Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для пошуку серверів популярної комп'ютерної гри
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-4

Максим КАЗАНСЬКИЙ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник проф. каф. ПІ Володимир БОНДАРЄВ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Казанському Максиму Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для пошуку серверів популярної комп'ютерної гри
затверджена наказом по університету від 19 травня 2025р. № 397 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 18.06.2025
3. Вихідні дані до роботи Розробка програмної системи, що забезпечує управління збірками модифікацій, призначене для користувачів гри Minecraft.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2025	*виконано*
2	Створення специфікації ПЗ	22.05.2025	*виконано*
3	Проєктування ПЗ	23.05.2025	*виконано*
4	Розробка ПЗ	28.05.2025	*виконано*
5	Тестування ПЗ	07.05.2025	*виконано*
6	Оформлення пояснювальної записки	09.06.2025	*виконано*
7	Підготовка презентації та доповіді	13.06.2025	*виконано*
8	Попередній захист	15.06.2025	*виконано*
9	Нормоконтроль, рецензування	16.06.2025	*виконано*
10	Здача роботи в електронний архів	17.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	18.06.2025	*виконано*

Дата видачі завдання «25 » « квітня » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. каф. ПІ Володимир БОНДАРЄВ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 53 стор., 9 рис., 13 джерел.

АКАУНТ, ІНТЕРФЕЙС, КОРИСТУВАЧ, ЛАУНЧЕР, МОДИФІКАЦІЯ, СИСТЕМА, ЗБІРКА.

У роботі представлено програмну систему для керування збірками модифікацій гри Minecraft. Розроблено серверну частину для обробки запитів і зберігання даних, вебінтерфейс для керування акаунтом користувача та настільний лаунчер для завантаження й запуску збірок. Система підтримує створення та редагування збірок, автентифікацію користувачів, базові адміністративні функції, а також взаємодію з репозиторіями модифікацій. Архітектура розроблена з урахуванням модульності та розширюваності. Запропоноване рішення дозволяє спростити процес підготовки модифікованих ігрових середовищ для кінцевого користувача.

ABSTRACT

ACCOUNT, INTERFACE, USER, LAUNCHER, MODIFICATION, SYSTEM, PACK.

This paper presents a software system for managing Minecraft modpacks. The developed system includes a backend component for request processing and data storage, a web interface for user account management, and a desktop launcher for downloading and launching modpacks. The system supports the creation and editing of modpacks, user authentication, basic administrative functions, and interaction with mod repositories. The architecture was designed with modularity and scalability in mind. The proposed solution simplifies the preparation of modified game environments for end users.

ЗМІСТ

ВСТУП

Темою кваліфікаційної роботи є програмна система для пошуку серверів популярної комп’ютерної гри. В сучасному світі ринок дистрибуції комп’ютерних ігор є надзвичайно широким, зокрема й в українському регіоні. Проте навіть попри розмаїття пропозицій розробники не завжди здатні задовольнити всі потреби кінцевого споживача. У таких випадках на перший план виходить діяльність учасників ігрової спільноти, які за допомогою сторонніх засобів здійснюють модифікацію гри – змінюючи або розширюючи її функціональні можливості, механіки чи ігровий вміст. У нашому випадку програмна система розробляється з орієнтацією на гру Minecraft, яка слугує об’єктом її прикладного застосування.

У багатьох випадках користувачі прагнуть одночасно використовувати декілька, а іноді й велику кількість таких модифікацій. Проте не всі з них сумісні між собою, що призводить до конфліктів та помилок під час гри. Саме поєднання кількох модифікацій в одну структуровану одиницю отримало назву "збірка". Крім того, гостро стоїть питання синхронізації користувацького ігрового досвіду в умовах мережевої гри, де гравці мають спільно використовувати однаковий набір модифікацій для взаємодії у спільному віртуальному середовищі. Водночас більшість ігор, зокрема й зазначена в темі, не передбачають вбудованих інструментів для гнучкого керування модами або їх групування, що вимагає від гравця додаткових технічних навичок та знань у сфері файлової структури гри, сумісності модифікацій і мережевої взаємодії.

Метою програмної системи є створення зручного засобу для пошуку спільних збірок модифікацій та відповідних сумісних серверів для популярної комп’ютерної гри, що не має вбудованого функціоналу для реалізації подібної задачі. За рахунок компетентного формування та тестування збірок з модифікаціями усуваються потенційні конфлікти між окремими модами, що дозволяє покращити ігровий досвід користувача та спростити процес налаштування середовища для спільної гри.

Завданням роботи є створення повноцінної клієнт-серверної архітектури, яка включає веб-сайт для взаємодії з обліковим записом користувача та настільний клієнтський додаток для завантаження, встановлення та запуску збірок. Для досягнення поставленої мети необхідно виконати наступні кроки:

- провести аналіз наявних рішень (TLauncher, CurseForge, ModRinth, FTB App);
- розробити архітектуру системи, що включає серверну, клієнтську (веб) та настільний додаток;
- реалізувати зручний та інтуїтивно зрозумілий інтерфейс користувача;
- забезпечити стабільну роботу функціоналу обрання, встановлення та запуску збірок;
- провести тестування системи в умовах, наближених до реальних сценаріїв використання.

Галуззю застосування розробленого продукту є сегмент індустрії комп’ютерних ігор, пов’язаний з використанням модифікацій для розширення стандартного функціоналу гри. Потенційними користувачами є як окремі гравці, що бажають грати на публічних серверах із попередньо узгодженими збірками, так і невеликі групи осіб або спільноти, які мають намір організувати приватну ігрову сесію в уніфікованому середовищі. Перспективи розвитку проєкту також дозволяють розглядати його як платформу для поширення власних збірок, а в окремих випадках – як допоміжний освітній інструмент у межах проєктного або ігрового навчання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Модифікації ігрового контенту, або скорочено «моди», з’явилися ще у 1980-х роках, коли виникли перші персональні комп’ютери та відповідні ігри. Однак через складність процесу внесення змін та обмежений доступ до комп’ютерної техніки, така практика довгий час залишалася малопоширеною. Поштовх до розвитку модифікацій надала гра Doom, випущена на початку 1990-х студією id Software. Попри те, що можливість внесення змін не передбачалась офіційно, відкритість структури гри дозволила спільноті створювати й обмінюватися новими рівнями, графікою та елементами геймплею.

Подальший якісний стрибок у популярності модифікацій відбувся з виходом Half-Life від Valve. Гнучка архітектура гри дала змогу створити повноцінні модифікації з новими механіками, серед яких – один із найвідоміших прикладів: Counter-Strike. Спочатку це був аматорський проєкт, створений ентузіастами, але згодом переріс у самостійну комерційну гру. Подібний підхід дозволив компанії Valve залучити до своїх продуктів численних незалежних розробників, в результаті чого з’явилися такі відомі ігри, як Team Fortress та Dota 2, що починались як неофіційні модифікації [1, 2].

На сьогоднішній день модифікації стали органічною частиною ігрової індустрії [3]. Більшість великих ігрових студій або відкрито підтримують їх створення, або не перешкоджають користувачам змінювати вміст гри, розуміючи, що це сприяє довготривалому інтересу до продукту. Моди й ігри утворюють симбіотичні відносини: популярність гри стимулює створення модифікацій, а цікаві моди, у свою чергу, підтримують або навіть збільшують інтерес до самої гри.

Одним з найпоказовіших прикладів такої взаємодії є гра Minecraft. Це проєкт у жанрі «пісочниці», що був створений незалежним розробником із початково обмеженим функціоналом, але з широкими можливостями для творчості користувача. Завдяки відкритій структурі гри та активній модифікаційній спільноті, Minecraft отримав велику популярність ще до того, як у

2014 році права на нього придбала компанія Microsoft. Успішність гри значною мірою забезпечена спільнотою, яка розробляє й поширює модифікації, що розширюють функціональність та змінюють ігровий досвід. Це підкреслює актуальність теми роботи [4].

Попри це, офіційна підтримка модифікацій з боку розробника залишається на дуже обмеженому рівні. Весь процес встановлення, налаштування та управління модами був і залишається справою користувачів і незалежних спільнот. Саме вони створюють сторонні інструменти, які дозволяють запускати змінений ігровий клієнт, але їх використання потребує певного рівня технічної грамотності. В результаті, для незначної частини гравців цей процес лишається доступним, тоді як більшість стикається з труднощами ще на етапі ознайомлення з базовими інструментами.

Найбільш поширеними категоріями модифікацій є: графічні (які змінюють візуальну складову гри), поведінкові (впливають на ігрові механіки та баланс), та технічні (додають нові об’єкти або системи зі складною логікою роботи). Для забезпечення сумісності модів та їх запуску були створені спеціальні «ядра» – програмні компоненти, що дозволяють змінити спосіб завантаження гри. Найбільш відомими серед них є Forge [5] та Fabric [6], які використовуються для різних версій Minecraft і часто мають несумісні між собою моди. Користувачам доводиться самостійно шукати, встановлювати й налаштовувати відповідні версії ядра, модифікацій та клієнта гри, що потребує додаткових зусиль і знань [7].

Окрему складність становить запуск багатокористувацьких сесій. Гравці, які хочуть зіграти разом із друзями, повинні мати доступ до відповідного ігрового сервера, налаштованого з урахуванням вибраних модифікацій. Це потребує або технічної компетентності, або використання сторонніх сервісів, що не завжди є безпечними чи зручними. Для пересічного користувача подібні бар’єри можуть стати причиною відмови від використання модів або багатокористувацької гри взагалі.

На ринку існує кілька популярних інструментів і лаунчерів для роботи з модифікаціями. Наприклад, TLauncher має інтегровану підтримку модифікацій,

але характеризується обмеженим функціоналом у плані керування збірками, а також мав репутаційні проблеми, пов’язані з використанням неофіційного коду. FTB App орієнтований на англомовну аудиторію та великі готові збірки, але часто не надає готових рішень для організації багатокористувацької гри. CurseForge та Modrinth мають власні вебплатформи з великою кількістю модифікацій, проте їх вміст формується виключно користувачами, що ускладнює навігацію, перевірку сумісності та запуск модифікованої версії гри.

Таким чином, ринок інструментів для взаємодії з модифікаціями Minecraft лишається фрагментованим, із низкою бар’єрів для кінцевого користувача. Це створює передумови для розробки нових, більш дружніх до користувача рішень [8].

1.2 Виявлення та вирішення проблем

Однією з головних проблем, яка постає перед користувачами, є питання сумісності модифікацій. Через відсутність єдиного офіційного підходу до їхньої інтеграції в гру, спільнота розробників модифікацій формувалася довільно, із власними підходами та стандартами. На сьогодні існує кілька незалежних ядер (наприклад, Forge, Fabric), які не забезпечують повної взаємної підтримки. Багато модифікацій сумісні лише з одним конкретним ядром, що обмежує їх комбінування у межах однієї збірки. Додатково, платформи розповсюдження накладають власні вимоги – наприклад, Modrinth приймає лише модифікації з відкритим вихідним кодом. Це підвищує безпеку користувачів, але обмежує доступ до деяких популярних модифікацій. Таким чином, існує потреба у структуризації доступних модифікацій та прозорій навігації між ядрами, платформами й версіями гри.

Ще однією важливою проблемою є висока складність створення власних збірок. Користувач має не лише сформувати концепцію збірки та володіти знаннями про доступні модифікації, але й витратити значний час на її тестування. Виникнення конфліктів між модифікаціями (наприклад, через дублювання функціональності або несумісні API) вимагає компетенцій, що не є типовими для

звичайного гравця. Тому доцільно впровадити процедуру попередньої валідації та тестування збірок, аби зменшити ймовірність конфліктів до мінімуму.

Важливим аспектом є синхронізація ігрового середовища між користувачами в багатокористувацьких сесіях. Навіть незначна розбіжність у версіях модифікацій між клієнтом та сервером може призвести до помилок запуску або некоректної роботи гри. Це особливо критично при великій кількості модифікацій, коли контроль за сумісністю ускладнюється. Вирішенням цієї проблеми є надання користувачам попередньо зібраних та узгоджених модифікаційних збірок, а також забезпечення доступу до відповідних версій ігрових серверів.

Ще один аспект – персоналізація користувача та зберігання його даних. Збереження збірок, створених або встановлених користувачем, у межах його профілю дозволяє легко синхронізувати прогрес і конфігурації між пристроями. Додатково, це відкриває можливості для розвитку соціальної взаємодії, як-от обмін збірками між друзями або спільна гра з використанням однакових налаштувань.

Нарешті, слід зазначити потенціал для реалізації творчих амбіцій спільноти. Надання користувачам платформи для публікації власних збірок з можливістю отримання визнання або винагороди за популярність може мотивувати до створення якісного контенту. Такий підхід є взаємовигідним: користувач отримує інструмент самореалізації, а система – постійне оновлення актуального контенту.

1.3 Постановка задачі

З огляду на виявлені проблеми, метою є створення програмної системи, що дозволяє користувачам зручно знаходити, тестувати та встановлювати збірки модифікацій Minecraft, які мають попередньо узгоджену сумісність із сервером. Реалізація такої системи дає змогу суттєво знизити технічні та часові бар’єри для гравців, що сприятиме залученню ширшої аудиторії.

Програмна система має включати два основні компоненти: веб-інтерфейс та настільний клієнт.

Веб-інтерфейс є точкою входу для нових користувачів і забезпечує доступ до базової функціональності: реєстрація облікового запису, перегляд доступних збірок, ознайомлення з їх описами, популярністю, рейтингами та інша взаємодія з контентом. Крім того, саме через веб-інтерфейс користувач завантажує настільний клієнт.

Настільний клієнт виконує функції завантаження, інсталяції та запуску збірок, а також автоматичної синхронізації конфігурацій відповідно до обраної збірки. Такий поділ функціональності дозволяє ефективно розділити задачі управління контентом і безпосередньої взаємодії з ігровим середовищем.

Важливою функціональністю системи є можливість додавання користувацьких збірок. Це відкриває потенціал для залучення активної спільноти розробників та ентузіастів, які можуть ділитися власними рішеннями. Крім того, функція публічного поширення збірок дозволяє охопити тих користувачів, які раніше не мали наміру встановлювати систему, проте зацікавилися певною конкретною збіркою, рекомендованою іншими.

Для зниження вхідного бар’єру передбачено імпорт збірок із зовнішніх платформ, таких як CurseForge або Modrinth. Це дозволить користувачам перенести вже сформовані конфігурації у нове середовище без потреби ручного переналаштування, що є особливо важливим для досвідчених гравців.

Таким чином, запропонована програмна система не лише вирішує наявні технічні проблеми, а й створює основу для формування спільноти користувачів та розробників навколо спільної екосистеми.

1.4 Категорії користувачів використання системи

Враховуючи широку аудиторію гравців Minecraft, систему доцільно проєктувати з урахуванням типових сценаріїв її використання. Це дозволяє краще зрозуміти потреби майбутніх користувачів, а також закласти відповідні вимоги до функціональності.

Користувач-початківець – це гравець, який не має досвіду самостійного встановлення модифікацій або вже стикався з труднощами на цьому шляху. Його

очікування – простота у встановленні та гарантія, що збірка запуститься без помилок. Для такого користувача є важливими доступ до перевірених збірок, можливість ознайомлення з описом, скріншотами, відгуками, простий спосіб інсталяції одним натисканням та автоматичне встановлення сумісної серверної частини або принаймні інструкція.

Досвідчений гравець – це категорія зазвичай має досвід роботи з модифікаціями та цікавиться створенням власних збірок. Їх основними запитами є можливість зібрати власну конфігурацію з уже доступних модифікацій, валідація на конфлікти та сумісність, збереження та поширення збірок серед друзів, накопичення репутації або статистики використання його збірки.

Адміністратор приватного сервера – має за мету уніфікувати клієнтське середовище всіх гравців, що приєднуються до сервера. Для них важливими аспектами є прив’язка серверу до конкретної збірки, доступ до завантаження серверної версії у кілька натиснень й перевірка відповідності версій клієнтів і сервера та можливість автоматизованого поширення збірки серед нових гравців.

Користувач, що змінює пристрої – у межах одного акаунта може змінювати пристрій (наприклад, між ПК удома та ноутбуком в іншому місці). Для таких випадків важливою є синхронізація переліку встановлених збірок, можливість швидкого доступу до останніх версій у будь-який час, єдина точка входу (акаунт) для продовження гри без повторення процесу налаштування.

Гравці соціально-орієнтованих спільнот – гравці, які хочуть грати в одній модифікованій системі зі своїми друзями або у межах спільнот в соціальних мережах. Для них ключовим є зручне поширення збірки (наприклад, публічне посилання), відстеження змін у збірці, якщо автор її оновлює, варіативність: публічна або приватна збірка, контроль доступу.

Таким чином, чітке усвідомлення цільових груп користувачів і моделі їх поведінки дозволяє ще на етапі формулювання задачі передбачити очікувану функціональність системи, а згодом – спростити її архітектуру за рахунок фокусування на найтиповіших сценаріях.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Програма система, що розроблюється є комплексним інструментом для взаємодії з модифікованими збірками гри Minecraft. Вона поєднує функції браузера збірок, інсталятора та центра керування модифікаціями, орієнтованого як на новачків, так і на досвідчених користувачів. Основною метою системи є спрощення процесу пошуку, встановлення, тестування та поширення збірок модифікацій, що сумісні з конкретними версіями клієнтів і серверів Minecraft.

Система має двокомпонентну архітектуру.

Веб-застосунок, що слугує точкою входу для користувача. Він дозволяє створити обліковий запис, переглядати доступні збірки, фільтрувати їх за характеристиками (версія Minecraft, кількість модів, рейтинг), переглядати скріншоти, опис і відгуки. Також через веб-інтерфейс реалізується завантаження настільного застосунку.

Настільний застосунок, що відповідає за встановлення збірок, запуск клієнта гри, тестування сумісності модифікацій та інтеграцію з наявним клієнтом Minecraft. У перспективі – підтримка встановлення серверної частини та валідації конфігурації для багатокористувацької гри.

Як згадувалося раніше, цільова аудиторія охоплює:

– новачків, які хочуть грати з модифікаціями без складного налаштування;

– гравців зі середнім досвідом, які бажають легко керувати збірками;

– адміністраторів приватних серверів, які хочуть забезпечити синхронізовану ігрову середу;

– ентузіастів, зацікавлених у створенні й публікації власних збірок;

– користувачів, що грають на кількох пристроях.

Програмна система не є аналогом звичного лаунчера, а радше розширенням екосистеми Minecraft, зосередженим на модифікованому геймплеї та зниженні технічного бар'єру для роботи з модами.

Для забезпечення очікуваної поведінки системи та покриття типових сценаріїв її використання, до програмної системи висуваються наступні функціональні вимоги. Вони згруповані за компонентами системи та відображають ключові можливості, необхідні для користувачів.

Розглянемо вимоги до веб-застосунку.

Первісним та очевидним є реєстрація та автентифікація користувачів: можливість створення облікового запису, вхід та вихід з системи, відновлення пароля.

Функціонально основним є перегляд каталогу збірок, їх фільтрація за версією гри, кількістю модифікацій, датою оновлення чи рейтингом. Зручним вдасться сортування за популярністю, новизною тощо. Та головним для вибору є доступ до детальної інформації про збірку (назва, опис, скріншоти, список модів, автор, рейтинг, кількість завантажень).

Для запуску самої гри з обраними модифікаціями, логічним буде завантаження настільного застосунку для подальшої роботи зі збірками.

Не в останню чергу, наявність функціонал з керування профілем користувача: перегляд і редагування особистих даних, список завантажених/використаних збірок, опубліковані збірки, а також публікація власних збірок, куди входить додавання опису, скріншотів, версій модифікацій, контроль доступу (публічна/приватна) та можливість оновлення опублікованої збірки.

Розглядаючи вимоги до самого настільного застосунку, варто зазначити можливість обов’язкову синхронізацію з обліковим записом, що передбачає автоматичне завантаження доступних користувачеві збірок чи синхронізацію налаштувань.

Процес встановлення збірки являє собою послідовний процес автоматичного завантаження модифікацій та допоміжних файлів, перевірки сумісності клієнта гри з обраною збіркою та можливість одночасного збереження декількох профілів (збірок).

Це дозволяє відразу виконати користувачеві запуск гри з вибраною збіркою, без потреби ручного налаштування. Якщо ж наданий каталог не задовільнить його, то передбачається опція імпорту збірок з інших платформ або з локального сховища.

Для полегшення роботи розробників модифікацій, можуть використовуватися засоби для валідації модифікацій, що сприяє виявленню конфліктів. Оскільки певні конфлікти можуть бути доволі поширеними, то для них можуть бути дійсними автоматичні рекомендації щодо їх усунення.

Для повного покриття вимог, ймовірна опція завантаження серверної частини. Це може полегшити отримання користувачем відповідної версії сервера для обраної збірки. До цього можуть бути надані й рекомендації щодо запуску власного сервера.

Серед загальних функціональні вимог, найбільше цікавлять наступні аспекти.

Механізм оновлення, що має дозволити підтримку оновлення збірок без потреби повного перевстановлення чи повідомлення користувача про, власне, доступні оновлення.

Взаємодія між користувачами, яка може полягати у можливості ділитися збірками через посилання та перегляд статистики використання цієї збірки, як от рейтинги чи коментарі для прийняття певного рішення.

Перспективним вбачається багатоплатформеність, що дасть підтримку для роботи на основних ОС крім Windows, як от Linux чи macOS.

З огляду на розподілену структуру системи, що складається з веб-інтерфейсу, серверної частини та настільного клієнта, ключовою вимогою до архітектури є чіткий поділ відповідальностей між компонентами та стандартизовані програмні інтерфейси для взаємодії.

Для забезпечення стабільної та масштабованої взаємодії між компонентами системи запроваджується централізований REST API, реалізований на ASP.NET. Цей API є єдиною точкою доступу як для веб-інтерфейсу на Razor Pages [9], так і для настільного додатку, розробленого з використанням Avalonia UI [10]. Такий

підхід дозволяє дотримуватися принципів єдиного джерела істини (Single Source of Truth) щодо даних користувачів, збірок та модифікацій.

Інтерфейс API має відповідати наступним вимогам:

- передача даних у форматі JSON;
- автентифікація через JWT-токени для захисту приватних даних користувачів;
- підтримка версіювання API для збереження сумісності при оновленнях;
- обмеження доступу за ролями (напр. пересічний користувач, автор збірки, адміністратор).

Для зменшення навантаження на сервер передбачено кешування частини запитів, які рідко змінюються (наприклад, описів публічних збірок). Усі запити мають записуватися в журнал для моніторингу та відстеження збоїв.

Архітектура системи будується за принципами клієнт-серверної моделі з можливістю розширення у напрямку мікросервісної структури у майбутньому. У фокусі – масштабованість, підтримка розділеного розгортання компонентів (наприклад, розміщення серверної та клієнтської частини на різних хостах) і забезпечення безперервної доставки оновлень (CI/CD).

З боку клієнтів особлива увага приділяється стійкості до втрати з’єднання та автоматичному повтору критичних запитів. Це актуально особливо для настільного додатку, який має працювати на різних пристроях з непередбачуваними умовами доступу до мережі.

Інтерфейс користувача виконує роль основної точки взаємодії користувача з програмною системою, тож до нього висувається низка вимог щодо зручності, доступності та послідовності. З урахуванням мультиплатформеності проєкту, інтерфейси повинні бути узгоджені стилістично й функціонально, при цьому адаптовані до особливостей кожного середовища.

Серед загальних вимог, однією з ключових таких до інтерфейсу програмної системи є візуальна послідовність і єдність оформлення. Це передбачає використання спільного дизайну для веб-інтерфейсу та настільного застосунку: однакова кольорова гама, піктограми, логіка розміщення елементів і навігації.

Такий підхід дозволяє користувачеві швидко орієнтуватися між різними компонентами системи та мінімізує когнітивне навантаження.

Функціональність системи має бути доступною максимально швидко, без зайвих кроків – користувач повинен мати змогу виконати основні дії (наприклад, запустити збірку або створити нову) буквально в кілька натискань. Це стосується як навігації, так і структури розділів у застосунку.

Система повинна підтримувати повну локалізацію, щонайменше українською та англійською мовами. Це забезпечує доступність для ширшої аудиторії та відповідає очікуванням користувачів з різних регіонів.

Адаптивність інтерфейсу також є критичною: як веб-інтерфейс, так і настільний застосунок повинні коректно відображатись на пристроях із різними розмірами екранів. Особливу увагу слід приділити коректній роботі на нетипових роздільностях, що часто зустрічаються на ноутбуках чи моніторах з високою щільністю пікселів.

Інтерфейс повинен чітко інформувати користувача про помилки, а також надавати підказки або інструкції у доступній формі, без технічної термінології, яка потребує спеціальних знань. Простота й зрозумілість формулювань допомагають уникнути фрустрації та сприяють успішній взаємодії з системою.

Веб-інтерфейс (Razor Pages) орієнтований на швидку та інтуїтивну взаємодію з основним функціоналом. З самого початку користувач має доступ до простого механізму реєстрації та входу в систему, без зайвих кроків або технічних складнощів. Після входу забезпечується легкий доступ до перегляду наявних збірок, включаючи їх візуальні матеріали (скріншоти), опис, перелік модифікацій та оцінки інших користувачів.

Ключовим елементом веб-інтерфейсу є система фільтрації та сортування. Вона дозволяє відібрати збірки за різними ознаками, такими як дата додавання, рівень популярності, кількість модифікацій або наявність певних тегів. Завдяки цьому користувач має змогу швидко знайти потрібний варіант без зайвого перегляду всього каталогу.

У рамках профілю користувача реалізовано перегляд особистих даних, а також перелік встановлених, уподобаних та створених збірок. Це дозволяє підтримувати контроль над власною активністю в системі, а також швидко повертатися до вже використаних матеріалів або оновлювати їх при потребі.

Настільний застосунок, створений на основі Avalonia UI, надає змогу встановлювати збірки без зайвих зусиль – усього за один клік. Програма самостійно перевіряє наявність усіх потрібних файлів, виявляє конфлікти між модифікаціями та встановлює відповідні залежності. Користувач бачить стан збірки у режимі реального часу – її завантаження, наявність оновлень або можливі помилки.

Важливою функцією є підтримка створення власних збірок. Застосунок дає змогу додавати обрані модифікації, автоматично перевіряти їхню сумісність між собою та з клієнтом гри, зберігати результат і, за бажання, оприлюднювати його для інших користувачів. Таким чином, програма одночасно виконує функції інсталятора та інструмента створення контенту.

Окремою перевагою є інтеграція з серверами – користувач може пов’язати певну збірку з конкретним сервером, що забезпечує спільну гру без необхідності ручного налаштування. Також передбачено запуск у локальному режимі, без підключення до мережі.

Серед додаткових вимог – підтримка темного оформлення інтерфейсу, що знижує навантаження на зір і є актуальною вимогою сучасного дизайну. Це стосується як веб-версії, так і настільного застосунку, де стилістика повинна бути єдиною.

Користувацький інтерфейс має повністю підтримувати навігацію з клавіатури, що особливо важливо для людей із обмеженими можливостями або тих, хто вважає за краще працювати без миші. У веб-версії також передбачена відповідність базовим стандартам доступності для інтерфейсів.

Усі форми введення даних повинні містити механізми перевірки в реальному часі. Це означає, що ще до надсилання форми система має сигналізувати про помилки, як-от неправильний формат електронної адреси чи

занадто короткий пароль. Такий підхід мінімізує кількість помилок на стороні користувача й підвищує зручність у використанні.

Нефункціональні вимоги визначають обмеження та характеристики програмної системи, що не залежать від конкретної реалізації функціоналу, але істотно впливають на користувацький досвід, масштабованість та підтримуваність системи.

У контексті надійності, система повинна бути здатна до безвідмовної роботи впродовж тривалого часу, зокрема при високому навантаженні (одночасні запити до бази даних, завантаження файлів). Веб-сервер та сервер авторизації мають бути ізольовані від частини, що відповідає за завантаження ресурсів, для зменшення збоїв у разі часткових відмов.

Масштабована архітектура повинна дозволяти збільшення кількості користувачів без необхідності значних змін у кодовій базі. У цьому контексті доречне застосування кешування для зменшення навантаження на сервер при частих запитах (наприклад, при перегляді популярних збірок).

Продуктивність системи має забезпечувати швидкий відгук інтерфейсів і короткий час очікування на основні дії. Вебзастосунок повинен реагувати на взаємодію користувача не довше ніж за 300 мс у типовому сценарії. Завантаження збірки через настільну програму не повинно перевищувати 10 секунд за умов звичайного з’єднання з мережею. Такі параметри дозволяють уникнути затримок, які можуть викликати роздратування або змусити користувача відмовитися від використання продукту.

Безпека користувача є пріоритетною на всіх рівнях. Усі облікові дані передаються виключно через захищене з’єднання (HTTPS), а автентифікація відбувається за допомогою токенів із обмеженим терміном дії (JWT), що унеможливлює їх безстрокове використання. Настільний застосунок працює в ізольованому середовищі та не має права змінювати налаштування поза власним каталогом, що виключає ризик несанкціонованого втручання в систему користувача.

Підтримуваність та розширюваність реалізовані завдяки модульній побудові системи. Компоненти проєкту слабо пов’язані між собою, що дає змогу легко оновлювати або замінювати окремі частини без впливу на всю систему. Уся конфігурація виконується через описові файли з коментарями, а програмний код супроводжується документацією. Система допускає додавання нових джерел модифікацій без необхідності повної переробки клієнтської логіки.

У процесі проєктування та реалізації системи необхідно враховувати низку обмежень, що впливають на вибір технологій, архітектури та підходів до розробки. Також для формування чітких вимог формулюються припущення, що встановлюють очікувані умови експлуатації.

Обмеження:

- система орієнтована лише на версію Java-видання Minecraft; підтримка інших видань (наприклад, Bedrock) не передбачається, через принципіальну різницю версій [11];
- не передбачається ручне встановлення модифікацій користувачем поза межами системи; користувач взаємодіє винятково через інтерфейс;
- настільний додаток працює лише на операційних системах Windows та Linux (x64); підтримка macOS – у перспективі;
- усі збірки повинні зберігатись централізовано, а доступ до них – здійснюватись через бекенд, що обмежує сторонні механізми розповсюдження;
- веб-інтерфейс не призначений для адміністративного керування системою – ці функції реалізуються окремо.

Припущення:

- користувач має стабільне інтернет-з’єднання, необхідне для авторизації, завантаження та синхронізації збірок;
- усі модифікації, що входять до збірок, мають відкриту або умовно відкриту ліцензію, що дозволяє їх вільне використання [12];
- користувач має мінімальні технічні навички для встановлення настільного додатку та запуску Minecraft;

- основний потік користувачів працює у межах одного облікового запису, що забезпечує персоналізацію досвіду;
- автори збірок діють добросовісно, публікуючи працездатні комбінації модів та не вставляючи стороннього або шкідливого коду.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Діаграма варіантів використання відображає основні сценарії взаємодії користувачів із системою керування збірками модифікацій. Вона ілюструє три ключові ролі: Гість, Гравець та Адміністратор. Кожна з цих ролей має певний набір доступних функцій (див. рис. 3.1).

Рисунок 3.1 – Загальна діаграма варіантів використання для програмної системи (рисунок виконано самостійно)

Гість може переглядати збірки, створити акаунт, завантажити лаунчер та пройти автентифікацію. Після входу користувач отримує розширений функціонал як Гравець – створення власних збірок, керування ними, редагування профілю, запуск збірок тощо.

Адміністратор, як Гравець із ширшими повноваженнями, додатково отримує можливість керування користувачами (зокрема скидання паролів) та модерації контенту (модифікацій та збірок).

Включено деякі системні дії, зокрема автоматичне надсилання сповіщень на пошту після певних користувацьких дій (наприклад, скидання та підтвердження зміни паролю), що демонструє взаємодію між користувачем та серверною логікою.

На рисунку 3.2 зображено діаграму сутностей системи, яка моделює структуру реляційної бази даних, що використовується серверною частиною для зберігання й обробки інформації. Діаграма охоплює основні сутності, їхні атрибути та зв’язки між ними.

Система підтримує зберігання користувацьких акаунтів, ролей, модифікацій, збірок та їх версій, а також реалізує механізми безпеки, як-от підтвердження електронної пошти й скидання пароля за допомогою токенів.

Особливу увагу приділено наступним аспектам.

Багатомовність: реалізована через окремі таблиці локалізацій для модів, збірок, серверів та тегів, що дозволяє гнучко відображати інтерфейс відповідною мовою.

Версії: моди та збірки мають окремі сутності для представлення версій (mod_versions, mod_pack_versions), що забезпечує можливість оновлення та зворотної сумісності.

Зв’язки типу багато-до-багатьох: реалізовані через допоміжні таблиці (наприклад, mod_version_mod_packs), які забезпечують можливість включення декількох модів до різних збірок.

Безпека: таблиці email_verification_tokens і password_reset_tokens дозволяють реалізувати стандартну функціональність підтвердження електронної пошти та скидання пароля.

Усі сутності побудовані з урахуванням вимог до цілісності, масштабованості та подальшого розвитку системи.

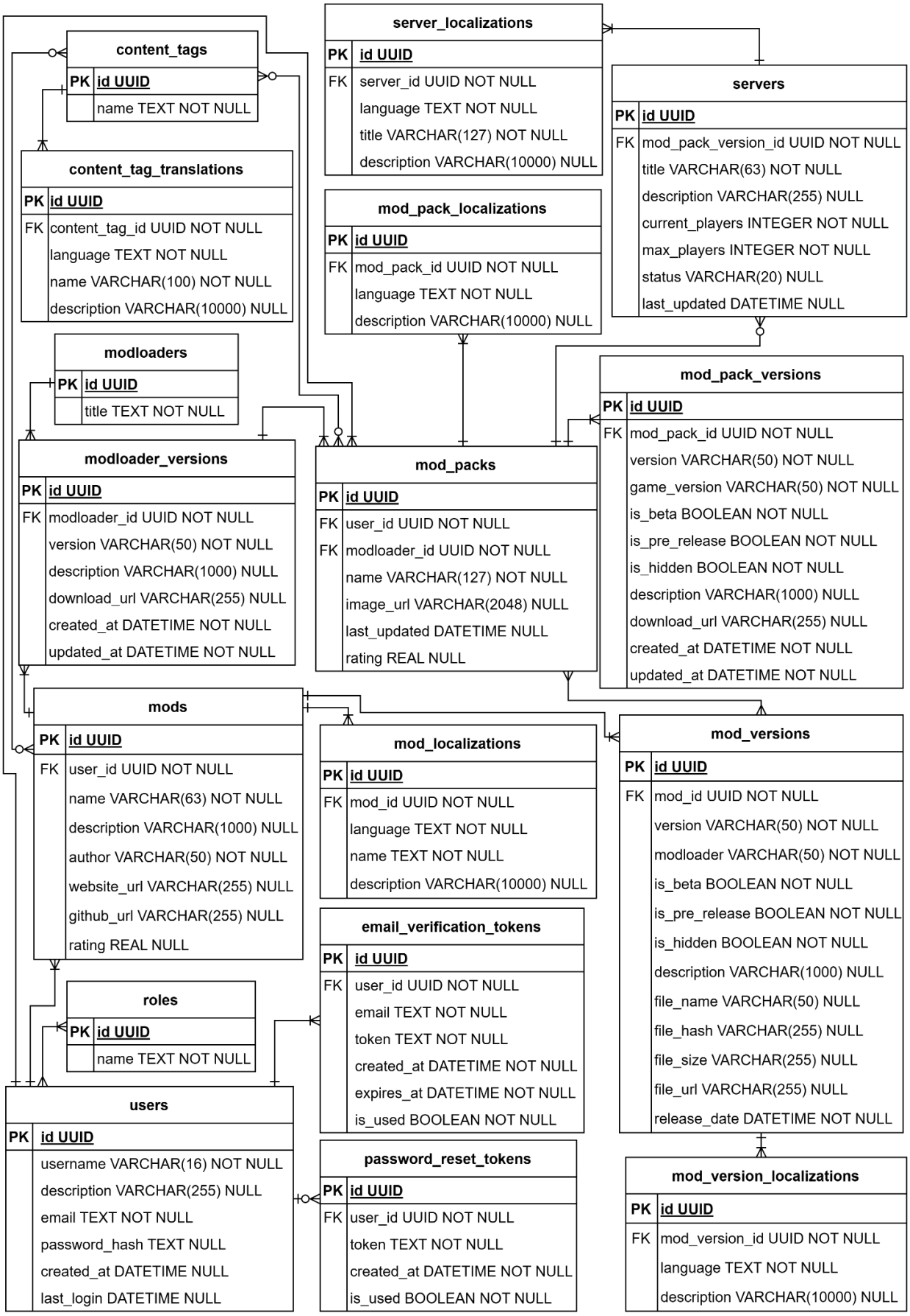

Рисунок 3.2 – Діаграма сутностей програмної системи (рисунок виконано самостійно)

На діаграмі послідовностей показано (див. рис. 3.3) почерговість дій, що відбуваються під час процесу авторизації користувача в системі. Учасниками процесу є:

- користувач – ініціює запит на вхід;
- клієнт – фронтенд застосунку (наприклад, вебсайт або лаунчер);
- сервер – обробляє запити та відповідає за перевірку автентичності;
- база даних – містить облікові записи користувачів.

Процес починається з того, що користувач вводить свій нікнейм та пароль. Клієнт надсилає ці дані на сервер. Далі виконуються такі кроки:

Після надсилання даних, сервер спочатку перевіряє наявність користувача з указаним нікнеймом. Якщо такий запис існує, система з’ясовує, чи підтверджено електронну адресу, пов’язану з цим обліковим записом. У разі позитивного результату виконується перевірка правильності пароля шляхом порівняння хешів. Якщо всі умови виконано, формується відповідь про успішну автентифікацію, яка містить, наприклад, сесію або JWT-токен. У випадку будь-якої помилки користувачеві повертається чітке повідомлення про причину – наприклад, «Користувача не знайдено», «Пошта не підтверджена» або «Невірний пароль».

Дана діаграма також демонструє базову обробку помилок та забезпечує основу для майбутнього розширення функціональності, зокрема:

- додавання двофакторної автентифікації;
- внесення в журнал спроб входу;
- лімітування спроб входу для захисту від підбору паролей.

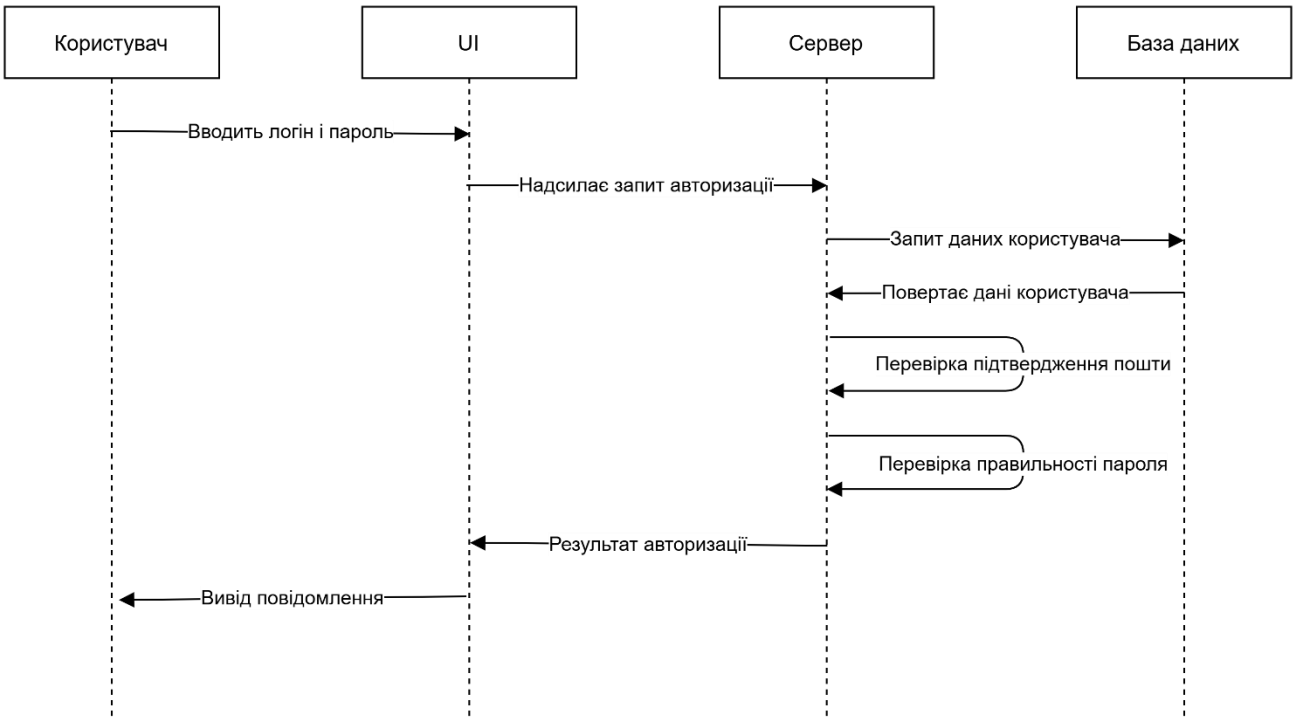

Рисунок 3.3 – Діаграма послідовностей, що відображає процес авторизації у системі (рисунок виконано самостійно)

3.2 Проєктування архітектури ПЗ

Розроблена система має не просту, але структуровану архітектуру, яка забезпечує ефективне виконання функцій, пов’язаних із переглядом, завантаженням, запуском та керуванням збірками модифікацій для гри Minecraft. При проєктуванні архітектури програмного забезпечення було обрано багаторівневу (n-tier) архітектуру, що дозволяє чітко розмежувати відповідальність між клієнтською частиною, серверною логікою, а також системою зберігання даних.

До складу системи входять наступні основні компоненти:

– веб-клієнт на базі Razor Pages, що надає інтерфейс для взаємодії користувача з акаунтом, перегляду модифікацій та збірок;
– серверна частина на основі ASP.NET Core, яка реалізує бізнес-логіку, обробляє HTTP-запити, керує базою даних та автентифікацією;

- десктопний додаток (лаунчер) розроблений за допомогою Avalonia UI, що надає функціонал завантаження та запуску збірок безпосередньо з ПК користувача;
- реляційна база даних, що зберігає інформацію про користувачів, ролі, модифікації, збірки та їх залежності;
- зовнішні сервіси для надсилання електронних листів (наприклад, підтвердження реєстрації або скидання пароля).

Серед компонентів проявляють взаємодію наступні.

Клієнтська частина. Веб-інтерфейс дозволяє реєструватися, входити в акаунт, переглядати публічну інформацію, створювати власні збірки тощо. Лаунчер отримує дані з сервера, дозволяє завантажувати обрану збірку модів, зберігає їх локально та запускає гру з відповідними параметрами.

Серверна частина. Реалізує REST API з використанням ASP.NET Core. Впроваджено базову автентифікацію, авторизацію за ролями, обробку помилок та відправлення системних повідомлень (через email). Структура контролерів і сервісів побудована з урахуванням принципів розділення відповідальностей (Separation of Concerns).

Сховище даних. Для взаємодії з базою даних використовується Entity Framework Core (EF Core). Структура зберігання реалізована у вигляді сутностей, що відповідають таблицям у БД (наприклад, User, UserRole, Mod, Modpack, ModVersion). Здійснено валідацію даних за допомогою атрибутів (Data Annotations), що гарантує правильність введеної інформації.

У проєкті застосовано кілька перевірених шаблонів проєктування, що забезпечують гнучкість та зрозумілу структуру коду. Зокрема, шаблон repository ізолює доступ до даних, dependency injection відповідає за впровадження залежностей, а DTO (об’єкти передавання даних) мінімізують обсяг інформації, що передається через API. Логіка Razor Pages реалізована відповідно до архітектурного підходу MVC, що спрощує підтримку та масштабування вебінтерфейсу.

На поточному етапі реалізації система вже підтримує хешування паролів перед збереженням у базу даних, перевірку електронної пошти під час реєстрації, можливість відновлення доступу до облікового запису через email-підтвердження, а також авторизацію з урахуванням ролей користувача. Це створює базову, але надійну основу для безпечної взаємодії з системою.

У планах щодо розширення функціоналу безпеки передбачено впровадження журналювання входів, підтримку двофакторної автентифікації та обмеження частоти запитів до API. Це дозволить захистити систему від спроб підбору паролів і забезпечити контроль за підозрілою активністю.

Побудована архітектура надає широкі можливості для подальшого масштабування. Передбачено винесення роботи з великими файлами (модифікації, збірки) в окремі сервіси, запровадження кешування для зменшення навантаження на базу даних, а також додавання нових ролей або функцій без суттєвих змін у загальній логіці системи.

3.3 Проєктування структури зберігання даних

Основою системи зберігання даних у розробленому програмному забезпеченні є реляційна база даних, структура якої відповідає вимогам до збереження облікових записів користувачів, модифікацій, збірок та супровідної інформації. Зв'язок між логікою програми та базою даних реалізовано з використанням технології Entity Framework Core, яка забезпечує об'єктно-реляційне відображення сутностей та взаємодію з базою через LINQ-запити.

У структурі системи виділяються ключові сутності, кожна з яких виконує визначену функцію. Користувач (User) зберігає інформацію про акаунт – нікнейм, пошту, хеш паролю та дати активності. Ролі (Role) визначають рівень доступу, а зв’язок між ними й користувачами реалізовано через проміжну таблицю. Модифікації (Mod) описують окремі елементи, що входять до збірок, а їхні версії (ModVersion) фіксують сумісність і технічні деталі. Збірка (Modpack) об’єднує набір модів і має власника, опис, дату створення та зображення, тоді як ModpackVersion дає змогу підтримувати багатоверсійність. Нарешті, таблиця

зв’язку між версіями модів і збірок (ModpackMod) дозволяє гнучко комбінувати вміст для різних конфігурацій.

Між сутностями реалізовані такі типи зв’язків:

– один-до-багатьох (наприклад, один користувач – багато збірок);
– багато-до-багатьох (наприклад, багато користувачів – багато ролей; багато модів – багато збірок).

Особливістю проєктування є чітке розділення об'єктів модифікацій і збірок за версіями, що дозволяє реалізувати систему оновлення та зворотної сумісності. Крім того, реалізовано обмеження на валідацію даних безпосередньо через атрибути моделей, що забезпечує узгодженість введеної інформації вже на рівні серверної логіки.

Для забезпечення цілісності даних передбачено використання первинних і зовнішніх ключів, обмеження цілісності, обов’язковість полів (через атрибути Required) та валідацію форматів (наприклад, електронної пошти або ніків).

Таке проєктування дає змогу забезпечити масштабованість і підтримуваність програмного забезпечення, дозволяючи легко додавати нові функції або змінювати логіку збереження даних без потреби в повному перегляді архітектури.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

У контексті цілей проєкту, програмні рішення базуються довкола вирішення наступних проблем:

– надійна реєстрація;
– надійна аутентифікація;
– надійна авторизація;
– відновлення втраченого доступу;
– керування вмістом (модифікаціями, збірками та їх описом).

Для вирішення проблеми надійної реєстрації, виконуються два етапи: внесення ідентифікаційних даних та їх верифікація. Ключовими верифікаційними даними є: логін (псевдонім), пароль та пошта користувача (див рис. 4.1).

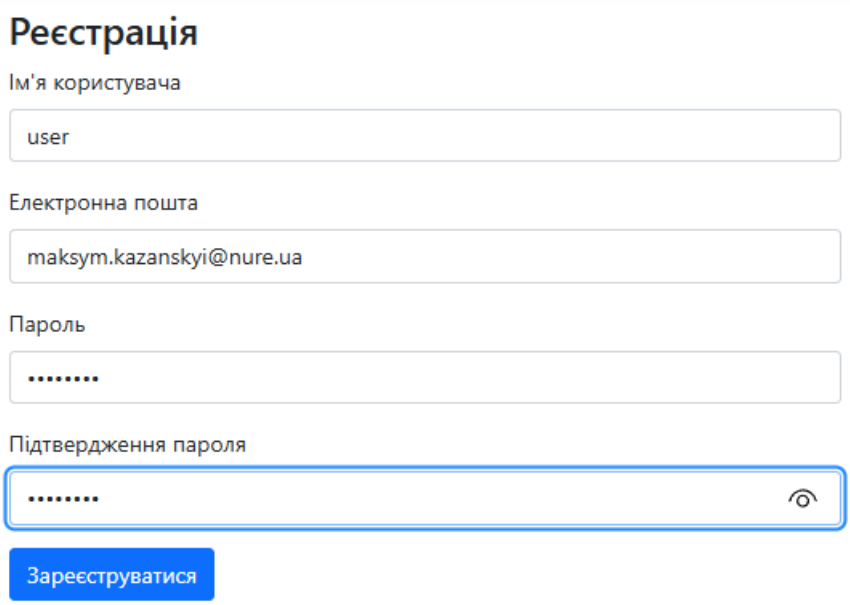

Рисунок 4.1 – Форма реєстрації в системі (рисунок виконано самостійно)

Якщо логін та пароль, будуть використовуватися у якості верифікаційних даних для аутентифікації, то пошта слугує окремою точкою верифікації наданих даних на предмет автентичності дій користувача, до якого прив’язана пошта зовнішнього сервісу (напр. Gmail чи Ukr.net). Це слугує формою перевірки дії користувача третьою стороною, що контролює можливість створення профілю користувача та редагування його критичних даних. У випадку реєстрації, дані про

користувача заносяться у систему, але не є дійсними для аутентифікації, поки дію реєстрації не буде підтверджено через пошту (див. рис. 5.2):

```
var emailVerificationToken = _context.EmailVerificationTokens
    .FirstOrDefault(t => t.UserId == user.Id && t.IsUsed == false);

if (emailVerificationToken != null)
{
    ModelState.AddModelError(string.Empty,
        "Email not confirmed. Please check your email.");
    return Page();
}
```

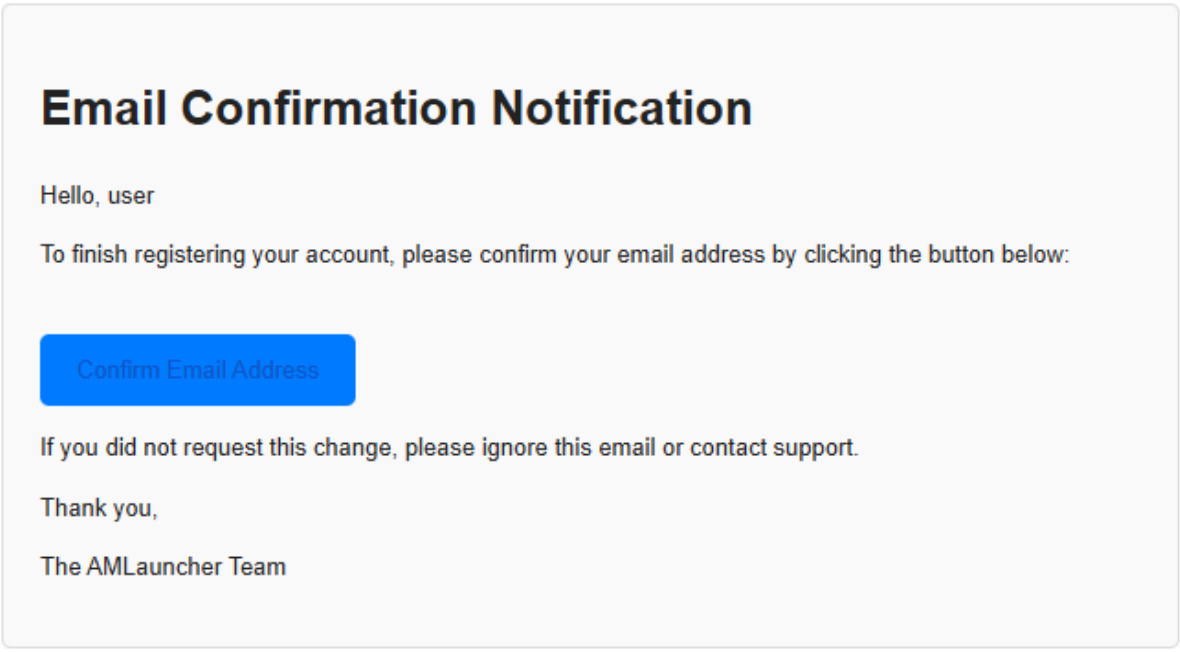

Рисунок 4.2 – Вміст повідомлення на пошті для завершення реєстрації (рисунок виконано самостійно)

Це забезпечує захист від використання пошти третіми особами, що можуть використати дані користувача проти його згоди для користування системою.

Досягнення надійної аутентифікації досягається шляхом збереження пароля профілю у вигляді хешу:

```
var passwordHasher = new PasswordHasher<User>();
var newUser = new User
{
    Username = Input.Username,
    Description = "",
    Email = Input.Email,
    PasswordHash = passwordHasher.HashPassword(
        null, Input.Password),
    CreatedAt = DateTime.UtcNow,
    LastLogin = DateTime.UtcNow
};
```

Відповідно, його валідація не виконується шляхом розшифровки, що унеможливлює використання цих даних у випадках витоку даних. Це відповідає сучасним критеріям безпеки програмних систем.

Для надійної авторизації використовується JWT:

```
builder.Services
    .AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = "Cookies";
        options.DefaultChallengeScheme = "Cookies";
        options.DefaultSignInScheme = "Cookies";
    })
    .AddCookie("Cookies", options =>
    {
        options.LoginPath = "/Account/Login";
        options.LogoutPath = "/Account/Logout";
        options.ExpireTimeSpan = TimeSpan.FromHours(1);
    })
    .AddJwtBearer(options =>
    {
        var jwtSettings = builder.Configuration.GetSection("Jwt");
        var key = jwtSettings["Key"];

        options.TokenValidationParameters = new()
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = jwtSettings["Issuer"],
            ValidAudience = jwtSettings["Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(key)),
        };
    });
```

Це набір даних, що надається користувачеві у випадку успішної аутентифікації. Організація у форматі JSON дозволяє легко опрацьовувати наявні там дані, а шифрування не дасть дані про користувачам третім особам. Використовуваний як ключ, JWT може бути скопійований, викрадений чи використовуватися третіми особами за відсутності власник. Для протидії цьому, він має строк спливу, після якого ключ стає недійсним. Єдиний спосіб подовжити сесію – це щоб користувач й надалі користувався системою.

Відновлення втраченого доступу стає у пригоді зазвичай, коли користувач забуває свій пароль. Оскільки, через раніше описану специфіку зберігання даних паролю, він не може бути відновленим, то існує окрема процедура відновлення

доступу через встановлення нового, як правило, тимчасового паролю, згенерований системою. У такому випадку користувач ініціює процедуру скидання та створення нового паролю (див. рис. 4.3, 4.4).

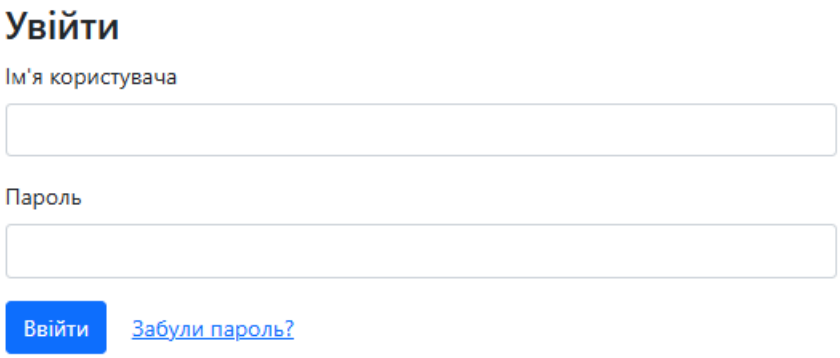

Рисунок 4.3 – Форма аутентифікації з опцією відновлення паролю (рисунок виконано самостійно)

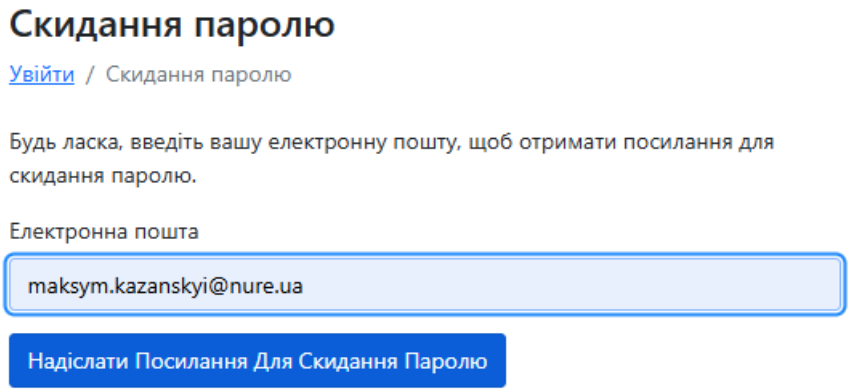

Рисунок 4.4 – Форма ініціації процедури скидання паролю (рисунок виконано самостійно)

Для підтвердження початку процедури скидання паролю, на вказану пошту користувача відправляється лист з підтвердженням на ініціювання процедури (див. рис. 4.5). Це запобігає несанкціонованій зміні паролю, коли третя особа знаючи електронну пошту захоче зашкодити користувачеві. Після підтвердження, користувач встановлює власний пароль (див. рис. 4.6) й після підтвердження йому пропонується ввести логін та новий пароль у вікні аутентифікації. У кабінеті користувач він піддається редагуванню, так щоб користувач зміг встановити власний. В особливих випадках, адміністрація може власноруч запустити процес

створення нового паролю, минаючи процедуру підтвердження цього рішення користувачем.

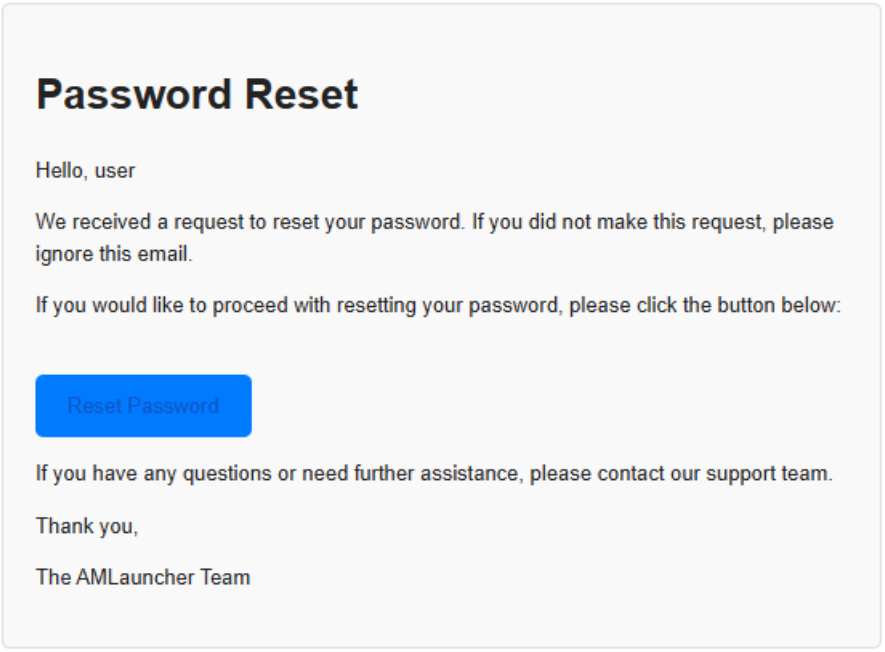

Рисунок 4.5 – Вміст повідомлення на пошті для скидання паролю (рисунок виконано самостійно)

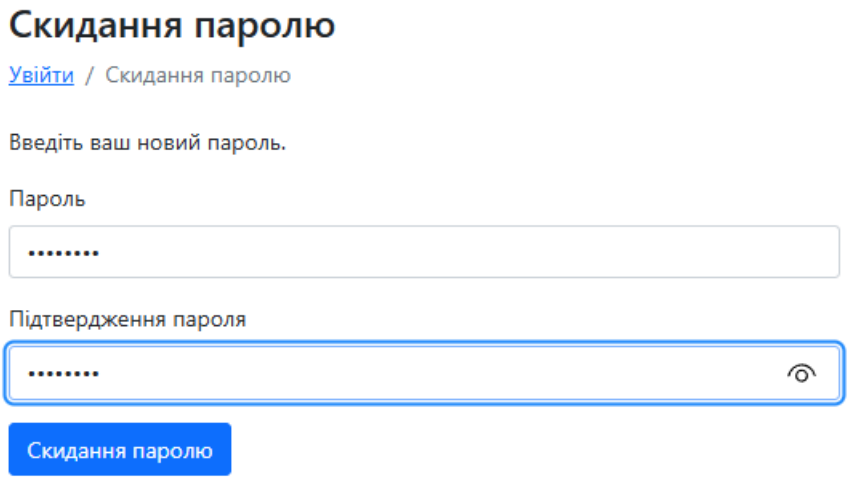

Рисунок 4.6 – Форма скидання паролю (рисунок виконано самостійно)

У повній мірі, редагувати контент може лише адміністрація. В принципі, концентрація уваги саме на можливостях адміністрації у плані редагування контенту є першочерговою, оскільки дозволяє позначити межі допустимих застосувань для користувачів. Наприклад, користувач може створити збірку й дати якийсь ярлик (tag), але перелік цих ярликів визначається адміністрацією, для

поліпшення зручності пошуку шляхом уніфікації. Складовими збірки, крім назви та опису, є версії цієї збірки. Складовими версії збірки є версії модифікацій. Версія модифікації є складовою модифікації самої по собі. Зв’язки між модифікаціями та збірками через їх версії забезпечують можливість еволюції збірки та модифікації окремо й дають підтримку кількох версії, як стабільних, так і експериментальних. Таким чином паралельно може експлуатуватися як звичайна версія, для нормальної користувальницької гри, так і тестова версія, для налагодження помилок, дослідження можливостей адміністраторами.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

При розробці програмної системи використовувалися різні підходи за різних вимог. В особливості, це було модульне, інтеграційне та системне тестування. Паралельно враховувалися функціональні та нефункціональні тестування разом з димовим та регресійним. Спільним є ручне тестування по принципу білого ящика, оскільки код перевірявся лише розробником.

Модульне тестування, в основному, проводилося в рамках впровадження нового модуля. Наприклад, тестування сервісу відправки електронних повідомлень. Використання саме ручного тестування для цієї частини коду зумовлюється її взаємодією з реальними системами, де їх автономна експлуатація вимагає додаткових ресурсів на обробку результатів.

Інтеграційне тестування вже являло собою зв’язок між, наприклад, зміною паролю у профілі користувача, та сповіщенням користувача на його електронну пошту про це.

Системне тестування, наприклад, включало весь шлях від реєстрації до зміни паролю. Це також являло собою форму регресійного тестування. Системне тестування виконувалося при додаванні значної кількості функціональності, та після проходження інтеграційного тестування з некритичними помилками.

У рамках функціональні тестувань перевірці також підлягали форми логіну та реєстрації, оскільки серед функціональних вимог були вимоги додаткової верифікації (підтвердження через пошту), довжини псевдоніму та паролю, форми відновлення паролю. Також система перегляду та редагування контенту (модифікації, збірки і т.д.). Система зміни мови для різних частин програмної системи.

Нефункціональні вимоги перевірялися зручністю навігації та орієнтації між елементами у системі (наприклад, при переході від головної сторінки до редагування опису користувача).

Постійними були димове та статичне тестування. Димове тестування полягало у перемиканні між елементами сторінки, шукаючи явні помилки при функціонуванні критичних систем (аутентифікація). Статичне тестування

виконувалося лише середовищем розробки, й шукало явні помилки у коді чи оформлені елементів керування.

ВИСНОВКИ

У результаті виконання роботи було розроблено програмне забезпечення для управління збірками модифікацій, призначене для користувачів гри Minecraft. Розроблена система включає серверну та клієнтську частину для взаємодії з акаунтом користувача й адміністративною панеллю, а також настільний лаунчер для завантаження та запуску збірок. Архітектура системи побудована відповідно до принципів розділення відповідальностей, що забезпечує масштабованість та підтримуваність проєкту.

Архітектура реалізована за принципом клієнт-серверної взаємодії. серверна частина побудована на основі ASP.NET із застосуванням Entity Framework для роботи з базою даних. Вебінтерфейс реалізований з використанням Razor Pages, що дозволяє створювати функціональні сторінки з можливістю авторизації, реєстрації та керування збірками. Настільний лаунчер створено з використанням фреймворку Avalonia.UI, що забезпечує кросплатформенність та сучасний вигляд інтерфейсу користувача.

У ході розробки було реалізовано ключові функціональні вимоги, а саме:

- система реєстрації, автентифікації та керування акаунтами користувачів;
- створення та запуск збірок модифікацій;
- базові адміністративні функції з керування вмістом;
- логіка взаємодії між частинами системи через API;
- проєктування модульної структури, що дозволяє в майбутньому розширювати функціональність без переробки всієї системи.

Для забезпечення ефективного процесу розробки використовувались сучасні підходи до проєктування програмного забезпечення, такі як:

- розробка на основі UML-діаграм (use case, sequence, entity);
- компонентне та багаторівневе проєктування;
- застосування принципів UI/UX-дизайну при створенні інтерфейсів;
- використання open-source рішень (Forge, Fabric) як частини модифікаційної екосистеми;

– підтримка розширюваності та потенціалу для масштабування.

Система пройшла тестування ключових сценаріїв, таких як створення акаунту, встановлення та запуск збірки. Було підтверджено стабільність взаємодії між компонентами, коректність відображення даних, і відповідність UI очікуванням цільової аудиторії. Незважаючи на наявність функціональних обмежень, структура проєкту закладає основу для подальшого вдосконалення.

У майбутньому можливо реалізувати:

– підтримку авторизації через сторонні сервіси (OAuth, Discord);
– розширення системи керування правами користувачів;
– покращення безпеки за рахунок впровадження двофакторної автентифікації;
– реалізацію рейтингової системи для збірок;
– автоматичну перевірку сумісності модів.

Розроблена система досягає поставленої мети – забезпечує зручний інструмент для пошуку, завантаження, запуску та керування збірками модифікацій. Побудова системи відповідно до сучасних вимог до архітектури програмного забезпечення, модульність реалізації та урахування потреб користувача свідчать про доцільність подальшого використання і розвитку створеного рішення.

Вихідний код, специфікації, відеодемонстрація та презентація описаної системи доступні на відповідному GitHub репозиторії [13].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. A Brief History of PC Game Modding? [Електронний ресурс] – URL: https://www.nvidia.com/en-us/geforce/news/history-of-pc-game-mods/ (дата звернення 06.05.2025)

2. Video game companies encourage 'modders' [Електронний ресурс] – URL: https://web.archive.org/web/20080506004712/http://www.hollywoodreporter.com/hr/search/article_display.jsp?vnu_content_id=1000484956 (дата звернення 06.05.2025)

3. Modders deserve recognition at The Game Awards, too [Електронний ресурс] – URL: https://www.polygon.com/game-awards-tga/483684/the-game-awards-tga-modder-nominations (дата звернення 06.05.2025)

4. Top Trends [Електронний ресурс] – URL: https://www.youtube.com/trends/articles/minecraft-trillion/ (дата звернення 06.05.2025)

5. Downloads for Minecraft Forge [Електронний ресурс] – URL: https://files.minecraftforge.net/net/minecraftforge/forge/ (дата звернення 06.05.2025)

6. Fabric Loader [Електронний ресурс] – URL: https://fabricmc.net/ (дата звернення 06.05.2025)

7. Бондарєв В. М., Черепанова Ю. Ю. Комп’ютерна симуляція термодинамічних процесів з навчальними цілями // Наукові праці Вінницького національного технічного університету. 2024. № 2. С. 6–16. DOI: https://doi.org/10.31649/2307-5376-2024-2-6-16

8. Бондарєв В., Черепанова Ю. Комп’ютерна симуляція небесної механіки з навчальними цілями // Інформаційні системи та технології : матеріали 13-ї Міжнар. наук.-техн. конф. (Харків, 26–28 листоп. 2024 р.). Харків : ХНУРЕ, 2024. URL: https://ist-conf-nure.com.ua

9. Introduction to Razor Pages in ASP.NET Core [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-9.0&tabs=visual-studio (дата звернення 06.05.2025)

10. Build apps for every device using .NET [Електронний ресурс] – URL: https://avaloniaui.net/ (дата звернення 06.05.2025)

11. The Difference between Java and Bedrock Editions [Електронний ресурс] – URL: https://www.minecraft.net/en-us/article/java-or-bedrock-edition (дата звернення 06.05.2025)

12. Beginner's Guide to Licensing your Mods [Електронний ресурс] – URL: https://blog.modrinth.com/p/licensing-guide (дата звернення 06.05.2025)

13. Репозиторій GitHub здобувача [Електронний ресурс] – URL: https://github.com/NureKazanskyiMaksym/2025_B_PI_PZPI-21-4_Kazanskyi_M_A (дата звернення 12.06.2025)