActions>
                                <Setter Property="Foreground"
Value="#0069D2"/>
                            </Trigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>

        <!-- Yellow donate button -->
        <Style x:Key="DonateButtonStyle" TargetType="Button">
            <Setter Property="Width" Value="110"/>
            <Setter Property="Height" Value="115"/>
            <Setter Property="FontFamily" Value="Inter"/>
            <Setter Property="Foreground" Value="#0069D2"/>
            <Setter Property="Background" Value="#FFD100"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="Button">
                        <Border Background="{TemplateBinding
Background}" CornerRadius="10" Padding="10">
                            <ContentPresenter
HorizontalAlignment="Center" VerticalAlignment="Center"/>
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </Window.Resources>

    <!-- Scalable layout -->
```

```
    <Viewbox Stretch="Uniform">
        <Grid Width="544" Height="544" Margin="20">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <!-- Logo -->
            <StackPanel Orientation="Horizontal"
VerticalAlignment="Top" Margin="0,0,0,30">
                <Image Source="/Icons/logo.png" Height="30"
Margin="0,0,10,0"/>
            </StackPanel>

            <!-- Logout -->
            <Button Template="{StaticResource
FlatImageButtonTemplate}"
                    HorizontalAlignment="Right"
                    VerticalAlignment="Top"
                    Width="36" Height="36"
                    Cursor="Hand"
                    Click="Logout_Click">
                <Image Source="/Icons/logout.png" Width="22"
Height="22"/>
            </Button>



            <!-- Welcome + Avatar -->
            <StackPanel Orientation="Horizontal"
VerticalAlignment="Bottom" Grid.Row="1">
                <StackPanel>
                    <TextBlock Text="Привіт!" FontSize="26"
FontWeight="Bold" Margin="17,0,0,0"/>
                    <TextBlock Text="Кожна ставка – це шанс
змінити життя" FontSize="20"
                               Margin="17,10,0,0" Width="280"
Height="77" TextWrapping="WrapWithOverflow"/>
                </StackPanel>
                <Image Source="/Icons/avatar.png" Height="120"
Width="120" Margin="60,0,0,0"/>
            </StackPanel>

            <!-- Navigation -->
            <UniformGrid Grid.Row="2" Columns="5"
HorizontalAlignment="Center" Margin="0,20,0,0">
                <Button Style="{StaticResource
RoundedButtonStyle}" Margin="0,5,0,5" Click="Auctions_Click"
Cursor="Hand" Width="100">
                    <StackPanel HorizontalAlignment="Center">
```

```
                        <Image Source="/Icons/Bit.png" Height="30"
Margin="0,0,0,5"/>
                        <TextBlock Text="Аукціони"
TextAlignment="Center"/>
                    </StackPanel>
                </Button>

                <Button Style="{StaticResource
RoundedButtonStyle}" Margin="0,5,0,5" Click="Activity_Click"
Cursor="Hand" Width="105">
                    <StackPanel HorizontalAlignment="Center">
                        <Image Source="/Icons/History.png"
Height="30" Margin="0,0,0,5"/>
                        <TextBlock Text="Моя активність"
TextAlignment="Center"/>
                    </StackPanel>
                </Button>

                <Button Style="{StaticResource
RoundedButtonStyle}" Margin="0,5,0,5" Click="CreateLot_Click"
Cursor="Hand" Width="100">
                    <StackPanel HorizontalAlignment="Center">
                        <Image Source="/Icons/CreatBit.png"
Height="30" Margin="0,0,0,5"/>
                        <TextBlock Text="Створити лот"
TextAlignment="Center"/>
                    </StackPanel>
                </Button>

                <Button Style="{StaticResource
RoundedButtonStyle}" Margin="0,5,0,5" Click="Profile_Click"
Cursor="Hand" Width="100">
                    <StackPanel HorizontalAlignment="Center">
                        <Image Source="/Icons/profile.png"
Height="30" Margin="0,0,0,5"/>
                        <TextBlock Text="Профіль"
TextAlignment="Center"/>
                    </StackPanel>
                </Button>

                <Button Style="{StaticResource DonateButtonStyle}"
Margin="0,5,0,5" Click="Donate_Click" Cursor="Hand" Width="100">
                    <StackPanel HorizontalAlignment="Center">
                        <Image Source="/Icons/Donate.png"
Height="30" Margin="0,0,0,5"/>
                        <TextBlock Text="Донат"
TextAlignment="Center"/>
                    </StackPanel>
                </Button>
            </UniformGrid>

            <!-- News -->
```

```
            <Border Background="White" CornerRadius="10"
Margin="0,57,0,56" Padding="10" Grid.Row="3">
                <StackPanel Orientation="Horizontal">
                    <Image Source="/Icons/news.png" Width="20"
Height="20" Margin="0,10,10,0" VerticalAlignment="Top"/>
                    <StackPanel>
                        <TextBlock Text="Останні новини"
FontSize="14" FontWeight="Bold" Foreground="#0069D2"
Margin="0,10,0,0"/>
                        <TextBlock Text="Будьте в курсі подій та
нових аукціонів." FontSize="12"/>
                    </StackPanel>
                </StackPanel>
            </Border>
        </Grid>
    </Viewbox>
</Window>
```