**Міністерство освіти і науки України**
**Тернопільський національний технічний університет імені Івана Пулюя**

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету )
Кафедра програмної інженерії
(повна назва кафедри)

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

«Бакалавр»
(назва освітнього ступеня)

на тему: Проєктування програмного забезпечення на основі мультисервісної архітектури для інтелектуального навчального контенту

Виконав(ла): студент(ка) IV курсу, групи СП – 41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Карпець Д.Ю. (прізвище та ініціали)

Керівник (підпис) Петрик М.Р. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) Яцишин В.В. (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
**Тернопільський національний технічний університет імені Івана Пулюя**

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

# ЗАВДАННЯ
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту _____

1. Тема роботи Проєктування програмного забезпечення на основі мультисервісної архітектури для інтелектуального навчального контенту

Керівник роботи Петрик Михайло Романович, доктор фізико-математичних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від « 06 » квітня 2026 року № _____

2. Термін подання студентом роботи 22.06.2026

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз вимог та огляд предметної області.
2. Проєктування та реалізація.
3. Тестування, впровадження та підтримка.
4. Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність, мета, задачі дослідження
3. Існуючі технології реалізації подібних систем.
4. Функціональні та нефункціональні вимоги .5. Загальна архітектура системи.
6. Варіанти використання. 7. Компоненти програми для налаштування параметрів системи.
8. Програмні засоби та технології. 9. Інтерфейси реалізації застосунку..
10. Тестування. 11. Висновки по роботі. 12. Слайди презентації.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Безпека життєдіяльності, основи охорони праці | Мариненко Сергій Юрійович доц. каф. МТ | | |
| | | | |

7. Дата видачі завдання ___6 квітня_____ 2026 р.

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів кваліфікаційної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | *Розробка технічного завдання* | *6.04 – 12.04* | |
| 2. | *Робота над першим розділом «Огляд предметної області»* | *13.04 – 03.05* | |
| 3. | *Робота над другим розділом «Проєктування та реалізація»* | *03.05 – 18.05* | |
| 4. | *Робота над третім розділом «Тестування, впровадження та підтримка»* | *19.05 – 22.05* | |
| 5. | *Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»* | *23.05 – 24.05* | |
| 6. | *Оформлення пояснювальної записки і графічного матеріалу* | *25.05 – 7.06* | |
| 7. | *Перевірка на академічний плагіат, перевірка керівником та консультантами* | *8.06 – 14.06* | |
| 8. | *Попередній захист кваліфікаційної роботи бакалавра* | *15.06 – 21.06* | |
| 9. | *Захист кваліфікаційної роботи бакалавра* | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Карпець Д. Ю. Проєктування програмного забезпечення на основі мультисервісної архітектури для інтелектуального навчального контенту : робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 - інженерія програмного забезпечення / наук. кер. М. Р. Петрик. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2026. 66 с. // С. – 66, табл. – 2, рис. – 8, бібліогр. – 24, додат. – 2 ,

Ключові слова: інженерія програмного забезпечення, веб-сервіс, онлайн-гра, React, Node.js, WebSocket, багатокористувацька система.

Мета роботи полягає у розробці веб-платформи, призначеної для інтеграції та запуску різноманітних інтелектуальних ігор.

У першому розділі описано особливості розробки веб-платформ, розглянуто проблеми тестування, висвітлено методи автоматизації та проаналізовано сучасні підходи до діагностики.

У другому розділі досліджено вимоги до системи, подано архітектуру підсистеми, обґрунтовано вибір технологій та моделювання її компонентів.

У третьому розділі описано реалізацію системи, проаналізовано її основні модулі, проведено тестування працездатності і функціоналу.

У четвертому розділі розглядається питання забезпечення безпечної експлуатації обладнання.

Об’єктом дослідження є процедура проєктування та розгортання веб-сервісу як інтерактивного простору для спільного дозвілля користувачів за іграми. Предметом дослідження виступають інструменти, математичні моделі та хмарні технології, що застосовуються для створення ігрового веб-сервісу. Методи дослідження охоплюють порівняльний аналіз аналогів на ринку, проєктування системної архітектури, верифікацію функціональних елементів, а також налаштування процесів автоматизованого розгортання (CI/CD).

# ABSTRACT

Denys Karpets. Development of a gaming web servise using cloud technologies : bachelor thesis : specialty 121 "Software Engineering": Ternopil Ivan Puluj National Technical University, 2026, 66p. // Pages – 66i, tables – 2, figures – 8, references – 24, appendices – 2.

Keywords: software engineering, web service, online game, React, Node.js, WebSocket, multiplayer game.

The objective of the study is to develop a web service designed for integrating and hosting various intellectual games.

The main task is to develop an web service that provides users room system for games to be played. To achieve the goal, component architecture methods, WebSocket technologies, as well as frontend (React) and backend (Node.js, NestJS) tools were used. The system was developed taking into account modern requirements for interactivity, scalability and ease of use. The work analyzed existing analogues, designed the architecture, implemented and tested the application. Attention was paid to technical and operational characteristics, in particular, speed and support for many users. The results of the work can be implemented both privately and on public platforms. The project has social significance as a tool for popularizing Ukrainian content and developing online communication.

The object of the study is the process of designing and deploying a web service that functions as an interactive platform for users to gather and play various games. The subject of the study encompasses the methods, models, and cloud-based technologies utilized in the development of the gaming web service. The research methods include a comparative analysis of competing systems, architectural modeling, functional component testing, and automated CI/CD deployment pipelines.

# ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) — програмний інтерфейс, що забезпечує взаємодію між окремими програмними компонентами та сервісами.

CRUD (Create, Read, Update, Delete) — набір базових операцій для створення, читання, оновлення та видалення даних.

HTTP (HyperText Transfer Protocol) — протокол передачі даних між клієнтом і сервером у мережі Інтернет.

HTTPS (HyperText Transfer Protocol Secure) — захищена версія протоколу HTTP, що використовує шифрування даних.

JSON (JavaScript Object Notation) — текстовий формат представлення та обміну структурованими даними.

JWT (JSON Web Token) — формат токена для безпечної передачі інформації між клієнтом і сервером.

MongoDB — документоорієнтована система керування базами даних класу NoSQL.

Node.js — серверне середовище виконання JavaScript, побудоване на рушії V8.

NoSQL (Not Only SQL) — клас нереляційних систем керування базами даних, орієнтованих на гнучке зберігання інформації.

React — JavaScript-бібліотека для створення інтерактивних користувацьких інтерфейсів.

REST API (Representational State Transfer Application Programming Interface) — архітектурний стиль побудови програмних інтерфейсів на основі HTTP-запитів.

Socket.IO — бібліотека для організації двостороннього обміну даними між клієнтом і сервером у режимі реального часу.

TypeScript — типізована надбудова над мовою JavaScript, що забезпечує додаткові засоби контролю коректності коду.

UI (User Interface) — сукупність засобів взаємодії користувача із програмною системою.

UX (User Experience) — загальне враження та досвід користувача під час роботи із програмним забезпеченням.

URL (Uniform Resource Locator) — уніфікований покажчик ресурсу в мережі Інтернет.

WebSocket — мережевий протокол, що забезпечує постійне двостороннє з'єднання між клієнтом і сервером.

SPA (Single Page Application) — веб-застосунок, у якому взаємодія відбувається без повного перезавантаження сторінки.

Кімната (Room) — логічне середовище взаємодії користувачів у межах системи, призначене для організації спільної роботи або ігрового процесу.

Хост (Host) — користувач, який створив кімнату та володіє правами керування її параметрами.

Стан гри (Game State) — набір даних, що описує поточний перебіг ігрової сесії та використовується для синхронізації між учасниками.

Зберігання в оперативній пам'яті (In-memory storage) — спосіб зберігання даних безпосередньо в пам'яті сервера протягом часу роботи застосунку.

Мультисервісна архітектура (Multiservice Architecture) — підхід до побудови програмного забезпечення, за якого окремі функціональні модулі системи реалізуються у вигляді незалежних сервісів або підсистем із чітко визначеними зонами відповідальності.

Комунікація в реальному часі (Real-Time Communication) — обмін даними між учасниками системи з мінімальними затримками та миттєвою синхронізацією стану.

Ігровий модуль — незалежний програмний компонент, що реалізує логіку окремої гри та взаємодіє із системою кімнат через визначений набір подій і правил.

Система кімнат — базова підсистема застосунку, яка забезпечує створення, налаштування та керування кімнатами незалежно від конкретної реалізації ігор.

Веб-застосунок — програмне забезпечення, доступ до якого здійснюється через веб-браузер із використанням мережевих технологій.

**ЗМІСТ**

# ВСТУП

Під час стрімкого розвитку цифрових інформаційних технологій та поширення форм навчання, інакших від очних, особливої актуальності набувають програмні рішення, які забезпечують інтерактивність між користувачами у режимі реального часу. Різні сучасні освітні платформи все частіше й частіше використовують різноманітні елементи гейміфікації для підвищення зацікавленості їхніх користувачів, покращення засвоєння матеріалу та розвитку навичок для командної роботи.

Існуючі програмні рішення дуже часто бувають орієнтовані на реалізацію окремих навчальних сценаріїв або конкретних ігор, що сильно обмежує можливості для їх подальшого розвитку та для їх адаптації до нових інших потреб. Крім того, одноманітний підхід до проєктування ускладнює масштабування та розширення функціональності та інтеграцію інших нових модулів. У зв’язку з цим виникає потреба для створення програмного забезпечення, побудованого на основі мультисервісної архітектури, що дозволить реалізувати дійсно універсальну систему для керування ігровими сесіями та забезпечить можливість для підключення різноманітного інтелектуального контенту.

Сучасні вебтехнології вже надають досить широкий спектр інструментів для створення подібних систем. Використання фреймворку React для розробки клієнтської частини дозволить створити динамічний користувацький інтерфейс із високим рівнем інтерактивності. Серверна частина, яка реалізована за допомогою перевірених Node.js та Express, забезпечує надійну обробку запитів користувачів, а застосування технології Socket.IO згори дає змогу організувати обмін даними між клієнтом і сервером у режимі реального часу. Для збереження даних використовується система керування базами даних MongoDB, що забезпечить гнучку та надійну роботу зі структурованою інформацією.

Дана кваліфікаційна робота є об’єктом дослідження і її метою є процес проєктування та розробки різноманітних програмних систем для організації багатокористувацької взаємодії з інтелектуальним навчальним контентом.

Предметом дослідження є архітектурні, програмні та технологічні рішення, що забезпечують функціонування системи кімнат, синхронізацію стану застосунку між користувачами та інтеграцію окремих ігрових модулів у єдине програмне середовище.

Метою роботи стало проєктування й розробка програмного забезпечення на основі мультисервісної архітектури для роботи з інтелектуальним навчальним контентом, що забезпечить створення й керування багатокористувацькими ігровими сесіями у реальному часі. Також особливу увагу було приділено розробці універсальної системи кімнат, що дозволить використовувати різні ігрові механіки без необхідності для того, щоб змінити базову архітектуру застосунку.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати предметної області й існуючі підходи до реалізації багатокористувацьких веб-сервісів;
– визначити вимоги до програмної системи;
– спроєктувати архітектуру застосунку;
– реалізувати механізм створення керування кімнатами користувачів;
– забезпечити обмін даними між учасниками в режимі реального часу;
– реалізувати інтеграцію ігрових модулів у межах єдиної системи;

Практична цінність роботи полягає у створенні програмної платформи, яка може використовуватися як основа для розробки та інтеграції нових навчальних або розважальних ігор. Розроблена архітектура забезпечує гнучкість розширення функціональності та спрощує підтримку програмного забезпечення.

Структура кваліфікаційної роботи складається зі вступу, теоретичного розділу, присвяченого аналізу предметної області та використаних технологій, практичного розділу, у якому розглядаються питання проєктування та реалізації програмного забезпечення, висновків і списку використаних джерел.

# 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Створення сучасних програмних систем для роботи з інтелектуальним навчальним контентом має мати врахування як особливостей організації взаємодії між користувачами, так і можливостей сучасних цифрових технологій. Особливої актуальності набувають веборієнтовані програмні рішення, що забезпечують багатокористувацьку взаємодію в режимі реального часу та дозволяють інтегрувати різноманітні освітні й ігрові механіки в межах єдиного програмного середовища.

Останніми роками спостерігається, що освітні платформи все частіше й частіше розвиваються, використовуючи різноманітні елементи гейміфікації для підвищення зацікавленості їхніх користувачів. Подібні системи поєднують елементи командної взаємодії, змагання та виконання інтелектуальних завдань, що позитивно впливає на мотивацію користувачів та ефективність засвоєння інформації. Водночас більшість існуючих рішень орієнтовані на конкретний тип навчального контенту або окрему гру, що обмежує можливості їх подальшого розширення та адаптації.

Одним із напрямів розвитку таких систем, з дуже великими перспективами, є застосування мультисервісної архітектури, що дозволить розділити функціональність на незалежні програмні модулі та забезпечити можливість подальшого масштабування системи. Такий підхід спростить підтримку програмного забезпечення, підвищить його гнучкість й дозволить інтегрувати нові сервіси без суттєвого впливу на вже реалізовані компоненти.

У даному розділі було виконано аналіз предметної області, розглянуто існуючі програмні рішення, що можуть бути використані як аналоги розроблюваної системи, а також досліджено сучасні технології, придатні для реалізації багатокористувацького веб-сервісу з підтримкою інтерактивного навчального контенту в режимі реального часу.

## 1.1 Огляд аналогів

Сучасний ринок програмного забезпечення вже пропонує значну кількість платформ, призначених для організації багатокористувацької взаємодії, проведення інтелектуальних ігор та/або реалізації навчального контенту. Такі рішення відрізняються за функціональними можливостями, архітектурними підходами та способами взаємодії користувачів. Аналіз вже існуючих аналогів дозволить визначити переваги та недоліки сучасних систем, а також сформувати вимоги до програмного забезпечення, що розробилось в межах даної роботи.

### 1.1.1 Веб-додаток Board Game Arena

Board Game Arena є одним із найбільших представників платформ для багатокористувацьких настільних ігор. Дана система надає користувачам доступ до великої кількості настільних ігор через веб-інтерфейс без необхідності встановлення якогось додаткового програмного забезпечення. Платформа підтримує: створення ігрових кімнат, автоматизацію правил гри та взаємодію між учасниками у реальному часі. Основною перевагою рішення є велика кількість доступних ігор та розвинена система організації ігрових сесій. Але водночас платформа орієнтована переважно на настільні ігри та не передбачає використання як універсального середовища для розміщення саме навчального контенту.

### 1.1.2 Веб-додаток Kahoot!

Kahoot! широко використовується в освітньому середовищі для проведення інтерактивних вікторин і тестувань. Дана платформа дозволяє організовувати навчальні сесії, до яких користувачі можуть приєднуватися за спеціальним кодом. Основними перевагами Kahoot! є простота використання, зручний інтерфейс та висока залученість учасників. З іншої сторони, функціональність системи обмежується переважно тестовими завданнями та не надає можливостей для реалізації складніших ігрових сценаріїв або інтеграції незалежних модулів.

### 1.1.3 Веб-додаток Quizizz

Quizizz це платформа, що поєднує можливості онлайн-тестування та елементи гейміфікації. Її система дозволяє проводити індивідуальні та групові навчальні активності, автоматично оцінити результати та сформувати статистичні звіти. На відміну від традиційних систем тестування, Quizizz активно використовує механізми змагання між учасниками, що позитивно впливає на рівень їх зацікавленості. Однак архітектура платформи орієнтована насамперед на роботу з тестовими завданнями і теж не підтримує створення довільних інтерактивних ігор із власними правилами.

### 1.1.4 Веб-додаток Jackbox Games

Jackbox Games – сервіс, що вартий окремої уваги, пропонує набір інтерактивних багатокористувацьких ігор для спільного проведення дозвілля. Платформа використовує систему кімнат та кодів підключення, що дозволяє дуже швидко організовувати спільні ігрові сесії. Значною перевагою рішення є велика різноманітність доступних механік взаємодії між учасниками. З іншої сторони, сервіс є закритою комерційною платформою, що не надає можливостей для інтеграції власного контенту або розширення функціональності користувачами.

### 1.1.5 Узагальнення результатів

Узагальнення результатів буде виконане порівнянням розглянутих платформ за основними критеріями, які є важливими для проєктування системи інтелектуального навчального ігрового контенту. До таких критеріїв належать робота через веб-браузер, багатокористувацька взаємодія, підтримка кімнат, робота в режимі реального часу, різні типи ігор, навчальна спрямованість та можливість розширення функціональності.

Таблиця 1.1 – Порівняльна характеристика існуючих платформ

| **Критерій** | **Board Game Arena** | **Kahoot!** | **Quizizz** | **Jackbox Games** |
|---|---|---|---|---|
| **1** | **2** | **3** | **4** | **5** |
| Робота через веб-браузер | Так | Так | Так | Частково |
| Багатокористувацька взаємодія | Так | Так | Так | Так |
| Робота в реальному часі | Так | Так | Так | Так |
| Підтримка кімнат | Так | Так | Так | Так |
| Різні типи ігор | Так | Ні | Ні | Так |
| Інтеграція власних ігор | Обмежено | Ні | Ні | Ні |
| Навчальна спрямованість | Частково | Так | Так | Ні |
| Можливість розширення системи | Обмежено | Ні | Ні | Ні |

Проведений аналіз показав, що сучасні платформи вже забезпечують ефективні механізми організації багатокористувацької взаємодії та підтримують роботу в режимі реального часу. Водночас більшість із них орієнтовані або на фіксований набір ігор, або на конкретний тип контенту, що не дозволяє можливості подальшого розвитку системи. І існуючі рішення зазвичай не передбачають гнучкого підключення нових ігрових модулів у межах єдиної архітектури.

Тому, існує доцільність створення програмного забезпечення, основою якого виступатиме універсальна система кімнат та взаємодії користувачів, а окремі ігри реалізовуватимуться у вигляді незалежних модульних сервісів. Такий підхід забезпечить масштабованість системи, спростить її підтримку та дозволить створити платформу для подальшого розширення інтелектуальним навчальним контентом.

## 1.2 Огляд існуючих технологій реалізації

Розробка сучасних веборієнтованих систем, що підтримують багатокористувацьку взаємодію в режимі реального часу, потребує використання комплексу програмних технологій, здатних забезпечити високу продуктивність, масштабованість та зручність подальшого розширення функціональності. Вибір технологічного стеку безпосередньо впливає на архітектуру програмного забезпечення, швидкість розробки та можливість інтеграції нових сервісів у майбутньому.

### 1.2.1 Фреймворк React

React є одним із найбільш поширених фреймворків для розробки клієнтської частини вебзастосунків. Він базується на компонентному підході до побудови інтерфейсу, що дозволяє створювати незалежні елементи користувацького інтерфейсу та повторно використовувати їх у різних частинах системи. Важливою перевагою React є використання віртуального DOM, що забезпечує ефективне оновлення сторінки без необхідності її повного перезавантаження. Завдяки цьому користувачі отримують швидкий та зручний інтерфейс навіть за умови частих змін даних, що зазвичай характерно для багатокористувацьких застосунків.

### 1.2.2 Платформа Node.js

Node.js була використана для реалізації серверної частини. Її особливістю є використання подієво-орієнтованої моделі виконання та неблокуючих операцій введення-виведення, що дозволяє ефективно обслуговувати велику кількість одночасних підключень. Такий підхід особливо актуальний для систем із підтримкою взаємодії в режимі реального часу, де сервер повинен оперативно реагувати на дії користувачів та забезпечувати синхронізацію стану між клієнтами.

### 1.2.3 Фреймворк Express

Express є фреймворком, що був використаний для спрощення розробки

серверної частини. Він надає набір інструментів для створення HTTP-інтерфейсів, маршрутизації запитів, обробки помилок та організації структури застосунку. Express також характеризується високою гнучкістю та невеликою кількістю обмежень щодо архітектури проєкту, що дозволяє адаптувати його під специфічні вимоги програмної системи.

### 1.2.4 Бібліотека Socket.io

Socket.IO це бібліотека, що була використана для вирішення проблеми ає організації обміну даними в реальному часі. Вона реалізує двосторонній обмін повідомленнями між клієнтом та сервером через WebSocket-з’єднання. Завдяки цьому система може миттєво передавати зміни стану кімнат, результати дій гравців та інші події всім учасникам ігрової сесії.

### 1.2.5 Система керування баз даних MongoDB

MongoDB є найпоширенішою системою керування бази даних. На відміну від реляційних баз даних, MongoDB використовує гнучку структуру документів, що дозволяє легко адаптувати модель даних до змін вимог проєкту. Такий підхід є особливо корисним під час розробки систем, у яких різні ігрові модулі можуть використовувати власні набори даних та правила взаємодії.

### 1.2.6 Контейнер Docker

Docker є одним із найбільш популярних рішень у цій сфері технологій контейнеризації. Контейнери дозволяють ізолювати програмні компоненти та забезпечити однакове середовище виконання незалежно від операційної системи або апаратної платформи. Використання Docker спрощує розгортання програмного забезпечення, підвищує його переносимість та зменшує ризик виникнення помилок, пов'язаних із відмінностями між середовищами розробки та експлуатації.

### 1.2.7 Узагальнення результатів

Для узагальнення результатів аналізу доцільно виконати порівняння

розглянутих технологій за їхнім призначенням та ключовими перевагами.

Таблиця 1.2 – Переваги використаних мною технологій

| **Технологія** | **Призначення** | **Основні переваги** |
|---|---|---|
| **1** | **2** | **3** |
| React | Клієнтська частина | Компонентний підхід, висока продуктивність, повторне використання компонентів |
| Node.js | Серверна платформа | Неблокуючі операції, висока швидкодія, підтримка великої кількості підключень |
| Express | Серверний фреймворк | Простота використання, гнучкість, швидка розробка API |
| Socket.IO | Робота в реальному часі | Двосторонній обмін даними, миттєва синхронізація стану |
| MongoDB | Зберігання даних | Гнучка структура документів, просте масштабування |
| Docker | Контейнеризація | Переносимість, ізоляція середовища, спрощення розгортання |

Проведений аналіз показує, що розглянутий набір технологій забезпечує всі необхідні можливості для створення багатокористувацької вебплатформи з підтримкою інтелектуального навчального контенту. React забезпечує побудову інтерактивного користувацького інтерфейсу, Node.js та Express відповідають за реалізацію серверної логіки, Socket.IO дозволяє організувати синхронізацію даних між користувачами в режимі реального часу, MongoDB забезпечує гнучке зберігання інформації, а Docker спрощує розгортання та супровід системи, як видно на рисунку 1.1.

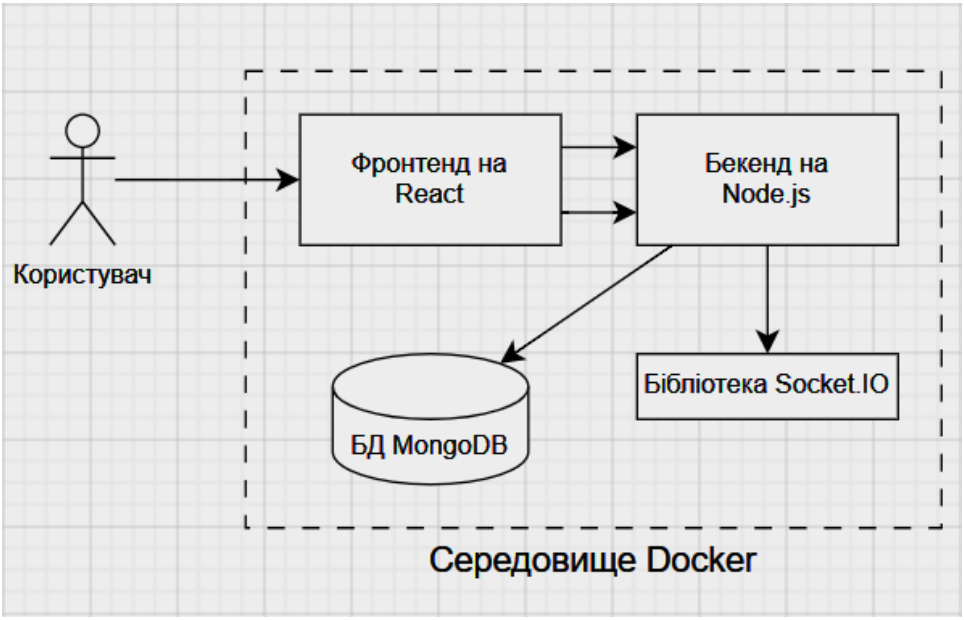


Рисунок 1.1 – Діаграма взаємодії технологій у структурі системи

Таким чином, обраний технологічний стек створює надійну основу для реалізації програмного забезпечення на базі мультисервісної архітектури та забезпечує можливість подальшого розширення системи шляхом інтеграції нових ігрових або навчальних модулів.

### 1.3 Висновки до першого розділу

У першому розділі було проведено ґрунтовний аналіз предметної області та досліджено сучасні підходи до розроблення багатокористувацьких веборієнтованих систем, призначених для організації інтелектуального, навчального та розважального контенту. Особливу увагу приділено вивченню платформ, що забезпечують спільну взаємодію користувачів, підтримують механізми гейміфікації навчального процесу та реалізують обмін даними в режимі реального часу. Це дозволило визначити ключові тенденції розвитку подібних програмних продуктів та встановити їхні основні функціональні особливості.

У ході дослідження було проаналізовано сучасні технології та інструменти розробки вебзастосунків. На основі проведеного аналізу обґрунтовано доцільність використання бібліотеки React для створення клієнтської частини системи завдяки

її компонентному підходу та високій продуктивності. Для реалізації серверної логіки обрано платформу Node.js та фреймворк Express, які забезпечують ефективну обробку запитів і зручність розроблення масштабованих вебсервісів. Для підтримки синхронізації даних між користувачами в режимі реального часу визначено доцільність використання бібліотеки Socket.IO. Як систему керування базами даних обрано MongoDB, що дозволяє гнучко працювати зі структурованими та напівструктурованими даними. Крім того, розглянуто використання технології Docker, яка забезпечує контейнеризацію застосунку, спрощує процес розгортання та підвищує переносимість програмного забезпечення між різними середовищами.

На основі отриманих результатів було сформовано та уточнено основні функціональні й нефункціональні вимоги до розроблюваної системи. Зокрема, програмне забезпечення повинно забезпечувати створення та адміністрування користувацьких кімнат, підтримувати надійний обмін даними в режимі реального часу, надавати можливість підключення незалежних ігрових модулів, а також гарантувати зручність подальшого розширення функціональності без необхідності суттєвої перебудови архітектури. Важливою вимогою також є забезпечення масштабованості, зручності використання та підтримки різних сценаріїв взаємодії користувачів.

Таким чином, результати першого розділу створюють необхідне теоретичне та методологічне підґрунтя для подальшого проєктування архітектури програмного забезпечення, вибору структури його компонентів та реалізації основних функціональних можливостей системи. Отримані висновки та сформовані вимоги будуть використані в наступних розділах роботи під час розроблення архітектурних рішень, моделювання взаємодії компонентів і практичної реалізації програмного продукту.

# 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ

Після проведення аналізу предметної області та існуючих технологій наступним етапом є проєктування програмного забезпечення, яке забезпечує організацію багатокористувацької взаємодії та підтримує інтеграцію інтелектуального навчального контенту у вигляді окремих ігрових модулів. На відміну від спеціалізованих платформ, орієнтованих на реалізацію одного типу активностей, розроблювана система повинна забезпечувати можливість підключення різних ігор у межах єдиного середовища взаємодії користувачів.

Основною особливістю проєкту є використання мультисервісного підходу до організації програмної логіки. У межах системи окремо реалізуються сервіси керування користувачами та кімнатами, а також незалежні модулі ігор, кожен із яких містить власну бізнес-логіку та набір механізмів взаємодії з клієнтською частиною. Такий підхід дозволяє спростити підтримку програмного забезпечення, забезпечити його масштабованість та створити умови для подальшого розширення функціональності.

У даному розділі розглядаються вимоги до системи, обґрунтовується вибір архітектурних рішень, описуються моделі даних та принципи організації взаємодії між компонентами. Окрема увага приділяється проєктуванню системи кімнат, механізму роботи в режимі реального часу та підходу до інтеграції незалежних ігрових модулів. Завершальним етапом є опис реалізації програмного забезпечення та основних технічних рішень, використаних під час розробки.

## 2.1 Вимоги до системи

Формування вимог є одним із найважливіших етапів проєктування програмного забезпечення, оскільки саме на їх основі визначаються архітектурні рішення, структура даних та механізми взаємодії між компонентами системи. Правильно сформульовані вимоги дозволяють забезпечити відповідність готового програмного продукту поставленим завданням та очікуванням користувачів.

Розроблювана система призначена для організації багатокористувацької взаємодії в межах віртуальних кімнат із можливістю запуску та використання різних ігрових модулів. Основною ідеєю проєкту є створення універсальної платформи, яка забезпечує керування користувачами та кімнатами незалежно від конкретної гри, тоді як логіка кожної гри реалізується у вигляді окремого функціонального модуля.

Під час визначення вимог було враховано необхідність забезпечення швидкої взаємодії між користувачами, підтримки синхронізації стану в режимі реального часу та можливості подальшого розширення системи новими іграми без суттєвої зміни вже реалізованих компонентів. Особлива увага приділялася простоті використання системи, мінімізації кількості дій, необхідних для початку роботи, а також забезпеченню стабільної роботи за наявності декількох одночасно активних кімнат.

Для систематизації вимог доцільно розділити їх на функціональні, нефункціональні та вимоги до користувацького інтерфейсу. Такий підхід дозволяє комплексно описати як можливості програмного забезпечення, так і критерії його якості та зручності використання.

### 2.1.1 Функціональні вимоги

Функціональні вимоги визначають перелік можливостей, які повинна забезпечувати система для кінцевого користувача. Вони описують сценарії взаємодії користувачів із програмним забезпеченням та перелік операцій, необхідних для організації багатокористувацьких ігрових сесій.

Особливістю розроблюваної системи є розділення функціональності на декілька незалежних підсистем. Центральним елементом виступає сервіс керування кімнатами, який забезпечує створення середовища взаємодії користувачів. Ігрові модулі працюють незалежно від механізму створення кімнат та використовують їх лише як спосіб організації спільної сесії.

До основних функціональних вимог системи належать:

1. Створення та ідентифікація користувача.

Система повинна забезпечувати можливість створення нового користувача із зазначенням псевдоніма. Для кожного користувача формується унікальний ідентифікатор, який використовується під час подальшої взаємодії із системою. Також повинна підтримуватися можливість зміни відображуваного імені без необхідності повторної реєстрації.

2. Керування кімнатами.

Користувач повинен мати можливість створити окрему кімнату для проведення ігрової сесії. Для кожної кімнати автоматично генерується унікальний код, який використовується іншими користувачами для приєднання.

Під час створення кімнати необхідно визначити:

- тип гри;
- максимальну кількість учасників;
- власника кімнати.

3. Приєднання до кімнати.

Система повинна дозволяти користувачам приєднуватися до існуючих кімнат за допомогою унікального коду. Перед приєднанням необхідно виконувати перевірку існування кімнати та наявності вільних місць.

4. Вихід із кімнати.

Користувач повинен мати можливість добровільно залишити кімнату. Після виходу список учасників має автоматично оновлюватися для всіх підключених клієнтів.

5. Керування учасниками кімнати.

Власник кімнати повинен мати додаткові права керування сесією. Зокрема, він повинен мати можливість видаляти учасників із кімнати до початку гри.

6. Запуск ігрової сесії.

Запуск гри повинен бути доступний лише власнику кімнати. Перед початком гри система повинна перевіряти відповідність кількості учасників мінімальним вимогам обраного ігрового модуля.

7. Синхронізація даних у режимі реального часу.

Усі зміни, пов’язані зі станом кімнати або гри, повинні миттєво передаватися всім підключеним користувачам без необхідності ручного оновлення сторінки.

8. Підключення незалежних ігрових модулів.

Архітектура системи повинна дозволяти додавання нових ігор без зміни логіки роботи кімнат та користувачів. Кожен ігровий модуль повинен містити власний механізм обробки дій гравців та керування ігровим станом.

У поточній реалізації платформи передбачено два ігрові модулі:

- «Хрестики-нулики»;
- «Дуже гучні бібліотекарі».

Модуль гри «Хрестики-нулики» забезпечує проведення двокористувацької партії з автоматичним контролем черговості ходів та перевіркою умов перемоги.

Модуль гри «Дуже гучні бібліотекарі» реалізує командний формат взаємодії користувачів, у якому гравці виконують завдання відповідно до випадково обраних категорій та літер. Система автоматично керує перебігом раундів та синхронізує поточний стан між усіма учасниками.

Для узагальнення функціональних можливостей системи доцільно побудувати діаграму варіантів використання (Use Case Diagram), яка відображає взаємодію користувача із ключовими підсистемами програмного забезпечення.

На рисунку 2.1 наведено діаграму варіантів використання розроблюваної системи.

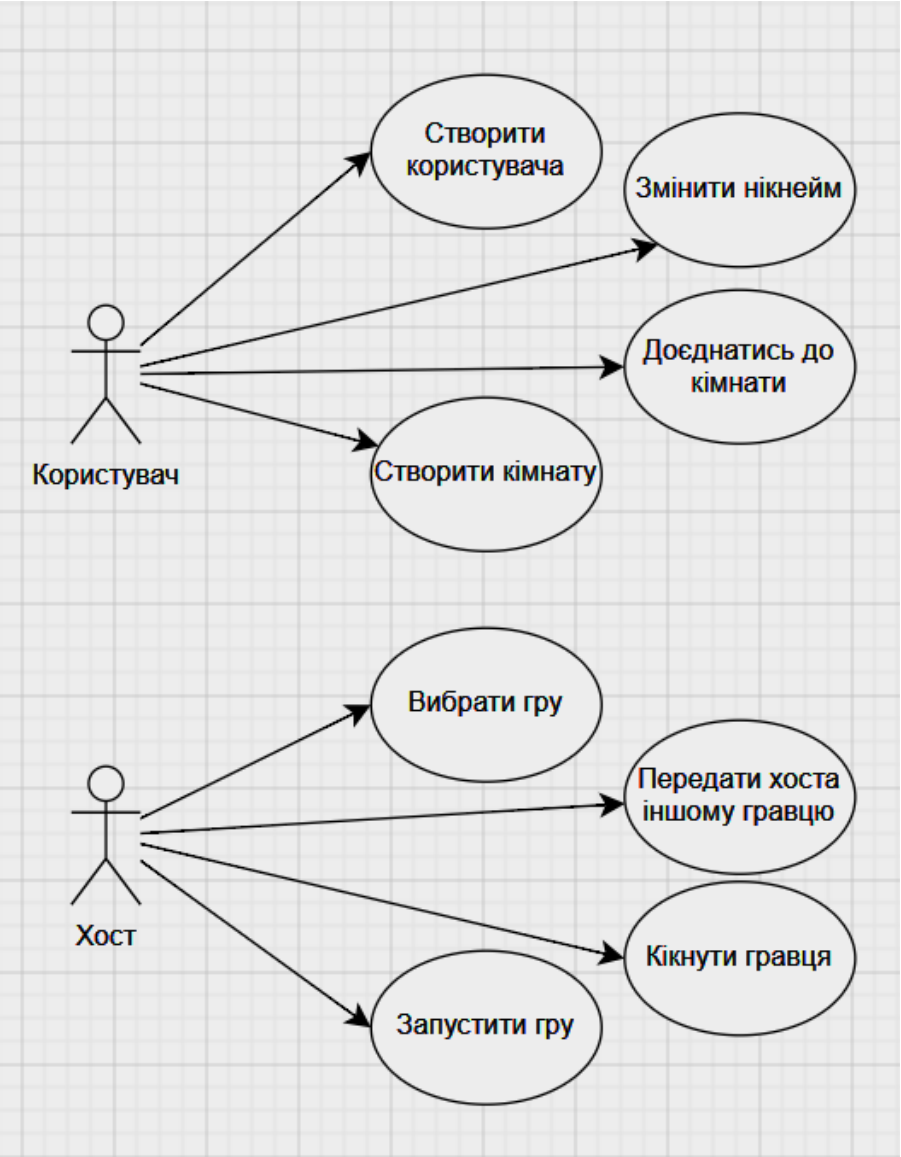


Рисунок 2.1 – Діаграма використань системи

### 2.1.2 Нефукціональні вимоги

Окрім функціональних можливостей, важливу роль у процесі проєктування програмного забезпечення відіграють нефункціональні вимоги. Вони визначають характеристики якості системи, особливості її експлуатації, продуктивність, надійність та зручність подальшого розвитку. Саме нефункціональні вимоги значною мірою впливають на вибір архітектурних рішень та використовуваних технологій.

Оскільки розроблювана система орієнтована на багатокористувацьку взаємодію в режимі реального часу, особлива увага приділяється швидкості обробки подій, стабільності передачі даних між клієнтською та серверною

частинами, а також можливості розширення функціональності шляхом підключення нових ігрових модулів.

#### 2.1.2.1 Вимоги до продуктивності

Система повинна забезпечувати швидке виконання основних операцій користувача, включаючи створення кімнат, приєднання до них та обробку ігрових дій. Взаємодія між учасниками повинна відбуватися практично без помітних затримок, оскільки навіть незначне відставання може негативно впливати на користувацький досвід під час багатокористувацької гри.

Для забезпечення високої швидкодії використовується технологія WebSocket, яка дозволяє підтримувати постійне двостороннє з'єднання між клієнтом і сервером. На відміну від традиційного підходу із періодичним опитуванням сервера, такий механізм дозволяє миттєво доставляти оновлення всім учасникам ігрової сесії.

Ігрові стани зберігаються в оперативній пам'яті сервера, що мінімізує кількість звернень до бази даних під час активної гри та додатково зменшує затримки при обробці ігрових подій.

#### 2.1.2.2 Вимоги до масштабованості

Архітектура системи повинна забезпечувати можливість подальшого розширення функціональності без необхідності суттєвої переробки вже реалізованих компонентів.

Досягнення цієї вимоги забезпечується шляхом розділення програмної логіки на незалежні модулі, кожен із яких відповідає за окрему предметну область. Зокрема, керування користувачами, кімнатами та ігровими процесами реалізується окремими сервісами та обробниками подій.

Такий підхід дозволяє додавати нові ігрові модулі без внесення змін до механізмів створення кімнат, автентифікації користувачів або обміну повідомленнями між клієнтом і сервером.

#### 2.1.2.3 Вимоги до модульності

Однією з ключових вимог до системи є забезпечення незалежності окремих функціональних компонентів.

Логіка роботи кімнат не повинна залежати від реалізації конкретних ігор. Аналогічно ігрові модулі не повинні містити інформації про внутрішню реалізацію сервісів керування кімнатами або користувачами. Взаємодія між компонентами повинна здійснюватися виключно через визначені інтерфейси та структури даних.

Подібний підхід спрощує супровід програмного забезпечення, зменшує ризик виникнення побічних ефектів під час внесення змін та покращує можливості повторного використання окремих компонентів.

#### 2.1.2.4 Вимоги до надійності

Система повинна коректно обробляти помилкові або неповні запити користувачів.

До типових ситуацій належать:

- спроба приєднання до неіснуючої кімнати;
- спроба запуску гри за недостатньої кількості учасників;
- виконання дій користувачем без відповідних прав;
- передавання некоректних параметрів під час ігрової взаємодії.

У разі виникнення подібних ситуацій система повинна повертати зрозумілі повідомлення про помилки без припинення роботи сервера або втрати даних інших користувачів.

Додатково необхідно забезпечити коректну обробку втрати з'єднання із клієнтом. У випадку відключення користувача система повинна автоматично оновлювати інформацію про склад кімнати та синхронізувати зміни між усіма учасниками.

#### 2.1.2.5 Вимоги до підтримуваності

Програмне забезпечення повинно мати зрозумілу структуру та бути придатним для подальшого розвитку.

Для цього застосовуються сучасні підходи до організації коду, зокрема:

- поділ на функціональні модулі;
- використання TypeScript для статичної типізації;
- розмежування рівнів представлення, бізнес-логіки та роботи з даними;
- централізована обробка помилок.

Завдяки цьому нові функціональні можливості можуть додаватися незалежно від уже реалізованих компонентів системи.

#### 2.1.2.6 Вимоги до сумісності

Клієнтська частина повинна коректно працювати в сучасних браузерах, які підтримують актуальні вебстандарти та технологію WebSocket.

До підтримуваних браузерів належать:

- Google Chrome;
- Microsoft Edge;
- Opera.

Інтерфейс користувача повинен коректно адаптуватися до різних розмірів екрана та забезпечувати комфортну роботу як на персональних комп'ютерах, так і на портативних пристроях.

#### 2.1.2.7 Вимоги до безпеки

Оскільки система не передбачає повноцінної реєстрації користувачів із використанням логіна та пароля, основним завданням механізмів безпеки є контроль коректності ідентифікаторів користувачів та перевірка прав доступу до окремих дій.

Серверна частина повинна виконувати перевірку всіх вхідних даних незалежно від результатів клієнтської валідації. Особлива увага приділяється перевірці прав власника кімнати під час запуску гри та керування учасниками.

Такий підхід дозволяє запобігти виконанню несанкціонованих дій навіть у випадку модифікації клієнтського програмного забезпечення користувачем.

Таким чином, сформульовані нефункціональні вимоги визначають основні критерії якості програмного забезпечення та безпосередньо впливають на вибір архітектури, технологічного стеку та підходів до реалізації системи.

### 2.1.3 Вимоги до інтерфейсу користувача

Інтерфейс користувача є ключовим елементом системи, оскільки саме через нього здійснюється вся взаємодія користувачів із функціоналом платформи. Оскільки розроблювана система орієнтована на багатокористувацькі ігри в режимі реального часу, інтерфейс повинен забезпечувати не лише зручність використання, але й швидке відображення змін стану гри та кімнати без необхідності перезавантаження сторінки.

Основною метою при проєктуванні інтерфейсу є створення інтуїтивно зрозумілої структури, яка дозволяє користувачу швидко виконувати основні дії: створення профілю, приєднання до кімнати, взаємодію з іншими гравцями та участь у ігровому процесі.

#### 2.1.3.1 Вимоги до UI

Інтерфейс повинен відповідати таким загальним вимогам:

- забезпечувати просту та логічну навігацію між сторінками системи;
- мінімізувати кількість дій, необхідних для виконання основних операцій;
- забезпечувати миттєве відображення змін стану гри та кімнати;
- бути адаптивним для різних типів пристроїв і роздільної здатності екрана;
- використовувати єдиний візуальний стиль для всіх компонентів системи.

Особлива увага приділяється реактивності інтерфейсу, оскільки система активно використовує WebSocket-з’єднання для синхронізації даних у реальному часі.

Головна сторінка повинна забезпечувати можливість створення нового користувача або входу в систему через введення псевдоніма. Після авторизації користувач отримує доступ до функціоналу створення або приєднання до кімнат.

Інтерфейс головної сторінки повинен бути максимально простим і не перевантаженим додатковими елементами, оскільки він виконує роль початкової точки входу в систему.

Інтерфейс створення кімнати повинен дозволяти користувачу:

- вибрати тип гри;
- задати максимальну кількість гравців;
- створити нову кімнату з автоматично згенерованим унікальним кодом.

Сторінка приєднання до кімнати повинна містити поле для введення коду кімнати та кнопку підтвердження. Після успішного приєднання користувач автоматично перенаправляється до лобі кімнати.

Лобі є центральним елементом взаємодії користувачів перед початком гри. Воно повинно відображати:

- список підключених гравців;
- інформацію про власника кімнати;
- обрану гру;
- максимальну кількість учасників;
- статус готовності до запуску гри.

Для власника кімнати додатково повинна бути доступна кнопка запуску гри, яка активується лише за умови виконання мінімальних вимог до кількості гравців.

Ігровий інтерфейс змінюється залежно від обраного ігрового модуля.

Для гри «Хрестики-нулики» інтерфейс повинен містити ігрове поле 3×3, яке відображає поточний стан гри та дозволяє гравцям виконувати ходи у відповідний момент їхнього ходу.

Для гри «Дуже гучні бібліотекарі» інтерфейс повинен відображати:

- поточну літеру раунду;
- список категорій;
- поточний рахунок команд;

- таймер раунду (за наявності);
- кнопки для нарахування балів.

Інтерфейс повинен автоматично блокувати або активувати елементи керування залежно від стану гри та прав користувача.

#### 2.1.3.2 Вимоги до взаємодії в реальному часі

Інтерфейс повинен миттєво відображати всі зміни, що відбуваються у кімнаті або грі. Це включає:

- оновлення списку гравців;
- зміни стану гри;
- хід гри;
- завершення раундів або ігрових сесій.

Для цього використовується механізм WebSocket, який забезпечує постійне з’єднання між клієнтом і сервером.

#### 2.1.3.3 Вимоги до візуального стилю

Інтерфейс повинен мати сучасний мінімалістичний дизайн, орієнтований на зручність користування. Основний акцент робиться на читабельності, контрастності та чіткій структурі елементів.

Рекомендується використання:

- темної або нейтральної кольорової палітри;
- акцентних кольорів для інтерактивних елементів;
- візуального розділення логічних блоків (лобі, гра, меню);
- анімацій для переходів між станами.

#### 2.1.3.4 Вимоги до помилок та повідомлень

Інтерфейс повинен коректно відображати повідомлення про помилки, які надходять із серверної частини. Повідомлення мають бути зрозумілими для користувача та не містити технічної термінології.

Приклад ситуацій:

неможливо приєднатися до кімнати;

- кімната переповнена;
- гра вже розпочата;
- недостатньо прав для виконання дії.

На рисунках 2.2-2.5 наведено скриншоти користувацького інтерфейсу системи, для кращого розуміння структури інтерфейсу.

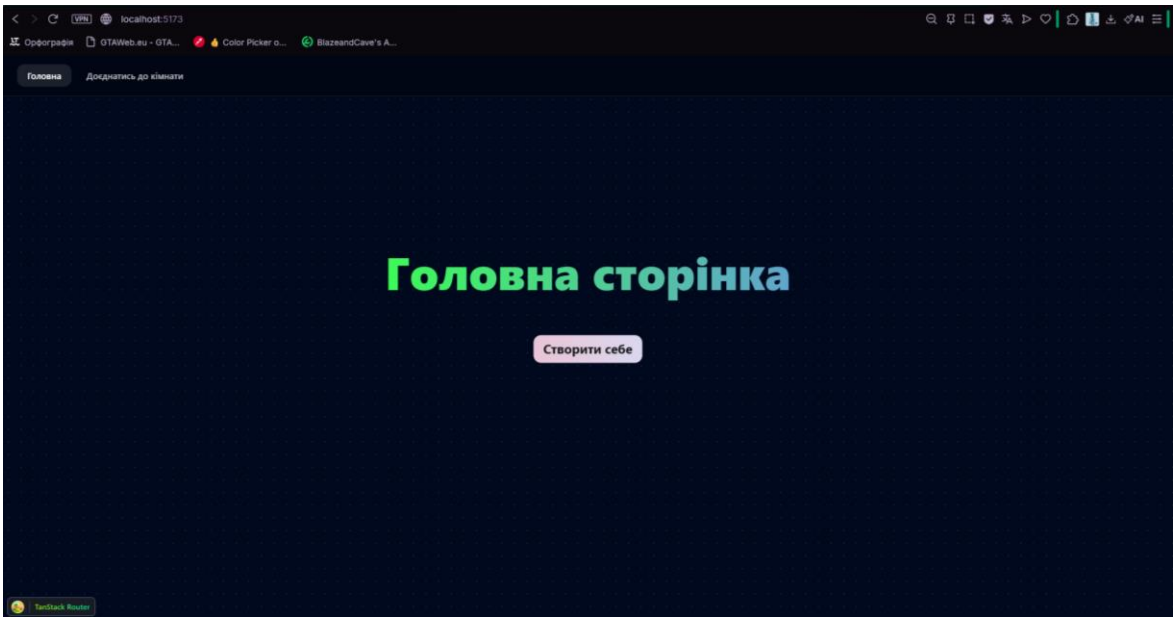


Рисунок 2.2 – Головна сторінка із нествореним користувачем

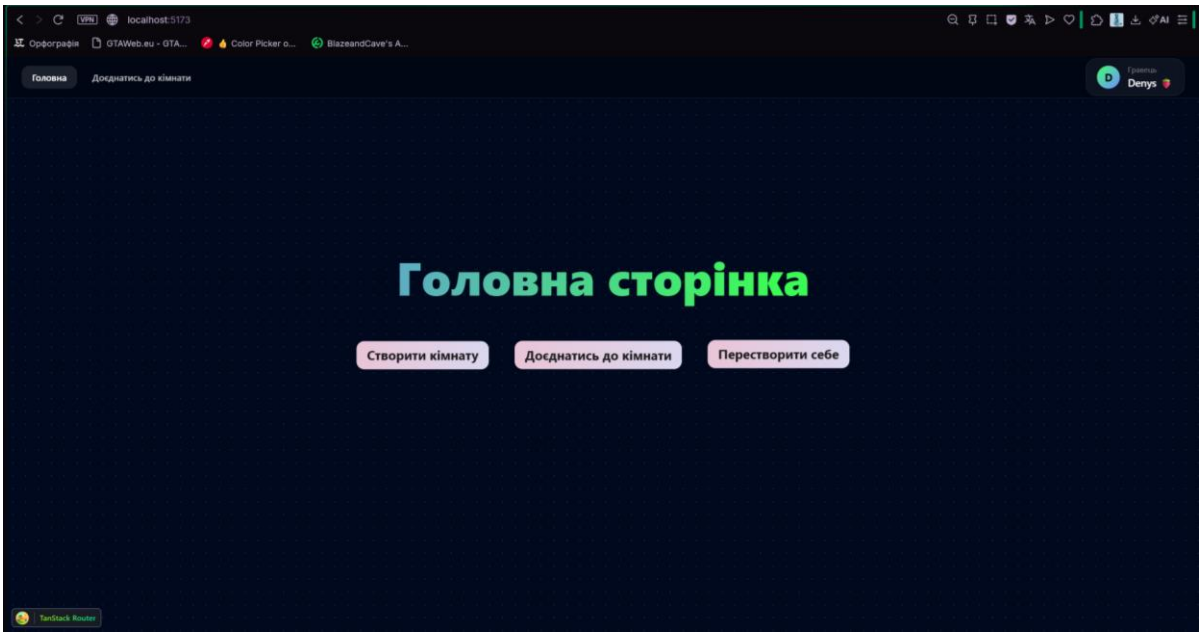


Рисунок 2.3 – Головна сторінка із створеним користувачем

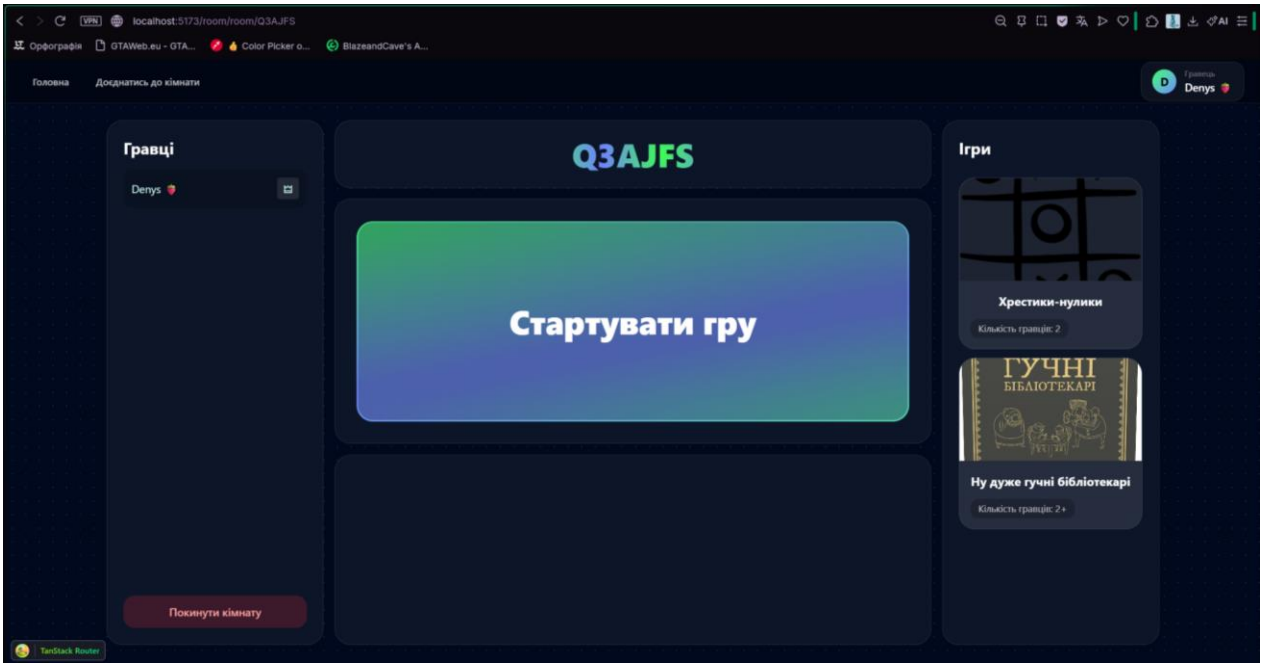


Рисунок 2.4 – Сторінка лобі

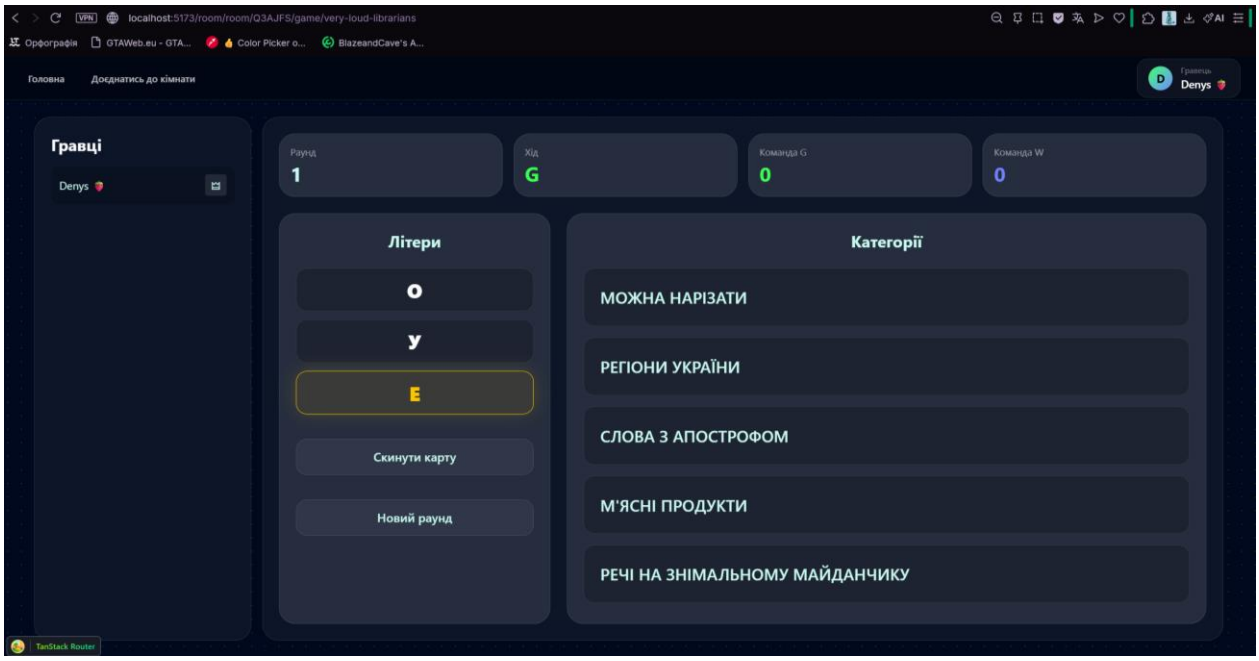


Рисунок 2.5 – Сторінка гри «Ну дуже гучні бібліотекарі»

## 2.2 Архітектура системи

Архітектура розроблюваного програмного забезпечення побудована за принципом монолітного серверного застосунку з модульною структурою

внутрішніх компонентів. Уся серверна логіка, включаючи REST API та механізм взаємодії в режимі реального часу, реалізована в межах одного Node.js застосунку, що працює на єдиному порті.

Такий підхід дозволяє зменшити складність розгортання системи, спростити взаємодію між компонентами та забезпечити низькі затримки при обміні даними між підсистемами.

Загальна архітектура системи базується на розділенні відповідальності між клієнтською та серверною частинами, а також на виділенні окремих модулів для реалізації бізнес-логіки ігор.

### 2.2.1 Загальна архітектура програмного забезпечення

Система складається з трьох основних рівнів:

- клієнтська частина (Frontend);
- серверна частина (Backend);
- рівень збереження даних (Database).

Взаємодія між компонентами відбувається через REST API та WebSocket-з’єднання.

Клієнтська частина відповідає за відображення інтерфейсу користувача та ініціює запити до серверної частини. Сервер обробляє бізнес-логіку, керує станом кімнат та ігрових сесій, а також забезпечує синхронізацію даних у режимі реального часу.

### 2.2.2 Архітектура клієнтської частини

Клієнтська частина реалізована з використанням бібліотеки React та побудована як SPA (Single Page Application). Для маршрутизації використовується TanStack Router, що забезпечує гнучке керування сторінками та вкладеними маршрутами.

Основні компоненти клієнтської частини:

- React компоненти — відображення інтерфейсу;

- Context API — управління глобальним станом (користувач, кімната, гра);
- Axios — взаємодія з REST API;
- Socket.IO Client — взаємодія в режимі реального часу.

Особливістю клієнтської частини є активне використання подієвої моделі, де більшість змін стану відбувається через WebSocket-івенти, що дозволяє уникнути надмірних HTTP-запитів.

### 2.2.3 Архітектура серверної частини

Серверна частина побудована на базі Node.js із використанням Express та Socket.IO.

Архітектура серверної частини є модульною та включає такі основні компоненти:

- Room Service — відповідає за створення та керування кімнатами;
- Player Service — відповідає за керування користувачами;
- Game Handlers — реалізують логіку конкретних ігрових модулів;
- Socket Layer — забезпечує обробку подій у реальному часі;
- REST API Layer — надає HTTP-інтерфейс для клієнтської частини.

Важливою особливістю є те, що ігрові модулі реалізовані незалежно один від одного та не мають прямої залежності від логіки кімнат. Кімната виступає лише як контейнер для групи користувачів та ідентифікатор контексту гри.

### 2.2.4 Організація взаємодії в режимі реального часу

Для забезпечення взаємодії між клієнтом і сервером у режимі реального часу використовується технологія WebSocket на базі Socket.IO.

Сервер підтримує постійне з’єднання з клієнтами, що дозволяє миттєво передавати оновлення стану кімнати або гри всім учасникам сесії.

Основні типи подій:

- приєднання та вихід із кімнати;
- оновлення стану кімнати;

- ігрові дії користувачів;
- завершення гри або раунду.

Завдяки такій моделі забезпечується синхронізація стану між усіма клієнтами без необхідності періодичного опитування сервера.

Для наочності було побудувано компонентну діаграму системи, яка відображає взаємодію між основними модулями.

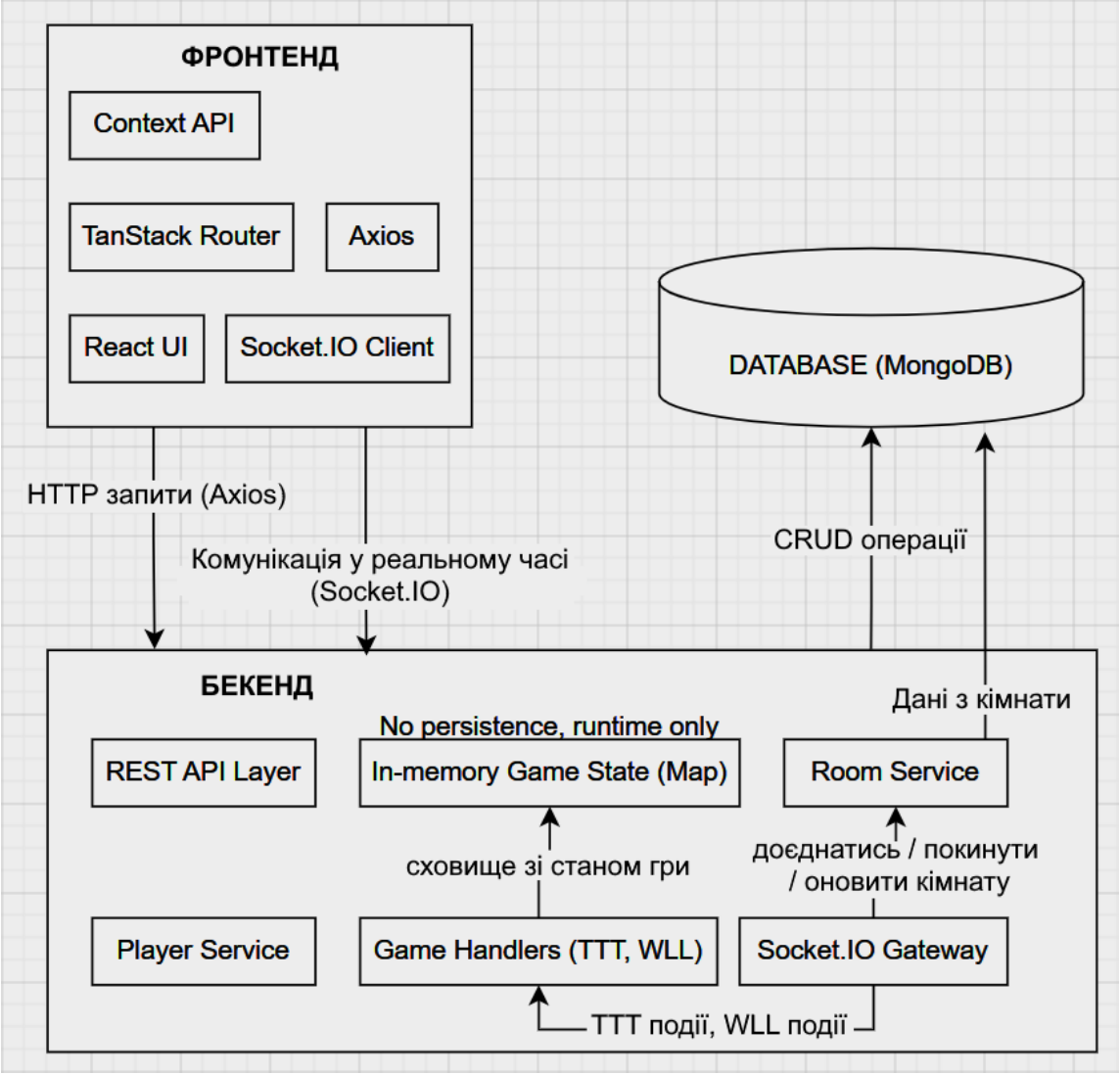


Рисунок 2.6 – Компонентна діаграма системи

## 2.3 Проєктування моделі даних

Проєктування моделі даних є важливим етапом розробки програмного забезпечення, оскільки воно визначає структуру збереження інформації та принципи її обробки в межах системи. У розроблюваному застосунку модель даних побудована з урахуванням поділу відповідальності між довготривалими сутностями, що зберігаються у базі даних, та тимчасовими станами, які існують лише під час активної взаємодії користувачів.

Особливістю системи є розділення даних на два типи: постійні дані (користувачі, кімнати) та тимчасові ігрові стани, які зберігаються в оперативній пам’яті сервера. Такий підхід дозволяє оптимізувати продуктивність системи та зменшити навантаження на базу даних під час ігрового процесу.

### 2.3.1 Модель користувача

Сутність користувача (Player) є базовою одиницею системи та використовується для ідентифікації учасників ігрових сесій.

Основні атрибути користувача:

- унікальний ідентифікатор (_id);
- відображуване ім’я (nickname);
- службові дані для автентифікації або сесійної ідентифікації.

Користувач створюється при першому вході в систему та зберігається у базі даних. Подальші дії користувача в системі прив’язуються до його унікального ідентифікатора.

### 2.3.2 Модель кімнати

Кімната (Room) є центральною структурою, що об’єднує користувачів у межах однієї ігрової сесії.

Структура кімнати включає такі поля:

- code — унікальний код кімнати, який використовується для приєднання;
- hostId — ідентифікатор користувача, який створив кімнату;
- playersId — список ідентифікаторів учасників кімнати;

- maxPlayers — максимальна кількість учасників;
- isGameStillOn — статус активності гри;
- game — тип обраної гри (перелік із GamesEnum).

Кімната виступає контейнером для організації взаємодії між користувачами та не містить внутрішньої логіки конкретної гри.

### 2.3.3 Модель ігрового стану

На відміну від користувачів та кімнат, ігрові стани не зберігаються у базі даних, а існують виключно в оперативній пам’яті сервера протягом активної сесії.

Для кожного типу гри створюється окремий клас стану. Наприклад, для гри «Хрестики-нулики» використовується клас TTTBoardState, який інкапсулює логіку ігрового поля та правила гри.

Ігрові стани зберігаються у структурі типу Map, де ключем виступає код кімнати:

```
private readonly games = new Map<string, TTTBoardState>();
```

Такий підхід дозволяє:

- швидко отримувати стан гри за кодом кімнати;
- уникати зайвих запитів до бази даних;
- забезпечувати низьку затримку оновлення стану гри.

Після завершення ігрової сесії відповідний стан видаляється з пам’яті сервера.

### 2.3.4 Обґрунтування вибору моделі даних

Запропонована модель даних базується на принципі розділення довготривалого та тимчасового збереження інформації. Постійні дані (користувачі та кімнати) зберігаються у базі даних для забезпечення їх доступності між сесіями, тоді як ігрові дані існують лише під час активної взаємодії користувачів.

Це дозволяє:

- зменшити навантаження на базу даних;
- підвищити швидкість обробки ігрових подій;

- спростити реалізацію механізмів реального часу;
- уникнути складності відновлення тимчасових ігрових станів.

Водночас такий підхід означає, що після перезапуску сервера активні ігрові сесії не відновлюються, що є прийнятним для розроблюваної системи, оскільки вона орієнтована на короткотривалі ігрові взаємодії.

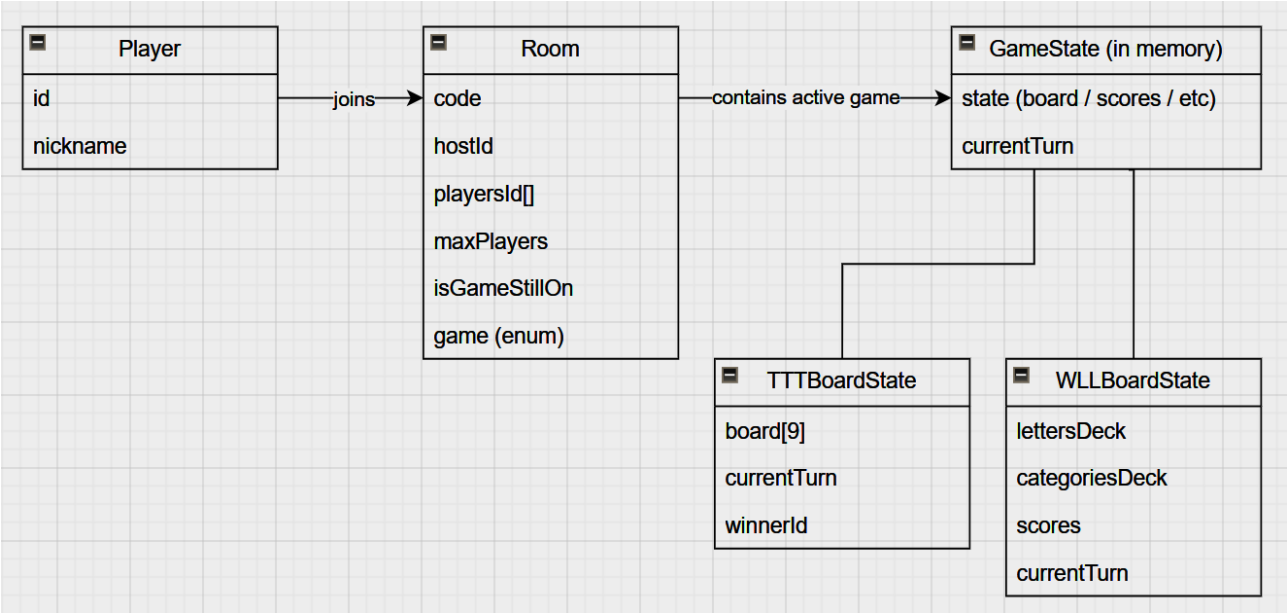


Рисунок 2.7 – Діаграма моделі даних

## 2.4 Проєктування та реалізація підсистеми керування кімнатами

Підсистема керування кімнатами є однією з ключових частин розроблюваного програмного забезпечення, оскільки саме вона забезпечує організацію користувачів у спільний контекст взаємодії та виступає базовим шаром для запуску будь-яких ігрових сценаріїв. У межах системи кімната є абстракцією, що об’єднує групу гравців та визначає поточний ігровий режим, однак не містить логіки конкретних ігор.

Особливістю реалізації є те, що керування кімнатами відбувається як через HTTP-запити (створення та отримання даних), так і через WebSocket-з’єднання, що забезпечує синхронізацію стану кімнати в реальному часі.

### 2.4.1 Основні функції підсистеми кімнат

Підсистема кімнат реалізує наступні базові функції:

- створення нової кімнати;
- приєднання користувача до кімнати;
- вихід користувача з кімнати;
- призначення та перевірка хоста;
- видалення користувача з кімнати (kick);
- ініціалізація запуску гри;
- синхронне оновлення стану кімнати для всіх учасників.

Таким чином, кімната виступає центральним координаційним елементом між користувачами та ігровими модулями.

### 2.4.2 Життєвий цикл кімнати

Життєвий цикл кімнати можна умовно поділити на кілька етапів:

1. Створення кімнати. Користувач створює кімнату та автоматично отримує статус хоста.
2. Очікування гравців. Інші користувачі приєднуються до кімнати за унікальним кодом.
3. Підготовка до гри. Хост обирає тип гри та перевіряє кількість гравців.
4. Запуск гри. Після виконання умов запускається відповідний ігровий модуль.
5. Завершення гри. Стан кімнати оновлюється, гра завершується або перезапускається.

### 2.4.3 Реалізація через WebSocket

Для забезпечення взаємодії в реальному часі використовується Socket.IO. Кожен клієнт приєднується до кімнати через відповідний socket room, що дозволяє серверу надсилати оновлення лише учасникам конкретної сесії.

Основні події WebSocket:

- join-room — приєднання користувача до кімнати;

- update-room — оновлення стану кімнати;
- start-game — ініціація запуску гри;
- kick-player — видалення користувача з кімнати;
- disconnect — автоматичне відключення користувача.

Передача оновлень виконується через механізм broadcast у межах кімнати, що забезпечує миттєву синхронізацію даних між усіма клієнтами.

### 2.4.4 Серверна логіка керування кімнатами

Серверна реалізація підсистеми базується на виділеному сервісі кімнат, який відповідає за збереження та модифікацію стану кімнат.

Основні операції включають:

- додавання гравця до кімнати;
- видалення гравця з кімнати;
- перевірка існування кімнати;
- перевірка прав хоста;
- оновлення стану гри в кімнаті.

Важливою особливістю є те, що сервіс кімнат не містить логіки ігор. Його функціональність обмежується лише управлінням списком учасників та базовими атрибутами кімнати, що відповідає принципу розділення відповідальностей.

### 2.4.5 Синхронізація стану кімнати

Стан кімнати зберігається на сервері та синхронізується між усіма підключеними клієнтами. При будь-якій зміні (приєднання, вихід, kick або старт гри) сервер формує оновлену модель кімнати та передає її всім учасникам через подію update-room.

Такий підхід дозволяє уникнути необхідності ручного оновлення стану на клієнті та забезпечує консистентність даних між усіма користувачами.

### 2.4.6 Обробка виняткових ситуацій

У процесі роботи підсистеми можливі різні виняткові ситуації, зокрема:

- спроба приєднання до неіснуючої кімнати;
- перевищення максимальної кількості гравців;
- спроба виконання дій не хостом;
- некоректне відключення клієнта.

Для обробки таких ситуацій використовується механізм перевірок на сервері з подальшою відправкою повідомлення про помилку через WebSocket або HTTP-відповідь.

## 2.5 Проєктування ігрового ядра системи

Ігрове ядро є ключовим програмним компонентом розроблюваного застосунку, який відповідає за виконання логіки конкретних ігор у межах обраної кімнати. Його основною задачею є ізоляція ігрових механік від підсистеми керування кімнатами та забезпечення незалежного життєвого циклу кожної ігрової сесії.

Загальний підхід до реалізації базується на концепції in-memory збереження стану гри, що дозволяє забезпечити високу швидкість обробки подій та мінімальну затримку при взаємодії користувачів у режимі реального часу.

### 2.5.1 Загальна концепція ігрового ядра

Ігрове ядро реалізує принцип розділення відповідальностей, згідно з яким:

- кімната (Room) відповідає лише за групу користувачів;
- ігрове ядро відповідає за правила та стан конкретної гри;
- серверна логіка координує обмін подіями між клієнтами та ігровим станом.

Таким чином, система дозволяє запускати різні ігри в межах однакової інфраструктури кімнат без внесення змін у логіку кімнатного сервісу.

### 2.5.2 Зберігання ігрових станів

Усі активні ігрові сесії зберігаються в оперативній пам’яті сервера у вигляді структури типу Map, де ключем виступає унікальний код кімнати.

Це дозволяє швидко отримувати доступ до стану конкретної гри та оновлювати його у відповідь на дії користувачів.

Основна структура має вигляд:

- ключ: roomCode
- значення: екземпляр класу стану гри

Наприклад:

- TTTBoardState — для гри «Хрестики-нулики»;
- WLLBoardState — для гри «Дуже гучні бібліотекарі».

Такий підхід дозволяє одночасно підтримувати різні типи ігор у різних кімнатах без конфліктів між їхніми станами.

### 2.5.3 Модель ігрового стану

Кожна гра реалізується у вигляді окремого класу, який інкапсулює:

- внутрішній стан гри;
- правила зміни стану;
- перевірку завершення гри;
- методи обробки дій гравців;
- метод отримання актуального стану для клієнта.

Приклад структури для Tic-Tac-Toe:

- ігрове поле (board);
- поточний хід (currentTurn);
- стан завершення гри (winner або null).

Приклад структури для WLL:

- колоди карт;
- поточний раунд;
- рахунок гравців;
- активна команда;

- поточні значення раунду.

Таким чином, кожен клас є повністю автономною моделлю гри.

### 2.5.4 Обробка ігрових подій

Взаємодія між клієнтом та ігровим ядром реалізується через WebSocket-події. Кожна дія користувача транслюється на сервер, де обробляється відповідним ігровим хендлером.

Типовий потік подій:

1. клієнт надсилає ігрову дію (наприклад, хід у грі);
2. сервер отримує подію через socket handler;
3. визначається відповідний ігровий стан за roomCode;
4. виконується метод зміни стану;
5. оновлений стан надсилається всім клієнтам кімнати.

Такий механізм забезпечує синхронізацію стану гри в реальному часі.

### 2.5.5 Ізоляція ігрової логіки

Важливим архітектурним рішенням є повна ізоляція ігрової логіки від інших підсистем. Room Service не містить інформації про правила гри, а лише передає ідентифікатор активної гри.

Це дозволяє:

- легко додавати нові ігри без зміни існуючого коду кімнат;
- уникнути залежностей між різними ігровими модулями;
- забезпечити масштабованість системи.

### 2.5.6 Життєвий цикл гри

Життєвий цикл ігрової сесії складається з таких етапів:

1. ініціалізація гри при старті кімнати;
2. створення екземпляра ігрового стану;
3. обробка дій гравців у реальному часі;
4. оновлення стану та синхронізація;

5. завершення гри та видалення стану з пам’яті.

Після завершення гри відповідний об’єкт видаляється з Map, що звільняє ресурси сервера.

## 2.6 Реалізація клієнтської частини веб-застосунку

Клієнтська частина розроблюваної системи відповідає за взаємодію користувача з ігровим середовищем, відображення стану кімнати та ігор, а також обробку подій у режимі реального часу. Основною метою реалізації є забезпечення реактивного інтерфейсу, який синхронізується із сервером без необхідності ручного оновлення сторінки.

Для реалізації клієнтської частини використано бібліотеку React, яка дозволяє будувати компонентну архітектуру та ефективно керувати станом інтерфейсу.

### 2.6.1 Архітектура клієнтської частини

Архітектура клієнтської частини базується на трьох основних рівнях:

- рівень маршрутизації (TanStack Router);
- рівень стану (React Context Providers);
- рівень комунікації із сервером (Socket.IO Client та HTTP API).

Такий поділ дозволяє логічно ізолювати відповідальності та спростити масштабування застосунку.

### 2.6.2 Управління станом застосунку

Для зберігання глобального стану використовується механізм React Context, який реалізує наступні домени:

- стан користувача (Player);
- стан кімнати (Room);
- стан активної гри (Game State).

Кожен із контекстів відповідає за власну частину даних і надає методи для їх оновлення. Такий підхід дозволяє уникнути надмірного “prop drilling” та спрощує доступ до спільних даних у будь-якому компоненті дерева.

Особливістю реалізації є те, що оновлення стану відбувається як через HTTP-запити, так і через WebSocket-події, що забезпечує актуальність даних у реальному часі.

### 2.6.3 Взаємодія із сервером у реальному часі

Для реалізації двонаправленої комунікації використовується Socket.IO Client. Після встановлення з’єднання клієнт приєднується до відповідної кімнати та починає отримувати події від сервера.

Основні типи подій:

- оновлення кімнати (update-room);
- оновлення стану гри;
- системні повідомлення про помилки;
- події входу та виходу гравців.

Обробка цих подій виконується через підписку на socket events у відповідних React hooks, що дозволяє синхронізувати серверний стан із локальним станом застосунку.

### 2.6.4 Реактивне оновлення стану

Ключовою особливістю клієнтської частини є реактивна модель оновлення даних. При отриманні події від сервера відповідний контекст оновлює свій стан, що автоматично призводить до повторного рендерингу компонентів, які залежать від цих даних.

Таким чином, користувацький інтерфейс завжди відображає актуальний стан системи без необхідності додаткових запитів.

### 2.6.5 Маршрутизація та організація сторінок

Для організації навігації використовується TanStack Router, який дозволяє будувати вкладену структуру маршрутів. Це особливо важливо для реалізації ігрових сцен, де:

- кімната є контейнером;
- гра є вкладеним маршрутом;
- окремі ігри реалізуються як незалежні сторінки.

Такий підхід дозволяє чітко розділити логіку кімнати та конкретних ігор.

### 2.6.6 Використання WebSocket у клієнтській архітектурі

Socket-з’єднання ініціалізується на рівні окремого модуля та використовується в різних частинах застосунку через загальний екземпляр. Це дозволяє уникнути дублювання підключень та забезпечує єдине джерело подій.

Важливим аспектом є синхронізація ініціалізації socket із даними користувача, оскільки авторизаційні дані передаються під час встановлення з’єднання.

### 2.6.7 Обробка помилок та станів завантаження

Клієнтська частина також передбачає обробку асинхронних станів, зокрема:

- завантаження даних користувача;
- завантаження інформації про кімнату;
- обробка помилок API та socket-з’єднань.

Це дозволяє уникнути некоректного відображення інтерфейсу та підвищує стабільність застосунку.

## 2.7 Висновки до другого розділу

У другому розділі кваліфікаційної роботи було здійснено проєктування та реалізацію програмного забезпечення, орієнтованого на організацію багатокористувацьких ігрових сесій у веб-середовищі. Основна увага була

зосереджена на побудові архітектури системи, яка забезпечує розділення відповідальностей між підсистемою керування кімнатами, ігровим ядром та клієнтською частиною.

Було визначено, що ключовим елементом системи є кімната як логічний контейнер для користувачів, яка не містить ігрової логіки, а лише забезпечує координацію учасників та передачу ідентифікатора активної гри. Такий підхід дозволив забезпечити слабке зв’язування між компонентами системи та створити основу для подальшого масштабування.

Реалізація клієнтської частини базується на компонентній архітектурі React із використанням контекстів для управління глобальним станом та WebSocket-з’єднань для синхронізації даних із сервером. Це дозволило забезпечити реактивне оновлення інтерфейсу та мінімізувати затримки при відображенні змін ігрового стану.

Використання Socket.IO як основного механізму взаємодії в режимі реального часу забезпечило двонаправлений обмін подіями між клієнтом та сервером, що є критично важливим для багатокористувацьких ігрових застосунків. Запропонована архітектура дозволяє легко розширювати систему новими іграми без внесення суттєвих змін у вже реалізовані компоненти.

Таким чином, у другому розділі було реалізовано повноцінну мультисервісну архітектуру веб-застосунку, яка поєднує керування кімнатами, ігрову логіку та клієнтську взаємодію в єдину систему, здатну підтримувати інтерактивні багатокористувацькі сценарії в реальному часі.

# 3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

У даному розділі розглядаються процеси тестування, впровадження та подальшої експлуатації розробленого програмного забезпечення. Оскільки система побудована на основі клієнт-серверної архітектури з використанням WebSocket-з'єднань, особливу увагу приділено перевірці коректності взаємодії між компонентами в режимі реального часу, стабільності роботи під час багатокористувацького навантаження та узгодженості стану між серверною та клієнтською частинами.

## 3.1 Стратегія тестування системи

Тестування розробленого програмного забезпечення здійснювалося на кількох рівнях, що дозволило комплексно перевірити коректність роботи як окремих модулів, так і всієї системи в цілому.

У межах проєкту було застосовано наступні типи тестування:

- модульне тестування серверної логіки;
- інтеграційне тестування взаємодії між сервісами;
- тестування WebSocket-комунікації;
- функціональне тестування клієнтської частини;
- ручне сценарне тестування ігрових процесів.

Особливу увагу було приділено тестуванню подієвої взаємодії, оскільки саме WebSocket є основним механізмом синхронізації стану між користувачами.

## 3.2 Тестування серверної частини

Серверна частина системи тестувалась на рівні бізнес-логіки, що включає обробку кімнат, керування гравцями та ігрові стани.

Основними сценаріями тестування були:

- створення та видалення кімнат;

- приєднання та вихід користувачів;
- перевірка обмеження максимальної кількості гравців;
- перевірка прав доступу хоста;
- ініціалізація та завершення ігрових сесій.

Додатково перевірялась коректність роботи обробників помилок. У випадках некоректних запитів сервер повертає структуроване повідомлення про помилку або надсилає відповідну подію через WebSocket.

Важливим аспектом тестування було підтвердження того, що ігрова логіка не залежить від підсистеми кімнат, що відповідає принципам модульної архітектури.

### 3.3 Тестування WebSocket взаємодії

Оскільки система базується на обміні подіями в режимі реального часу, було проведено окреме тестування WebSocket-з’єднань.

Перевірялись такі аспекти:

- коректність підключення та відключення клієнтів;
- приєднання до кімнати через socket room;
- розповсюдження подій між усіма учасниками;
- синхронізація стану гри між клієнтами;
- обробка одночасних подій від різних користувачів.

Особливу увагу приділено перевірці ситуацій конкурентного доступу, коли кілька користувачів одночасно виконують дії в межах однієї ігрової сесії. Було підтверджено, що серверна логіка коректно обробляє такі сценарії без порушення консистентності стану.

### 3.4 Тестування клієнтської частини

Клієнтська частина тестувалась у браузерному середовищі з акцентом на перевірку реактивного оновлення інтерфейсу та коректності відображення стану гри.

Основні сценарії включали:

- відображення списку гравців у кімнаті;
- оновлення стану кімнати після подій WebSocket;
- коректну маршрутизацію між сторінками;
- синхронізацію стану гри після дій користувача;
- обробку станів завантаження та помилок.

Додатково перевірено роботу контекстів React, які забезпечують глобальне збереження стану користувача, кімнати та активної гри.

### 3.5 Навантажувальне тестування

Для оцінки стабільності системи було проведено базове навантажувальне тестування, яке імітувало одночасне підключення декількох клієнтів до однієї кімнати та виконання ними ігрових дій.

Результати показали, що система здатна коректно обробляти одночасні події без втрати даних та розсинхронізації стану.

Завдяки використанню in-memory моделі збереження стану та ефективної обробки подій через WebSocket, затримки оновлення інтерфейсу залишаються мінімальними навіть при паралельній взаємодії користувачів.

### 3.6 Впровадження системи

Впровадження програмного забезпечення здійснюється у вигляді клієнт-серверного застосунку, який може бути розгорнутий на одному сервері без необхідності складної інфраструктури.

Серверна частина працює як Node.js застосунок, що слухає HTTP та WebSocket-з’єднання на одному порту, тоді як клієнтська частина розгортається як статичний веб-застосунок.

Основні етапи впровадження:

- встановлення залежностей;

- запуск серверної частини;
- збірка клієнтського застосунку;
- підключення клієнта до сервера через WebSocket.

### 3.7 Підтримка та розширення системи

Архітектура системи спроєктована таким чином, щоб забезпечити просту інтеграцію нових ігрових модулів без внесення змін у базову логіку кімнат.

Додавання нової гри передбачає:

- створення нового класу ігрового стану;
- реалізацію socket handler’ів для обробки подій;
- додавання ідентифікатора гри до переліку доступних режимів.

Таким чином, система є розширюваною та може використовуватися як універсальна платформа для різних типів ігор або інтерактивних сценаріїв.

### 3.8 Висновки до третього розділу

У третьому розділі було проведено тестування розробленого програмного забезпечення, описано підходи до перевірки його компонентів, а також розглянуто особливості впровадження та подальшої підтримки системи.

Результати тестування підтвердили коректність роботи основних модулів, стабільність WebSocket-комунікації та узгодженість стану між клієнтською і серверною частинами. Запропонована архітектура забезпечує можливість подальшого розширення функціональності без суттєвих змін у базовій структурі системи.

# 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

В цьому розділі необхідно проаналізувати важливі питання, котрі стосуються безпеки життєдіяльності та основ охорони праці.

## 4.1 Психологічні чинники небезпеки.

Психологічні чинники небезпеки є критично важливою складовою загальної системи безпеки життєдіяльності та охорони праці. Вони безпосередньо пов'язані з внутрішнім станом людини, її психічними процесами, індивідуальними рисами характеру, мотивацією та миттєвими емоційними реакціями. Усі ці фактори можуть спровокувати помилкові дії, критично знизити концентрацію уваги або призвести до усвідомленого чи неусвідомленого ігнорування правил безпеки. На відміну від технічних чи фізичних небезпек, які є видимими, матеріальними та піддаються інженерному контролю, психологічні чинники діють приховано й латентно. Проте саме вони, за світовою статистикою, стають першопричиною переважної більшості аварій, катастроф, травм та нещасних випадків на виробництві та в побуті, утворюючи так званий «людський фактор».

Серед базових психологічних чинників небезпеки особливе місце посідають стійкі індивідуальні властивості особистості, які формують її поведінковий базис. До них належать темперамент, характер, психічна рухливість та індивідуальна схильність до ризику. Наприклад, надмірно імпульсивні люди (холеричного типу) або особи з високим рівнем прихованої чи явної агресивності частіше приймають поспішні, необдумані та хаотичні рішення в екстремальних ситуаціях. Навпаки, надмірна флегматичність чи уповільненість реакції можуть завадити людині вчасно й адекватно зорієнтуватися під час раптової технічної відмови обладнання, де рахунок іде на секунди. Не менш небезпечним є такий прояв характеру, як хронічна недисциплінованість щодо загальноприйнятих правил, коли працівник свідомо ігнорує інструкції з безпеки та використання ЗІЗ (засобів індивідуального захисту), вважаючи їх зайвими, застарілими чи суто формальними.

Іншу велику групу становлять тимчасові або ситуативні психічні стани людини, які динамічно змінюються під впливом зовнішніх та внутрішніх обставин. Найпоширенішим і найпідступнішим серед них є втома та її хронічна форма — перевтома, а також сучасне явище професійного вигорання. Фізичне чи розумове виснаження суттєво уповільнює швидкість реакції, притупляє гостроту зору та слуху, послаблює функцію оперативної пам'яті, логічного та критичного мислення. У стані перевтоми людина переходить на рівень автоматичного функціонування, пропускає ледве помітні, але важливі сигнали небезпеки та припускається грубих помилок. До цієї ж категорії належить гострий та хронічний стрес, викликаний конфліктами в колективі, домашніми негараздами або фінансовими проблемами. Стрес тунельно звужує поле сприйняття («ефект тунельного зору»), змушуючи людину фокусуватися на внутрішніх переживаннях, а не на моніторингу безпеки робочого середовища.

Емоційні стани, такі як апатія, депресивні прояви або, навпаки, надмірне збудження, гіперактивність та ейфорія, однаковою мірою дезорганізують поведінку. Особливу загрозу на виробництві становить стан паніки, який виникає під час раптової аварійної ситуації за відсутності чіткого, доведеного до автоматизму алгоритму дій. Паніка повністю блокує кору головного мозку та логічне мислення, викликаючи або хаотичні, руйнівні рухи, або повне заціпеніння (ступор), що унеможливлює саморятування та надання допомоги іншим. Окремим, кримінально караним та найнебезпечнішим психофізіологічним чинником є стан алкогольного, наркотичного або токсичного сп'яніння, а також безконтрольний прийом деяких медикаментів (селативних, антигістамінних). Вони повністю викривлюють сприйняття реальності, відключають внутрішні психологічні гальма й викликають критично помилкове відчуття всемогутності, безкарності та безпеки.

Важливу роль відіграють також соціально-психологічні чинники та деформація мотиваційної сфери особистості. Часто небезпечна поведінка зумовлена хибною мотивацією. Це може бути прагнення заощадити час чи фізичні зусилля за рахунок пропуску «нудних» технологічних етапів безпеки, або ж хлопчаче бажання продемонструвати колегам свою «сміливість» та вищу

професійну майстерність через ризиковані трюки (так званий «синдром героя»). Негативний, токсичний мікроклімат у колективі, тиск з боку лінійного керівництва, яке вимагає виконання плану чи норми за будь-яку ціну, або повна відсутність базової культури безпеки на підприємстві формують у працівника байдуже, фаталістичне ставлення до власного життя та здоров'я, перетворюючи дрібні порушення норм на повсякденну норму життя.

Для ефективної мінімізації впливу психологічних чинників небезпеки на сучасних підприємствах впроваджується комплексний підхід: професійний психологічний відбір (тестування на етапі найму), регулярні інтерактивні інструктажі, тренінги на симуляторах з відпрацювання дій у критичних ситуаціях та обов'язкові щозмінні медичні й психофізіологічні огляди. Створення ергономічних та комфортних умов праці, оптимізація режимів роботи й відпочинку, впровадження кімнат психологічного розвантаження, а також формування здорового, підтримуючого психологічного клімату дозволяють суттєво знизити рівень емоційного напруження. Глибоке розуміння психології безпеки допомагає кожній людині розвивати внутрішній самоконтроль, вчасно розпізнавати перші ознаки власної втоми чи стресу та свідомо обирати безпечні, зрілі моделі поведінки в будь-якій життєвій чи виробничій ситуації.

### 4.2 Загальні вимоги безпеки з охорони праці для користувачів ПК.

З масовим і тотальним впровадженням комп'ютерної техніки в усі сфери професійної, освітньої та повсякденної діяльності виникла гостра потреба в чіткому законодавчому та нормативному регулюванні умов праці офісних працівників та інших користувачів персональних комп'ютерів (ПК). Робота за комп'ютером, попри уявну легкість і комфорт порівняно з фізичною працею, пов'язана з низкою серйозних шкідливих і небезпечних виробничих чинників. Серед них: тривале статичне напруження великих груп м'язів, перевтома та перенапруження зорового аналізатора, тривалий вплив низькочастотного електромагнітного випромінювання, надмірна сухість повітря через іонізацію, а також високе

психоемоційне та монотонне напруження. Загальні вимоги безпеки з охорони праці покликані нівелювати ці ризики, зберегти здоров'я працівника, запобігти розвитку хронічних професійних захворювань та забезпечити стабільно високу ефективність праці протягом усього робочого дня.

Вимоги безпеки перед початком роботи детально визначають правила самостійної підготовки робочого місця. Користувач ПК зобов'язаний провести візуальний огляд обладнання: перевірити цілісність сполучних кабелів, надійність заземлення, стан електричних вилок, розеток, а також відсутність тріщин та деформацій на корпусах монітора, системного блока та периферії. Категорично забороняється вмикати техніку в електромережу за наявності видимих пошкоджень ізоляції, оголених дротів або специфічних ознак чи запаху кіптяви. Робочу поверхню столу необхідно звільнити від сторонніх предметів, чашок, зайвих паперів та особистого одягу. Вони не лише захаращують простір, а й можуть перекривати вентиляційні решітки комп'ютера, що викликає його перегрів та створює пожежну небезпеку. Також на цьому етапі слід індивідуально відрегулювати висоту крісла, кут нахилу монітора та положення клавіатури під власні антропометричні дані для забезпечення фізіологічно правильної постави.

Правильна ергономіка робочого місця під час безпосереднього виконання обов'язків є фундаментом профілактики травматизму та захворювань. Монітор має бути розташований строго навпроти обличчя на відстані 50–70 сантиметрів (відстань витягнутої руки) від очей користувача. Його верхня кромка повинна знаходитися на рівні очей або на 10–15 градусів нижче, щоб природний погляд був спрямований трохи зверху вниз — це забезпечує оптимальне зволоження ока повікою та запобігає перенапруженню м'язів шиї і виникненню головного болю напруги. Клавіатуру і мишу слід розміщувати на такій відстані, щоб передпліччя вільно спиралися на стіл або регульовані підлокітники крісла, а кисті рук залишалися рівними й не згиналися в суглобах занадто сильно. Це мінімізує статичне навантаження на сухожилля та блокує розвиток тунельного синдрому (синдрому зап'ястного каналу). Робоче стілець або крісло повинно мати надійну

п'ятипроменеву опору на колесах, анатомічну спинку, що підтримує поперековий вигин хребта, та регулювання по висоті й нахилу.

Раціональна організація режиму праці та відпочинку є критично важливою вимогою для профілактики зорового (комп'ютерного зорового синдрому) та загального соматичного виснаження. Безперервна інтенсивна робота за екраном ПК не повинна перевищувати двох годин поспіль. Законодавство та медичні рекомендації з охорони праці наполягають на влаштуванні регламентованих коротких перерв тривалістю 10–15 хвилин через кожну годину (при високій інтенсивності введення даних) або дві години роботи (при творчій чи аналітичній діяльності). Під час цих перерв працівнику суворо рекомендується повністю залишити монітор, встати з крісла та виконати спеціальний комплекс вправ для очей (часте кліпання, кругові рухи, почергове фокусування на далеких і близьких об'єктах), а також легку фізичну розминку (нахили, потягування, розминання кистей) для відновлення порушеного кровообігу в ногах, органах малого таза, спині та шийно-плечовому поясі.

Суворі вимоги до навколишнього середовища в робочому приміщенні регулюють параметри мікроклімату, чистоти повітря та освітлення. Офісні приміщення повинні мати збалансоване поєднання природного та штучного освітлення. Джерела світла та робочі столи слід розміщувати так, щоб на екранах моніторів не виникало прямих або дзеркальних відблисків, які змушують око постійно перенапружуватися. Оптимальна температура в приміщенні має підтримуватися в межах 22–25 градусів за Цельсієм за відносної вологості 40–60%. Важливою вимогою безпеки (і концепції Clean Desk) є заборона вживання їжі на робочому місці та розміщення відкритих ємностей з водою, кавою чи іншими напоями безпосередньо поруч із комп'ютером та клавіатурою.

Вимоги безпеки в аварійних ситуаціях чітко й безкомпромісно регламентують дії користувача у разі виникнення будь-якої загрози. При виявленні перших ознак несправності — появі запаху горілої ізоляції чи пластику, диму, стороннього гудіння, тріску або іскріння всередині системного блока чи монітора — працівник повинен негайно припинити будь-які операції, за можливості

безпечно знеструмити обладнання (витягнути вилку з розетки або вимкнути мережевий фільтр), терміново повідомити про інцидент свого безпосереднього керівника або системного адміністратора та, у разі потреби, викликати пожежну службу й розпочати евакуацію. У разі раптового погіршення власного самопочуття (поява різкого «піску» або болю в очах, сильне запаморочення, прискорене серцебиття, оніміння пальців рук), користувач ПК зобов'язаний призупинити роботу, довести до відома колег чи керівника про свій стан та звернутися по медичну допомогу, оскільки ігнорування перших симптомів може призвести до серйозних хронічних розладів здоров'я.

### 4.3. Висновки до четвертого розділу

Аналіз психологічних чинників небезпеки та нормативних вимог до роботи за комп'ютером дозволяє зробити загальний висновок про те, що сучасна система охорони праці зміщує свій фокус із суто технічних аспектів на захист та оптимізацію діяльності самої людини.

Обидва тексти наочно демонструють, що безпека життєдіяльності є комплексним процесом, де внутрішній (психоемоційний) стан працівника тісно пов'язаний із зовнішньою (фізичною та ергономічною) організацією його робочого простору. Нехтування ергономічними стандартами при роботі за ПК (наприклад, відсутність перерв чи неправильна посадка) неминуче призводить до фізичної втоми, яка, своєю чергою, виступає потужним психологічним чинником небезпеки — притупляє увагу, викликає дратівливість і провокує помилки.

Таким чином, лише поєднання високої особистої культури безпеки, психологічного самоконтролю та суворого дотримання гігієнічних та ергономічних вимог здатне мінімізувати ризики «людського фактора», захистити здоров'я працівника та створити безпечне й ефективне виробниче середовище.

# ВИСНОВКИ

При виконанні кваліфікаційної роботи освітнього рівня «Бакалавр» було розроблено програмне забезпечення на основі мультисервісної архітектури для інтерактивного багатокористувацького веб-сервісу. Основна ідея проєкту була у створенні системи кімнат, що була би інтерактивним середовищ для запуску різного ігрового контенту у режимі реального часу.

Під час роботи було встановлено, що підходи до реалізації веб-ігор, які вже існуючі, дуже часто бувають сильно пов’язаними з конкретною ігровою логікою, що зазвичай ускладнює масштабування та повторне використання коду. Запропонований тут підхід, навпаки, базується на чіткому розділенні відповідальностей сервісу між системами кімнат та модулями ігор, що дозволить створювати ігри, що не залежать одні від одних в рамках єдиної інфраструктури.

У першому розділі роботи було проаналізовано предметну область інтерактивних веб-застосунків та існуючі аналоги таких подібних систем і встановлено, що велика кількість рішень містять обмеження у вигляді закритої архітектури, орієнтації на конкретні сценарії використання або недостатньо гнучкої реалізації взаємодії багатьох користувачів. Також було розглянуто технології сучасної веб-розробки, а саме React, Node.js та WebSocket, які забезпечують можливість побудови хороших динамічних систем із підтримкою взаємодії у реальному часі.

У другому розділі роботи було здійснено реалізацію основних компонентів системи та їх проєктування. Також було розроблено підсистему для керування кімнатами, що забезпечує організацію користувачів у спільний для них ігровий простір, а також ігрове ядро, що було побудоване на основі in-memory моделей стану. Ще реалізовано механізм збереження ігрових сесій у вигляді окремих класів стану, що дозволяє зберегти логіку та незалежність ігрових процесів унікально для кожної гри. Клієнтська фронтендна частина побудована на основі фреймворку React із використанням контекстів та WebSocket-з’єднань для забезпечення синхронізації стану між користувачами у реальному часу.

У третьому розділі було проведено швидке тестування розробленої системи, перевірено чи працюють коректно основні функціональні модулі, взаємодію між клієнтською та серверною частинами, а також стабільність комунікації WebSocket в умовах багатокористувацької взаємодії. Результати тестування підтвердили працездатність системи та коректність реалізації основних сценаріїв використання. Також було розглянуто питання щодо впровадження програмного забезпечення та визначено конкретні підходи до його подальшого розширення.

Практична цінність роботи полягає у гнучкій архітектурі, яка дозволяє використовувати дану систему як основу для різних типів інтерактивних веб-застосунків та веб-сервісів. Запропоноване рішення також може бути розширене шляхом додавання нових ігор без внесення будь-яких змін у базову логіку керування кімнатами, що значно збільшує масштабованість і повторну використовуваність розробленої системи.

У результаті виконання даної кваліфікаційної роботи було досягнуто поставленої мети – спроєктовано та реалізовано мультисервісну архітектуру веб-застосунку, що забезпечить ефективну організацію багатокористувацьких сесій та підтримає інтерактивну взаємодію в режимі реального часу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі Михалик Д.М., Цуприк Г.Б., Бревус В.М. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. 45 с.

2. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. – Boston: Addison-Wesley, 1994. – 395 p.

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

4. Fowler M. *Patterns of Enterprise Application Architecture*. – Boston: Addison-Wesley, 2002. – 533 p.

5. Newman S. *Building Microservices: Designing Fine-Grained Systems*. – Sebastopol: O'Reilly Media, 2021. – 618 p.

6. Richardson C. *Microservices Patterns*. – Shelter Island: Manning Publications, 2018. – 520 p.

7. Mozilla Developer Network. WebSocket API [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/docs/Web/API/WebSocket

8. Mozilla Developer Network. HTTP Overview [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/docs/Web/HTTP/Overview

9. React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev

10. TanStack Router Documentation [Електронний ресурс]. – Режим доступу: https://tanstack.com/router

11. TanStack Query Documentation [Електронний ресурс]. – Режим доступу: https://tanstack.com/query

12. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/docs

13. Express.js Documentation [Електронний ресурс]. – Режим доступу: https://expressjs.com

14. Socket.IO Documentation [Електронний ресурс]. – Режим доступу: https://socket.io/docs

15. MongoDB Documentation [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/docs

16. Mongoose Documentation [Електронний ресурс]. – Режим доступу: https://mongoosejs.com/docs

17. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com

18. TypeScript Documentation [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs

19. Axios Documentation [Електронний ресурс]. – Режим доступу: https://axios-http.com/docs

20. Vitest Documentation [Електронний ресурс]. – Режим доступу: https://vitest.dev

21. Open Web Application Security Project (OWASP). OWASP Top 10 [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten

22. Sommerville I. *Software Engineering*. – 10th Edition. – Boston: Pearson, 2015. – 816 p.

23. Пістун І.П. Безпека життєдіяльності. Навчальний посібник. – Суми: вид. „Університет кн.”, 2000. 301 с.

24. Жидецький В.Ц., Джигирей В.С., Мельников О.В. Основи охорони праці. – Львів: Афіша, 2000. 350 с.

# ДОДАТКИ

# ДОДАТОК А
Тези конференції

*IX Міжнародна студентська науково - технічна конференція*
*"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"*

УДК 004.41
Карпець Д.–ст. гр. СП-41
*Тернопільський національний технічний університет імені Івана Пулюя*

## ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ МУЛЬТИСЕРВІСНОЇ АРХІТЕКТУРИ ДЛЯ ІНТЕЛЕКТУАЛЬНОГО НАВЧАЛЬНОГО КОНТЕНТУ

Науковий керівник: д.ф.-м.н., проф. Петрик М. Р.

Karpets D.
*Ternopil Ivan Puluj National Technical University*

## DESIGNING SOFTWARE BASED ON MICROSERVICES ARCHITECTURE FOR INTELLIGENT EDUCATIONAL CONTENT

Supervisor: PhD Petryk M. R.

Ключові слова: мультисервісна архітектура, інтелектуальний контент, освітні системи, мікросервіси, програмне забезпечення.

Keywords: microservices architecture, intelligent content, educational systems, microservices, software design.

**Вступ.** Сучасні освітні платформи активно впроваджують інтелектуальні технології для адаптації навчального контенту під потреби користувачів. Це зумовлює необхідність створення гнучких, масштабованих та ефективних програмних систем. Мультисервісна архітектура (microservices) є одним із ключових підходів до розробки таких систем, оскільки дозволяє розділити функціональність на незалежні сервіси, що спрощує розробку, тестування та масштабування[1].

Особливістю інтелектуального навчального контенту є необхідність обробки великих обсягів даних, персоналізації матеріалів та інтеграції з різними сервісами, такими як системи рекомендацій, аналітики та оцінювання знань. Це вимагає застосування сучасних архітектурних підходів.

Мета роботи – дослідження принципів проєктування програмного забезпечення на основі мультисервісної архітектури для створення інтелектуального навчального контенту.

**Основна частина.** Мультисервісна архітектура передбачає розбиття системи на набір незалежних сервісів, кожен із яких виконує окрему функцію. Це дозволяє підвищити гнучкість системи та забезпечити її масштабованість.

Ключові аспекти проєктування включають:

1. Декомпозиція системи. Поділ функціональності на окремі сервіси, такі як управління користувачами, генерація контенту, аналітика та рекомендаційні системи. Це дозволяє кожному сервісу розвиватися незалежно.
2. Використання API. Взаємодія між сервісами здійснюється через чітко визначені інтерфейси (REST, gRPC), що забезпечує незалежність компонентів та спрощує інтеграцію.
3. Масштабованість. Кожен сервіс може масштабуватися окремо залежно від навантаження, що є важливим для систем із великою кількістю користувачів.
4. Використання контейнеризації. Технології, такі як Docker та Kubernetes, дозволяють ефективно розгортати та керувати мікросервісами[2].

5. Інтелектуалізація контенту. Застосування алгоритмів машинного навчання для персоналізації навчального матеріалу, аналізу поведінки користувачів та формування рекомендацій.

Особливу увагу приділено забезпеченню узгодженості даних між сервісами та обробці запитів у реальному часі. Використання черг повідомлень (наприклад, Kafka або RabbitMQ) дозволяє реалізувати асинхронну взаємодію між компонентами системи.

Крім того, важливим аспектом є забезпечення надійності системи, що досягається за рахунок відмовостійкості сервісів, балансування навантаження та моніторингу.

**Результати**. У результаті дослідження визначено основні підходи до проєктування програмного забезпечення на основі мультисервісної архітектури. Їх застосування дозволяє:

- підвищити масштабованість та гнучкість системи;
- забезпечити ефективну обробку навчального контенту;
- реалізувати персоналізацію навчання;
- спростити супровід та розвиток програмного забезпечення;

**Висновки**. У роботі досліджено особливості проєктування програмного забезпечення на основі мультисервісної архітектури для інтелектуального навчального контенту. Показано, що використання мікросервісів дозволяє створювати масштабовані, гнучкі та ефективні освітні системи, здатні адаптуватися до потреб користувачів.

**Список використаних джерел:**

[1] S. Newman, Building Microservices, 2nd ed., O'Reilly Media, 2021.

[2] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Kubernetes: Up and Running, 3rd ed., O'Reilly Media, 2022.

[3] M. Richards, N. Ford, Fundamentals of Software Architecture, O'Reilly Media, 2020.

# ДОДАТОК Б

https://github.com/TriggeerCat/Games-Webpage