

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:

**«Вебплатформа настільно-рольових ігор:
модулі організації та комунікації користувачів»**

Виконав студент групи ІПЗ-22-4
ЯРМОЛЕНКО Анатолій Миколайович

Керівник роботи:
ТАЛАВЕР Олег Володимирович

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

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

«ЗАТВЕРДЖУЮ»

Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЯРМОЛЕНКО Анатолій Миколайович**

Керівник роботи: **ТАЛАВЕР Олег Володимирович**

Тема роботи: **«Вебплатформа настільно-рольових ігор: модулі організації та комунікації користувачів»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **предметом дослідження є методи, алгоритми та програмні засоби розробки веборієнтованих систем для управління ігровими спільнотами та інтеграції сервісів реального часу. Об'єктом дослідження є процес організації та проведення настільних рольових ігор у дистанційному та змішаному режимах.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Талавер О.В.	02.03.2026р.	10.03.2026р.
2	Талавер О.В.	15.04.2026р.	30.04.2026р.
3	Талавер О.В.	07.05.2026р.	22.05.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел із 34 найменувань, 1 додатка (47 сторінок), 18 таблиць та 57 ілюстрацій у тексті. Повний обсяг роботи становить 141 сторінку, з них 94 сторінки основного тексту.

Метою роботи є проєктування та розробка вебплатформи для організації настільних рольових ігор та забезпечення комунікації між гравцями і майстрами.

У першому розділі охарактеризовано предметну область настільних рольових ігор, виконано огляд та порівняльний аналіз існуючих програмних рішень для планування та координації ігор, сформульовано задачу дослідження та обґрунтовано вибір технологій реалізації.

У другому розділі визначено функціональні та нефункціональні вимоги до підсистеми організації та комунікації, обрано архітектурний підхід та інструментальні засоби. Спроектовано структуру вебплатформи, модель даних і механізми взаємодії між компонентами, а також описано програмну реалізацію серверної та клієнтської частин.

У третьому розділі висвітлено конфігурацію середовища розгортання платформи та комплексні заходи із забезпечення безпеки даних. Наведено практичні приклади користувацького інтерфейсу, налаштування CI/CD процесів та результати тестування розробленої системи.

У результаті виконання роботи розроблено вебплатформу, що забезпечує повний цикл організації настільних рольових ігор та комунікації між учасниками: від створення кампанії, планування та погодження ігрових сесій у календарі до інтеграції системи сповіщень, відеозв'язку та чатів.

КЛЮЧОВІ СЛОВА: НАСТІЛЬНІ РОЛЬОВІ ІГРИ, WEBSOCKET, SELECTIVE FORWARDING UNIT, EXPRESS, REACT.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебплатформа настільно-рольових ігор: модулі організації та комунікації користувачів Пояснювальна записка	Лім.	Арк.	Аркуші
Розроб.		Ярмоленко А.М.						
Керівник		Талавер О.В.					3	141
Рецензент		Чижмотря О.В.				Житомирська політехніка, група ІПЗ-22-4		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis consists of an introduction, three chapters, conclusions, a list of references with 34 sources, 1 appendix (47 pages), 18 tables and 57 illustrations throughout the text. The total volume of the thesis is 141 pages, of which 94 pages constitute the main body.

The main goal of the thesis is to design and develop a web platform for organizing tabletop role-playing games and ensuring communication between players and game masters.

Chapter 1 characterizes the subject area of tabletop role-playing games, provides an overview and comparative analysis of existing software solutions for planning and coordinating games, formulates the research problem, and justifies the choice of implementation technologies.

Chapter 2 defines functional and non-functional requirements for the organization and communication subsystem, selects the architectural approach and development tools. The web platform structure, data model, and component interaction mechanisms are designed, alongside the programmatic implementation of the server and client parts.

Chapter 3 describes the deployment environment configuration for the platform and comprehensive security measures. Examples of the user interface, CI/CD pipeline setups, and the overall system testing results are presented.

As a result of the work, a web platform has been developed that provides a full cycle of tabletop role-playing game organization and communication between participants: from campaign creation, scheduling, and game session approval in the calendar, to the integration of a notification system, video communication, and chats.

KEYWORDS: TABLETOP ROLE-PLAYING GAMES, WEBSOCKET, SELECTIVE FORWARDING UNIT, EXPRESS, REACT.

		Ярмоленко А.М.			ПІЗ.КР.Б – 121 – 26 – ПІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	9
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ	11
1.1 Постановка задачі.....	11
1.2 Аналіз аналогів програмного продукту	13
1.3 Обґрунтування вибору архітектури системи	20
1.4 Обґрунтування вибору інструментальних засобів	26
Висновки до першого розділу.....	30
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ТА КОМУНІКАЦІЇ ГРАВЦІВ В НРІ	31
2.1 Визначення варіантів використання та об’єктно-орієнтованої структури системи	31
2.2 Розробка бази даних системи.....	39
2.3 Проєктування та реалізація алгоритмів роботи системи	49
2.4 Реалізація функціоналу модулів організації та комунікації	59
Висновки до другого розділу	64
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ...	65
3.1 Порядок встановлення та налаштування системи	65
3.2 Структура інтерфейсу платформи.....	69
3.3 Тестування роботи програмного комплексу	81
Висновки до третього розділу.....	88

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ.....	90
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	91
ДОДАТКИ.....	94
Додаток А.....	95

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HPI – настільні рольові ігри;

ACID (Atomicity, Consistency, Isolation, Durability) – набір властивостей надійності транзакцій;

APDEX (Application Performance Index) – індекс задоволеності продуктивністю застосунку;

API (Application Programming Interface) – інтерфейс прикладного програмування;

CI/CD (Continuous Integration / Continuous Delivery) – безперервна інтеграція та доставка;

CPU (Central Processing Unit) – центральний процесор;

CRM (Customer Relationship Management) – система управління взаємовідносинами з клієнтами;

CSR (Client-Side Rendering) – рендеринг на стороні клієнта;

CSRF (Cross-Site Request Forgery) – міжсайтова підробка запиту;

CSS (Cascading Style Sheets) – каскадні таблиці стилів;

DOM (Document Object Model) – об'єктна модель документа;

HMR (Hot Module Replacement) – гаряча заміна модулів;

HTTP (HyperText Transfer Protocol) – протокол передавання гіпертексту;

HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передавання гіпертексту;

ICE (Interactive Connectivity Establishment) – протокол встановлення інтерактивного з'єднання;

JSON (JavaScript Object Notation) – текстовий формат обміну даними;

JWT (JSON Web Token) – токен доступу у форматі JSON;

LFG (Looking for Group) – пошук гравців до ігрової групи;

MPA (Multi-Page Application) – багатосторінковий вебзастосунок;

NAT (Network Address Translation) – трансляція мережевих адрес;

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

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленої системи;

RPC (Remote Procedure Call) – віддалений виклик процедур;

SEO (Search Engine Optimization) – пошукова оптимізація;

SFU (Selective Forwarding Unit) – блок селективного перенаправлення медіапотоків;

SPA (Single Page Application) – односторінковий вебзастосунок;

SQL (Structured Query Language) – мова структурованих запитів;

SSD (Solid-State Drive) – твердотільний накопичувач;

SSE (Server-Sent Events) – події, що надсилаються сервером;

SSG (Static Site Generation) – статична генерація сайту;

SSR (Server-Side Rendering) – рендеринг на стороні сервера;

TCP (Transmission Control Protocol) – протокол керування передаванням;

TLS (Transport Layer Security) – протокол захисту транспортного рівня;

TURN (Traversal Using Relays around NAT) – ретрансляція трафіку для обходу NAT;

UDP (User Datagram Protocol) – протокол передавання датаграм користувача;

UML (Unified Modeling Language) – уніфікована мова моделювання;

VT (Virtual Tabletop) – віртуальний ігровий стіл;

WebRTC (Web Real-Time Communication) – технологія зв'язку в реальному часі у браузері;

WS (WebSocket) – протокол повнодуплексного двостороннього зв'язку в реальному часі.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

ВСТУП

Актуальність теми. Наразі індустрія настільних рольових ігор, таких як Dungeons & Dragons, Pathfinder та інші, переживає період стрімкого зростання та діджиталізації. Гравці та майстри гри все частіше переходять до онлайн-форматів або використовують цифрові інструменти для координації живих сесій. Проте організація ігрового процесу залишається складним багатофакторним завданням, що потребує одночасного врахування розкладу учасників, методів комунікації та створення специфічного ігрового середовища. Застосування сучасних архітектурних рішень та засобів вебпрограмування дає змогу реалізувати ефективний обмін повідомленнями в реальному часі, а також гарантувати надійність доставки системних сповіщень користувачам.

Існуючі комерційні аналоги часто мають обмежену гнучкість у моделях доступу або вимагають високої абонентської плати, що створює значний поріг входження. Внаслідок відсутності єдиного комплексного рішення значна частина потенційної аудиторії відмовляється від цього хобі або змушена використовувати одночасно кілька неспеціалізованих та часто несумісних між собою інструментів (месенджери, онлайн-календарі). Тому розробка спеціалізованої вебплатформи, яка об'єднує засоби планування, рольову модель та інтегровані комунікаційні модулі в межах єдиної екосистеми, є актуальним завданням для автоматизації процесів та розвитку тематичних спільнот.

Метою роботи є проєктування та розробка вебплатформи для організації настільних рольових ігор та забезпечення комунікації між гравцями і майстрами.

Для досягнення поставленої мети необхідно вирішити такі завдання: проаналізувати предметну область організації настільних рольових ігор та особливості взаємодії між користувачами; виконати огляд існуючих програмних рішень для планування та координації ігор і визначити їхні обмеження; обґрунтувати вибір архітектурного підходу та інструментальних засобів реалізації; спроектувати структуру вебплатформи та модель даних; програмно реалізувати серверну та клієнтську частини системи; провести функціональне тестування розробленого програмного продукту.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

Об’єктом дослідження є процес організації та проведення настільних рольових ігор у дистанційному та змішаному режимах, включаючи планування розкладу, формування груп та комунікацію учасників.

Предметом дослідження є методи, алгоритми та програмні засоби розробки веборієнтованих систем для управління ігровими спільнотами та інтеграції сервісів реального часу.

Методи дослідження. У роботі застосовано комплекс методів системного аналізу предметної області, принципи об’єктно-орієнтованого проектування та мову UML для моделювання архітектури. Програмна реалізація базується на сучасних підходах до вебпрограмування та методах проектування реляційних баз даних, а перевірка працездатності системи здійснювалася за допомогою методів функціонального тестування.

Практичне значення отриманих результатів полягає у створеному програмному рішенні, яке автоматизує процеси планування та управління заявками, що мінімізує адміністративне навантаження на організаторів ігор. Платформа спрощує залучення нових учасників через спеціалізовану підсистему запитів на приєднання та забезпечує централізоване середовище для комунікації і зберігання даних, що підвищує доступність хобі та покращує користувацький досвід.

За темою випускної кваліфікаційної роботи було опубліковано тези [1].

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Попри масовість настільних рольових ігор, їх організація залишається технічно складним процесом. Ключовою прикладною проблемою, на розв'язання якої спрямована робота, є фрагментація інструментарію: координація учасників, планування сесій, пошук гравців і відеозв'язок здійснюються в окремих, несумісних між собою сервісах. Це створює надмірне адміністративне навантаження на майстра гри, призводить до втрати ігрової інформації та підвищує поріг входу для новачків. Тому перед роботою постає задача створення єдиної централізованої вебплатформи, яка об'єднає ці функції у спільному середовищі.

Запропонована система повинна безшовно об'єднати в межах спільного інформаційного простору засоби календарного планування, гнучкі механізми пошуку та адміністрування учасників, а також вбудовані модулі комунікації. Консолідація цих функцій дозволить усунути організаційний хаос, знизити поріг входу для нових гравців та забезпечити цілісний ігровий досвід.

Функціонал майбутньої платформи базується на структурованій ієрархії ігрових сутностей, де центральне місце займають домени «Кампанія» та «Сесія». Система має підтримувати як довготривалі кампанії з постійним складом учасників, так і разові ігрові зустрічі (one-shots), що дозволяє адаптувати додаток під різні формати ігрової взаємодії. Для ефективного адміністрування ігрового процесу передбачається впровадження розгалуженої рольової моделі та гнучкої матриці прав доступу. Це дозволить реалізувати різні рівні видимості контенту: від відкритих публічних кампаній до повністю приватних ігор. Такий підхід забезпечує покриття найбільш актуальних та проблемних сценаріїв використання платформи.

Ключовим завданням автоматизації є оптимізація соціальної взаємодії через впровадження керованого механізму приєднання учасників (join-flow), що

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

підтримує режими вільного вступу, модерацію заявок майстром гри або доступ лише для членів спільноти. З метою подолання проблеми фрагментації інструментарію, перед системою ставиться задача розробки інтегрованого модуля комунікації. Він має забезпечувати текстову взаємодію у межах ізольованих чатів кампаній та сесій, а також містити необхідну інфраструктурну базу для інтеграції відеодзвінків у режимі реального часу, що дозволить проводити ігри безпосередньо у вебдодатку. Функціональну цілісність системи завершує модуль календаря, основна мета якого зводиться до зручного планування та пошуку підходящих ігрових сесій. Об'єднання цих модулів у спільному інтерфейсі мінімізує рутину для майстрів гри та зробить взаємодію з платформою максимально цілісною.

Крізь весь корисний функціонал проходить тема забезпечення інформаційної безпеки та захисту персональних даних, яка є критичною вимогою до розроблюваної платформи. Для запобігання несанкціонованому доступу система повинна реалізовувати надійну автентифікацію з обов'язковою верифікацією користувачів через електронну пошту. З метою захисту процесів комунікації, механізм приєднання до ігрових сесій має базуватися на криптографічно стійких посиланнях-запрошеннях, що унеможлиблює їх підбір сторонніми особами. Крім того, інфраструктура додатку повинна гарантувати сувору ізоляцію даних між різними ігровими кампаніями, створюючи безпечне середовище для взаємодії гравців.

Зважаючи на вищевикладене, для успішної реалізації вебплатформи необхідно вирішити комплекс взаємопов'язаних завдань, що охоплюють усі етапи проектування та розробки програмного продукту.

На початковому етапі передбачається ґрунтовне дослідження предметної області та аналіз існуючих на ринку рішень для організації настільних рольових ігор, що дозволить чітко окреслити функціональні межі майбутньої системи. На основі отриманих даних буде спроектовано інформаційну модель для надійного зберігання сутностей ігрових кампаній, окремих сесій, даних користувачів та їхньої взаємодії, після чого розпочнеться безпосередня розробка серверної

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

логіки додатку з вбудованою системою безпечної автентифікації та розмежуванням прав доступу на основі багаторівневої рольової моделі. Паралельно з цим передбачено створення механізмів управління складом учасників і логіки генерації захищених посилань-запрошень для доступу до приватних ігор. Фінальна стадія технічної реалізації полягатиме у створенні інтерактивного користувацького інтерфейсу з інструментами календарного планування, управління ігровими сутностями та комунікаційними модулями, після чого буде проведено комплексне тестування розробленого програмного забезпечення для детальної перевірки його працездатності, надійності та повної відповідності всім поставленим вимогам.

1.2 Аналіз аналогів програмного продукту

Проектування ефективної вебплатформи для координації НРІ потребує детального дослідження існуючих програмних рішень. На сьогодні ринок програмного забезпечення для НРІ-спільнот представлений широким спектром інструментів: від потужних віртуальних столів до спеціалізованих маркетплейсів для пошуку ігрових майстрів та локальних соціальних сервісів.

Для проведення всебічного оцінювання функціональних можливостей існуючих систем у контексті організації та комунікації користувачів було сформовано перелік ключових технічних і функціональних критеріїв, які дозволять об'єктивно порівняти наявні аналоги та визначити їхню відповідність вимогам майбутньої платформи.

Цей аналіз фокусується на гнучкості системи управління доступом та ролями, що передбачає наявність чіткої багаторівневої ієрархії від власника кампанії до гравця з можливістю точного налаштування їхніх прав, а також на підтримці розширених моделей видимості для створення як публічних ігор, так і повністю приватних ігрових сутностей. Важливими елементами порівняння є реалізація повноцінної системи подачі, розгляду та обробки заявок на участь від кандидатів, наявність спеціалізованого інструментарію календарного планування ігрових зустрічей з автоматичним урахуванням різних часових поясів учасників, а також глибина інтеграції комунікаційних інструментів у

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

вигляді централізованих внутрішніх чатів чи форумів безпосередньо на платформі. Крім того, особлива увага приділяється інтеграції засобів комунікації реального часу для забезпечення вбудованого аудіо- та відеозв'язку в екосистемі застосунку та впровадженню багаторівневої системи сповіщень, яка гарантує автоматичне інформування користувачів про всі ключові зміни стану системи.

Першою проаналізованою платформою став сервіс Roll20, який є одним із найстаріших та найбільш впізнаваних рішень на ринку саме віртуальних ігрових столів [2]. Тим не менш, система також пропонує комплексний набір інструментів для організації сесій та забезпечує зв'язок за допомогою спеціалізованих форумів (рис. 1.1). Завдяки тривалому перебуванню на ринку, платформа сформувала найбільшу базу користувачів, що є ключовим фактором для соціальної складової продукту.

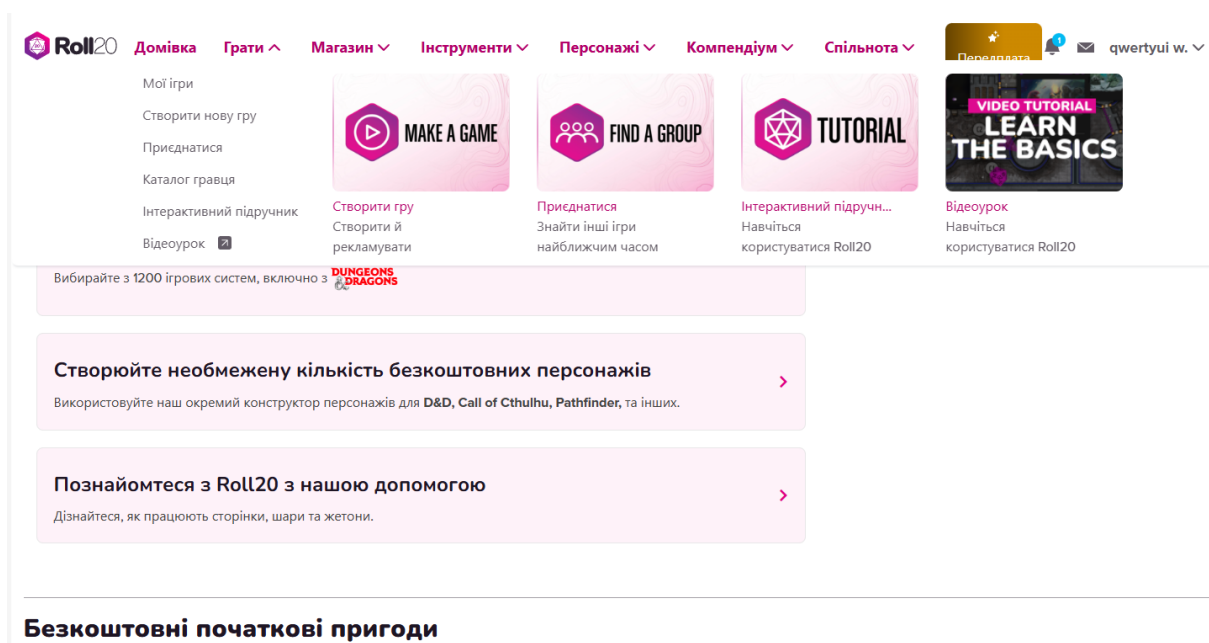


Рис. 1.1. Головна сторінка та функціонал сайту Roll20

Важливою перевагою сервісу є вбудована система пошуку ігор Looking for Group (рис. 1.2), яка дозволяє майстрам ефективно залучати нових гравців через відкриті оголошення. Наявність інтегрованого аудіо та відеозв'язку забезпечує синхронну комунікацію безпосередньо у браузері. Це дозволяє користувачам розпочати ігровий процес без необхідності встановлення стороннього програмного забезпечення.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

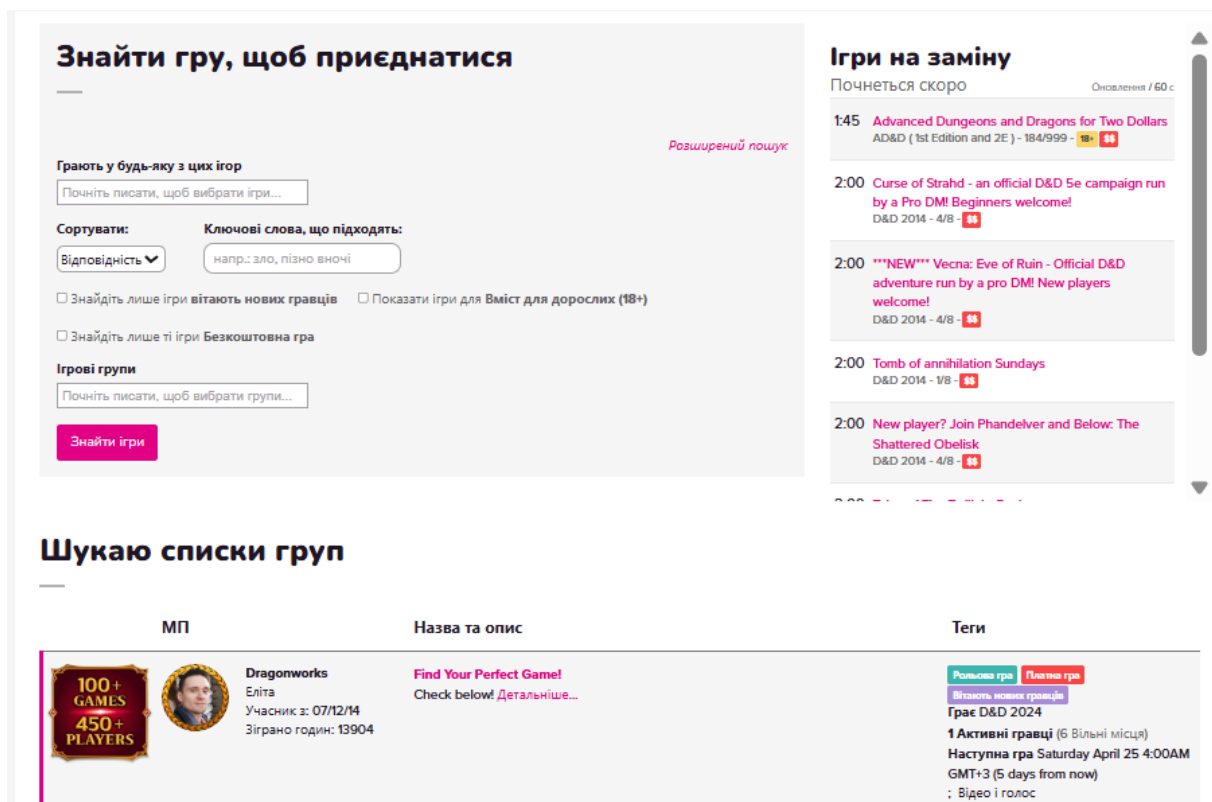


Рис. 1.2. Сторінка пошуку ігрових сесій на платформі Roll20

Проте архітектура платформи має низку недоліків у контексті сучасного користувацького досвіду. Інтерфейс системи перевантажений надлишковими елементами, що значно підвищує поріг входу для новачків. Процес навігації між модулем пошуку, налаштуваннями кампанії та безпосередньо ігровою сесією є фрагментованим, що ускладнює швидкий доступ до необхідної організаційної інформації поза межами активної ігрової сесії.

Організаційний функціонал Roll20 демонструє обмеженість у гнучкому управлінні правами доступу та автоматизації обробки заявок. Система сповіщень функціонує переважно через електронну пошту, що не відповідає вимогам до оперативності в реальному часі. Через відсутність розвинених інструментів асинхронної комунікації та структурованих чатів, ігрові спільноти часто змушені переносити координацію у Discord чи інші сторонні месенджери.

Наступним логічним кроком став розгляд сервісу StartPlaying.games, який представляє собою провідний міжнародний маркетплейс, що спеціалізується на професійній організації настільних рольових ігор [3]. Платформа виступає посередником між професійними майстрами гри та гравцями, фокусуючись на

зручності комерційного бронювання сесій. Такий підхід дозволив системі стати ключовим інструментом для монетизації ігрового дозвілля в глобальному масштабі.

Однією з найсильніших сторін сервісу є високотехнологічний модуль пошуку та підбору ідеальної гри для будь-якого користувача. Гравці отримують доступ до деталізованої вітрини ігор з можливістю гнучкої фільтрації за ігровими системами та тривалістю (рис. 1.3). Система відгуків та профілі майстрів забезпечують високий рівень довіри та соціальної верифікації учасників.

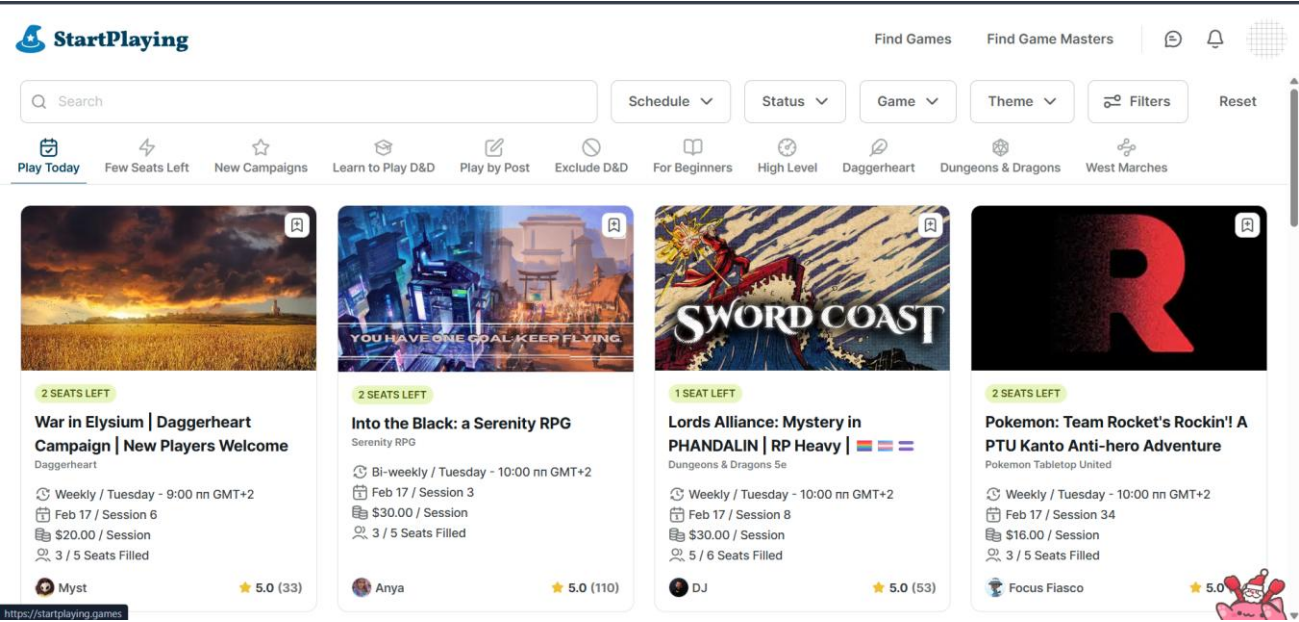


Рис. 1.3. Сторінка пошуку гри на StartPlaying

Платформа реалізує ефективний механізм обробки заявок на участь, інтегрований з платіжними шлюзами для автоматизації фінансових розрахунків. Проте жорстка комерційна орієнтація обмежує використання сервісу для некомерційних кампаній чи закритих дружніх ігор. Це створює бар'єри для спільнот, що потребують організаційного інструментарію без обов'язкової прив'язки до публічного маркетплейсу.

Суттєвим недоліком StartPlaying є значна фрагментація комунікаційного процесу та відсутність внутрішнього ігрового середовища. Платформа не пропонує розвинених засобів асинхронного спілкування або вбудованих інструментів для проведення сесій. Після завершення бронювання користувачі

змушені переходити на сторонні сервіси, як-от Telegram, що призводить до розпорошення ігрових даних та втрати організаційного контексту.

Рольова модель застосунку є досить лінійною і зосереджена на базових атрибутах членства у конкретній ігровій сесії. Відсутність інтегрованих чатів для довготривалих кампаній та засобів прямої аудіовізуальної взаємодії всередині екосистеми змушує учасників використовувати декілька програмних продуктів паралельно. Це негативно впливає на цілісність користувацького досвіду та ускладнює координацію великих ігрових груп.

Ще одним важливим об'єктом дослідження став сервіс GeekerApp, який є новим вітчизняним програмним продуктом, що позиціонується переважно як соціальна платформа для пошуку однодумців та організації локальних заходів [4]. На відміну від глобальних маркетплейсів чи VTT-систем, основний акцент розробників зроблено на розвитку регіональних спільнот любителів настільних ігор загалом (включаючи НРІ) та координації ігрових сесій у фізичному просторі.

Перевагою сервісу є простота онбордингу та ефективна фільтрація подій за конкретними містами. Платформа дозволяє майстрам та організаторам швидко анонсувати офлайн-зустрічі, а новачкам знаходити ігри поблизу. Проте така жорстка географічна прив'язка робить систему практично непридатною для організації віддалених ігор, що суттєво звужує її застосовність у сучасних реаліях.

Архітектура застосунку демонструє низьке покриття розширених користувацьких сценаріїв організації ігор. Функціонал зосереджений виключно на форматі одноразових подій, залишаючи поза увагою засоби для ведення довготривалих кампаній, що є критичним саме для сегменту НРІ. Через орієнтацію на разові зустрічі в системі немає інструментів для збереження хронології сесій, довгострокового відстеження прогресу кампаній, а також інструментів комунікації. Це змушує спільноти використовувати сторонні месенджери та хмарні сховища для менеджменту ігрових груп. У системі відсутня гнучка багаторівнева рольова модель та складні матриці доступів до ігрових сутностей (рис. 1.4).

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

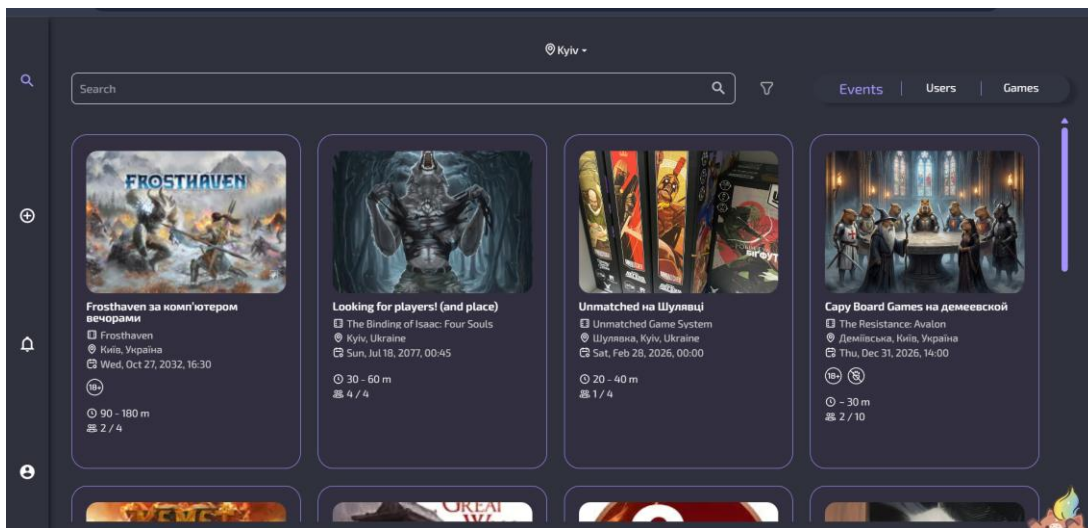


Рис. 1.4. Сторінка пошуку гри на GeekerApp

Подібно до більшості аналогів, GeekerApp виконує роль організаційної вітрини: після підтвердження участі платформа не пропонує інтегрованого середовища для подальшої комунікації. Відсутність внутрішніх чатів та засобів утримання контексту змушує учасників переносити координацію у сторонні месенджери.

Окрім зазначених семи критеріїв, до порівняння додано дві ознаки, що характеризують ринкову спеціалізацію кожного з аналогів, а саме наявність комерційного бронювання ігор та підтримку офлайн-заходів. Це дозволяє наочніше показати, що кожен із розглянутих сервісів сфокусований на власній вузькій ніші. Зведені результати дослідження функціональних можливостей обраних аналогів подано в порівняльній таблиці 1.1.

Таблиця 1.1.

Порівняльна характеристика аналогів

Характеристика	Roll20	StartPlaying	GeekerApp	Власна розробка
Гнучкість системи управління доступом та ролями	+	—	—	+
Підтримка розширених моделей видимості	+	+	—	+
Реалізація системи заявок	—	+	+	+

Інструментарій календарного планування	—	+	—	+
Інтеграції комунікаційних інструментів (чати або форуми)	+	—	—	+
Реалізація аудіо-відео зв'язку	+	—	—	+
Система сповіщень	—	+	+—	+
Комерційне бронювання ігор	—	+	—	+—
Підтримка офлайн-заходів	—	—	+	—

Проведений аналіз існуючих програмних рішень дозволяє зробити висновок, що сучасний ринок платформ для настільних рольових ігор є достатньо розвиненим, проте суттєво фрагментованим. Більшість розглянутих систем фокусуються на вирішенні вузькоспеціалізованих завдань: Roll20 пропонує потужний функціонал віртуального столу, але має застарілі підходи до управління LFG-спільнотами; StartPlaying забезпечує відмінну автоматизацію бронювання та календаря, проте жорстко обмежений комерційною моделлю; GeekerApp задовольняє базові потреби пошуку локальних офлайн-зустрічей, але не розрахований на адміністрування дистанційних кампаній.

Спільною архітектурною та функціональною проблемою проаналізованих аналогів є відсутність консолідованої екосистеми, яка б органічно поєднувала гнучкий організаційний менеджмент із розвиненими засобами комунікації. Для забезпечення повноцінного ігрового процесу користувачі змушені використовувати декілька розрізнених сервісів (маркетплейси для пошуку, планувальники для розкладу, сторонні месенджери типу Discord для спілкування). Це призводить до розпорошення ігрових даних, втрати контексту та ускладнення користувацького досвіду.

З огляду на виявлені недоліки існуючих аналогів, розробка власної спеціалізованої вебплатформи є актуальним та обґрунтованим завданням, яке дозволить об'єднати розрізнений інструментарій в єдину екосистему.

Головною перевагою проєкту стане інтегрована багаторівнева система доступу, що передбачає реалізацію складної рольової моделі всередині ігрових кампаній із підтримкою гнучких рівнів видимості для публічних, приватних та доступних лише за посиланням сутностей. Окрім цього, платформа забезпечить повну консолідацію комунікаційних засобів завдяки впровадженню внутрішніх синхронних чатів та підсистем реального часу для аудіо- й відеозв'язку безпосередньо у контекст кампаній та сесій, що повністю усуне потребу у сторонньому програмному забезпеченні. Нарешті, важливе значення матиме універсальний механізм онбордингу, який поєднає процеси подачі та обробки заявок із гнучким керуванням учасниками, загальною системою сповіщень та модулем календарного планування, що автоматично адаптується до різних часових поясів і функціонує без обов'язкової прив'язки до платіжних шлюзів.

Об'єднання цих функціональних модулів у межах єдиного інтерфейсу дозволить мінімізувати адміністративну рутину для організаторів ігор та забезпечить максимально зручну взаємодію користувачів із платформою.

1.3 Обґрунтування вибору архітектури системи

Проектування архітектури вебплатформи для організації НРІ вимагає врахування низки специфічних функціональних та нефункціональних вимог, що безпосередньо впливають на стабільність та масштабованість продукту. Основними чинниками, які визначають архітектурний вигляд системи, є необхідність підтримки складної багаторівневої моделі доступу, управління динамічними станами ігрових кампаній та сесій, а також забезпечення високого рівня інформаційної безпеки при ітеративному розширенні функціоналу. Вибір архітектури має забезпечити баланс між швидкістю розробки та можливістю еволюції системи без докорінного переписування програмного коду.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

Для визначення оптимальної структури було проведено порівняльний аналіз глобальних архітектурних патернів, зокрема мікросервісного та монолітного підходів.

Мікросервісна архітектура пропонує високий ступінь ізоляції, дозволяючи незалежно масштабувати окремі компоненти та використовувати різні технологічні стеки для різних модулів. Проте вона суттєво підвищує складність операційного супроводу, розгортання та забезпечення транзакційної цілісності даних. Оскільки ключові домени платформи (кампанії, сесії та права доступу) є щільно пов'язаними на рівні бізнес-логіки, використання мікросервісів на початковому етапі розробки призвело б до надмірних витрат ресурсів на організацію міжсервісної взаємодії та синхронізацію станів.

Натомість традиційна монолітна архітектура, попри простоту реалізації, несе ризики перетворення системи на «велику грудку бруду» (Big Ball of Mud) у міру зростання функціоналу [5]. Тому для розроблюваної платформи було обрано компромісну концепцію модульного моноліту. Цей підхід дозволяє зберігати переваги єдиного розгортання та спрощеного управління даними, водночас забезпечуючи чітку логічну ізоляцію функціональних блоків. Кожен модуль системи (наприклад, модуль кампаній чи модуль безпеки) відповідає за окрему бізнес-область і взаємодіє з іншими через чітко визначені інтерфейси. Це закладає фундамент для майбутньої декомпозиції системи в окремі сервіси у разі критичного зростання навантаження, що відповідає критеріям гнучкості та економічної доцільності.

Важливим етапом проєктування інтерфейсної частини системи є вибір між концепціями багатосторінкових (MPA) та односторінкових застосунків (SPA) [6]. Традиційна модель MPA, яка передбачає рендеринг кожної сторінки на стороні сервера та її повне перезавантаження при переході між маршрутами, є менш ефективною для платформ із високим ступенем інтерактивності. Для HPI-платформи, де користувач постійно взаємодіє зі складними динамічними елементами (інтерактивні календарі, панелі управління ролями, чати в реальному часі), затримки, характерні для MPA, є неприпустимими. Тому архітектурне

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

рішення базується на використанні SPA-моделі, що забезпечує безшовний перехід між екранами та високу швидкість відгуку за рахунок завантаження основного каркаса додатку один раз і подальшого динамічного оновлення лише необхідних фрагментів інтерфейсу.

Вибір конкретного методу реалізації SPA залежить від балансу між швидкістю першого завантаження, вимогами до пошукової оптимізації (SEO) та складністю інфраструктури. На сучасному етапі розвитку вебтехнологій виділяють три домінантні стратегії рендерингу: на стороні клієнта (CSR), на стороні сервера (SSR) та статичну генерацію (SSG) [7]. Порівняльний аналіз цих підходів для вибору оптимального рішення представлено в таблиці 1.2.

Таблиця 1.2.

Порівняльна характеристика стратегій рендерингу вебдодатків

Характеристика	Рендеринг на стороні клієнта	Рендеринг на стороні сервера	Статична генерація
Місце генерації HTML	У браузері користувача	На сервері (на кожен запит)	На етапі збірки проєкту
Швидкість першого завантаження	Низька (залежить від розміру JS)	Висока	Найвища
Пошукова оптимізація (SEO)	Посередня	Відмінна	Відмінна
Вимоги до сервера	Мінімальні (статичний хостинг)	Високі (Node.js сервер)	Мінімальні (статичний хостинг)
Сфера застосування	Вебдодатки з авторизацією, CRM, адмін-панелі	Маркетплейси, медіа-портали	Блоги, документація, лендінги

З огляду на результати порівняння, для реалізації платформи обрано підхід Client-Side Rendering. Оскільки основний функціонал системи (управління кампаніями, сесіями та ролями) доступний користувачам лише після автентифікації, високі показники SEO не є пріоритетними. Водночас CSR

дозволяє забезпечити максимальну інтерактивність та гнучкість управління станом додатку (state management) без надмірного навантаження на серверні ресурси.

Проте автономність клієнтської частини вимагає наявності потужного та структурованого джерела даних. Це обумовлює необхідність проектування надійної архітектури серверної частини, здатної ефективно обробляти асинхронні запити від SPA-клієнта та забезпечувати цілісність складних бізнес-правил платформи.

Для реалізації цього внутрішня структура серверної частини розроблюваної платформи базується на принципах багат шарової архітектури у межах обраного модульного моноліту [8]. Такий підхід дозволяє чітко розмежувати відповідальність компонентів системи, що є критично важливим для стабільної обробки складних бізнес-правил API-платформи. Логічна структура сервера передбачає відокремлення та послідовну взаємодію кількох основних рівнів.

Транспортний шар відповідає за прийом асинхронних запитів від SPA-клієнта, первинну валідацію вхідних даних та маршрутизацію до відповідних обробників.

Шар прикладних сервісів виконує роль оркестратора бізнес-процесів. Він координує виконання сценаріїв використання (наприклад, створення ігрової сесії або обробку заявки на вступ), взаємодіючи з іншими рівнями системи.

Доменний шар є центральним елементом архітектури. Тут зосереджена логіка перевірки бізнес-інваріантів, формалізована у вигляді політик та правил доступу. Винесення правил перевірки ролей в окремий шар дозволяє створити єдине джерело істини для системи безпеки, мінімізуючи ризики несанкціонованого доступу.

Шар доступу до даних забезпечує взаємодію з базою даних через ORM-інтерфейс. Це дозволяє абстрагуватися від конкретних SQL-запитів і працювати безпосередньо з об'єктами предметної області.

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

Окремої уваги при проектуванні потребує підсистема взаємодії в реальному часі, оскільки платформа поєднує три принципово різні за характером сценарії обміну даними: доставку системних сповіщень, текстове спілкування, а також аудіо- та відеозв'язок. Кожен із них висуває власні вимоги до напрямку передавання, частоти повідомлень і пропускну здатності, тому застосування єдиної технології для всіх випадків було б неоптимальним. З огляду на це для кожного типу взаємодії обрано окреме технологічне рішення.

Для доставки системних сповіщень обрано технологію Server-Sent Events. Сповіщення мають однонапрямлений характер, тобто передаються лише від сервера до клієнта, не потребують зворотного каналу й виникають порівняно нечасто. SSE працює поверх звичайного HTTP-з'єднання, є простішою у реалізації за повноцінний двосторонній канал і забезпечує автоматичне перевідновлення з'єднання на боці браузера, що робить її оптимальною для цього класу задач [9].

Для модуля текстового спілкування застосовано протокол WebSocket. На відміну від сповіщень, чат передбачає двосторонній обмін повідомленнями з низькою затримкою, коли і сервер, і клієнт можуть ініціювати передавання даних будь-якої миті. WebSocket встановлює стійке з'єднання поверх одного TCP-сокета, що дозволяє уникнути накладних витрат на повторне встановлення HTTP-запитів і забезпечує оперативність, потрібну для живого спілкування учасників у межах ізольованих чатів кампаній та сесій [10].

Найвищі вимоги до інфраструктури висуває підсистема аудіо- та відеозв'язку, оскільки вона передає неперервні медіапотоки великого обсягу в режимі реального часу. Для цього використано стек технологій WebRTC, що забезпечує передавання медіа з мінімальною затримкою. Проте за схемою «кожен-з-кожним» (mesh) навантаження на клієнтів зростає квадратично зі збільшенням кількості учасників, що є неприйнятним для групових ігрових сесій. Тому застосовано архітектуру селективного перенаправлення медіапотоків (SFU): кожен учасник надсилає свій потік на медіасервер, який перенаправляє його решті учасників, що суттєво знижує навантаження на

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

клієнтські пристрої та забезпечує масштабованість групових дзвінків [1, 11]. Для подолання обмежень трансляції мережевих адрес (NAT) і гарантування з'єднання навіть у складних мережевих умовах інфраструктуру доповнено TURN-сервером. Керування станом дзвінка (приєднання учасників, узгодження параметрів медіа) здійснюється через сигнальний канал на основі того ж протоколу WebSocket.

Узагальнене порівняння обраних технологій реального часу наведено в таблиці 1.3.

Таблиця 1.3.

Порівняння технологій взаємодії в реальному часі

Характеристика	SSE (сповіщення)	WebSocket (чат)	WebRTC + SFU (дзвінки)
Напрямок передавання	Однонаправлений	Двосторонній	Двосторонній
Тип даних	Текстові події	Текстові повідомлення	Аудіо-/відеопотоки
Транспорт	HTTP	TCP (єдиний сокет)	UDP (медіа) + WS (сигналінг)
Затримка	Низька	Дуже низька	Мінімальна
Призначення	Системні сповіщення	Живий чат	Групові аудіо-/відеодзвінки

Узагальнено описані архітектурні рішення наведено на рисунку 1.5. На схемі відображено клієнт-серверну організацію платформи, у якій односторінковий застосунок взаємодіє із сервером трьома незалежними каналами – синхронними запитами, односпрямованим потоком сповіщень та двостороннім з'єднанням, а безперервні аудіо- та відеопотоки передаються окремо через медіасервер з обходом трансляції мережевих адрес. Серверну частину організовано як модульний моноліт із чітким поділом на транспортний, сервісний і доменний шари та шар доступу до даних, що взаємодіють із реляційною базою даних і сховищем кешу.

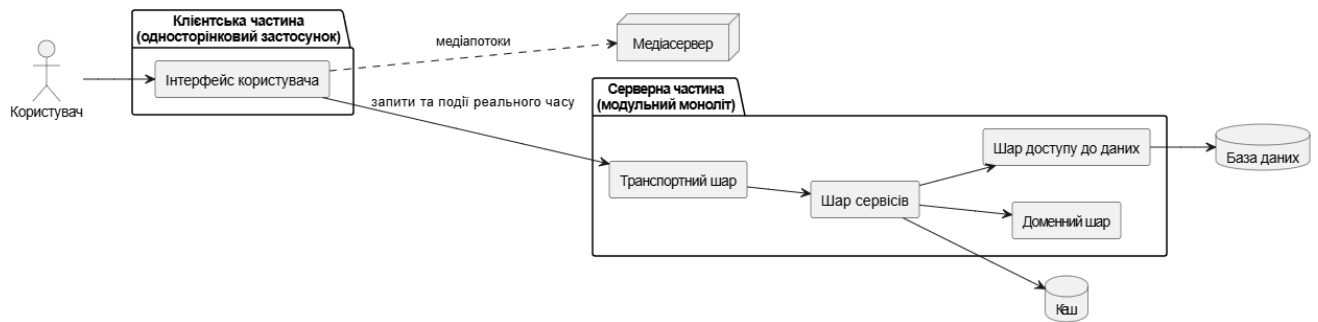


Рис. 1.5. Узагальнена архітектура системи

Таким чином, обрана клієнт-серверна архітектура – односторінковий застосунок на боці клієнта та модульний моноліт із багат шаровою структурою на боці сервера є компромісно-оптимальною для поточного масштабу платформи. Вона забезпечує високу швидкість взаємодії, надійний контроль над складною доменною логікою ролей і закладає можливості для ітеративного розвитку. Водночас диференційований підхід до взаємодії в реальному часі, за якого для кожного сценарію обрано окрему профільну технологію, дозволяє оптимально покрити всі режими комунікації без надлишкових інфраструктурних витрат. Запропонований архітектурний стек створює надійний фундамент для автоматизації процесів координації учасників настільних рольових ігор.

1.4 Обґрунтування вибору інструментальних засобів

Вибір програмних засобів реалізації вебплатформи для організації НРІ безпосередньо впливає з архітектурних рішень, прийнятих у попередньому параграфі, зокрема використання концепції односторінкового застосунку та клієнт-серверної взаємодії через REST API та канали реального часу. Основними критеріями вибору інструментарію є забезпечення високої інтерактивності інтерфейсу, стабільність роботи в умовах реального часу, безпека передачі даних та наявність розвиненої спільноти розробників для підтримки технологій.

Для реалізації клієнтської частини у роботі обрано бібліотеку React. Вибір даної технології обумовлений компонентно-орієнтованим підходом до побудови інтерфейсів, що ідеально підходить для створення складних модулів, таких як інтерактивні картки сесій, системи керування заявками та дашборди ігрових кампаній. Використання механізму Virtual DOM дозволяє React ефективно

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк. 26
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

оновлювати лише необхідні елементи сторінки без її повного перезавантаження, що критично важливо для динамічного досвіду та мінімізації навантаження на ресурси браузера [12].

Інструментом автоматизації збирання та середовищем розробки обрано Vite. На відміну від традиційних засобів (наприклад, Webpack), Vite забезпечує миттєве «гаряче» перезавантаження модулів (HMR) [13]. Це дозволяє значно пришвидшити цикл розробки та тестування інтерфейсних рішень платформи.

Для ефективного управління станом додатку обрано бібліотеку Zustand. Враховуючи складність рольової моделі системи, Zustand дозволяє створити легковагові та масштабовані сховища даних [14]. Це забезпечує швидку синхронізацію між різними частинами інтерфейсу (наприклад, миттєве оновлення імені та аватара користувача в навігаційній панелі, картці профілю та налаштуваннях відразу після зміни даних користувачем) без надмірного ускладнення архітектури коду, характерного для інших рішень.

Окрім локального стану інтерфейсу, платформа активно працює із серверними даними, тому для їх отримання, кешування та синхронізації застосовано бібліотеку TanStack React Query [15]. Вона бере на себе керування станом запитів до REST API (кешування відповідей, фонове оновлення, обробку станів завантаження та помилок), що дозволяє лишити у сховищі Zustand винятково клієнтський стан інтерфейсу. Маршрутизацію в межах односторінкового застосунку реалізовано засобами бібліотеки React Router.

Візуальна стилізація та реалізація дизайну користувацького інтерфейсу виконується за допомогою утилітарного CSS-фреймворку Tailwind CSS [16]. Використання підходу «utility-first» дозволяє створювати гнучкі та легкі для підтримки стилі, що забезпечує швидке завантаження сторінок та стабільність візуальної складової. Даний вибір обґрунтований необхідністю підтримки єдиної стилістики складних графічних елементів платформи, таких як інтерактивні календарі та багатокомпонентні панелі управління, що вимагають високої точності позиціювання та візуальної цілісності на будь-якій робочій станції користувача.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

Для розробки серверної частини обрано платформу Node.js та вебфреймворк Express. Вибір Node.js зумовлений її подієво-орієнтованою моделлю та неблокуючим введенням/виведенням, що є оптимальним для додатків з великою кількістю одночасних запитів, характерних для ігрових платформ [17]. Використання середовища виконання Node.js дозволяє забезпечити високу швидкість обробки даних та уніфікацію мови програмування на всіх рівнях розробки системи. Фреймворк Express виступає як легковаговий інструмент для побудови RESTful API, що дозволяє чітко структурувати логіку обробки маршрутів, автентифікації та взаємодії з бізнес-логікою [18].

Для реалізації описаних у попередньому параграфі каналів взаємодії в реальному часі застосовано відповідні бібліотеки. Двосторонній обмін повідомленнями у чатах та сигналінг дзвінків реалізовано за допомогою бібліотеки ws, легковагової реалізації протоколу WebSocket для середовища Node.js [19]. Передавання аудіо- та відеопотоків за архітектурою SFU забезпечує медіасервер mediasoup у поєднанні з клієнтською бібліотекою mediasoup-client, а для подолання обмежень трансляції мережевих адрес використано TURN-сервер coturn [11, 20]. Доставку системних сповіщень реалізовано через механізм Server-Sent Events без додаткових залежностей [9].

Безпеку доступу та захист облікових записів на платформі забезпечують механізм токенів JWT для надійної автентифікації сесій, а також бібліотека bcrypt, яка використовується для стійкого хешування паролів перед їх збереженням [21, 22].

Окрему процедуру верифікації нових користувачів, а також підтвердження їхніх дій через електронну пошту, реалізовано в системі за допомогою спеціалізованих засобів nodemailer [23].

Важливою складовою серверної архітектури є використання Prisma ORM. Це дозволяє абстрагуватися від написання сирих SQL-запитів, забезпечуючи безпечну типізацію та автоматичну генерацію міграцій [24]. Використання Prisma значно пришвидшує процес розробки та зменшує кількість помилок при

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

роботі зі складною схемою даних, яка включає численні зв'язки між користувачами, ігровими кампаніями та сесіями.

Як основну систему керування базами даних обрано реляційну базу даних PostgreSQL. Дана СКБД характеризується високою надійністю, підтримкою складних запитів та відповідністю принципам ACID, що є критично важливим для збереження цілісності ігрових даних та конфіденційної інформації користувачів [25]. Структура даних платформи НРІ передбачає розгалужену ієрархію доступів, для якої реляційна модель PostgreSQL є найбільш ефективною.

Для оптимізації роботи системи та зниження навантаження на основну БД використовується сховище структур даних у пам'яті Redis. Воно застосовується для кешування сесій користувачів, реалізації механізмів обмеження частоти запитів та зберігання тимчасових даних [26]. Комбінація PostgreSQL та Redis дозволяє досягти оптимального балансу між надійним довготривалим зберіганням даних та високою швидкістю відгуку інтерфейсу.

Для забезпечення стабільності розгортання та ізоляції середовища виконання всіх компонентів системи обрано технологію контейнеризації Docker та інструмент оркестрації Docker Compose. Використання Docker дозволяє упакувати клієнтську частину, серверний додаток, базу даних PostgreSQL та сховище Redis у незалежні контейнери, що гарантує ідентичність роботи платформи як на локальних машинах розробників, так і на сервері [27]. Docker Compose використовується для опису взаємозв'язків між сервісами, автоматизації їх запуску та налаштування мережевої взаємодії, що значно спрощує адміністрування платформи та її масштабування у майбутньому [28].

Наведені технології та інструменти відповідають фактичному середовищу розгортання платформи. Слід відзначити високі вимоги серверу до ресурсів системи, в особливості вимоги підсистеми відеозв'язку. Тим не менш, поточних ресурсів достатньо для цільового масштабу. Детальніший аналіз необхідних ресурсів проведено в останньому розділі.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до першого розділу

У першому розділі проаналізовано предметну область організації настільних рольових ігор та обґрунтовано вибір технологій реалізації майбутньої платформи. Сформульовано задачу дослідження, проведено порівняльний аналіз наявних рішень, обрано архітектурний підхід та інструментальні засоби розробки.

Під час постановки задачі визначено ключову прикладну проблему – фрагментацію інструментарію, через яку координація учасників, планування сесій і комунікація здійснюються в розрізнених сервісах. Окреслено основні функціональні напрями платформи: управління кампаніями та сесіями, гнучку рольову модель і вбудовані засоби комунікації, а також наскрізну вимогу інформаційної безпеки.

У результаті аналізу аналогів Roll20, StartPlaying та GeekerApp встановлено, що кожне з рішень зосереджене на вузькій ніші й жодне не поєднує гнучкого організаційного менеджменту з розвиненими засобами комунікації. Це підтвердило доцільність розробки власної спеціалізованої платформи, що консолідує ці функції в єдиному середовищі.

Обґрунтовано вибір клієнт-серверної архітектури: для клієнта обрана модель односторінкового застосунку, для сервера – модульний моноліт із багатошаровою структурою. Для підсистеми взаємодії в реальному часі застосовано диференційований підхід, за якого для системних сповіщень обрано технологію SSE, для чату та сигналінгу WebSocket, а для групового відеозв'язку стек WebRTC за архітектурою SFU.

Обґрунтовано технологічний стек реалізації, що включає бібліотеку React і збирач Vite на боці клієнта, середовище Node.js із фреймворком Express на боці сервера, систему керування базами даних PostgreSQL разом із Prisma ORM та сховище Redis.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ТА КОМУНІКАЦІЇ ГРАВЦІВ В НРІ

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

Основою для успішного проєктування будь-якого програмного комплексу є детальний аналіз вимог до майбутнього продукту, що дозволяє визначити його функціональні та нефункціональні характеристики з урахуванням потреб користувачів різних типів [29]. У контексті розробки веборієнтованої платформи для організації НРІ, головним завданням є створення зручного середовища для взаємодії між організаторами ігор та їх учасниками. Відповідно до цього, проєктована система повинна підтримувати складну логіку доступів та статусів для ігрових кампаній та окремих сесій, яка тісно пов'язана із розгалуженою системою ролей користувачів, серед яких виділяються специфічні майстер, гравець, власник, а також загальні адміністратор та користувач. Такий підхід вимагає чіткого розмежування можливостей кожного типу користувача, щоб забезпечити коректне управління ігровим процесом та захист даних.

Задля формування ядра програмного продукту та опису конкретних дій, які користувачі зможуть виконувати в межах платформи, сформулюємо функціональні вимоги до системи. Оскільки основна мета застосунку полягає в автоматизації організаційних процесів НРІ, система має надавати інструментарій для повного циклу управління ігровими подіями. Це включає створення нових ігрових світів, планування розкладу конкретних ігрових зустрічей, а також адміністрування складу учасників. Особлива увага приділяється механізмам призначення та зміни ролей, що дозволяє не тільки гнучко передавати права управління від творця та власника кампанії чи сесії до безпосереднього ведучого гри, а ще й задовольняти неочевидні та унікальні ігрові ситуації (наприклад, пошук майстра групою гравців). Деталізований перелік основних функціональних вимог, сформований на основі аналізу предметної області, наведено у таблиці 2.1.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

Таблиця 2.1.

Функціональні вимоги до розроблюваної системи

Назва вимоги	Опис вимоги	Пріоритет
Реєстрація та авторизація	Можливість створення персонального облікового запису користувача, безпечного входу в систему та управління профілем.	Високий
Управління кампаніями	Надання інструментів для створення, редагування та перегляду ігрових кампаній з відповідною прив'язкою до творця.	Високий
Управління сесіями	Можливість планування нових ігрових зустрічей в межах існуючої кампанії та окремих сесій формату one-shot, зміни їх статусів та розкладу.	Високий
Розподіл ролей та прав доступу	Забезпечення коректної роботи розгалуженої системи специфічних ролей (GM, PLAYER, OWNER) та автоматичне розмежування доступу до функціоналу платформи.	Високий
Текстова комунікація	Надання ізольованих чатів у межах кампаній та сесій з підтримкою редагування повідомлень і системних подій.	Високий
Аудіо- та відеозв'язок	Можливість проведення групових відеодзвінків безпосередньо у вебдодатку під час ігрової сесії.	Високий
Система сповіщень	Багатоканальне інформування користувачів (у застосунку, Telegram) про ключові події платформи.	Середній

Окрім функціональних можливостей, критично важливим аспектом є дотримання нефункціональних вимог, зокрема щодо часових характеристик реакції системи на дії користувача. Зважаючи на те, що вебдодаток

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

розробляється як SPA, система повинна забезпечувати миттєвий відгук інтерфейсу під час навігації між сторінками кампаній та розкладом сесій. Крім того, нефункціональні вимоги охоплюють параметри безпеки: усі конфіденційні дані користувачів, включаючи паролі та особисту інформацію, мають бути надійно зашифровані або захищені. Програмний комплекс також повинен бути масштабованим, що передбачає можливість розширення бази даних та додавання нових модулів (наприклад, інтеграції сторонніх сервісів чи генераторів персонажів) без необхідності повної переробки існуючої архітектури.

Для коректної роботи описаного функціоналу необхідно детально розглянути та розмежувати вищезгадані ролі. Згідно з бізнес-логікою та специфікою НРІ, ролі зареєстрованого Користувача вимагають подальшої деталізації в межах ігрової сесії або кампанії. Будь-який зареєстрований користувач може виступати у трьох специфічних ролях: гравець, майстер та власник.

Хоча в більшості випадків майстер та власник сесії будуть однією особою, збереження ролі власника є критично важливим. Це дозволяє організувати зустрічі групам, які не мають постійного майстра, забезпечуючи механізм для керування заявками та регулювання діяльності групи. Своєю чергою, виокремлення ролі майстра відображає функціональний розподіл під час безпосереднього проведення гри: керування станом сесії, складом учасників та відеодзвінком. Така роль також закладає точку розширення для майбутньої інтеграції із суміжним модулем віртуального ігрового столу, що розробляється поза межами цієї роботи.

Перейдемо до описів конкретних акторів. Загальний користувач має базовий набір можливостей, що дозволяють повноцінно брати участь у житті платформи. До його ключових функцій належить керування профілем, що включає як основні дані, так і можливість додавання інтеграцій (наприклад Telegram бот). Важливою частиною є керування гаманцем, що охоплює поповнення та оплату сесій. Також користувач залучений до основного

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

функціоналу платформи: створення сесій або кампаній та приєднатися до вже існуючих. Короткий опис акторів, для наочності, представлено в таблиці 2.2.

Таблиця 2.2.

Опис акторів системи та їхніх ключових функцій

Актор		Опис та ключові обов'язки	
Гість		Незареєстрований користувач має доступ лише до процедур реєстрації та входу в систему; ігровий простір повністю прихований за бар'єром автентифікації.	
Користувач	Гравець	Користувач, чия заявка на участь була схвалена. Зосереджений на безпосередній участі в грі. Має доступ до закритих матеріалів сесій, внутрішнього чату та активного відеодзвінка під час сесії.	Базовий стан авторизованого користувача. Має доступ до управління власним профілем, пошуку публічних ігор, перегляду загальної статистики та подачі заявок на участь у сесіях та кампаніях.
	Майстер	Організатор та модератор безпосереднього ігрового процесу. Володіє правами на запуск сесії та відеодзвінка, зміну статусів учасників, видалення учасників, а також завершення і скасування сесії.	
	Власник	Творець ігрової кампанії. Здійснює управління складом учасників та опрацювання заявок, має ексклюзивне право призначати або змінювати майстра гри, передавати володіння кампанією. Може видаляти окремі сесії своєї кампанії.	
Адміністратор		Здійснює глобальне управління платформою. Має права на модерацію контенту та управління обліковими записами користувачів.	

Окремо треба виділити гостя, або незареєстрованого користувача. На теперішньому етапі розвитку платформи було прийнято рішення приховати ігровий простір від гостя за бар'єром реєстрації. Це зумовлено загальними архітектурними особливостями платформи та її вузькоспеціалізованим характером. В майбутньому незареєстровані користувачі будуть потрапляти на цільову сторінку з описом платформи, після чого перенаправлятимуться на реєстрацію для безпосереднього залучення до ігрового простору.

Для формалізації функціональних вимог та опису взаємодії між виділеними акторами та системою було розроблено UML-діаграми варіантів використання [30]. Враховуючи розгалужену систему ролей, архітектуру прецедентів логічно розділено на дві підсистеми: глобальну системну взаємодію (базові операції авторизації та адміністрування) та доменну взаємодію (специфічні процеси ігрових кампаній та сесій).

На рисунку 2.1 наведено діаграму варіантів використання для базових системних ролей: гостя та адміністратора.

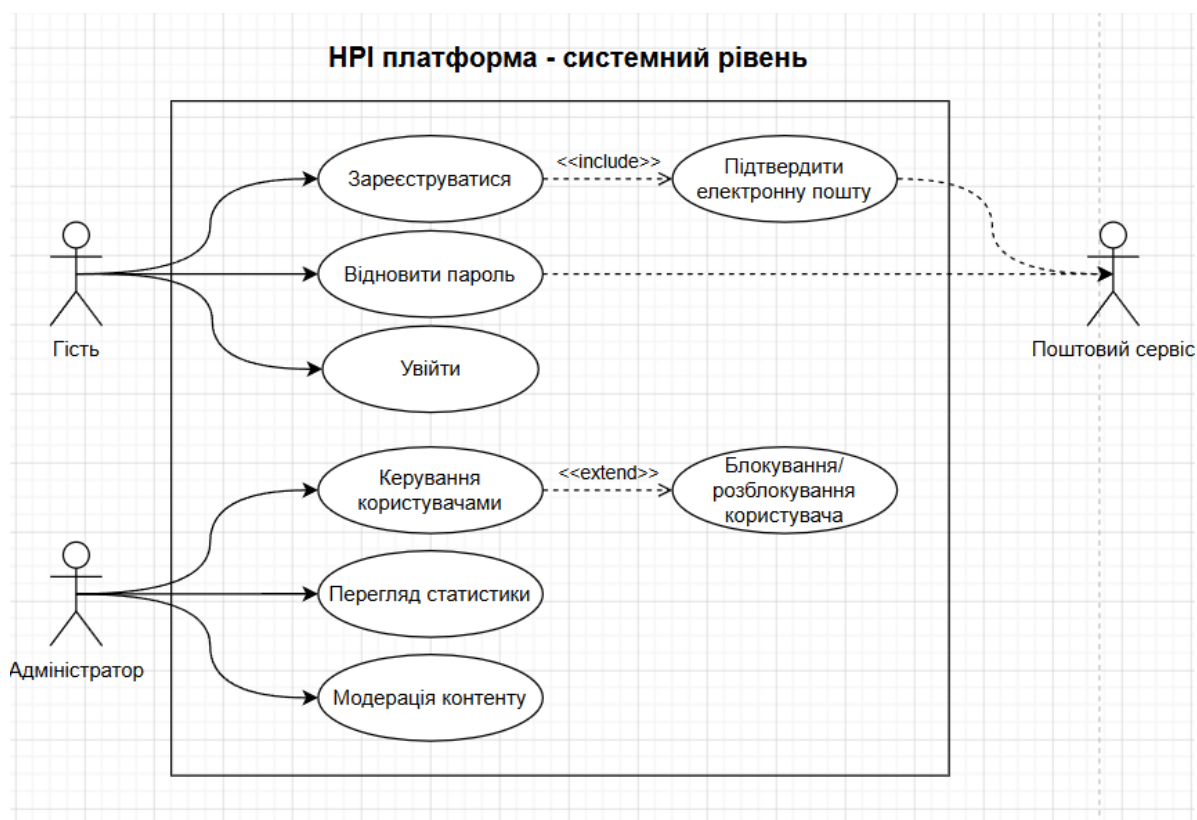


Рис. 2.1. Діаграма варіантів використання для глобальних системних ролей

Як видно з діаграми, неавторизований користувач має мінімальний набір прецедентів, зосереджених навколо процесу автентифікації: «Зареєструватися» та «Увійти в систему». Прецедент «Зареєструватися» є точкою входу, після виконання якого (з обов'язковим підтвердженням електронної пошти) гість набуває статусу користувача. У перспективі для незареєстрованих відвідувачів передбачено цільову сторінку (лендінг) з описом платформи, проте на поточному етапі її винесено за межі реалізованого функціоналу.

На рисунку 2.2 представлена розширена діаграма варіантів використання, яка описує доменну логіку платформи та взаємодію специфічних ігрових ролей: користувача, гравця, майстра гри та власника кампанії.

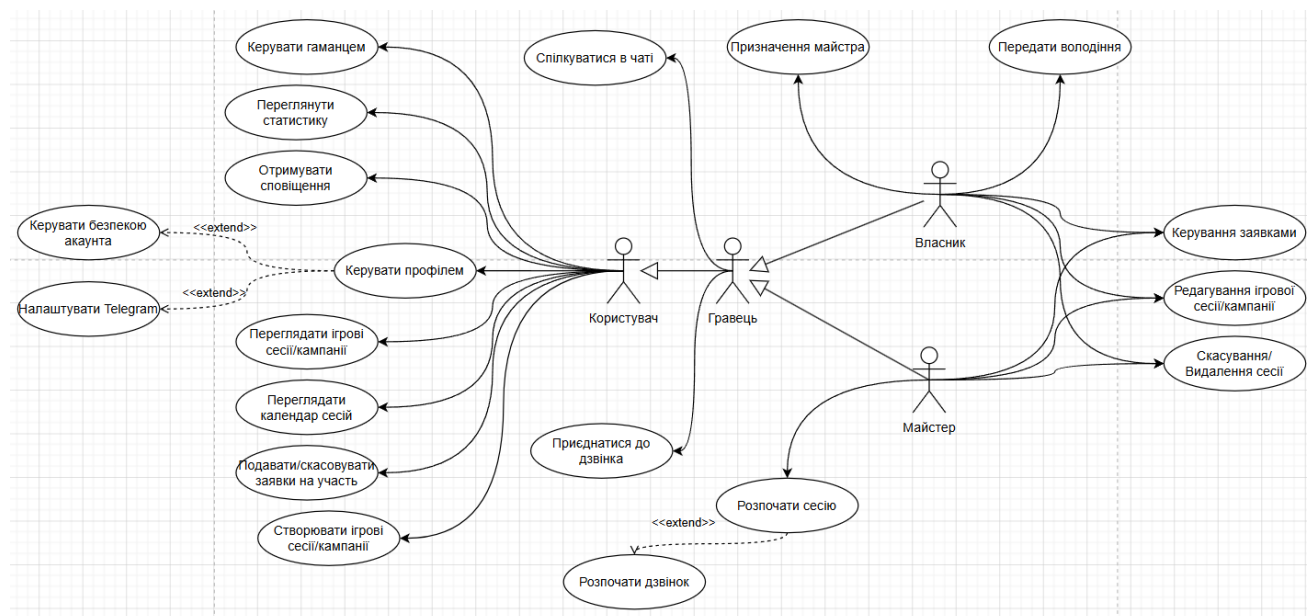


Рис. 2.2. Діаграма варіантів використання для доменних ігрових ролей

Ця діаграма відображає складну ієрархію доступів. Авторизований користувач має можливість керувати власним профілем, взаємодіяти з віртуальним гаманцем, а також ініціювати базові ігрові процеси: приєднуватися до існуючих ігор (подача заявок) або створювати нові сесії чи кампанії. Створення нової ігрової сутності автоматично наділяє користувача роллю власника.

Цей статус включає специфічні адміністративні прецеденти в межах кампанії: він має ексклюзивне право керувати загальним складом групи

(«Керувати заявками»), формувати розклад сесій та, найголовніше, здійснювати прецедент «Призначення майстра».

Прецеденти майстра гри зосереджені на безпосередньому проведенні ігрового процесу. Прецеденти майстра гри зосереджені на проведенні та керуванні станом сесії: зміні статусів учасників, завершенні і скасуванні сесії, а також керуванні дзвінком. Дія «Почати зустріч» також надає можливість ініціювати запуск відео- та аудіосервісів через зв'язок <<extend>>.

Після визначення функціональних вимог та сценаріїв використання системи, наступним етапом логічного проектування є розробка об'єктно-орієнтованої моделі. Для візуалізації статичної структури системи, її базових сутностей, їхніх атрибутів, методів та взаємозв'язків між ними побудовано UML-діаграму класів. Ця модель є перехідною ланкою між вимогами користувачів та фізичною схемою бази даних додатку, оскільки вона не лише структурує дані, а й інкапсулює бізнес-логіку через відповідні методи.

Особливістю спроектованої моделі є наявність асоціативних класів, які вирішують проблему відношення «багато-до-багатьох» та дозволяють уникнути надмірності даних.

Такий підхід до моделювання забезпечує високий рівень нормалізації даних на логічному рівні. Використання методів у межах класів дозволяє чітко розмежувати відповідальність між окремими модулями системи, що спрощує подальшу реалізацію контролерів на стороні сервера. Це, у свою чергу, мінімізує зачеплення (coupling) між компонентами платформи та закладає надійний фундамент для її подальшого масштабування. Крім того, чітка інкапсуляція бізнес-логіки дозволяє гнучко модифікувати правила взаємодії користувачів без необхідності перебудови всієї архітектури бази даних.

На рисунку 2.3 наведено діаграму класів доменної області платформи для настільних рольових ігор.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

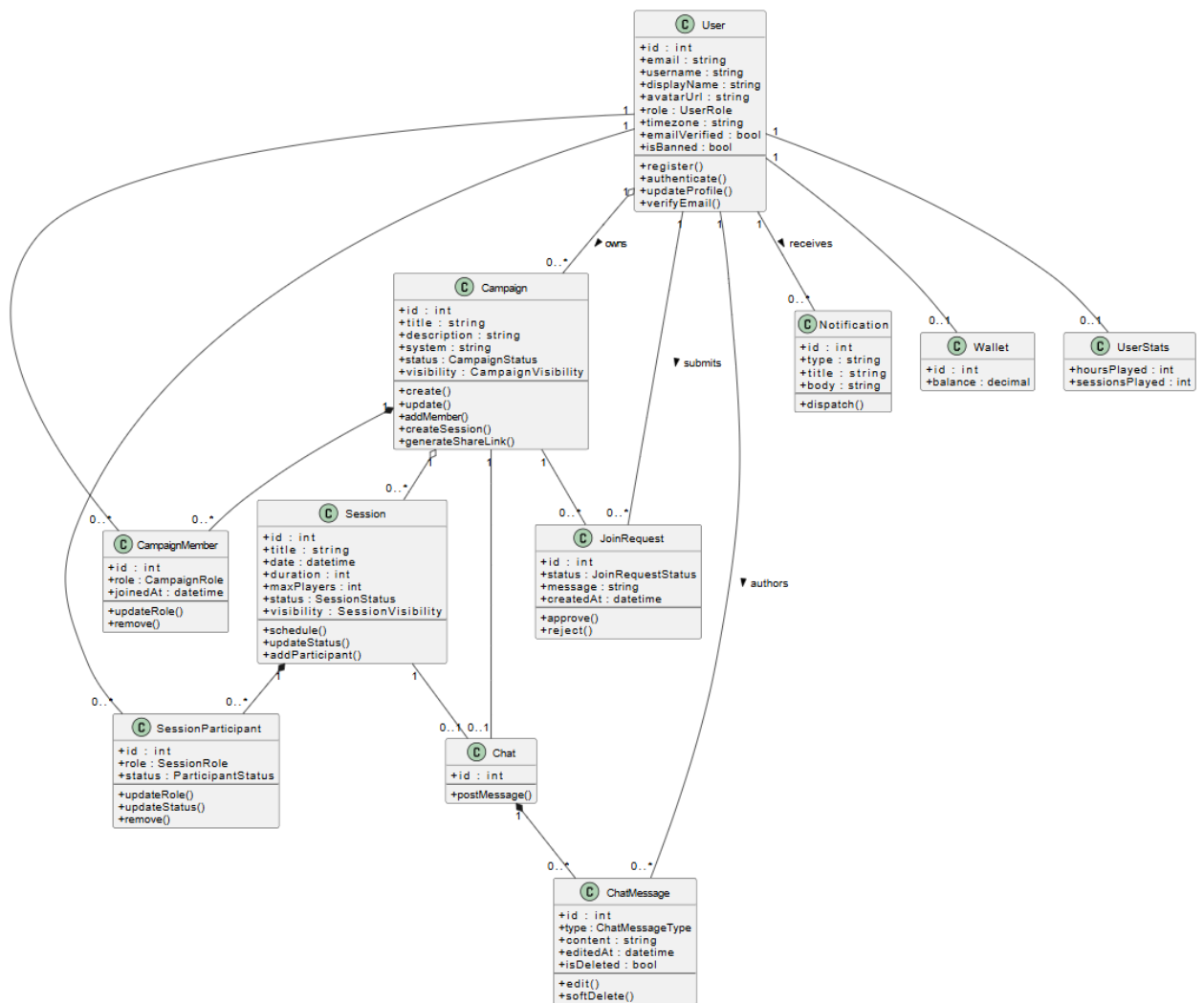


Рис. 2.3. Об'єктно-орієнтована модель (діаграма класів) системи

Архітектура платформи побудована навколо кількох ключових сутностей, які інкапсулюють дані та відповідну бізнес-логіку.

Клас User є базовою сутністю системи, що містить ідентифікаційні дані (ідентифікатор, електронна пошта, ім'я) та визначає глобальний рівень доступу через атрибут role (наприклад, ADMIN або USER). Клас містить стандартні методи управління життєвим циклом облікового запису (register, authenticate, updateProfile, verifyEmail). Він має зв'язки типу "один-до-багатьох" з іншими сутностями, оскільки один користувач може бути творцем багатьох кампаній або учасником багатьох ігрових сесій.

Клас Campaign представляє довготривалу ігрову сутність. Містить загальну інформацію про ігровий світ (title, description, status) та логічно підпорядкована конкретному користувачеві через ownerId. Для управління своїм

станом клас використовує такі методи, як `addMember` (додавання нового учасника) та `createSession` (ініціалізація нової ігрової зустрічі в межах кампанії).

Клас `Session` відображає конкретну ігрову подію. Може існувати як незалежно (`one-shot`), так і бути прив'язаною до конкретної кампанії через `campaignId`. Клас керує часовими рамками (`date`, `duration`) та станами видимості (`visibility`). Метод `updateStatus` відповідає за перехід сесії від етапу планування до активної гри.

Класи `CampaignMember` та `SessionParticipant` є критично важливими для реалізації розгалуженої матриці доступу, описаної раніше. Вони зберігають доменні ролі (`OWNER`, `GM`, `PLAYER`) конкретного Користувача (`userId`) у межах конкретної Кампанії (`campaignId`) або Сесії (`sessionId`). Наявність методів `updateRole` та `remove` дозволяє гнучко управляти складом учасників та передавати права на управління ігровим процесом без зміни глобального статусу користувача в системі.

Спроектowana об'єктно-орієнтована структура забезпечує цілісність та несуперечливість даних, усуває надмірність інформації та дозволяє системі ефективно обробляти складні сценарії взаємодії між учасниками ігрового процесу.

2.2 Розробка бази даних системи

Для збереження даних та забезпечення стабільної роботи розроблюваної платформи для настільних рольових ігор обрано реляційну систему керування базами даних PostgreSQL. Ця СУБД відрізняється високою надійністю, відповідністю стандартам SQL та ефективною роботою зі складними зв'язками між таблицями. В якості прошарку доступу до даних використовується Prisma ORM, що дозволяє декларативно описувати об'єктно-реляційне відображення, автоматизує створення міграцій та забезпечує сувору типізацію запитів на рівні серверної логіки.

Відповідно до об'єктно-орієнтованої моделі предметної області, описаної у попередньому параграфі, дані було нормалізовано, що дозволило сформулювати оптимальну фізичну структуру бази даних платформи. Основу системи

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

становлять таблиці, які відповідають за збереження профілів користувачів та їхніх персональних метрик. Зокрема, головна таблиця User містить облікові записи, налаштування профілів, системні ролі та автентифікаційні дані. З нею тісно пов'язані відокремлена таблиця UserStats, призначена для відстеження ігрової статистики користувачів (кількості зіграних годин та сесій), а також таблиця Wallet, яка зберігає актуальну інформацію про внутрішній фінансовий баланс. Будь-які зміни балансу та рух коштів фіксуються у таблиці Transaction, що забезпечує прозору історію фінансових операцій користувача.

Центральним ядром організації ігрового процесу є сукупність таблиць, які описують логіку кампаній та сесій. Загальні дані про довготривалі ігри (назва, опис, система правил, статус та параметри видимості) зберігаються у таблиці Campaign. Для реалізації відношення «багато-до-багатьох» між користувачами та кампаніями, а також для чіткої фіксації ігрової ролі учасника використовується зв'язкова таблиця CampaignMember. Якщо кампанія є закритою, процес вступу координується через таблицю JoinRequest, яка забезпечує функціонування системи заявок та зберігає історію їх розгляду. Безпосередній розклад зустрічей, їх тривалість, вартість і статус проведення описуються в таблиці Session, тоді як безпосередній склад гравців на кожну конкретну зустріч та стан їх підтвердження визначаються через зв'язкову таблицю SessionParticipant.

Комунікаційна складова та механізми взаємодії користувачів реалізовані через окремі спеціалізовані модулі. Історія текстового спілкування у межах кампаній чи окремих сесій зберігається за допомогою таблиць Chat і ChatMessage. Паралельно на платформі функціонує розвинена підсистема багатоканальних сповіщень із гарантованою доставкою, роботу якої забезпечує комплекс взаємопов'язаних таблиць: Notification, NotificationRecipient, NotificationPreference, OutboxEvent та DeliveryAttempt.

Нарешті, безпеку, контроль доступу та стабільність сеансів гарантує набір службових таблиць. До них належать RefreshToken, EmailVerificationToken та EmailChangeToken, які безпосередньо керують життєвим циклом сеансів

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

авторизації, забезпечують механізми оновлення токенів безпеки та відповідають за верифікацію адрес електронної пошти користувачів.

Детальний опис структури збереження даних кожної з наведених таблиць, включно з переліком полів, типами даних, їхнім призначенням та обмеженнями, наведено у таблицях 2.3 – 2.12.

Центральне місце у структурі бази даних посідає таблиця облікових записів User, з якою прямо або опосередковано пов'язані всі інші сутності системи (табл. 2.3).

Таблиця 2.3.

Опис структури таблиці User

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор користувача
email	String	Електронна пошта (унікальне поле для авторизації)
username	String	Логін користувача (унікальне псевдо)
password	String	Хешоване значення пароля
role	Enum (UserRole)	Глобальна системна роль (USER, ADMIN)
avatarUrl	String	Посилання на зображення профілю
displayName	String	Відображуване ім'я користувача
bio	String	Текстовий опис профілю ("Про себе")
timezone	String	Часовий пояс для розрахунку розкладу ігор
language	String	Мова інтерфейсу (за замовчуванням "uk")
isDeleted	Boolean	Прапорець логічного видалення акаунта (Soft Delete)
createdAt / updatedAt	DateTime	Системні часові мітки створення та оновлення запису
lastActiveAt	DateTime	Час останньої активності на платформі

Таблиця 2.4 демонструє структуру сутності UserStats, яку в межах архітектурного паттерну проєктування бази даних було винесено в окрему таблицю з відношенням «один-до-одного» до користувача з метою оптимізації продуктивності та уникнення надлишкового блокування головної таблиці User під час частих оновлень ігрових метрик.

Таблиця 2.4.

Опис структури таблиці UserStats

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор запису статистики
userId	Int (FK)	Зовнішній ключ, посилання на таблицю User
sessionsPlayed	Int	Загальна кількість зіграних сесій у ролі гравця
hoursPlayed	Int	Акумуляована кількість годин ігрового часу

Організаційний контур даних формують таблиці кампаній та сесій разом зі зв'язковими таблицями учасників, які реалізують гнучку рольову модель (табл. 2.5 – 2.8).

Базові параметри довготривалих ігрових світів, включаючи системні налаштування приватності та механізми безпечного доступу, детально описано в таблиці 2.5.

Таблиця 2.5.

Опис структури таблиці Campaign

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор кампанії
title	String	Назва ігрової кампанії
description	String	Детальний опис ігрового світу та правил
system	String	Використовувана ігрова система
status	Enum (CampaignStatus)	Поточний стан

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

visibility	Enum (CampaignVisibility)	Рівень доступу до кампанії
shareTokenHash	String	Унікальний криптографічний хеш для приєднання за посиланням
ownerId	Int (FK)	Зовнішній ключ, вказує на користувача-творця кампанії
createdAt	DateTime	Часова мітка створення кампанії
updatedAt	DateTime	Часова мітка останньої зміни кампанії

Для реалізації гнучкого розмежування прав доступу користувачів у межах конкретної кампанії та збереження історії їхнього приєднання використовується зв'язкова сутність, структуру якої наведено в таблиці 2.6.

Таблиця 2.6.

Опис структури таблиці CampaignMember

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор запису членства
userId	Int (FK)	Зовнішній ключ, ідентифікатор користувача
campaignId	Int (FK)	Зовнішній ключ, ідентифікатор кампанії
role	Enum (CampaignRole)	Роль користувача у контексті кампанії
joinedAt	DateTime	Дата та час приєднання користувача до кампанії

Специфікація окремих ігрових подій (як планових сесій у межах кампаній, так і самостійних ваншотів) разом із часовими та фінансовими метриками зустрічі зафіксована у структурі таблиці 2.7.

Таблиця 2.7.

Опис структури таблиці Session

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор ігрової сесії
campaignId	Int (FK)	Зовнішній ключ кампанії (може бути порожнім для One-Shot ігор)
title	String	Назва конкретної ігрової зустрічі
description	String	Опис подій або вимог до сесії
date	DateTime	Запланована дата та час початку гри
duration	Int	Запланований час гри в хвилинах
status	Enum (SessionStatus)	Статус проведення
system	String	Система обраної НРІ
visibility	Enum (SessionVisibility)	Рівень приватності сесії
maxPlayers	Int	Ліміт кількості учасників сесії
price	Decimal	Ціна сесії за людину
heldAmount	Decimal	Загальна кількість зарезервованих коштів
ownerId	Int (FK)	Ідентифікатор користувача, який запланував сесію

Оперативний облік складу гравців на кожну конкретну ігрову зустріч, контроль лімітів та фіксація поточного стану підтвердження їхньої участі здійснюється відповідно до архітектури таблиці 2.8. Важливо не плутати статус участі у конкретній сесії (керований через сутність SessionParticipant) із загальним запитом на приєднання до кампанії (JoinRequest), оскільки перший відображає готовність та фінансовий стан гравця щодо відвідування окремої ігрової зустрічі (включаючи одноразові ігри), тоді як другий визначає довгострокове право користувача входити до складу ігрової групи загалом.

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Таблиця 2.8.

Опис структури таблиці SessionParticipant

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор запису участі
sessionId	Int (FK)	Зовнішній ключ, ідентифікатор конкретної сесії
userId	Int (FK)	Зовнішній ключ, ідентифікатор гравця
role	Enum (SessionRole)	Роль у конкретній грі (зазвичай синхронізується з роллю кампанії)
status	Enum (ParticipantStatus)	Статус підтвердження участі
isGuest	Boolean	Відображає чи є учасник гостем

Окрему групу сутностей утворюють таблиці комунікаційної підсистеми. Внутрішні чати прив'язуються до кампанії або сесії та накопичують історію повідомлень, а підсистема сповіщень забезпечує надійне інформування користувачів про ключові події платформи (табл. 2.9 – 2.12).

Логічне групування текстових каналів та їхня безпосередня ізоляція у контексті конкретної ігрової кампанії або окремої сесії реалізована через сутність, структуру якої наведено в таблиці 2.9.

Таблиця 2.9.

Опис структури таблиці Chat

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор чату
campaignId	Int (FK)	Зовнішній ключ кампанії (для чату кампанії)
sessionId	Int (FK)	Зовнішній ключ сесії (для чату сесії)
createdAt	DateTime	Час створення чату

Для збереження хронології текстової взаємодії користувачів, підтримки системних системних логів та реалізації запланованої в майбутньому системи забезпечення механізму м'якого видалення повідомлень використовується таблиця 2.10.

Таблиця 2.10.

Опис структури таблиці ChatMessage

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор повідомлення
chatId	Int (FK)	Зовнішній ключ чату
authorId	Int (FK)	Автор повідомлення (порожній для системних)
type	Enum (ChatMessageType)	Тип повідомлення (USER, SYSTEM)
content	String	Текст повідомлення
isDeleted	Boolean	Прапорець логічного видалення
editedAt	DateTime	Час останнього редагування

Централізований облік системних та доменних подій платформи здійснюється відповідно до структури таблиці 2.11.

Таблиця 2.11.

Опис структури таблиці Notification

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор сповіщення
eventKey	String	Ключ доменної події, що спричинила сповіщення
severity	Enum (NotificationSeverity)	Рівень важливості (INFO...CRITICAL, SECURITY)
category	String	Категорія сповіщення
title / body	String	Заголовок і текст сповіщення
createdAt	DateTime	Час формування

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

Зв'язування згенерованих системою сповіщень із конкретними користувачами-отримувачами покладено на сутність, описану в таблиці 2.12.

Таблиця 2.12.

Опис структури таблиці NotificationRecipient

Поле	Тип даних	Призначення
id	Int (PK)	Ідентифікатор запису доставки
notificationId	Int (FK)	Зовнішній ключ сповіщення
userId	Int (FK)	Зовнішній ключ отримувача
status	Enum (RecipientStatus)	Стан (ACTIVE, ARCHIVED)
readAt	DateTime	Час прочитання

Загальну структуру бази даних та зв'язки між таблицями наведено на схемах даних (рис. 2.4 – 2.5).

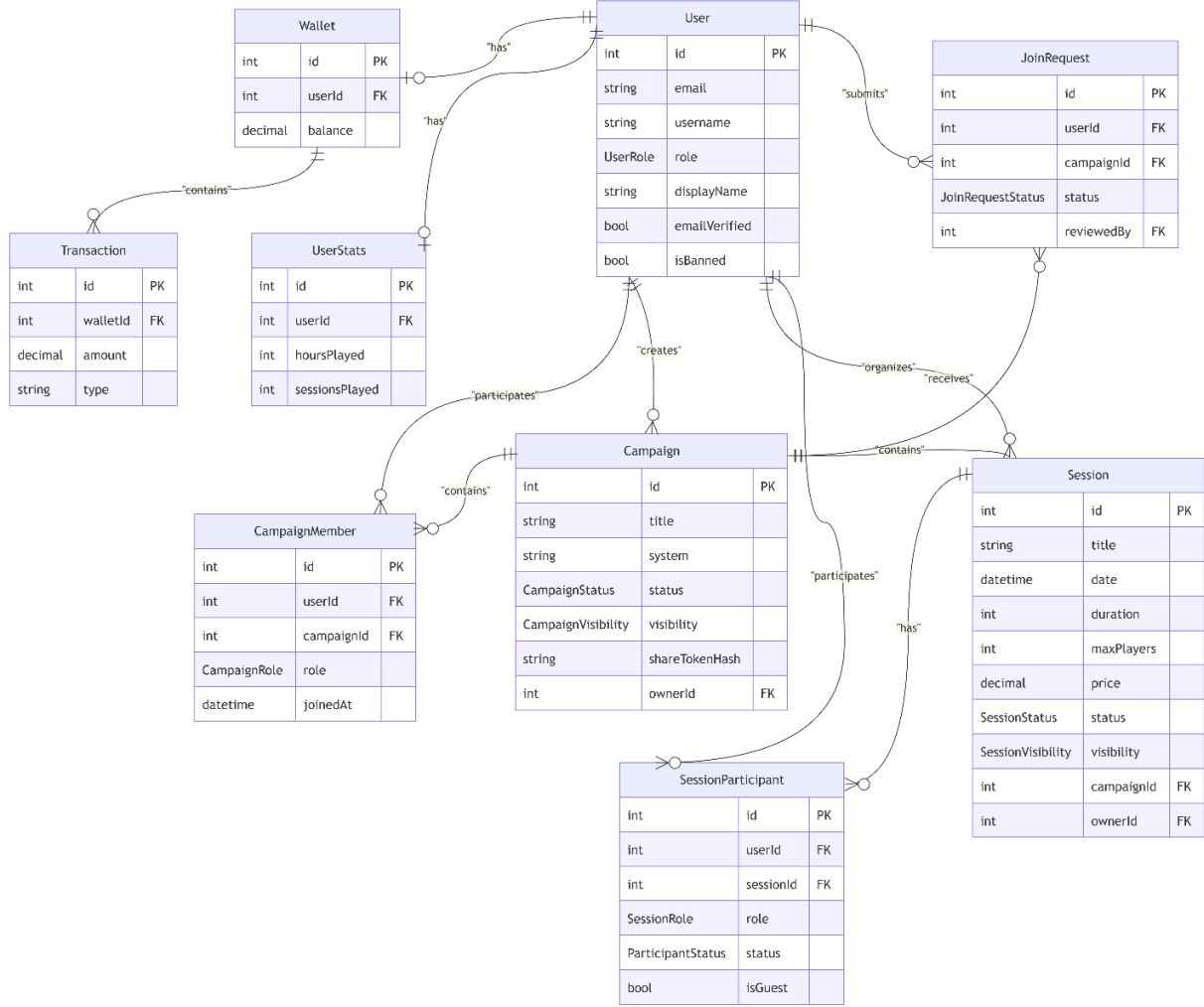


Рис. 2.4. Схема даних: організаційне ядро

На рис. 2.4 відображено організаційне ядро бази даних. Навколо центральної сутності User будуються кампанії та сесії: зв'язкові таблиці CampaignMember і SessionParticipant реалізують відношення «багато-до-багатьох» зі збереженням доменних ролей, а JoinRequest фіксує процес приєднання учасників. Допоміжний фінансовий контур (Wallet, Transaction) і статистику (UserStats) винесено в окремі таблиці, пов'язані з користувачем відношенням «один-до-одного». Комунікаційну частину системи через її обсяг виокремлено в окрему схему (рис. 2.5).

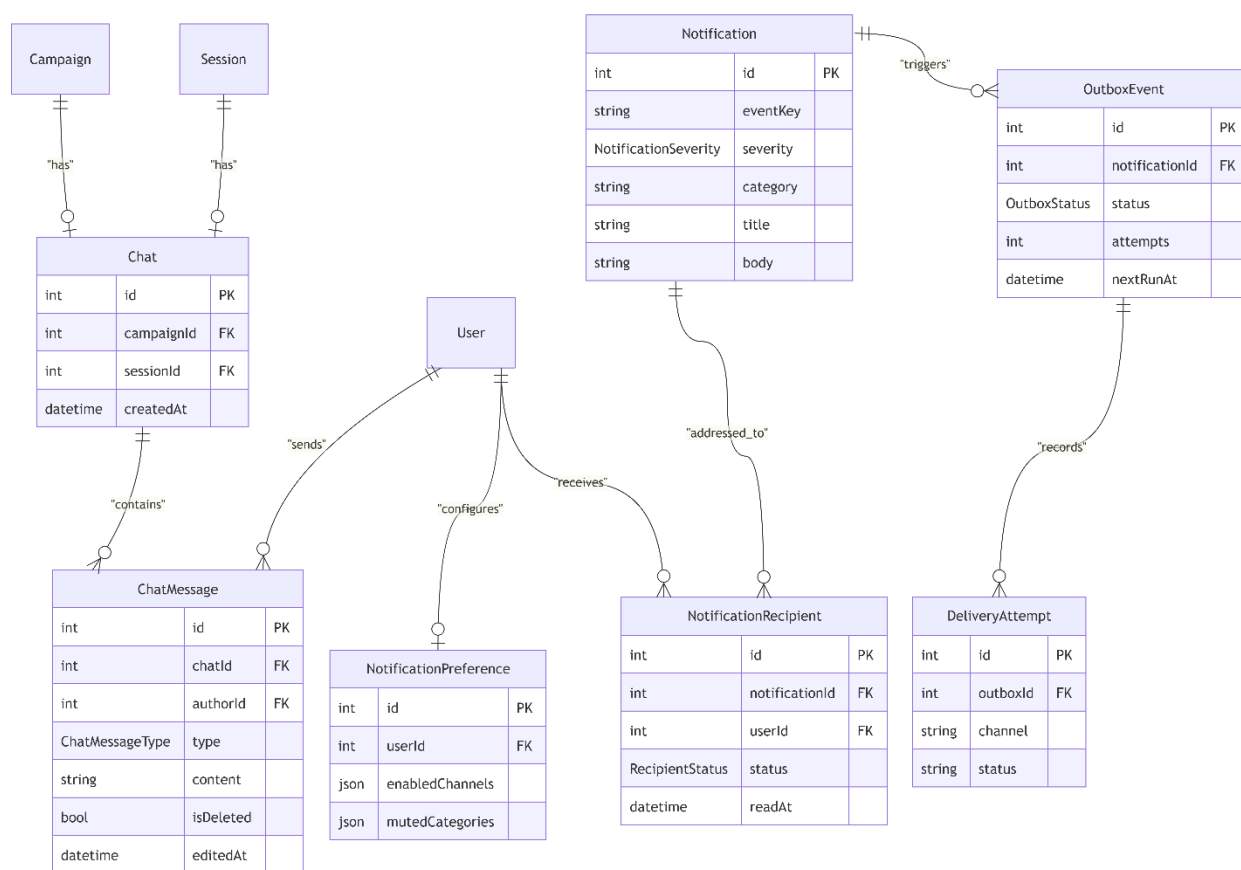


Рис. 2.5. Схема даних: комунікація та сповіщення

На рис. 2.5 наведено підсистему комунікації та сповіщень. Чати прив'язуються до кампанії або сесії, а доставка сповіщень побудована за ланцюгом Notification – NotificationRecipient. Гарантованість надсилання забезпечують таблиці OutboxEvent і DeliveryAttempt, що реалізують патерн transactional outbox з повторними спробами доставки різними каналами [31].

Спроектowana структура відповідає третій нормальній формі: зв'язкові таблиці CampaignMember і SessionParticipant усувають надмірність при відношеннях «багато-до-багатьох» і зберігають доменні ролі без дублювання. Для оптимізації вибірок передбачено індекси за зовнішніми ключами та полями фільтрації (дата і система сесій, статус, ключі чату та черги сповіщень). Цілісність даних гарантовано на рівні PostgreSQL: каскадне видалення (ON DELETE CASCADE) узгоджено вилучає залежні записи, тоді як зв'язок сесії з кампанією використовує ON DELETE SET NULL, що зберігає разові ігри при видаленні кампанії. Таким чином, спроектована модель є надійним фундаментом для серверної логіки та клієнтського інтерфейсу.

2.3 Проектування та реалізація алгоритмів роботи системи

Після детального опису архітектурних рішень та фізичної структури бази даних, наступним критично важливим етапом проектування є формалізація динамічних аспектів функціонування платформи. Якщо досі було визначено, які дані зберігаються в системі та як вони структуровані, то розробка алгоритмів дозволяє визначити логіку їх обробки, правила переходу між станами ключових об'єктів та послідовність взаємодії між клієнтською і серверною частинами застосунку. Це забезпечує передбачуваність і надійність виконання бізнес-процесів, що є критичним для систем із розгалуженою моделлю доступу.

Задля забезпечення повноти опису та візуалізації логіки роботи платформи у даному параграфі застосовано інструментарій мови UML. Моделювання динамічної поведінки системи реалізовано через комплексне поєднання різних типів діаграм, де стани описують повний життєвий цикл ігрових сесій, а діаграми активностей деталізують алгоритми прийняття рішень та перевірки умов доступу. Водночас часова взаємодія між клієнтською частиною у вигляді SPA, сервером, що поєднує REST API та WebSockets, і базою даних під час виконання ключових операцій візуалізується за допомогою діаграм послідовності. Завершує це цілісне уявлення діаграма компонентів, яка безпосередньо відображає архітектурну взаємодію всіх програмних модулів системи.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

Такий підхід дозволяє формалізувати складні процеси, такі як Join-flow та механізми управління ролями, що мінімізує ймовірність виникнення логічних помилок на етапі програмної реалізації.

Одним із ключових елементів системи є сутність ігрової сесії, яка має власний життєвий цикл. Зміна станів цієї сутності суворо регламентована та залежить як від дій користувачів (з відповідними правами доступу), так і від автоматизованих серверних процесів. Для наочної демонстрації цього життєвого циклу розроблено діаграму станів, наведену на рисунку 2.6.

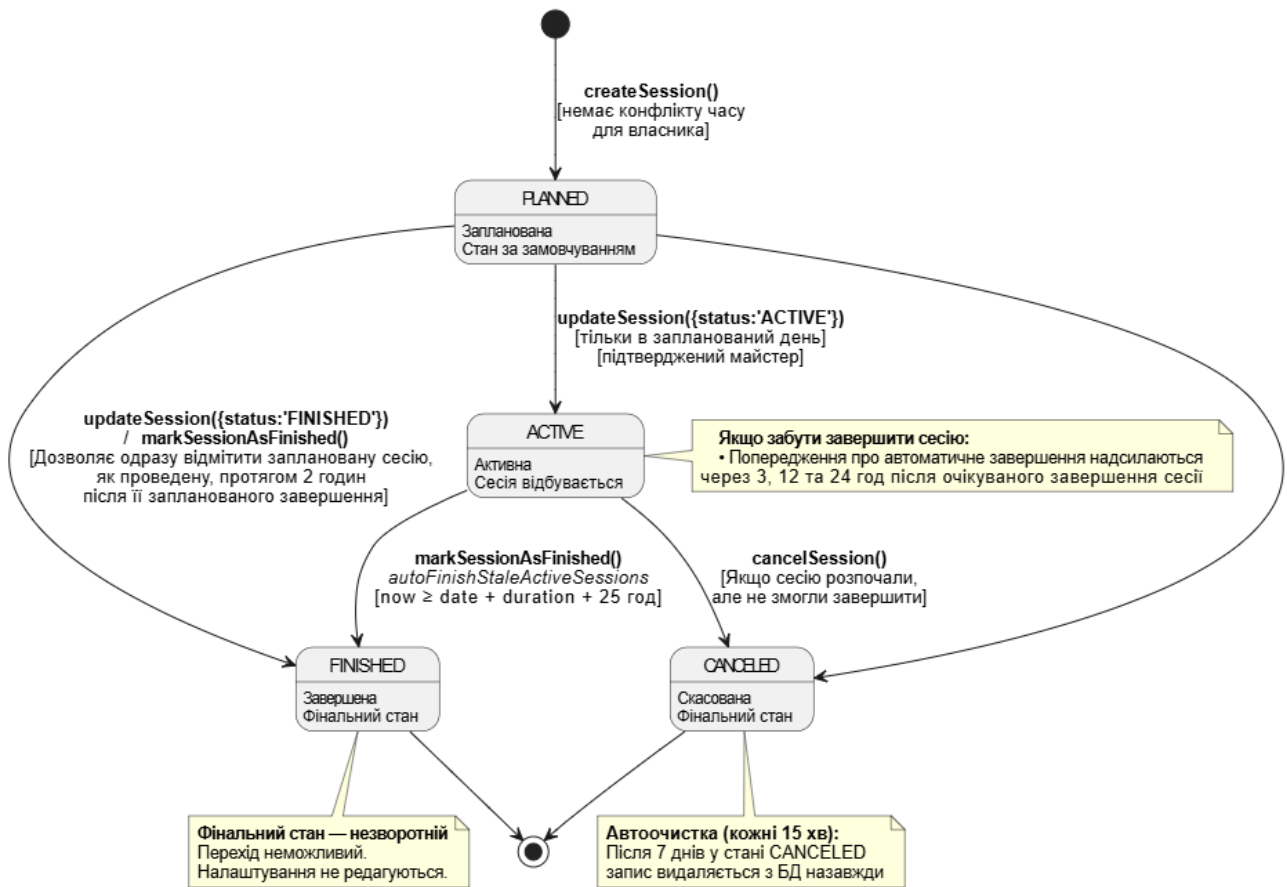


Рис. 2.6. Діаграма станів ігрової сесії

На рисунку 2.6 представлено логіку переходів між чотирма основними станами ігрової сесії.

Запланована: це початковий стан сесії, який встановлюється за замовчуванням після її успішного створення методом createSession(). Умова успішного створення – відсутність часових конфліктів у календарі творця сесії. У цьому стані дозволено повне редагування налаштувань, але виключно власником або майстром кампанії.

Активна: стан, що позначає безпосередній процес проведення гри. Перехід до нього можливий лише із запланованого стану в день проведення сесії за умови підтвердження з боку майстра гри шляхом виклику `updateSession({status: 'ACTIVE'})`. На цьому етапі зміна ключових налаштувань сесії блокується для забезпечення цілісності ігрового процесу.

Завершена: фінальний і незворотний стан сесії. Перехід до нього ініціюється Майстром гри через виклик `markSessionAsFinished()` або автоматично системою через заплановане завдання якщо поточний час перевищує час початку сесії, її тривалість та додатковий буфер у 25 годин. Будь-яке подальше редагування інформації у цьому стані неможливе.

Скасована: альтернативний фінальний стан, який застосовується у разі відміни гри. Перехід може бути ініційований вручну (Майстром або Власником через `cancelSession()`) або автоматично за певних умов: відсутність призначеного Майстра у день гри; якщо сесія залишається у стані `PLANNED` понад 3 дні після запланованої дати; або якщо призначений Майстер не розпочав підтверджену гру протягом 24 годин після планового часу.

Для запобігання накопиченню неактуальних даних у базі даних (БД) реалізовано механізм автоматичної очистки. Скасовані сесії безповоротно видаляються з БД через 7 днів після переходу у стан `CANCELED` системним процесом `autoDeleteCanceledSessions`.

Окрім переходів між станами, життєвий цикл сесії супроводжується фоновими процесами сповіщення, які виконуються за розкладом кожні 15 хвилин і не змінюють стан сесії, а лише інформують учасників. До них належать розсилання нагадувань за 24 години та за 1 годину до початку гри та ескалаційні попередження Майстрові про незавершену активну сесію через 3, 12 і 24 години після планового завершення. У поєднанні з уже описаними автоматичними переходами це забезпечує самоочищення системи та підтримання коректності станів без ручного втручання адміністратора.

Як логічне продовження опису життєвого циклу ігрової сесії, необхідно деталізувати алгоритм її ініціалізації. Для візуалізації послідовності дій та умов,

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

що супроводжують цей процес, розроблено діаграму активностей створення сесії (рис. 2.7). Процес ініціюється користувачем через клієнтський додаток шляхом заповнення форми з базовими параметрами майбутньої гри, такими як дата, час початку, запланована тривалість та налаштування видимості. Після відправки запиту на сервер система першочергово виконує перевірку прав доступу ініціатора. Оскільки ініціалізація сесії в межах існуючої кампанії є привілейованою операцією, алгоритм перевіряє наявність у користувача ролі власника або майстра гри у цій кампанії. У разі відсутності відповідних прав процес негайно переривається із поверненням помилки доступу.

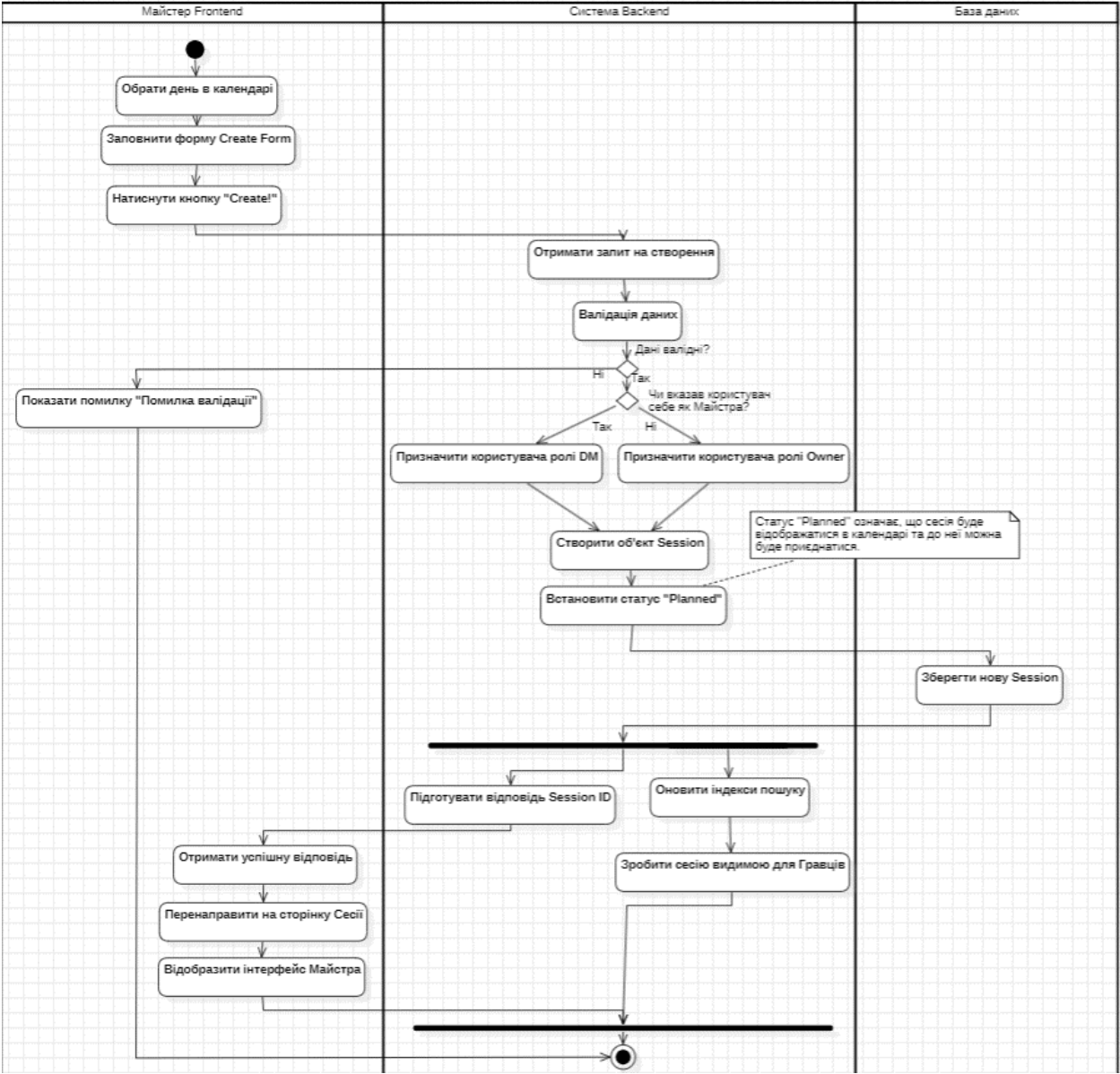


Рис. 2.7. Діаграма активностей процесу створення ігрової сесії

Після успішної авторизації сервер валідує дані та перевіряє календар на відсутність часових конфліктів (накладання нової гри на інші сесії майстра). У разі помилки клієнт отримує сповіщення для коригування даних.

Якщо перевірка успішна, сервер створює нову ігрову сесію зі статусом «Запланована», зберігає її в базі даних та відправляє підтвердження клієнту. Після цього інтерфейс користувача оновлюється, відображаючи гру в розкладі.

Для детального аналізу взаємодії між клієнтською частиною, серверним API та базою даних під час виконання складних бізнес-операцій було побудовано діаграму послідовності. У якості показового процесу обрано алгоритм приєднання користувача до ігрової сесії, оскільки він задіює всі рівні архітектури застосунку та включає комплексну перевірку прав доступу. Діаграму послідовності даного процесу наведено на рисунку 2.8.

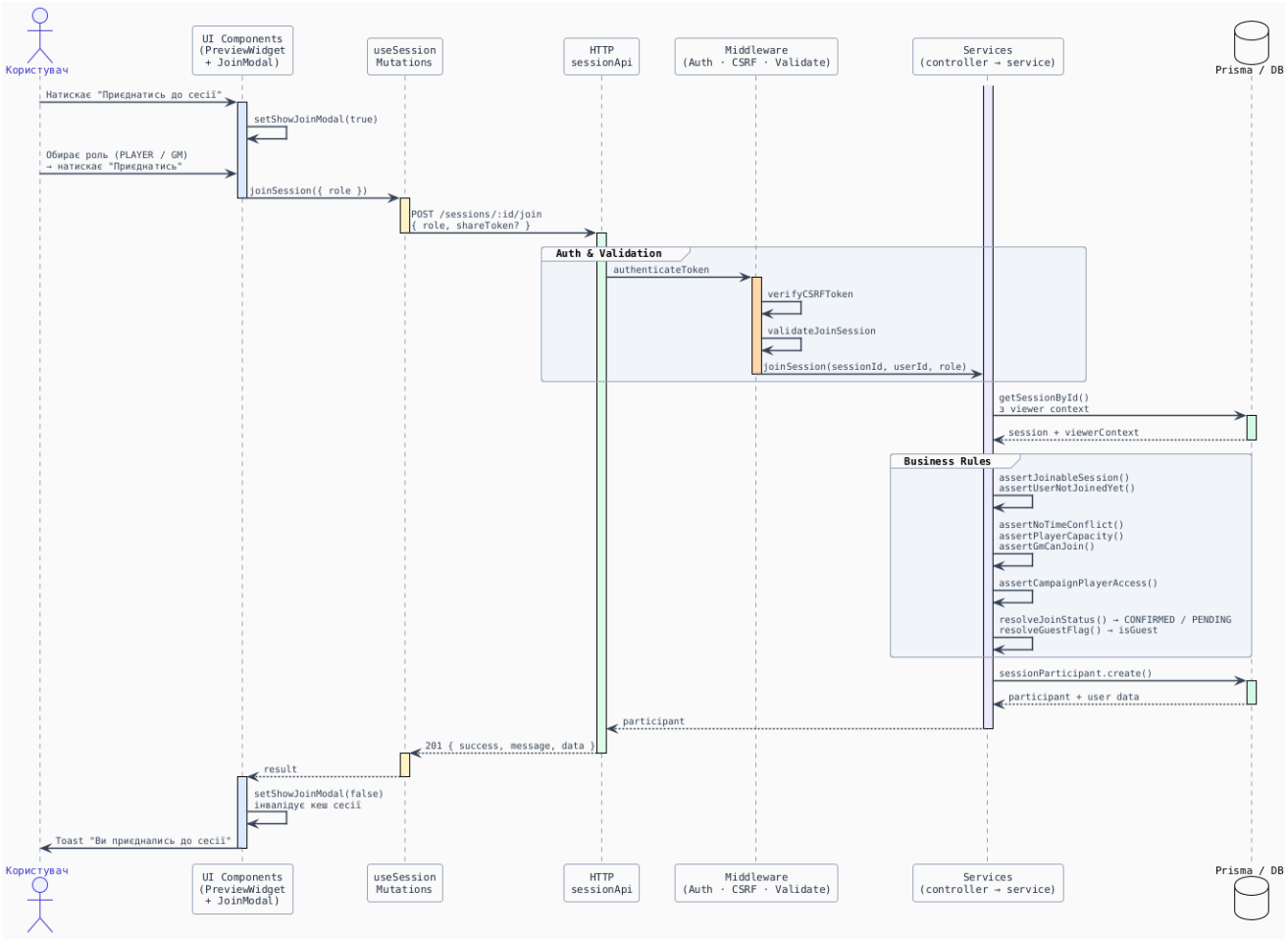


Рис. 2.8. Діаграма послідовності процесу приєднання до ігрової сесії

Процес приєднання ініціюється користувачем через клієнтський інтерфейс, де після натискання кнопки вступу та вибору ролі (гравець або майстер гри) запит передається до шару React-компонентів. За допомогою спеціалізованих хуків мутацій формується HTTP POST запит до серверного маршруту /sessions/:id/join. На серверній стороні обробка запиту проходить крізь ієрархію ізольованих архітектурних шарів, починаючи з проміжного програмного забезпечення. На цьому етапі система виконує критичні безпекові перевірки, що включають автентифікацію токена користувача, контроль CSRF-захисту та первинну валідацію вхідних параметрів, таких як обрана роль та наявність токена доступу.

Після успішного проходження фільтрів безпеки управління передається сервісному шару, де сервіс обробки сесій взаємодіє з базою даних через Prisma ORM для отримання повної інформації про ігрову зустріч із урахуванням контексту поточного користувача. Далі процес переходить до шару бізнес-правил, де реалізовано найважливіший етап валідації предметної області. Система в автоматичному режимі перевіряє можливість приєднання до сесії, відсутність дублювання записів про участь, контролює часові конфлікти в персональному календарі гравця та аналізує наявність вільних місць згідно з установленими лімітами наповненості. Крім того, перевіряються права доступу користувача до батьківської кампанії, на основі чого алгоритм визначає фінальний статус заявки та встановлює атрибути гостьового доступу.

У разі успішного підтвердження всіх бізнес-інваріантів сервіс ініціює транзакцію в базі даних для створення нового запису учасника. Завершальним етапом є формування зворотного потоку даних, за якого база даних повертає створену сутність сервісу, а той через контролер відправляє клієнту відповідь із HTTP-статусом 201 Created. Отримання успішного підтвердження на стороні фронтенду призводить до автоматичного закриття модального вікна, інвалідації локального кешу для миттєвого оновлення інтерфейсу та виведення візуального сповіщення користувачеві про успішне завершення операції. Такий комплексний

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

підхід гарантує консистентність даних на всіх рівнях системи та забезпечує надійність виконання бізнес-процесів.

Описані вище алгоритми охоплювали організаційний контур системи. Окрему групу становлять процеси підсистеми комунікації реального часу. Першим із них є обмін текстовими повідомленнями у межах чатів кампаній та сесій, діаграму послідовності якого наведено на рисунку 2.9.

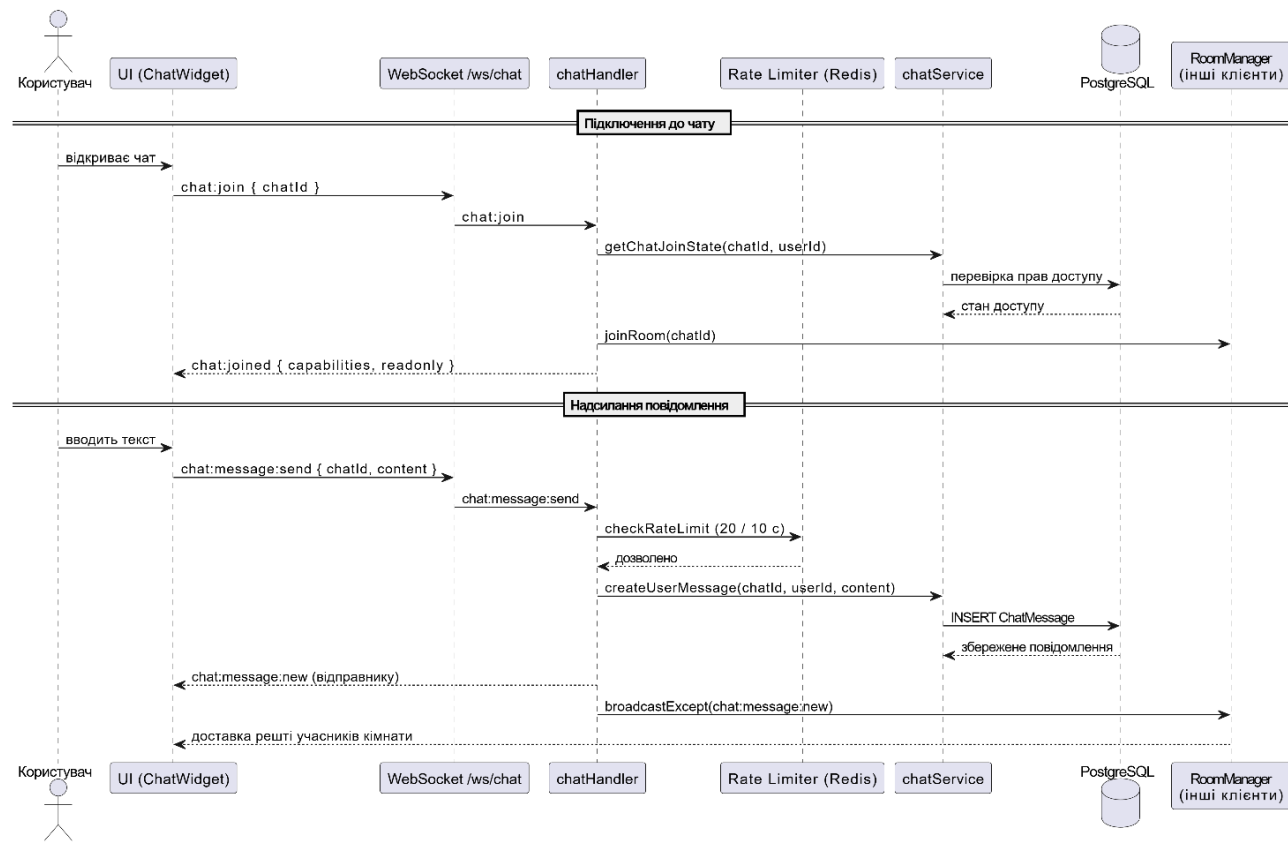


Рис. 2.9. Діаграма послідовності обміну повідомленнями в чаті

Обмін повідомленнями реалізовано поверх постійного WebSocket-з'єднання (маршрут /ws/chat). Після приєднання до кімнати чату сервер повертає набір можливостей користувача та ознаку «лише для читання». Кожне вихідне повідомлення проходить через обмеження частоти (20 повідомлень за 10 секунд), зберігається в базі даних, після чого негайно повертається відправникові (для оптимістичного оновлення інтерфейсу за clientMessageId) та розсилається решті учасників кімнати механізмом широкомовлення.

Поряд із текстовим спілкуванням платформа підтримує синхронний аудіо-та відеозв'язок під час ігрових сесій. На відміну від чату, цей процес передбачає

узгодження медіапараметрів між учасниками та медіасервером. Діаграму послідовності встановлення відеозв'язку наведено на рисунку 2.10.

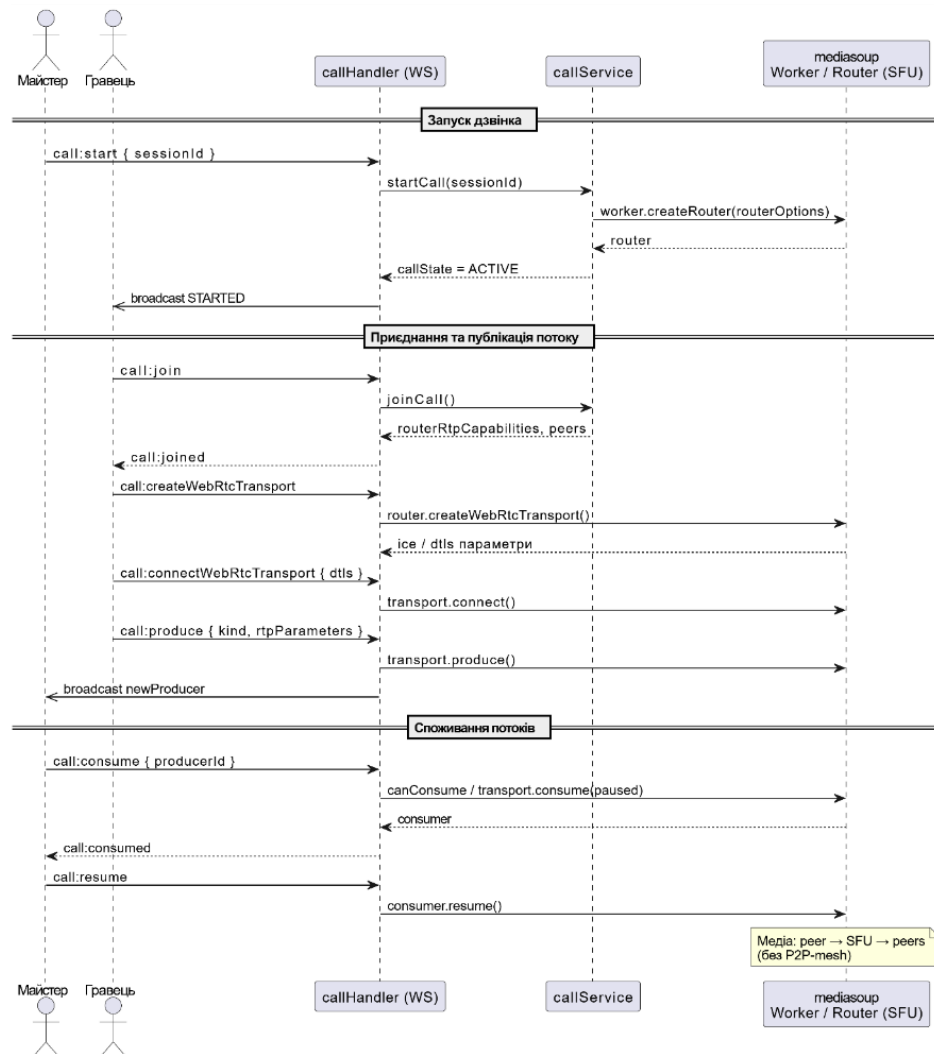


Рис. 2.10. Діаграма послідовності встановлення відеозв'язку

Сигналінг відеозв'язку відбувається через той самий WebSocket-канал за RPC-моделлю. Майстер ініціює дзвінок, що призводить до створення на медіасервері mediasoup окремого маршрутизатора (router) для сесії. Кожен учасник отримує RTP-можливості маршрутизатора, створює та під'єднує WebRTC-транспорти, після чого публікує власні аудіо- та відеопотоки (produce). Інші учасники сповіщаються про нового мовника потоку й споживають його (consume) через SFU-сервер, що пересилає медіа без використання схеми «кожен-з-кожним» і знижує навантаження на клієнтів.

Спільною для всіх описаних модулів є потреба своєчасно інформувати користувачів про зміни стану системи. Цю функцію виконує підсистема

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

сповіщень, побудована за патерном transactional outbox для гарантованої доставки. Відповідний процес наведено на рисунку 2.11.

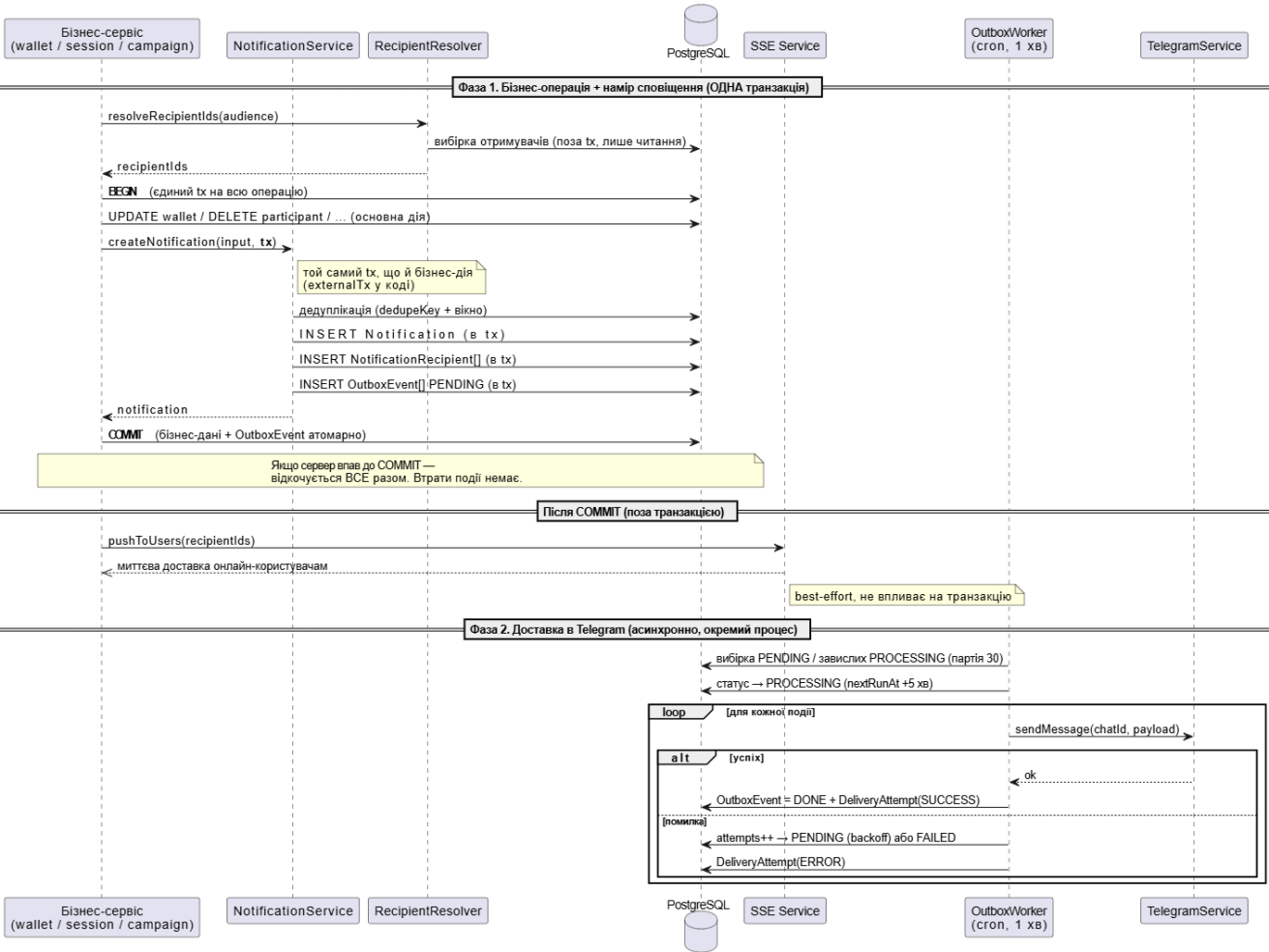


Рис. 2.11. Діаграма послідовності доставки сповіщень

Доставка сповіщень побудована за патерном transactional outbox, що гарантує надсилання навіть за тимчасової недоступності зовнішніх сервісів. На першій фазі в межах однієї транзакції визначається коло отримувачів, виконується дедуплікація за ключем події, створюється запис сповіщення з адресатами, а для користувачів з увімкненим Telegram-каналом записи черги доставки зі статусом PENDING. Онлайн-користувачам сповіщення доставляється негайно каналом Server-Sent Events. На другій фазі окремий фоновий обробник (cron, щохвилини) партіями опрацьовує чергу: успішні надсилання позначаються як DONE, а помилкові повторюються з експоненційною затримкою до вичерпання ліміту спроб, при цьому кожна спроба фіксується в таблиці DeliveryAttempt для аудиту. Узагальнено взаємодію

всіх розглянутих модулів, від організаційного ядра до підсистем комунікації та сповіщень, представлено на діаграмі компонентів (рис. 2.12).

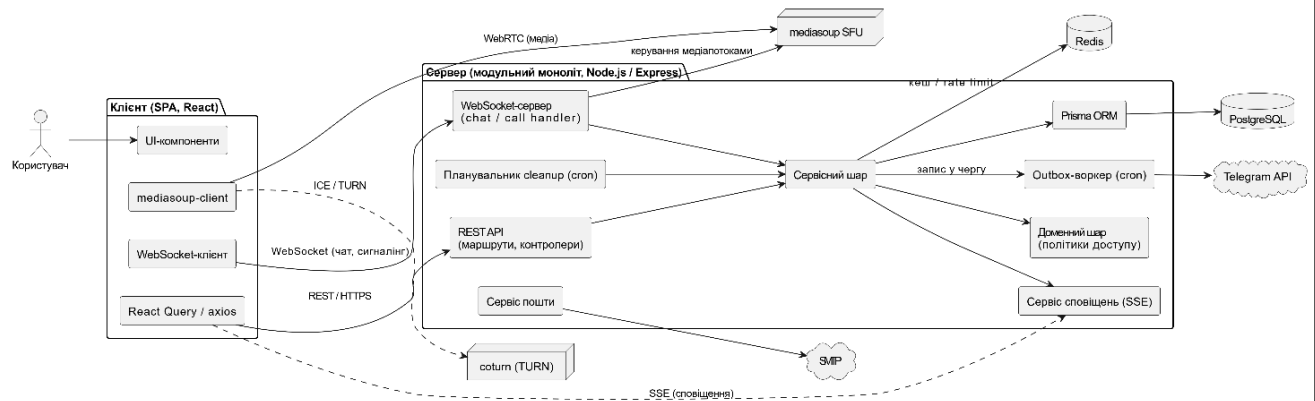


Рис. 2.12. Діаграма взаємодії програмних модулів

Діаграма відображає взаємодію основних модулів системи в межах обраної багатошарової архітектури. Клієнтський SPA-застосунок взаємодіє із сервером трьома незалежними каналами: синхронними REST-запитами, однонаправленим каналом SSE для сповіщень та двостороннім Websocket-з'єднанням для чату й сигналінгу дзвінків; медіапотоки передаються напряду через SFU-сервер mediasoup з обходом NAT через coturn. На сервері запити проходять крізь шари API, сервісів, доменної логіки та Prisma ORM до PostgreSQL, а допоміжні модулі (Redis, підсистема сповіщень з outbox-воркером, поштовий сервіс і планувальник фонових завдань) забезпечують кешування, надійну доставку та автоматизацію.

У підсумку, застосування інструментарію UML дозволило вичерпно формалізувати як статичну структуру, так і динамічну поведінку системи. Деталізовано життєвий цикл ігрової сесії з урахуванням ручних та автоматичних переходів, алгоритми створення сесії та приєднання учасників із багаторівневою перевіркою прав доступу, а також процеси підсистеми комунікації реального часу, такі як обмін повідомленнями у чатах, встановлення групового відеозв'язку за схемою SFU та гарантовану доставку сповіщень за патерном transactional outbox. Узагальнена діаграма компонентів відображає взаємодію цих модулів у межах багатошарової архітектури. Сформований опис створює надійне підґрунтя для етапу програмної реалізації, мінімізує ймовірність архітектурних

помилки та забезпечує передбачуваність роботи системи з розгалуженою моделлю доступу.

2.4 Реалізація функціоналу модулів організації та комунікації

Реалізацію основних принципів архітектури та прикладної логіки доцільно розглянути на прикладі приєднання користувача до ігрової сесії. Саме цей сценарій задіює всі рівні системи та містить найскладнішу доменну логіку перевірки прав доступу, поєднуючи ключові сутності платформи: користувача, сесію, її учасників і батьківську кампанію.

Реалізацію побудовано за принципами багат шарової архітектури в межах модульного моноліту, що забезпечує чітке розділення відповідальностей. Подальший опис побудовано за порядком проходження запиту крізь систему: від прийому HTTP-запиту, крізь шари валідації, контролера, прикладних сервісів та доменних правил, до атомарного збереження в базі даних і публікації сповіщень. Цьому сценарію відповідає діаграма послідовності, наведена раніше на рисунку 2.8.

Транспортний рівень, тобто взаємодію з клієнтом, реалізовано засобами фреймворку Express. Кожен маршрут оголошується декларативно як ланцюг проміжного програмного забезпечення, що виконується до основного обробника. Маршрут не містить бізнес-логіки, його роль полягає лише у формуванні конвеєра обробки та делегуванні запиту контролеру.

```
router.post(
 ('/:id/join',
  [authenticateToken, verifyCSRFToken, ...validateJoinSession],
  (req, res, next) => sessionParticipantsController.joinSession(req, res,
  next)
);
```

Конвеєр послідовно виконує автентифікацію за JWT-токеном (authenticateToken), перевірку захисту від міжсайтових запитів (verifyCSRFToken) та валідацію вхідних даних. Лише після успішного проходження всіх фільтрів керування передається контролеру.

Перевірку коректності параметрів (Валідація вхідних даних) реалізовано декларативно за допомогою бібліотеки Joi. Кожне правило описує очікувану

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

структуру запиту, а спеціальне middleware застосовує схему ще до виклику контролера.

```
const joinSessionBodySchema = Joi.object({
  role: Joi.string().trim().valid('PLAYER', 'GM').optional(),
  shareToken: Joi.string().trim().min(10).max(255).optional(),
});
const validateJoinSession = [
  validateParams(sessionIdParamsSchema),
  validateBody(joinSessionBodySchema),
];
```

Винесення валідації в окремий шар запобігає обробці некоректних даних ще до звернення до доменної логіки та бази даних. У разі порушення правила клієнт одразу отримує структурований опис помилки, що дозволяє інтерфейсу підсвітити некоректне поле.

На наступному рівні контролер виконує трансляцію HTTP-запиту у виклик прикладного сервісу та формування відповіді. Він не містить бізнес-логіки: вилучає параметри із запиту, делегує виконання сервісу та повертає результат у стандартизованому форматі з кодом 201 Created.

```
async joinSession(req, res, next) {
  try {
    const { id: sessionId } = req.params;
    const userId = req.user.id;
    const { role = 'PLAYER', shareToken = null } = req.body || {};
    const participant = await sessionService.joinSession(
      sessionId, userId, { role, shareToken });
    res.status(201).json({
      success: true,
      message: participant.status === 'CONFIRMED'
        ? 'Ви приєдналися до сесії!' : 'Заявку на участь відправлено!',
      data: participant,
    });
  } catch (error) { next(error); }
}
```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

Помилки не обробляються безпосередньо, а передаються через `next(error)` до централізованого обробника помилок, який формує відповідь з відповідним HTTP-кодом.

Основна бізнес-логіка прихована на рівні сервісів. Сервіс виступає оркестратором сценарію: завантажує контекст сесії, послідовно застосовує доменні перевірки, виконує атомарне збереження та ініціює сповіщення.

```
async joinSession(sessionId, userId, options = {}) {
  const { role = 'PLAYER', shareToken = null } = options;
  const session = await resolveGetSessionById(sessionId, userId, {
    shareToken });
  const normalizedRole=normalizeJoinRole(role, AppError, ERROR_CODES);
  await validateSessionJoinConditions({ userId, session, normalizedRole
  });
  const participant = await prisma.$transaction(async (tx) => {
    const newParticipant = await tx.sessionParticipant.create({
      data: {
        userId, sessionId: parseId(sessionId), role: normalizedRole,
        status: resolveJoinStatus({ normalizedRole, session }),
        isGuest: resolveGuestFlag({ normalizedRole, session })
      },
      include: { user: { select: { id: true, username: true ... } } }
    });
    if (normalizedRole === 'PLAYER' && session.price > 0) {
      await walletService.processSessionPayment(userId, session, tx);
    }
    await notifyManagersStatus(session, newParticipant, tx);
    return newParticipant;
  });
  return participant;}
```

Доменний рівень виступає коренем всього цього ланцюга. Усі бізнес-інваріанти винесено в окремі чисті функції доменного шару, що робить їх єдиним джерелом істини та полегшує тестування. Так, `assertJoinableSession` перевіряє допустимість приєднання за станом сесії, а `resolveJoinStatus` визначає, чи буде учасник підтверджений автоматично, чи потрапить у чергу заявок.

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

```

function assertJoinableSession(session, AppError, ERROR_CODES) {
  if (!JOINABLE_SESSION_STATUSES.has(session.status))    // лише стан
  PLANNED
    throw new AppError(ERROR_CODES.SESSION_JOIN_STATUS_FORBIDDEN);
  if (session.campaign?.status === 'FINISHED')
    throw new AppError(ERROR_CODES.SESSION_JOIN_CAMPAIGN_FINISHED);
  if (new Date(session.date) < new Date())                // гра вже
  розпочалась
    throw new AppError(ERROR_CODES.SESSION_JOIN_ALREADY_STARTED);
}
function resolveJoinStatus({ normalizedRole, session }) {
  if (normalizedRole !== 'PLAYER') return 'PENDING';
  if (!session.campaignId && session.visibility === 'PUBLIC') return
  'CONFIRMED';
  return 'PENDING';
}

```

Інфраструктурний рівень. Збереження запису учасника виконується в межах транзакції Prisma, що гарантує атомарність зміни (виклик `prisma.$transaction`). Для операцій зі зміною стану сесії (підтвердження, скасування) додатково застосовано песимістичне блокування рядка `SELECT ... FOR UPDATE`, що унеможливорює стани гонитви за умов одночасних запитів, наприклад, перевищення ліміту учасників або подвійне списання коштів.

Публікація сповіщень. Після успішного збереження сервіс ініціює сповіщення відповідальних осіб. Доставку реалізовано за патерном `transactional outbox`: у межах однієї транзакції створюється запис сповіщення, його адресати та, за потреби, події черги доставки в зовнішні канали; онлайн-користувачам сповіщення надсилається негайно через `Server-Sent Events`. На відміну від систем із брокером повідомлень, у модульному моноліті події обробляються в межах процесу, що спрощує супровід без втрати надійності доставки.

```

async createNotifcation(input, externalTx = null) {
  const recipientIds = await this.resolveRecipientIds(input);
  if (recipientIds.length === 0) return null;
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

```

const execute = async (tx) => {
  const notification = await tx.notification.create({ data: { ...input
} }));
  await this._attachRecipients(tx, notification.id, recipientIds);
  await      this._createTelegramOutboxEvents(tx,      notification,
recipientIds); // чегра
  return { notification, recipientIds };
};
const result = externalTx ? await execute(externalTx)
                          : await this.prisma.$transaction(execute);
this.pushToConnectedUsers(result.notification,      recipientIds);    //
миттєво через SSE
return result.notification;
}

```

Таким чином, проходження єдиного запиту демонструє, як багат шарова архітектура забезпечує чітке розділення відповідальностей: транспорт і валідація відсіюють некоректні запити, контролер лишається тонким, прикладний сервіс оркеструє сценарій, доменний шар зосереджує бізнес-правила, а інфраструктурний рівень гарантує атомарність і надійну доставку сповіщень. Повні лістинги розглянутих класів наведено в Додатку А.

Завершальною ланкою розглянутого сценарію є клієнтська частина. Запити до серверного API інкапсульовано у спеціалізованих хуках мутацій на основі бібліотеки TanStack React Query, що звільняє компоненти інтерфейсу від прямої роботи з мережевим рівнем. Після отримання успішної відповіді (код 201) хук автоматично інвалідує відповідні фрагменти локального кешу, унаслідок чого пов'язані сутності, такі як картка сесії, список учасників та календар, оновлюються без повного перезавантаження сторінки. Паралельно інтерфейс закриває модальне вікно та виводить користувачеві візуальне сповіщення про результат операції. Такий підхід забезпечує миттєвий відгук інтерфейсу та узгодженість відображуваних даних із фактичним станом системи на сервері.

Розглянутий сценарій приєднання до сесії є показовим, оскільки за тією самою багат шаровою схемою: маршрут із конвеєром middleware, декларативна

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

валідація, тонкий контролер, прикладний сервіс-оркестратор, чисті доменні правила та атомарний інфраструктурний рівень, реалізовано й інші модулі платформи: управління кампаніями, текстовий чат, відеозв'язок і підсистему сповіщень. Уніфікована обробка помилок через єдиний тип `AppError` та централізований обробник, а також послідовне застосування транзакцій гарантують однорідність кодової бази та передбачуваність поведінки системи. Це суттєво спрощує супровід, тестування та подальше розширення функціоналу без порушення цілісності наявних компонентів.

Висновки до другого розділу

У другому розділі виконано комплексне проєктування організаційно-комунікаційної підсистеми платформи – від формалізації вимог до програмної реалізації. На основі визначених функціональних і нефункціональних вимог виокремлено ролі користувачів, побудовано діаграми варіантів використання, а також розроблено об'єктно-орієнтовану модель предметної області.

Спроектовано та нормалізовано фізичну структуру реляційної бази даних під керуванням PostgreSQL із застосуванням Prisma ORM, де структуру схеми деталізовано окремо для організаційного ядра й комунікаційної підсистеми. Цілісність даних забезпечено індексами, зовнішніми ключами та каскадним видаленням.

Засобами мови UML формалізовано динамічну поведінку системи: побудовано діаграму станів життєвого циклу сесії, діаграми активностей і послідовності для ключових процесів, а також архітектурну діаграму взаємодії програмних модулів.

Втілення спроектованих рішень у коді продемонстровано на прикладі наскрізного сценарію приєднання до сесії, що проходить крізь усі шари архітектури. Це дозволило підтвердити практичне застосування таких інженерних підходів, як розподіл відповідальностей між шарами, інкапсуляція бізнес-правил у домені, конкурентна цілісність за рахунок транзакцій та гарантована доставка сповіщень за патерном `transactional outbox`.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ

3.1 Порядок встановлення та налаштування системи

Розроблену вебплатформу спроектовано для розгортання у вигляді контейнеризованого застосунку, що гарантує ідентичність середовища виконання на машині розробника та на робочому сервері. Усі компоненти системи запускаються й керуються засобами Docker і Docker Compose, а їхні образи попередньо збираються та публікуються у реєстрі контейнерів, тож на цільовий сервер вони лише завантажуються без потреби складати застосунок безпосередньо на ньому.

Програмний комплекс складається з чотирьох незалежних контейнерів. Контейнер бази даних виконує екземпляр PostgreSQL, контейнер кешу виконує Redis, серверний контейнер містить застосунок Node.js разом із вбудованим медіасервером mediasoup, а клієнтський контейнер на основі Nginx обслуговує статичні файли односторінкового застосунку та забезпечує захищене з'єднання за протоколом HTTPS. Серверний образ збирається багатоетапно: спершу встановлюються залежності та генерується клієнт Prisma, після чого з підсумкового образу вилучаються інструменти розробки, а сам застосунок запускається від непривілейованого користувача, що зменшує розмір образу та підвищує безпеку. Загальну конфігурацію розгортання, її вузли та канали взаємодії наведено на рисунку 3.1.

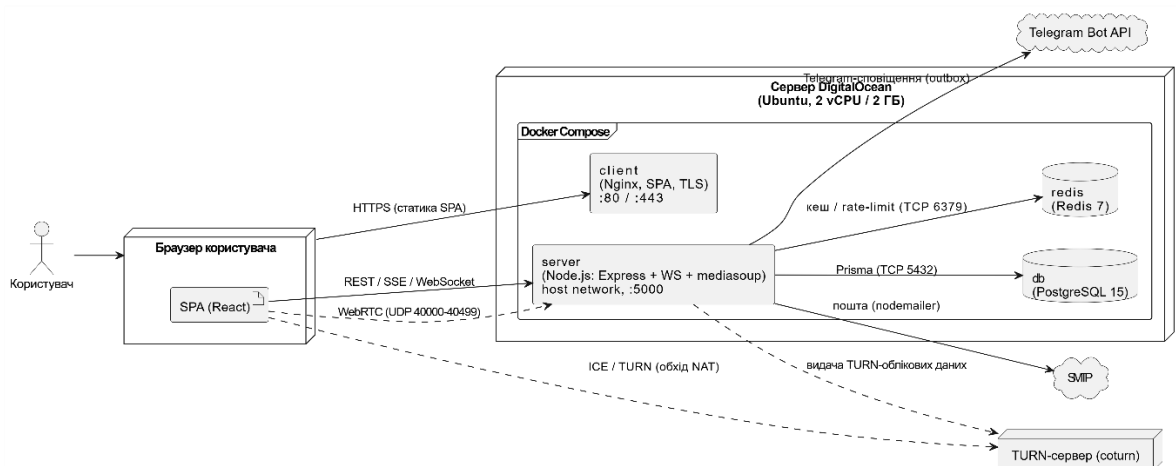


Рис. 3.1. Діаграма розгортання

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

Серверний контейнер працює в режимі мережі хоста, що необхідно для прямого обміну медіапотокami за протоколом WebRTC у визначеному діапазоні портів. Зовнішню взаємодію забезпечують окремий TURN-сервер coturn для обходу трансляції мережевих адрес, поштовий сервіс для надсилання листів та Telegram Bot API для доставки сповіщень. Розподіл обчислювальних ресурсів між контейнерами задано явними обмеженнями, наведеними у таблиці 3.1.

Таблиця 3.1.

Склад контейнерів та ресурсні обмеження

Сервіс	Образ	CPU	RAM	Порти
db	postgres:15	0,75	512 МБ	5432 (внутрішній)
redis	redis:7-alpine	0,25	128 МБ	6379 (внутрішній)
server	Node.js + mediasoup	1,0	768 МБ	5000, 40000–40499/UDP
client	Nginx	0,25	64 МБ	80, 443

Конфігурування системи виконується через змінні середовища, що відповідає принципу відокремлення налаштувань від програмного коду [32]. Чутливі дані, як-от ключі підписування токенів, секрет шифрування посилань-запрошень, паролі бази даних та облікові дані поштового і Telegram-сервісів, не зберігаються у репозиторії, а передаються лише на етапі розгортання. Обов'язкові параметри позначено в описі сервісів як критичні, тому за їхньої відсутності запуск переривається з відповідним повідомленням, що унеможлиблює старт системи з неповною конфігурацією. Окремі групи змінних відповідають за підключення до бази даних і кешу, параметри безпеки, надсилання електронної пошти, роботу медіасервера та TURN, а також за інтеграцію з Telegram і розклад фонових завдань.

Послідовність встановлення розпочинається з підготовки сервера під керуванням операційної системи Linux зі встановленими Docker та Docker Compose. Після заповнення файлу конфігурації та розміщення TLS-сертифікатів, що монтуються до клієнтського контейнера, розгортання ініціюється єдиною командою запуску, яка завантажує потрібні образи й піднімає всі сервіси.

Черговість запуску контролюється автоматично: серверний контейнер стартує лише після того, як база даних і кеш повідомлять про свою готовність через відповідні перевірки стану, а клієнтський тільки після готовності сервера. Внутрішню послідовність ініціалізації серверного застосунку наведено на рисунку 3.2.

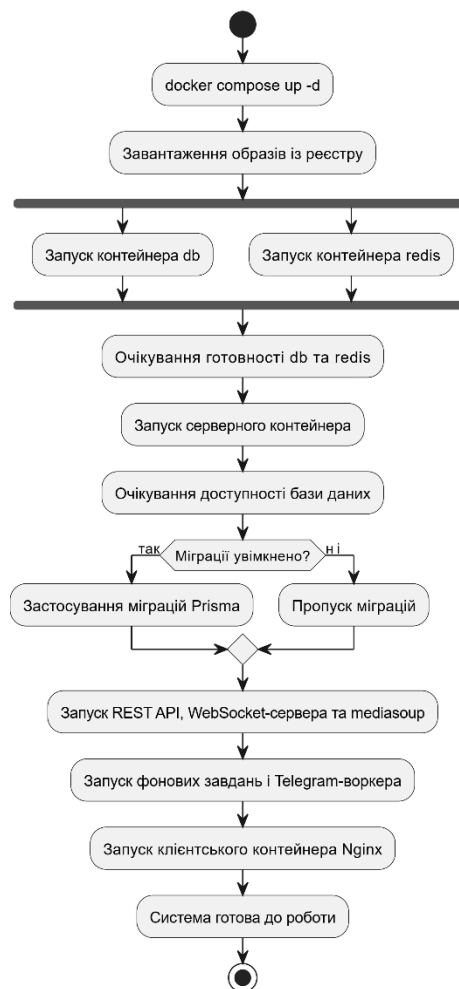


Рис. 3.2. Діаграма активностей процесу запуску

Під час старту серверний застосунок спершу очікує доступності бази даних, періодично надсилаючи перевірочний запит, після чого автоматично застосовує міграції схеми засобами Prisma, якщо їх не вимкнено окремим параметром. Лише після успішного оновлення структури бази даних ініціюються основні підсистеми: транспортний шар REST API, сервер веб-сокетів, медіасервер mediasoup, а також фонові заплановані завдання з очищення прострочених токенів і неактуальних сесій та обробник черги Telegram-

сповіщень. Такий підхід забезпечує самодостатній запуск системи без ручного втручання адміністратора та коректний порядок ініціалізації залежних компонентів.

Розроблену систему розгорнуто на хмарному сервері DigitalOcean, а процес її оновлення повністю автоматизовано засобами безперервної доставки. Конвеєр розгортання запускається автоматично після успішного проходження перевірок основного конвеєра інтеграції та складається з двох послідовних етапів складання й публікації образів і власне розгортання на сервері, що відображено на рисунку 3.3.

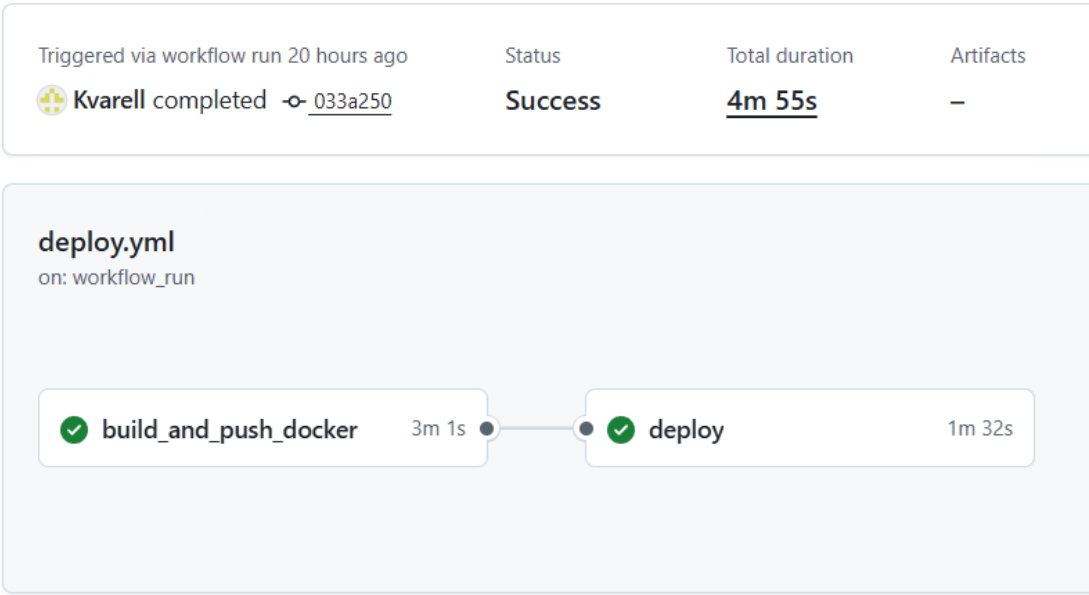


Рис. 3.3. Конвеєр автоматичного розгортання

На першому етапі система складає образи серверної та клієнтської частин і публікує їх у реєстрі контейнерів, позначаючи кожен образ унікальним ідентифікатором відповідної версії програмного коду. Застосування кешування проміжних шарів збірки суттєво пришвидшує повторні складання, тому цей етап займає лише кілька хвилин. Така ідентифікація образів за версіями забезпечує повну відтворюваність розгортання та можливість точно відновити будь-який попередній стан системи.

На другому етапі система підключається до сервера захищеним каналом, завантажує опубліковані образи та перезапускає контейнери, очікуючи підтвердження їхньої готовності через визначені перевірки стану. Якщо хоча б

один сервіс не запустився коректно, спрацьовує механізм автоматичного повернення до попередньої робочої версії, що мінімізує час простою та унеможлиблює розгортання непрацездатної конфігурації. Завдяки такому підходу оновлення платформи відбувається без ручного втручання адміністратора й з гарантованою надійністю процесу експлуатації.

Розмежування прав доступу побудовано на дворівневій рольовій моделі. Глобальний рівень визначає базові системні ролі звичайного користувача та адміністратора, тоді як доменний рівень задає ролі власника, майстра гри та гравця в межах конкретних кампаній і сесій. Контроль доступу забезпечується сукупністю механізмів, автентифікацією на основі токенів JWT, захистом від міжсайтових запитів, обмеженням частоти звернень засобами Redis, перевіркою дозволених джерел запитів та доменними політиками, що централізовано визначають дозволені дії для кожної ролі. Доступ до приватних ігор за посиланням додатково захищено криптографічними токенами, описаними у попередньому розділі.

Таким чином, обрана модель розгортання забезпечує відтворюване та кероване встановлення системи з мінімальним ручним налаштуванням, а описані механізми конфігурування й розмежування доступу створюють підґрунтя для безпечної експлуатації платформи, інтерфейс якої розглянуто в наступному параграфі.

3.2 Структура інтерфейсу платформи

Клієнтську частину платформи реалізовано у вигляді односторінкового застосунку з клієнтською маршрутизацією, що забезпечує плавні переходи між екранами без повного перезавантаження сторінки. Інтерфейс побудовано навколо єдиного макета, у якому верхня навігаційна панель надає доступ до основних розділів, центру сповіщень та меню профілю, а основна робоча область відображає вміст обраного розділу. Усі екрани адаптовано під різні розміри пристроїв. Далі розглянуто структуру та основні елементи інтерфейсу за функціональними групами.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

Взаємодія користувача з платформою розпочинається з процедур автентифікації. Реєстрація нового облікового запису виконується на сторінці, наведеній на рисунку 3.4.

The image shows a registration form titled "Реєстрація" (Registration) with the subtitle "Створіть новий акаунт" (Create a new account). It contains three input fields: "Нікнейм" (Nickname), "Email", and "Пароль" (Password). Below the fields is an orange button labeled "Зареєструватись" (Register). At the bottom, there is a link "Вже є акаунт? Увійти" (Already have an account? Log in).

Рис. 3.4. Сторінка реєстрації користувача

Авторизація зареєстрованого користувача здійснюється через форму входу, зображену на рисунку 3.5.

The image shows a login form titled "Вхід" (Login) with the subtitle "Раді бачити вас знову!" (Happy to see you again!). It contains two input fields: "Email" and "Пароль" (Password). Below the fields is an orange button labeled "Увійти" (Log in). At the bottom, there are two links: "Забули пароль?" (Forgot password?) and "Ще немає акаунту? Зареєструватись" (Don't have an account? Register).

Рис. 3.5. Сторінка входу в систему

Оскільки система вимагає обов'язкового підтвердження електронної пошти, після реєстрації користувачеві відображається відповідний екран із подальшими інструкціями. При цьому користувачу відправляється на пошту лист, для підтвердження реєстрації (рис. 3.6).

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

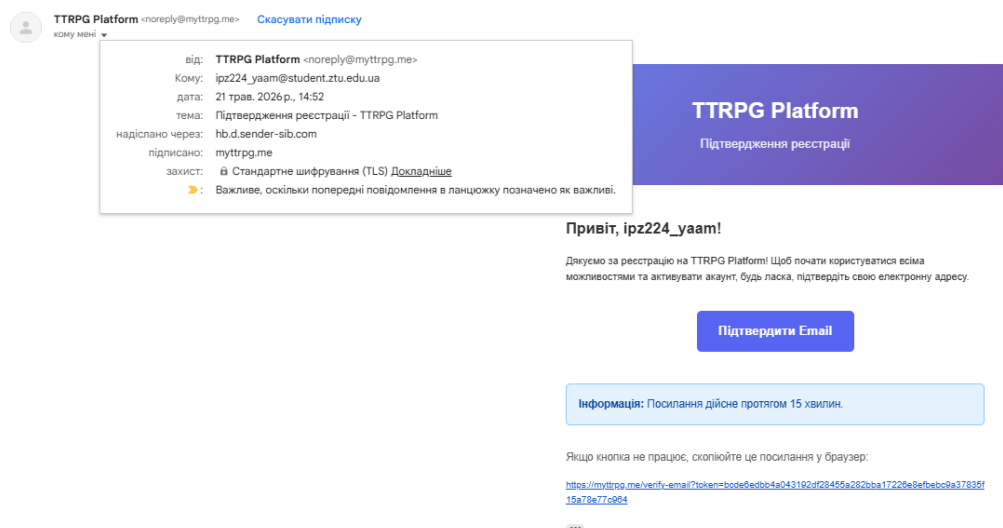


Рис. 3.6. Електронний лист з підтвердженням пошти

На випадок втрати доступу передбачено механізм відновлення пароля, форму якого наведено на рисунку 3.7. Після заповнення користувачу аналогічним до попереднього чином прийде повідомлення на пошту.

Рис. 3.7. Форма відновлення пароля

Після успішного входу користувач потрапляє на головну сторінку, що виконує роль персонального дашборду та відображає найближчу заплановану сесію разом із допоміжними віджетами (рис. 3.8).

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

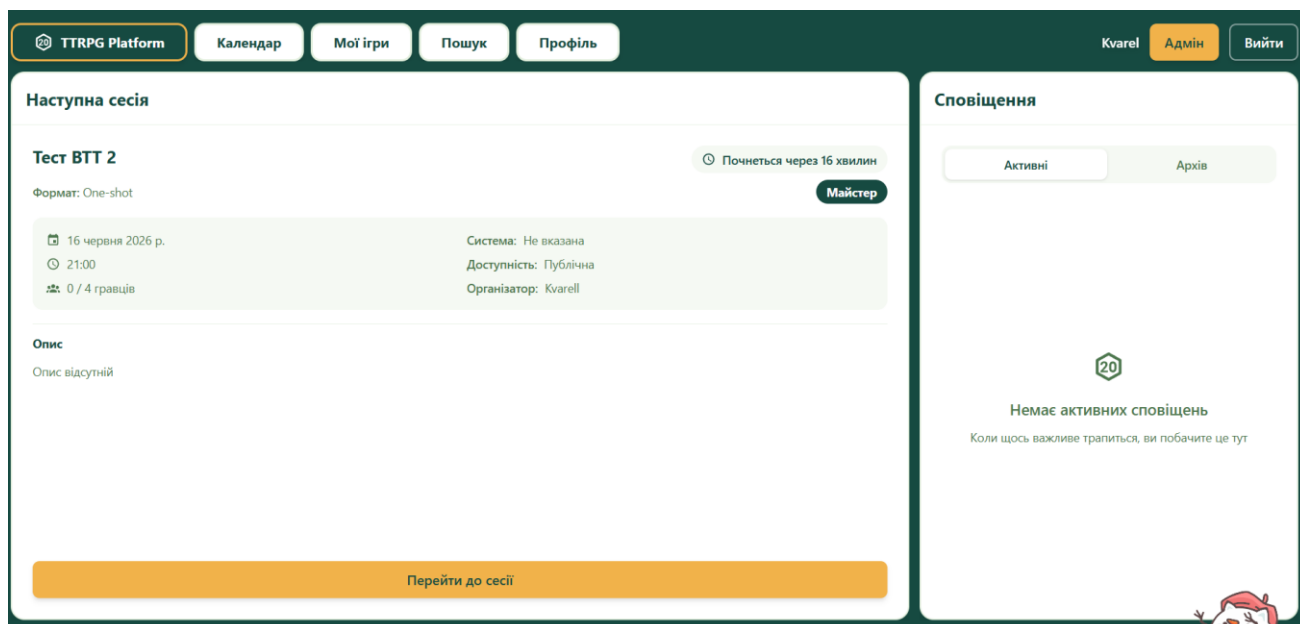


Рис. 3.8. Головна сторінка користувача

Інтерфейс зберігає зручність і на мобільних пристроях завдяки адаптивній верстці, приклад якої наведено на рисунку 3.9.

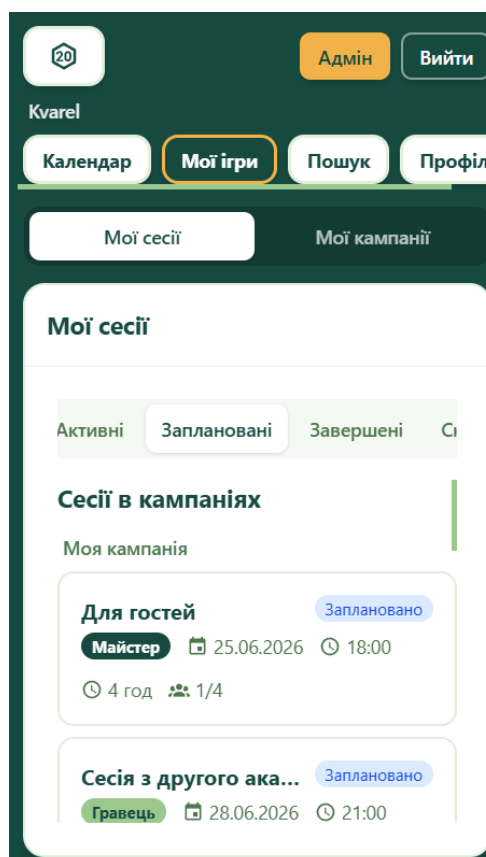


Рис. 3.9. Адаптивний вигляд інтерфейсу на мобільному пристрої

Для пошуку нових ігор передбачено каталог публічних кампаній та сесій з гнучкою фільтрацією, зображений на рисунку 3.10.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Талавер О.В.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

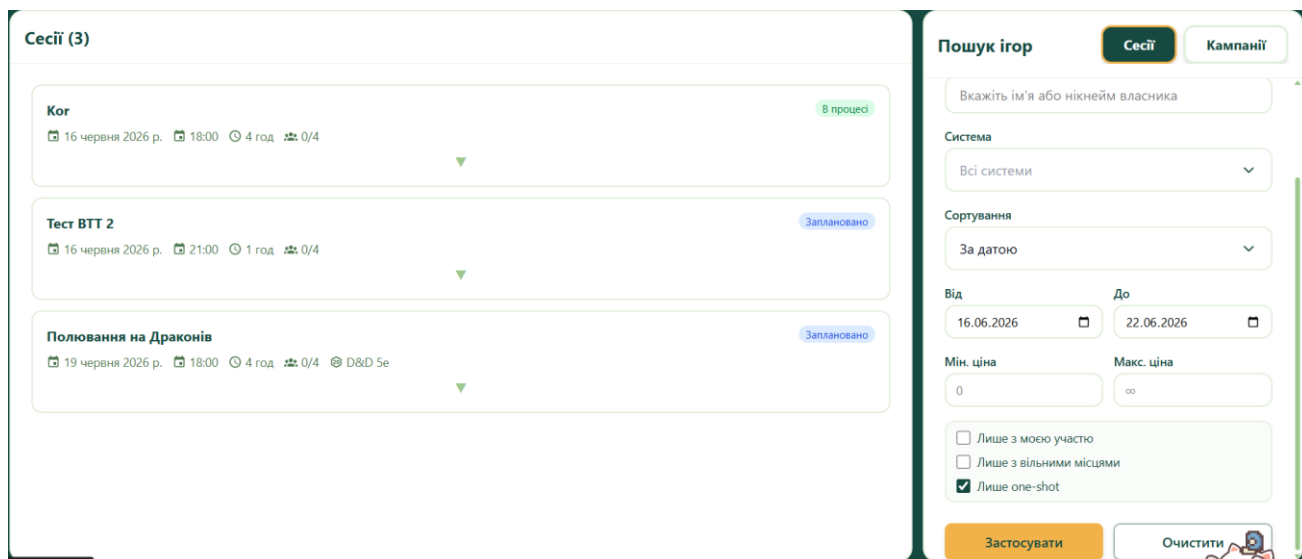


Рис. 3.10. Каталог доступних сесій з фільтрами та сортуванням

Окремо реалізовано пошук разових ігрових сесій за допомогою зручного календаря, що значно спрощує планування (рис. 3.11).

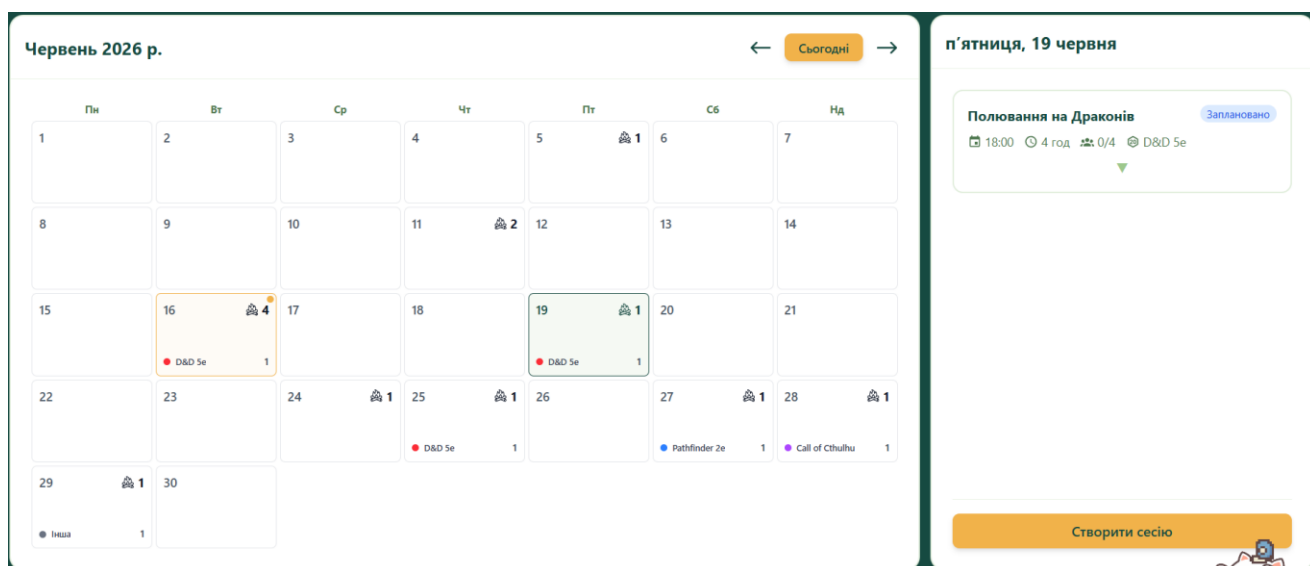


Рис. 3.11. Календар доступних користувачу сесій

На цьому ж екрані в користувача є можливість створити ігрову разову сесію на обраний в календарі день, заповнивши детальну форму у якій задаються дата, тривалість, обмеження кількості учасників та інше (рис. 3.12). При цьому користувачу виводяться підказки про обраний ним тип доступності та, якщо користувач створює платну сесію, про відсутність можливості змінити вартість в майбутньому. Крім того, слід зазначити, що для користувача була реалізована можливість створювати сесії з пошуком майстра, для цього теж передбачене окреме поле в формі.

Створити сесію

← Назад

Я буду Майстром
Керуватиму сесією самостійно

Шукаю Майстра
Я організатор, Майстром буде інший

Дата і час *

19.06.2026 18:00

Час гри

4 години

Макс. гравців

4

Ціна (грн)

100

Доступність

Публічна

Ціну гри неможливо буде змінити після її створення.

Публічна: відображається користувачам, заявки гравців підтверджуються автоматично.

Скасувати

Створити

Рис. 3.12. Форма створення ігрової сесії

Центральною організаційною сутністю платформи є кампанія. Її сторінка містить опис ігрового світу, склад учасників та перелік пов'язаних сесій, як показано на рисунку 3.13.

Наступна сесія

Для гостей

Почнеться через 9 днів

Кампанія: Моя кампанія

25 червня 2026 р.
18:00
1 / 4 гравців

Система: D&D 5e
Доступність: Гостьова
Організатор: Kvarrell

Опис
Мур мау

Перейти до сесії

Сесії кампанії (4)

ЗАПЛАНОВАНІ

Для гостей
Заплановано

25 червня 2026 р. 18:00 1/4 гравців
Мур мау

Сесія з другого акаунту...
Заплановано

28 червня 2026 р. 21:00 1/4 гравців
чіво?

Какаду та два маленьких поро...
Заплановано

29 червня 2026 р. 18:00 1/6 гравців
Про що це взагалі

СКАСОВАНІ

Створити сесію

Рис. 3.13. Сторінка ігрової кампанії

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк. 74
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Редагування кампанії виконується через відповідну форму, доступну на сторінці кампанії та наведену на рисунку 3.14.

Налаштування кампанії

Паспорт кампанії

В процесі

Назва кампанії*

Моя кампанія

Опис

Ну там, велетні €, дааааа

Ігрова система

Видимість

Dungeons & Dragons 5e

За заявкою

За заявкою: кампанія відображатиметься в глобальному пошуку і до неї зможуть подавати заявку.

Зберегти зміни

Рис. 3.14. Форма редагування кампанії

Власник кампанії керує її складом, призначає ролі та виключає учасників за допомогою інтерфейсу, зображеного на рисунку 3.15.

Налаштування кампанії

Учасники (5)

Ігрова система

Видимість

Dungeons & Dragons 5e

За заявкою

За заявкою: кампанія відображатиметься в глобальному пошуку і до неї зможуть подавати заявку.

Зберегти зміни

Небезпечна зона

Завершити кампанію

Після завершення кампанії не можна буде додавати нові сесії або приєднуватися до них.

Передача прав власності

Передача прав власності змінить власника. Ви станете звичайним учасником цієї кампанії.

Оберіть...

Передати права кампанії

Kvarell Майстер

ipz224_yaam Майстер

DIMITRIk Майстер

TOLIKLOH Майстер

Gnom1 Гравець

GM

GM

GM

Гравець

Рис. 3.15. Керування складом учасників кампанії

Заявки на приєднання нових гравців опрацьовуються в окремому розділі, що дозволяє схвалювати або відхиляти кандидатів (рис. 3.16).

Учасники (3/5)

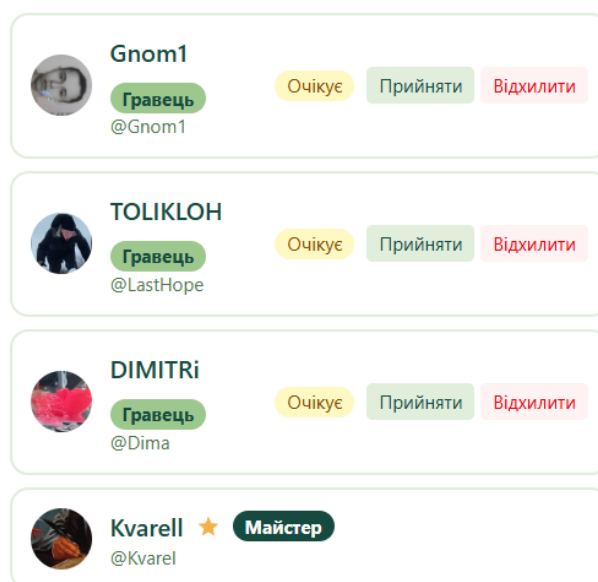


Рис. 3.16. Опрацювання заявок на приєднання

Безпосередній ігровий процес організовується навколо сесій. Сторінка сесії відображає її деталі, склад учасників та доступні дії залежно від ролі користувача (рис. 3.17).

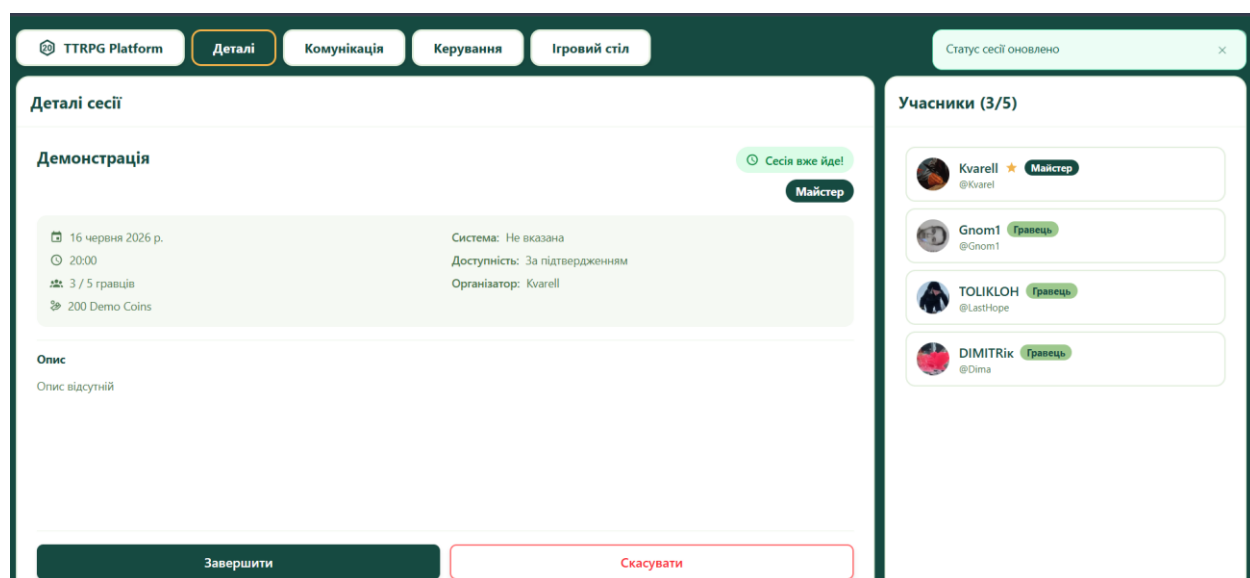


Рис. 3.17. Сторінка ігрової сесії

Деякі дії на платформі потребують додаткового підтвердження. Це зроблено з метою унеможливити випадкові дії та зробити користувацький інтерфейс більш дружнім. Так, наприклад, підтвердження для подання заявки до обраної сесії використовує модальне вікно (рис. 3.18). В цьому випадку

користувач також може обрати роль, оскільки власник не вказав себе майстром гри.

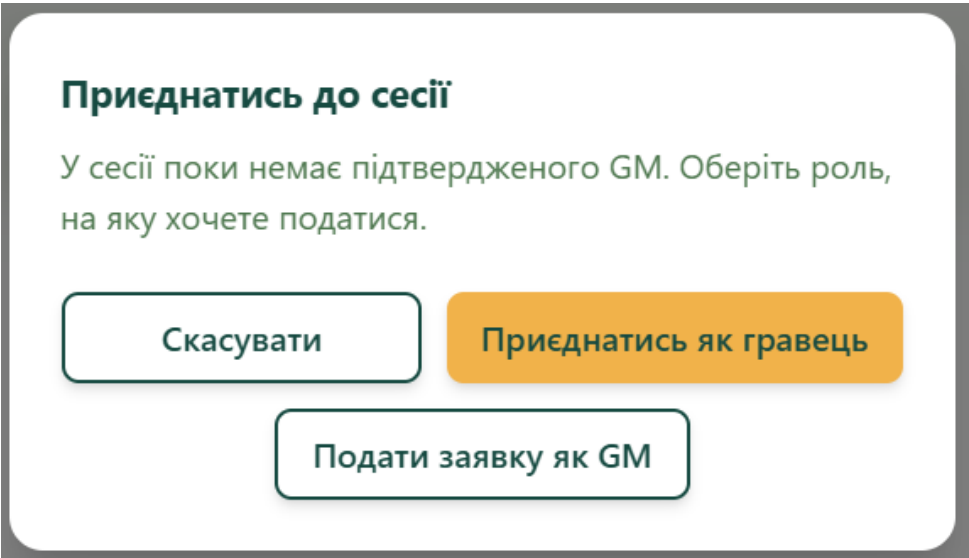


Рис. 3.18. Модальне вікно приєднання до сесії

Комунікацію учасників забезпечує підсистема реального часу. Текстове спілкування в межах кампаній і сесій реалізовано у вигляді чату, наведеного на рисунку 3.19.

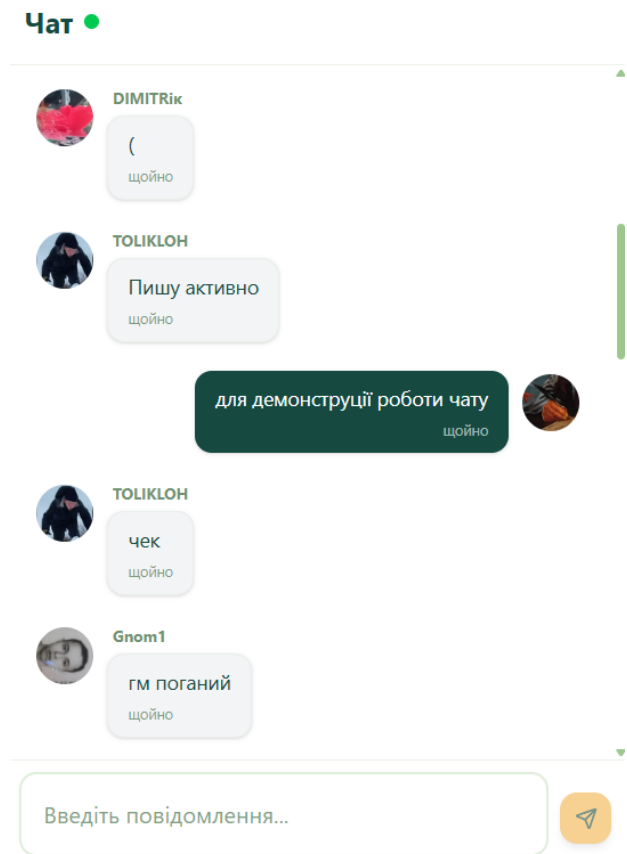


Рис. 3.19. Чат кампанії та сесії

Для проведення ігор безпосередньо у застосунку передбачено груповий відеозв'язок із сіткою учасників, зображений на рисунку 3.20. Учасник дзвінка керує власними аудіо- та відеопотоками за допомогою панелі керування.

Відеодзвінок

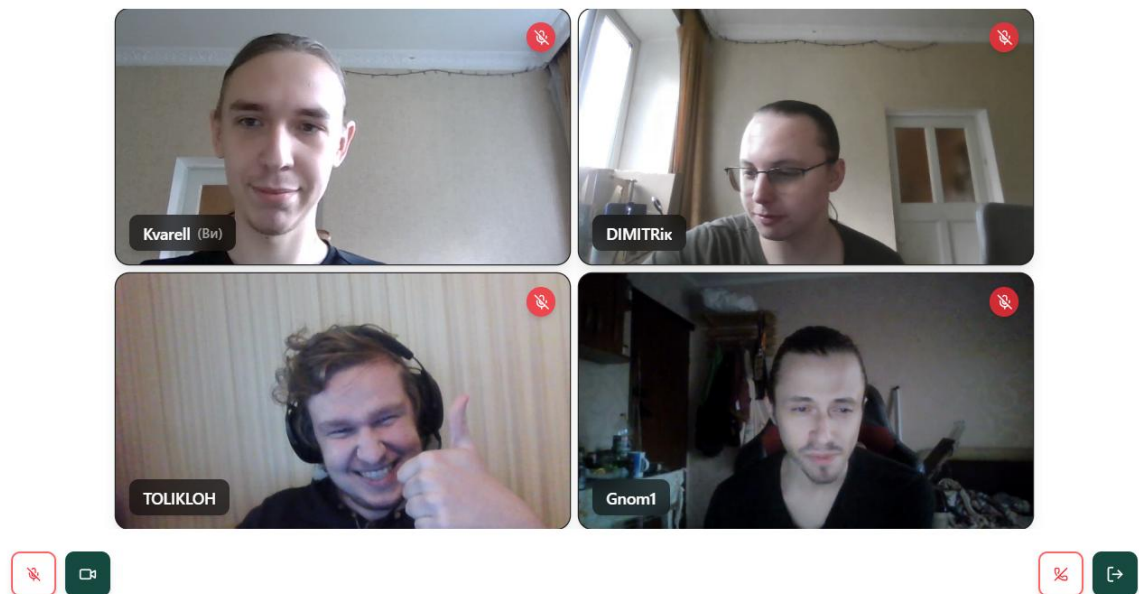


Рис. 3.20. Інтерфейс групового відеодзвінка

Про ключові події платформи користувач дізнається через центр сповіщень та повідомлення, що надходять у реальному часі (рис. 3.21).

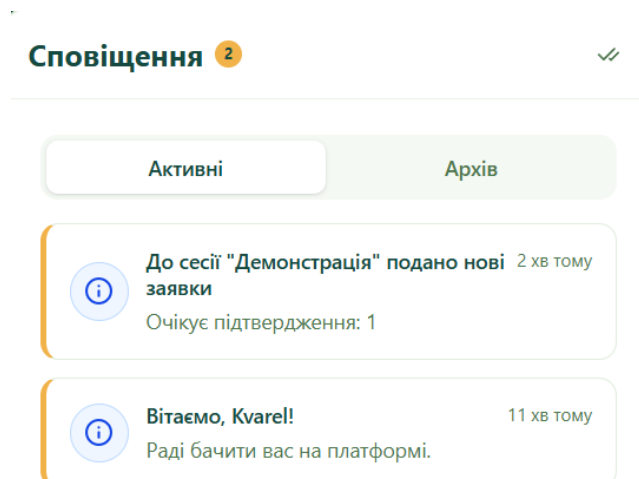


Рис. 3.21. Центр сповіщень

Система сповіщень також дозволяє користувачам реалізувати інтеграцію з Telegram-ботом, який буде дублювати сповіщення з платформи. З будь-якого сповіщення можна перейти до платформи щоб виконати якісь дії, або просто ознайомитись з даними (рис. 3.22).

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Талавер О.В.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

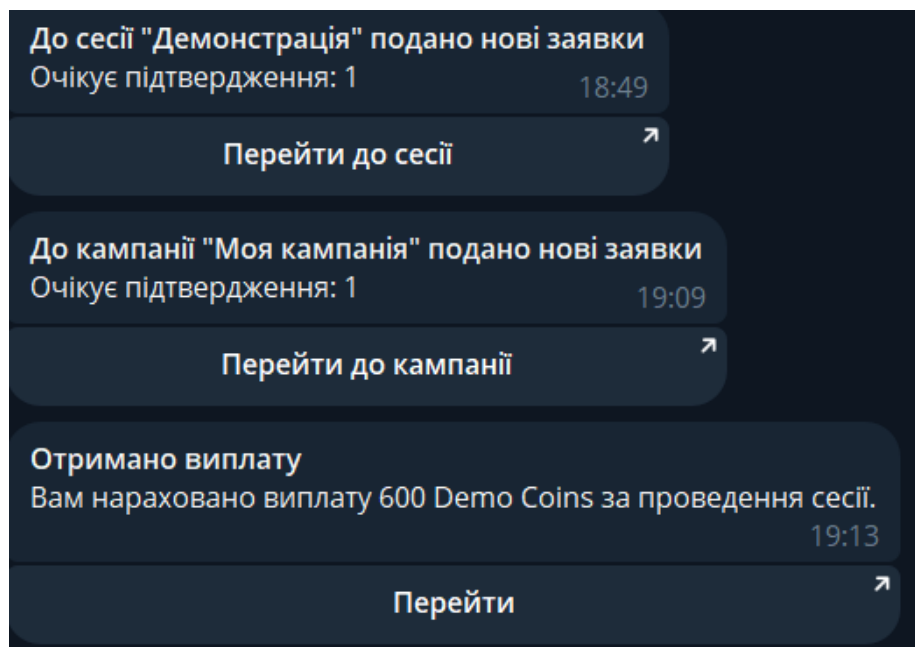


Рис. 3.22. Сповіщення з Telegram

Персональні дані та налаштування зосереджено в розділі профілю. Сторінку профілю користувача наведено на рисунку 3.23.

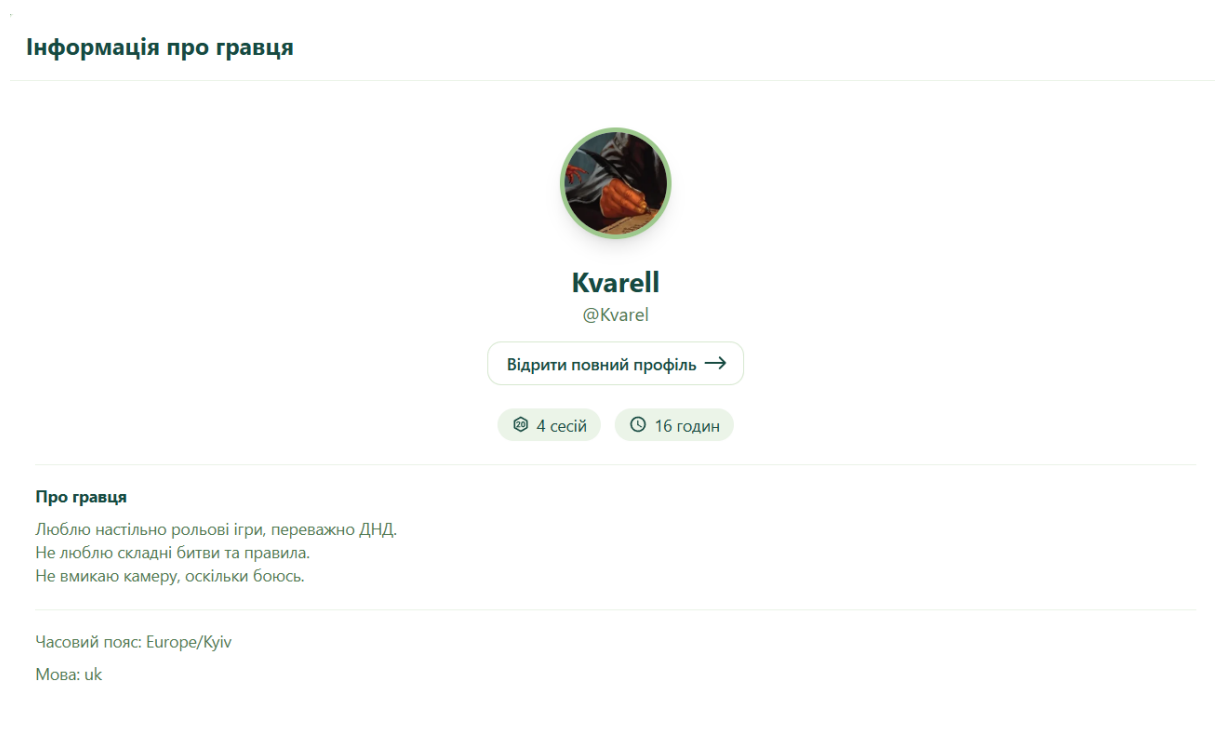


Рис. 3.23. Сторінка профілю користувача

Окремий розділ відповідає за безпеку акаунта і дозволяє змінити пароль чи електронну пошту та видалити обліковий запис. Більшість з цих дій розцінюються як критичні, тому потребують додаткового підтвердження через введення пароля або пошту (рис. 3.24).

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

Безпека акаунту

[Змінити email](#)

На новий email буде надіслано лист з посиланням для підтвердження.
Посилання діє 15 хвилин.

Видалення акаунту
Ця дія незворотна!

Увага! Ця дія незворотна!
При видаленні акаунту буде втрачено:

- Усі ваші персонажі та їх історія
- Кампанії, де ви були власником
- Історія сесій та повідомлення
- Баланс гаманця та транзакції
- Статистика та досягнення

[Я розумію, продовжити видалення](#)

Рис. 3.24. Налаштування безпеки акаунта

Адміністрування платформи виконується через окрему панель. Загальну статистику системи наведено на рисунку 3.25.

Адмін-панель Огляд Користувачі Кампанії Сесії Kvarel На головну Вийти

Користувачі 18 Кампанії 1 Сесії 17 Активні сесії 1

Рис. 3.25. Панель адміністратора зі статистикою платформи

Керування обліковими записами, зокрема блокування та розблокування користувачів, здійснюється в розділі, зображеному на рисунку 3.26.

Адмін-панель Огляд Користувачі Кампанії Сесії Kvarel На головну Вийти							
14	aaa	ab@a.a	USER	10.06.2026	0	0	Активний
13	123	kostiag123ogames@gmail.com	USER	10.06.2026	0	0	Активний
12	aaaa	a@a.a	USER	10.06.2026	0	0	Активний
11	Gnom1	ipz224_bdv@student.ztu.edu.ua	USER	24.05.2026	0	4	Активний
10	ipz224_yaam	ipz224_yaam@student.ztu.edu.ua	USER	21.05.2026	0	1	Активний

Рис. 3.26. Керування обліковими записами користувачів

Модерація створеного користувачами вмісту, а саме кампаній і сесій, виконується через відповідний інтерфейс (рис. 3.27).

Пошук за назвою або GM... Знайти Всі статуси						
ID	Назва	GM	Кампанія	Статус	Дата	Гравці
29	Тест BTT 2	Kvarel	One-shot	Заплановано	16.06.2026	1/4
28	Демонстрація	Kvarel	One-shot	Завершена	16.06.2026	4/5
27	Ког	Gnom	One-shot	Активна	16.06.2026	1/4

Рис. 3.27. Модерація кампаній і сесій

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

Наведені екрани демонструють, що інтерфейс платформи цілісно охоплює весь життєвий цикл взаємодії користувача, від реєстрації та пошуку ігор до організації кампаній, спілкування в реальному часі та адміністрування. Підтвердження коректності роботи описаного функціоналу наведено в наступному параграфі.

3.3 Тестування роботи програмного комплексу

Для підтвердження коректності та надійності реалізованого програмного комплексу застосовано багаторівневу стратегію тестування. Вона поєднує автоматизоване тестування серверної та клієнтської частин, наскрізне функціональне тестування у реальному браузері, перевірку інфраструктурного компонента відеозв'язку, навантажувальне тестування та безперервну інтеграцію всіх перевірок у конвеєр розробки. Такий підхід дозволяє виявляти помилки на ранніх етапах і гарантувати стабільність системи при подальшому розвитку.

Серверну частину покрито набором із 59 автоматизованих тестів, реалізованих засобами вбудованого інструмента тестування середовища Node.js. Тести поділено на чотири категорії, наведені в таблиці 3.2, кожна з яких відповідає за окремий рівень перевірки.

Таблиця 3.2.

Категорії автоматизованих серверних тестів

Категорія	Кількість	Призначення
unit	36	перевірка окремих функцій, сервісів і доменних правил в ізоляції
integration	18	наскрізні сценарії на рівні API та бази даних
security	3	безпекові інваріанти системи
policy	2	правила доступу та видимості ресурсів

Модульні тести виконуються в ізоляції та не потребують зовнішніх залежностей, результат їхнього успішного проходження наведено на рисунку 3.28.

```

✓ createCallHandler (6.743328ms)
✓ WS Call Handler (13.035657ms)
[2026-06-16 23:30:18.544] INFO: Prisma Client
[2026-06-16 23:30:18.579] INFO: [SSE] Запущен
[2026-06-16 23:30:18.686] DEBUG: CallRoom created
  sessionId: "session-1"
i tests 303
i suites 12
i pass 303
i fail 0
i cancelled 0
i skipped 0
i todo 0
i duration_ms 4193.97164

```

Рис. 3.28. Результат проходження модульних тестів сервера

Інтеграційні тести, на відміну від модульних, перевіряють взаємодію компонентів із реальною базою даних, тому перед їхнім запуском автоматично піднімається окрема ізольована тестова база даних у контейнері, що унеможливорює вплив на робочі дані. Процес запуску інтеграційних тестів зображено на рисунку 3.29.

```

t20f0@Kvarel:~/ttrpg-platform/server$ npm run test:integration

> server@1.0.0 test:integration
> docker compose --profile test up -d db_test redis_test && cross-env DATABASE_URL='postgres://ttrpg_test?schema=public' npx prisma db push && cross-env DATABASE_URL='postgres://ttrpg_test?schema=public' REDIS_URL='redis://localhost:6380' node --test tests/integration

[+] Running 2/2
  ✓ Container ttrpg-platform-redis_test-1 Started
  ✓ Container ttrpg-platform-db_test-1 Started
  ◇ injected env (45) from ../.env // tip: ~ auth for agents [www.vestauth.com]
  Loaded Prisma config from prisma.config.js.

```

Рис. 3.29. Запуск інтеграційних тестів з ізольованою тестовою базою даних

Результат успішного проходження інтеграційних тестів наведено на рисунку 3.30.

```

✓ WS Chat lifecycle: joining,
i tests 96
i suites 0
i pass 96
i fail 0
i cancelled 0
i skipped 0
i todo 0
i duration_ms 15210.042526

```

Рис. 3.30. Результат виконання інтеграційних тестів сервера

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Талавер О.В.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

Клієнтську частину протестовано на двох рівнях. Модульні тести компонентів та логіки інтерфейсу реалізовано засобами Vitest; результат їхнього виконання зображено на рисунку 3.31.

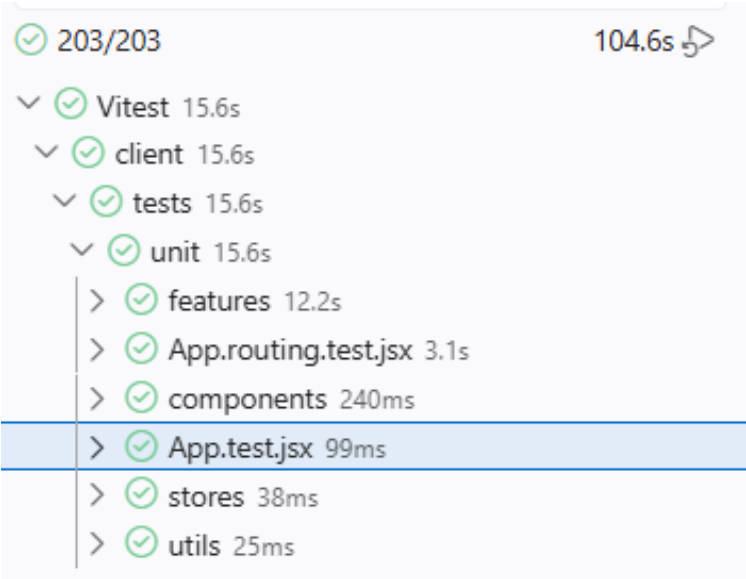


Рис. 3.31. Результат проходження тестів клієнтської частини

Наскрізне функціональне тестування реалізовано за допомогою інструмента Playwright, який автоматизує дії користувача в реальному браузері та перевіряє ключові користувацькі сценарії, в нашому випадку такі критичні дії, як видалення акаунта та вхід в систему за допомогою логіна [33]. Результат виконання наскрізних тестів наведено на рисунку 3.32.

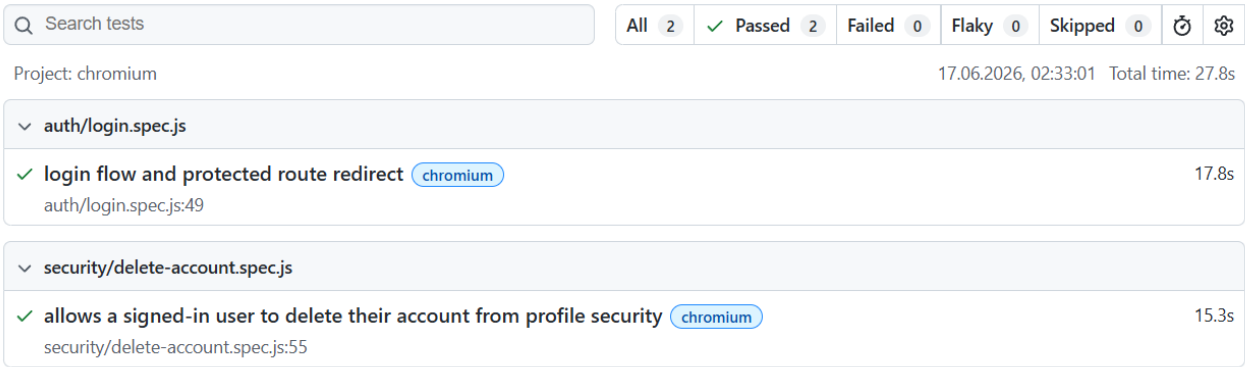


Рис. 3.32. Результат виконання наскрізних тестів Playwright

Для оцінки повноти тестування виміряно покриття коду. Покриття серверної частини обчислюється вбудованим інструментом Node.js і виводиться у консоль, як показано на рисунку 3.33.

WS			
ws-auth.middleware.js	15.65	100.00	0.00
ws-call.handler.js	41.23	59.26	70.00
ws-chat.handler.js	91.14	76.92	100.00
ws-room.manager.js	68.81	61.11	75.00
ws-router.js	65.08	58.33	100.00
ws-server.js	14.78	100.00	0.00
ws-utils.js	89.33	75.00	100.00
ws-vtt.handler.js	10.77	100.00	0.00
all files	55.85	71.54	46.64
end of coverage report			

Рис. 3.33. Звіт про покриття коду серверної частини

Покриття клієнтської частини формується у вигляді інтерактивного звіту, наведеного на рисунку 3.34. Тут слід зазначити, що наскрізні тести Playwright не беруть участі в підрахунку покриття.

All files

26.09% Statements 2368/9073 19.66% Branches 1368/6955 20.44% Functions 493/2411 27.13% Lines 2240/8255

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements
src	78.57%

Рис. 3.34. Звіт про покриття коду клієнтської частини

Окремої уваги потребує перевірка працездатності підсистеми відеозв'язку, зокрема обходу трансляції мережевих адрес. Під час встановлення з'єднання за технологією WebRTC протокол ICE збирає кандидатів локальних, відображених та ретрансляційних типів. Наявність та успішне використання ретрансляційного кандидата, отриманого від TURN-сервера coturn, підтверджує, що з'єднання може бути встановлене навіть у складних мережевих умовах, коли пряме сполучення неможливе. Список зібраних ICE-кандидатів із ретрансляційним кандидатом TURN-сервера наведено на рисунку 3.35.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Талавер О.В.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

https://myttrpg.me/session/26?tab=communication, {"iceServers":[{"urls":["turn:167.172.111.89:3478"]}], "bundlePolicy": "max-bundle", "alwaysN

ICE connection state: new => "checking" => "connected"
 Connection state: new => "connecting" => "connected"
 Signaling state: new => "have-local-offer" => "stable"
 ICE Candidate pair: 193.194.126.206:63331 <=> 167.172.111.89:40439
 ▼ ICE candidate grid

Candidate (pair) id	State / Candidate type	Network type / address	Port	Protocol / candidate type	(Pair) Priority	Bytes sent / received
CP21mU/NaR_BYCV TvV	succeeded	wifi		udp	0x402710ffdcf63c00	2245823 / 10321
I21mU/NaR	local-candidate	193.194.126.206	63331	prfix	0x6e7b1eff110 31518 255	
IBYCVTvV	remote-candidate	167.172.111.89	40439	host	0x402710ff	
CP2YrzNRDt_dkFE35 mT	waiting	wifi		tcp active	0x4026acffb4f63c00	0 / 0
I2YrzNRDt	local-candidate	192.168.50.63	9	host	0x5a7b1eff90 31518 255	
IdkFE35mT	remote-candidate	167.172.111.89	40040	host	0x4026acff	
CP8VdbIVik_BYCVTv fV	waiting	wifi		udp	0x37b1fff804e2200	0 / 0
I8VdbIVik	local-candidate	167.172.111.89	50719	relay(udp) turn:167.172.111.89:3478?transport=udp	0x37b1fff3 31519 255	
IBYCVTvV	remote-candidate	167.172.111.89	40439	host	0x402710ff	
CPJixlMwh9_dkFE35 mT	waiting	ethernet		tcp active	0x4026acffb4fa3c00	0 / 0
I8VdbIVik	local-candidate	192.168.50.63	9	host	0x5a7b1eff	

Рис. 3.35. TURN-сервер у списку ICE-кандидатів з'єднання

Для оцінки продуктивності системи під навантаженням проведено навантажувальне тестування засобами Apache JMeter [34]. Як цільовий обрано ендпоінт календаря сесій, який відповідає за репрезентативний сценарій перегляду даних користувачами. Автоматично згенерований звіт продемонстровано на рисунку 3.36. В даному випадку тестування проводилось в 50 потоків, які додаються протягом 20 секунд та викликають ендпоінт 60 секунд кожен.

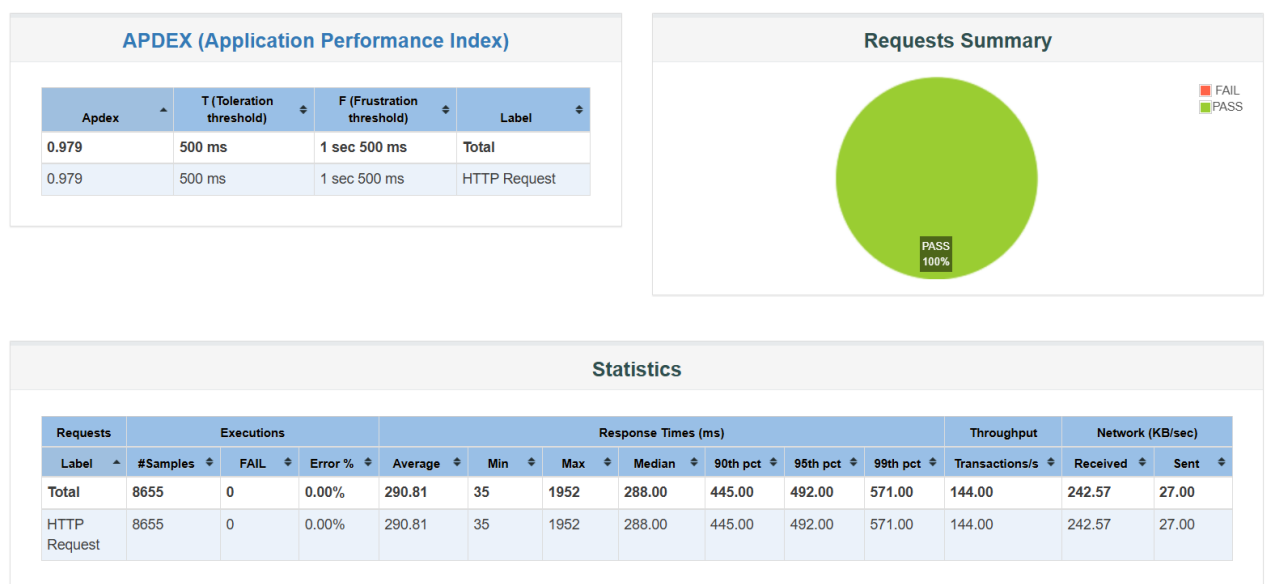


Рис. 3.36. Звіт навантажувального тестування JMeter

Для наочності тестування виконувалося послідовно для 50, 100, 200 та 500 одночасних віртуальних користувачів; узагальнені результати наведено в таблиці 3.3.

Таблиця 3.3.

Результати навантажувального тестування

Потоки	Throughput (req/s)	Avg (мс)	p95 (мс)	Помилки, %	APDEX
50	144	291	492	0.00%	0,979
100	172,33	484,46	737	0.00%	0,749
200	173,74	957,49	1296	0.00%	0,539
500	144,38	2857,74	3907,30	0.00%	0,039

Аналіз результатів показує, що пропускна здатність системи зростає до приблизно 170–174 запитів за секунду за 100–200 одночасних користувачів, після чого виходить на насичення й навіть дещо знижується за 500 потоків. Це свідчить про досягнення межі продуктивності, зумовленої обмеженням обчислювального ресурсу серверного контейнера. Водночас середній час відгуку та 95-й перцентиль зростають із навантаженням, від 291 мс за 50 користувачів до 2858 мс за 500, оскільки за межею насичення запити починають накопичуватися в черзі. Показник задоволеності APDEX залишається відмінним за помірною навантаження та поступово знижується за екстремальних значень. Принципово важливо, що частка помилок на всіх рівнях навантаження дорівнює нулю: система зберігає коректність і стабільність навіть у стані перевантаження, поступаючись лише у швидкості, але не в надійності.

Причину виходу пропускної здатності на плато підтверджує моніторинг процесорного ресурсу. На панелі керування хостингу DigitalOcean завантаження процесора droplet'a у пік тесту становить близько 30 відсотків, що наведено на рисунку 3.37.

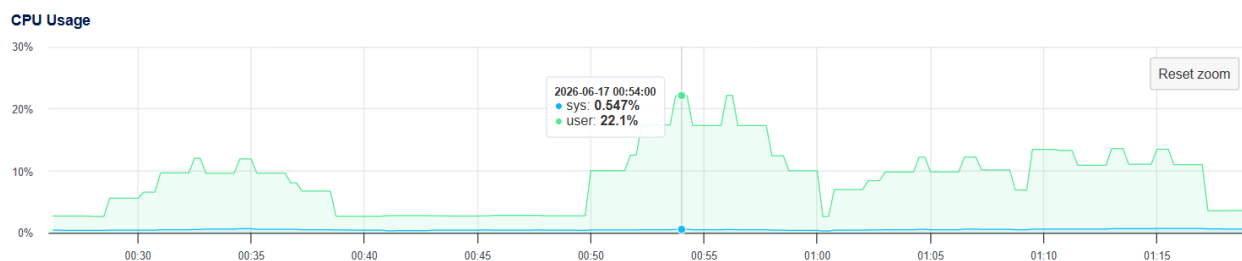


Рис. 3.37. Завантаження процесора сервера на панелі DigitalOcean

Водночас моніторинг на рівні контейнерів засобами Docker демонструє, що серверний контейнер у момент пікового навантаження споживає понад 100 відсотків виділеного процесорного ядра, як показано на рисунку 3.38.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
b3fc1921f9d7	ttrpg_server	103.87%	262.6MiB / 768MiB	34.19%	0B / 0B	27.4MB / 3.73MB	17
f8f28efc952a	ttrpg_db	3.74%	36.13MiB / 512MiB	7.06%	19.2MB / 25.1MB	6.86MB / 40.7MB	11
65eb5cdcb870	ttrpg_redis	0.79%	6.176MiB / 128MiB	4.82%	5.35MB / 4.2MB	5.79MB / 1.13MB	6

Рис. 3.38. Завантаження серверного контейнера за даними Docker

Зіставлення цих показників доводить, що обмежувальним чинником продуктивності є саме процесорний ресурс серверного застосунку, а не загальна потужність сервера. Фактично ми отримуємо підстави перерозглянути розподіл навантаження між контейнерами, наведений в першому параграфі цього розділу. Тим не менш, отримані результати узгоджуються з цільовим масштабом платформи, орієнтованим на невеликі ігрові групи: за реалістичного навантаження система демонструє відмінний відгук, а для обслуговування значно більшої кількості одночасних користувачів передбачено вертикальне масштабування серверних ресурсів, зазначене в першому розділі.

Усі описані перевірки інтегровано в автоматизований конвеєр безперервної інтеграції, що запускається при кожному внесенні змін до репозиторію. Конвеєр послідовно виконує сканування на наявність секретів і вразливостей, перевірку якості коду статичним аналізатором, складання застосунку, прогін усіх категорій тестів із вимірюванням покриття та наскрізне тестування. Структуру конвеєра в системі GitHub Actions наведено на рисунку 3.39.

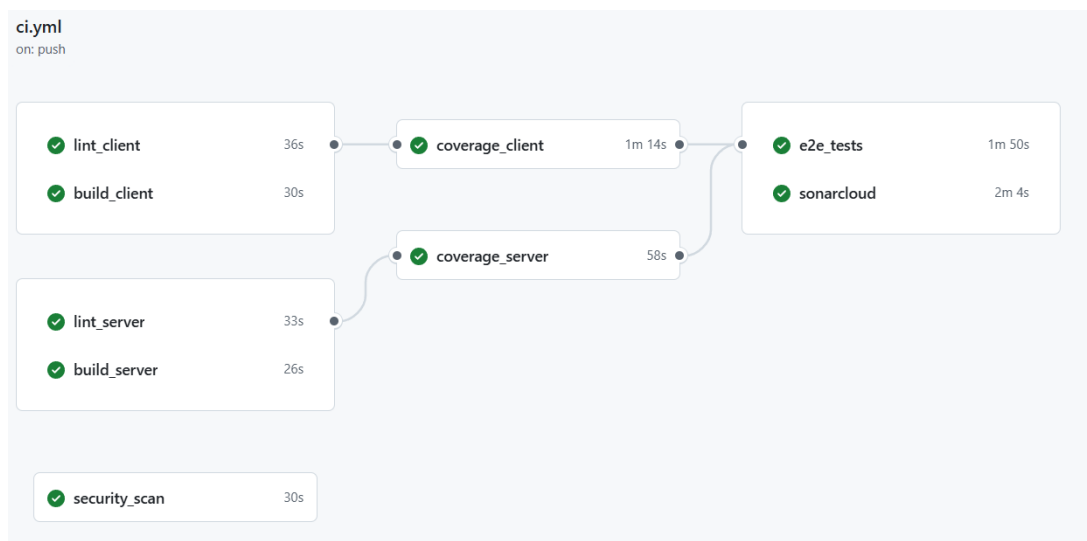


Рис. 3.39. Конвеєр безперервної інтеграції у GitHub Actions

Аналогічний конвеєр налаштовано й у системі GitLab CI, що забезпечує гнучкість процесу розробки; його структуру зображено на рисунку 3.40.

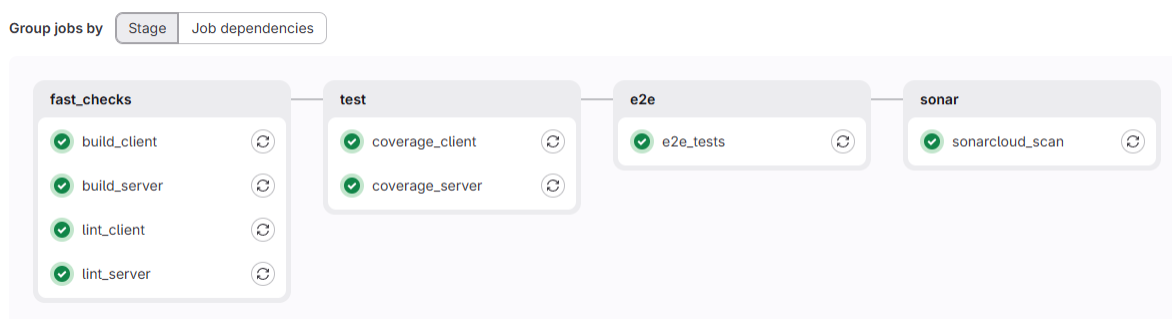


Рис. 3.40. Конвеєр безперервної інтеграції у GitLab CI

Завдяки автоматичному запуску всіх перевірок перед інтеграцією змін будь-яка регресія виявляється негайно, що гарантує стабільність основної гілки коду та надійність подальшого розгортання, описаного у параграфі 3.1.

Висновки до третього розділу

У третьому розділі описано встановлення, інтерфейс та тестування реалізованого програмного комплексу, що підтвердило працездатність і коректність розробленої платформи.

Систему розгорнуто на хмарному сервері у вигляді набору контейнерів, керованих засобами Docker і Docker Compose, з автоматичним застосуванням міграцій та запуском підсистем без ручного втручання. Процес оновлення автоматизовано засобами безперервної доставки з автоматичним відкатом у разі

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

невдачі; також описано конфігурування системи та дворівневу модель розмежування прав доступу.

Розглянуто структуру користувацького інтерфейсу, що цілісно охоплює весь цикл взаємодії користувача – від реєстрації та пошуку ігор до організації кампаній, спілкування в реальному часі й адміністрування платформи.

Працездатність системи підтверджено багаторівневим тестуванням, що охопило модульні, інтеграційні та наскрізні тести серверної й клієнтської частин із контролем покриття коду, перевірку відеозв'язку через ретрансляційний сервер та навантажувальне тестування. Останнє показало стабільну роботу без помилок і визначило межу продуктивності, що відповідає цільовому масштабу, а інтеграція перевірок у конвеєр CI/CD гарантує підтримання якості коду.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

ВИСНОВКИ

У межах випускної кваліфікаційної роботи досягнуто поставленої мети, спроектовано та розроблено вебплатформу для організації настільних рольових ігор і забезпечення комунікації між гравцями та майстрами гри.

Проаналізовано предметну область організації настільних рольових ігор та виконано порівняльний огляд наявних програмних рішень. Встановлено, що спільною проблемою існуючих сервісів є відсутність консолідованої екосистеми, яка б поєднувала гнучкий організаційний менеджмент із розвиненими засобами комунікації, через що користувачі змушені застосовувати кілька розрізнених інструментів. Це обґрунтувало доцільність розробки власної спеціалізованої платформи.

Спроектовано об'єктно-орієнтовану структуру системи та модель даних, формалізовано динамічну поведінку ключових процесів засобами мови UML, а отримані рішення втілено у програмному коді з дотриманням принципів багатошарової архітектури. Реалізовано надійну рольову модель доступу, безпечну автентифікацію, криптографічно захищені посилання-запрошення та гарантовану доставку сповіщень.

Систему контейнеризовано й розгорнуто на хмарному сервері та підтверджено її працездатність багаторівневим тестуванням, що охопило модульні, інтеграційні та наскрізні тести, перевірку відеозв'язку через ретрансляційний сервер і навантажувальне тестування. Випробування показали стабільну роботу системи без помилок та визначили межу її продуктивності, що відповідає цільовому масштабу застосування. Розроблена платформа консолідує засоби планування, адміністрування та комунікації в єдиному середовищі, знижує адміністративне навантаження на організаторів і поріг входу для нових учасників, а її модульна архітектура створює підґрунтя для подальшого розширення функціоналу.

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ярмоленко А.М., Талавер О.В., Архітектурний аналіз та обґрунтування вибору SFU-сервера для побудови системи відеоконференцзв'язку на основі WebRTC. Тези доповідей науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 11-15 травня 2026 року. Житомир: Державний університет «Житомирська політехніка», 2026.

2. Roll20: Online virtual tabletop for tabletop role-playing games [Електронний ресурс]. URL: <https://roll20.net/> (дата звернення: 17.05.2026).

3. StartPlaying: Find your next tabletop RPG game [Електронний ресурс]. URL: <https://startplaying.games/> (дата звернення: 17.05.2026).

4. GeekerApp: платформа для пошуку ігор та однодумців [Електронний ресурс]. URL: <https://geekerapp.com/> (дата звернення: 17.05.2026).

5. Foote B., Yoder J. Big Ball of Mud. Pattern Languages of Program Design 4 / Eds. N. Harrison, B. Foote, H. Rohnert. Boston : Addison-Wesley, 2000. С. 654–692. URL: <https://www.laputan.org/mud/> (дата звернення: 17.05.2026).

6. Single-page application. MDN Web Docs Glossary [Електронний ресурс]. URL: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> (дата звернення: 17.05.2026).

7. Osmani A., Miller J. Rendering on the Web. web.dev [Електронний ресурс]. URL: <https://web.dev/articles/rendering-on-the-web> (дата звернення: 17.05.2026).

8. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.

9. Server-Sent Events. MDN Web Docs [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата звернення: 17.05.2026).

10. Fette I., Melnikov A. The WebSocket Protocol : RFC 6455. Internet Engineering Task Force, 2011 [Електронний ресурс]. URL: <https://www.rfc-editor.org/info/rfc6455/> (дата звернення: 17.05.2026).

11. Mediasoup: Cutting Edge WebRTC Video Conferencing [Електронний ресурс]. URL: <https://mediasoup.org/documentation/> (дата звернення: 17.05.2026).

		Ярмоленко А.М.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Талавер О.В.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

12. React: The library for web and native user interfaces [Електронний ресурс]. URL: <https://react.dev/> (дата звернення: 17.05.2026).
13. Vite: Next Generation Frontend Tooling [Електронний ресурс]. URL: <https://vite.dev/> (дата звернення: 17.05.2026).
14. Zustand: State management solution for React [Електронний ресурс]. URL: <https://zustand.docs.pmnd.rs/> (дата звернення: 17.05.2026).
15. TanStack Query: Asynchronous state management for React [Електронний ресурс]. URL: <https://tanstack.com/query/> (дата звернення: 17.05.2026).
16. Tailwind CSS: A utility-first CSS framework [Електронний ресурс]. URL: <https://tailwindcss.com/> (дата звернення: 17.05.2026).
17. Node.js Documentation [Електронний ресурс]. URL: <https://nodejs.org/docs/v22.0.0/> (дата звернення: 17.05.2026).
18. Express: Web framework for Node.js [Електронний ресурс]. URL: <https://expressjs.com/> (дата звернення: 17.05.2026).
19. Ws: a Node.js WebSocket library [Електронний ресурс]. URL: <https://github.com/websockets/ws> (дата звернення: 17.05.2026).
20. Coturn: TURN/STUN server implementation [Електронний ресурс]. URL: <https://github.com/coturn/coturn> (дата звернення: 17.05.2026).
21. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. Internet Engineering Task Force, 2015 [Електронний ресурс]. URL: <https://www.rfc-editor.org/rfc/rfc7519> (дата звернення: 17.05.2026).
22. Bcrypt: A library to help hash passwords. npm [Електронний ресурс]. URL: <https://www.npmjs.com/package/bcrypt> (дата звернення: 17.05.2026).
23. Nodemailer: Send emails with Node.js [Електронний ресурс]. URL: <https://nodemailer.com/> (дата звернення: 17.05.2026).
24. Prisma ORM Documentation [Електронний ресурс]. URL: <https://www.prisma.io/docs> (дата звернення: 17.05.2026).
25. PostgreSQL 15 Documentation [Електронний ресурс]. URL: <https://www.postgresql.org/docs/15/> (дата звернення: 17.05.2026).

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

26. Redis Documentation [Електронний ресурс]. URL: <https://redis.io/docs/> (дата звернення: 17.05.2026).

27. Docker Documentation [Електронний ресурс]. URL: <https://docs.docker.com/> (дата звернення: 17.05.2026).

28. Docker Compose Documentation [Електронний ресурс]. URL: <https://docs.docker.com/compose/> (дата звернення: 17.05.2026).

29. Wiegers K., Beatty J. Software Requirements. 3rd ed. Redmond : Microsoft Press, 2013. 672 p.

30. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2003. 208 p.

31. Richardson C. Pattern: Transactional outbox. Microservices.io [Електронний ресурс]. URL: <https://microservices.io/patterns/data/transactional-outbox.html> (дата звернення: 17.05.2026).

32. Wiggins A. The Twelve-Factor App [Електронний ресурс]. URL: <https://12factor.net/> (дата звернення: 17.05.2026).

33. Playwright: End-to-end testing for modern web apps [Електронний ресурс]. URL: <https://playwright.dev/> (дата звернення: 17.05.2026).

34. Apache JMeter User's Manual [Електронний ресурс]. URL: <https://jmeter.apache.org/usermanual/> (дата звернення: 17.05.2026).

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

Лістинг вихідного коду

Лістинг session-participants.service.js:

```

const { Prisma } = require('@prisma/client');
const { logger } = require('../lib/logger');
const VALID_PARTICIPANT_STATUSES = new Set(['PENDING', 'CONFIRMED', 'DECLINED']);
const JOINABLE_SESSION_STATUSES = new Set(['PLANNED']);

function parseId(value) {
  return Number.parseInt(value, 10);
}

function normalizeJoinRole(role, AppError, ERROR_CODES) {
  const normalizedRole = String(role || 'PLAYER').toUpperCase();

  if (!['PLAYER', 'GM'].includes(normalizedRole)) {
    throw new AppError(ERROR_CODES.SESSION_JOIN_ROLE_INVALID);
  }

  return normalizedRole;
}

function assertJoinableSession(session, AppError, ERROR_CODES) {
  if (!JOINABLE_SESSION_STATUSES.has(session.status)) {
    throw new AppError(ERROR_CODES.SESSION_JOIN_STATUS_FORBIDDEN, null, {
      status: session.status,
    });
  }

  if (session.campaign?.status === 'FINISHED') {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        throw new AppError(ERROR_CODES.SESSION_JOIN_CAMPAIGN_FINISHED);
    }

    if (new Date(session.date) < new Date()) {
        throw new AppError(ERROR_CODES.SESSION_JOIN_ALREADY_STARTED);
    }
}

async function assertUserNotJoinedYet({ prisma, sessionId, userId,
AppError, ERROR_CODES }) {
    const existingParticipant = await
prisma.sessionParticipant.findUnique({
        where: {
            userId_sessionId: { userId, sessionId: parseId(sessionId) },
        },
    });

    if (existingParticipant) {
        throw new AppError(ERROR_CODES.SESSION_JOIN_ALREADY_PARTICIPANT);
    }
}

function assertPlayerCapacity({ normalizedRole, session, AppError,
ERROR_CODES }) {
    if (normalizedRole !== 'PLAYER') {
        return;
    }

    const playerCount = session.participants.filter((participant) =>
participant.role === 'PLAYER').length;

    if (playerCount >= session.maxPlayers) {
        throw new AppError(ERROR_CODES.SESSION_JOIN_NO_FREE_PLAYER_SLOTS);
    }
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		96

```

}

function assertGmCanJoin({ normalizedRole, session, permissionHelpers,
  AppError, ERROR_CODES }) {
  if (normalizedRole !== 'GM') {
    return;
  }

  const confirmedGm = permissionHelpers._getConfirmedGm(session);
  if (confirmedGm) {
    throw new AppError(ERROR_CODES.SESSION_GM_ALREADY_EXISTS);
  }
}

function assertCampaignPlayerAccess({
  normalizedRole,
  session,
  AppError,
  ERROR_CODES,
}) {
  const isCampaignSession = Boolean(session.campaignId);
  const isCampaignMember = Boolean(session.viewer?.isCampaignMember);

  if (
    normalizedRole === 'PLAYER'
    && isCampaignSession
    && session.visibility === 'PRIVATE'
    && !isCampaignMember
  ) {
    throw new
AppError(ERROR_CODES.SESSION_JOIN_PRIVATE_CAMPAIGN_MEMBERS_ONLY);
  }
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		97

```

}

function resolveJoinStatus({ normalizedRole, session }) {
  const isCampaignSession = Boolean(session.campaignId);
  const isCampaignMember = Boolean(session.viewer?.isCampaignMember);

  if (normalizedRole !== 'PLAYER') {
    return 'PENDING';
  }

  if (!isCampaignSession && session.visibility === 'PUBLIC') {
    return 'CONFIRMED';
  }

  if (isCampaignSession && session.visibility === 'PRIVATE' &&
isCampaignMember) {
    return 'CONFIRMED';
  }

  return 'PENDING';
}

function resolveGuestFlag({ normalizedRole, session }) {
  return normalizedRole === 'PLAYER'
    && Boolean(session.campaignId)
    && session.visibility === 'PUBLIC'
    && !session.viewer?.isCampaignMember;
}

function assertValidParticipantStatus(status, AppError, ERROR_CODES) {
  if (!INVALID_PARTICIPANT_STATUSES.has(status)) {
    throw new AppError(ERROR_CODES.SESSION_PARTICIPANT_STATUS_INVALID);
  }
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		98

```

    }
}

function assertSessionIsActiveForParticipantManagement(session, AppError,
ERROR_CODES) {
    if (['FINISHED', 'CANCELED'].includes(session.status)) {
        throw new
AppError(ERROR_CODES.SESSION_PARTICIPANT_MANAGEMENT_UNAVAILABLE);
    }
}

async function findParticipantOrThrow({ prisma, participantId, sessionId,
AppError, ERROR_CODES }) {
    const participant = await prisma.sessionParticipant.findUnique({
        where: { id: parseId(participantId) },
    });

    if (!participant?.sessionId || participant.sessionId !==
parseId(sessionId)) {
        throw new AppError(ERROR_CODES.SESSION_PARTICIPANT_NOT_FOUND);
    }

    return participant;
}

async function declinePendingParticipant({
    prisma,
    participant,
    participantId,
    AppError,
    ERROR_CODES,
    notificationService,
    walletService,

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }) {
      if (participant.status !== 'PENDING') {
        throw new
        AppError(ERROR_CODES.SESSION_PARTICIPANT_DECLINE_PENDING_ONLY);
      }

      return await prisma.$transaction(async (tx) => {
        const session = await tx.session.findUnique({
          where: { id: participant.sessionId },
          select: { id: true, title: true, price: true },
        });

        if (!session) {
          throw new AppError(ERROR_CODES.SESSION_PARTICIPANT_NOT_FOUND);
        }

        await tx.sessionParticipant.delete({
          where: { id: parseId(participantId) },
        });

        const decPrice = new Prisma.Decimal(session.price || 0);
        if (participant.role === 'PLAYER' && decPrice.gt(0)) {
          await walletService.refundFunds(participant.userId, session.id,
            decPrice, tx);

          await tx.session.update({
            where: { id: session.id },
            data: {
              heldAmount: { decrement: decPrice },
            },
          });
        }
      })
    }
  }
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (notificationService) {
      try {
        await notificationService.createNotification({
          eventKey:
            `session_declined:${participant.userId}:${session.id}`,
          type: 'SESSION_PARTICIPATION_DECLINED',
          severity: 'ERROR',
          category: 'session',
          title: 'Заявку відхилено',
          body: `Вашу заявку на сесію "${session.title || 'Нова сесія'}"
            відхилено.`,
          link: `/session/${session.id}`,
          recipientIds: [participant.userId],
          metadata: {
            sessionId: session.id,
            userId: participant.userId,
          },
        }, tx);
      } catch (err) {
        logger.error({ err, userId: participant.userId, sessionId:
          session.id }, 'Помилка відправки сповіщення про відхилення заявки');
      }
    }

    return {
      ...participant,
      status: 'DECLINED',
    };
  });
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		101

```

async function confirmGmParticipant({
  prisma,
  participantId,
  sessionId,
  notificationService,
}) {
  const participantIdInt = parseId(participantId);
  const sessionIdInt = parseId(sessionId);

  return prisma.$transaction(async (tx) => {
    const updatedParticipant = await tx.sessionParticipant.update({
      where: { id: participantIdInt },
      data: { status: 'CONFIRMED' },
      include: {
        user: {
          select: { id: true, username: true, displayName: true,
avatarUrl: true },
        },
      },
    });

    await tx.sessionParticipant.deleteMany({
      where: {
        sessionId: sessionIdInt,
        role: 'GM',
        status: 'PENDING',
        NOT: { id: participantIdInt },
      },
    });

    if (notificationService && updatedParticipant) {
      const session = await tx.session.findUnique({

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		102

```

        where: { id: sessionIdInt },
        select: { id: true, title: true },
    });
    if (session) {
        try {
            await notificationService.createNotification({
                eventKey:
                `session_confirmed:${updatedParticipant.userId}:${sessionIdInt}`,
                type: 'SESSION_PARTICIPATION_CONFIRMED',
                severity: 'SUCCESS',
                category: 'session',
                title: 'Ви додані до сесії',
                body: `Вас додано до сесії "${session.title || 'Нова
сесія'}".`,
                link: `/session/${sessionIdInt}`,
                recipientIds: [updatedParticipant.userId],
                metadata: {
                    sessionId: sessionIdInt,
                    participantId: updatedParticipant.id,
                    role: updatedParticipant.role,
                },
            }, tx);
        } catch (err) {
            logger.error({ err, userId: updatedParticipant.userId,
sessionId: sessionIdInt }, 'Помилка відправки сповіщення про
підтвердження GM');
        }
    }
}

return updatedParticipant;
});
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		103

```

async function updateParticipantStatusRecord({ prisma, participantId,
status, notificationService }) {

  return prisma.$transaction(async (tx) => {

    const participant = await tx.sessionParticipant.update({
      where: { id: parseId(participantId) },
      data: { status },
      include: {
        session: { select: { id: true, title: true } },
        user: { select: { id: true, username: true, displayName: true,
avatarUrl: true } },
      },
    });

    if (notificationService && status === 'CONFIRMED' &&
participant.session) {
      try {
        await notificationService.createNotification({
          eventKey:
`session_confirmed:${participant.userId}:${participant.sessionId}`,
          type: 'SESSION_PARTICIPATION_CONFIRMED',
          severity: 'SUCCESS',
          category: 'session',
          title: 'Ви додані до сесії',
          body: `Вас додано до сесії "${participant.session.title ||
'Нова сесія'}".`,
          link: `/session/${participant.sessionId}`,
          recipientIds: [participant.userId],
          metadata: {
            sessionId: participant.sessionId,
            participantId: participant.id,
            role: participant.role,
          },
        });
      }
    }
  });
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        }, tx);
    } catch (err) {
        logger.error({ err, userId: participant.userId, sessionId:
participant.sessionId }, 'Помилка відправки сповіщення про підтвердження
учасника');
    }
}

return participant;
});
}

```

```

function createSessionParticipantsService({
    prisma,
    AppError,
    ERROR_CODES,
    sessionQueryService,
    getSessionById,
    resolveSessionContext,
    datetimeHelpers,
    assertNoSessionTimeConflict,
    permissionHelpers,
    notificationService,
    walletService,
}) {
    const resolveGetSessionById = sessionQueryService?.getSessionById ||
getSessionById;

    const resolveSessionContextFn =
sessionQueryService?.resolveSessionContext || resolveSessionContext;

    const assertNoSessionTimeConflictFn = assertNoSessionTimeConflict ||
(async (userId, targetStart, targetDuration, options = {}) => {
        return datetimeHelpers._assertNoSessionTimeConflict(
            { prisma, AppError, ERROR_CODES },

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		105

```

        userId,
        targetStart,
        targetDuration,
        options
    );
});

if (
    typeof resolveGetSessionById !== 'function'
    || typeof resolveSessionContextFn !== 'function'
    || typeof assertNoSessionTimeConflictFn !== 'function'
) {
    throw new TypeError('Сервіс учасників сесії вимагає залежності
сервісу запитів сесії');
}

async function notifyManagersAboutJoinRequest(session, participant, tx)
{
    if (!notificationService) return;
    const existingPending = session.participants.filter(
        (p) => p.status === 'PENDING'
    ).length;
    const pendingCount = participant.status === 'PENDING' ?
existingPending + 1 : existingPending;

    const sessionTitle = session.title || 'Нова сесія';

    try {
        await notificationService.createNotification({
            eventKey: `session_join_requests:${session.id}`,
            type: 'SESSION_JOIN_REQUESTS_UPDATED',
            severity: 'INFO',

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		106

```

    category: 'session',
    title: `До сесії "${sessionTitle}" подано нові заявки`,
    body: `Очікує підтвердження: ${pendingCount}`,
    link: `/session/${session.id}`,
    audience: ['session_managers'],
    context: { sessionId: session.id },
    dedupeKey: `session:${session.id}:join_requests`,
    dedupeWindowMs: 10 * 60 * 1000, // 10 minutes
    metadata: {
      sessionId: session.id,
      pendingCount,
      requesterId: participant.userId,
      role: participant.role,
    },
  }, tx);
} catch (err) {
  logger.error({ err, sessionId: session.id }, 'Помилка відправки
сповіщення менеджерам про нову заявку');
}
}

async function notifyManagersAboutConfirmedJoin(session, participant,
tx) {
  if (!notificationService) return;

  const userName = participant.user?.displayName ||
participant.user?.username || 'Новий учасник';

  const roleLabel = participant.role === 'GM' ? 'гравець-майстер' :
'гравець';

  const sessionTitle = session.title || 'Нова сесія';

  try {
    await notificationService.createNotification({

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		107

```

        eventKey:
`session_confirmed_join:${session.id}:${participant.userId}`,
        type: 'SESSION_PARTICIPANT_JOINED',
        severity: 'INFO',
        category: 'session',
        title: `До сесії "${sessionTitle}" приєднався новий учасник`,
        body: `${userName} (${roleLabel}) приєднався до сесії.`,
        link: `/session/${session.id}`,
        audience: ['session_managers'],
        context: { sessionId: session.id },
        metadata: {
            sessionId: session.id,
            participantId: participant.id,
            userId: participant.userId,
            role: participant.role,
        },
    }, tx);
} catch (err) {
    logger.error({ err, userId: participant.userId, sessionId:
session.id }, 'Помилка відправки сповіщення менеджерам про приєднання
учасника');
}
}

return {
    async joinSession(sessionId, userId, options = {}) {
        const { role = 'PLAYER', shareToken = null } = options;

        const session = await resolveGetSessionById(sessionId, userId, {
shareToken });

        const normalizedRole = normalizeJoinRole(role, AppError,
ERROR_CODES);

        assertJoinableSession(session, AppError, ERROR_CODES);

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		108

```
    await assertUserNotJoinedYet({ prisma, sessionId, userId, AppError,
ERROR_CODES });
```

```
    await assertNoSessionTimeConflictFn(
        userId,
        session.date,
        session.duration,
        {
            excludeSessionId: session.id,
            conflictErrorCode: ERROR_CODES.SESSION_TIME_CONFLICT_PLAYER,
        }
    );
```

```
    assertPlayerCapacity({ normalizedRole, session, AppError,
ERROR_CODES });
```

```
    assertGmCanJoin({ normalizedRole, session, permissionHelpers,
AppError, ERROR_CODES });
```

```
    assertCampaignPlayerAccess({ normalizedRole, session, AppError,
ERROR_CODES });
```

```
const participant = await prisma.$transaction(async (tx) => {
    const newParticipant = await tx.sessionParticipant.create({
        data: {
            userId,
            sessionId: parseId(sessionId),
            role: normalizedRole,
            status: resolveJoinStatus({ normalizedRole, session }),
            isGuest: resolveGuestFlag({ normalizedRole, session }),
        },
        include: {
            user: { select: { id: true, username: true, displayName:
true, avatarUrl: true } },
        },
    });
```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		109

```

    });

    const decPrice = new Prisma.Decimal(session.price || 0);
    if (normalizedRole === 'PLAYER' && decPrice.gt(0)) {
        await walletService.reserveFunds(userId, session.id, decPrice,
tx);

        await tx.session.update({
            where: { id: session.id },
            data: {
                heldAmount: { increment: decPrice },
            },
        });
    }

    if (newParticipant.status === 'PENDING') {
        await notifyManagersAboutJoinRequest(session, newParticipant,
tx);
    } else if (newParticipant.status === 'CONFIRMED') {
        await notifyManagersAboutConfirmedJoin(session, newParticipant,
tx);
    }

    return newParticipant;
});

return participant;
},

async leaveSession(sessionId, userId) {
    const participant = await prisma.sessionParticipant.findUnique({
        where: {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				110
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        userId_sessionId: { userId, sessionId: parseId(sessionId) },
    },
    include: {
        session: {
            select: {
                id: true,
                ownerId: true,
                status: true,
                date: true,
                price: true,
            },
        },
    },
});

if (!participant) {
    throw new AppError(ERROR_CODES.SESSION_LEAVE_NOT_PARTICIPANT);
}

if (['FINISHED', 'CANCELED'].includes(participant.session.status))
{
    throw new
AppError(ERROR_CODES.SESSION_LEAVE_FINISHED_OR_CANCELED_FORBIDDEN);
}

if (participant.session.status === 'ACTIVE') {
    throw new AppError(ERROR_CODES.SESSION_LEAVE_ACTIVE_FORBIDDEN);
}

if (participant.role === 'GM' && participant.session.ownerId ===
userId) {
    throw new AppError(ERROR_CODES.SESSION_OWNER_GM_LEAVE_FORBIDDEN);
}

```

		Ярмоленко А.М.			ПІЗ.КР.Б – 121 – 26 – ПІЗ	Арк.
		Талавер О.В.				111
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    return await prisma.$transaction(async (tx) => {
      await tx.sessionParticipant.delete({
        where: {
          userId_sessionId: { userId, sessionId: parseId(sessionId) },
        },
      });

      const decPrice = new Prisma.Decimal(participant.session.price ||
0);

      if (participant.role === 'PLAYER' && (participant.status ===
'PENDING' || participant.status === 'CONFIRMED') && decPrice.gt(0)) {
        await walletService.refundFunds(userId, participant.session.id,
decPrice, tx);

        await tx.session.update({
          where: { id: participant.session.id },
          data: {
            heldAmount: { decrement: decPrice },
          },
        });
      }

      return participant;
    });
  },

  async getSessionParticipants(sessionId, userId = null) {
    const session = await resolveGetSessionById(sessionId, userId);
    return session.participants;
  },

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				112
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    async updateParticipantStatus(sessionId, participantId, requesterId,
status, options = {}) {
        const { preloadedSession = null } = options;

        const session = await resolveSessionContextFn(sessionId,
requesterId, preloadedSession);

        assertValidParticipantStatus(status, AppError, ERROR_CODES);
        assertSessionIsActiveForParticipantManagement(session, AppError,
ERROR_CODES);

        const participant = await findParticipantOrThrow({
            prisma,
            participantId,
            sessionId,
            AppError,
            ERROR_CODES,
        });

        if (status === 'DECLINED') {
            return declinePendingParticipant({
                prisma,
                participant,
                participantId,
                AppError,
                ERROR_CODES,
                notificationService,
                walletService,
            });
        }

        if (participant.role === 'GM') {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				113
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (!permissionHelpers._isSessionOwner(session, requesterId)) {
            throw new AppError(ERROR_CODES.SESSION_GM_REQUESTS_OWNER_ONLY);
        }

        if (status === 'CONFIRMED') {
            return confirmGmParticipant({
                prisma,
                participantId,
                sessionId,
            });
        }

        } else if (!permissionHelpers._canManageParticipants(session,
requesterId)) {
            throw new
AppError(ERROR_CODES.SESSION_PARTICIPANTS_MANAGE_OWNER_OR_CONFIRMED_GM_ON
LY);
        }

        return updateParticipantStatusRecord({ prisma, participantId,
status, notificationService });
    },
    async removeParticipant(sessionId, participantId, requesterId,
options = {}) {
        const { preloadedSession = null } = options;

        const session = await resolveSessionContextFn(sessionId,
requesterId, preloadedSession);

        if (['FINISHED', 'CANCELED'].includes(session.status)) {
            throw new
AppError(ERROR_CODES.SESSION_PARTICIPANT_REMOVAL_UNAVAILABLE);
        }

        const participant = await prisma.sessionParticipant.findUnique({
            where: { id: parseId(participantId) },
        });
    });

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				114
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (!participant?.sessionId || participant.sessionId !==
parseId(sessionId)) {
            throw new AppError(ERROR_CODES.SESSION_PARTICIPANT_NOT_FOUND);
        }

        const requesterIsSessionOwner =
permissionHelpers._isSessionOwner(session, requesterId);

        if (participant.userId === session.ownerId &&
!requesterIsSessionOwner) {
            throw new AppError(ERROR_CODES.SESSION_OWNER_REMOVAL_FORBIDDEN);
        }

        if (participant.role === 'GM') {
            const canManageGm = permissionHelpers._isSessionOwner(session,
requesterId)
                || permissionHelpers._isCampaignOwnerOverride(session,
requesterId);

            if (!canManageGm) {
                throw new AppError(ERROR_CODES.SESSION_GM_REMOVAL_OWNER_ONLY);
            }

            if (session.status !== 'PLANNED') {
                throw new AppError(ERROR_CODES.SESSION_NO_GM_KICK_ACTIVE);
            }

            if (participant.userId === session.ownerId) {
                throw new
AppError(ERROR_CODES.SESSION_OWNER_GM_ROLE_REMOVAL_FORBIDDEN);
            }

            } else if (!permissionHelpers._canManageParticipants(session,
requesterId)) {
                throw new
AppError(ERROR_CODES.SESSION_PARTICIPANTS_REMOVAL_OWNER_OR_CONFIRMED_GM_O
NLY);
            }
        }

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				115
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    await prisma.$transaction(async (tx) => {
      await tx.sessionParticipant.delete({
        where: { id: parseId(participantId) },
      });
      const decPrice = new Prisma.Decimal(session.price || 0);
      if (participant.role === 'PLAYER' && (participant.status ===
        'PENDING' || participant.status === 'CONFIRMED') && decPrice.gt(0)) {
        await walletService.refundFunds(participant.userId, session.id,
          decPrice, tx);
        await tx.session.update({
          where: { id: session.id },
          data: {
            heldAmount: { decrement: decPrice },
          },
        });
      }
    });
  },
  async kickGm(sessionId, requesterId, options = {}) {
    const { preloadedSession = null } = options;
    const session = await resolveSessionContextFn(sessionId,
      requesterId, preloadedSession);

    const canKickGm = permissionHelpers._isSessionOwner(session,
      requesterId)
      || permissionHelpers._isCampaignOwnerOverride(session,
        requesterId);

    if (!canKickGm) {
      throw new AppError(ERROR_CODES.SESSION_GM_KICK_OWNER_ONLY);
    }

    if (session.status !== 'PLANNED') {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		116

```

        throw new AppError(ERROR_CODES.SESSION_NO_GM_KICK_ACTIVE);
    }

    const confirmedGm = permissionHelpers._getConfirmedGm(session);

    if (!confirmedGm) {
        throw new AppError(ERROR_CODES.SESSION_NO_CONFIRMED_GM);
    }

    if (confirmedGm.userId === session.ownerId) {
        throw new AppError(ERROR_CODES.SESSION_OWNER_GM_KICK_FORBIDDEN);
    }

    await prisma.sessionParticipant.delete({
        where: { id: confirmedGm.id },
    });

    return { success: true };
},
};
}
module.exports = createSessionParticipantsService;

```

Лістинг access-rules.js:

```

const RESOURCE_TYPES = Object.freeze({
    CAMPAIGN: 'CAMPAIGN',
    SESSION: 'SESSION',
});

const SESSION_KINDS = Object.freeze({
    ONE_SHOT: 'ONE_SHOT',
    CAMPAIGN_SESSION: 'CAMPAIGN_SESSION',

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		117

```
});
```

```
const ACCESS_VISIBILITY = Object.freeze({  
  PUBLIC: 'PUBLIC',  
  PRIVATE: 'PRIVATE',  
  LINK_ONLY: 'LINK_ONLY',  
});
```

```
const VIEWER_TYPES = Object.freeze({  
  ANONYMOUS: 'ANONYMOUS',  
  AUTHENTICATED_OUTSIDER: 'AUTHENTICATED_OUTSIDER',  
  OWNER: 'OWNER',  
  CAMPAIGN_MEMBER: 'CAMPAIGN_MEMBER',  
  SESSION_PARTICIPANT: 'SESSION_PARTICIPANT',  
});
```

```
const JOIN_MODES = Object.freeze({  
  OPEN: 'OPEN',  
  REQUEST: 'REQUEST',  
  MEMBERS_ONLY: 'MEMBERS_ONLY',  
  NOT_APPLICABLE: 'NOT_APPLICABLE',  
});
```

```
const CAMPAIGN_ACCESS_RULES = Object.freeze({  
  [ACCESS_VISIBILITY.PUBLIC]: Object.freeze({  
    outsiderDiscoverable: true,  
    outsiderCanOpenDirectly: true,  
    outsiderNeedsShareToken: false,  
    joinMode: JOIN_MODES.REQUEST,  
  }),  
  [ACCESS_VISIBILITY.LINK_ONLY]: Object.freeze({
```

		Ярмоленко А.М.			ПІЗ.КР.Б – 121 – 26 – ПІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		118

```

    outsiderDiscoverable: false,
    outsiderCanOpenDirectly: false,
    outsiderNeedsShareToken: true,
    joinMode: JOIN_MODES.REQUEST,
  )),
});

const SESSION_ACCESS_RULES = Object.freeze({
  [SESSION_KINDS.ONE_SHOT]: Object.freeze({
    [ACCESS_VISIBILITY.PUBLIC]: Object.freeze({
      outsiderDiscoverable: true,
      outsiderCanOpenDirectly: true,
      outsiderNeedsShareToken: false,
      joinMode: JOIN_MODES.OPEN,
    )),
    [ACCESS_VISIBILITY.PRIVATE]: Object.freeze({
      outsiderDiscoverable: true,
      outsiderCanOpenDirectly: true,
      outsiderNeedsShareToken: false,
      joinMode: JOIN_MODES.REQUEST,
    )),
    [ACCESS_VISIBILITY.LINK_ONLY]: Object.freeze({
      outsiderDiscoverable: false,
      outsiderCanOpenDirectly: false,
      outsiderNeedsShareToken: true,
      joinMode: JOIN_MODES.REQUEST,
    )),
  )),
  [SESSION_KINDS.CAMPAIGN_SESSION]: Object.freeze({
    [ACCESS_VISIBILITY.PUBLIC]: Object.freeze({
      outsiderDiscoverable: true,

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		119

```

        outsiderCanOpenDirectly: true,
        outsiderNeedsShareToken: false,
        joinMode: JOIN_MODES.REQUEST,
    )),
    [ACCESS_VISIBILITY.PRIVATE]: Object.freeze({
        outsiderDiscoverable: false,
        outsiderCanOpenDirectly: false,
        outsiderNeedsShareToken: false,
        joinMode: JOIN_MODES.MEMBERS_ONLY,
    )),
  )),
});

function getViewerType(context = {}) {
  if (context.isOwner) return VIEWER_TYPES.OWNER;
  if (context.isParticipant) return VIEWER_TYPES.SESSION_PARTICIPANT;
  if (context.isMember || context.isCampaignMember) return
VIEWER_TYPES.CAMPAIGN_MEMBER;
  if (context.userId) return VIEWER_TYPES.AUTHENTICATED_OUTSIDER;
  return VIEWER_TYPES.ANONYMOUS;
}

function isEntitledCampaignViewer(context = {}) {
  return Boolean(context.isOwner || context.isMember ||
context.isCampaignMember);
}

function isEntitledSessionViewer(context = {}) {
  return Boolean(context.isOwner || context.isParticipant ||
context.isCampaignMember);
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				120
Змн.	Арк.	№ докум.	Підпис	Дата		

```

function isOutsider(context = {}) {
    return !isEntitledCampaignViewer(context) &&
    !isEntitledSessionViewer(context);
}

function getCampaignRule(visibility) {
    return CAMPAIGN_ACCESS_RULES[visibility] || null;
}

function getSessionRule({ visibility, isCampaignSession }) {
    const kind = isCampaignSession ? SESSION_KINDS.CAMPAIGN_SESSION :
    SESSION_KINDS.ONE_SHOT;
    return SESSION_ACCESS_RULES[kind]?.[visibility] || null;
}

function canDiscoverCampaign(context = {}) {
    const rule = getCampaignRule(context.visibility);

    if (!rule) return false;
    if (isEntitledCampaignViewer(context)) return true;

    return rule.outsiderDiscoverable;
}

function canOpenCampaign(context = {}) {
    const rule = getCampaignRule(context.visibility);

    if (!rule) return false;
    if (isEntitledCampaignViewer(context)) return true;
    if (context.isPendingJoinRequester) return true;
    if (rule.outsiderCanOpenDirectly) return true;
    if (rule.outsiderNeedsShareToken) {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				121
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        return Boolean(context.userId && context.hasValidShareToken);
    };

    return false;
}

function campaignRequiresShareTokenForOutsider(context = {}) {
    const rule = getCampaignRule(context.visibility);
    return Boolean(rule?.outsiderNeedsShareToken);
}

function getCampaignJoinMode(context = {}) {
    const rule = getCampaignRule(context.visibility);
    return rule ? rule.joinMode : JOIN_MODES.NOT_APPLICABLE;
}

function canManageCampaign(context = {}) {
    return Boolean(context.isOwner);
}

function canDiscoverSession(context = {}) {
    if (!context) return false;
    const rule = getSessionRule({
        visibility: context.visibility,
        isCampaignSession: Boolean(context.isCampaignSession),
    });

    if (!rule) return false;
    if (isEntitledSessionViewer(context)) return true;

    return rule.outsiderDiscoverable;
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				122
Змн.	Арк.	№ докум.	Підпис	Дата		

```

}

function canOpenSession(context = {}) {
  if (!context) return false;
  const rule = getSessionRule({
    visibility: context.visibility,
    isCampaignSession: Boolean(context.isCampaignSession),
  });

  if (!rule) return false;
  if (isEntitledSessionViewer(context)) return true;
  if (context.isPendingParticipant) return true;
  if (context.isCampaignSession && context.hasValidCampaignShareToken)
return true;
  if (rule.outsiderCanOpenDirectly) return true;
  if (rule.outsiderNeedsShareToken) {
    return Boolean(context.userId && context.hasValidShareToken);
  }

  return false;
}

function sessionRequiresShareTokenForOutsider(context = {}) {
  if (!context) return false;
  const rule = getSessionRule({
    visibility: context.visibility,
    isCampaignSession: Boolean(context.isCampaignSession),
  });

  return Boolean(rule?.outsiderNeedsShareToken);
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				123
Змн.	Арк.	№ докум.	Підпис	Дата		

```

function getSessionJoinMode(context = {}) {
  if (!context) return JOIN_MODES.NOT_APPLICABLE;
  const rule = getSessionRule({
    visibility: context.visibility,
    isCampaignSession: Boolean(context.isCampaignSession),
  });

  return rule ? rule.joinMode : JOIN_MODES.NOT_APPLICABLE;
}

function canManageSession(context = {}) {
  if (!context) return false;
  return Boolean(context.isOwner || context.isConfirmedGm);
}

function canManageSessionParticipants(context = {}) {
  if (!context) return false;
  return Boolean(context.isOwner || context.isConfirmedGm);
}

module.exports = {
  ACCESS_VISIBILITY,
  CAMPAIGN_ACCESS_RULES,
  JOIN_MODES,
  RESOURCE_TYPES,
  SESSION_ACCESS_RULES,
  SESSION_KINDS,
  VIEWER_TYPES,
  canDiscoverCampaign,
  canDiscoverSession,
  canManageCampaign,

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				124
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    canManageSession,
    canManageSessionParticipants,
    canOpenCampaign,
    canOpenSession,
    campaignRequiresShareTokenForOutsider,
    getCampaignJoinMode,
    getCampaignRule,
    getSessionJoinMode,
    getSessionRule,
    getViewerType,
    isOutsider,
    sessionRequiresShareTokenForOutsider,
};

```

Лістинг notification.service.js:

```

const { prisma: defaultPrisma } = require('../lib/prisma');
const { AppError, ERROR_CODES } = require('../constants/errors');

const defaultNotificationRecipientResolver =
  require('../notification/notification-recipient-resolver');

const defaultSseService = require('../notification/notification-
sse.service');

```

```

class NotificationService {
  constructor(deps = {}) {
    this.prisma = deps.prisma || defaultPrisma;
    this.recipientResolver = deps.recipientResolver ||
defaultNotificationRecipientResolver;
    this.sseService = deps.sseService || defaultSseService;
  }

  async createNotification(input, externalTx = null) {
    const {
      eventKey,
      type,

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				125
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    severity,
    category,
    title,
    body,
    link,
    metadata,
    dedupeKey,
    aggregationKey,
    expiresAt,
    source,
    recipientIds,
    audience,
    context,
    dedupeWindowMs,
} = input;

```

```

const resolvedRecipientIds = await this.resolveRecipientIds({
  recipientIds,
  audience,
  context,
});

```

```

if (resolvedRecipientIds.length === 0) {
  return null;
}

```

```

const execute = async (tx) => {
  const existingNotification = await
this._findExistingNotificationForDedupe(tx, {
    dedupeKey,
    dedupeWindowMs,
  });

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				126
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (existingNotification) {
        const attachedRecipientIds = await this._attachRecipients(tx,
existingNotification.id, resolvedRecipientIds);
        return {
            notification: existingNotification,
            attachedRecipientIds,
        };
    }

    const notification = await tx.notification.create({
        data: {
            eventKey,
            type,
            severity,
            category,
            title,
            body,
            link,
            metadata: metadata || {},
            dedupeKey,
            aggregationKey,
            expiresAt,
            source,
        },
    });

    const attachedRecipientIds = await this._attachRecipients(tx,
notification.id, resolvedRecipientIds);

    await this._createTelegramOutboxEvents(tx, notification,
attachedRecipientIds);

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				127
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return {
        notification,
        attachedRecipientIds,
    };
};

let result;
if (externalTx) {
    result = await execute(externalTx);
} else {
    result = await this.prisma.$transaction(execute);
}

this.pushToConnectedUsers(result.notification,
result.attachedRecipientIds);

return result.notification;
}

async resolveRecipientIds(input = {}) {
    const { recipientIds = [], audience, context = {} } = input;
    const resolvedIds = new Set((recipientIds || []).filter(Boolean));

    let audiences = [];
    if (Array.isArray(audience)) {
        audiences = audience;
    } else if (audience) {
        audiences = [audience];
    }

    for (const audienceKey of audiences) {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				128
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    const audienceRecipientIds = await
this.recipientResolver.resolve(audienceKey, context);

    audienceRecipientIds.forEach((userId) => {
        if (userId) {
            resolvedIds.add(userId);
        }
    });
}

if (context.excludeUserId) {
    const excludedId = Number.parseInt(context.excludeUserId, 10);
    resolvedIds.delete(context.excludeUserId);
    if (Number.isInteger(excludedId)) {
        resolvedIds.delete(excludedId);
    }
}

return [...resolvedIds];
}

async _findExistingNotificationForDedupe(tx, options = {}) {
    const { dedupeKey, dedupeWindowMs } = options;
    if (!dedupeKey) {
        return null;
    }

    const where = { dedupeKey };
    if (Number.isFinite(dedupeWindowMs) && dedupeWindowMs > 0) {
        where.createdAt = {
            gte: new Date(Date.now() - dedupeWindowMs),
        };
    }
}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				129
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return tx.notification.findFirst({
  where,
  orderBy: { createdAt: 'desc' },
});
}

async _attachRecipients(tx, notificationId, recipientIds) {
  const uniqueRecipientIds = [...new Set((recipientIds ||
[]).filter(Boolean))];
  if (uniqueRecipientIds.length === 0) {
    return [];
  }

  const existingRecipients = await tx.notificationRecipient.findMany({
    where: {
      notificationId,
      userId: { in: uniqueRecipientIds },
    },
    select: { userId: true },
  });

  const existingRecipientIds = new
Set(existingRecipients.map((recipient) => recipient.userId));

  const newRecipientIds = uniqueRecipientIds.filter((userId) =>
!existingRecipientIds.has(userId));

  if (newRecipientIds.length === 0) {
    return [];
  }

  await tx.notificationRecipient.createMany({

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				130
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        data: newRecipientIds.map((userId) => ({
            notificationId,
            userId,
            status: 'ACTIVE',
        })),
        skipDuplicates: true,
    });

    return newRecipientIds;
}

async _createTelegramOutboxEvents(tx, notification, recipientIds) {
    if (!recipientIds || recipientIds.length === 0) return;

    const users = await tx.user.findMany({
        where: {
            id: { in: recipientIds },
            telegramChatId: { not: null },
        },
        select: {
            id: true,
            telegramChatId: true,
            notificationPreference: {
                select: { enabledChannels: true },
            },
        },
    });

    const outboxPayloads = [];

    for (const user of users) {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				131
Змн.	Арк.	№ докум.	Підпис	Дата		

```

let channels = [];

if (user.notificationPreference?.enabledChannels) {
  channels =
Array.isArray(user.notificationPreference.enabledChannels)
  ? user.notificationPreference.enabledChannels
  : [];
}

if (channels.includes('TELEGRAM')) {
  outboxPayloads.push({
    notificationId: notification.id,
    type: 'TELEGRAM_NOTIFICATION',
    payload: {
      chatId: user.telegramChatId,
      title: notification.title,
      body: notification.body,
      severity: notification.severity,
      link: notification.link,
    },
    status: 'PENDING',
    attempts: 0,
    maxAttempts: 3,
  });
}

}

if (outboxPayloads.length > 0) {
  await tx.outboxEvent.createMany({
    data: outboxPayloads,
  });
}

}

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		132

```

async listNotificationsForUser(userId, options = {}) {
  const { status, limit = 20, offset = 0 } = options;

  const where = { userId };
  if (status) {
    where.status = status;
  }

  const [recipients, total] = await Promise.all([
    this.prisma.notificationRecipient.findMany({
      where,
      include: {
        notification: true,
      },
      orderBy: { createdAt: 'desc' },
      take: limit,
      skip: offset,
    }),
    this.prisma.notificationRecipient.count({ where }),
  ]);

  const notifications = recipients.map((r) => ({
    id: r.notification.id,
    recipientId: r.id,
    eventKey: r.notification.eventKey,
    type: r.notification.type,
    severity: r.notification.severity,
    category: r.notification.category,
    title: r.notification.title,
    body: r.notification.body,
  }));

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				133
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    link: r.notification.link,
    metadata: r.notification.metadata,
    status: r.status,
    readAt: r.readAt,
    archivedAt: r.archivedAt,
    createdAt: r.createdAt,
  )));

  return {
    notifications,
    pagination: {
      total,
      limit,
      offset,
      hasMore: offset + notifications.length < total,
    },
  };
}

```

```

async getUnreadCount(userId) {
  return this.prisma.notificationRecipient.count({
    where: {
      userId,
      status: 'ACTIVE',
    },
  });
}

```

```

async markAsRead(userId, notificationId) {
  const recipient = await this.prisma.notificationRecipient.findFirst({
    where: {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				134
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        userId,
        notificationId,
    },
});

if (!recipient) {
    throw new AppError(ERROR_CODES.NOTIFICATION_NOT_FOUND);
}

if (recipient.status === 'ARCHIVED') {
    return recipient;
}

const now = new Date();

return this.prisma.notificationRecipient.update({
    where: { id: recipient.id },
    data: {
        status: 'ARCHIVED',
        readAt: recipient.readAt || now,
        archivedAt: recipient.archivedAt || now,
    },
});
}

async markManyAsRead(userId, notificationIds) {
    const recipients = await this.prisma.notificationRecipient.findMany({
        where: {
            userId,
            notificationId: { in: notificationIds },
            status: 'ACTIVE',

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		135

```

    },
  });

  if (recipients.length === 0) {
    return 0;
  }

  const result = await this.prisma.notificationRecipient.updateMany({
    where: {
      id: { in: recipients.map((r) => r.id) },
    },
    data: {
      status: 'ARCHIVED',
      readAt: new Date(),
      archivedAt: new Date(),
    },
  });

  return result.count;
}

pushToConnectedUsers(notification, recipientIds) {
  if (!notification || !recipientIds || recipientIds.length === 0) {
    return;
  }

  const payload = {
    id: notification.id,
    eventKey: notification.eventKey,
    type: notification.type,
    severity: notification.severity,
  }

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		136

```

        category: notification.category,
        title: notification.title,
        body: notification.body,
        link: notification.link,
        metadata: notification.metadata,
        createdAt: new Date().toISOString(),
        status: 'ACTIVE',
    };

    this.sseService.pushToUsers(recipientIds, payload);
}
}

function createNotificationService(deps = {}) {
    return new NotificationService(deps);
}

module.exports = new NotificationService();
module.exports.NotificationService = NotificationService;
module.exports.createNotificationService = createNotificationService;

```

Лістинг token.helper.js:

```

const crypto = require('node:crypto');

function resolveShareTokenSecret() {
    const secret = process.env.SHARE_TOKEN_SECRET;

    if (secret) {
        return secret;
    }

    if (typeof process.env.NODE_TEST_CONTEXT === 'string') {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				137
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return 'test-share-token-secret';
}

return null;
}

function getShareTokenEncryptionKey() {
    const shareTokenSecret = resolveShareTokenSecret();

    if (!shareTokenSecret) {
        throw new Error('SHARE_TOKEN_SECRET must be configured for share
token encryption');
    }

    return crypto.createHash('sha256').update(shareTokenSecret).digest();
}

function hashToken(token) {
    return crypto.createHash('sha256').update(token).digest('hex');
}

function createRawAndHashedToken(bytes = 32) {
    const rawToken = crypto.randomBytes(bytes).toString('hex');
    return {
        rawToken,
        tokenHash: hashToken(rawToken),
    };
}

function createRawAndHashedShareToken(bytes = 24) {
    const rawToken = crypto.randomBytes(bytes).toString('base64url');
    return {

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				138
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    rawToken,
    tokenHash: hashToken(rawToken),
  };
}

function encryptShareToken(rawToken) {
  const key = getShareTokenEncryptionKey();
  const iv = crypto.randomBytes(12);
  const cipher = crypto.createCipheriv('aes-256-gcm', key, iv);
  const ciphertext = Buffer.concat([
    cipher.update(String(rawToken), 'utf8'),
    cipher.final(),
  ]);
  const authTag = cipher.getAuthTag();

  return [
    iv.toString('base64url'),
    authTag.toString('base64url'),
    ciphertext.toString('base64url'),
  ].join('.');
}

function decryptShareToken(encryptedToken) {
  const serialized = String(encryptedToken || '').trim();

  if (!serialized) {
    throw new Error('Encrypted share token is empty');
  }

  const [ivPart, authTagPart, ciphertextPart] = serialized.split('.');
  if (!ivPart || !authTagPart || !ciphertextPart) {

```

		Ярмоленко А.М.			ПІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		139

```

        throw new Error('Encrypted share token payload is malformed');
    }

    const key = getShareTokenEncryptionKey();
    const decipher = crypto.createDecipheriv(
        'aes-256-gcm',
        key,
        Buffer.from(ivPart, 'base64url')
    );
    decipher.setAuthTag(Buffer.from(authTagPart, 'base64url'));

    const plaintext = Buffer.concat([
        decipher.update(Buffer.from(ciphertextPart, 'base64url')),
        decipher.final(),
    ]);

    return plaintext.toString('utf8');
}

function createRawEncryptedAndHashedShareToken(bytes = 24) {
    const { rawToken, tokenHash } = createRawAndHashedShareToken(bytes);
    return {
        rawToken,
        tokenHash,
        tokenEncrypted: encryptShareToken(rawToken),
    };
}

module.exports = {
    hashToken,
    createRawAndHashedToken,

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		140

```

createRawAndHashedShareToken,
encryptShareToken,
decryptShareToken,
createRawEncryptedAndHashedShareToken,
};

```

		Ярмоленко А.М.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Талавер О.В.				141
Змн.	Арк.	№ докум.	Підпис	Дата		