

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Кросплатформний лаунчер для ігор»

Виконав студент групи ПЗ-21-2ВЕТРОВ
Андрій Олександрович

Керівник роботи:
ТУРЕНКО Єлизавета Єгорівна

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного забезпечення

Тетяна ВАКАЛЮК

«14» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВЕТРОВ Андрій Олександрович**

Керівник роботи: **ТУРЕНКО Єлизавета Єгорівна**

Тема роботи: **«Кроссплатформний лаунчер для ігор»**,
затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: **процес управління ігровими проєктами та спільнотою за допомогою кроссплатформного лаунчера, розробленого на основі модульної клієнт-серверної архітектури.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єлизавета ТУРЕНКО	18.02.2025р.	18.02.2025р.
2	Єлизавета ТУРЕНКО	20.04.2025р.	20.04.2025р.
3	Єлизавета ТУРЕНКО	04.05.2025р.	04.05.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота на здобуття освітнього ступеня бакалавра присвячена розробці програмного продукту та представлена у формі прототипу кросплатформного ігрового лаунчера і супровідної пояснювальної записки. Пояснювальна записка обсягом 83 сторінки містить 26 рисунків та 8 таблиць, що детально ілюструють етапи дослідження.

Основною метою роботи є моделювання та розробка універсального ігрового лаунчера, що використовує сучасні вебтехнології для подолання проблеми фрагментації ігрового ринку, забезпечення високої продуктивності та глибокої інтеграції з різними ігровими платформами. У ході дослідження було виконано детальний аналіз ринку та існуючих аналогів, на основі якого спроектовано гнучку, масштабовану архітектуру та розроблено інтуїтивно зрозумілий користувацький інтерфейс. Ефективність прийнятих дизайнерських рішень була верифікована за допомогою предиктивного тестування юзабіліті з використанням інструменту Attention Insight. Для програмної реалізації проєкту було обґрунтовано та обрано технологічний стек, що поєднує мову Rust для серверної частини, яка забезпечує безпеку та швидкодію, і фреймворк Tauri у комбінації з Svelte для створення легкої та ефективної клієнтської частини.

Архітектура розгортання системи базується на використанні технологій контейнеризації Docker та вебсервера Nginx [3] для надійності та гнучкості. Кінцевим результатом роботи є комплексне програмне рішення, що надає потужний інструментарій як для гравців, так і для адміністраторів ігрових спільнот, пропонуючи дворівневу систему управління.

КЛЮЧОВІ СЛОВА: ЛАУНЧЕР, КРОСПЛАТФОРМНІСТЬ, TAURI, RUST, SVELTE, ІГРОВІ СЕРВІСИ, ВЕБТЕХНОЛОГІЇ, ПРОЄКТУВАННЯ ПЗ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Кросплатформний Лаунчер для ігор	Літ.	Арк.	Аркушів
Розроб.		Ветров А.В.						
Керівник		Туренко Є.Є					3	83
						ФІКТ Гр. ІПЗ-21-2		
Рецензент		Концидайло А.М						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The final qualification work for the bachelor's degree is dedicated to the development of a software product and is presented in the form of a prototype of a cross-platform game launcher and an accompanying explanatory note. The 83-page explanatory note contains 26 images and 8 tables illustrating the stages of the research in detail.

The main goal of the study is to model and develop a universal gaming launcher that uses modern web technologies to overcome the problem of gaming market fragmentation, ensure high performance and deep integration with various gaming platforms. The research included a detailed analysis of the market and existing analogues, which was used to design a flexible, scalable architecture and develop an intuitive user interface. The effectiveness of the design decisions was verified through predictive usability testing using the Attention Insight tool. For the programmatic implementation of the project, a technology stack was justified and chosen that combines the Rust language for the server side, which ensures security and performance, and the Tauri framework in combination with Svelte to create a light-weight and efficient client side.

The system's deployment architecture is based on Docker containerization technologies and the Nginx web server for reliability and flexibility. The end result is a comprehensive software solution that provides powerful tools for both players and administrators of gaming communities, offering a two-level management system.

KEYWORDS: LAUNCHER, CROSS-PLATFORM, TAURI, RUST, SVELTE, GAMING SERVICES, WEB TECHNOLOGIES, SOFTWARE DESIGN.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ КРОСПЛАТФОРМНОГО ЛАУНЧЕРА ДЛЯ ІГОР	9
1.1 Постановка задачі.....	9
1.2. Аналіз аналогів програмного продукту	11
1.3 Дослідження технологій реалізації	16
1.4 Розробка макету	17
Висновки до першого розділу.....	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ КРОСПЛАТФОРМНОГО ЛАУНЧЕРА	22
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	22
2.2. Визначення вимог до системи	27
2.3. Проєктування об'єктно-орієнтованої моделі та структури даних.....	32
2.4 Проєктування бази даних	35
2.5. Моделювання взаємодії та комунікацій	36
2.6. Вибір патернів проєктування та визначення компонентів системи	38
Висновки до другого розділу	44
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З КРОСПЛАТФОРМНИМ ЛАУНЧЕРОМ ДЛЯ ІГОР	46
3.1. Розробка	46
3.2. Інтерфейс та порядок роботи	51
3.3. Процес встановлення та налаштування параметрів платформи	54
3.4. Тестування програмного забезпечення.....	58
Висновки до третього розділу.....	63
ВИСНОВКИ	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	67
ДОДАТКИ	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACID – Атомарність, Узгодженість, Ізольованість, Довговічність (Atomicity, Consistency, Isolation, Durability)

API – Прикладний програмний інтерфейс (Application Programming Interface)

CEO – Головний виконавчий директор (Chief Executive Officer)

CPU – Центральний процесор (Central Processing Unit)

CRUD – Створення, Читання, Оновлення, Видалення (Create, Read, Update, Delete)

CSRF – Підробка міжсайтових запитів (Cross-Site Request Forgery)

CSS – Каскадні таблиці стилів (Cascading Style Sheets)

DTO – Об'єкт передачі даних (Data Transfer Object)

HTML – Мова гіпертекстової розмітки (HyperText Markup Language)

HTTP – Протокол передачі гіпертексту (Hypertext Transfer Protocol)

HTTPS – Захищений протокол передачі гіпертексту (Hypertext Transfer Protocol Secure)

IDE – Інтегроване середовище розробки (Integrated Development Environment)

IPC – Міжпроцесна взаємодія (Inter-Process Communication)

JS – JavaScript

JSON – Нотація об'єктів JavaScript (JavaScript Object Notation)

JWT – Веб-токен JSON (JSON Web Token)

ORM – Об'єктно-реляційне відображення (Object-Relational Mapping)

OS – Операційна система (Operating System)

P2P – Пірингова (однорангова) мережа (Peer-to-Peer)

RAM – Оперативна пам'ять (Random Access Memory)

REST – Передача репрезентативного стану (Representational State Transfer)

SQL – Мова структурованих запитів (Structured Query Language)

SSL – Протокол захищених сокетів (Secure Sockets Layer)

TS – TypeScript

UI – Користувацький інтерфейс (User Interface)

UML – Уніфікована мова моделювання (Unified Modeling Language)

UX – Досвід користувача (User Experience)

WCAG – Настанови з доступності вебвмісту (Web Content Accessibility Guidelines)

XSS – Міжсайтовий скриптинг (Cross-Site Scripting)

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. У сучасному ігровому світі, де геймери та ігрові проєкти переходять з однієї платформи на іншу, потреба в універсальному лаунчері стає все більш актуальною. Сьогодні, завдяки веб-сервісам та кросплатформним рішенням, функції програм можуть стати доступними користувачам на різноманітних платформах через Інтернет. Це стосується і ігрової індустрії, де потреба у таких рішеннях активно зростає. Використання веб-технологій у створенні лаунчерів дозволяє забезпечити уніфікований доступ до ігрового контенту, незалежно від операційної системи користувача.

Причина активного впровадження мультиплатформних рішень полягає у їх здатності використовувати відкриті стандарти та протоколи для обміну та передачі даних, що забезпечує широку сумісність та легкість інтеграції з різними системами та сервісами. Кросплатформні лаунчери для ігор, засновані на цих принципах, можуть значно поліпшити досвід користувачів та ентузіастів, надаючи їм доступ до широкого спектру ігор та арі на будь-якому пристрої.

Мета даної роботи – моделювання та розробка лаунчера, який використовує веб-технології для забезпечення своєї роботи та інтеграції з різними ігровими платформами та сервісами. Це включає в себе виконання наступних завдань:

- Аналіз існуючих рішень та визначення функціональних потреб користувачів.
- Вибір технологій для реалізації кросплатформного лаунчера.
- Розробка архітектури та інтерфейсу лаунчера.
- Інтеграція з ігровими сервісами та платформами.
- Розробка системи лаунчера.

Об’єктом дослідження цієї дипломної роботи виступають мультиплатформні технології та методи створення веб-сервісів, що застосовуються в ігровій індустрії. Це широке поняття охоплює архітектурні підходи, як-от REST API для комунікації між клієнтом і сервером, та сучасні фреймворки для побудови користувацьких інтерфейсів, що здатні працювати на різних операційних системах.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		7

Предметом, у свою чергу, є безпосереднє застосування цих технологій для проєктування та розробки універсального ігрового лаунчера. Створення такого продукту передбачає осмислений вибір та інтеграцію сучасних інструментів. Для серверної частини (бекенду) було обрано мову Rust через її виняткову продуктивність, гарантії безпеки пам'яті та чудові можливості для побудови високонавантажених, конкурентних систем. Клієнтська частина реалізована за допомогою зв'язки Tauri [10] + Svelte [4]: Tauri дозволяє створювати легкі та безпечні настільні додатки, використовуючи системний веб-рушій, а Svelte як компілятор генерує мінімальний та швидкий код для інтерфейсу. Для надійного зберігання даних користувачів, ігрових профілів та налаштувань використовується реляційна база даних PostgreSQL [11], відома своєю стабільністю та відповідністю ACID-принципам. Окрім цього, для забезпечення гнучкості розгортання, ізоляції середовища та масштабування інфраструктури, планується залучити технологію контейнеризації Docker [5, 28] та панель управління Pterodactyl [6].

Внаслідок цього, створення кросплатформного лаунчера для ігор є не лише актуальним, але й комплексним інженерним завданням, що повністю відповідає сучасним тенденціям у розробці програмного забезпечення. Ця робота спрямована на подолання фундаментальних викликів, пов'язаних із фрагментацією ігрового ринку, коли користувачі змушені оперувати десятками окремих програм, що споживають ресурси системи. Кінцевою метою є створення потужного та універсального інструменту, який би централізував ігровий досвід, зробивши його значно зручнішим та доступнішим для широкого кола користувачів. Більше того, розробка такого продукту є кроком до формування незалежної екосистеми, що дозволить замінити закриті, комерційно орієнтовані та потенційно ворожі програмні рішення, надаючи користувачам більше контролю та прозорості.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ КРОСПЛАТ-ФОРМНОГО ЛАУНЧЕРА ДЛЯ ІГОР

1.1 Постановка задачі

Метою даної випускної кваліфікаційної роботи є розробка універсального ігрового лаунчера, який використовує сучасні веб-технології для забезпечення крос-платформної сумісності та інтеграції з різноманітними ігровими сервісами. У сучасному фрагментованому ігровому ринку, де користувачі та адміністратори змушені взаємодіяти з численними платформами та лаунчерами на різних пристроях та операційних системах, створення єдиної точки доступу до ігор стає дуже актуальним завданням. Універсальний лаунчер повинен спростити взаємодію гравців з адміністрацією, об'єднавши їх в одному місці. Крім того, розробка власного рішення є важливим кроком для створення альтернативи іншим програмним продуктам.

Розробка такого продукту вимагає комплексного підходу, який розпочинається з глибокого аналізу існуючих на ринку лаунчерів (як офіційних, - від великих видавців, - так і сторонніх агрегаторів) та ретельного вивчення потреб і очікувань цільової аудиторії. Необхідно чітко визначити ключові функціональні вимоги, які користувачі вважають пріоритетними. До таких вимог, як правило, належать: ефективне управління користувачами та об'єднаною бібліотекою ігор з різних джерел, безшовна інтеграція з акаунтами популярних платформ (наприклад, Curseforge, Modrinth), автоматичне відстеження та встановлення оновлень для модів та модпаків, також надійна можливість запуску ігор, незалежно від їх початкової платформи через єдиний інтерфейс.

На основі проведеного аналізу буде здійснено обґрунтований вибір оптимального стеку технологій. Для розробки серверної частини (бекенду), що відповідає за логіку, управління даними та взаємодію з API сторонніх сервісів, планується використання мови програмування Rust. Цей вибір зумовлений високою продуктивністю, надійністю та гарантіями безпеки пам'яті, що є критично важливим для стабільної роботи сервісу. Клієнтська частина (фронтенд), тобто сам настільний додаток, буде реалізована за допомогою фреймворку Tauri у поєднанні з бібліотекою Svelte. Tauri дозволяє створювати легкі, швидкі та по-справжньому

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		9

кроссплатформні настільні додатки з використанням веб-технологій (HTML, CSS, JavaScript/TypeScript), а Svelte, як компілятор, забезпечує високу продуктивність інтерфейсу користувача. Для зберігання даних користувачів, інформації про ігри, налаштувань та інших релевантних даних, буде обрано відповідну систему управління базами даних, наприклад, гнучку NoSQL базу даних MongoDB або реляційну PostgreSQL, залежно від фінальної архітектури даних та вимог до масштабованості. Важливою складовою проєкту є також забезпечення надійного розгортання, моніторингу та управління інфраструктурою додатка, для чого планується використання технологій контейнеризації Docker [5, 28] та, можливо, панелі управління Pterodactyl для спрощення адміністрування серверів.

Процес розробки включатиме детальне проєктування архітектури системи, яка має бути гнучкою, модульною та масштабованою для майбутнього розширення функціоналу. Особливу увагу приділено розробці інтуїтивно зрозумілого, естетично привабливого та зручного користувацького інтерфейсу (UI/UX). Ключовий і найскладніший етап роботи - реалізація інтеграції з API різноманітних ігрових сервісів та платформ для коректного отримання даних про ігри користувача, синхронізації статусів акаунтів та забезпечення можливості надійного запуску ігор безпосередньо через розроблений лаунчер. Після реалізації основного функціоналу буде проведено комплексне тестування, що включатиме функціональне тестування, тестування інтерфейсу, тестування навантаження та перевірку сумісності на цільових операційних системах (Windows, macOS, Linux).

Отже, фінальним результатом роботи має стати функціональний прототип кроссплатформного ігрового лаунчера. Цей прототип продемонструє практичні переваги використання сучасних веб-технологій та підходів для створення універсальних ігрових рішень, здатних ефективно об'єднати розрізнені ігрові екосистеми, суттєво покращити загальний досвід користувачів та запропонувати їм зручну й надійну альтернативу існуючим рішенням.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		10

1.2. Аналіз аналогів програмного продукту

Для розробки якісного програмного продукту необхідно ретельно проаналізувати конкурентів на ринку. Вивчити їх користувацький інтерфейс, функціональні можливості, сильні і слабкі сторони. Також порівняти їх між собою, щоб знайти свою нішу і вирізнитися з-поміж інших.

Modrinth визначає себе як платформу, створену для покращення світу модінгу відеоігор, з початковим фокусом виключно на Minecraft: Java Edition. Платформа прагне стати центральним місцем для всіх потреб гравців, авторів та розробників модів. Вона складається з трьох основних компонентів: веб-сайту Modrinth.com для перегляду та обміну контентом, десктопного додатку Modrinth App та Modrinth API, що слугує основою для взаємодії з базою даних платформи. [7]

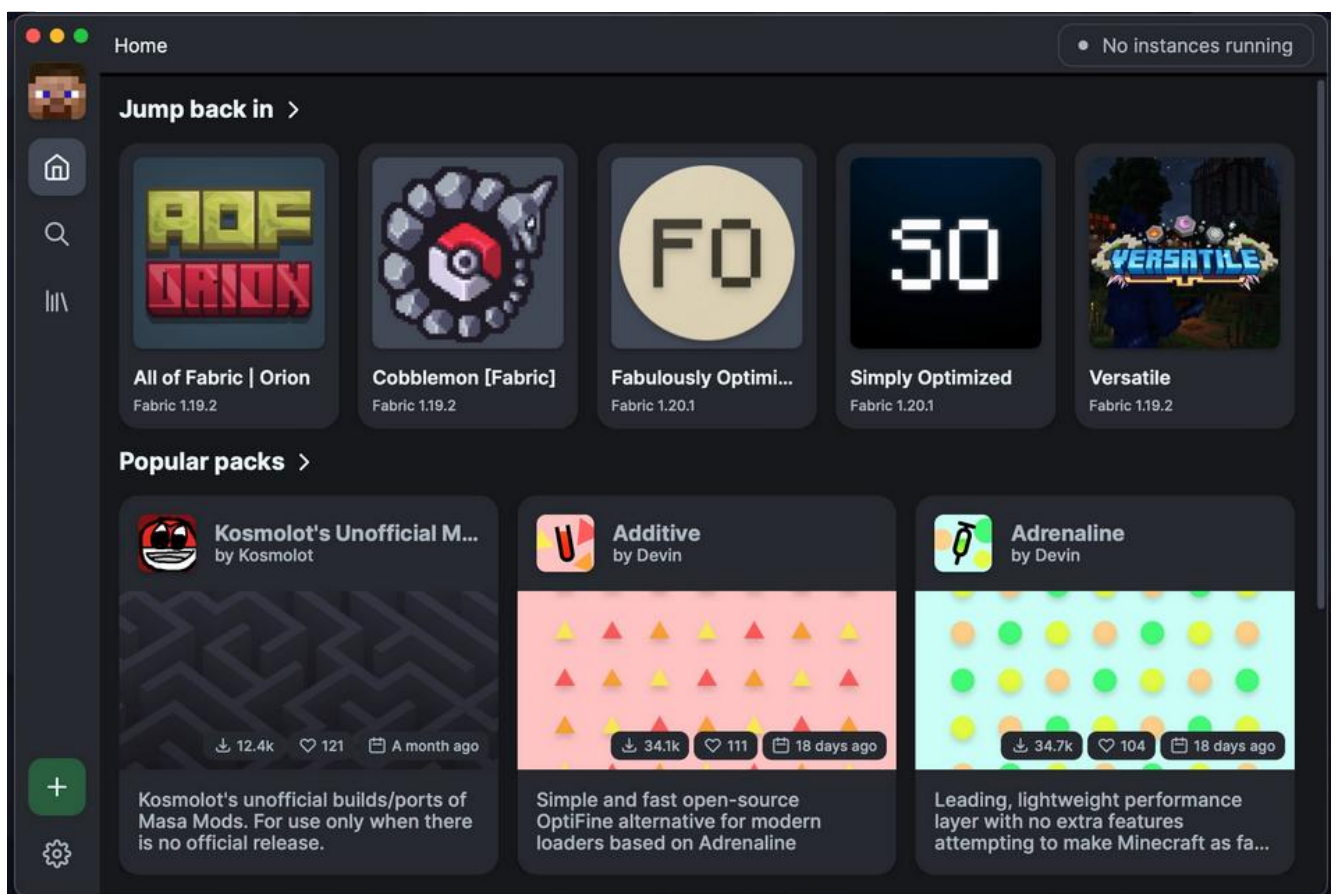


Рис. 1.1 - Десктопний додаток Modrint

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		11

До **переваг** даного програмного продукту можна віднести високу продуктивність та ефективність, відкритий вихідний код, сучасний та чистий інтерфейс, безпеку та відсутність реклами.

До **недоліків** можна віднести обмежені можливості налаштування та залежність від екосистеми Modrint.

CurseForge – одна з найбільших платформ для модифікацій Minecraft та інших ігор, де користувачі можуть завантажувати, встановлювати та обмінюватися модами, ресурсами та текстурами. Співпрацює з такою програмою, як Overwolf, для автоматизації завантаження та оновлення модів. Платформа пропонує широкий спектр функцій для комунікації між користувачами.[8]

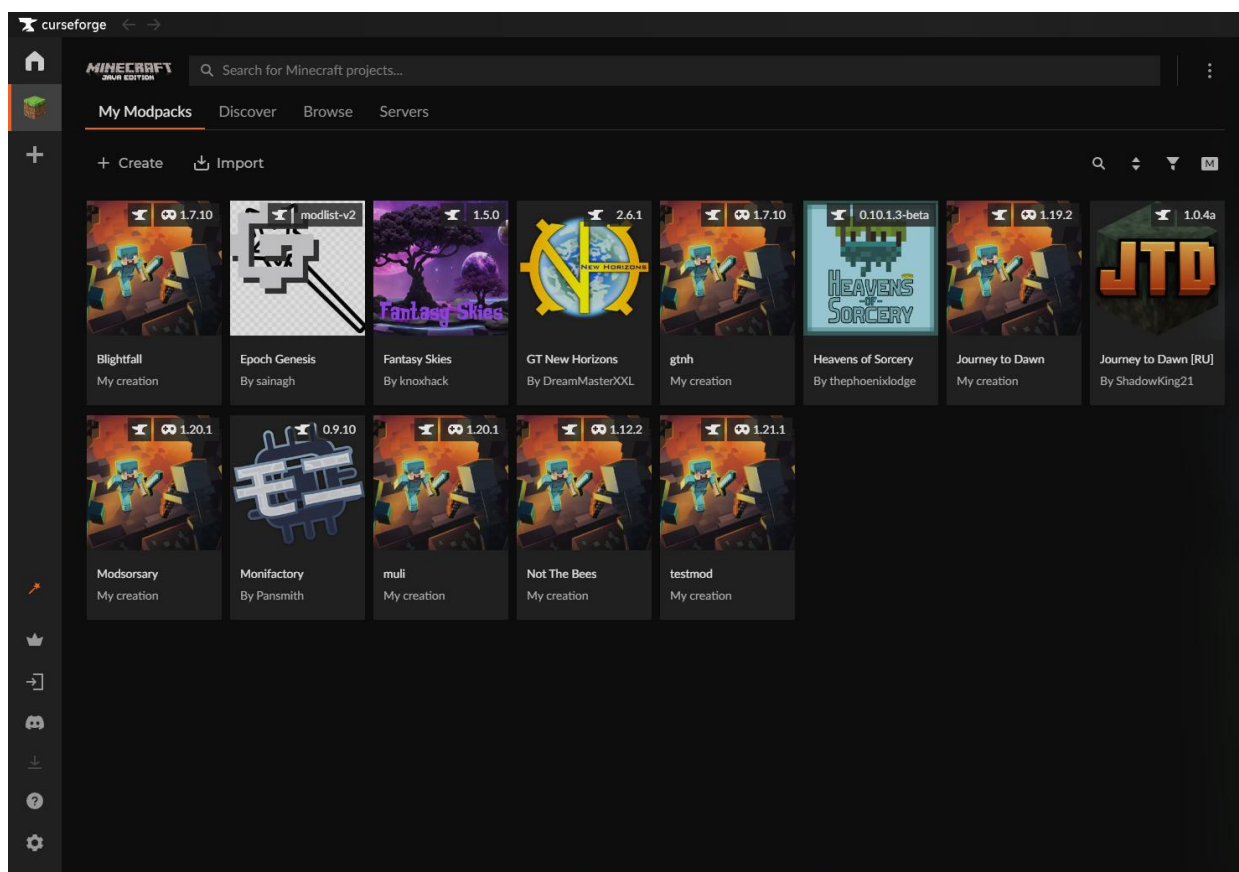


Рис. 1.2 - Десктопний додаток CurseForge

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		12

До **переваг** цього програмного продукту можна віднести величезну бібліотеку контенту, інтеграції з іграми та програмами, систему рекомендацій і рейтингу модів.

Недоліками можна вважати високе споживання системних ресурсів, наявність реклами, проблеми конфіденційності та збору даних, обмежену кастомізацію та контроль, залежність від екосистеми CurseForge. Також ця платформа підтримує лише обмежену кількість ігор.

Prism Launcher – це лаунчер з відкритим вихідним кодом, розроблений для надання користувачам потужних інструментів для керування кількома ігровими екземплярами (інстансами), обліковими записами та модифікаціями. Він позиціонується як інструмент, який фокусується на свободі користувача та можливості вільного розповсюдження.[1]

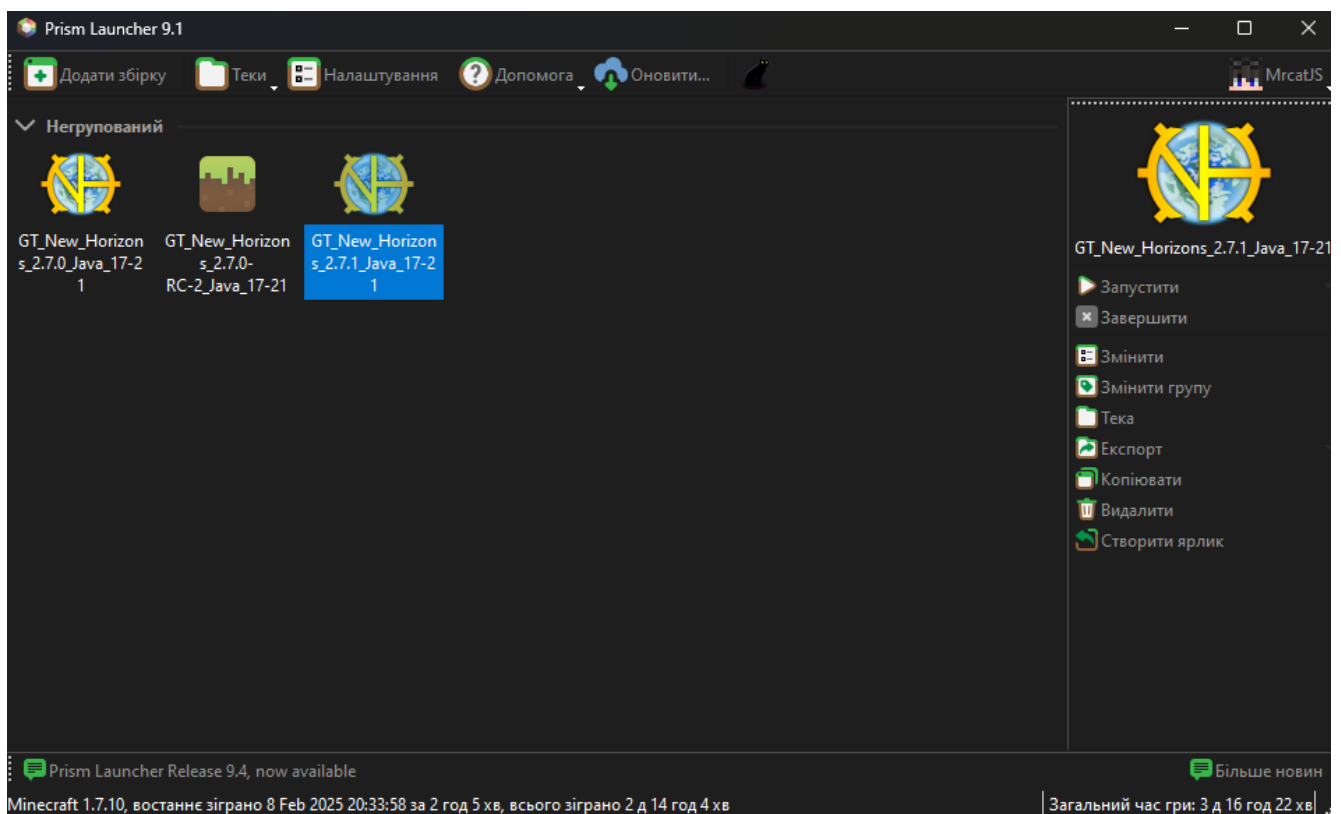


Рис. 1.3 - Десктопний додаток Prism Launcher

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Перевагами даного програмного продукту можна назвати високу продуктивність та легкість, завантаження контенту з різних платформ, повне керування екземплярами гри, високий рівень кастомізації та контролю, відсутність реклами.

Недоліки - складний інтерфейс.

Таблиця порівняння аналогів:

Порівняння аналогів Табл. 1.1

Критерій (1)	Modrinth (2)	CurseForge (3)	PrismLauncher (4)	Розроблюваний додаток (5)
Комфортний інтерфейс	+	-	-	+
Кастомізація	-	-	+	+
Продуктивність	Faster	Slower	Faster	Faster
Споживання ресурсів	Low	High	Low	Low
Реклама	-	+	-	-
Підходить для адміністрації	-	-	-	+
Розширення плагінами	-	-	-	+
Інтеграція з Pterodactyl	-	-	-	+
Система ролей та дозволів	-	-	-	+
Інструмент вирішення конфліктів модів	-	+/-	+/-	+

1	2	3	4	5
Оверлей для discord	-	+	-	+
Відстеження досягнень	-	+	-	+
Синхронізація налаштувань між пристроями	-	-	-	+

Детальний аналіз функціоналу та характеристик існуючих лаунчерів, представлений у Табл. 1.1, чітко демонструє, що кожен з розглянутих продуктів має свої унікальні сильні сторони та переваги, які можуть задовольняти потреби певних сегментів користувачів. Modrinth приваблює сучасним дизайном та відкритістю, CurseForge – величезною базою контенту та інтеграцією з Overwolf, а Prism Launcher – гнучкістю, контролем та продуктивністю. Проте, жоден з них не пропонує комплексного рішення, яке б поєднувало зручний та сучасний інтерфейс, високу продуктивність та безпеку, розширені інструменти для адміністрування та серверів, а також незалежність від рекламних моделей. Саме цю нішу покликаний зайняти розроблюваний додаток.

Використання передового технологічного стеку надає "YundiLauncher" значні потенційні переваги над конкурентами, побудованими на менш ефективних або більш ресурсоємних технологіях (як Electron чи Java для UI). Це дозволяє очікувати вищої продуктивності, меншого споживання ресурсів та підвищеного рівня безпеки. Запланована гнучкість інтеграції створює фундамент для довгострокового розвитку та адаптації лаунчера до мінливих потреб ринку. Особливий акцент на розширених інструментах адміністрування, інтеграції з Pterodactyl вигідно відрізняє "YGL" від існуючих рішень, орієнтованих переважно на кінцевих гравців. Отже, розробка є обґрунтованою та перспективною, маючи на меті створення потужного, безпечного та універсального інструменту для широкого кола користувачів та адміністраторів.

1.3 Дослідження технологій реалізації

При розробці кросплатформних лаунчерів для ігор важливо вибрати технології, які забезпечать гнучкість, високу продуктивність та широку сумісність з різними ігровими платформами та операційними системами.

Розглядалися наступні основні стеки технологій: **Tauri** з використанням **Svelte** для UI + **Rust** для бекенду та логіки (цей підхід використовує Tauri для створення легкої та швидкої настільної оболонки за допомогою веб-технологій (Svelte для інтерфейсу), а основну логіку та бекенд реалізує на Rust, що забезпечує високу продуктивність та безпеку); **Tauri та Svelte + C#** для бекенду (схожий підхід, де Tauri та Svelte використовуються для фронтенду, але бекенд реалізується на C#, взаємодія між фронтом і бекендом відбувається через механізм міжпроцесної взаємодії IPC, наданий Tauri); **Avalonia UI + C#** для бекенду та логіки (цей варіант передбачає використання фреймворку Avalonia UI для створення кросплатформного інтерфейсу на C# з використанням XAML-подібної розмітки, а також реалізацію бекенду та логіки на тому ж C#).

Таблиця порівняння стеків технологій:

Порівняння стеків технологій табл. 1.2

Критерій (1)	Tauri (Svelte) + Rust (2)	Tauri (Svelte) + C# (3)	Avalonia UI + C# (4)
Продуктивність (UI)	Faster	Faster	Good
Продуктивність (Бекенд)	Lowest	Low	Average
Розмір застосунку	Lowest	Small	Medium
Безпека	Highest	Good	Good
Розробка UI	Svelte	Svelte	XAML
Безпека (Memory)	Highest	Good	Good
Інтеграція UI та логіки	IPC/API	IPC/API	Direct Call

Підсумовуючи порівняння, для даного проєкту було обрано стек Tauri + Svelte + Rust. Цей вибір обґрунтований найкращими показниками продуктивності як для користувацького інтерфейсу, так і для бекенду, найменшим розміром кінцевого додатку та найвищим рівнем безпеки. Зокрема, мова Rust, що лежить в основі серверної та частково клієнтської логіки, забезпечує безпрецедентну швидкість та гарантії безпеки пам'яті без необхідності у збирачі сміття, що є ідеальним для розробки надійних і високонавантажених систем. Svelte, у свою чергу, як компілятор, генерує мінімальний та високоефективний код, що робить користувацький інтерфейс надзвичайно швидким та чутливим. Фреймворк Tauri об'єднує ці переваги, створюючи легку настільну оболонку, яка, на відміну від аналогів, не включає в себе повноцінний браузер.

Можна сказати, що обраний технологічний стек є комплексним та сучасним рішенням. Для розробки серверної частини та клієнтської логіки було обрано мову програмування Rust. Створення настільного додатку покладається на фреймворк Tauri, а для розробки користувацького інтерфейсу використовується Svelte. У якості системи управління базами даних передбачається інтеграція з PostgreSQL [11] через API, що гарантує цілісність та надійність даних завдяки відповідності ACID-принципам. Весь процес розробки та розгортання організовано з використанням стандартних індустріальних інструментів: Docker [5, 28] для створення портативних та ізольованих середовищ, Pterodactyl для спрощення управління серверами, Git для контролю версій та Visual Studio Code як основне середовище розробки. В сукупності, цей технологічний стек є міцним фундаментом для створення швидкого, безпечного, кросплатформного та легкого лаунчера з сучасним інтерфейсом.

1.4 Розробка макету

В основі проєктування програмного забезпечення лаунчера лежить фундаментальний принцип, орієнтований на користувача. Ключовим завданням є створення не просто функціонального інструменту, а інтуїтивно зрозумілого та комфортного середовища, яке пріоритезує та покращує користувацький досвід (UX). Сучасний користувач очікує від додатків не лише виконання заявлених функцій, а й швидкості, реактивності та візуальної привабливості. Тому процес розробки

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		17

макету був комплексним і охоплював детальне проектування кожної ключової точки взаємодії, перетворюючи потенційно складні процеси на прості та логічні кроки.

Перше враження користувача формується ще до того, як він отримає доступ до основного функціоналу. Саме тому екран завантаження був спроектований не як пасивний індикатор очікування, а як активний елемент, що задає тон усій подальшій взаємодії. Замість статичного зображення, макет передбачає плавну та мінімалістичну анімацію, наприклад, ледь помітну пульсацію логотипа або м'який фоновий градієнтний ефект. Така анімація не лише виглядає сучасно, але й створює психологічне відчуття швидкості та реактивності додатку. Більше того, цей екран виконує важливу технічну функцію: поки користувач спостерігає за анімацією, у фоновому режимі відбувається ініціалізація основних компонентів системи, завантаження конфігурацій та перевірка наявності критичних оновлень. Це дозволяє значно скоротити час "холодного старту" і забезпечити майже миттєвий перехід до екрану авторизації, що є важливим показником якості UX.

Екран авторизації та реєстрації є критичною точкою, оскільки будь-яка складність на цьому етапі може відштовхнути нового користувача. Тому його макет розроблено за принципами максимальної чистоти та функціональності. Інтерфейс позбавлений зайвих візуальних елементів, щоб ніщо не відволікало від головної мети. Користувачеві пропонуються чітко підписані поля для введення логіну та пароля, а також зрозумілі кнопки для основних дій: "Увійти", "Зареєструватися" та "Забули пароль?". Для покращення досвіду та прискорення процесу входу, макет включає можливість авторизації через популярні сторонні сервіси, такі як Google або Discord, що дозволяє користувачам увійти в один клік. Важливою частиною є продумана система зворотного зв'язку. Макет передбачає миттєву валідацію введених даних ще до відправки на сервер: наприклад, поле підсвічується червоним, якщо email введено у неправильному форматі, або з'являється підказка про вимоги до складності пароля. Повідомлення про помилки (наприклад, "Неправильний пароль" або "Користувач з таким іменем вже існує") є конкретними, зрозумілими і допомагають користувачеві швидко виправити ситуацію.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				18
Змн.	Арк.	№ докум.	Підпис	Дата		

Після успішної авторизації користувач потрапляє на головну сторінку, або дашборд, який спроектовано як центральний інформаційний та навігаційний хаб. Макет побудований за принципом модульної сітки, що дозволяє логічно структурувати великий обсяг інформації та зробити її легкою для сприйняття. Верхню, найпомітнішу частину займає динамічний банер або карусель, де відображаються найважливіші новини, анонси подій або промо-матеріали. Нижче розташована стрічка персоналізованих рекомендацій. Цей блок не є статичним; він аналізує історію активності користувача — у які ігри він грає найчастіше, яким жанрам віддає перевагу — і на основі цих даних пропонує новий контент або сервери, які можуть його зацікавити. Окрема секція відводиться під соціальну активність: тут користувач може бачити, у що грають його друзі, які досягнення вони нещодавно отримали, що створює відчуття спільноти та стимулює до взаємодії.

Сторінка профілю конкретної гри є деталізованим центром, що надає вичерпну інформацію та всі необхідні інструменти для керування. Щоб уникнути перевантаження користувача, макет розділено на логічні вкладки. Вкладка "Огляд" містить загальний опис, ключові особливості та офіційні трейлери. "Скріншоти" та "Відео" дозволяють візуально оцінити гру. "Системні вимоги" надають чітку технічну інформацію. "Спільнота" може інтегрувати відгуки гравців, посібники та посилання на форуми. Центральним елементом всього екрану є велика, візуально виділена кнопка "Грати" або "Встановити". Її статус динамічно змінюється, відображаючи поточний стан: вона може показувати прогрес завантаження, сигналізувати про наявність оновлення або просто запускати гру.

Профіль користувача - це його особистий простір всередині екосистеми лаунчера. Макет цієї сторінки дозволяє гравцеві виразити свою індивідуальність. Користувач може завантажити власний аватар, написати короткий опис про себе та обрати фон для профілю. Важливою частиною є "вітрина", де гравець може виставити свої улюблені ігри або найцінніші досягнення, якими він пишається. Окрім візуальної кастомізації, профіль надає доступ до деталізованої статистики: загальний час у грі, прогрес по досягненнях, історія матчів. Для соціальної взаємодії передбачені інструменти для управління списком друзів. Окрему увагу приділено

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		19

налаштуванням приватності, що дозволяє користувачеві гнучко контролювати, яка інформація про його активність буде видимою для інших.

Нарешті, розділ налаштувань, хоча й часто є утилітарним, спроектований з такою ж увагою до UX, як і решта додатку. Щоб користувач не губився у великій кількості опцій, макет структурований за чіткими категоріями: "Загальні" (вибір мови, налаштування поведінки додатку при старті системи), "Завантаження" (вибір папки для встановлення ігор, можливість обмеження швидкості завантаження), "Інтерфейс" (вибір світлої або темної теми, налаштування масштабу елементів), "Сповіщення" (детальний контроль над тим, які повідомлення отримувати) та "Акаунт" (управління паролем, підключеними соціальними мережами). Кожен елемент макету, від головної сторінки до найменшого перемикача в налаштуваннях, проектується з урахуванням принципів адаптивності. Це гарантує, що інтерфейс буде коректно та зручно відображатися на екранах з різною роздільною здатністю та співвідношенням сторін, забезпечуючи єдиний, цілісний та позитивний користувацький досвід на всіх підтримуваних платформах.

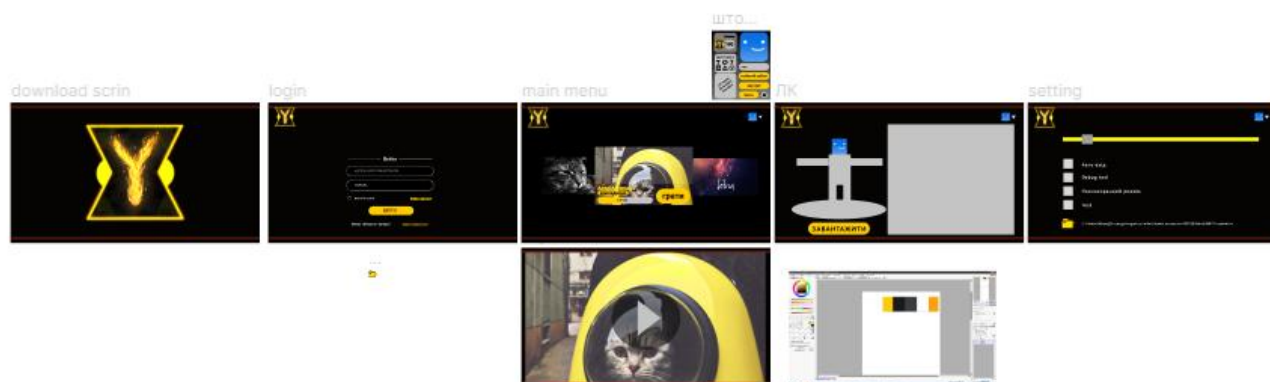


рис. 1.4 - Замальовки макету продукту

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		20

Висновки до першого розділу

У першому розділі роботи обґрунтовано актуальність створення універсального кросплатформного ігрового лаунчера на фоні сучасного фрагментованого ринку. Метою роботи визначено моделювання та розробка такого лаунчера із використанням веб-технологій з ключовими завданнями, що включають аналіз ринку, вибір технологій, проєктування архітектури та інтерфейсу, інтеграцію з сервісами та безпосередню розробку системи.

Детальний аналіз предметної області дозволив розглянути існуючі програмні аналоги, такі, як Modrinth, CurseForge та Prism Launcher. Виявлено їх сильні та слабкі сторони за критеріями зручності інтерфейсу, можливостей кастомізації, продуктивності, наявності реклами та специфічних функцій для адміністраторів. Порівняльний аналіз показав, що жоден з існуючих продуктів не пропонує комплексного рішення, яке б поєднувало зручність для гравців, гнучкість налаштувань та інструменти для адміністрування, що підтверджує необхідність розробки нового лаунчера.

На основі аналізу потреб користувачів та функціоналу конкурентів визначено основні функціональні вимоги до майбутнього продукту. Вони включають ефективне управління обліковими записами користувачів, створення єдиної бібліотеки ігор з різних джерел, забезпечення безшовної інтеграції з API популярних ігрових платформ та сервісів, автоматизацію процесу оновлення модів та модпаків, а також надання надійної функції запуску ігор через уніфікований інтерфейс.

Проведено дослідження та порівняння потенційних стеків технологій для реалізації проєкту, включаючи варіанти Tauri (Svelte) + Rust, Tauri (Svelte) + C# та Avalonia UI + C#. Оцінка за критеріями продуктивності інтерфейсу та бекенду, кінцевого розміру додатку та рівня безпеки дозволила обґрунтувати вибір стеку Tauri для фронтенду (з використанням Svelte для UI) та мови Rust для розробки бекенду.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		21

РОЗДІЛ 2. ПРОЄКТУВАННЯ КРОСПЛАТФОРМНОГО ЛАУНЧЕРА

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Проектування кросплатформного лаунчера для ігор починається з визначення його функціональності та взаємодії з користувачами. Лаунчер являє собою вебінтерфейс, де користувачі можуть реєструватися, переглядати та запускати доступні ігри, налаштовувати ігровий процес відповідно до своїх потреб. Окрім того, важливою частиною лаунчера є адміністративна панель, яка дозволяє керувати користувачами, іграми та надавати технічну підтримку.

Система передбачає декілька таких типів. Ключовими акторами є звичайний **Користувач** та **Адміністратор**, чії ролі визначають базову функціональність платформи.

Користувач є основним споживачем послуг платформи. Його взаємодія з системою охоплює повний цикл використання ігрового контенту: від першого знайомства до активної гри. Користувач може створити власний обліковий запис, використовуючи процес реєстрації, авторизуватися для доступу до свого профілю та функцій платформи. Йому доступний каталог ігор, де він може не лише переглядати детальну інформацію про кожну гру (описи, скріншоти, системні вимоги), але й використовувати фільтри та сортування (за жанром, рейтингом, датою випуску) для зручного пошуку. Важливою функцією є можливість завантаження та запуску обраних ігор безпосередньо через лаунчер, який виступає єдиною точкою входу до ігрової бібліотеки. Крім того, користувач має змогу налаштовувати як інтерфейс самого лаунчера (наприклад, вибір теми оформлення, мови інтерфейсу, налаштування сповіщень), так і параметри конкретної гри (графічні налаштування, розкладка управління, рівень гучності звуку), адаптуючи їх під власні вподобання та технічні можливості свого обладнання. Проходження гри або участь у багатокористувацьких режимах є основною метою використання платформи для цього актора, і лаунчер має забезпечити максимально плавний та безперебійний доступ до цих процесів.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Адміністратор, на відміну від звичайного користувача, володіє значно ширшими повноваженнями та відповідає за стабільну, безпечну та коректну роботу всієї платформи. Його обов'язки охоплюють кілька ключових сфер. По-перше, це **управління користувачами**: адміністратор може не лише переглядати список зареєстрованих користувачів та їх профілі, але й активно керувати акаунтами – створювати нові (наприклад, для тестування або для персоналу), редагувати дані існуючих користувачів, тимчасово блокувати або остаточно видаляти акаунти порушників правил платформи чи у випадку підозрілої активності. Важливою частиною управління є гнучке призначення ролей та детальних прав доступу, що дозволяє точно контролювати, які функції та розділи платформи доступні різним групам користувачів (наприклад, модераторам, тестувальникам, звичайним гравцям). По-друге, адміністратор відповідає за **управління контентом**: він додає нові ігри на платформу, ретельно заповнює та оновлює всю супутню інформацію (описи, жанри, теги, системні вимоги, зображення, відео), перевіряє контент на відповідність встановленим правилам та стандартам платформи (наприклад, вікові обмеження, відсутність заборонених матеріалів), видаляє неприйнятні матеріали або застарілі, неактуальні ігри. По-третє, адміністратор займається **технічною підтримкою**: він обробляє запити користувачів, що надходять через інтегровану систему тікетів або інші канали зв'язку, діагностує та допомагає вирішувати технічні проблеми, пов'язані з роботою лаунчера, процесом завантаження чи запуску ігор, та надає кваліфіковані консультації щодо використання платформи.

Хоча для базової функціональності достатньо ролей Користувача та Адміністратора, у складніших конфігураціях платформи можуть існувати й інші спеціалізовані ролі. Наприклад, рис. 2.2, **Технічний персонал** може відповідати за обслуговування серверної інфраструктури (моніторинг навантаження, оновлення ПЗ, резервне копіювання), оптимізацію продуктивності та забезпечення безпеки системи від зовнішніх загроз. **Керівник (СЕО)** або менеджер проєкту може мати доступ до розширених аналітичних звітів, статистики використання платформи, визначаючи стратегічні напрямки розвитку та приймаючи ключові бізнес-рішення. Гнучкість системи ролей дозволяє адаптувати платформу під конкретні бізнес-потреби та

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		23

організаційну структуру компанії, забезпечуючи ефективне управління на всіх рівнях. Чітке визначення всіх акторів та їх можливостей є фундаментом для подальшого проєктування варіантів використання та архітектури системи.

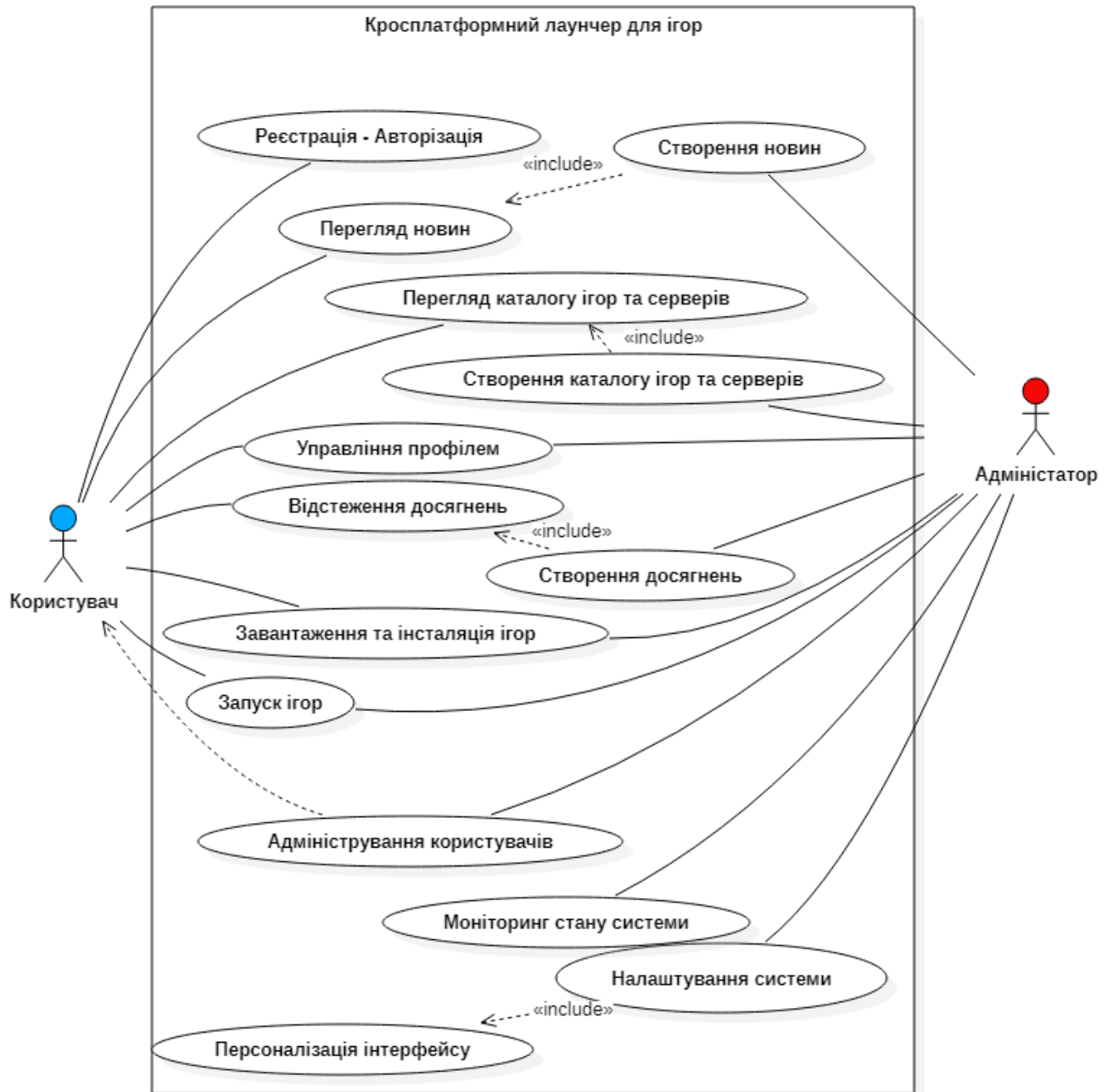


Рис. 2.1 - Діаграма використання програмної системи та взаємодії акторів

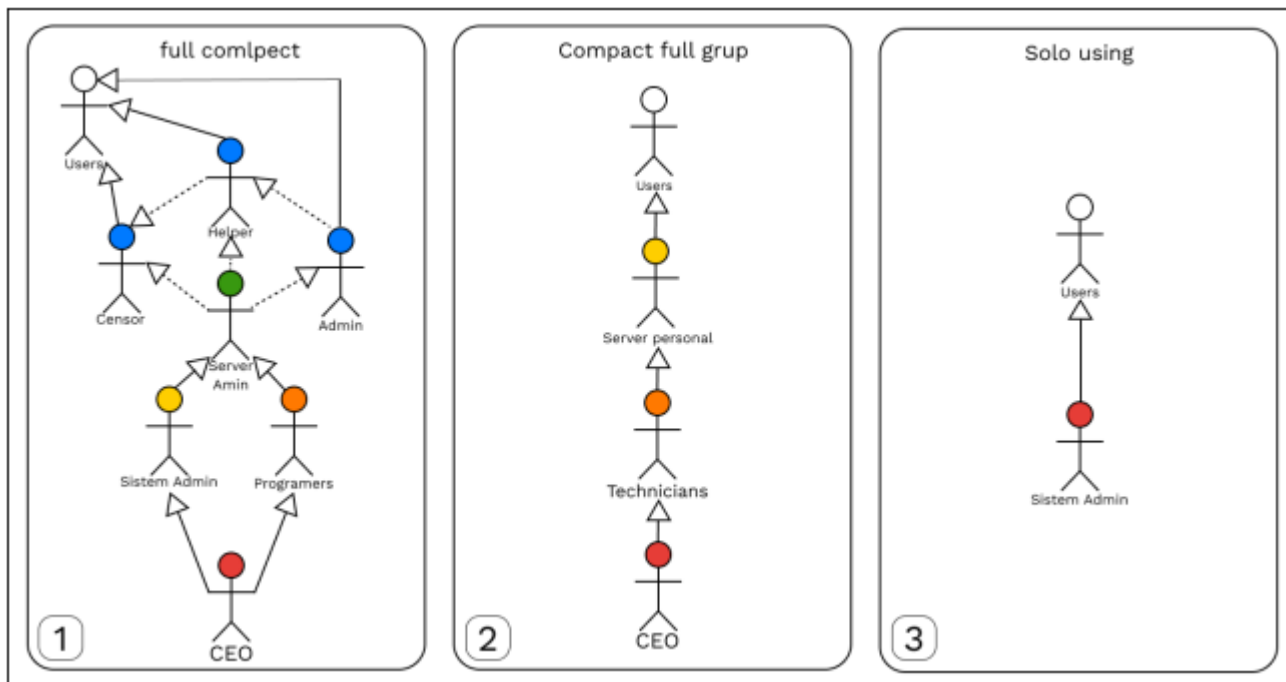


Рис. 2.2 - Діаграма можливих ролей адміністрації

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.2

Реєстр варіантів використання (Користувач) Табл. 2.1.

D0	Основні актори (1)	Найменування (2)	Формулювання (3)
D1	Користувач	Авторизація / реєстрація	Користувач здійснює реєстрацію або авторизацію в системі для отримання доступу до її функцій.
D2		Перегляд новин	Користувач переглядає останні новини, пов'язані з іграми та сервісами системи.
D3		Перегляд каталогу ігор та серверів	Користувач переглядає доступний каталог ігор та серверів в системі.
D4		Управління профілем	Користувач може змінювати або оновлювати свій профіль в системі.
D5		Відстеження досягнень	Користувач відстежує свої досягнення в іграх.

0	1	2	3
D6	Користувач	Завантаження та інсталяція ігор	Користувач може завантажувати і встановлювати ігри з каталогу.
D7		Запуск ігор	Користувач запускає ігри з системи після їх інсталяції.
D8		Персоналізація інтерфейсу	Користувач налаштовує інтерфейс системи згідно зі своїми уподобаннями.

Реєстр варіантів використання (Адміністратор) Табл. 2.2

D0	Основні актори (1)	Найменування (2)	Формулювання (3)
D1	Адміністратор	Авторизація	Адміністратор здійснює авторизацію в системі для отримання доступу до її функцій.
D2		Створення новин	Адміністратор додає новини в систему для інформування користувачів
D3		Створення каталогу ігор та серверів	Адміністратор створює та управляє каталогом ігор та серверів в системі.
D4		Створення досягнень	Адміністратор створює нові досягнення для користувачів в системі.
D5		Адміністрування користувачів	Адміністратор керує обліковими записами користувачів, зокрема, їх реєстрацією та доступом.
D6		Моніторинг стану системи	Адміністратор відстежує поточний стан та роботу системи, включаючи її ресурси та процеси.

0	1	2	3
D7	Адміністратор	Налаштування системи	Адміністратор налаштовує параметри та конфігурації системи для забезпечення її ефективної роботи.

2.2. Визначення вимог до системи

На основі ретельного аналізу предметної області, виявлених варіантів використання та очікувань цільової аудиторії, яка включає як звичайних гравців, так і адміністраторів ігрових спільнот, сформульовані детальні вимоги до розроблюваного кросплатформного лаунчера. Ці вимоги є основою для подальшого проєктування та розробки, визначаючи функціональні можливості, атрибути якості та технічні обмеження системи. Вимоги традиційно поділяються на три основні категорії: функціональні, нефункціональні та системні.

Функціональні вимоги визначають конкретні дії та операції, які система повинна бути здатною виконувати, формуючи ядро її поведінки. Першочерговою вимогою є реалізація надійної системи аутентифікації та авторизації. Це означає, що система повинна надавати безпечний механізм для реєстрації нових користувачів та входу для вже існуючих. Критично важливою є імплементація системи контролю доступу на основі ролей (RBAC), де кожній ролі, такій як "Користувач" або "Адміністратор", буде присвоєно чітко визначений набір дозволів, що обмежує доступ до певних функцій та даних.

Наступною важливою вимогою є управління обліковими записами. Звичайні користувачі повинні мати можливість легко редагувати інформацію у своєму профілі, наприклад, змінювати пароль, аватар або контактну електронну пошту. Адміністратори, в свою чергу, повинні володіти повним набором інструментів для адміністрування акаунтів, включаючи операції створення, читання, оновлення та видалення (CRUD), а також можливість тимчасового блокування користувачів та гнучкого призначення ролей.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		27

Центральним елементом лаунчера є управління каталогом, що надає єдину точку доступу до всіх ігор, серверів та модифікацій. Адміністратори повинні мати інтуїтивно зрозумілі інструменти для додавання нового контенту та редагування метаданих, таких, як описи, жанри, системні вимоги та зображення. Для користувачів необхідно реалізувати ефективні механізми пошуку, сортування та фільтрації, що дозволить їм швидко знаходити бажаний контент.

Система повинна забезпечувати надійні процеси завантаження, встановлення та оновлення ігрового контенту. Лаунчер має коректно завантажувати файли з визначених джерел, перевіряти їх цілісність, встановлювати у вказану користувачем директорію та підтримувати механізми автоматичного або ручного оновлення. Ключовою є функція запуску ігрових процесів. Лаунчер повинен вміти коректно запускати виконувані файли ігор, передаючи їм необхідні параметри командного рядка для автентифікації або застосування специфічних налаштувань.

Для ігор з підтримкою модифікацій, як-от Minecraft, критично важливою є функція управління ігровими профілями. Це дозволить користувачам створювати ізольовані "екземпляри" гри, кожен зі своїм унікальним набором модів та конфігурацій, що унеможливило б конфлікти між ними. Для інформування спільноти передбачена система новин, де адміністратори зможуть публікувати оголошення, які будуть відображатися всім користувачам.

Для реалізації всього перерахованого вище необхідна адміністративна панель - централізований інтерфейс для виконання всіх завдань з управління платформою. Взаємодія між клієнтською та серверною частинами буде здійснюватися через добре документований внутрішній API на базі архітектури REST. Нарешті, для забезпечення універсальності лаунчера необхідна глибока інтеграція з зовнішніми API сервісів Modrinth, CurseForge та Pterodactyl, що дозволить отримувати дані про моди та керувати ігровими серверами безпосередньо з інтерфейсу додатку.

Нефункціональні вимоги визначають атрибути якості та загальні характеристики системи, які є не менш важливими за її функціональність, оскільки безпосередньо впливають на користувацький досвід та довгострокову життєздатність продукту.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		28

Продуктивність є однією з ключових вимог. Система повинна бути чутливою та швидкою, щоб не викликати роздратування у користувачів. Це означає, що інтерфейс користувача має реагувати на дії (кліки, введення тексту) практично миттєво, з затримкою не більше 100-200 мілісекунд. Час відповіді на запити до сервера, наприклад, завантаження списку ігор або новин, не повинен перевищувати 1-2 секунди за умови стабільного інтернет-з'єднання. Серверна частина, розроблена на Rust, повинна бути оптимізована для ефективної обробки великої кількості одночасних запитів без суттєвого зниження швидкості відповіді. Це досягається завдяки асинхронності та багатопотоковості, які є сильними сторонами обраного стека. Важливо також, щоб клієнтський додаток не споживав надмірну кількість ресурсів (CPU, RAM), особливо коли він працює у фоновому режимі, аби не впливати на загальну продуктивність комп'ютера користувача під час гри.

Надійність системи означає її здатність працювати стабільно та передбачувано. Цільовим показником доступності серверної частини (uptime) є 99.9%, що мінімізує час простою сервісу. Для досягнення цієї мети необхідно реалізувати механізми коректної обробки помилок як на клієнті (зрозумілі для користувача повідомлення про помилку), так і на сервері (детальне логування для швидкої діагностики проблем). Обов'язковою є реалізація стратегії резервного копіювання бази даних, наприклад, щоденні автоматичні бекапи з політикою зберігання протягом кількох тижнів, що дозволить відновити дані у випадку апаратного збою або іншого непередбачуваного інциденту.

Юзабіліті (зручність використання) є критично важливим для утримання користувачів. Інтерфейс лаунчера повинен бути інтуїтивно зрозумілим, логічним та послідовним. Новий користувач повинен мати змогу освоїти основні функції, такі, як встановлення та запуск гри, протягом перших 5-10 хвилин використання без необхідності звертатися до документації. Важливо забезпечити чіткий візуальний зворотний зв'язок про стан тривалих операцій (наприклад, індикатори прогресу для завантаження файлів) та результати дій користувача. Дизайн має бути не тільки функціональним, але й естетично привабливим, відповідаючи сучасним тенденціям.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Безпека є абсолютним пріоритетом. Паролі користувачів ні в якому разі не повинні зберігатися у відкритому вигляді. Замість цього вони мають зберігатися у вигляді хеш-сум, отриманих за допомогою стійких до злому алгоритмів, таких, як Argon2 або bcrypt, з використанням унікальної "солі" для кожного користувача. Всі комунікації між клієнтським додатком та сервером повинні відбуватися виключно через захищений протокол HTTPS, що шифрує дані та унеможливорює їх перехоплення. Серверний API та адміністративна панель повинні бути захищені від поширених вебвразливостей, таких, як міжсайтовий скриптинг (XSS) та підробка міжсайтових запитів (CSRF)[17], шляхом ретельної валідації та санітизації всіх вхідних даних.

Масштабованість означає, що архітектура системи повинна бути спроектована з урахуванням майбутнього зростання кількості користувачів та даних. Серверна частина повинна підтримувати горизонтальне масштабування, тобто можливість додавання нових серверних екземплярів для розподілу навантаження. Використання Docker-контейнерів та безстанової архітектури серверного додатку значно спрощує цей процес. База даних також повинна мати потенціал для масштабування, наприклад, шляхом створення реплік для читання, що дозволить розподілити навантаження на запити.

Підтримуваність системи визначає, наскільки легко буде вносити зміни, виправляти помилки та додавати новий функціонал у майбутньому. Це досягається за рахунок чистого, добре структурованого коду, що відповідає прийнятим стандартам кодування (наприклад, офіційним рекомендаціям для мови Rust). Модульна архітектура, де різні частини системи є слабо зв'язаними, дозволяє розробляти та тестувати їх незалежно. Використання системи контролю версій Git є обов'язковим, а наявність вичерпних коментарів у коді та зовнішньої документації (особливо для API) значно спрощує процес підтримки та розвитку проєкту.

Системні вимоги детально описують необхідне апаратне та програмне оточення, яке є обов'язковою умовою для коректної та стабільної роботи як клієнтської, так і серверної частин розробленого лаунчера.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		30

Для клієнтської частини першочерговою вимогою є широка кросплатформна сумісність. Це означає забезпечення повноцінної роботи на актуальних версіях найпоширеніших настільних операційних систем: Windows (зокрема, Windows 10 та 11, 64-розрядні версії), macOS (останні підтримувані версії) та популярних дистрибутивах Linux (таких як Ubuntu, Fedora, Arch Linux, з урахуванням різних середовищ робочого столу, наприклад, GNOME та KDE). Щодо апаратного забезпечення, клієнт повинен мати достатній обсяг оперативної пам'яті (RAM). Мінімально необхідний обсяг становить 2-4 ГБ для базової роботи самого лаунчера, але для комфортного ігрового процесу, коли лаунчер працює у фоні разом із сучасною грою, рекомендовано мати 8 ГБ або більше. Це запобігатиме зниженню продуктивності через активне використання файлу підкачки. Також необхідно мати достатньо вільного місця на диску, яке буде використовуватися не лише для інсталяції самого лаунчера, але й для кешування даних, зберігання логів, завантажених ігор та їхніх модифікацій, обсяг яких може сягати десятків гігабайтів. Крім того, для завантаження контенту, оновлень та взаємодії з сервером необхідне стабільне ширококосмугове підключення до мережі Інтернет.

Для серверної частини зазвичай рекомендується використання операційних систем на базі ядра Linux (наприклад, Ubuntu Server, CentOS). Такий вибір обґрунтований їх доведеною стабільністю, високою продуктивністю, гнучкістю налаштувань, кращими механізмами безпеки та меншими ліцензійними витратами (порівняно з Windows Server). Для розгортання серверного додатку необхідно встановити відповідну версію середовища виконання та компілятор мови Rust. Також потрібно розгорнути обрану систему управління базами даних (PostgreSQL) та налаштувати вебсервер Nginx [3], який буде виконувати ролі зворотного проксі, балансувальника навантаження та сервера для роздачі статичних файлів. Використання технологій контейнеризації, зокрема Docker та Docker Compose [5, 28], є вкрай рекомендованим. Це значно спрощує процес розгортання, оскільки дозволяє "запакувати" додаток та всі його залежності в ізольовані контейнери, забезпечуючи ідентичність середовища на всіх етапах — від розробки до продуктивного використання. Конкретні вимоги до ресурсів сервера (кількість ядер CPU, обсяг RAM, дисковий

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		31

простір) напряду залежать від очікуваного навантаження. Для невеликого проєкту може бути достатньо віртуального сервера з 2 ядрами CPU та 4 ГБ RAM, тоді як для великої платформи з тисячами одночасних користувачів знадобиться значно потужніша інфраструктура з можливістю горизонтального масштабування.

2.3. Проєктування об'єктно-орієнтованої моделі та структури даних

Після визначення функціональних можливостей та вимог до системи, наступним критично важливим кроком є проєктування структури даних та об'єктно-орієнтованої моделі. Цей процес є архітектурним фундаментом, що перетворює абстрактні вимоги на конкретний план реалізації. Він передбачає глибоку ідентифікацію ключових інформаційних сутностей (таких як користувачі, ігри, сервери) та програмних об'єктів, які будуть їх представляти в коді. Для кожної сутності визначаються її атрибути (дані, що вона зберігає) та методи (поведінка, яку вона може виконувати), а також встановлюються логічні зв'язки та взаємодії між ними.

Якісне проєктування на цьому етапі є запорукою створення гнучкої, масштабованої та, що найважливіше, підтримуваної системи. Саме ця модель визначає, як дані будуть організовані в базі даних, як компоненти системи будуть обмінюватися інформацією, і наскільки легко буде додавати новий функціонал у майбутньому. Розроблена модель стає основою для створення схеми бази даних у PostgreSQL та відповідних структур (struct) і трейтів (trait) у серверній частині на Rust, а також станів (stores) та компонентів у клієнтській частині на Svelte.

Для побудови об'єктно-орієнтованої моделі було визначено ключові класи, що відображають основні поняття предметної області. В основі моделі лежить клас User (Користувач), що представляє всіх учасників системи. Він інкапсулює фундаментальні облікові дані (унікальний ідентифікатор, логін, надійно хешований пароль, email), статус (активний, заблокований) та призначені ролі. Його методи (login(), register(), updateProfile(), changePassword()) реалізують базову логіку управління акаунтом. Для розширення даних про користувача та відокремлення публічної інформації від приватної використовується тісно пов'язаний з ним клас Profile (Профіль), який зберігає нікнейм, аватар, ігрові вподобання та агрегує

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		32

список отриманих досягнень (UserAchievement). Система контролю доступу на основі ролей (RBAC) реалізується через класи Role (Роль) та Permission (Дозвіл).

Кожен User має одну або декілька Role (наприклад, "Гравець", "Модератор", "Адміністратор"), а кожна роль, у свою чергу, має набір об'єктів Permission, які надають право виконувати певні дії (наприклад, "публікувати новини" або "банити користувачів").

Ядром контентної частини є клас Game (Гра), що містить вичерпну статичну інформацію про гру: назву, детальний опис, розробника, видавця, жанри, теги, системні вимоги, посилання на медіафайли (обкладинки, скріншоти, трейлери) та інструкції щодо її встановлення та запуску. Управління колекцією всіх доступних ігор здійснює клас-менеджер GameCatalog (Каталог Ігор), який інкапсулює логіку взаємодії з базою даних, надаючи методи для пошуку, фільтрації та сортування ігор. Для підтримки модифікацій, особливо актуальної для таких ігор, як Minecraft, створено два важливих класи. GameInstance (Ігровий Екземпляр) дозволяє користувачам створювати повністю ізольовані конфігурації однієї гри, кожна зі своїм шляхом встановлення, налаштуваннями запуску та унікальним набором модів. Це вирішує проблему конфліктів. Клас Mod (Модифікація) описує конкретний мод, зберігаючи його назву, версію, автора, джерело (Modrinth, CurseForge), сумісність та залежності. Система також оперує сутностями Server (Сервер) для зберігання інформації про доступні ігрові сервери (назва, IP-адреса, порт) та ServerManager (Менеджер Серверів) для управління ними, включаючи моніторинг статусу та інтеграцію з Pterodactyl API. Інформаційні потоки представлені класами News (Новина) та Update (Оновлення), а ігрові досягнення - класом Achievement.

Взаємозв'язки між цими класами визначають архітектуру системи. Звичайний асоціативний зв'язок між User та його GameInstance (один користувач може мати багато ігрових екземплярів). Відношення "частина-ціле" реалізується через агрегацію, коли частини можуть існувати окремо (наприклад, GameCatalog "містить" об'єкти Game, але гра може існувати і поза каталогом), та більш сильну композицію, коли частина не може існувати без цілого (наприклад, UserProfile є невід'ємною частиною User, і при видаленні користувача його профіль також

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		33

знищується). Для створення ієрархій та перевикористання коду використовується успадкування, що дозволяє, наприклад, створити клас AdminUser, який розширює User, додаючи специфічні адміністративні методи (banUser(), manageContent()).

Гнучкість системи підвищується завдяки реалізації інтерфейсів, які визначають контракти поведінки. Інтерфейс Authenticatable може бути реалізований класом User і вимагати наявності методів getId() та validatePassword(), що дозволяє сервісу аутентифікації працювати з будь-яким об'єктом, що задовольняє цей контракт.

Такий підхід, що поєднує успадкування, інтерфейси та абстрактні класи, дозволяє побудувати гнучку та розширювану архітектуру. Наприклад, можна створити абстрактний клас ContentItem з атрибутами "назва", "опис", "дата створення" і реалізувати його в класах Game, Mod та News, щоб уніфікувати операції з різними типами контенту, наприклад, при пошуку. Візуальне представлення всіх описаних класів, їх атрибутів, методів та взаємозв'язків здійснюється за допомогою діаграми класів UML (Рисунок 2.3). Ця діаграма є не просто ілюстрацією, а критично важливим робочим документом і інструментом комунікації між розробниками. Вона слугує як креслення системи, що допомагає мати єдине, узгоджене розуміння її структури перед початком та під час активної фази розробки.

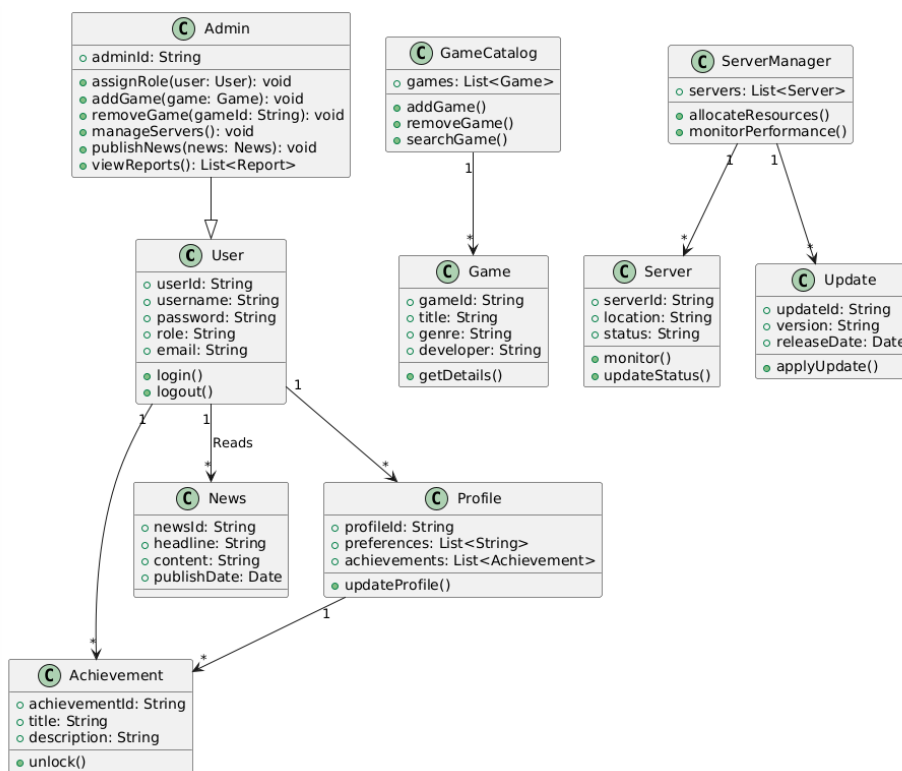


Рис. 2.3 - Діаграма класів системи

2.4 Проєктування бази даних

Для зберігання даних, критично важливих для роботи лаунчера, таких, як інформація про користувачів та глобальні налаштування системи, була спроектована реляційна база даних. Вибір PostgreSQL [11] обґрунтований потребою у забезпеченні гарантій ACID, що є фундаментальним для надійних облікових систем. Це гарантує, що операції, наприклад, реєстрація нового користувача, будуть виконані повністю або не виконані взагалі, запобігаючи частковим та неузгодженим станам даних. Взаємодія з базою даних у програмному коді здійснюється через шар абстракції, відомий як контекст даних (DatabaseContext), що є центральним елементом технологій об'єктно-реляційного відображення (ORM). Цей підхід дозволяє розробникам працювати з таблицями як з колекціями об'єктів у коді, що значно спрощує операції CRUD і відокремлює бізнес-логіку від деталей реалізації SQL-запитів.

Отже, код стає чистішим, більш підтримуваним та менш схильним до помилок типу SQL-ін'єкцій, оскільки ORM-фреймворк бере на себе генерацію параметризованих запитів.

Структура бази даних включає дві основні таблиці: Users та Settings.

Users - таблиця, призначена для зберігання облікових записів користувачів. Вона містить поля для унікального ідентифікатора (Id), логіна (Login), надійно хешованого пароля (Password), електронної пошти (Email) та токена доступу (AccessToken). Особливу увагу приділено безпеці: поле Password зберігає не сам пароль, а його хеш, отриманий за допомогою сильного криптографічного алгоритму з використанням "солі", що унеможливорює його відновлення навіть у випадку компрометації бази даних. Поле AccessToken використовується для підтримки сесій користувачів, дозволяючи їм взаємодіяти з API без повторної передачі логіна та пароля.

Settings – таблиця, яка зберігає глобальні налаштування лаунчера, що дозволяє адміністраторам динамічно керувати конфігурацією системи через адміністративну панель без необхідності вносити зміни в код та перезапускати сервер. Вона зазвичай містить лише один запис. Сюди входять параметри підключення до зовнішніх сховищ файлів (StorageHost, StorageLogin, StoragePassword), ключі

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		35

доступу до API сторонніх сервісів (наприклад, CurseForgeKey), типи сховищ та протоколів (StorageType, TextureProtocol), а також опція, що дозволяє вмикати або вимикати реєстрацію нових користувачів (RegistrationIsEnabled). Це забезпечує гнучкість в управлінні інфраструктурою та інтеграціями.

На рисунку 2.4 представлена UML-діаграма, що візуалізує структуру бази даних та її контекст.

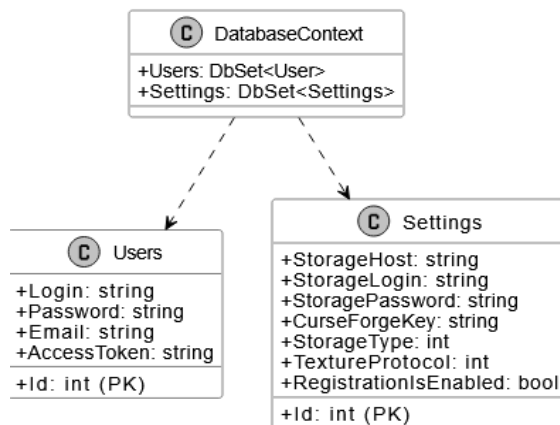


Рис. 2.4 - Схема бази даних

2.5. Моделювання взаємодії та комунікацій

Після визначення статичної структури системи за допомогою діаграми класів, необхідно змоделювати динамічну поведінку – як об'єкти цих класів взаємодіють між собою для реалізації варіантів використання. Для цього в UML використовуються діаграми взаємодії, основними з яких є діаграми послідовності та діаграми комунікації (раніше відомі як діаграми кооперації).

Діаграми послідовностей чітко ілюструють часовий порядок взаємодій між об'єктами. Вони показують об'єкти (або акторів) та вертикальні "лінії життя", що представляють їх існування в часі. Горизонтальні стрілки між лініями життя зображують повідомлення (виклики методів), що надсилаються від одного об'єкта до іншого, в тій послідовності, в якій вони відбуваються. Це робить діаграми послідовності надзвичайно корисними для розуміння потоку керування в конкретному сценарії. Наприклад, можна детально показати кроки, які відбуваються, коли користувач обирає гру для завантаження: від кліку на кнопку в інтерфейсі (UI), через виклик методу в клієнтській логіці (ClientLogic), надсилання запиту до серверного API (ServerAPI), обробку запиту на сервері (GameService, DownloadManager), отримання відповіді та, нарешті, початок процесу завантаження на клієнті. Діаграма

послідовності наочно демонструє, який об'єкт ініціює дію, які методи викликаються та в якому порядку. [Рис. А.1 - Діаграма послідовності]

Діаграми комунікації (співпраці), на відміну від діаграм послідовності, зосереджені на структурній організації об'єктів, які беруть участь у взаємодії, та на зв'язках між ними. Вони також показують обмін повідомленнями, але акцент робиться на відносинах між об'єктами, а не на часовій послідовності. Повідомлення нумеруються, щоб показати порядок, але візуально цей порядок менш очевидний, ніж на діаграмі послідовності. Ці діаграми корисні для розуміння "мережі" комунікацій між об'єктами в рамках певного сценарію, показуючи, які об'єкти безпосередньо взаємодіють один з одним. Наприклад, діаграма комунікації може показати, як дія адміністратора з оновлення каталогу ігор (AdminPanel -> GameCatalogService -> DatabaseAccessor -> GameRepository) впливає на різні компоненти системи та як інформація розповсюджується між ними.

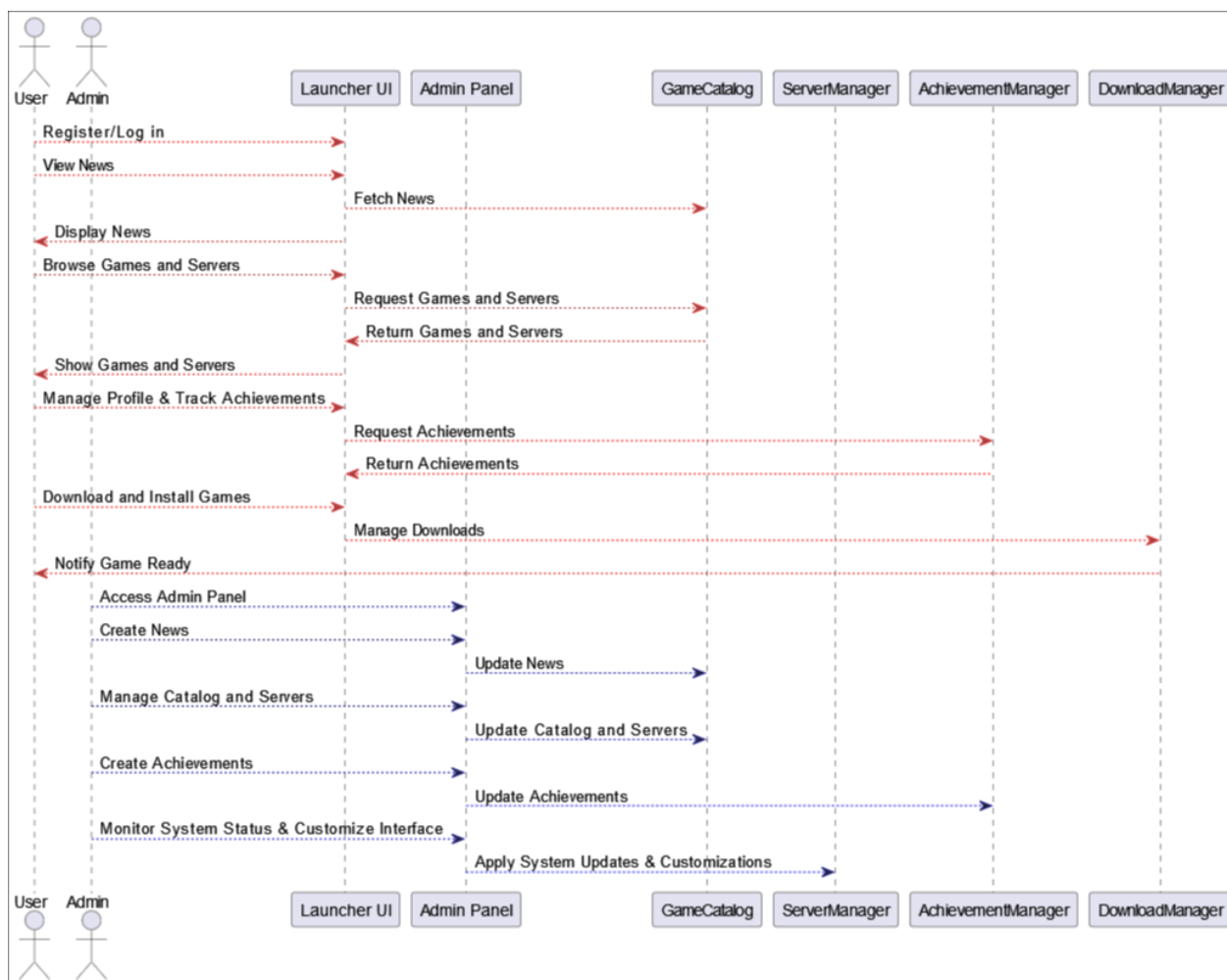


Рис. 2.4 - Діаграма комунікації

Розуміння порядку взаємодії та використання відповідних діаграм UML є фундаментальним для точного моделювання та ефективної комунікації щодо поведінки та структури програмної системи. У контексті розробки ігрового лаунчера, спільне використання діаграм класів (для статичної структури), діаграм послідовності (для часової логіки конкретних сценаріїв) та діаграм комунікації (для візуалізації зв'язків між взаємодіючими об'єктами) дозволяє отримати цілісне та глибоке уявлення про функціонування системи. Це допомагає виявити потенційні проблеми на ранніх етапах проєктування, уточнити відповідність класів та забезпечити узгодженість між різними частинами системи, від взаємодії користувача з інтерфейсом до складних адміністративних завдань та взаємодії з серверною частиною.

2.6. Вибір патернів проєктування та визначення компонентів системи

Після визначення архітектури, класів та їх взаємодій, наступним логічним кроком є вибір відповідних патернів проєктування та деталізація компонентів системи. Патерни проєктування – це перевірені часом, типові рішення для поширених проблем у проєктуванні програмного забезпечення. Їх використання дозволяє створювати більш гнучкий, масштабований, підтримуваний та зрозумілий код, уникаючи "винаходу велосипеда" для стандартних задач. Для кроссплатформного проєкту запуску ігор зі стеком технологій Rust (бекенд), Tauri (оболонка) та Svelte, було прийнято рішення ефективно застосувати декілька ключових патернів, що охоплюють як клієнтську, так і серверну частини.

Для чіткого структурування клієнтської частини, попри те, що Svelte сам по собі є реактивним фреймворком, було доцільно застосувати принципи архітектурного патерна **Model-View-Controller (MVC)** або його варіації **Model-View-ViewModel (MVVM)**. У цій парадигмі, View (Представлення) реалізується компонентами Svelte, що відповідають за візуалізацію даних та обробку вводу користувача. Model (Модель) представляє дані та бізнес-логіку, що з ними пов'язана, включаючи стани, отримані з сервера. Роль Controller/ViewModel виконує посередник, який обробляє логіку представлення, реагує на дії користувача та оновлює Модель і Представлення. Таке чітке розділення відповідальності (separation of concerns)

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				38
Змн.	Арк.	№ докум.	Підпис	Дата		

значно покращує структуру коду, полегшує тестування окремих частин та спрощує подальшу підтримку інтерфейсу.

Для інкапсуляції дій користувача або системних операцій як на клієнтській, так і на серверній стороні, ефективним є патерн **Command (Команда)**. Такі операції, як "Завантажити гру", "Оновити профіль" або "Запустити сервер", реалізуються як окремі об'єкти-команди. Кожен такий об'єкт містить всю необхідну інформацію для свого виконання, що дозволяє параметризувати дії, ставити їх у чергу для послідовного виконання, реалізовувати функціонал скасування та повторення (undo/redo), а також вести детальний журнал операцій. На сервері цей підхід особливо корисний для обробки запитів до API, де кожен запит можна розглядати як команду (Рис. 2.6).

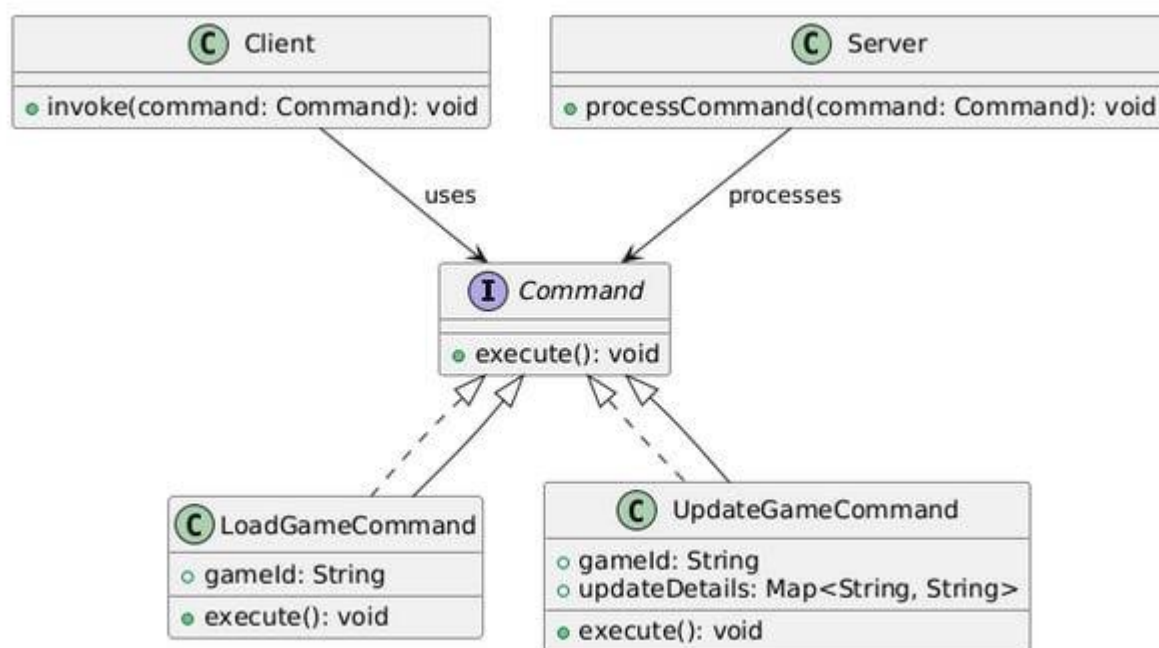


Рис. 2.6 - Діаграма класів, що реалізує Command Pattern

У випадках, коли система повинна підтримувати кілька способів виконання певної задачі, і ці способи можуть обиратися динамічно, застосовується патерн **Strategy (Стратегія)**. Наприклад, у лаунчері можуть існувати різні стратегії завантаження файлів (пряме завантаження з сервера, через P2P-мережу), різні алгоритми перевірки цілісності файлів (MD5, SHA-256) або різні методи автентифікації. Патерн дозволяє інкапсулювати кожен такий алгоритм в окремий клас, роблячи їх

взаємозамінними. Клієнтський код при цьому взаємодіє з абстрактним інтерфейсом стратегії, не залежачи від її конкретної реалізації, що суттєво підвищує гнучкість системи та спрощує додавання нових алгоритмів у майбутньому (Рис. 2.7).

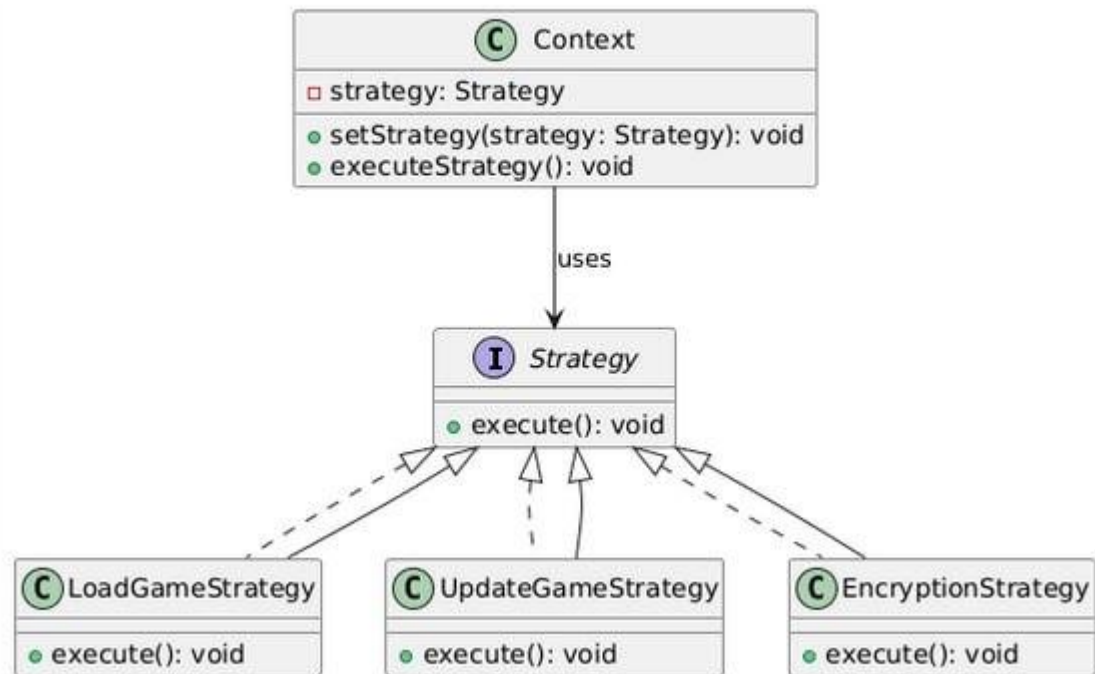


Рис. 2.7 - Діаграма класів, що реалізує Strategy Pattern

Для створення об'єктів без жорсткої прив'язки до їх конкретних класів використовуються породжуючі патерни, такі як **Factory Method (Фабричний метод)** та **Abstract Factory (Абстрактна фабрика)**. Фабричний метод визначає інтерфейс для створення об'єкта, але делегує рішення про те, який саме клас створювати, своїм підкласам. Абстрактна фабрика, у свою чергу, надає інтерфейс для створення цілих сімейств пов'язаних об'єктів. На серверній стороні ці патерни є корисними для створення різних типів відповідей API, об'єктів доступу до даних для різних СУБД або для генерації ігрових профілів з різними початковими конфігураціями. Це сприяє слабкій зв'язності компонентів (loose coupling) і значно полегшує розширення системи новими типами об'єктів (Рис. 2.8).

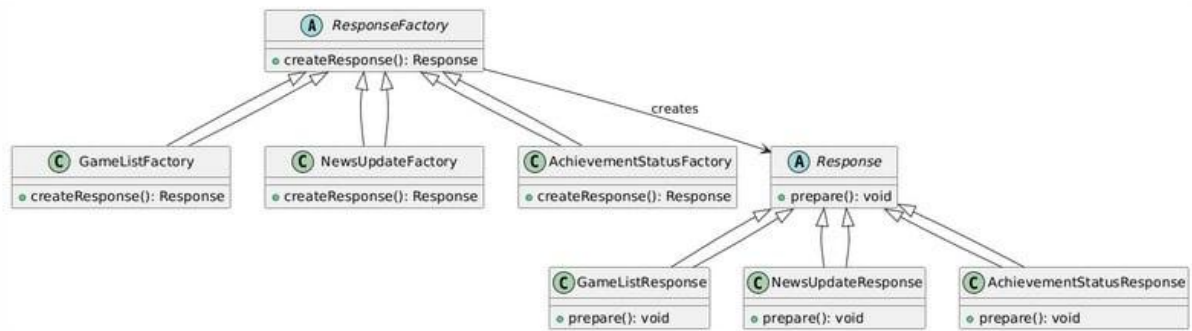


Рис. 2.8 - Діаграма класів, що реалізує Factory Pattern

Для управління глобальними ресурсами або конфігураціями, що повинні існувати в єдиному екземплярі, застосовується патерн **Singleton (Одинак)**. Він гарантує, що клас має лише один екземпляр, і надає глобальну точку доступу до нього. Хоча цей патерн слід використовувати обережно, щоб не створювати неявних залежностей, він є корисним для таких завдань, як управління глобальними налаштуваннями додатку, централізований менеджер логування або єдиний екземпляр сервісу для взаємодії з апаратним забезпеченням. В мові Rust ідіоматичним та безпечним рішенням для реалізації цього патерна є використання спеціалізованих бібліотек, таких як `lazy_static` або `once_cell`.

Реалізація реактивності в клієнтському додатку значно спрощується завдяки патерну **Observer (Спостерігач)**. Він дозволяє об'єктам ("суб'єктам") автоматично сповіщати залежні від них об'єкти ("спостерігачів") про будь-які зміни свого стану. Наприклад, коли оновлюється статус завантаження гри (суб'єкт), відповідні компоненти інтерфейсу (спостерігачі), такі як прогрес-бар або текстовий індикатор, можуть автоматично оновитися, щоб відобразити новий прогрес.

Хоча фреймворк Svelte має потужні вбудовані механізми реактивності, явне використання патерна "Спостерігач" може бути корисним для організації складних потоків даних або взаємодії між різними модулями, що не пов'язані ієрархічно.

Нарешті, для абстрагування шару доступу до даних на серверній стороні широко використовується патерн **Repository (Репозиторій)**. Репозиторій надає інтерфейс, що імітує роботу з колекцією об'єктів (наприклад, `UserRepository`, `GameRepository`). Через цей інтерфейс бізнес-логіка може отримувати, зберігати та

видаляти дані, не маючи жодного уявлення про деталі реалізації сховища (чи це SQL-запити, взаємодія з ORM, чи робота з NoSQL базою). Такий підхід централізує всю логіку доступу до даних, значно покращує можливості тестування (дозволяючи легко підміняти реальні репозиторії на тестові "заглушки") та робить код бізнес-логіки чистішим та незалежним від способу зберігання даних.

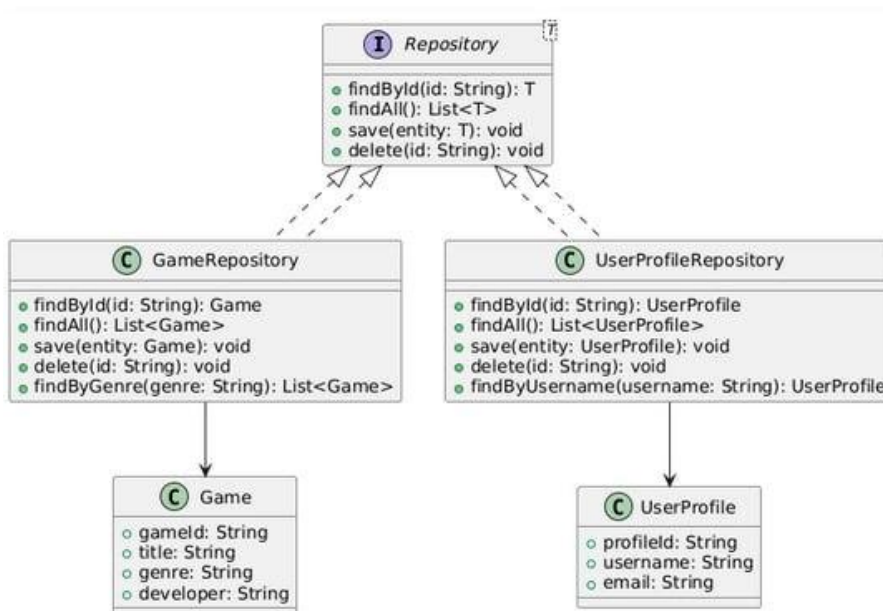


Рис. 2.10 - Діаграма класів, що реалізує Repository Pattern

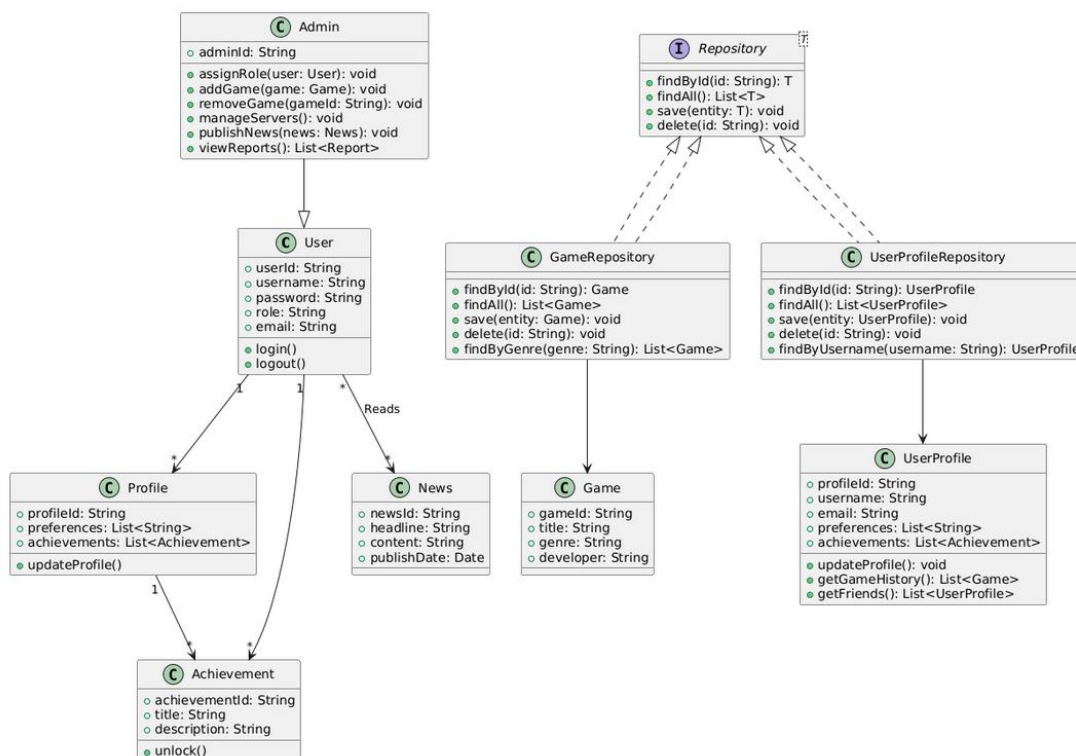


Рис. 2.11 - Діаграма класів

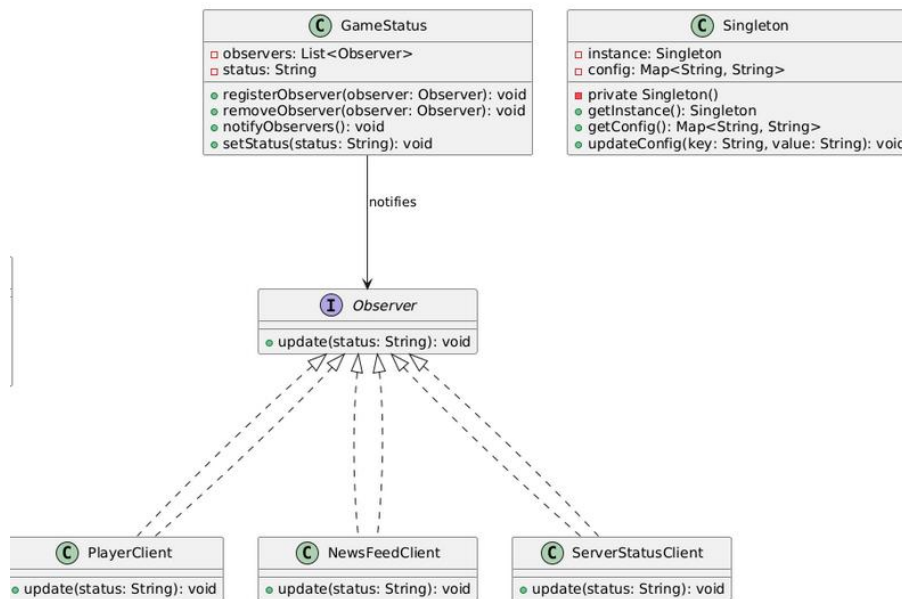


Рис. 2.12 - Діаграма класів

Вибір та правильне застосування цих (та потенційно інших) патернів проектування дозволить створити більш структуровану, гнучку та підтримувану систему. Для клієнтської частини (Tauri/Svelte) ключовим є управління станом UI та ефективна комунікація з бекендом (Rust). Для серверної частини (Rust) важливими є обробка запитів, управління паралелізмом, абстрагування доступу до даних та забезпечення надійності та масштабованості. Визначення компонентів системи:

Визначення компонентів системи

На основі обраного стеку технологій та архітектурних рішень можна визначити основні компоненти системи та їх взаємодію.

Клієнтська частина

Настільна оболонка_забезпечує кросплатформність, доступ до системних API, управління вікнами та взаємодію між веб-ін-терфейсом та Rust-бекендом клієнта. Веб-інтерфейс реалізує користувацький інтерфейс за допомогою компонентів Svelte. Відповідає за відображення даних, обробку введення користувача та взаємодію з клієнтською логікою. Включає компоненти для відображення списку ігор, новин, профілю користувача, налаштувань, індикаторів завантаження тощо.

Управління станом - це модуль або бібліотека (наприклад, вбудовані stores Svelte, Zustand або Redux) для централізованого управління станом додатку (дані користувача, список ігор, стан завантажень).

Клієнтська логіка є частиною логіки, що виконується на клієнті. Може включати валідацію введення, підготовку запитів до сервера, обробку відповідей, управління локальними файлами (завантаження, встановлення, запуск ігор – часто делегується Rust-частині Tauri).

Мережевий модуль відповідає за комунікацію з серверною частиною через визначений API (HTTP/HTTPS, WebSocket).

Серверна частина

Обробники API - модулі, що відповідають за прийом та обробку запитів від клієнта (наприклад, з використання фреймворків Actix-Web, Axum або Warp). Валідують вхідні дані, викликають відповідні сервіси бізнес-логіки та формують відповіді.

Шар бізнес-логіки містить основну логіку додатку: управління користувачами, автентифікація, управління каталогом ігор, обробка замовлень на завантаження, логіка досягнень тощо. Взаємодіє з шаром доступу до даних.

Шар доступу до даних абстрагує взаємодію з базою даних. Реалізує патерн "Репозиторій" для роботи з сутностями (User, Game, Mod тощо). Використовує ORM (наприклад, Diesel) або бібліотеки для роботи з SQL/NoSQL (наприклад, SQLx, mongodb).

База даних застосовується для зберігання персистентних даних системи, а модулі інтеграції - компоненти для взаємодії з зовнішніми API (Modrinth, CurseForge, Pterodactyl). Фонові служби можуть використовуватися для виконання періодичних завдань, таких, як перевірка оновлень, синхронізація даних, обробка черг завдань.

Висновки до другого розділу

У другому розділі було проведено етап проектування кросплатформного ігрового лаунчера, який є логічним продовженням аналізу, виконаного в першому розділі. Було детально визначено основних акторів системи – Користувача та

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Адміністратора – та описано їх ключові варіанти використання платформи, що дозволило сформулювати чітке уявлення про очікувану функціональність.

На основі аналізу варіантів використання та потреб цільової аудиторії були сформульовані детальні функціональні, нефункціональні та системні вимоги. Функціональні вимоги окреслили конкретні можливості системи, такі, як управління акаунтами, каталогом ігор та модів, процеси завантаження, встановлення та запуску ігор, а також адміністративні функції. Нефункціональні вимоги визначили ключові атрибути якості, зокрема, продуктивність, надійність, юзабіліті, безпеку, масштабованість та підтримуваність. Системні вимоги встановили необхідне апаратне та програмне забезпечення для клієнтської та серверної частин.

Було розроблено попередню об'єктно-орієнтовану модель системи та визначено основні сутності даних (User, Game, Mod, Server тощо), їх атрибути та взаємозв'язки. Ця модель слугує основою для проєктування бази даних та реалізації структур даних у програмному коді.

Розглянуто підходи до моделювання динамічної взаємодії між об'єктами системи за допомогою діаграм послідовності та комунікації UML, що дозволяє візуалізувати потоки керування та обмін повідомленнями в рамках ключових сценаріїв роботи лаунчера.

На завершення, було проаналізовано доцільність застосування поширених патернів проєктування (MVC/MVVM, Command, Strategy, Factory, Singleton, Observer, Repository) для покращення структури, гнучкості та підтримуваності коду як клієнтської, так і серверної частин. Також було визначено основні логічні компоненти системи (UI, управління станом, клієнтська логіка, мережевий модуль, обробники API, бізнес-логіка, шар доступу до даних, база даних, інтеграції) та описано загальну архітектуру розгортання.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		45

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З КРОСПЛАФОРМНИМ ЛАУНЧЕРОМ ДЛЯ ІГОР

3.1. Розробка

Процес розробки програмного комплексу було розділено на два основні напрямки: створення серверної частини та клієнтського застосунку, що взаємодіють між собою через розроблений REST API.

Серверну частину було реалізовано мовою програмування Rust з використанням веб-фреймворку Actix-Web, що відомий своєю високою продуктивністю та ефективним використанням ресурсів завдяки своїй асинхронній архітектурі, побудованій на базі Tokio. Архітектура бекенду була побудована за багатошаровим принципом, що забезпечує чітке розділення відповідальності та полегшує подальшу підтримку та розширення системи.

Шар API розроблений відповідно до принципів RESTful архітектури, є вхідною точкою для всіх комунікацій з клієнтської сторони. Було реалізовано структурований набір кінцевих точок (endpoints) для обробки HTTP-запитів. Сюди входять публічні та захищені маршрути. Маршрути для аутентифікації, такі, як POST /api/v1/users/signup та POST /api/v1/users/signin, відповідають за реєстрацію нових користувачів та видачу JWT-токенів для існуючих. Інші кінцеві точки, як-от GET /api/v1/profiles для отримання ігрових профілів або GET /api/v1/integrations/news/list для новин, забезпечують доступ до публічних даних.

Функції управління, такі, як створення та оновлення профілів (POST /api/v1/profiles, PUT /api/v1/profiles) та керування плагінами (POST /api/v1/plugins/install), є захищеними і вимагають ролі адміністратора. Захист маршрутів реалізовано за допомогою проміжного програмного забезпечення (middleware) в Actix-Web, яке перевіряє наявність та валідність JWT-токена в заголовку Authorization. Токен містить інформацію про ідентифікатор користувача та його роль, що дозволяє гнучко контролювати доступ до різних ресурсів.

Особливу увагу було приділено безпеці та цілісності даних. Для валідації всіх вхідних даних використовувалася комбінація бібліотек для десеріалізації JSON у структури Rust та validator для декларативного визначення правил валідації. Для

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		46

об'єктів передачі даних (DTO), таких, як ProfileCreateDto або UserCreateDto, були реалізовані валідатори, що перевіряють формат email, довжину пароля, наявність обов'язкових полів тощо. Це унеможливило потрапляння некоректних даних у бізнес-логіку системи.

Шар сервісів інкапсулює основну логіку застосунку, діючи як посередник між шаром API та шаром доступу до даних. Кожен сервіс відповідає за певний бізнес-домен. AuthIntegrationHandler та AuthService містять логіку, пов'язану з автентифікацією через різні платформи, а також з реєстрацією та входом користувачів, включаючи генерацію JWT-токенів та хешування паролів за допомогою криптографічно стійкого алгоритму Argon2. ProfileHandler та ProfileService відповідають за створення, оновлення, компіляцію та отримання інформації про ігрові профілі. GitHubService, у свою чергу, містить логіку для роботи з репозиторіями GitHub, що може використовуватися для отримання даних про плагіни чи оновлення. Такий підхід, що базується на принципі єдиної відповідальності, дозволив зберегти чітке розділення обов'язків, полегшив тестування бізнес-логіки в ізоляції (шляхом мокування залежностей) та підвищив загальну підтримуваність коду.

Шар доступу до даних. Для взаємодії з базою даних PostgreSQL [11] було обрано бібліотеку sqlx, яка забезпечує асинхронну роботу та перевірку SQL-запитів на етапі компіляції, що значно зменшує ризик помилок під час виконання. Були створені репозиторії для кожної сутності (UserRepository, SettingsRepository), які інкапсулюють всю логіку взаємодії з базою даних. UserRepository містить методи create_user, find_user_by_email, update_user_role тощо. Цей підхід дозволив абстрагувати виконання SQL-запитів та надати зручний і типізований інтерфейс для шару сервісів, повністю відокремивши бізнес-логіку від деталей роботи з базою даних. Управління пулом з'єднань до бази даних здійснюється за допомогою sqlx::PgPool, екземпляр якого створюється при старті додатку і передається в стан Actix-Web, що робить його доступним для всіх обробників запитів.

Адміністративна панель, як ключовий інструмент управління платформою, була реалізована з урахуванням потреб у гнучкості, швидкодії та зручності використання. Вона побудована на трьох основних компонентах: сучасному

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		47

користувачькому інтерфейсі на базі Svelte та shadcn-svelte, надійних механізмах взаємодії з серверним API та глибокій інтеграції зі сторонніми сервісами, що є критично важливими для функціонування лаунчера.

Користувачький інтерфейс адміністративної панелі було створено з використанням фреймворку Svelte. Вибір цієї технології був зумовлений її компілятивною природою: Svelte не виконується у браузері користувача як традиційні фреймворки, а натомість компілює компоненти у високоефективний, оптимізований імперативний JavaScript-код ще на етапі збірки. Це забезпечує мінімальний розмір кінцевого пакету та надзвичайно високу швидкість відгуку інтерфейсу, що є критично важливим для панелі, яка може оперувати великими обсягами даних. Для прискорення розробки та забезпечення єдиного, сучасного візуального стилю було використано бібліотеку компонентів shadcn-svelte. Вона надає набір готових, доступних (accessible) та легко кастомізованих будівельних блоків для інтерфейсу.

В рамках розробки було створено цілий ряд реактивних Svelte-компонентів, що стали основою UI:

Були розроблені спеціалізовані інформаційні картки, такі, як ProfileCard для відображення ключової інформації про ігровий профіль (назва, версія, дата створення), GameServerCard для моніторингу статусу ігрових серверів та IntegrationCard для управління інтеграціями. Кожна картка містить не тільки інформацію, а й елементи керування, наприклад, кнопки для редагування чи видалення.

Для відображення списків гравців, профілів та файлів були реалізовані складні табличні компоненти (інтерактивні таблиці). Вони підтримують клієнтську та серверну пагінацію, сортування по будь-якій колонці, а також текстову фільтрацію в реальному часі. Це дозволяє адміністраторам ефективно працювати з великими наборами даних, швидко знаходячи потрібну інформацію.

Модальні вікна та діалоги. Для всіх операцій створення та редагування сутностей (профілів, ігрових серверів, налаштувань автентифікації) використовуються модальні вікна. Це дозволяє уникнути перезавантаження сторінки і забезпечує плавний користувачький досвід.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		48

Форми з валідацією. Усі форми вводу даних побудовані з використанням бібліотеки zod для валідації схем. Це дозволяє визначати правила валідації на рівні схеми даних і автоматично відображати змістовні повідомлення про помилки користувачу в реальному часі, ще до відправки даних на сервер, що значно покращує UX та зменшує кількість некоректних запитів.

Управління станом: для керування станом застосунку було використано вбудовані Svelte stores (writable, readable, derived). Це дозволило декларативно описувати стан і реактивно оновлювати UI при його зміні, для відображення статусу профілю або списку сповіщень. Для управління серверним станом (кешування, синхронізація) було використано @tanstack/svelte-query.

Розробка клієнтської частини лаунчера. Одним з найважливіших етапів проєкту була розробка клієнтського застосунку, з яким безпосередньо взаємодіє кінцевий користувач,. Цей процес можна умовно розділити на три взаємопов'язані частини: створення користувацького інтерфейсу, що забезпечує візуальну складову та взаємодію; реалізація фонові логіки, що відповідає за всі "невидимі" операції та інтеграція із зовнішніми сервісами для розширення функціональності.

Клієнтський застосунок було створено з використанням фреймворку Tauri, який дозволив досягти оптимального поєднання переваг сучасних веб-технологій для побудови інтерфейсу та високої продуктивності й безпеки мови Rust для виконання фонових завдань.

Користувацький інтерфейс було розроблено на Svelte, сучасному компіляторі для створення реактивних користувацьких інтерфейсів. Було створено набір компонентів для відображення списків ігрових профілів, де кожен елемент показує назву сервера, його статус та кількість гравців онлайн; сторінок профілю користувача з можливістю редагування даних та перегляду статистики; динамічної стрічки новин, що завантажується з сервера; детальних налаштувань самого лаунчера, що дозволяють користувачу конфігурувати параметри завантаження, зовнішній вигляд та інші аспекти; а також окремої сторінки для управління модифікаціями, де користувач може переглядати, встановлювати та видаляти опціональні моди.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				49
Змн.	Арк.	№ докум.	Підпис	Дата		

Для управління станом застосунку використовувалися вбудовані в Svelte механізми, відомі як stores. Це дозволило ефективно синхронізувати дані між різними, навіть непов'язаними, компонентами. Наприклад, стан аутентифікації користувача зберігався у writable store, що дозволяло іншим компонентам, таким, як навігаційна панель, реактивно оновлюватися, відображаючи або приховуючи певні елементи залежно від того, чи увійшов користувач в систему.

Комунікація з серверною частиною здійснювалася через HTTP-запити до розробленого REST API. Для взаємодії з локальною файловою системою, запуску ігор та інших системних операцій використовувався механізм команд Tauri. Було створено набір асинхронних Rust-функцій, які інкапсулювали складну логіку і викликалися з JavaScript-коду через безпечний IPC-міст, наданий Tauri. Це дозволило виконувати ресурсомісткі або системно-залежні операції, такі, як перевірка файлів гри на цілісність шляхом порівняння їх хеш-сум із даними з сервера, завантаження відсутніх або оновлених файлів з відображенням прогресу, запуск ігрового процесу з передачею необхідних параметрів командного рядка токена автентифікації, а також робота з локальним сховищем для збереження налаштувань та кешованих даних користувача. Такий підхід дозволив виконувати ці операції у продуктивному та безпечному Rust-оточенні, ізольованому від веб-контенту.

Інтеграції: для розширення функціональності лаунчера було реалізовано модулі для взаємодії з декількома зовнішніми сервісами.

Було створено клієнтів для API таких платформ, як Modrinth та CurseForge. Це дозволило отримувати актуальну інформацію про модифікації та модпаки, включаючи їх описи, версії та залежності, а також реалізувати функціонал для встановлення та управління опціональними модами безпосередньо з інтерфейсу лаунчера.

Крім того, було реалізовано інтеграцію з Discord через функціонал Rich Presence, що дозволяє відображати детальний статус гравця в його профілі Discord (наприклад, назву гри, сервер, на якому він грає, тощо).

Для моніторингу та відстеження помилок у роботі застосунку на стороні клієнта було інтегровано Sentry SDK. Це дозволяє розробникам отримувати автоматичні

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		50

звіти про збої та винятки, що виникають у користувачів, що значно прискорює процес діагностики та виправлення проблем.

Такий комплексний підхід до розробки клієнтської частини дозволив створити надійну, безпечну та продуктивну систему, де кожен компонент виконує свою чітко визначену роль, забезпечуючи при цьому зручний та функціональний користувацький досвід.

3.2. Інтерфейс та порядок роботи

Інтерфейс лаунчера було розроблено з акцентом на інтуїтивну зрозумілість, мінімалізм та простоту використання, аби забезпечити комфортну та ефективну взаємодію як для звичайних гравців, так і для адміністраторів. Дизайн-філософія полягала у зменшенні когнітивного навантаження на користувача шляхом пропонування чітко структурованої навігації та логічно згрупованих функцій. Кожен елемент інтерфейсу був розміщений з урахуванням типових сценаріїв використання, щоб мінімізувати кількість кліків, необхідних для виконання ключових дій, таких, як запуск гри або підключення до сервера.

При першому запуску користувач бачить екран авторизації (Рис. 3.1), який є єдиною точкою входу до системи. Тут він може увійти в існуючий акаунт, ввівши логін та пароль, або зареєструвати новий. Процес реєстрації є максимально спрощеним, вимагаючи введення лише унікального імені користувача, адреси електронної пошти та паролю. З метою забезпечення безпеки пароль ніколи не зберігається у відкритому вигляді; натомість він проходить через процес одностороннього хешування з використанням сучасних криптографічних алгоритмів Argon2 перед збереженням в базі даних. Це гарантує, що навіть у випадку компрометації бази даних злоумисники не зможуть отримати доступ до паролів користувачів.

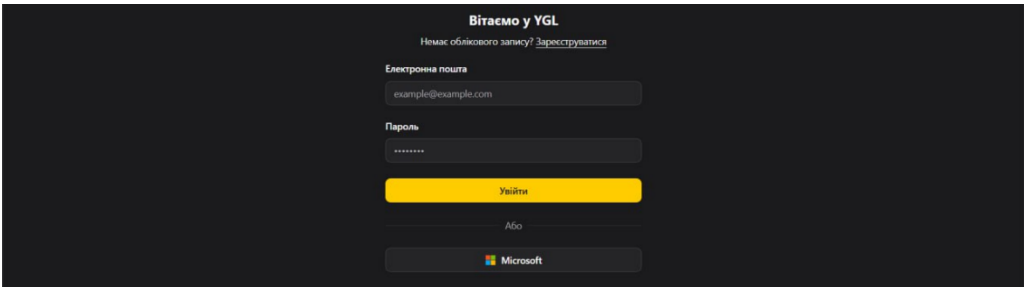


Рис. 3.1 – Макет сторінки авторизації

Після успішної авторизації користувач потрапляє на головну сторінку лаунчера (Рис. 3.2). Вона спроектована як динамічний інформаційний хаб, що одразу занурює гравця у життя ігрової спільноти. Центральне місце займають інтерактивні блоки: блок останніх новин проєкту інформує про важливі оновлення, заплановані події чи турніри. Список рекомендованих ігор та серверів не є статичним; він може персоналізуватися на основі ігрових уподобань користувача та показувати поточний статус сервера та кількість гравців онлайн, що дозволяє приймати рішення про приєднання до гри. Стрічка активності в реальному часі може демонструвати не лише досягнення друзів, а й значущі події спільноти, що сприяє соціальній взаємодії та відчуттю причетності. Вертикальна навігаційна панель зліва є константою інтерфейсу, що забезпечує миттєвий та передбачуваний доступ до ключових розділів: персоналізованої бібліотеки ігор, загального списку серверів з фільтрацією та пошуком, профілю користувача для перегляду статистики та налаштувань, а також розділу налаштувань самого лаунчера.

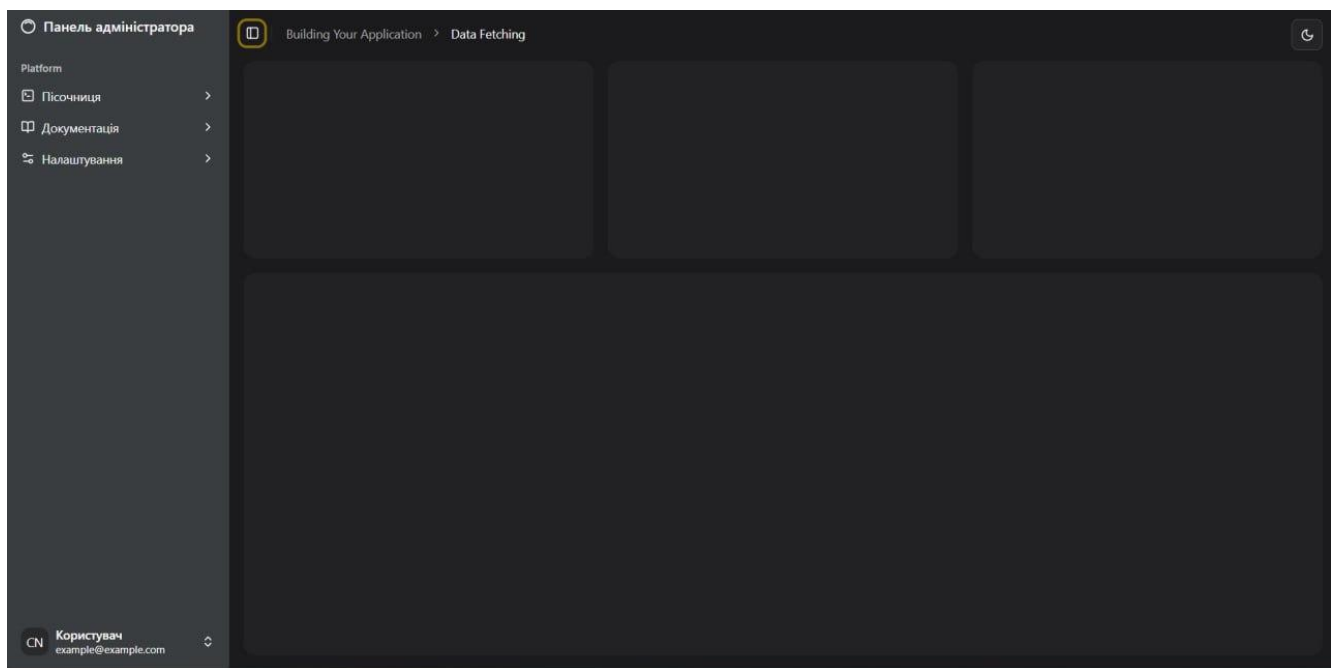


Рис. 3.2 - Макет головної сторінки

Сторінка конкретної гри є вичерпним джерелом інформації та центром керування. Окрім опису, медіа-контенту та системних вимог, тут реалізовано потужний функціонал управління модами. Глибока інтеграція з API платформ Modrinth та CurseForge дозволяє користувачам переглядати, шукати та встановлювати

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		52

модифікації в один клік. Система автоматично аналізує метадані моду, перевіряючи його залежності від інших модів чи версії гри, та попереджає користувача про можливі конфлікти ще до встановлення. Ключовою особливістю є можливість створення ізольованих ігрових профілів (модпаків). Наприклад, користувач може мати профіль "Ваніль+" з кількома модами для покращення якості життя, профіль "Технічний" з десятками індустріальних модів і профіль "Магічний" з відповідними доповненнями. Кожен профіль має свій окремий набір конфігураційних файлів та модів, що дозволяє легко та безпечно перемикатися між абсолютно різними ігровими досвідами без необхідності ручного копіювання файлів.

Для адміністрування платформи було свідомо реалізовано два окремі, але взаємодоповнюючі інструменти, щоб розмежувати стратегічні та оперативні завдання. Перший інструмент — це веб-панель адміністрування, яка є централізованим рішенням для стратегічного управління платформою. Доступна через будь-який браузер, вона надає захищений інтерфейс для задач, що не вимагають негайної реакції. В її межах адміністратор може виконувати широкий спектр дій. Наприклад, у розділі управління користувачами він може не лише переглядати список і блокувати порушників, а й гнучко призначати ролі модераторам, які отримують право редагувати контент, але не мають доступу до налаштувань серверів. Управління контентом дозволяє через зручні форми додавати нові ігри, писати й публікувати новини, редагувати описи ігрових серверів. Розділ аналітики надає ключові метрики для розуміння стану платформи, такі, як графіки щоденних активних користувачів, найпопулярніші ігри та сервери, що є безцінним для прийняття бізнес-рішень.

Другим інструментом є розширена функціональність у самому лаунчері, яка орієнтована на оперативне управління ігровою інфраструктурою. Отримавши відповідні права, адміністратор бачить у своєму лаунчері додаткові елементи керування. Ключовою особливістю є пряма інтеграція з API панелі Pterodactyl. Це дозволяє адміністратору, не перемикаючись між програмами, виконувати миттєві дії. Наприклад, отримавши скаргу на роботу сервера, адміністратор може безпосередньо з лаунчера переглянути його консоль в реальному часі, щоб діагностувати

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		53

помилку, і одразу ж перезавантажити сервер. Такий підхід, що відокремлює управління контентом (веб) від оперативного управління серверами (лаунчер), підвищує безпеку, оскільки не надає оперативного доступу до серверів через загальнодоступний веб-інтерфейс, та значно підвищує ефективність роботи адміністраторів у кризових ситуаціях.

3.3. Процес встановлення та налаштування параметрів платформи

Процес розгортання та налаштування розробленого кросплатформного лаунчера є комплексним завданням, що вимагає конфігурації як клієнтської, так і серверної частин системи для забезпечення їх коректної та стабільної взаємодії. Архітектура системи була свідомо спроектована модульною, аби бути гнучкою, безпечною та масштабованою, що є критично важливим для довгострокового розвитку та підтримки продукту.

Клієнтська частина лаунчера, з якою безпосередньо взаємодіє користувач, розповсюджується у вигляді скомпільованого настільного додатку, створеного за допомогою фреймворку Tauri. Цей вибір був зумовлений кількома ключовими перевагами над аналогами, такими, як Electron. По-перше, Tauri створює значно менші за розміром та більш продуктивні додатки, оскільки він використовує системний веб-переглядач замість вбудовування повноцінного браузера Chromium. По-друге, що є надзвичайно важливим, основна логіка клієнта та його бекенд написані на Rust, а не на JavaScript. Це не тільки підвищує швидкодію, але й гарантує безпеку роботи з пам'яттю, що робить додаток більш стабільним та захищеним від багатьох класів вразливостей. Завдяки Tauri, лаунчер забезпечує нативну роботу на трьох основних операційних системах: Windows, macOS та Linux, надаючи користувачам відчуття роботи зі звичайною програмою. Для встановлення користувачу достатньо завантажити єдиний інсталяційний файл для своєї ОС, що не вимагає попереднього встановлення жодних додаткових залежностей, як-от Node.js, що значно спрощує процес першого знайомства з платформою.

Серверна частина, що є ядром системи та відповідає за всю бізнес-логіку, розгортається за допомогою технології контейнеризації Docker. Такий підхід інкапсулює серверний додаток на Rust разом з усіма його залежностями в ізольований,

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		54

портативний контейнер. Це гарантує, що застосунок буде працювати в однаковому та консистентному середовищі, незалежно від специфіки хостингу, повністю вирішуючи класичну проблему "у мене на комп'ютері все працює". Для управління розгортанням та оркестрації контейнерів може бути використаний як стандартний інструмент Docker Compose [5, 28] для простих налаштувань, так і більш просунуті системи, наприклад, панель управління Pterodactyl. Хоча Pterodactyl переважно відомий в спільноті як інструмент для ігрових серверів, його архітектура, заснована на Docker, дозволяє адаптувати його для управління довільними додатками, надаючи зручний веб-інтерфейс для адміністрування, моніторингу та управління життєвим циклом контейнерів. Крім основного API для клієнтської частини, серверний додаток також обслуговує захищену веб-панель для управління контентом та користувачами платформи.

Ключовим компонентом інфраструктури є веб-сервер Nginx [3], який виступає у ролі зворотнього проксі (reverse proxy) перед серверним додатком. Це є стандартною архітектурною практикою, що надає значні переваги у безпеці, продуктивності та гнучкості. Всі запити з Інтернету спочатку надходять на Nginx, який вже потім перенаправляє їх до розробленого мною Rust-додатку. Цей підхід виконує декілька функцій: по-перше, обробка та термінація HTTPS-з'єднань, що знімає навантаження з основного додатку та централізує управління SSL-сертифікатами; по-друге, розподіл (балансування) навантаження, що дозволяє в майбутньому запускати кілька екземплярів серверного додатку та рівномірно розподіляти запити між ними для забезпечення високої доступності та продуктивності; по-третє, ефективна роздача статичних файлів (зображення, стилі, скрипти), оскільки Nginx робить це значно швидше, ніж сервер додатків.

Загальна схема розгортання, представлена на рисунку 3.1, ілюструє фізичну архітектуру системи та її розміщення на серверах.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		55

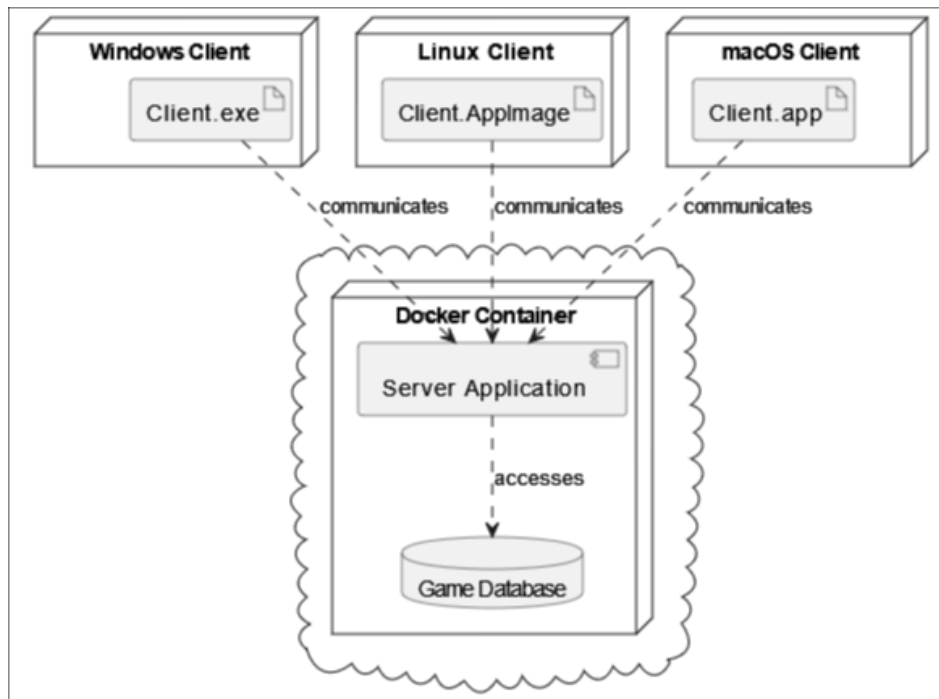


Рис. 3.3 - Схема розгортання

Схема взаємодії компонентів системи

Якщо діаграма розгортання (Рис. 3.3) фокусується на фізичному розміщенні артефактів, тобто, на яких серверах чи вузлах будуть виконуватися частини програми, то діаграма компонентів (Рис. 3.4) переносить увагу на логічну архітектуру програмного комплексу. Вона є концептуальною картою системи, що абстрагується від деталей реалізації та обладнання. Ця діаграма візуалізує систему як набір взаємопов'язаних "чорних скриньок" — компонентів, кожен з яких має чітко визначену відповідальність. Ключовим елементом тут є поняття інтерфейсів, які виступають формальними контрактами, що визначають, як компоненти взаємодіють між собою. Завдяки візуалізації цих компонентів, їх інтерфейсів та залежностей, діаграма дозволяє системним архітекторам і розробникам отримати цілісне розуміння потоків даних та управління, а також оцінити вплив потенційних змін в одному модулі на всю систему загалом.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		56

звертається до зовнішніх сервісів (External Services), таких, як Modrinth, CurseForge та Pterodactyl, через їх API для отримання інформації про модифікації та управління ігровими серверами.

Такий поділ на компоненти забезпечує низьку зв'язність та високу згуртованість системи. Наприклад, компонент бази даних може бути замінений з PostgreSQL [11] на іншу СУБД без необхідності внесення змін у клієнтський додаток, оскільки зміни торкнуться лише реалізації інтерфейсу Database Access у межах серверного додатку.

3.4. Тестування програмного забезпечення

Тестування розробленого програмного продукту є невід'ємним та критично важливим етапом життєвого циклу розробки, який забезпечує якість та надійність кінцевого рішення. Його мета виходить далеко за рамки простого виявлення програмних помилок і охоплює комплексну перевірку відповідності продукту заявленим функціональним вимогам, оцінку його продуктивності під навантаженням та, що є ключовим, валідацію позитивного користувацького досвіду. У контексті даного проєкту особливу увагу було приділено тестуванню юзабіліті, оскільки саме від зручності, інтуїтивної зрозумілості та ефективності інтерфейсу залежить, чи зможуть користувачі легко взаємодіяти з платформою, і чи сформується у них лояльність до продукту.

Для проведення тестування юзабіліті ще на етапі проєктування було обрано інноваційний сервіс Attention Insight. Цей інструмент використовує передові алгоритми штучного інтелекту, що були навчені на величезних масивах даних про рухи очей реальних користувачів, для створення високоточних предиктивних моделей уваги. Такий підхід, відомий як предиктивне відстеження погляду (predictive eye-tracking), дозволяє з високою достовірністю спрогнозувати, на які саме елементи дизайну користувачі звернуть увагу в перші кілька секунд взаємодії, і в якій послідовності. Це надає унікальну можливість отримати швидкий, об'єктивний та економічно ефективний зворотний зв'язок щодо розроблених макетів ще до написання першого рядка коду. Таким чином, вдається проактивно виявляти потенційні

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				58
Змн.	Арк.	№ докум.	Підпис	Дата		

проблеми в інформаційній ієрархії, невдалому розміщенні елементів та загальній візуальній комунікації, запобігаючи дорогим виправленням на пізніх стадіях.

В рамках тестування було проведено поглиблений аналіз двох ключових екранів, що є першими точками контакту користувача з лаунчером: сторінки авторизації та головного меню. Для кожного макету було згенеровано та проаналізовано три типи візуалізацій: теплові мапи, мапи фокусування та мапи контрастності.

Детальний аналіз тестування сторінки авторизації. Теплова мапа (Рис. 3.5) чітко продемонструвала, що найвища концентрація уваги (червоні та жовті зони) прогнозовано припадає на ключові інтерактивні елементи: поля для введення логіну та пароля, а також на основну кнопку "Увійти". Це є ідеальним сценарієм, оскільки він підтверджує, що дизайн ефективно спрямовує користувача до виконання цільової дії, не відволікаючи його на другорядні елементи, такі як фонове зображення.



Рис. 3.5 - Температурна Мапа сторінки авторизації

Мапа фокусування (Рис. 3.6), що імітує людський периферичний зір, доповнила ці дані. Вона показала, що основні елементи форми входу залишаються чіткими та розбірливими у фокусі, тоді як менш важливі частини інтерфейсу візуально розмиваються. Це свідчить про добре продуману візуальну ієрархію, яка допомагає

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		59

користувачеві миттєво ідентифікувати головні елементи керування та уникнути когнітивного перевантаження.

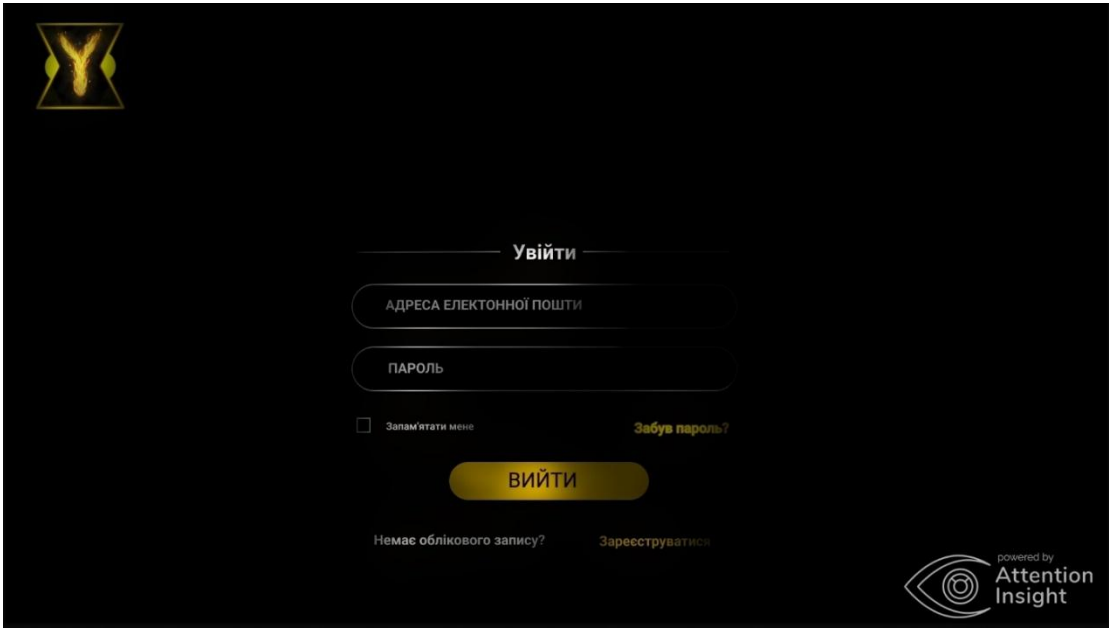


Рис. 3.6 - Мапа Фокусування сторінки авторизації

Мапа контрастів (Рис. 3.7) була використана для оцінки доступності дизайну, що є важливим аспектом для забезпечення інклюзивності. Аналіз показав, що всі текстові елементи та кнопки мають достатній коефіцієнт контрастності відносно фону, що відповідає рекомендаціям WCAG (Web Content Accessibility Guidelines) та забезпечує їх легку читабельність для людей з вадами зору. Загальна оцінка чіткості дизайну (Clarity Score), яку сервіс розраховує на основі аналізу візуального безладу та групування елементів, склала 81 бал зі 100. Це високий показник, що свідчить про те, що інтерфейс сприймається як чистий, організований та зрозумілий для нового користувача.

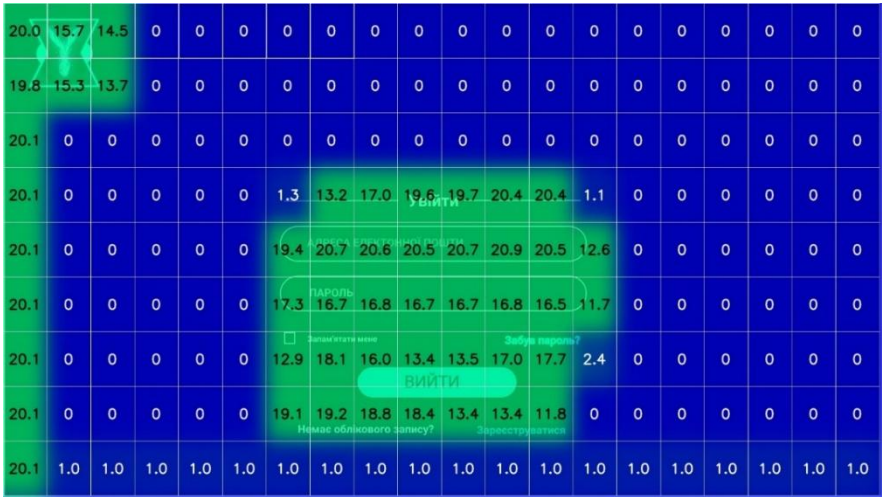


Рис 3.7 - Мапа контрастів сторінки авторизації

Детальний аналіз тестування головного меню. Аналогічний глибокий аналіз було проведено і для макету головної сторінки. Теплова мапа (Рис. 3.8) показала збалансований розподіл уваги. Користувачі, згідно з прогнозом, звертають увагу як на центральну контентну частину, де розміщені новини та огляди ігор (що стимулює залученість), так і на навігаційну панель зліва (що забезпечує орієнтацію в системі). Такий розподіл є позитивним сигналом, оскільки вказує на те, що користувач одночасно і споживає контент, і розуміє, як отримати доступ до інших функцій. Якби увага була сконцентрована виключно на одній області, це могло б свідчити про те, що інша є непомітною або нецікавою для користувача.

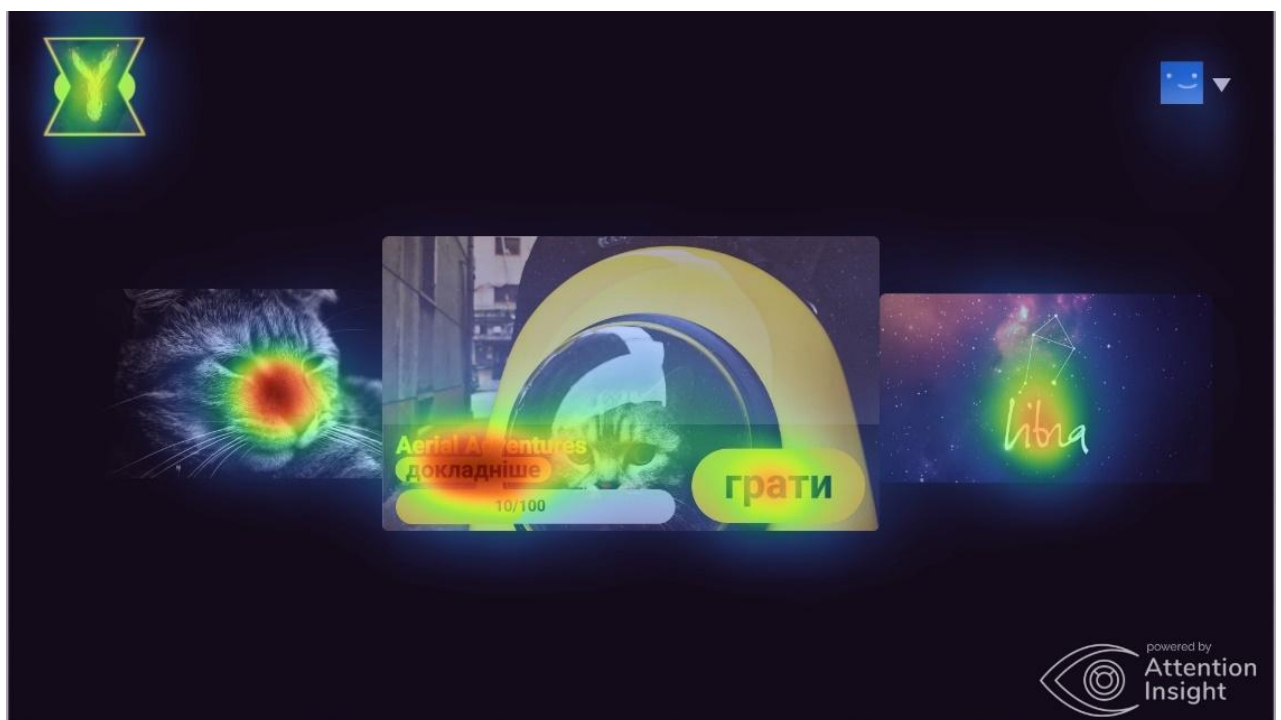


Рис. 3.8 - Температурна мапа головного меню

Мапа фокусування (Рис. 3.9) для головного меню підтвердила, що ключові інтерактивні елементи, такі, як пункти навігації та кнопки запуску ігор, знаходяться у чіткій фокусній зоні, що дозволяє користувачам швидко їх ідентифікувати та взаємодіяти з ними.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				61
Змн.	Арк.	№ докум.	Підпис	Дата		

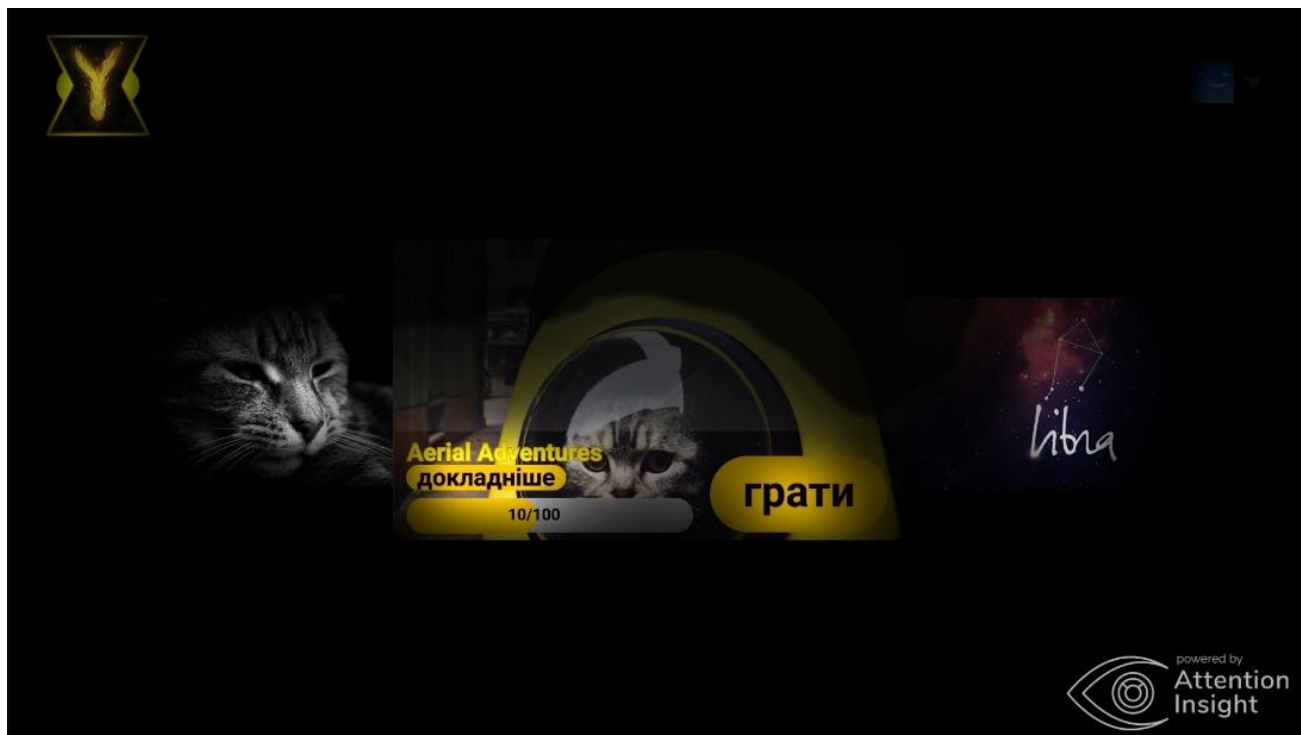


Рис. 3.9 - Мапа фокусування головного меню

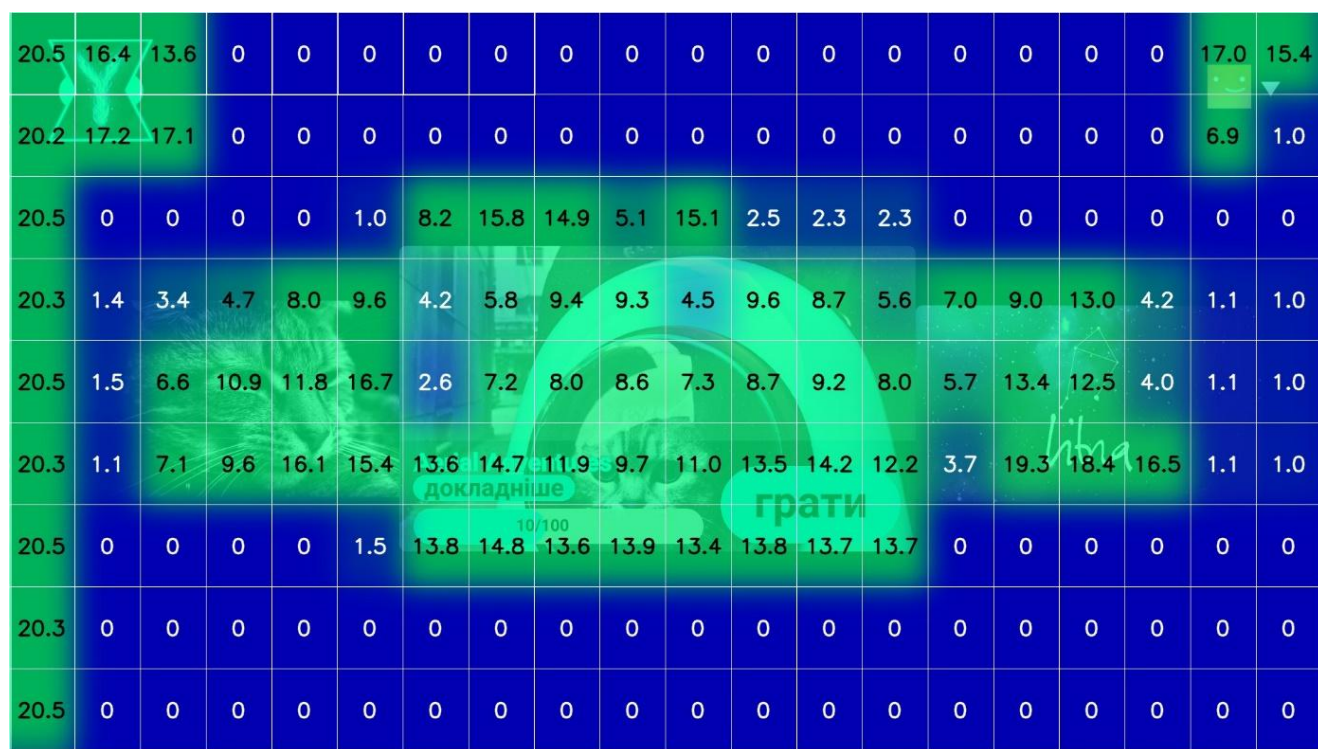


Рис. 3.10 - Мапа контрастів головного меню

Загалом, результати предиктивного тестування юзабіліті надали цінні, кількісно вимірювані дані, які об'єктивно підтвердили ефективність прийнятих дизайнерських рішень. Вони дозволили з високою впевненістю перейти до етапу розробки, значно мінімізувавши ризики створення незручного або незрозумілого для

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				62
Змн.	Арк.	№ докум.	Підпис	Дата		

користувача інтерфейсу. Такий проактивний підхід не лише підвищує якість кінцевого продукту, але й оптимізує процес розробки, запобігаючи потенційним витратам на редизайн на більш пізніх етапах проєкту.

Висновки до третього розділу

У третьому розділі було всебічно розглянуто практичні аспекти реалізації, розгортання та використання розробленого кросплатформного ігрового лаунчера, що дозволяє зробити низку обґрунтованих висновків.

По-перше, представлено та обґрунтовано архітектуру розгортання, що є фундаментом усієї системи. Вибір сучасного технологічного стека, що включає клієнтську частину на базі Rust та Tauri, а серверну — у вигляді Docker-контейнерів під управлінням Nginx[3], не був випадковим. Таке рішення забезпечило не лише високу продуктивність та безпеку застосунку, але й його кросплатформність, гнучкість у розгортанні та чудову масштабованість. Це доводить, що закладена архітектура є життєздатною та готовою до майбутнього зростання навантаження і розширення функціоналу.

По-друге, було детально проаналізовано інтерфейс програми та логіку взаємодії з ним для ключових груп користувачів. Продемонстровано, що лаунчер пропонує інтуїтивно зрозумілий та функціональний досвід як для звичайних гравців, так і для адміністраторів. Для гравців були реалізовані всі необхідні функції — від простої авторизації до комплексного управління ігровими профілями та модами. Для адміністраторів було створено потужну дворівневу систему управління: стратегічну (веб-панель для управління контентом та користувачами) та оперативну (інтегровані в лаунчер інструменти для управління ігровими серверами). Такий розподіл обов'язків підвищує ефективність та безпеку адміністрування.

По-третє, наведено переконливі результати тестування юзабіліті, проведеного на етапі проєктування за допомогою інструменту Attention Insight. Цей крок підтверджує зрілий підхід до розробки, орієнтований на користувача. Аналіз теплових мап, мап фокусування та контрастності надав об'єктивні, кількісні дані, які підтвердили, що ключові елементи інтерфейсу (форми входу, кнопки, навігаційні панелі) є помітними, зрозумілими та правильно розташованими. Це дозволило

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		63

уникнути потенційних проблем з юзабіліті ще до початку активної фази кодування, що є економічно вигідним та зменшує ризики.

Можна констатувати, що робота, описана в цьому розділі, демонструє повний цикл створення якісного програмного продукту: від вибору оптимальної архітектури та ретельного проєктування користувацького досвіду до його валідації за допомогою сучасних інструментів. Результатом є готовий до використання, стабільний та зручний програмний комплекс, що повністю відповідає поставленим на початку роботи цілям та вимогам.

		<i>Ветров А.О.</i>			ІПЗ.КР.Б – 121 – 25 – ПЗ	<i>Арк.</i>
		<i>Туренко Є.Є</i>				
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		64

ВИСНОВКИ

Результатом кваліфікаційної роботи став повноцінний програмний комплекс, що дає відповідь на гостру проблему розрізненості сучасних ігрових платформ. Було поставлено за мету змодельовати та створити універсальний кросплатформний ігровий лаунчер, що централізує ігровий досвід, спираючись на передові вебтехнології для досягнення високої продуктивності та надання розширених інструментів управління. Створення такого продукту було спрямоване не лише на підвищення зручності для гравців, але й на формування життєздатної, незалежної та безпечної альтернативи існуючим комерційним рішенням.

В основі розробки лежить всебічний аналіз предметної області та наявних на ринку продуктів, таких, як Modrinth, CurseForge та Prism Launcher. Вивчення їх функціоналу, переваг та обмежень дозволило чітко окреслити вільну нішу для застосунку, який би вдало поєднував сучасний дизайн, високу швидкодію, гнучкість налаштувань та унікальні інструменти для адміністраторів ігрових спільнот. На базі цього аналітичного етапу були сформовані вичерпні функціональні та нефункціональні вимоги, що стали дорожньою картою для всього процесу проєктування.

Вибір технологічного стеку став ключовим стратегічним рішенням. Після ретельного порівняння альтернатив було обрано синергетичну комбінацію: мова програмування Rust для серверної частини та клієнтської логіки, фреймворк Taui для побудови настільної оболонки та компілятор Svelte для створення користувацького інтерфейсу. Це рішення продиктоване прагненням до безкомпромісної продуктивності, гарантій безпеки роботи з пам'яттю, мінімального розміру фінального продукту та блискавичної реакції інтерфейсу. Інфраструктурну основу складають реляційна СУБД PostgreSQL для цілісності даних, технологія контейнеризації Docker для портативності та веб-сервер Nginx для гнучкості розгортання.

Проєктування системи відбувалося згідно з принципами модульності та інкапсуляції. Була створена об'єктно-орієнтована модель, що визначила фундаментальні сутності, їхні властивості та взаємозв'язки. Для наочного представлення архітектури та динаміки взаємодій активно використовувалися нотації UML, зокрема

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		65

діаграми класів, послідовності та компонентів. Застосування перевірених часом патернів проєктування, включно з MVC, Repository, Command та Strategy, забезпечило створення гнучкої, підтримуваної та розширюваної кодової бази як для клієнта, так і для сервера. Важливим підтвердженням зрілості інженерного підходу стала проактивна валідація дизайнерських рішень. Ефективність розроблених макетів отримала об'єктивну оцінку за допомогою предиктивного тестування юзабіліті в сервісі Attention Insight. Аналіз теплових карт та зон фокусування підтвердив, що спроектовані інтерфейси є інтуїтивно зрозумілими, а найважливіші елементи керування логічно розташовані в зонах підвищеної уваги.

Підсумком роботи є готовий до експлуатації програмний комплекс, що складається з клієнтського застосунку та серверної частини, які стабільно взаємодіють через REST API. Розроблений лаунчер надає користувачам єдину точку входу до їхніх ігор та серверів, реалізує зручне управління ігровими профілями, забезпечує безшовну інтеграцію з платформами Modrinth та CurseForge для роботи з модами, а також підтримує соціальні функції на кшталт Discord Rich Presence. Відмінною рисою продукту стала потужна дворівнева система адміністрування. Стратегічне управління контентом, користувачами та їхніми ролями відбувається через захищену веб-панель. Водночас оперативне управління ігровою інфраструктурою, таке, як моніторинг та перезавантаження серверів, реалізовано безпосередньо в лаунчері адміністратора завдяки інтеграції з API панелі Pterodactyl.

Створений програмний продукт повністю досягає поставлених на початку цілей, наочно демонструючи потенціал сучасних технологій для побудови високоякісних кросплатформних рішень. Запропонована архітектура виявилася гнучкою, масштабованою та готовою до подальшого зростання та розвитку. Кінцевий результат є не просто прототипом, а цілісним, стабільним та функціональним інструментом, що ефективно відповідає на виклик фрагментації ігрових екосистем та надає відчутні переваги як для звичайних гравців, так і для адміністраторів ігрових проєктів.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		66

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Prism Launcher. Prismlauncher. info. [Електронний ресурс] - Режим доступу до ресурсу: <https://prismlauncher.org/>.
2. The rustc book. The rustc book. book. [Електронний ресурс] - Режим доступу до ресурсу: <https://doc.rust-lang.org/rustc/index.html>.
3. Nginx. docs. [Електронний ресурс] - Режим доступу до ресурсу: <https://nginx.org/en/docs/>.
4. Svelte. Introduction. [Електронний ресурс] - Режим доступу до ресурсу: <https://svelte.dev/docs/svelte/overview>.
5. Docker. [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.docker.com/>.
6. Pterodactyl. introduction. [Електронний ресурс] - Режим доступу до ресурсу: <https://pterodactyl.io/project/introduction.html>.
7. Modrinth. app. [Електронний ресурс] - Режим доступу до ресурсу: <https://modrinth.com/app>.
8. CurseForge. app. [Електронний ресурс] - Режим доступу до ресурсу: <https://www.curseforge.com/download/app>.
9. Attentioninsight. features. [Електронний ресурс] - Режим доступу до ресурсу: <https://attentioninsight.com/features/>.
10. Tauri. start. [Електронний ресурс] - Режим доступу до ресурсу: <https://tauri.app/start/>.
11. PostgreSQL. [Електронний ресурс] - Режим доступу до ресурсу: <https://www.postgresql.org/>.
12. actix web. docs. [Електронний ресурс] - Режим доступу до ресурсу: <https://actix.rs/docs/>.
13. Shadcn-svelte. docs. [Електронний ресурс] - Режим доступу до ресурсу: <https://shadcn-svelte.com/docs/>.
14. Swagger. docs. [Електронний ресурс] - Режим доступу до ресурсу: <https://swagger.io/docs/>.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		67

15. jwt. introduction. [Електронний ресурс] - Режим доступу до ресурсу:
<https://jwt.io/introduction>.
16. Docs sqlx. [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.rs/sqlx/latest/sqlx/>.
17. Tanstack. query. [Електронний ресурс] - Режим доступу до ресурсу:
<https://tanstack.com/query/v4/docs/framework/svelte/overview>.
18. Discord. docs. [Електронний ресурс] - Режим доступу до ресурсу:
<https://discord.com/developers/docs/intro>.
19. Sentry. docs. [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.sentry.io/>.
20. @Admin. Curseforge vs Modrinth - Which one to choose?. theserverhost.
post. [Електронний ресурс] - <https://theserverhost.com/blog/post/curseforge-vs-modrinth>.
21. Agrawal J. About Modrinth. Support.modrinth. About Modrinth. [Електронний ресурс] - <https://support.modrinth.com/en/articles/8800818-about-modrinth>.
22. Eshwarla P. Rust Servers, Services, and Apps. Manning, 2023. 328 с.
23. Geewax J. API Design Patterns. Manning, 2021. 479 с.
24. Gough J. Mastering API Architecture. O'Reilly Media, 2022. 286 с.
25. Harley M. Pentesting APIs : a practical guide to discovering, fingerprinting, and exploiting apis. Packt Publishing, 2024. 290 с.
26. Indrasiri K. Design Patterns for Cloud Native Applications. O'Reilly Media, 2021. 311 с.
27. Mcnamara T. Rust in Action. Manning, 2021. 456 с.
28. Poulton N. Docker Deep Dive. Independently published, 2024. 249 с.
29. Samson C. F. Asynchronous Programming in Rust. Packt Publishing, 2024. 306 с.
30. Stew G. Rust for Network Programming and Automation. 2-ге вид. Git-forGits, 2024. 294 с.
31. Sundaramoorthy S. UML Diagramming : a case study approach. Auerbach Publications, 2022. 430 с.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко С.Є				68
Змн.	Арк.	№ докум.	Підпис	Дата		

32. Westerveld D. API Testing and Development with Postman. 2-ге вид. Packt Publishing, 2024. 358 с.

33. Winteringham M. Software Testing with Generative AI. Manning, 2024. 304 с.

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		69

ДОДАТКИ

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		70

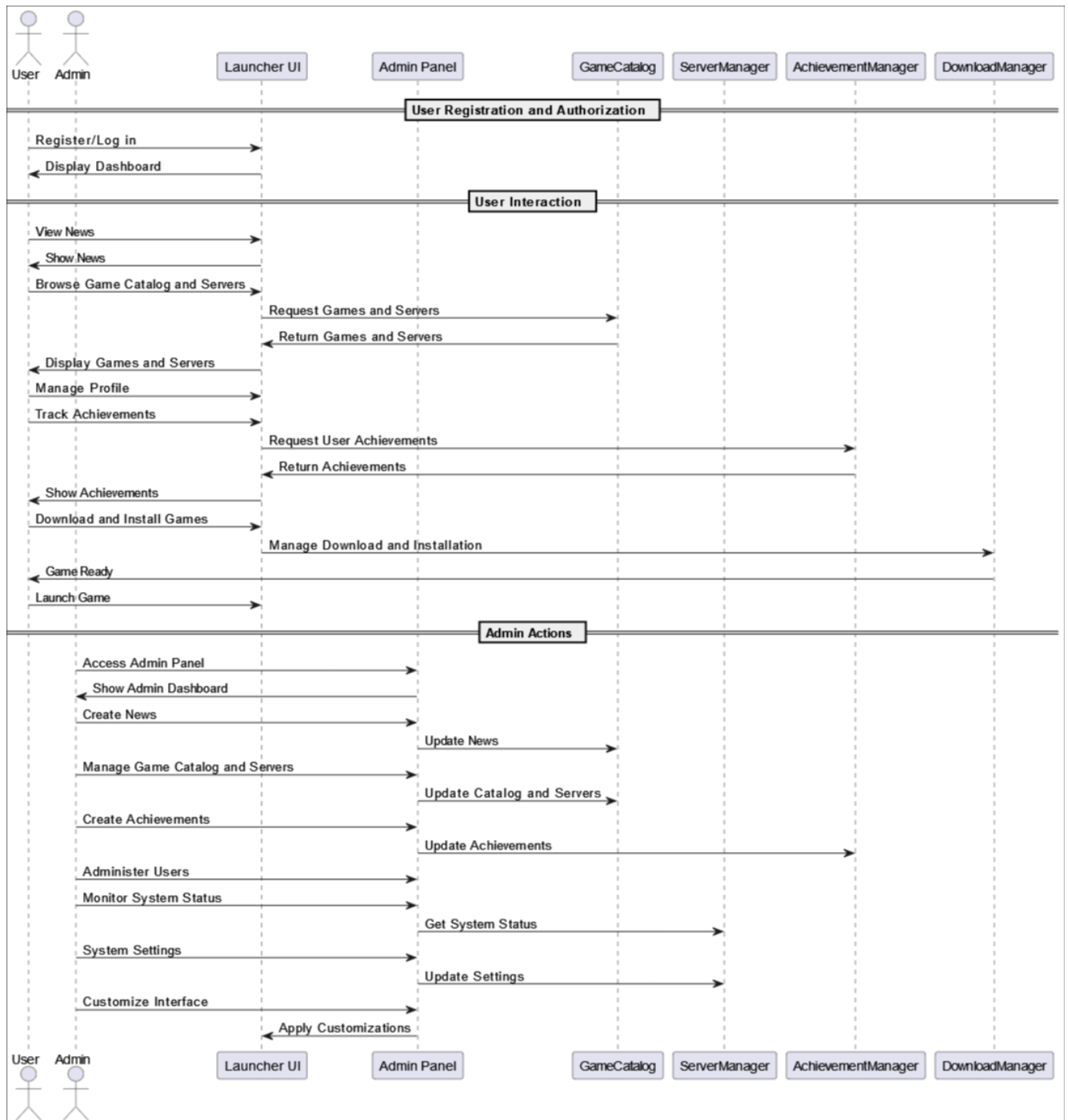


Рис. А.1 - Діаграма послідовності

CustomTitlebar

```
<script lang="ts">
  import { getCurrentWindow } from '@tauri-apps/api/window'

  const appWindow = getCurrentWindow();

  async function handleMaximize() {
    const isMaximized = await appWindow.isMaximized();
    if (isMaximized) {
      await appWindow.unmaximize();
    } else {
      await appWindow.maximize();
    }
  }
</script>

<div class="titlebar">
  <div class="drag-region" data-tauri-drag-region>
    <span class="title">My App</span>
  </div>

  <div class="window-controls">
    <button
      aria-label="Minimize"
      on:click={() => appWindow.minimize()}>
      <svg width="8" height="1" viewBox="0 0 8 1">
        <rect width="8" height="1" fill="currentColor" />
      </svg>
    </button>

    <button
      aria-label="Maximize"
      on:click={handleMaximize}>
      <svg width="8" height="8" viewBox="0 0 8 8">
        <rect width="6" height="6" x="1" y="1" fill="none" stroke="currentColor" stroke-width="1" />
      </svg>
    </button>

    <button
      aria-label="Close"
      on:click={() => appWindow.close()}>
      <svg width="8" height="8" viewBox="0 0 8 8">
        <path d="M1.5 1.5L6.5 6.5M1.5 6.5L6.5 1.5" stroke="currentColor" stroke-width="1"/>
      </svg>
    </button>
  </div>
</div>
```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		72

Auth launcher

```
<script lang="ts">
import { auth } from '$lib/stores/auth';
import { goto } from '$app/navigation';

let isLogin = true;
let error = "";

let formData = {
  email: "",
  username: "",
  password: "",
  confirmPassword: ""
};

async function handleSubmit() {
  error = "";

  try {
    if (isLogin) {
      const response = await mockLogin(formData.email, formData.password);
      auth.login(response.user, response.token);
      goto('/');
    } else {
      if (formData.password !== formData.confirmPassword) {
        error = 'Passwords do not match';
        return;
      }

      const response = await mockRegister(formData);
      auth.login(response.user, response.token);
      goto('/');
    }
  } catch (e) {
    error = e.message;
  }
}

function toggleForm() {
  isLogin = !isLogin;
  error = "";
  formData = {
    email: "",
    username: "",
    password: "",
    confirmPassword: ""
  };
}

async function mockLogin(email: string, password: string) {
  await new Promise(resolve => setTimeout(resolve, 1000));
```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		73

```

if (email === 'test@test.com' && password === 'password') {
  return {
    user: {
      id: '1',
      email,
      username: 'TestUser'
    },
    token: 'mock-token'
  };
}

throw new Error('Invalid credentials');
}

async function mockRegister(data: typeof formData) {

  await new Promise(resolve => setTimeout(resolve, 1000));

  return {
    user: {
      id: '1',
      email: data.email,
      username: data.username
    },
    token: 'mock-token'
  };
}
</script>

<div class="auth-container">
  <div class="auth-card">
    <div class="auth-header">
      <h1>{isLogin ? 'Login' : 'Register'}</h1>
      <p>
        {isLogin
          ? "Don't have an account?"
          : "Already have an account?"}
        <button class="toggle-btn" on:click={toggleForm}>
          {isLogin ? 'Register' : 'Login'}
        </button>
      </p>
    </div>

    {#if error}
      <div class="error-message">
        {error}
      </div>
    {/if}

    <form on:submit|preventDefault={handleSubmit} class="auth-form">
      <div class="form-group">
        <label for="email">Email</label>

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

<input
  type="email"
  id="email"
  bind:value={formData.email}
  required
  placeholder="Enter your email"
/>
</div>

{#if !isLogin}
  <div class="form-group">
    <label for="username">Username</label>
    <input
      type="text"
      id="username"
      bind:value={formData.username}
      required
      placeholder="Choose a username"
    />
  </div>
{/if}

<div class="form-group">
  <label for="password">Password</label>
  <input
    type="password"
    id="password"
    bind:value={formData.password}
    required
    placeholder="Enter your password"
  />
</div>

{#if !isLogin}
  <div class="form-group">
    <label for="confirmPassword">Confirm Password</label>
    <input
      type="password"
      id="confirmPassword"
      bind:value={formData.confirmPassword}
      required
      placeholder="Confirm your password"
    />
  </div>
{/if}

<button type="submit" class="submit-btn">
  {isLogin ? 'Login' : 'Create Account'}
</button>
</form>

{#if isLogin}

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		75

```

    <button class="forgot-password">Forgot Password?</button>
  {/if}
</div>
</div>

```

Library

```

<script lang="ts">
  let searchQuery = "";
  let selectedCategory = 'all';

  $: filteredGames = games.filter(game => {
    const matchesSearch = game.title.toLowerCase().includes(searchQuery.toLowerCase());
    const matchesCategory = selectedCategory === 'all' || game.category === selectedCategory;
    return matchesSearch && matchesCategory;
  });

  const categories = ['all', 'RPG', 'Action', 'Strategy', 'Adventure'];
</script>

<div class="library">
  <div class="controls">
    <div class="search">
      <input
        type="text"
        placeholder="Search games..."
        bind:value={searchQuery}
      />
    </div>
    <div class="categories">
      {#each categories as category}
        <button
          class:active={selectedCategory === category}
          on:click={() => selectedCategory = category}
        >
          {category}
        </button>
      {/each}
    </div>
  </div>

  <div class="games-grid">
    {#each filteredGames as game}
      <div class="game-card">
        <div class="game-image" style="background-image: url({game.image})">
          <div class="game-overlay">
            <button class="play-btn">Play Now</button>
          </div>
        </div>
        <div class="game-info">

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		76

```

<h3>{game.title}</h3>
<div class="game-stats">
  <span>Last played: {game.lastPlayed}</span>
  <span>Playtime: {game.playtime}</span>
</div>
<div class="game-actions">
  <button class="action-btn">Details</button>
  <button class="action-btn">Uninstall</button>
</div>
</div>
</div>
{/each}
</div>
</div>

```

News

```

<script lang="ts">

const categories = ['All', 'Updates', 'DLC', 'Development', 'Community'];
let selectedCategory = 'All';

$: filteredNews = selectedCategory === 'All'
  ? news
  : news.filter(item => item.category === selectedCategory);
</script>

<div class="news">
  <div class="news-header">
    <h1>Latest News</h1>
    <div class="categories">
      {#each categories as category}
        <button
          class:active={selectedCategory === category}
          on:click={() => selectedCategory = category}
        >
          {category}
        </button>
      {/each}
    </div>
  </div>
</div>

<div class="news-grid">
  {#each filteredNews as item}
    <article class="news-card">
      <div class="news-image" style="background-image: url({item.image})">
        <div class="news-category">{item.category}</div>
      </div>
      <div class="news-content">
        <div class="news-meta">
          <span class="date">{item.date}</span>
          <span class="author">{item.author}</span>

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		77

```

</div>
<h2>{item.title}</h2>
<p>{item.excerpt}</p>
<button class="read-more">Read More</button>
</div>
</article>
{/each}
</div>

<div class="newsletter">
  <div class="newsletter-content">
    <h2>Stay Updated</h2>
    <p>Subscribe to our newsletter to get the latest gaming news and updates.</p>
    <form class="subscribe-form">
      <input type="email" placeholder="Enter your email" />
      <button type="submit">Subscribe</button>
    </form>
  </div>
</div>
</div>

```

Prifile

```

<script lang="ts">

</script>

<div class="profile">
  <div class="profile-header">
    <div class="avatar">
      <img src={user.avatar} alt={user.username} />
    </div>
    <div class="user-info">
      <h1>{user.username}</h1>
      <div class="stats">
        <div class="stat">
          <span class="value">{user.level}</span>
          <span class="label">Level</span>
        </div>
        <div class="stat">
          <span class="value">{user.gamesOwned}</span>
          <span class="label">Games</span>
        </div>
        <div class="stat">
          <span class="value">{user.achievements}</span>
          <span class="label">Achievements</span>
        </div>
      </div>
    </div>
  </div>
  <div class="profile-content">

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		78

```

<section class="achievements">
  <h2>Recent Achievements</h2>
  <div class="achievements-list">
    {#each user.recentAchievements as achievement}
      <div class="achievement-card">
        <div class="achievement-info">
          <h3>{achievement.title}</h3>
          <p class="game">{achievement.game}</p>
          <p class="description">{achievement.description}</p>
        </div>
        <div class="achievement-date">
          {achievement.date}
        </div>
      </div>
    {/each}
  </div>
</section>

<section class="gaming-stats">
  <h2>Gaming Stats</h2>
  <div class="stats-grid">
    <div class="stat-card">
      <h3>Total Playtime</h3>
      <span class="stat-value">{user.stats.totalPlaytime}</span>
    </div>
    <div class="stat-card">
      <h3>Favorite Genre</h3>
      <span class="stat-value">{user.stats.favoriteGenre}</span>
    </div>
    <div class="stat-card">
      <h3>Average Playtime</h3>
      <span class="stat-value">{user.stats.averagePlaytime}</span>
    </div>
  </div>
</section>
</div>
</div>

```

Settings

```

<script lang="ts">

const languages = ['en', 'ua', 'de', 'fr', 'es'];
const resolutions = ['1920x1080', '2560x1440', '3840x2160'];
const graphicsQualities = ['low', 'medium', 'high', 'ultra'];

function saveSettings() {
  // TODO: Implement settings save
  console.log('Settings saved:', settings);
}
</script>

<div class="settings">

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		79

```

<h1>Settings</h1>

<div class="settings-grid">
  <section class="settings-section">
    <h2>Appearance</h2>
    <div class="setting-group">
      <label>
        Theme
        <select bind:value={settings.appearance.theme}>
          <option value="dark">Dark</option>
          <option value="light">Light</option>
        </select>
      </label>

      <label>
        Language
        <select bind:value={settings.appearance.language}>
          {#each languages as lang}
            <option value={lang}>{lang.toUpperCase()}</option>
          {/each}
        </select>
      </label>

      <label class="checkbox">
        <input
          type="checkbox"
          bind:checked={settings.appearance.notifications}
        />
        Enable notifications
      </label>
    </div>
  </section>

  <section class="settings-section">
    <h2>Gaming</h2>
    <div class="setting-group">
      <label class="checkbox">
        <input
          type="checkbox"
          bind:checked={settings.gaming.autoUpdate}
        />
        Auto-update games
      </label>

      <label class="checkbox">
        <input
          type="checkbox"
          bind:checked={settings.gaming.showFps}
        />
        Show FPS counter
      </label>
    </div>
  </section>
</div>

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

<label>
  Resolution
  <select bind:value={settings.gaming.gameResolution}>
    {#each resolutions as resolution}
      <option value={resolution}>{resolution}</option>
    {/each}
  </select>
</label>

<label>
  Graphics Quality
  <select bind:value={settings.gaming.graphicsQuality}>
    {#each graphicsQualities as quality}
      <option value={quality}>{quality}</option>
    {/each}
  </select>
</label>
</div>
</section>

<section class="settings-section">
  <h2>Privacy</h2>
  <div class="setting-group">
    <label class="checkbox">
      <input
        type="checkbox"
        bind:checked={settings.privacy.shareGameActivity}
      />
      Share game activity
    </label>

    <label class="checkbox">
      <input
        type="checkbox"
        bind:checked={settings.privacy.showOnline}
      />
      Show online status
    </label>

    <label class="checkbox">
      <input
        type="checkbox"
        bind:checked={settings.privacy.allowFriendRequests}
      />
      Allow friend requests
    </label>
  </div>
</section>
</div>

<div class="actions">
  <button class="save-btn" on:click={saveSettings}>Save Changes</button>

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		81

```
</div>
</div>
```

Layout launcher

```
<script>
  import TitleBar from '$lib/CustomTitlebar.svelte';
  import { page } from '$app/stores';
  import { auth } from '$lib/stores/auth';
  import './app.css';

  function handleLogout() {
    auth.logout();
  }
</script>

<div class="app">
  <TitleBar />
  <nav>
    <div class="logo">GameLauncher</div>
    <div class="nav-links">
      <a href="/" class:active={$page.url.pathname === '/'}>Home</a>
      <a href="/library" class:active={$page.url.pathname === '/library'}>Library</a>
      <a href="/news" class:active={$page.url.pathname === '/news'}>News</a>
    </div>
    <div class="profile">
      {#if $auth.isAuthenticated}
        <a href="/profile">Profile</a>
        <a href="/settings">Settings</a>
        <button class="auth-btn" on:click={handleLogout}>Logout</button>
      {:else}
        <a href="/auth" class="auth-btn">Login</a>
      {/if}
    </div>
  </nav>

  <main>
    <slot />
  </main>
</div>
```

Main page

```
<script lang="ts">
  import { invoke } from '@tauri-apps/api/core';
  import { WebviewWindow } from '@tauri-apps/api/webviewWindow';

  let name = $state('');
  let greetMsg = $state('');

  async function greet(event: Event) {
    event.preventDefault();
    greetMsg = await invoke("greet", { name });
  }
</script>
```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		82

```

}
</script>

<div class="home">
  <section class="hero">
    <h1>Welcome to GameLauncher</h1>
    <p>Your ultimate gaming destination</p>
  </section>

  <section class="featured">
    <h2>Featured Games</h2>
    <div class="games-grid">
      {#each featuredGames as game}
        <div class="game-card">
          <div class="game-image" style="background-image: url({game.image})">
            <div class="game-overlay">
              <button class="play-btn">Play Now</button>
            </div>
          </div>
          <div class="game-info">
            <h3>{game.title}</h3>
            <p>{game.description}</p>
          </div>
        </div>
      {/each}
    </div>
  </section>

  <section class="quick-actions">
    <div class="action-card">
      <h3>Library</h3>
      <p>Access your games</p>
      <a href="/library" class="action-btn">View Library</a>
    </div>
    <div class="action-card">
      <h3>Store</h3>
      <p>Discover new games</p>
      <a href="/store" class="action-btn">Browse Store</a>
    </div>
    <div class="action-card">
      <h3>News</h3>
      <p>Latest updates</p>
      <a href="/news" class="action-btn">Read More</a>
    </div>
  </section>
</div>

```

		Ветров А.О.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
		Туренко Є.Є				
Змн.	Арк.	№ докум.	Підпис	Дата		83