МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ВЕБПЛАТФОРМА ДЛЯ ПРОВЕДЕННЯ ОНЛАЙН-ВІКТОРИН У РЕАЛЬНОМУ ЧАСІ З МЕХАНІКАМИ ГЕЙМІФІКАЦІЇ

Виконав: студент групи 1П-22
Спеціальності
121 Інженерія програмного забезпечення
Федір МАРЕНИЧ
Керівник:
Станіслав МАРЧЕНКО

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувачка циклової комісії КІ та ІТ

_____ Наталя ФАЛЬЧЕНКО
(підпис)

«_____» _____2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Мареничу Федору Анатолійовичу

1. Тема кваліфікаційної роботи Вебплатформа для проведення онлайн-вікторин у реальному часі з механіками гейміфікації

Керівник роботи Марченко Станіслав Віталійович, викладач першої категорії

затверджені наказом закладу фахової передвищої освіти від «06» жовтня 2025 року № 84/1-у.

2. Строк подання студентом кваліфікаційної роботи 02.06.2026

3. Вихідні дані до кваліфікаційної роботи: середовище виконання .NET 8; вебфреймворк ASP.NET Core; бібліотека ASP.NET Core SignalR для реалізації взаємодії в реальному часі; Entity Framework Core для об'єктно-реляційного відображення та роботи з базою даних; система керування базами даних PostgreSQL; механізм JWT-авторизації; IMemoryCache для зберігання оперативного стану активної ігрової сесії; мови HTML, CSS і JavaScript для реалізації клієнтської частини; Swagger UI для перевірки та документування REST API; xUnit для модульного тестування бізнес-логіки; Docker і Docker Compose для контейнеризованого розгортання програмного продукту; Git як система контролю версій; PlantUML для побудови UML-діаграм.

4. Зміст кваліфікаційної роботи: огляд предметної області (характеристика предметної області; технологічний контекст розробки платформи; аналіз наявних рішень; постановка задачі на розроблення); проєктування та реалізація вебплатформи Real-Time Quiz Hub (специфікація вимог до програмного

забезпечення; концептуальне моделювання системи; попередня архітектура програмного забезпечення; моделювання даних для програмного забезпечення; реалізація програмного продукту); тестування, розгортання та супровід вебплатформи Real-Time Quiz Hub (стратегія та план тестування; модульне тестування; функціональне тестування REST API; розгортання та супроводження програмного продукту).

5. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	14.10.2025	
2	Розділ 1. Огляд предметної області	09.12.2025	
3	Розділ 2. Проєктування та реалізація вебплатформи Real-Time Quiz Hub	10.03.2026	
4	Розділ 3. Тестування, розгортання та супровід вебплатформи Real-Time Quiz Hub	28.04.2026	
5	Висновки	12.05.2026	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2026	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2026	
8	Подання кваліфікаційної роботи на затвердження завідувачці циклової комісії (за 7 днів до захисту)	10.06.2026	

Студент _____ Федір МАРЕНИЧ
(підпис)

Керівник роботи _____ Станіслав МАРЧЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційну роботу присвячено розробленню вебплатформи Real-Time Quiz Hub, призначеної для створення, проведення та адміністрування онлайн-вікторин у режимі реального часу. Актуальність теми зумовлена потребою закладів освіти, тренінгових центрів і організацій у доступному програмному рішенні для інтерактивного оцінювання знань, яке не залежить від обмежень безкоштовних тарифів комерційних сервісів і може бути адаптоване до україномовного середовища.

У роботі проаналізовано предметну область онлайн-вікторин, типові сценарії їх використання, механіки гейміфікації та технології забезпечення синхронної взаємодії користувачів. Розглянуто функціональні обмеження популярних платформ Kahoot!, Quizizz та AhaSlides, що дало змогу сформулювати вимоги до власного програмного продукту. Особливу увагу приділено підтримці ігрових сесій, таблиці лідерів, гостьової участі, авторизації користувачів, адміністративного керування контентом і збереження результатів.

Програмний продукт реалізовано з використанням ASP.NET Core, SignalR, Entity Framework Core, PostgreSQL, JWT-авторизації та клієнтської частини на HTML, CSS і JavaScript. Для передавання подій у реальному часі використано SignalR-хаб, а для зберігання оперативного стану активної сесії застосовано in-memory-підхід. Постійні дані системи зберігаються в реляційній базі PostgreSQL. Реалізовано функції створення й проходження вікторин, надсилання відповідей, оновлення таблиці лідерів, нарахування балів, перегляду результатів і керування запитаннями та відповідями через адміністративний інтерфейс. Платформа має модульну архітектуру, підтримує подальше розширення функціональності та може бути розгорнута в контейнеризованому середовищі.

Ключові слова: ОНЛАЙН-ВІКТОРИНА, ВЕБПЛАТФОРМА, SIGNALR, ГЕЙМІФІКАЦІЯ, WEBSOCKET, ASP.NET CORE, POSTGRESQL, JWT.

ANNOTATION

The qualification paper focuses on the development of Real-Time Quiz Hub, a web platform designed for creating, running, and administering real-time online quizzes. The relevance of the topic is determined by the need for an accessible software solution that can support interactive knowledge assessment in educational institutions, training centers, and organizations without relying on the limitations of free commercial services and with the possibility of adaptation to a Ukrainian-language environment.

The paper examines the domain of online quiz platforms, typical usage scenarios, gamification mechanics, and technologies for synchronous user interaction. The functional limitations of popular platforms such as Kahoot!, Quizizz, and AhaSlides are analyzed in order to define the requirements for the proposed software product. Special attention is paid to live quiz sessions, leaderboards, guest participation, user authorization, administrative content management, and result storage.

The software product is implemented using ASP.NET Core, SignalR, Entity Framework Core, PostgreSQL, JWT-based authorization, and a client-side interface based on HTML, CSS, and JavaScript. A SignalR hub is used to transmit real-time events, while the operational state of an active session is stored in memory. Persistent system data are stored in a PostgreSQL relational database. The implemented functionality includes quiz creation and participation, answer submission, leaderboard updates, score calculation, result viewing, and administrative management of questions and answers. The platform has a modular architecture, supports further functional extension, and can be deployed in a containerized environment.

Keywords: ONLINE QUIZ, WEB PLATFORM, SIGNALR, GAMIFICATION, WEBSOCKET, ASP.NET CORE, POSTGRESQL, JWT.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

Real-Time Quiz Hub	Вебплатформа онлайн-вікторин у реальному часі, розроблена в межах кваліфікаційної роботи для створення, проведення та адміністрування вікторин із механіками гейміфікації.
MVP	Minimum Viable Product – мінімально життєздатний продукт. Початкова функціонально завершена версія системи, що реалізує основні можливості платформи.
FR	Functional Requirement — функціональна вимога. Описує конкретну дію або функцію, яку повинна виконувати система.
NFR	Non-Functional Requirement – нефункціональна вимога. Визначає якісні характеристики системи: продуктивність, безпеку, надійність, масштабованість, зручність використання.
UC	Use Case – прецедент використання. Типовий сценарій взаємодії користувача із системою.
RTM	Requirements Traceability Matrix – матриця відстежуваності вимог. Таблиця, що пов’язує вимоги з тест-кейсами та результатами перевірки.
TC	Test Case – тестовий випадок. Формалізований сценарій перевірки функції або вимоги системи.
JWT	JSON Web Token – вебтокен у форматі JSON. Використовується для автентифікації та авторизації користувачів у системі.
SignalR	ASP.NET Core SignalR – бібліотека для реалізації двосторонньої комунікації між сервером і клієнтами в режимі реального часу.
WebSocket	Мережевий протокол для постійного двостороннього з’єднання між клієнтом і сервером. Використовується для передавання подій у реальному часі.
SSE	Server-Sent Events – механізм передавання подій від сервера до клієнта, який може використовуватися як резервний транспорт у SignalR.

ASP.NET Core	Кросплатформовий вебфреймворк Microsoft, використаний для реалізації серверної частини платформи.
EF Core	Entity Framework Core – ORM-фреймворк для роботи з базою даних через об’єктну модель.
ORM	Object-Relational Mapping – об’єктно-реляційне відображення. Підхід, за якого таблиці бази даних зіставляються з класами програмної моделі.
IMemoryCache	Кеш у пам’яті процесу ASP.NET Core. У MVP використовується для зберігання оперативного стану активної ігрової сесії.
DI	Dependency Injection – впровадження залежностей. Механізм передавання залежностей через контейнер сервісів.
PBL	Points-Badges-Leaderboards – бали, відзнаки, таблиці лідерів. Базовий набір механік гейміфікації.
ADR	Architecture Decision Record — запис архітектурного рішення, у якому фіксується важливе технічне рішення, його обґрунтування та наслідки.

ВСТУП

У сучасному освітньому та корпоративному середовищі залученість учасників до навчального процесу є одним із ключових чинників його ефективності. Традиційні форми оцінювання знань – паперові тести, усні опитування, лекційні контрольні – дедалі частіше поступаються місцем інтерактивним цифровим форматам, здатним підтримувати мотивацію та зворотний зв’язок у режимі реального часу. Онлайн-вікторини з елементами гейміфікації стали одним із найбільш задокументованих інструментів такого типу: систематичні дослідження фіксують переміщення середнього результату учнів із 50-го до 72-го перцентиля при їх регулярному застосуванні [1; 2], а окремі дослідження у медичній освіті виявили зростання академічних показників до 45% [3].

Попри широку представленість комерційних рішень у цьому сегменті, практика їх використання в українських освітніх закладах і організаціях стикається з низкою об’єктивних обмежень. Провідні платформи – Kahoot!, Quizizz та AhaSlides – обмежують кількість учасників у безкоштовній версії до 50–100 осіб, не підтримують відкритих типів запитань без платної підписки, не мають україномовного інтерфейсу та позбавлені адміністративних інструментів, необхідних для корпоративного тестування. Це змушує освітні установи або оплачувати підписки з обмеженим бюджетом, або відмовлятися від цих інструментів на користь менш ефективних форматів.

Означена ситуація зумовлює актуальність розробки відкритої вебплатформи, позбавленої штучних обмежень комерційних моделей і адаптованої до потреб україномовної аудиторії. При цьому технологічне завдання є нетривіальним: синхронна взаємодія десятків учасників у реальному часі вимагає відповідної транспортної інфраструктури, а поєднання гейміфікованого досвіду з адміністративними функціями потребує виваженого архітектурного рішення.

Об’єктом дослідження є процеси організації та проведення інтерактивного оцінювання знань у режимі реального часу засобами вебтехнологій.

Предметом дослідження є методи та технології проєктування і реалізації вебплатформи для онлайн-вікторин із механіками гейміфікації.

Метою кваліфікаційної роботи є проєктування та реалізація вебплатформи Real-Time Quiz Hub для проведення онлайн-вікторин у реальному часі з механіками гейміфікації, що підтримує синхронний і асинхронний режими, різні типи запитань та адміністративні інструменти управління.

Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати наявні платформи для проведення інтерактивних вікторин, виявити їхні функціональні та ліцензійні обмеження і на цій основі сформулювати вимоги до власного рішення;

– дослідити механіки гейміфікації та підходи до організації синхронної взаємодії учасників у режимі реального часу;

– спроєктувати архітектуру вебплатформи, що забезпечує одночасну роботу значної кількості учасників та підтримує синхронний і асинхронний режими проведення вікторин;

– реалізувати функціональність створення та проведення вікторин із різними типами запитань, системою нарахування балів і таблицею лідерів у реальному часі;

– реалізувати адміністративні інструменти управління користувачами, вікторинами та результатами тестування;

– провести тестування розробленої платформи та оцінити відповідність отриманого рішення сформульованим вимогам.

Практична цінність роботи полягає в тому, що розроблена платформа Real-Time Quiz Hub може застосовуватися закладами освіти, тренінговими центрами та організаціями для проведення інтерактивного оцінювання знань без обмежень на кількість учасників і типи запитань, притаманних безкоштовним версіям комерційних аналогів. Україномовний інтерфейс та вбудовані адміністративні інструменти роблять платформу придатною як для навчального, так і для

корпоративного тестування, а відкритість рішення дає змогу адаптувати його до специфічних потреб конкретної установи без додаткових ліцензійних витрат.

Апробація результатів, представлених у межах кваліфікаційної роботи, здійснювалась на XVIII Студентській науково-практичній конференції студентів, аспірантів та молодих вчених «Тенденції розвитку ІТ-технологій в Україні» [4].

РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика предметної області

Стрімкий розвиток інформаційно-комунікаційних технологій зумовив виникнення нового класу програмних засобів для організації інтерактивного навчання та оцінювання знань. Одним із таких засобів є онлайн-вікторина – програмний продукт, що забезпечує формування наборів запитань, їх послідовне пред’явлення учасникам через веббраузер або мобільний застосунок, автоматичну перевірку відповідей і відображення результатів. На відміну від паперових тестів, онлайн-вікторини уможливлюють одночасну участь географічно розподілених користувачів, автоматизоване підрахування балів та миттєве надання зворотного зв’язку.

Невід’ємною характеристикою сучасних вікторинних платформ є режим реального часу. Відповідно до ДСТУ 2226-93 «Автоматизовані системи. Терміни та визначення», режим реального часу визначається як режим обробки інформації, за якого затримка між введенням даних і отриманням результату не перевищує часу, допустимого для керування технологічним процесом або для взаємодії з користувачем [5]. У контексті вікторинних платформ це означає, що відповідь учасника, відображення таймера зворотного відліку та оновлення рейтингової таблиці мають відбуватися з затримкою, що не є відчутною для людського сприйняття, тобто в межах 100–300 мілісекунд.

Другим ключовим концептом предметної області є гейміфікація. Автори [6] визначають гейміфікацію як застосування елементів дизайну ігор та ігрових принципів у неігровому контексті з метою підвищення залученості, мотивації та досягнення освітніх або організаційних цілей. До типових механік гейміфікації відносять нарахування балів, присвоєння відзнак (badges), відображення рейтингових таблиць (leaderboards), системи рівнів і прогресу, а також елементи змагальності.

Поєднання онлайн-формату та механік гейміфікації породило окремий клас програмних рішень – інтерактивні вікторинні платформи реального часу,

представниками яких є Kahoot!, Quizizz, Mentimeter, AhaSlides тощо. У галузі формальної освіти вікторинні платформи використовуються на рівні початкової та середньої школи для поточного оцінювання, повторення матеріалу та контролю засвоєння тем. У вищій школі вони застосовуються для активізації аудиторії під час лекцій, організації самоперевірки студентів та проведення рубіжного контролю. Особливого значення ці інструменти набули у системах дистанційного навчання, де підтримання залученості студентів є одним із критичних завдань педагогічного проєктування [7].

У галузі корпоративного навчання вікторинні платформи задіяні в процесах онбордингу нових співробітників, обов'язкового compliance-тестування (перевірки знань внутрішніх регламентів, норм охорони праці, антикорупційних вимог), а також щорічної атестації персоналу. У сфері розваг та масових заходів вікторини використовуються як формат командних ігор, квізів та інтерактивних розминок. Тут домінують вимоги до масштабованості, стабільності з'єднання та видовищності відображення результатів.

Важливість онлайн-вікторинних платформ підтверджується значним масивом незалежних наукових досліджень. Зокрема, за даними офіційної сторінки досліджень Kahoot!, платформа є предметом понад 200 незалежних рецензованих наукових праць, виконаних дослідниками з понад 50 країн світу [2]. Це свідчить про те, що Kahoot! перетворився не лише на комерційний продукт, а й на визнаний об'єкт педагогічних досліджень.

Серед конкретних задокументованих результатів застосування таких платформ виділяються дані двох незалежних клінічних досліджень у медичній освіті. Автори [8], проводячи дослідження серед студентів-медиків Університету Майо та Університету Уейна (США), встановили, що 93% учасників надали перевагу інтерактивним заняттям із використанням Kahoot! порівняно з традиційними лекціями, а 90% висловили бажання застосовувати платформу в інших навчальних дисциплінах. Ці показники є статистично значущими та демонструють суттєвий перерозподіл навчальних уподобань студентів на користь гейміфікованого формату.

Паралельне дослідження, проведене в [3] серед студентів медичного факультету в Туреччині, виявило статистично достовірний зв’язок між регулярним використанням Kahoot! у навчальному процесі та зростанням академічних результатів на підсумкових іспитах на 45%. Автори пов’язують цей ефект зі зростанням мотивації до самостійного повторення матеріалу, яке стимулюється змагальним форматом платформи.

Сукупність наведених даних підтверджує, що онлайн-вікторини з механіками гейміфікації є не лише технологічним трендом, а й педагогічно обґрунтованим інструментом підвищення якості навчання в різних контекстах. Для розуміння вимог до програмної реалізації такої платформи необхідно детально розглянути типові сценарії її використання.

1.1.1 Типові сценарії використання вікторинних платформ

Проведений раніше аналіз предметної області засвідчує, що онлайн-вікторинні платформи застосовуються в принципово різних організаційних та технічних контекстах. Кожен з таких контекстів формує специфічний набір вимог до функціональності, продуктивності та зручності використання системи. З метою систематизації цих вимог було виокремлено та детально описано шість типових сценаріїв використання онлайн-вікторин.

Сценарій 1 – синхронна вікторина у класі. Основними акторами в цьому сценарії є вчитель або викладач, що виступає у ролі ведучого, та учні або студенти як учасники. Вікторина проводиться в умовах спільного фізичного простору: ведучий демонструє запитання на проєкторі або інтерактивній дошці, а учасники відповідають зі своїх індивідуальних пристроїв: смартфонів, планшетів або ноутбуків. Використання спільного екрана для відображення запитань усуває потребу в читанні з малих екранів і підтримує спільну увагу аудиторії [9]. Метою сценарію є швидке повторення або поточне оцінювання вивченого матеріалу у форматі, що стимулює конкурентну мотивацію. Ведучий запускає сесію, учасники приєднуються за кодом або посиланням, після чого

запитання відображаються синхронно для всіх. Система нараховує бали з урахуванням не лише правильності, а й швидкості відповіді.

Вимоги до системи в межах цього сценарію включають підтримку одночасного підключення від 30 до 100 учасників без деградації продуктивності; відображення стану таймера та рейтингової таблиці в реальному часі; гостьовий вхід учасників без попередньої реєстрації. Критичним обмеженням є стабільність з'єднання в умовах шкільної або університетської Wi-Fi-мережі, де якість сигналу може суттєво варіюватися.

Сценарій 2 – дистанційна синхронна вікторина. Актори залишаються тими ж, однак сценарій відрізняється тим, що всі учасники перебувають у різних фізичних локаціях та підключаються до вікторини паралельно з участю у відеоконференції (Zoom, Google Meet, Microsoft Teams). Запитання відображаються безпосередньо на пристроях учасників, а не на спільному екрані. Метою є реалізація інтерактивного елементу дистанційного заняття з підтриманням залученості аудиторії. Цей формат набув значного поширення під час і після пандемії COVID-19, коли освітні заклади масово перейшли на дистанційне навчання. Технічна складність полягає в тому, що навантаження на мережу учасника подвоюється: одночасно підтримуються з'єднання з сервісом відеоконференцій та вікторинною платформою.

Вимоги до системи: мінімальний обсяг трафіку від клієнта до сервера вікторини (оскільки частина пропускної здатності витрачається на відеоконференцію); адаптивний інтерфейс для різних розмірів екранів. Базове обмеження – відсутність можливості спільного екрана для відображення запитань, тому запитання мають бути чітко відображені на пристрої кожного учасника.

Сценарій 3 – асинхронне домашнє завдання з дедлайном. Актори: викладач, що формує завдання, та студенти, що проходять його самостійно. На відміну від попередніх сценаріїв, тут відсутня синхронна сесія: викладач публікує вікторину з визначеним дедлайном, а кожен учасник проходить її у зручний для себе час до настання терміну. Метою є організація самостійної

роботи та діагностика рівня засвоєння матеріалу перед контрольним заходом. Цей режим принципово відрізняється від синхронного відсутністю таймера відповіді та рейтингової таблиці в реальному часі, натомість акцент зміщується на детальний зворотний зв’язок після завершення.

Вимоги до системи: підтримка режиму асинхронного проходження; встановлення та автоматичне відстеження дедлайну; збереження та накопичення результатів усіх учасників; відображення агрегованої аналітики для викладача після завершення терміну. Базове обмеження – необхідність попередньої реєстрації учасників або механізму ідентифікації для коректного збереження результатів.

Сценарій 4 – корпоративне навчання та compliance-тестування. Актори: HR-менеджер або адміністратор навчання, що формує тести та керує обліковими записами, а також співробітники, що проходять обов’язкову атестацію. Цей сценарій характеризується формальним характером тестування: результати мають юридичне або адміністративне значення – підтвердження проходження інструктажу, допуск до виконання певних функцій. Метою є автоматизована перевірка відповідності знань співробітників корпоративним стандартам. Як зазначено в дослідженні [10], гейміфіковані формати навчання достовірно підвищують рівень засвоєння знань та результативність виконання робочих завдань. Водночас, у корпоративному контексті важлива не лише ефективність, а й відповідальність за результати: система має зберігати дані про проходження тестів та надавати адміністратору можливість їх перегляду та експортування.

Вимоги до системи: адміністративна панель управління обліковими записами та результатами; можливість призначення тестів конкретним групам користувачів; постійне зберігання результатів із відміткою часу проходження; обмеження кількості спроб. Базове обмеження – необхідність обов’язкової автентифікації учасників для забезпечення достовірності даних.

Сценарій 5 – інтерактивна лекція або воркшоп. Актори: доповідач або тренер та аудиторія слухачів. Цей сценарій близький до Сценарію 1, однак відрізняється контекстом: вікторина використовується не для оцінювання, а як

інструмент утримання уваги та живого зворотного зв’язку під час виступу. Доповідач вбудовує запитання в структуру презентації для перевірки попереднього розуміння теми, збору думок аудиторії або стимулювання дискусії. Метою є підвищення інтерактивності заходу та залученості учасників. Кількість учасників може варіюватися від десятків до сотень осіб (на великих конференціях). Критичним є можливість швидкого запуску запитань без виходу з режиму презентації та миттєва агрегація відповідей у вигляді діаграм.

Вимоги до системи: мінімальний інтерфейс для ведучого (запуск одним кліком); швидке приєднання учасників без реєстрації; відображення агрегованих результатів у реальному часі. Критичне обмеження – масштабованість при великій кількості одночасних підключень.

Сценарій 6 – підготовка до іспиту та самооцінювання. Актори: студент, що самостійно практикується. Цей сценарій не передбачає ведучого та конкурентного елементу. Студент обирає набір запитань за певною темою, проходить його у власному темпі та отримує детальний зворотний зв’язок: які відповіді були правильними, де допущено помилки та яким є рекомендований матеріал для повторення. Метою є самодіагностика прогалин у знаннях та цілеспрямована підготовка до контрольного заходу. Цей режим безпосередньо пов’язаний із концепцією формативного оцінювання, що набула широкого поширення в сучасній педагогіці [11].

Вимоги до системи: відображення правильних відповідей із поясненнями після завершення; збереження історії спроб для відстеження динаміки; відсутність таймера або його необов’язковість. Базове обмеження – достатня наповненість бази запитань для забезпечення варіативності проходжень.

Для систематизації описаних сценаріїв та виявлення узагальнених вимог до платформи складено таблицю 1.1. Аналіз сукупності описаних сценаріїв дає змогу виявити кілька узагальнених вимог, яким має відповідати повноцінна вікторинна платформа: система повинна підтримувати обидва режими проведення: синхронний (сценарії 1, 2, 5) та асинхронний (сценарії 3, 4, 6), оскільки жоден із них не є надмірним; платформа має забезпечувати

масштабованість для підтримки від одного до декількох сотень одночасних учасників залежно від контексту; 3. необхідне постійне зберігання результатів та надання аналітики, щонайменше для сценаріїв 3, 4 та 6; система має підтримувати як гостьовий вхід (сценарії 1, 2, 5), так і автентифікацію зареєстрованих користувачів (сценарії 3, 4, 6); інтерфейс повинен бути адаптованим для різних пристроїв, оскільки учасники можуть підключатися зі смартфонів, планшетів або комп'ютерів у будь-якому з описаних контекстів.

За результатами попереднього аналізу сценаріїв використання було визначено, що різні контексти застосування платформи висувають принципово різні вимоги до формату оцінювання. Вибір типу запитання визначає когнітивну складність завдання, можливість автоматичної перевірки відповіді та технічну складність реалізації.

За формою відповіді у сучасних вікторинних платформах виокремлюють сім основних типів [12]: запитання з однією правильною відповіддю (single-choice); запитання з кількома правильними відповідями (multiple select); запитання типу «правильно-направильно» (true/false); запитання на встановлення відповідності (matching); запитання на впорядкування (ordering); запитання з відкритою відповіддю (open-ended); запитання на заповнення пропуску (fill-in-the-blank). Найпоширенішими є single-choice запитання: вони забезпечують широке тематичне охоплення та повністю автоматичну перевірку, проте при двох-чотирьох варіантах ймовірність вгадування є статистично значущою. На протилежному кінці спектру знаходиться питання з відкритими відповідями. Цей тип забезпечує найглибшу діагностику розуміння, однак автоматична перевірка потребує методів обробки природної мови, що суттєво ускладнює реалізацію. Типи «правльно-неправильно» та зіставлення відповідей займають проміжне положення: прості в реалізації, але обмежені за когнітивною глибиною.

Таблиця 1.1 – Характеристика типових сценаріїв використання онлайн-вікторин

Сценарій	Актори	Синхронність	Орієнтовна кількість учасників	Ключові вимоги до системи	Базове обмеження
1. Синхронна вікторина у класі	Викладач, учні	Синхронний	20–100	Реальний час, рейтинг, гостьовий вхід	Якість Wi-Fi у навчальному приміщенні
2. Дистанційна синхронна вікторина	Викладач, студенти	Синхронний	20–100	Мінімальний трафік, адаптивний інтерфейс	Паралельне навантаження від відеоконференції
Асинхронне завдання з дедлайном	Викладач, студенти	Асинхронний	20–200	Дедлайн, збереження результатів, аналітика	Ідентифікація учасників
Корпоративне compliance-тестування	HR-адміністратор, співробітники	Асинхронний	10–500	Адмін-панель, постійне зберігання, обмеження спроб	Обов’язкова автентифікація
Інтерактивна лекція або воркшоп	Доповідач, аудиторія	Синхронний	50–500	Масштабованість, швидкий запуск, агрегація результатів	Велика кількість одночасних підключень
Підготовка до іспиту та самооцінювання	Студент (самостійно)	Асинхронний	1	Зворотний зв’язок, історія спроб	Достатня наповненість бази запитань

Розгляд типів запитань крізь призму таксономії Блума дозволяє встановити відповідність між форматом запитання та рівнем когнітивної складності [13]. Нижні рівні таксономії (запам’ятовування, розуміння) ефективно перевіряються автоматизованими типами. Рівні аналізу та вищі потребують open-ended або есе-форматів із ручним оцінюванням, що жодна з розглянутих безкоштовних платформ повноцінно не підтримує. Зведену характеристику типів запитань подано у таблиці 1.2.

Таблиця 1.2 — Порівняння типів запитань у вікторинах

Тип запитання	Когнітивний рівень (Блум)	Автоматична перевірка	Найкраще для	Технічна складність
Single-choice	Запам’ятовування, розуміння	Так	Широке охоплення, факти	Низька
Multiple select	Розуміння, застосування	Так (з частковим балом)	Множинні зв’язки	Середня
True/False	Запам’ятовування	Так	Швидка діагностика	Низька
Matching	Запам’ятовування, розуміння	Так	Класифікації, означення	Середня
Ordering	Застосування, аналіз	Так	Процедурне знання, хронологія	Середня
Open-ended	Аналіз, оцінювання, творчість	Часткова або ні	Аргументація, глибина розуміння	Висока
Fill-in-the-blank	Запам’ятовування, застосування	Так (точне збігання)	Терміни, числові значення	Середня

Вибір типів запитань суттєво залежить від предметної галузі. У точних науках переважають задачі з однозначною відповіддю, проте виникає специфічна технічна вимога – підтримка рендерингу математичних формул (LaTeX або MathML), без якої запитання доводиться формулювати словесно [14]. У медичній та природничій освіті домінують запитання на основі сценаріїв (клінічні кейси), для яких характерний великий обсяг тексту умови та потреба у відображенні зображень. У гуманітарних науках вища педагогічна цінність відкритих відповідей стикається з неможливістю їх автоматичної перевірки, а підтримка кирилиці стає критичною вимогою до платформи. У корпоративному навчанні акцент зміщується на scenario-based multiple choice та matching-запитання, а обов’язковими стають збереження результатів та обмеження кількості спроб [10].

За результатами аналізу для майбутньої платформи Real-Time Quiz Hub обрано три типи запитань: single-choice, multiple select та open-ended. Single-choice охоплює найширший спектр тематик і є основним типом для сценаріїв 1-5. Multiple select підвищує дискримінаційну здатність тестів і відповідає рівням застосування та аналізу таксономії Блума, що є критичним для сценаріїв 3 та 6.

Включення open-ended усуває ключову прогалину існуючих безкоштовних рішень, жодне з яких не підтримує цей тип без платної підписки [15–17].

1.1.2 Механіки гейміфікації та їхня ефективність

Освітня ефективність платформи визначається не лише якістю запитань, а й тим, у який спосіб система мотивує учасників до залучення, повторення матеріалу та досягнення результату. Нарахування балів є базовою механікою більшості вікторинних платформ. Учасник отримує певну кількість балів за кожну правильну відповідь, причому в синхронному режимі бали, як правило, нараховуються з урахуванням швидкості відповіді: чим швидше надано правильну відповідь, тим вищий бал. Ця механіка формує безпосередній зворотний зв'язок та почуття прогресу після кожного запитання [6]. Водночас у літературі зафіксовано, що надмірне акцентування на швидкості відповіді може дезорієнтувати учасників із нижчим темпом обробки інформації, зокрема в академічному контексті [18].

Рейтингові таблиці (leaderboards) відображають поточний рейтинг учасників за кількістю набраних балів у реальному часі або після завершення сесії. Ця механіка активує соціальне порівняння та змагальну мотивацію. Систематичний огляд [18] засвідчив, що використання рейтингових таблиць у складі гейміфікованих освітніх середовищ має виражений позитивний ефект. Разом із тим, автори зазначають, що для учасників із хронічно низькими позиціями в рейтингу ця механіка може діяти демотивуючим чином, тому її ізольоване застосування без компенсаторних механік є педагогічно ризикованим.

Відзнаки та досягнення (badges та achievements) є позачасовими нагородами, що присвоюються за виконання певних умов: перше місце у вікторині, серія правильних відповідей підряд, участь у заданій кількості сесій тощо. На відміну від рейтингових таблиць, відзнаки мають накопичувальний характер і не вступають у пряму конкуренцію з іншими учасниками, що робить їх більш інклюзивним мотиваційним інструментом [6]. Систематичний

огляд [19] охопив 90 освітніх інтервенцій із застосуванням гейміфікації та підтвердив, що відзнаки та системи досягнень належать до найбільш стабільно ефективних механік підвищення залученості у широкому спектрі освітніх контекстів.

Смуги прогресу та рівні (progress bars та levels) забезпечують безперервну візуалізацію просування учасника до цілі. Ці механіки особливо ефективні в асинхронних сценаріях, де відсутній соціальний тиск від одночасного перебування поряд з іншими учасниками. Візуальний прогрес знижує відчуття когнітивного навантаження та формує так зване «відчуття майже досягнутого» (near-completion effect), що стимулює завершення розпочатого завдання [20].

Окремо слід розрізняти гейміфікацію рівня сесії та постійну (persistent) гейміфікацію. Механіки, прив’язані до окремої сесії, зокрема бали та рейтинг у реальному часі, забезпечують миттєву залученість, проте після завершення вікторини їхній стимулюючий ефект зникає. Натомість постійна гейміфікація передбачає накопичення прогресу учасника між сесіями у вигляді персонального профілю гравця: сукупного досвіду, динамічно обчислюваного рівня та каталогу здобутих відзнак. Такий профіль формує довгострокову мотиваційну траєкторію, оскільки кожна нова сесія сприймається як внесок у кумулятивний результат, а не як ізольована подія. У дослідженнях персистентні системи прогресу та досягнень пов’язують із підвищенням утримання (retention) учасників і стабільнішою повторною участю, що особливо цінно у сценаріях тривалого самонавчання та періодичного корпоративного перенавчання (сценарії 4 та 6) [10]. Для систематизації розглянутих механік та порівняння їх реалізації у провідних платформах складено таблицю 1.3.

Узагальнення результатів наукових досліджень щодо ефективності гейміфікації дозволяє сформулювати важливий висновок. Застосування так званого PBL-підходу (Points–Badges–Leaderboards) без урахування педагогічного контексту та індивідуальних характеристик учасників забезпечує переважно короткостроковий ефект підвищення залученості, який не обов’язково трансформується в стійке поліпшення результатів навчання [18].

Таблиця 1.3 — Механіки гейміфікації та їх вплив

Механіка	Опис	Вплив на мотивацію	Реалізація в Kahoot!	Реалізація в Quizizz
Нарахування балів	Бали за правильну та швидку відповідь	Безпосередній зворотний зв’язок, відчуття прогресу	Так, з урахуванням швидкості	Так, з урахуванням швидкості
Рейтингові таблиці	Поточний рейтинг учасників у реальному часі	Змагальна мотивація	Так, після кожного запитання	Так, після кожного запитання
Відзнаки та досягнення	Нагороди за виконання умов	Довгострокова залученість, інклюзивна мотивація	Обмежено (у платних версіях)	Так
Смуга прогресу та рівні	Візуалізація просування до цілі	Підтримка зусиль, ефект майже досягнутого	Ні	Так
Миттєвий зворотний зв’язок	Відображення правильної відповіді після вибору	Корекція хибних уявлень, дидактичний ефект	Так	Так
Стрік (серія відповідей)	Бонусні бали або відзнака за N правильних підряд	Стимулює уважність та концентрацію	Ні	Так

Найвищі та найстабільніші освітні результати досягаються при комбінованому застосуванні механік, де змагальні елементи (рейтинг, бали) доповнюються індивідуальними (відзнаки, прогрес, миттєвий зворотний зв’язок), а їх конкретний набір адаптується до цілей, аудиторії та предметної області й доповнюється персистентними механіками (накопичувальний профіль, рівні, відзнаки) для підтримання довгострокового утримання учасників [18; 19].

1.2 Технологічний контекст розробки платформи

Механіки гейміфікації реалізуються лише за умови своєчасної та надійної доставки даних між сервером і клієнтами. У синхронних сценаріях використання вікторинної платформи (сценарії 1, 2 та 5) критичним є те, щоб усі учасники отримали нове запитання, оновлення таймера та зміни рейтингової таблиці практично одночасно. Затримка, що перевищує 300-500 мс, стає помітною для учасника та підриває справедливість нарахування балів залежно від швидкості відповіді. Вибір транспортного протоколу для реалізації такої комунікації є

одним із ключових архітектурних рішень при проєктуванні вікторинної платформи. Для обґрунтованого вибору технології передавання даних наведено порівняльну таблицю 1.4, що систематизує розглянуті підходи за ключовими характеристиками.

Таблиця 1.4 – Порівняння технологій забезпечення комунікації в реальному часі

Технологія	Напрям передачі	Затримка	Накладні витрати протоколу	Постійне з’єднання	Підтримка у браузерах
Short Polling	Клієнт → Сервер (ініціатива клієнта)	Висока (інтервал опитування)	Дуже високі (повний HTTP)	Ні	Усі
Long Polling	Клієнт → Сервер (з очікуванням)	Середня	Середні	Часткова	Усі
Server-Sent Events	Сервер → Клієнт (однобічний)	Низька	Низькі	Так	Більшість
WebSocket (RFC 6455)	Двобічний	Мінімальна	Мінімальні	Так	Усі сучасні
ASP.NET Core SignalR	Двобічний (з автодеграда-цією)	Мінімальна (WebSocket)	Мінімальні	Так	Усі сучасні

Протокол WebSocket, стандартизований у документі RFC 6455 «The WebSocket Protocol» [18], вирішує зазначені обмеження шляхом встановлення повнодуплексного постійного з’єднання між клієнтом і сервером. Процедура починається зі стандартного HTTP-запиту з заголовком Upgrade: websocket; після підтвердження сервером з’єднання переходить у режим WebSocket, в якому обидві сторони можуть надсилати повідомлення в довільний момент без додаткового узгодження. Заголовки HTTP при цьому не надсилаються повторно, що різко знижує обсяг службового трафіку. Саме ця характеристика робить WebSocket оптимальним вибором для синхронних вікторинних сценаріїв.

За результатами порівняння встановлено, що для синхронних сценаріїв 1 та 2, де критичним є мінімальна затримка при одночасному підключенні десятків учасників, єдиними технологічно адекватними варіантами є WebSocket або

ASP.NET Core SignalR. Використання SignalR надає додаткову перевагу завдяки автоматичній деградації до SSE або Long Polling у середовищах із обмеженою підтримкою WebSocket, що підвищує сумісність платформи з корпоративними мережами з обмежувальними проксі-серверами (актуально для сценарію 4).

Вибір технології зберігання даних у системах із комунікацією в реальному часі визначається двома принципово різними класами вимог. З одного боку, система має забезпечувати швидкий доступ до даних поточної сесії: стану вікторини, поточних балів учасників, відповідей, що надходять одночасно від десятків клієнтів. З іншого боку, після завершення сесії ці дані мають бути надійно збережені для подальшого перегляду аналітики викладачем або адміністратором (сценарії 4 та 6). Ці два класи вимог зумовлюють застосування різних технологій зберігання в рамках одного застосунку.

Реляційні бази даних є традиційним вибором для забезпечення довгострокового персистентного зберігання структурованих даних. СКБД PostgreSQL – об’єктно-реляційна система управління базами даних із відкритим кодом, що підтримує повний набір транзакцій відповідно до стандарту ACID (Atomicity, Consistency, Isolation, Durability), складні запити з об’єднанням таблиць, зберігання JSON-документів, повнотекстовий пошук та розширюваність через користувацькі типи даних [20]. У контексті вікторинної платформи PostgreSQL є природним вибором для зберігання облікових записів користувачів, наборів запитань, налаштувань вікторин та архіву результатів сесій.

Проте реляційні бази даних, навіть оптимізовані, не призначені для обслуговування запитів із затримкою менше одної мілісекунди, що є вимогою під час активної ігрової сесії. Запис відповіді кожного з 100 учасників до PostgreSQL у режимі реального часу, із одночасним читанням для оновлення рейтингової таблиці, створював би значне навантаження на дисковий ввід-вивід та загрожував би деградацією часу відповіді під навантаженням [21; 22].

Для вирішення цієї задачі в високонавантажених системах традиційно застосовується Redis – сховище даних типу «ключ-значення», що функціонує в

оперативній пам’яті. Redis зберігає дані безпосередньо в RAM, забезпечуючи час доступу менше одної мілісекунди для операцій читання та запису, і підтримує такі структури даних, як рядки, списки, множини, відсортовані множини та хеш-таблиці [22]. Відсортовані множини Redis є особливо зручними для реалізації рейтингових таблиць: кожен учасник зберігається як елемент множини із числовим значенням (балом), а операція отримання топ-N учасників виконується за логарифмічний час [22]. Крім того, Redis підтримує механізм підпски для широкомовного розсилання повідомлень, що є цінним у розподілених системах із кількома інстансами застосунку.

У межах кваліфікаційної роботи, однак, замість Redis було обрано механізм кешування в оперативній пам’яті засобами платформи .NET – IMemoryCache, задокументований у офіційній документації Microsoft [23]. Це рішення обґрунтовується такими аргументами:

– відповідно до попередніх нефункціональних вимог MVP (до 100 учасників в одній сесії), навантаження на сховище поточного стану сесії є достатньо малим, щоб оперативна пам’ять серверного процесу слугувала ефективним буфером без потреби у відокремленому Redis-сервері;

– використання IMemoryCache усуває зовнішню залежність у вигляді окремого Redis-інстансу, що спрощує розгортання та знижує інфраструктурні вимоги до середовища виконання;

– архітектура застосунку спроєктована таким чином, що заміна IMemoryCache на Redis у майбутньому (при масштабуванні понад 100 учасників або при переході до мульти-інстансного розгортання) потребує мінімальних змін завдяки абстракції IDistributedCache у ASP.NET Core [24].

Отже, обрана архітектура зберігання даних є гібридною: PostgreSQL забезпечує персистентне зберігання всіх структурованих даних між сесіями, тоді як IMemoryCache обслуговує «гарячі» дані активної сесії – поточний стан, бали та відповіді учасників – зі швидкістю, що є достатньою для реалізації комунікації в реальному часі в межах визначеного масштабу MVP. Обрана стратегія зберігання є доцільною для поточного масштабу платформи та забезпечує

можливість еволюційного переходу до Redis при збільшенні навантаження без переписування бізнес-логіки.

При безпосередній реалізації WebSocket у ASP.NET Core розробник вимушений вручну управляти підключеннями, групами клієнтів, серіалізацією повідомлень та обробкою розривів з'єднання. Для спрощення цієї задачі в екосистемі .NET розроблено бібліотеку ASP.NET Core SignalR, що є абстракційним шаром над транспортним рівнем і надає API вищого рівня для реалізації комунікації в реальному часі [19]. SignalR автоматично обирає найкращий із доступних транспортних протоколів у такому пріоритеті: WebSocket → Server-Sent Events → Long Polling. Якщо клієнт або проміжний проксі-сервер не підтримують WebSocket, SignalR прозоро деградує до SSE або Long Polling без жодних змін у коді застосунку. Крім транспортної абстракції, SignalR надає механізм хабів (Hubs) – серверних класів, методи яких можна викликати безпосередньо з клієнтського JavaScript-коду, а також вбудовані засоби управління групами підключень, що є критично важливим для ізоляції сесій різних вікторин [19].

1.3 Аналіз наявних рішень

З метою обґрунтування доцільності розробки нової платформи здійснено аналіз трьох найбільш поширених програмних рішень у галузі онлайн-вікторин із елементами гейміфікації: Kahoot!, Quizizz та AhaSlides. Платформи відібрано за критеріями популярності в освітньому середовищі, наявності синхронного режиму проведення та задокументованої аудиторії серед викладачів і тренерів. Аналіз кожного рішення проводиться у розрізі механіки гейміфікації, підходу до забезпечення комунікації в реальному часі, підтримки типів запитань та обмеження безкоштовних версій.

Платформа Kahoot! (рис. 1.1) є однією з перших і найпопулярніших платформ для проведення синхронних вікторин, що позиціонується як «ігрова навчальна платформа» і широко застосовується у закладах загальної середньої та вищої освіти, а також у корпоративних тренінгах [25].

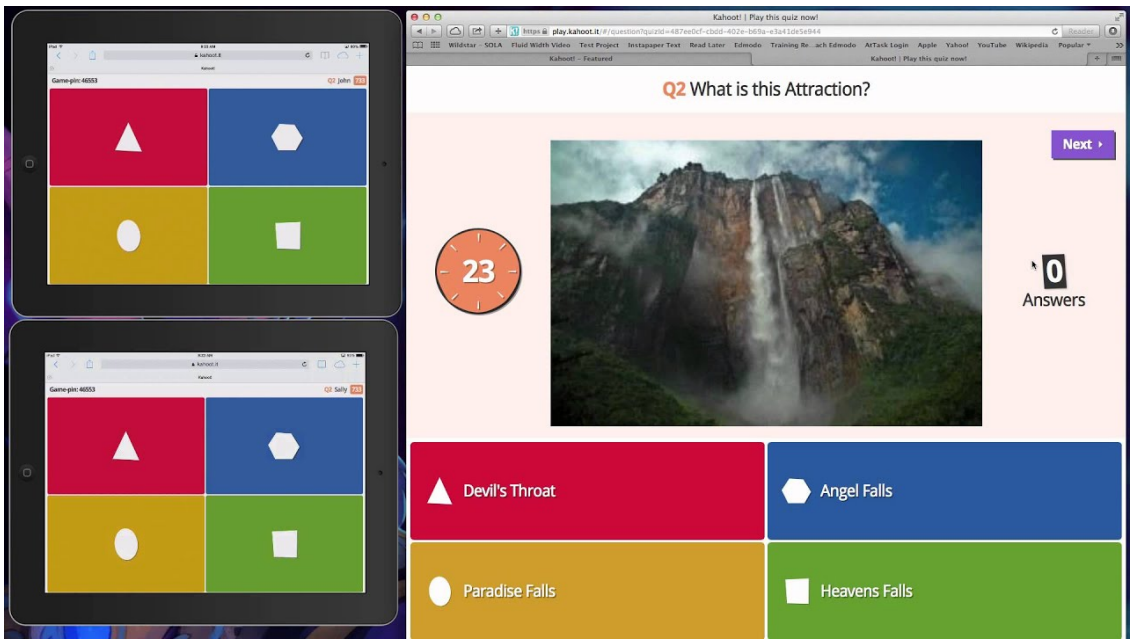

Рисунок 1.1 – Вигляд платформи Kahoot!

З точки зору механік гейміфікації, Kahoot! реалізує типовий набір PBL з акцентом на змагальність та швидкість. Нарахування балів відбувається за комбінованою формулою, що враховує як правильність відповіді, так і час, витрачений на її вибір: учасник, який відповів правильно і першим, отримує максимальну кількість балів. Після кожного запитання відображається проміжна таблиця лідерів, що актуалізує соціальне порівняння та підтримує змагальну атмосферу. Після завершення вікторини платформа відображає «п’єдестал» – підсумкове нагородження трійки лідерів. Бейджі та довгострокові досягнення у безкоштовній версії не передбачені; командний режим доступний лише на платних тарифах.

З технічної точки зору, Kahoot! використовує синхронну модель проведення, де сервер централізовано керує станом сесії: розсилає запитання, запускає таймер та збирає відповіді від усіх учасників одночасно. Поведінкові характеристики платформи (практично миттєве відображення результатів, підтримка одночасних відповідей сотень гравців) свідчать про застосування персистентних з’єднань типу WebSocket або аналогічного механізму. Додатково платформа підтримує асинхронний режим «Challenge», за якого ведучий

публікує вікторину у відкладеному форматі, а учасники проходять її самостійно у зручний час.

Щодо підтримуваних типів запитань, у безкоштовній версії Kahoot! доступні лише multiple choice та true/false [25]. Типи запитань open-ended, puzzle та slider доступні виключно на платних тарифах. Це суттєво обмежує педагогічний потенціал платформи в контексті сценаріїв самооцінювання та виконання домашніх завдань (сценарії 3 та 6).

Основні обмеження безкоштовної версії: кількість учасників у живій сесії обмежена до 50 осіб для освітнього базового акаунту [25]; розширені типи запитань, брендоване оформлення та детальний експорт результатів доступні лише на платних тарифах. Інтерфейс платформи не локалізовано українською мовою. Вагомим структурним обмеженням є залежність класичного режиму від спільного екрана: запитання у повному обсязі відображаються лише на пристрої ведучого, тоді як на смартфонах гравців – лише кольорові кнопки варіантів відповідей, що ускладнює дистанційне використання (сценарій 2).

Платформа Quizizz (рис. 1.2) виникла як альтернатива Kahoot! з принципово іншою концепцією: акцент зміщено з колективного змагання у класі на індивідуалізоване навчання [21]. Ключова відмінність полягає в тому, що запитання та варіанти відповідей відображаються безпосередньо на пристрої кожного гравця, а не на спільному екрані, що усуває залежність від проєктора та уможливлює комфортне дистанційне використання.

З точки зору гейміфікації, Quizizz застосовує більш різноманітний набір механік порівняно з Kahoot!. Нарахування балів орієнтоване переважно на правильність, а не швидкість відповіді, що знижує стресове навантаження на учасників. Платформа реалізує механіку «Streak» – серію правильних відповідей поспіль, що нараховує додатковий бонус, стимулюючи стабільність, а не лише імпульсивну швидкість. Передбачені «Power-ups» – ігрові підказки (наприклад, вилучення двох неправильних варіантів), що учасники можуть накопичувати та використовувати стратегічно. Після кожного запитання гравцю відображається кумедна реакція у вигляді мему залежно від правильності вибору. Підсумкова

таблиця лідерів формується наприкінці вікторини або може переглядатися учасниками в будь-який момент через застосунок.

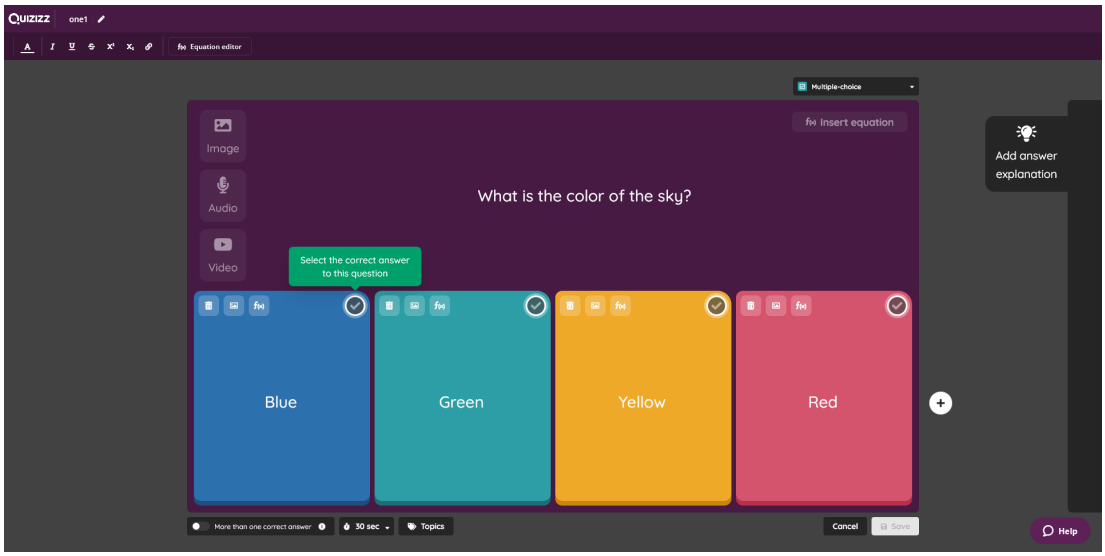

Рисунок 1.2 – Вигляд платформи Quizizz []

З технічного боку, Quizizz підтримує два режими: «Live Game» (синхронний, де ведучий керує темпом) та «Homework» (асинхронний, де учасники проходять вікторину самостійно до встановленого дедлайну). У синхронному режимі учасники можуть рухатись у власному темпі, що відрізняє платформу від Kahoot!. Це дещо знижує навантаження на канал зв'язку, але водночас зменшує ефект колективного змагання в реальному часі.

У безкоштовній версії Quizizz підтримує ширший набір типів запитань порівняно з Kahoot!: multiple choice, true/false, fill-in-the-blank та matching. Однак тип open-ended із автоматичною або напівавтоматичною перевіркою у безкоштовному тарифі не підтримується, що підтверджує раніше виявлену системну прогалину.

Обмеження безкоштовної версії: до 100 гравців у сесії; до 20 активних вікторин у базовому акаунті [21]. У безкоштовному режимі на сторінках гри відображається реклама, що є незручністю для навчального середовища. Кастомізація оформлення та завантаження власних мемів доступні лише у

платних тарифах. Інтерфейс підтримує близько семи мов, але українська локалізація відсутня.

Платформа AhaSlides (рис. 1.3) позиціонується як платформа для інтерактивних презентацій, а не як спеціалізований інструмент для вікторин [22]. Вікторинні блоки є лише одним із типів слайдів поряд із живими опитуваннями, Q&A-сесіями та хмарами слів. Цільова аудиторія – ведучі, доповідачі та викладачі, яким необхідно урізноманітнити презентацію інтерактивними активностями.

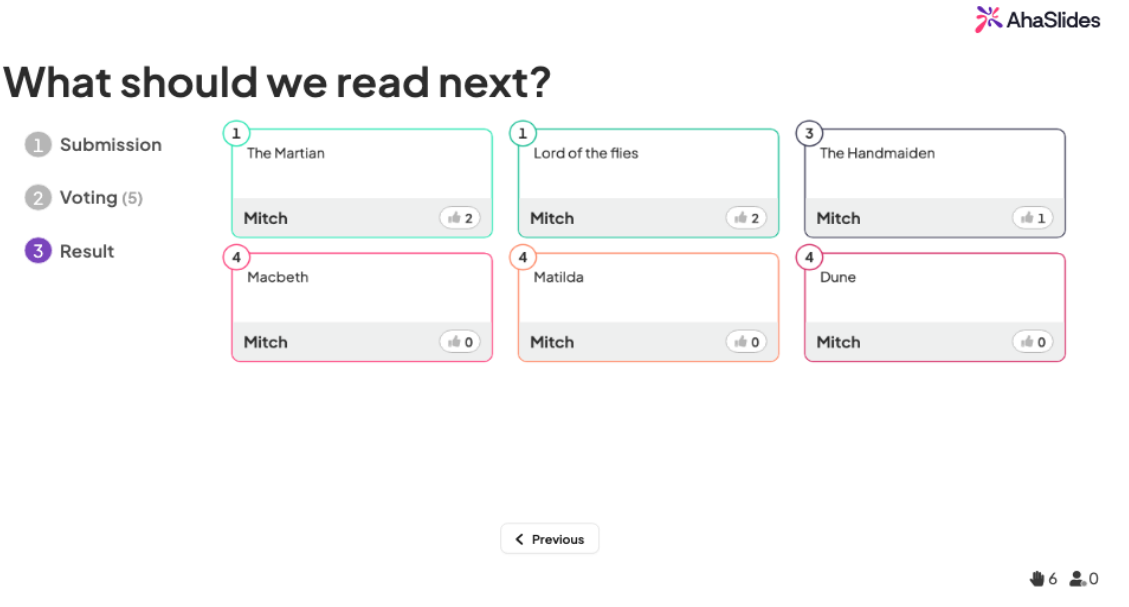

Рисунок 1.3 – Вигляд платформи AhaSlides

З точки зору гейміфікації, AhaSlides реалізує базовий набір: нарахування балів за правильність відповіді та таблиця лідерів із можливістю налаштування частоти її відображення. Яскравих ігрових ефектів (музика, меми, анімації) платформа не надає за замовчуванням, стиль оформлення нейтральний та ближчий до ділового. Специфічною механікою залученості є кнопка «аплодисментів», що дозволяє аудиторії надсилати емодзі-реакції в реальному часі під час презентації. Командного режиму, смуг правильних відповідей та системи бейджів платформа не передбачає.

AhaSlides підтримує виключно синхронний (живий) режим проведення: учасники підключаються за посиланням або кодом, і ведучий сам регулює темп

показу слайдів [22]. Асинхронний режим проходження відсутній. З технічної точки зору, платформа є браузерним застосунком без вимоги встановлення додатків; наявна інтеграція з PowerPoint та Zoom.

У безкоштовній версії AhaSlides підтримуються лише multiple choice та відкриті словесні відповіді у форматі хмари слів, що не є повноцінним open-ended із перевіркою. Обмеження безкоштовної версії: до 50 учасників одночасно; не більше 5 запитань-вікторин в одній презентації; логотип AhaSlides відображається на всіх слайдах. Офіційна українська локалізація відсутня, хоча введення тексту кирилицею підтримується.

Проведений аналіз дозволяє встановити, що кожна з розглянутих платформ має виражену спеціалізацію і закриває лише частину з шести попередньо визначених типових сценаріїв використання. Жодна з розглянутих платформ не підтримує україномовний інтерфейс [15–17], що є системною прогалиною для україномовної освітньої та корпоративної аудиторії.

Отже, виявлено чотири ключові незакриті потреби, що формують простір для розроблення нової платформи: відсутність безкоштовного рішення з повноцінною адміністративною панеллю для корпоративного сценарію; відсутність підтримки open-ended запитань без платної підписки в будь-якій з платформ; відсутність україномовного інтерфейсу в жодному з розглянутих рішень; відсутність у безкоштовних версіях повноцінної постійної (persistent) гейміфікації – накопичувального профілю гравця з досвідом, рівнями та відзнаками, що зберігаються між сесіями. Kahoot! не надає довгострокових досягнень у безкоштовному тарифі, AhaSlides не передбачає системи відзнак узагалі, а Quizizz реалізує лише фрагментарні елементи без єдиного персистентного профілю, що обмежує підтримку довгострокової мотивації у сценаріях тривалого самонавчання та повторної участі (сценарії 4 та 6).

1.4 Постановка задачі на розроблення

На підставі аналізу предметної галузі, типових сценаріїв використання та існуючих програмних рішень сформульовано задачу на розробку вебплатформи

Real-Time Quiz Hub – відкритого україномовного рішення для проведення онлайн-вікторин у реальному часі з механіками гейміфікації, що охоплює підтримку синхронного та асинхронного режимів проведення, різних типів запитань та адміністративних інструментів управління. Систематизовані потреби наведено в таблиці 1.5.

Таблиця 1.5 – Потреби користувачів за результатами аналізу сценаріїв

№	Потреба	Сценарії	Зацікавлена сторона
П-01	Можливість проводити синхронну вікторину для 50 і більше учасників одночасно без деградації продуктивності	Сценарій 1, 2, 5	Викладач, тренер, доповідач
П-02	Підтримка асинхронного режиму проходження з визначенням дедлайну та автоматичним його відстеженням	Сценарій 3, 4	Викладач, HR-адміністратор
П-03	Збереження результатів усіх учасників та надання аналітики після завершення сесії	Сценарій 3, 4, 6	Викладач, HR-адміністратор, студент
П-04	Гостьовий вхід учасників без попередньої реєстрації для синхронних форматів	Сценарій 1, 2, 5	Учень, студент, слухач
П-05	Підтримка різних типів запитань: single-choice, multiple select та open-ended	Сценарій 3, 4, 5, 6	Викладач, HR-адміністратор
П-06	Адміністративна панель управління обліковими записами та призначення вікторин групам	Сценарій 4	HR-адміністратор
П-07	Повністю україномовний інтерфейс для ведучого та учасників	Усі сценарії	Усі зацікавлені сторони
П-08	Адаптивний інтерфейс для мобільних пристроїв із відображенням повного тексту запитання на пристрої учасника	Сценарій 1, 2, 3, 6	Учень, студент
П-09	Можливість самостійного вибору вікторини із загального каталогу доступних вікторин	Сценарій 3, 5, 6	Викладач, студент, доповідач
П-10	Збереження особистого ігрового прогресу (досвід, рівень, здобуті відзнаки) між сесіями для підтримання довгострокової мотивації	Сценарій 4, 6	Студент, HR-адміністратор

Для чіткого визначення меж системи встановлено перелік основних обмежень і припущень, в межах яких реалізується платформа. Щодо типу застосунку, платформа реалізується як вебзастосунок, доступний через браузер, без розробки нативних мобільних додатків для iOS та Android. Мобільна доступність забезпечується виключно через адаптивний вебінтерфейс.

Мовою інтерфейсу платформи визначено українську. Підтримка інших мов у межах цієї роботи не передбачена, однак архітектура застосунку допускає майбутню локалізацію.

Платформа розрахована на підтримку до 100 учасників в одній сесії одночасно. Цей параметр перевищує ліміти безкоштовних версій двох із трьох розглянутих конкурентних рішень і є достатнім для охоплення всіх визначених сценаріїв використання.

Платформа проєктується для розгортання як у локальному середовищі (on-premise), так і у хмарному середовищі загального призначення без прив’язки до конкретного хмарного провайдера; розгортання здійснюється засобами контейнеризації (Docker) для забезпечення відтворюваності середовища виконання. Передбачена підтримка гостьового входу учасників синхронних вікторин (без реєстрації, за кодом сесії) та автентифіковані облікові записи для ведучих і адміністраторів. Інтеграція зі сторонніми провайдерами автентифікації у межах MVP не реалізується.

Загалом, у першому розділі здійснено комплексний огляд предметної області онлайн-вікторин із механіками гейміфікації. Виділено шість типових сценаріїв використання, що формують узагальнені вимоги до підтримки обох режимів проведення, масштабованості та збереження результатів. Систематизовано типологію запитань у взаємозв’язку з таксономією Блума та обґрунтовано вибір трьох типів для реалізації. Проведений порівняльний аналіз платформ Kahoot!, Quizizz та AhaSlides виявив чотири ключові незакриті потреби: відсутність безкоштовного рішення з адміністративною панеллю, відсутність підтримки open-ended запитань без платної підписки, відсутність україномовного інтерфейсу та відсутність персистентної гейміфікації з накопичувальним профілем гравця у безкоштовних версіях розглянутих рішень. За межами поточного скоупу залишаються асинхронний режим проведення вікторини з дедлайном; повноцінна адміністративна панель із графічним інтерфейсом; розширена аналітика для ведучого (розподіл відповідей по варіантах, середній час відповіді, прогрес по запитаннях).

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ REAL-TIME QUIZ HUB

2.1 Специфікація вимог до програмного забезпечення

Вимоги до програмного забезпечення було поділено на три категорії: функціональні (FR), нефункціональні (NFR) та користувацькі сценарії (UC), що забезпечує трасованість між потребами користувачів, конкретною поведінкою системи та атрибутами якості.

Функціональні вимоги визначають спостережувану поведінку системи у відповідь на входи або події, тобто те, що система повинна робити, а не як вона це реалізує [26]. Для Real-Time Quiz Hub вимоги формалізовано з урахуванням трьох ролей: Гравця (Player), Ведучого (Host) та Адміністратора (Admin). Слід зазначити, що між акторами існує ієрархія повноважень. Ведучий може виконувати всі прецеденти Гравця і додатково контролює хід сесії. Адміністратор має найширші права доступу, може виконувати як адміністративні, так і функції ведучого, проте у поточній MVP-реалізації роль адміністратора обмежена API-інтерфейсом без окремого повноцінного графічного інтерфейсу управління. Межі системи та зв’язки між акторами і прецедентами відображено на UML-діаграмі прецедентів, наведеній на рис. 2.1.

Ключовим проєктним рішенням є гостьовий вхід Гравця за нікнеймом без реєстрації, що знижує поріг входу. Ролі Host та Admin вимагають реєстрації та JWT-токена [27]. Вимоги визначено методом аналізу варіантів використання та зіставлення з обмеженнями аналогів (Kahoot!, Quizizz, AhaSlides) з розділу 1. Повний перелік наведено в таблиці 2.1.

Функціональні вимоги FR-01-FR-12 охоплюють повний цикл проведення онлайн-вікторини, від автентифікації та наповнення банку запитань до завершення сесії, нарахування досвіду й присвоєння досягнень. Механіки постійної гейміфікації (XP, рівні, бейджі), раніше заплановані, реалізовано в поточній версії. Усі вимоги підлягають верифікації в розділі 3.

Таблиця 2.1 – Функціональні вимоги до Real-Time Quiz Hub

ID	Назва	Роль	Опис	Демо-сценарій
FR-01	Автентифікація та авторизація	Всі ролі	Host та Admin реєструються через POST /api/auth/register і входять через POST /api/auth/login, отримуючи JWT-токен із роллю у claims [27]; токен зберігається на клієнті в localStorage (ключі quizhub_token, quizhub_user). Гравець входить гостьово за нікнеймом. Інтерфейс автентифікації реалізовано на чистому HTML із власним CSS, взаємодія із сервером через fetch. Привілейовані ендпоінти захищено перевіркою ролі на рівні контролерів.	Вхід із валідними та невалідними даними; запит до захищеного ендпоінту без токена → очікується HTTP 401 Unauthorized.
FR-02	Управління варіантами відповідей	Admin	Адміністратор виконує повний CRUD варіантів відповідей через групу ендпоінтів /api/admin/answers (GET, POST, GET/{id}, PUT/{id}, DELETE/{id}); усі запити вимагають JWT-токена з роллю Admin.	Послідовне виконання операцій CRUD через Swagger UI.
FR-03	Управління запитаннями	Admin	Адміністратор виконує повний CRUD запитань через /api/admin/questions; підтримуються три типи: single-choice, multiple-select (автоматична перевірка) та open-ended (ручна перевірка ведучим).	Створення запитання кожного типу, оновлення та видалення через Swagger.
FR-04	Запуск ігрової сесії	Host, Admin	Ведучий або адміністратор надсилає POST /api/Quiz/start; система ініціалізує in-memory сесію (поточний індекс = 0, порожній список гравців) і повертає перше запитання.	HTTP 200 OK з першим запитанням; наявність ініціалізованого стану сесії.
FR-05	Отримання наступного запитання	Player, Host	Клієнт надсилає GET /api/Quiz/{quizId}/next; система повертає наступне запитання за OrderIndex і збільшує лічильник; за вичерпання запитань повертає статус для завершення.	Послідовні звернення: коректна зміна запитань за встановленим порядком.
FR-06	Надсилання відповіді	Player	Гравець надсилає POST /api/Quiz/{quizId}/submit з обраним варіантом; система перевіряє правильність і нараховує бали в in-memory структурі сесії, повертаючи результат (правильно / неправильно).	Правильна та неправильна відповідь: відповідне повідомлення й оновлення балів.
FR-07	Real-time оновлення лідерборду (SignalR)	Усі учасники сесії	Після обробки відповіді сервіс викликає метод хабу QuizHub, який оновлює рейтинг і надсилає подію BroadcastLeaderboard усім клієнтам групи сесії [25]; оновлення відбувається без перезавантаження сторінки.	Кілька клієнтів в одній сесії: лідерборд оновлюється в усіх без явного запиту.

Продовження таблиці 2.1

FR-08	Завершення сесії	Host	Ведучий надсилає POST /api/Quiz/{quizId}/end; система підраховує фінальний результат кожного гравця (X / TotalQuestions), повертає зведену таблицю й очищає in-memory стан сесії.	Виклик після всіх запитань: коректність підрахунку; повторний /next повертає помилку.
FR-09	Перегляд інформації про вікторину	Host, Admin	Ведучий або адміністратор надсилає GET /api/Quiz/{quizId}; система повертає метадані вікторини: назву, кількість запитань, поточний стан сесії.	Отримання об'єкта вікторини з перевіркою полів.
FR-10	Нарахування балів у сесії	Система (автома-тично)	Система автоматично нараховує бали за кожну правильну відповідь під час сесії; накопичений рахунок визначає позицію в лідерборді (FR-07) і зберігається в in-memory структурі до завершення сесії (FR-08).	Кілька гравців із різними результатами: коректне ранжування й динаміка позицій.
FR-11	Вибір вікторини	Player, Host	Перегляд каталогу вікторин через GET /api/quizzes та отримання конкретної вікторини через GET /api/quizzes/{id} для подальшого запуску. Дозволяє обрати одну з наявних вікторин замість єдиного банку запитань.	Отримання списку вікторин; вибір за ідентифікатором і запуск сесії на її основі.
FR-12	Механіки постійної гейміфікації	Система (автома-тично)	Після завершення сесії клієнт надсилає POST /api/scores/complete; сервіс обчислює підсумковий бал, нараховує XP у таблиці UserStats (поле TotalScore), динамічно визначає рівень гравця через LevelHelper (рівень у БД не зберігається) і присвоює відповідні бейджі (таблиці Badges, UserBadges). Глобальний рейтинг доступний через GET /api/leaderboard.	Завершення сесій із різними результатами: перевірка нарахованого XP, зміни рівня при перетині порогу та присвоєних бейджів.

Нефункціональні вимоги (NFR) визначають атрибути якості програмної системи – обмеження на те, як реалізується функціональність, а не на саму функціональність [28]. Невиконання NFR, на відміну від невиконання FR, часто не виявляється під час звичайного функціонального тестування. Для їх перевірки потрібні окремі процедури вимірювання.

Для Real-Time Quiz Hub нефункціональні вимоги сформульовано з урахуванням специфіки систем реального часу. До ключових характеристик таких систем належать: висока частота подій (кожна відповідь гравця генерує

подію оновлення для всіх учасників), необхідність синхронного оновлення стану розподіленого набору клієнтів, потреба у захисті облікових даних та забезпечення доступності для широкої аудиторії, яка може використовувати пристрої різного класу. Кожна NFR супроводжується вимірюваною метрикою та чітко визначеною процедурою підтвердження, що виконуватиметься у розділі 3, що забезпечує можливість об’єктивної верифікації атрибутів якості. Повний перелік нефункціональних вимог наведено у таблиці 2.2.

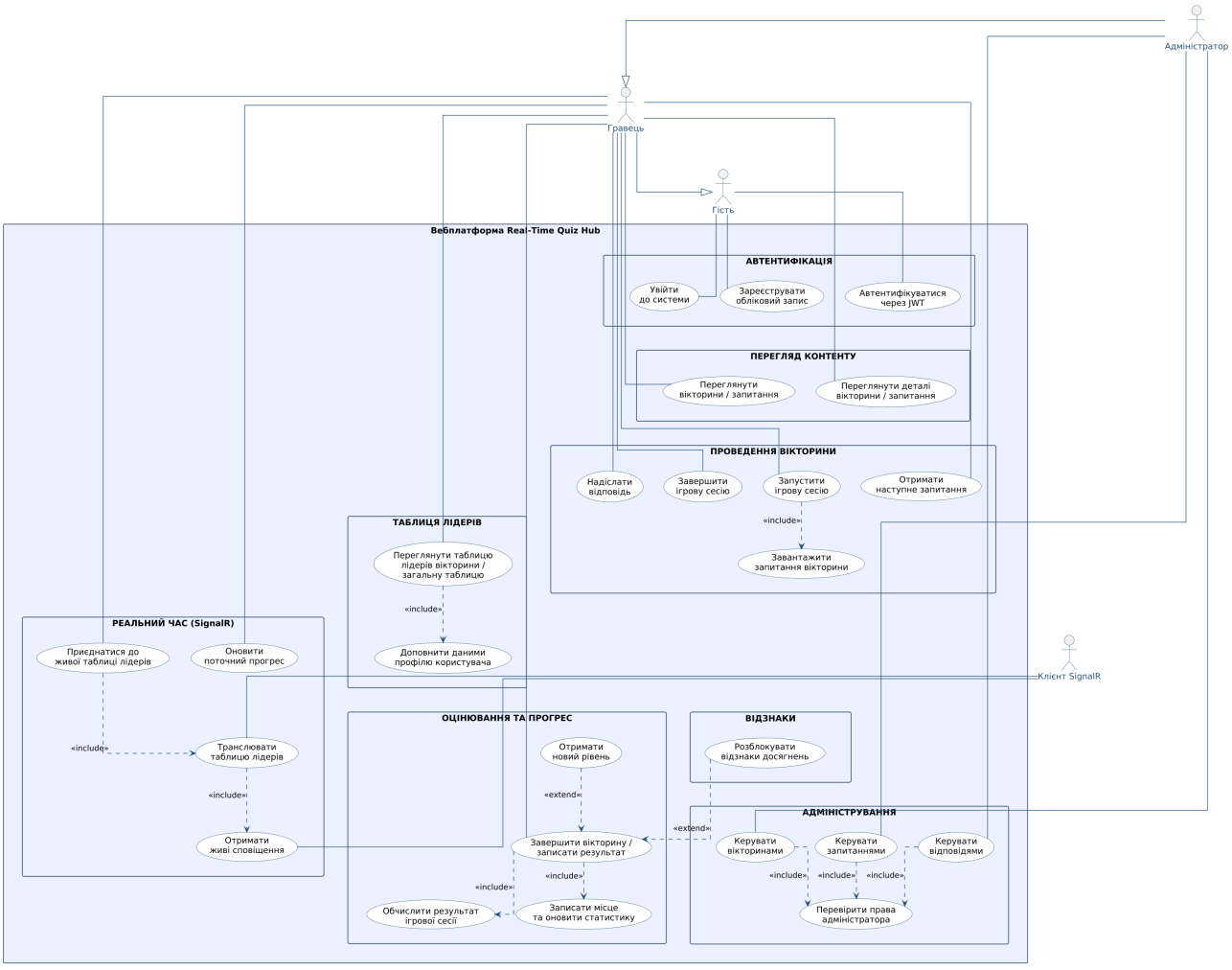

Рисунок 2.1 – UML-діаграма прецедентів системи Real-Time Quiz Hub

Визначені атрибути якості є критичними для real-time платформи з кількох взаємопов’язаних причин. Продуктивність (NFR-01) безпосередньо впливає на ігровий досвід: затримка понад 2 секунди між відповіддю гравця та оновленням лідерборду руйнує відчуття синхронності, що є ключовою цінністю live-режиму

на відміну від асинхронних аналогів. Надійність (NFR-02) є особливо важливою з огляду на цільову аудиторію: освітні заклади та корпоративні тренінги часто проводяться у середовищах з нестабільним мобільним з’єднанням.

Таблиця 2.2 – Нефункціональні вимоги до Real-Time Quiz Hub

ID	Атрибут якості	Формулювання	Метрика вимірювання	Процедура підтвердження
NFR-01	Продуктив-ність	Затримка від надсилання відповіді гравцем до оновлення лідерборду в усіх клієнтів сесії ≤ 2 с за 100 одночасних гравців.	Час (мс) від POST /api/quiz/{quizId}/submit до отримання події BroadcastLeaderboard кожним клієнтом; P50/P95/P99.	Навантажувальний тест: 100 SignalR-клієнтів одночасно відповідають і викликають UpdateProgress; замір часу до BroadcastLeaderboard; поріг P95 ≤ 2000 мс.
NFR-02	Надійність	Після розриву WebSocket гравець повертається до активної сесії без втрати нікнейму та балів.	Бінарна: 1 – нікнейм і CorrectAnswers збережено після реконекту, 0 – втрачено.	Примусовий розрив з’єднання → автоматичний reconnect → повторний RegisterUser → перевірка, що in-memory сесія (ключ quizId:nickname) повертає той самий нік і ба
NFR-03	Безпека	Паролі зберігаються лише як PBKDF2-хеш (ASP.NET Core Identity PasswordHasher); ендпоінти з [Authorize] доступні тільки за валідним JWT; доступ до БД – винятково через параметризований EF Core LINQ.	К-сть [Authorize]-ендпоінтів, дь викликів FromSqlRaw/ExecuteSqlRaw (0); к-сть паролів у відкритому вигляді в БД (0).	Запит до [Authorize]-ендпоінта без JWT → очікувано HTTP 401; grep по кодовій базі на raw-SQL = 0; перевірка таблиці users – поле PasswordHash містить лише PBKDF2-хеші.
NFR-04	Зручність використання	Приєднання до сесії за нікнеймом ≤ 3 кроки; коректне відображення на екранах від 320px.	Кількість кроків приєднання (≤3); Task Success Rate (%); адаптивність без горизонтального скролу на ширині 320-1080px.	Інспекція потоку (ввід ніку → «Старт» = 2 кроки); usability-тест ≥5 респондентів; перевірка верстки в DevTools на 320 / 520 / 880px.
NFR-05	Локалізація	Весь видимий інтерфейс лише українською: форми, кнопки, повідомлення про помилки та підказки.	% локалізованих рядків видимого UI (100%).	Ручний огляд усіх екранів на наявність неукраїнських рядків UI.

Продовження таблиці 2.2

NFR-06	Підтримуваність	Шарова архітектура: бізнес-логіка Services/ не залежить від Controllers/ чи Hubs/ і покрита unit-тестами з підставними залежностями (Moq/фейки).	Бінарна: 0 using-директив RealTmeQuizHub.Hubs у Services/ та Repository/; наявність прохідних unit-тестів Services/.	Статичний аналіз using-директив (grep) = 0 заборонених залежностей; запуск dotnet test – тести Services/ зелені з мок-залежностями (без реальної БД/хабу).
NFR-07	Масштабованість	Персистентний рейтинг інкапсульовано за інтерфейсом IQuizLeaderboardStore, тож його реалізацію (in-memory) можна замінити (наприклад, на Redis) без змін у споживачах (Services/,	Кількість споживачів, що звертаються до конкретної реалізації QuizLeaderboardStore замість інтерфейсу IQuizLeaderboardStore (0).	Архітектурний огляд: усі cnrboardStore; підміна реєстрації в DI (Program.cs) на іншу реалізацію не потребує правок у консюмерах.

Безпека (NFR-03) є обов’язковою вимогою для будь-якої системи, що зберігає облікові дані, навіть у MVP-версії: компрометація паролів або несанкціонований доступ до адмін-функцій можуть заподіяти шкоду користувачам і дискредитувати платформу. Зручність використання (NFR-04) та локалізація (NFR-05) безпосередньо визначають доступність платформи для цільової аудиторії. Вимога «не більше 3 кроків» для гравця є не просто зручністю, а конкурентною перевагою: у класному середовищі втрата 2-3 хвилин на технічні труднощі з входом руйнує динаміку заняття. Підтримуваність (NFR-06) і масштабованість (NFR-07) є архітектурними вимогами, що забезпечують стійкість проєкту у часі. Завдяки ізоляції шарів через Repository-патерн [29] заміна IMemoryCache на Redis не потребуватиме змін у Services. Цю залежність архітектурно задокументовано у підрозділі 2.3 (ADR-02). Аналогічно, NFR-03 (безпека) підтримується архітектурним рішенням: параметризовані запити EF Core є наслідком використання Repository-патерну, а не окремо доданим засобом захисту, що демонструє взаємозалежність атрибутів якості на рівні проєктних рішень.

Ключові користувацькі сценарії формалізовано в табл. 2.3 у форматі користувацьких історій та «Дано / Коли / Тоді» (критерій приймання), що забезпечує однозначну верифікацію кожного сценарію під час тестування. Визначені сценарії UC-01-UC-04 охоплюють основні потоки взаємодії для кожного з трьох акторів системи і формують основу для функціонального тестування.

Таблиця 2.3 – Ключові користувацькі сценарії Real-Time Quiz Hub

ID	Роль	Сценарій	Критерій приймання
UC-01	Гравець	Як гравець, я хочу приєднатися до активної ігрової сесії, ввівши лише нікнейм, щоб мати змогу брати участь у вікторині без необхідності проходити реєстрацію.	Дано: активна сесія існує у системі, ведучий запустив POST /api/Quiz/start; Коли: гравець вводить нікнейм і натискає кнопку "Почати"; Тоді: гравець реєструється у SignalR-хабі та додається до групи сесії, лідерборд оновлюється для всіх учасників через BroadcastLeaderboard, гравець бачить перше запитання вікторини – не більше 3 кроків від відкриття сторінки (NFR-04).
UC-02	Ведучий	Як ведучий, я хочу запустити вікторину та послідовно отримувати запитання, щоб мати змогу керувати ходом гри у реальному часі.	Дано: ведучий автентифікований і має валідний JWT-токен у заголовку Authorization; Коли: ведучий надсилає POST /api/Quiz/start, а потім послідовно викликає GET /api/Quiz/{quizId}/next; Тоді: система повертає запитання у правильному порядку відповідно до OrderIndex, сесія залишається активною у in-memory сховищі між запитами.
UC-03	Гравець	Як гравець, я хочу надіслати відповідь та одразу побачити оновлений лідерборд, щоб відстежувати свою позицію відносно інших учасників після кожного запитання.	Дано: гравець підключений до сесії та бачить активне запитання; Коли: гравець обирає варіант відповіді і натискає "Відповісти" — система виконує POST /api/Quiz/{quizId}/submit; Тоді: система повертає Toast-повідомлення "Правильно!" або "Неправильно", бали гравця оновлюються, подія BroadcastLeaderboard надходить до всіх учасників сесії через SignalR не пізніше ніж через 2 секунди (NFR-01).
UC-04	Адміністратор	Як адміністратор, я хочу додати нове запитання до банку через API, щоб розширювати контент платформи без необхідності прямого доступу до бази даних.	Дано: адміністратор автентифікований і JWT-токен містить роль Admin у claims [27]; Коли: адміністратор надсилає POST /api/admin/questions із коректним тілом запиту (текст, тип, ліміт часу, бали); Тоді: запитання зберігається у базі даних PostgreSQL, ендпоінт повертає HTTP 201 Created з об’єктом створеного запитання, запитання доступне при GET /api/admin/questions.

Варто звернути увагу на трасованість між сценаріями та нефункціональними вимогами: UC-01 є основним сценарієм перевірки NFR-04 (зручність – не більше 3 кроків), UC-03 – основним сценарієм перевірки NFR-01 (затримка лідерборду ≤ 2с). Такий зв'язок гарантує, що нефункціональні атрибути якості перевіряються у реалістичному контексті використання, а не в ізольованих синтетичних тестах.

2.2 Концептуальне моделювання системи

Концептуальне моделювання дає змогу формалізувати логіку роботи системи до рівня, що є незалежним від конкретної технологічної реалізації, і слугує сполучною ланкою між специфікацією вимог та архітектурними рішеннями. Алгоритм проведення онлайн-вікторини у синхронному режимі відображає повний цикл взаємодії між клієнтом, REST API та SignalR-хабом – від відкриття сторінки до завершення сесії та звільнення ресурсів. Діаграму послідовності наведено на рис. 2.2.

Користувач відкриває вебсторінку (клієнтська частина Bootstrap + JavaScript завантажується без жодних запитів до API), вводить нікнейм – єдиний ідентифікатор гравця без реєстрації і натискає «Почати». Клієнт надсилає POST /api/Quiz/start, сервер ініціалізує сесію в in-memory сховищі (quizId, currentIndex = 0, порожній список учасників) і повертає quizId разом із першим запитанням. Одразу після цього настає перша ключова точка SignalR: клієнт встановлює WebSocket-з'єднання з QuizHub, сервер додає його ConnectionId до SignalR-групи сесії та транслює оновлений склад учасників через BroadcastLeaderboard [25]. Саме тут формується підписка на всі подальші реальночасові оновлення, без необхідності опитувати REST API, що відповідає розмежуванню відповідальностей за ADR-04.

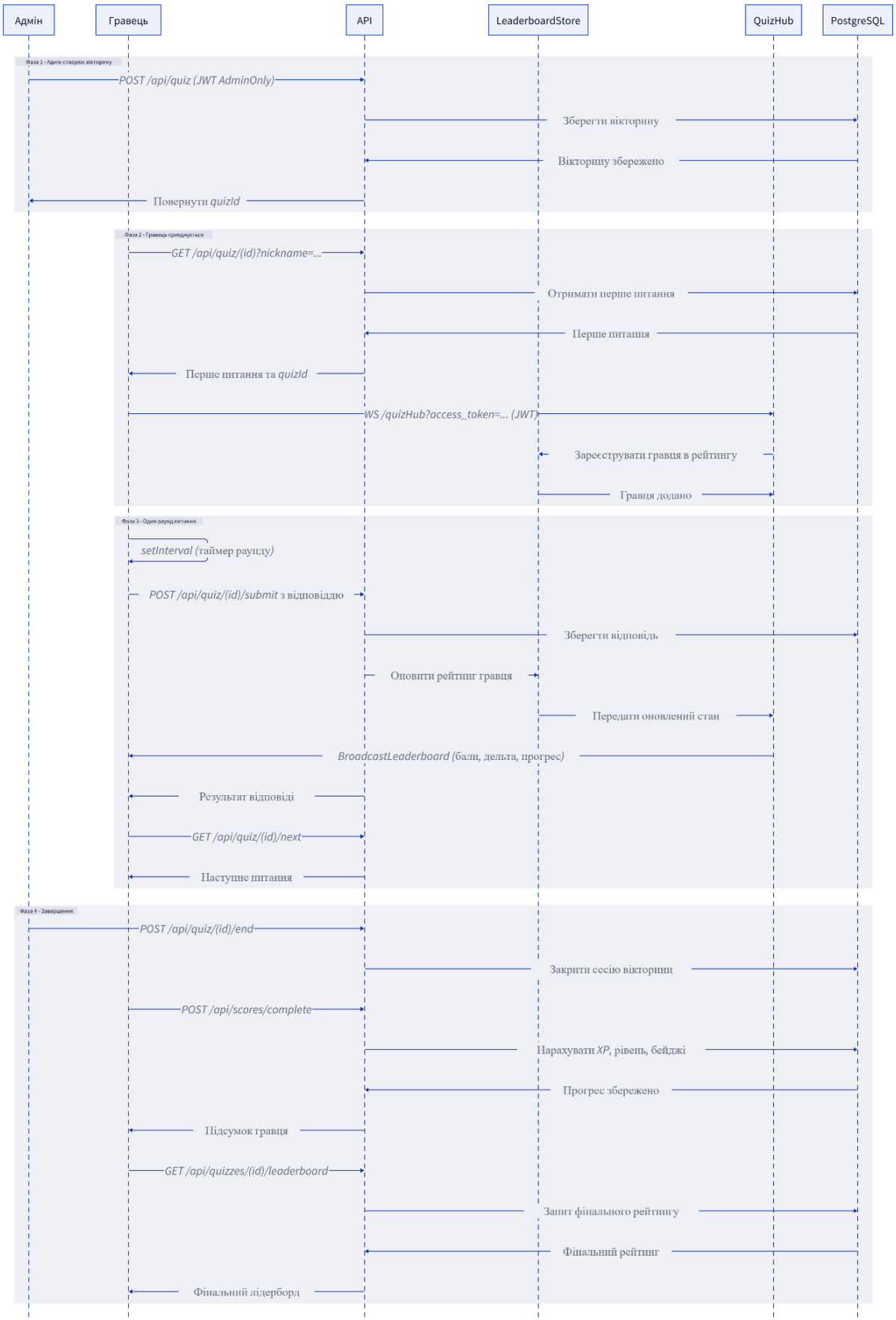

Рисунок 2.2 – Діаграма послідовності типового сценарію проведення онлайн-вікторини у синхронному режимі

Протягом основного ігрового циклу, який повторюється TotalQuestions разів, клієнт отримує чергове запитання через GET /api/Quiz/{quizId}/next (сервер витягує його з PostgreSQL за currentIndex [21]) і відображає його гравцеві; очікування відповіді обробляється винятково на клієнті, без порожніх запитів до сервера. Відповідь надсилається через POST /api/Quiz/{quizId}/submit: для single-choice та multiple-select шар Services/ автоматично перевіряє правильність за полем IsCorrect, для open-ended у MVP передбачено заглушку ручної перевірки. Залежно від результату гравець бачить Toast «Правильно!» з нарахуванням балів або «Неправильно» без зміни рахунку. Далі настає друга ключова точка SignalR: оновлений лідерборд транслюється через BroadcastLeaderboard одночасно всім клієнтам групи [25]. Це принципово ефективніше за REST polling, який при 100 гравцях генерував би ~200 запитів/с лише на оновлення лідерборду [24; 25]. Якщо запитання ще є, цикл повертається до завантаження наступного.

У ході завершення сесії запит POST /api/Quiz/{quizId}/end запускає фінальний підрахунок: результат кожного гравця формується як X / TotalQuestions і відображається разом із фінальною таблицею лідерів. Після цього сесія видаляється з in-memory сховища, а SignalR-група розпускається; у MVP результати не записуються до PostgreSQL. Кнопка «Грати знову» повертає гравця до початку алгоритму з новим quizId та новим WebSocket-з’єднанням. Загалом алгоритм демонструє чіткий розподіл: REST API обслуговує операції «запит-відповідь» (старт, запитання, відповідь, завершення), а SignalR – push-сповіщення (реєстрація, трансляція лідерборду), що відповідає ADR-04 і дозволяє тестувати канали незалежно.

2.3 Попередня архітектура програмного забезпечення

Документування ключових архітектурних рішень здійснювалось за допомогою ведення реєстру (ADR). Для Real-Time Quiz Hub визначено п’ять ключових рішень, наведених у табл. 2.4. Компромісом за першим рішенням щодо вибору протоколу реальночасової комунікації стала складніша конфігурація на

proxy та в хмарі (sticky sessions або Redis backplane для масштабування); також клієнту потрібна бібліотека @microsoft/signalr. У той же час, зберігання стану активної сесії за допомогою IMemoryCache призводить до прив'язаності стану до процесу: не синхронізується при перезапуску чи горизонтальному масштабуванні. Для production-версії програмного забезпечення потрібен Redis.

Обрані механізми автентифікації призводять до того, що токен не відкликати до закінчення TTL без підтримання чорного списку або refresh-механізму; refresh-токени в MVP не реалізовано, компрометація токена дає доступ до завершення TTL. Ще одним компромісом стало те, що клієнт підтримує два канали (HTTP і WebSocket) одночасно; тобто потрібна синхронізація стану між ними. Разом з тим, контейнеризація додаватиме залежність від Docker у середовищі розробника; дані PostgreSQL зберігаються лише в томі pgdata (потребує керування резервним копіюванням), а секрети в .env вимагатимуть обережного поводження.

Описані рішення взаємопов'язані: ADR-01 (SignalR) зумовив ADR-04 (розподіл REST/SignalR); ADR-02 (in-memory стан) спирається на ADR-01, забезпечуючи швидке формування лідерборду без зайвих звернень до PostgreSQL; ADR-03 (JWT) є передумовою безпечного виконання ADR-04; ADR-05 (контейнеризація) забезпечує відтворюване розгортання всієї сукупності. Архітектура є оптимальною для MVP-скоупу й підготовленою до еволюції: перехід IMemoryCache → Redis (ADR-02) і додавання refresh-токенів (ADR-03) можуть бути реалізовані незалежно, без перепроєктування системи.

На основі визначених базових акторів та вимог до програмної системи було проведено архітектурне моделювання засобами C4-діаграми контейнерів (рис. 2.3). Клієнтський застосунок має реалізовуватись як нативний клієнт на HTML, CSS і JavaScript (ES6+) без використання додаткових фронтенд-фреймворків. Усі контракти взаємодії з бекендом визначено за двома стеками комунікації: HTTPS/REST/JSON для операцій із семантикою запит-відповідь і WSS/SignalR/JSON для оновлень у реальному часі.

Таблиця 2.4 – Ключові архітектурні рішення проєкту

ID	Рішення	Розглянуті альтернативи	Обране рішення	Обґрунтування
ADR-01	Протокол комунікації в реальному часі	WebSocket (прямий), Server-Sent Events, Long Polling	ASP.NET Core SignalR (WebSocket із автодеграда-цією)	Full-duplex для двостороннього обміну (відповідь гравця / трансляція лідерборду); автодеградація до SSE або Long Polling забезпечує сумісність; груповий broadcast через Groups API без ітерації по з'єднаннях.
ADR-02	Зберігання стану активної сесії	Redis, IMemoryCache, локальні змінні класу, PostgreSQL	In-memory через IMemoryCache із можливістю заміни на Redis без змін у Services/	Для MVP (до 100 гравців) Redis надлишковий; ізоляція шарів дозволяє замінити кеш лише зміною реєстрації в Program.cs; запис у PostgreSQL на кожну відповідь – надмірне навантаження.
ADR-03	Механізм автентифіка-ції	JWT-токени, Cookie-сесії, OAuth2/OIDC	JWT у заголовку Authorization: Bearer {token}	Stateless – спрощує масштабування; роль (Player / Host / Admin) у claims дозволяє перевіряти авторизацію без звернення до БД; стандарт де-факто для REST API.
ADR-04	Розподіл відповідальн остей REST і SignalR	Усі операції через SignalR; усі операції через REST (real-time через polling)	CRUD та автентифікація — через REST; real-time події — через SignalR	REST дає чітку семантику запит-відповідь, коди HTTP, кешування й незалежне тестування; SignalR оптимальний для push-подій; розподіл знижує зв'язність Controllers/ та Hubs/.
ADR-05		Запуск без контейнерів; Docker + docker-compose; повна оркестрація (Kubernetes)	Docker із docker-compose: сервіси app і db, том pgdata для даних PostgreSQL, конфігурація через .env [30; 31]	Забезпечує відтворюваність середовища, ізоляцію залежностей та однакову поведінку на різних машинах; спрощує розгортання й перенесення між середовищами; повна оркестрація для MVP надлишкова.

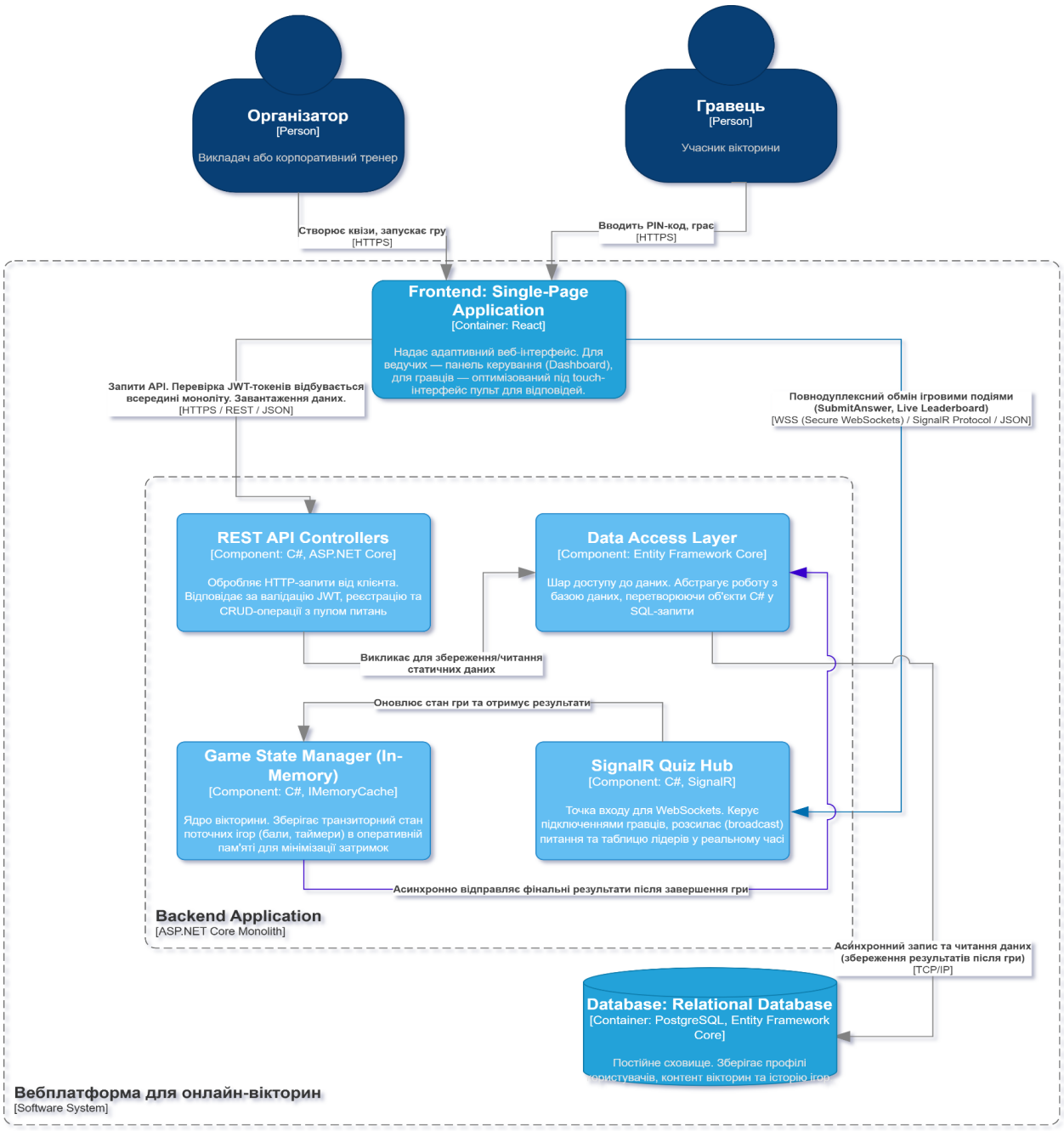

Рисунок 2.3 – C4-діаграма контейнерів системи Real-Time Quiz Hub

Бекенд передбачається для реалізації як модульний моноліт на C# (ASP.NET Core) із підключеним SignalR. Він обробляє HTTP-запити REST API, керує in-memory станом активних сесій, транслює події через WebSocket і містить бізнес-логіку. Вибір моноліту замість мікросервісів обґрунтовано в ADR-02: для MVP-скоупу (до 100 гравців) додаткова складність мікросервісів не виправдана, а шарувата структура забезпечує достатнє розділення відповідальностей.

База даних є реляційною та проєктується засобами СКБД PostgreSQL та ORM-фреймворку Entity Framework Core. Вона зберігає облікові записи та ролі користувачів, банк запитань і варіантів відповідей, каталог вікторин та дані гейміфікації. Стан активних сесій у реальному часі до PostgreSQL не записується (обґрунтовано в ADR-02). Взаємодія відбувається винятково через шар Repository/ з параметризованими запитами EF Core, що виконує вимогу NFR-03 (захист від SQL-ін'єкцій).

На рівні кешу заплановано використання IMemoryCache (.NET) – вбудованого механізму кешування в пам'яті процесу [23], достатнього для MVP-скоупу. Завдяки ізоляції шарів (NFR-07) перехід на зовнішній Redis можливий без змін у Services/ та Repository/.

2.4 Моделювання даних для програмного забезпечення

Для Real-Time Quiz Hub модель даних є гібридною: частина даних зберігається у реляційній базі даних PostgreSQL для забезпечення стійкості між сеансами, а оперативний стан активних ігрових сесій підтримується у пам'яті процесу. У ході аналізу бізнес-правил було виокремлено 9 сутностей, зв'язки між якими наведено в табл. 2.5. Зв'язок «багато-до-багатьох» між вікторинами та запитаннями реалізовано через таблицю QuizQuestions, а між користувачами та бейджами – через таблицю UserBadges. Схему спроєктовано засобами Entity Framework Core за підходом Code First: сутності визначено як POCO-класи в директорії Models/, а схема таблиць генерується механізмом міграцій EF Core. ER-діаграму бази даних наведено на рис. 2.4.

Таблиця Users зберігає облікові записи зареєстрованих користувачів. Ключові атрибути: Id (первинний ключ), Username, Email, PasswordHash (bcrypt-хеш пароля; відкритий пароль у базі не зберігається, що відповідає вимозі NFR-03), Role (Player / Host / Admin – значення передається у JWT claims при автентифікації), CreatedAt. Таблиця є центральною для механізму автентифікації: при запиті POST /api/auth/login сервер знаходить запис за

Username або Email, верифікує PasswordHash і формує JWT-токен із відповідною роллю.

Таблиця 2.5 – Зв’язки між сутностями бази даних Real-Time Quiz Hub

Сутність 1	Кардинальність	Сутність 2	Коментар
Questions	1 : N	Answers	Запитання має довільну кількість варіантів; варіант належить рівно одному запитанню (зовнішній ключ QuestionId). Видалення запитання каскадно видаляє його варіанти (EF Core Cascade Delete).
Quizzes	1 : N	QuizQuestions	Вікторина містить упорядкований набір записів зв’язку.
Questions	1 : N	QuizQuestions	Одне запитання може входити до кількох вікторин.
Users	1 : N	UserScores	Один користувач має багато записів балів – по одному на кожну зіграну вікторину.
Users	1 : 1	UserStats	Агрегована статистика – рівно один рядок на користувача (первинний ключ UserId).
Users	1 : N	UserBadges	Користувач може заробити багато бейджів.
Badges	1 : N	UserBadges	Один бейдж може бути присвоєний багатьом користувачам.

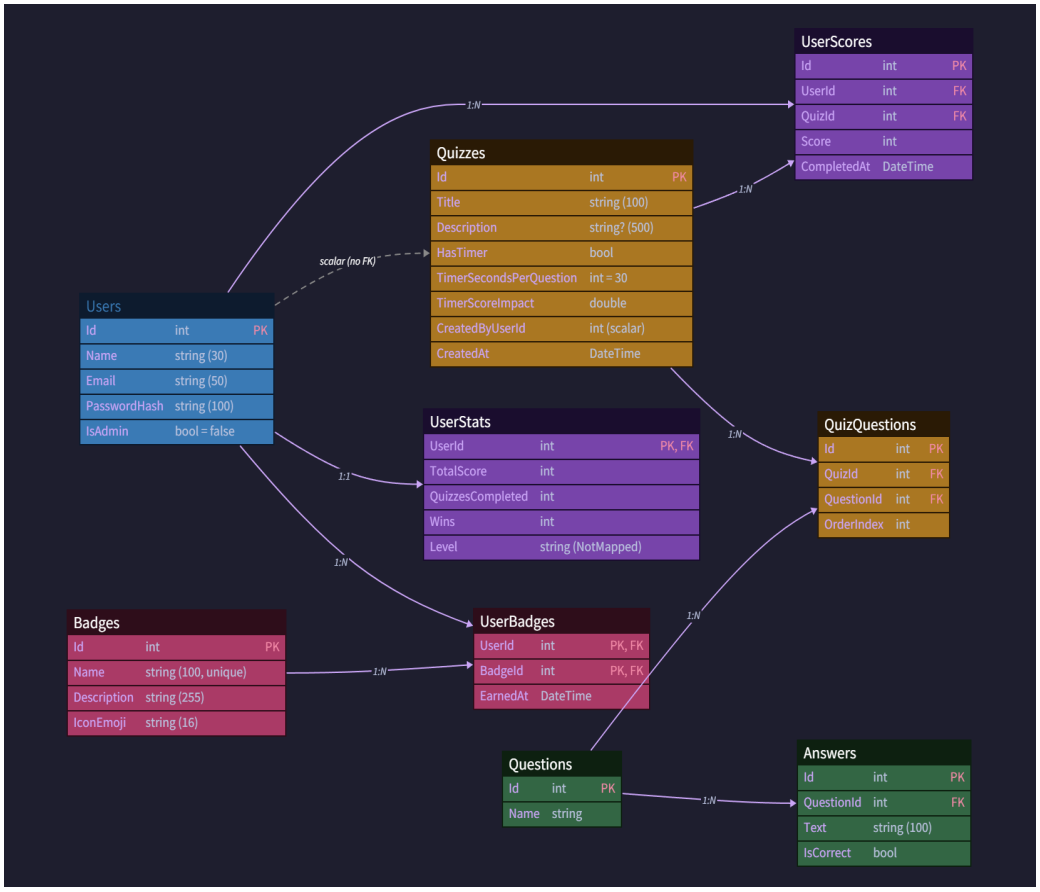

Рисунок 2.4 – ER-діаграма бази даних Real-Time Quiz Hub

Таблиця Questions є банком запитань. Ключові атрибути: Id, Text (текст запитання), Type (single-choice / multiple-select / open-ended), TimeLimit (ліміт часу на відповідь у секундах, може бути відсутній), Points (кількість балів за правильну відповідь), OrderIndex (порядковий номер). Запитання пов’язуються з вікторинами за схемою «багато-до-багатьох» через проміжну таблицю QuizQuestions, що дозволяє повторно використовувати одне запитання в різних вікторинах.

Таблиця Answers зберігає варіанти відповідей до запитань типів single-choice і multiple-select. Атрибути: Id, QuestionId (зовнішній ключ на таблицю Questions), Text, IsCorrect. Зв’язок «багато-до-одного» дозволяє запитанню мати довільну кількість варіантів; для запитань типу open-ended записи не створюються, оскільки перевірка таких відповідей виконується вручну. Поле IsCorrect є основою автоматичної перевірки відповідей у шарі Services/.

Таблиця Quizzes містить метадані вікторин як окремих іменованих наборів запитань, з яких формується каталог платформи. Склад запитань кожної вікторини задається через таблицю QuizQuestions.

Таблиця QuizQuestions є явною проміжною сутністю, що реалізує зв’язок «багато-до-багатьох» між вікторинами та запитаннями. Атрибути: Id (первинний ключ), QuizId (зовнішній ключ на таблицю Quizzes), QuestionId (зовнішній ключ на таблицю Questions), OrderIndex (порядковий номер запитання в межах конкретної вікторини). Власний первинний ключ і поле порядку відрізняють цю таблицю від неявного зв’язку «багато-до-багатьох» та дозволяють керувати послідовністю запитань індивідуально для кожної вікторини.

Таблиця UserScores зберігає бал за кожну окрему зіграну вікторину (поле Score) і містить багато рядків на одного користувача — по одному на кожну завершену сесію. Ці записи є джерелом даних для глобального та поквікторинного рейтингів.

Таблиця UserStats зберігає агреговану статистику користувача — рівно один рядок на обліковий запис (первинний ключ UserId). Ключовим атрибутом є

TotalScore (сумарна кількість досвіду, накопиченого за всі зіграні вікторини), що є основою механіки рівнів гейміфікації.

Рівень гравця (Level) у базі даних не зберігається: відповідну властивість позначено атрибутом [NotMapped] і виключено з відображення в конфігурації контексту, тому EF Core не створює для неї стовпця. Рівень обчислюється динамічно на основі поля TotalScore допоміжним класом LevelHelper, який є єдиним джерелом градації. Передбачено п’ять рівнів: «Новачок» (0–499), «Знавець» (500–1499), «Досвідчений» (1500–3999), «Експерт» (4000–7999) та «Майстер вікторин» (8000 балів і більше). Межі діапазонів неперервні, тому рівень завжди узгоджений із поточним значенням TotalScore без потреби зберігати його окремо.

Таблиця Badges є каталогом визначень бейджів (досягнень), доступних для отримання. Таблиця UserBadges фіксує зароблені користувачами бейджі, реалізуючи зв’язок «багато-до-багатьох» між таблицями Users і Badges.

Система Real-Time Quiz Hub оперує двома принципово різними категоріями даних, що відрізняються за характером, частотою змін та вимогами до збереження. Для кожної категорії обрано відповідне сховище, що є архітектурним рішенням ADR-02.

Розмежування двох зон зберігання відображено нижче. Зона 1 – персистентне сховище (PostgreSQL). До неї належать дані, що повинні зберігатися між сесіями та після перезапуску сервера: облікові записи користувачів та їх ролі (таблиця Users), банк запитань із варіантами відповідей (таблиці Questions та Answers), а також результати після завершення сесій (заплановано у повній версії через таблиці Sessions та SessionParticipants). Дані цієї зони змінюються відносно рідко: облікові записи створюються при реєстрації, запитання – при наповненні банку, результати – один раз після завершення кожної сесії. Запити до PostgreSQL виконуються виключно через шар Repository/ із використанням параметризованих запитів EF Core [32], що забезпечує безпеку (NFR-03) та ізоляцію бізнес-логіки від деталей збереження.

Зона 2 – оперативний стан (in-memory). До неї належать дані, що існують винятково протягом активної ігрової сесії і не потребують довгострокового збереження: список підключених гравців та їх нікнейми (прив'язані до SignalR ConnectionId), поточна позиція у запитаннях для кожної активної сесії (quizId → currentIndex), поточні бали кожного гравця (використовуються для формування об'єкта лідерборду при кожному BroadcastLeaderboard). Дані цієї зони змінюються з дуже високою частотою: при 100 одночасних гравцях рахунок може оновлюватися до 100 разів на секунду під час відповідей на одне запитання.

Механізм переносу між зонами реалізується при виклику POST /api/Quiz/{quizId}/end. Шар Services/ отримує сигнал завершення сесії, витягує з in-memory структури агрегований результат (кількість правильних відповідей кожного гравця за всю сесію), формує підсумкові записи і, у повній версії, персистує їх до таблиці SessionParticipants у PostgreSQL. Після успішного запису in-memory структури поточної сесії очищаються: запис за ключем quizId видаляється зі сховища, SignalR-група для цього quizId розпускається, пам'ять процесу звільняється.

2.5 Реалізація програмного продукту

Детальну структуру шарів наведено у табл. 2.5. Принципово важливою властивістю описаної архітектури є напрямок залежностей: верхні шари залежать від нижніх, але не навпаки. Controllers/ та Hubs/ залежать від Services/ через інтерфейси, проте Services/ не містить жодних посилань на ASP.NET Core MVC або SignalR. Це означає, що бізнес-логіка у Services/ є повністю ізольованою від транспортного шару і може виконуватися та тестуватися у будь-якому середовищі.

Використання DI-контейнера є архітектурною передумовою для тестованості системи відповідно до NFR-06. Оскільки залежності впроваджуються через конструктори у вигляді інтерфейсів (наприклад, IQuizService, IQuestionRepository), під час unit-тестування реальні реалізації можуть бути замінені mock-об'єктами. Це дозволяє тестувати шар Services/ без

реального підключення до PostgreSQL або SignalR-хабу: тести виконуються ізольовано, детерміновано і без зовнішніх залежностей, що є ключовою умовою надійного автоматизованого тестування [33]. Аналогічно, саме завдяки DI-контейнеру заміна IMemoryCache на Redis у шарі Repository/ не потребуватиме змін у Services/ — новий клас реалізуватиме той самий інтерфейс і буде зареєстрований у Program.cs, що відповідає вимозі NFR-07.

Таблиця 2.5 – Шари архітектури бекенду Real-Time Quiz Hub

Шар	Директорія	Відповідальність	Технологія / Патерн	Залежить від
Контролери	Controllers/	Обробка вхідних HTTP-запитів, маршрутизація, серіалізація запит/відповідь, перевірка JWT-авторизації на рівні атрибутів [Authorize]	ASP.NET Core MVC, атрибути авторизації	Services/
SignalR-хаб	Hubs/	Отримання повідомлень від клієнтів (SubmitAnswer), делегування бізнес-логіки до Services/, надсилання подій усім клієнтам групи (BroadcastLeaderboard) [25]	ASP.NET Core SignalR, групи клієнтів	Services/
Сервіси	Services/	Бізнес-логіка: перевірка відповідей, нарахування балів, управління станом ігрових сесій, формування об’єкта лідерборду	Патерни DI, SOLID, ізоляція від транспортного шару	Repository /
Репозиторій	Repository/	Ізоляція запитів до бази даних та кешу; надання абстракції доступу до даних для шару Services/	Патерн Repository [29], EF Core	Data/
Моделі	Models/	POCO-класи сутностей: Users, Questions, Answers та четверта таблиця	Чисті класи C# без залежностей	—
Дані	Data/	Конфігурація контексту Entity Framework Core; відображення сутностей на таблиці бази даних	EF Core, AppDbContext [32]	Models/
Міграції	Migrations/	Версіонування схеми бази даних PostgreSQL; збереження історії змін структури таблиць	EF Core Migrations [32]	Data/
Точка входу	Program.cs	Налаштування DI-контейнера, конвеєру middleware (HTTPS → аутентифікація → авторизація → SignalR → маршрутизація), конфігурація CORS, JWT, Swagger	ASP.NET Core	Всі шари

У підрозділі описано втілення обґрунтованих раніше архітектурних і проєктних рішень: структуру кодової бази, програмний інтерфейс REST, реальночасовий канал на основі SignalR і клієнтську частину.

Кодову базу організовано за шарувальним принципом, що відповідає архітектурі з підрозділу 2.3 та забезпечує розділення відповідальностей (NFR-06). Основні директорії й файли наведено в таблиці 2.6.

Таблиця 2.6 – Структура репозиторію Real-Time Quiz Hub

Директорія / файл	Призначення
Controllers/	REST-контролери: автентифікація, адміністрування запитань і відповідей, керування вікторинами, рейтинги та результати
Hubs/	SignalR-хаб QuizHub (методи SubmitAnswer, BroadcastLeaderboard)
Services/	Бізнес-логіка: перевірка відповідей, керування сесіями, нарахування балів і XP, присвоєння бейджів
Repository/	Доступ до бази даних через Entity Framework Core
Models/	Дев'ять сутностей (Users, Questions, Answers, Quizzes, QuizQuestions, UserScores, UserStats, Badges, UserBadges) та допоміжний клас LevelHelper для обчислення рівня
Data/	AppDbContext.cs — конфігурація EF Core
Migrations/	Міграції схеми бази даних
wwwroot/	Статичні файли клієнта: HTML, CSS, JavaScript
Program.cs	DI-контейнер, middleware, налаштування CORS, JWT та SignalR
Dockerfile, docker-compose.yml, .env	Контейнеризація та конфігурація середовища (ADR-05)

Центральною точкою конфігурації є Program.cs, де реєструються залежності й визначається порядок middleware (HTTPS → автентифікація → авторизація → маршрутизація → SignalR). Коректна послідовність є критичною для роботи захищених ендпоінтів і real-time каналу. REST API реалізує операції з чіткою семантикою запит-відповідь (ADR-04). Ендпоінти згруповано за функціональним призначенням; повний перелік наведено в таблиці 2.7.

Захищені ендпоінти валідуються за схемою Authorization: Bearer {token}; перевірку налаштовано через AddAuthentication().AddJwtBearer() у Program.cs [27]. Жива ігрова сесія проходить цикл: POST /api/Quiz/start → повторювані GET /api/Quiz/{quizId}/next і POST /api/Quiz/{quizId}/submit → POST /api/Quiz/{quizId}/end. Після завершення підсумковий результат персистується через POST /api/scores/complete, що нараховує XP, оновлює

рівень (LevelHelper) і присвоює бейджі, реалізуючи механіки постійної гейміфікації (FR-12).

Таблиця 2.7 – Ендпоінти REST API

Метод	Ендпоінт	Призначення	JWT
Автентифікація (AuthController)			
POST	/api/auth/register	Реєстрація користувача	Ні
POST	/api/auth/login	Вхід, видача JWT-токена	Ні
Адміністрування запитань (AdminQuestionsController)			
GET	/api/admin/questions	Список запитань	Так
POST	/api/admin/questions	Створити запитання	Так
GET	/api/admin/questions/{id}	Запитання за ID	Так
PUT	/api/admin/questions/{id}	Оновити запитання	Так
DELETE	/api/admin/questions/{id}	Видалити запитання	Так
Адміністрування відповідей (AdminAnswersController)			
GET	/api/admin/answers	Список варіантів	Так
POST	/api/admin/answers	Створити варіант	Так
GET	/api/admin/answers/{id}	Варіант за ID	Так
PUT	/api/admin/answers/{id}	Оновити варіант	Так
DELETE	/api/admin/answers/{id}	Видалити варіант	Так
Каталог вікторин			
GET	/api/quizzes	Перелік вікторин (каталог)	Ні
GET	/api/quizzes/{id}	Вікторина за ID	Ні
GET	/api/quizzes/{id}/leaderboard	Рейтинг конкретної вікторини	Ні
POST	/api/quizzes	Створити вікторину	Так
DELETE	/api/quizzes/{id}	Видалити вікторину	Так
Жива ігрова сесія (QuizController)			
POST	/api/Quiz/start	Запуск сесії	Так
GET	/api/Quiz/{quizId}/next	Наступне запитання	Ні
POST	/api/Quiz/{quizId}/submit	Надіслати відповідь	Ні
GET	/api/Quiz/{quizId}	Інформація про сесію	Ні
POST	/api/Quiz/{quizId}/end	Завершити сесію	Так
Рейтинги та результати			
GET	/api/leaderboard	Глобальний рейтинг	Ні
POST	/api/scores/complete	Підсумок сесії: нарахування XP, рівня, бейджів	Так

Оновлення лідерборду в реальному часі реалізовано через хаб QuizHub (ADR-01). Клієнт підключається до кінцевої точки /quizhub і викликає метод SubmitAnswer(quizId, answer); сервіс оновлює бали в in-memory структурі і транслює подію BroadcastLeaderboard усім клієнтам групи сесії, ідентифікованої за quizId.

Бізнес-логіку винесено у Services/ та ін'єктовано через DI, тому хаб не містить прямих обчислень – це забезпечує тестованість компонентів (NFR-06).

Клієнтську частину реалізовано на нативних вебтехнологіях — HTML, CSS і JavaScript (ES6+) без застосування фронтенд-фреймворку. Взаємодія з REST API відбувається через fetch, а підключення до real-time каналу — через клієнтську бібліотеку SignalR. JWT-токен зберігається в localStorage (ключі quizhub_token, quizhub_user) і додається до заголовків захищених запитів. Ключові екрани інтерфейсу наведено в таблиці 2.8.

Таблиця 2.8 — Компоненти клієнтської частини

Компонент	Призначення	Технологія
Сторінка автентифікації	Гостьовий вхід за нікнеймом, реєстрація та логін	HTML, CSS, JS
Ігровий екран	Запитання, варіанти відповідей, таймер	HTML, CSS, JS
Лідерборд	Таблиця позицій, оновлюється через SignalR	JavaScript
Екран результату	Підсумок X / TotalQuestions, нарахований XP і рівень	HTML, CSS, JS
Адмін-панель	Управління запитаннями та відповідями	HTML, CSS, JS

Вигляд ігрового екрана наведено на рис. 2.5, панель адміністратора – на рис. 2.6. Поточна реалізація клієнта є функціонально завершеною та задовольняє вимогу зручності використання (NFR-04).

Отже, в розділі 2 виконано повний цикл аналітико-модельного та інженерно-технічного опрацювання платформи Real-Time Quiz Hub – від формалізації вимог до реалізованої системи. Спочатку було визначено та формалізовано вимоги: дванадцять функціональних вимог (FR-01-FR-12) у табличній формі з демонстраційними критеріями, сім нефункціональних вимог

із метриками та процедурами підтвердження (NFR-01-NFR-07), а також ключові користувацькі сценарії.

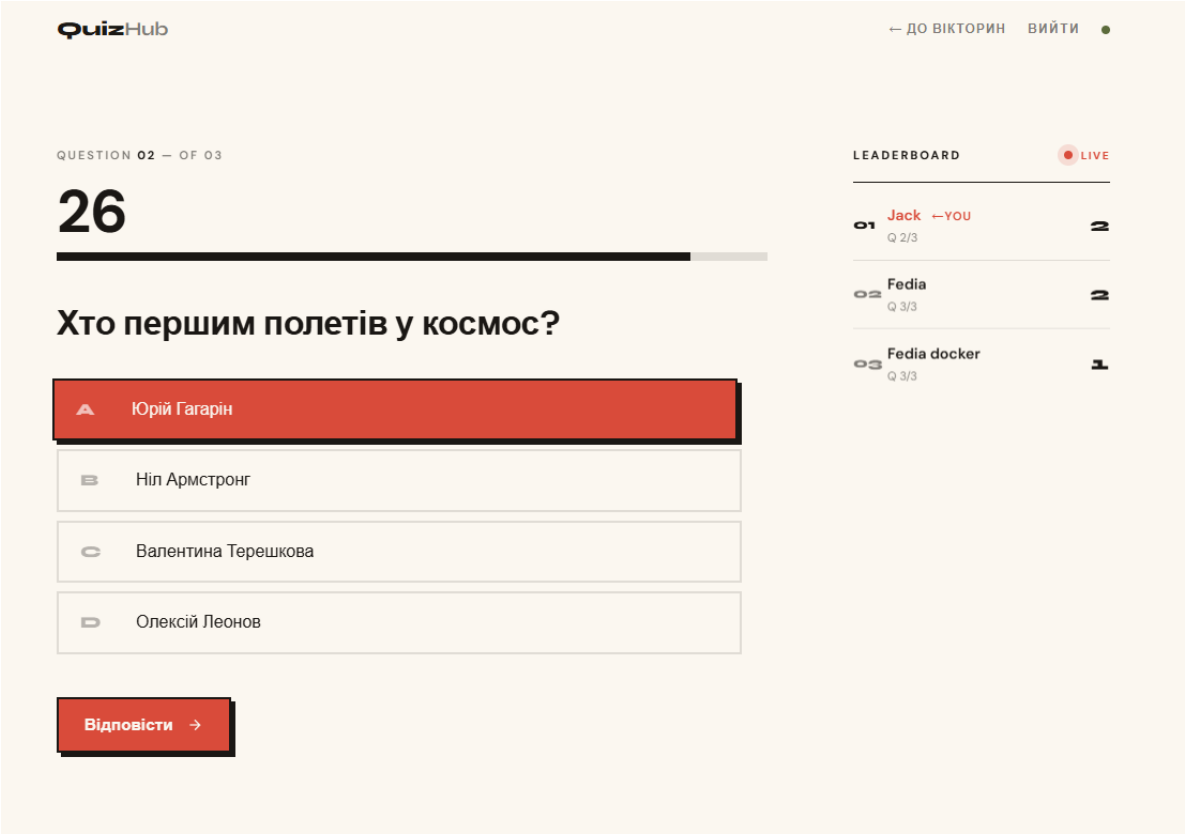

Рисунок 2.5 — Ігровий екран платформи Real-Time Quiz Hub

Далі побудовано концептуальні моделі: схему алгоритму ігрової сесії, що відображає повний цикл від входу гравця до завершення вікторини, та глосарій предметної області. Потім обґрунтовано архітектуру: діаграму контейнерів (C4-модель, рівень 2) і шарувату структуру, що забезпечує виконання NFR-06 та NFR-07; п’ять архітектурних рішень (ADR-01-ADR-05) документують ключові інженерні компроміси, зокрема контейнеризацію засобами Docker.

Наступним кроком було спроєктовано модель даних: реалізовану схему з дев’яти таблиць, що покриває облікові записи, банк запитань, каталог вікторин і механіки постійної гейміфікації; рівень гравця обчислюється динамічно й у базі не зберігається. Описано гібридний підхід до зберігання – персистентний PostgreSQL та оперативний in-memory стан активної сесії.

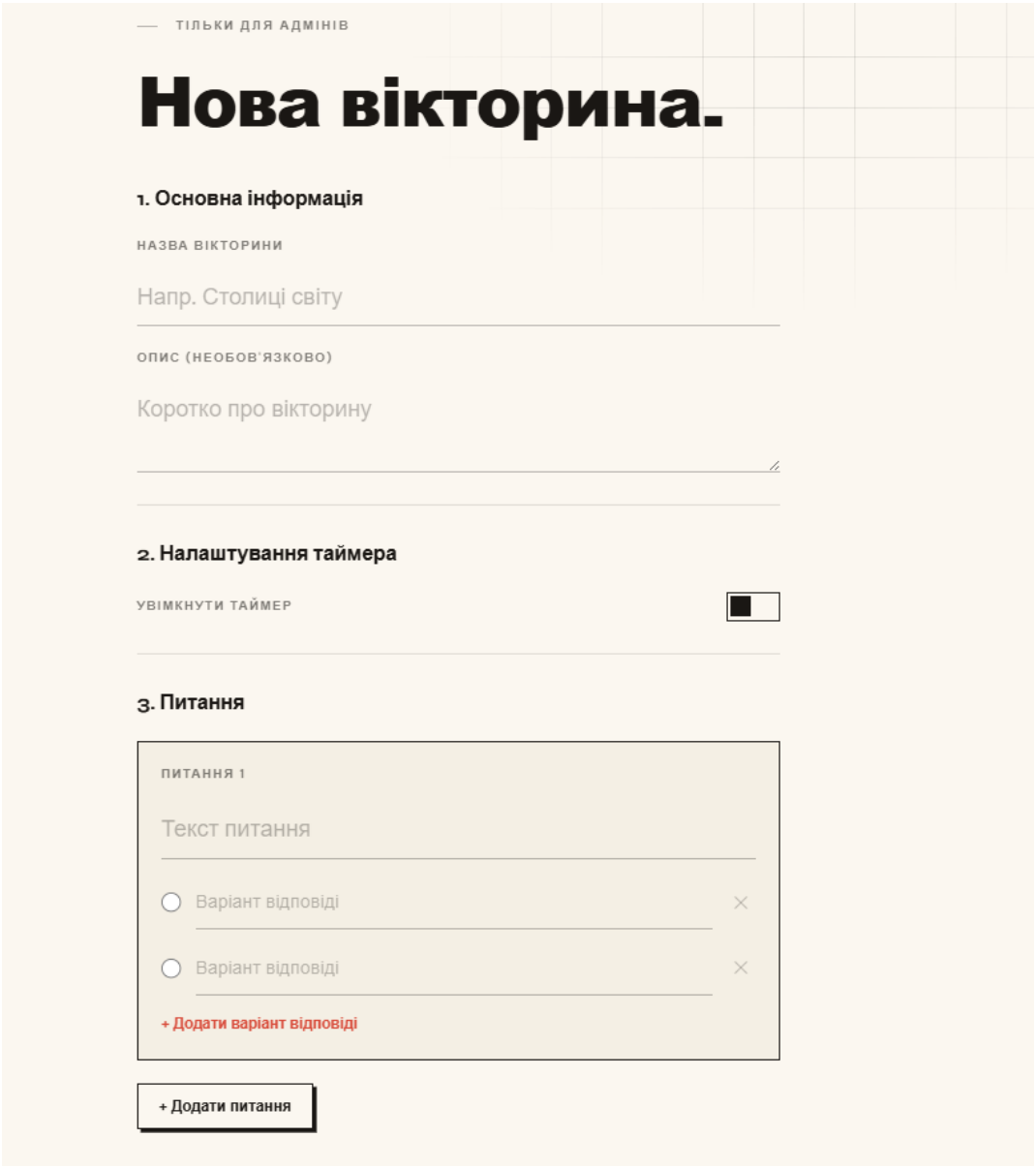

Рисунок 2.6 – Панель адміністратора

Наприкінці було описано реалізовану систему: REST API з кількох функціональних груп (автентифікація, адміністрування контенту, каталог вікторин, рейтинги та результати сесій), SignalR-хаб для real-time оновлення лідерборду (SubmitAnswer → BroadcastLeaderboard), клієнт на нативних вебтехнологіях і механіки гейміфікації (нарахування XP, динамічний розрахунок рівня, присвоєння бейджів). Систему контейнеризовано засобами Docker для відтворюваного розгортання.

РОЗДІЛ 3 ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА СУПРОВІД ВЕБПЛАТФОРМИ REAL-TIME QUIZ HUB

3.1 Стратегія та план тестування

Стратегія тестування вебплатформи Real-Time Quiz Hub побудована за принципом поетапного охоплення системи: від ізольованої перевірки окремих компонентів до комплексної валідації REST API та користувацького інтерфейсу. Такий підхід забезпечує виявлення дефектів якнайближче до місця їх виникнення і мінімізує вартість їх усунення. Для проєкту визначено чотири рівні тестування. Перший рівень – модульне тестування – спрямований на ізольовану перевірку бізнес-логіки шару Services/ засобами фреймворку xUnit [34]. Цей шар обрано пріоритетним об’єктом автоматизованого тестування, оскільки він містить критичну логіку управління ігровими сесіями, нарахування балів та ізоляції стану гравців. Шари Controllers/ та Hubs/ є тонкими обгортками над сервісами і перевіряються на рівні функціонального тестування.

Другий рівень – функціональне тестування – забезпечує перевірку коректності поведінки REST API через HTTP-запити. Інструментом виступає вбудований Swagger UI, що автоматично генерується бібліотекою Swashbuckle.AspNetCore [35] та дозволяє виконувати запити безпосередньо в браузері без встановлення додаткових клієнтів. На цьому рівні підтверджується коректність HTTP-статус-кодів, структури відповідей і JWT-захисту.

Третій рівень – ручне UI-тестування – охоплює перевірку клієнтської частини платформи (Bootstrap+JavaScript) та реального часу SignalR-з’єднань у браузері Google Chrome із застосуванням Chrome DevTools. Цей рівень підтверджує виконання вимог щодо адаптивності інтерфейсу (NFR-04), повноти україномовної локалізації (NFR-05) та коректності відображення лідерборду в реальному часі (FR-07).

Узагальнена стратегія тестування наведена у табл. 3.1. У таблиці для кожного рівня зазначено метод, інструментарій, об’єкт перевірки, поточний

статус та перелік функціональних і нефункціональних вимог, які охоплюються відповідним рівнем.

Таблиця 3.1 – Стратегія тестування платформи Real-Time Quiz Hub

Рівень тестування	Метод	Інструмент	Об’єкт тестування	Статус	Покриває вимоги
Модульне	Unit-тестування	xUnit [34]	Шар Services/ (клас QuizService)	Реалізовано (18 тестів)	NFR-06; FR-04-FR-10
Функціональне	Ручне API-тестування	Swagger UI (Swashbuckle) [35]	Контролери REST API (Controllers/)	Реалізовано (10 ТС)	FR-01–FR-09; NFR-03
Ручне UI	Тестування інтерфейсу в браузері	Google Chrome, Chrome DevTools	Bootstrap+JS-клієнт, SignalR-з’єднання	Реалізовано	NFR-04; NFR-05; FR-07

Коректне визначення критеріїв початку та завершення тестування є обов’язковим елементом плану тестування, оскільки дозволяє чітко обмежити обсяг перевірки й уникнути передчасного або неповного закриття тестового циклу. Критерії початку тестування (entry criteria) для платформи Real-Time Quiz Hub визначено так. Для модульного тестування необхідним є наявність реалізованого шару Services/ з інтерфейсом IQuizService, а також підготовленого набору фейкових тестових даних (_fakeQuestions). Для функціонального тестування обов’язковим є запущений застосунок на локальному хості та доступний Swagger UI за адресою https://localhost:{port}/swagger. Для ручного тестування UI необхідне розгорнуте клієнтське середовище з підключеним Bootstrap+JavaScript і доступним SignalR-хабом.

Критерії завершення тестування (exit criteria) визначають умови, за яких тестовий цикл вважається закритим. Для модульного тестування критеріями успішного завершення є: виконання усіх 18 unit-тестів з результатом «пройдено» (зелений статус у тестовому раннері), відсутність тестів зі статусом «провалено» або «пропущено», а також підтвердження покриття шару Services/ на рівні, що задовольняє вимогу NFR-06. Для функціонального тестування критеріями є: виконання усіх 10 тест-кейсів (ТС-01–ТС-10) із фактичними

результатами, що відповідають очікуваним; підтвердження JWT-захисту (TC-04 повертає 401 Unauthorized без токена). Для ручного UI-тестування – відсутність критичних дефектів, що унеможливлюють виконання основних сценаріїв.

Тестове середовище в усіх реалізованих рівнях є локальним: операційна система Windows, .NET 8 SDK, PostgreSQL 15, браузер Google Chrome актуальної версії. Ізоляція тестового середовища від виробничого забезпечується тим, що застосунок запускається командою dotnet run у режимі Development, який активує Swagger UI та розширене логування.

Отже, визначена стратегія тестування охоплює чотири рівні перевірки, встановлює чіткі критерії початку і завершення для кожного з них, узгоджена з повним переліком функціональних і нефункціональних вимог та забезпечує достатній рівень верифікації MVP для його демонстрації і захисту.

3.2 Модульне тестування

Об’єктом модульного тестування у роботі обрано прикладні сервіси шару Services/ (QuizSessionService, ScoreService, BadgeService, LeaderboardService) як носії бізнес-логіки, а також обрані контролери (AuthController, LeaderboardController), що формують відповіді API. Сервіс QuizSessionService реалізує повний цикл вікторини: ініціалізацію сесії, навігацію питаннями, перевірку відповідей і завершення гри; ScoreService відповідає за нарахування балів і ведення лідербордів; BadgeService – за нагородження бейджами.

Тестовий проєкт організовано як окремий xUnit-проєкт у складі рішення (solution), що забезпечує незалежність збірки, спрощує CI-інтеграцію та розмежовує виробничий і тестовий код. Тести структуровано у п’ять класів: QuizSessionServiceTests, ScoreServiceTests, BadgeServiceTests, AuthControllerTests та LeaderboardControllerTests; допоміжні фейкові реалізації репозиторіїв винесено в файл Fakes/ScoringFakes.cs. Детермінованість забезпечується фіксованим набором _fakeQuestions – трьома заздалегідь визначеними питаннями з відомими правильними відповідями. Доступ до даних абстраговано через інтерфейси репозиторіїв, що дозволяє у тестах застосовувати

фейкові ін-меморі реалізації (Fakes/ScoringFakes.cs) без запущеного сервера PostgreSQL. Завдяки цьому тести виконуються цілком ізольовано та відповідають принципу FIRST (Fast, Isolated, Repeatable, Self-verifying, Timely).

Тест-кейси модульного тестування систематизовано у табл. 3.2; вони відповідають 33 реалізованим xUnit-тестам, при цьому споріднені тести згруповано у межах одного запису. Для ілюстрації реалізованих тестів нижче наведено п’ять ключових лістингів для QuizSessionService: ініціалізація сесії (лістинг 3.1), нарахування балів (лістинг 3.2), завершення циклу питань (лістинг 3.3), очищення пам’яті після гри (лістинг 3.4) та ізоляція паралельних сесій (лістинг 3.5). Вони демонструють характерну особливість усіх 33 тестів: кожен тест починається зі свіжого стану сервісу, що унеможливлює взаємний вплив тестів при паралельному або послідовному виконанні. Це забезпечується ін’єкцією залежностей у конструкторі тестового класу, де для кожного тесту створюється новий екземпляр сервісу з чистим станом.

Лістинг 3.1 – StartQuizAsync: перевірка ініціалізації сесії

```
[Fact]
public async Task StartQuizAsync_МаєПовернутиСесіюЗПравильнимиДаними()
{
  var session = await _service.StartQuizAsync("quiz1", "Alice");
  Assert.Equal("quiz1", session.QuizId);
  Assert.Equal("Alice", session.Nickname);
  Assert.Equal(3, session.TotalQuestions);
  Assert.Equal(1, session.CurrentQuestionIndex);
  Assert.Equal(_fakeQuestions[0].Name, session.CurrentQuestion.Name);
}
```

Лістинг 3.2 – SubmitAnswerAsync: нарахування балів за правильну відповідь

```
[Fact]
public async Task
SubmitAnswerAsync_МаєЗбільшитиCorrectAnswersПриПравильнійВідповіді()
{
  await _service.StartQuizAsync("quiz1", "Alice");
  await _service.SubmitAnswerAsync("quiz1", "Мова програмування", "Alice");
  var session = await _service.GetQuizSessionAsync("quiz1", "Alice");
  Assert.Equal(1, session!.CorrectAnswers);
}
```

Таблиця 3.2 – Тест-кейси модульного тестування

ID	Метод	Назва тесту	Передумова	Очікуваний результат
UT-01	StartQuizAsync	Сесія містить правильні дані	QuizSessionService ініціалізовано з _fakeQuestions	QuizId="quiz1", Nickname="Alice", TotalQuestions=3, Index=1
UT-02	StartQuizAsync	Ізоляція гравців: Аліса і Боб	QuizSessionService ініціалізовано	Дві незалежні сесії, стан не перетинається
UT-03	GetQuizSessionAsync	Доступ до власного прогресу	Сесія для Аліси існує	Сесія повернена, CorrectAnswers=0
UT-04	GetQuizSessionAsync	Null для неіснуючого гравця	Сесія не створювалась	Повернено null
UT-05	GetNextQuestionAsync	Успішний перехід до питання 2	Сесія запущена (Index=1)	Index=2, питання повернено
UT-06	GetNextQuestionAsync	Null після останнього питання	3 питання вже пройдено	Повернено null (кінець вікторини)
UT-07	GetNextQuestionAsync	Ізоляція переходів між гравцями	Аліса і Боб грають quiz1	Index Аліси=2, Index Боба=1
UT-08	SubmitAnswerAsync	Бали за правильну відповідь	Сесія запущена	CorrectAnswers=1
UT-09	SubmitAnswerAsync	Відхилення неіснуючого варіанту	Сесія запущена	CorrectAnswers=0
UT-10	EndQuizAsync	Видалення сесії після завершення	Сесія запущена	GetSession повертає null
UT-11	EndQuizAsync	Ізоляція видалення: сесія Боба збережена	Аліса і Боб грають	Сесія Боба незмінена
UT-12	Ізоляція сесій	Два квізи одночасно незалежні	quiz-A і quiz-B запущені	quiz-A.Index=2, quiz-B.Index=1
UT-13	QuizSessionService (інші)	Додаткові сценарії класу (решта 7 тестів)	Сесія/дані ініціалізовані	Усі пройдено
UT-14	CalculateAnswerScore	Формула балів за відповідь (з таймером і без)	ScoreService з фейковими даними	За таймером застосовано бонус; без таймера – базовий бал
UT-15	CalculateSessionScore	Підсумок балів сесії	Сесія з правильними й хибними відповідями	Сумуються лише правильні відповіді
UT-16	RecordCompletionAsync	Збереження результату й статистики	Фейковий репозиторій	Бал і статистику збережено
UT-17	RecordCompletionAsync	Ізоляція лідербордів різних вікторин	Дві вікторини	Лідерборди вікторин не змішуються
UT-18	BadgeService	Бейдж «Перша перемога»	Перша перемога гравця	Бейдж FirstWin нараховано
UT-19	BadgeService	Бейдж «Перфекціоніст»	Бездоганний результат	Бейдж Perfectionist нараховано
UT-20	BadgeService	Без дублювання бейджа	Бейдж уже отримано	Дубль не створюється
UT-21	LeaderboardController.GetGlobal	Глобальний рейтинг з рангом, рівнем, бейджами	Є статистика гравців	Список із рангом, рівнем і бейджами

Продовження таблиці 3.2

UT-22	LeaderboardController.GetGlobal	Порожній рейтинг	Статистики немає	Повернено порожній список
UT-23	AuthController	Реєстрація (новий e-mail / дублікат)	БД користувачів (фейк)	Новий — Ok з токеном; дублікат — Conflict
UT-24	AuthController	Вхід (правильні / хибні дані)	Користувач існує	Правильні — Ok з токеном; хибний пароль — Unauthorized

Лістинг 3.3 – GetNextQuestionAsync: завершення циклу питань

```
[Fact]
public async Task GetNextQuestionAsync_МаєПовернутиNullКолиПитаньБільшеНемає()
{
  await _service.StartQuizAsync("quiz1", "Alice");
  await _service.GetNextQuestionAsync("quiz1", "Alice");
  await _service.GetNextQuestionAsync("quiz1", "Alice");
  var result = await _service.GetNextQuestionAsync("quiz1", "Alice");
  Assert.Null(result);
}
```

Лістинг 3.4 – EndQuizAsync: очищення пам’яті після завершення гри

```
[Fact]
public async Task EndQuizAsync_МаєВидалитиСесіюПіслязавершення()
{
  await _service.StartQuizAsync("quiz1", "Alice");
  await _service.EndQuizAsync("quiz1", "Alice");
  var session = await _service.GetQuizSessionAsync("quiz1", "Alice");
  Assert.Null(session);
}
```

Повний запуск тестового набору виконано засобами CLI-команди dotnet test. Усі 33 тести завершилися зі статусом «пройдено» (Passed); тестів зі статусом «провалено» (Failed) або «пропущено» (Skipped) не виявлено. Результати запуску раннера xUnit продемонстровано на рис. 3.1. Усі тести відтворювані, детерміновані та не потребують зовнішньої інфраструктури для виконання.

Лістинг 3.5 – Ізоляція: незалежність двох паралельних сесій

```
[Fact]
public async Task Сесії_РізнихКвізівМаютьБутиНезалежними()
{
  await _service.StartQuizAsync("quiz-A", "Alice");
```

```
    await _service.StartQuizAsync("quiz-B", "Alice");
    await _service.GetNextQuestionAsync("quiz-A", "Alice");
    var sessionA = await _service.GetQuizSessionAsync("quiz-A", "Alice");
    var sessionB = await _service.GetQuizSessionAsync("quiz-B", "Alice");
    Assert.Equal(2, sessionA!.CurrentQuestionIndex);
    Assert.Equal(1, sessionB!.CurrentQuestionIndex);
}
```

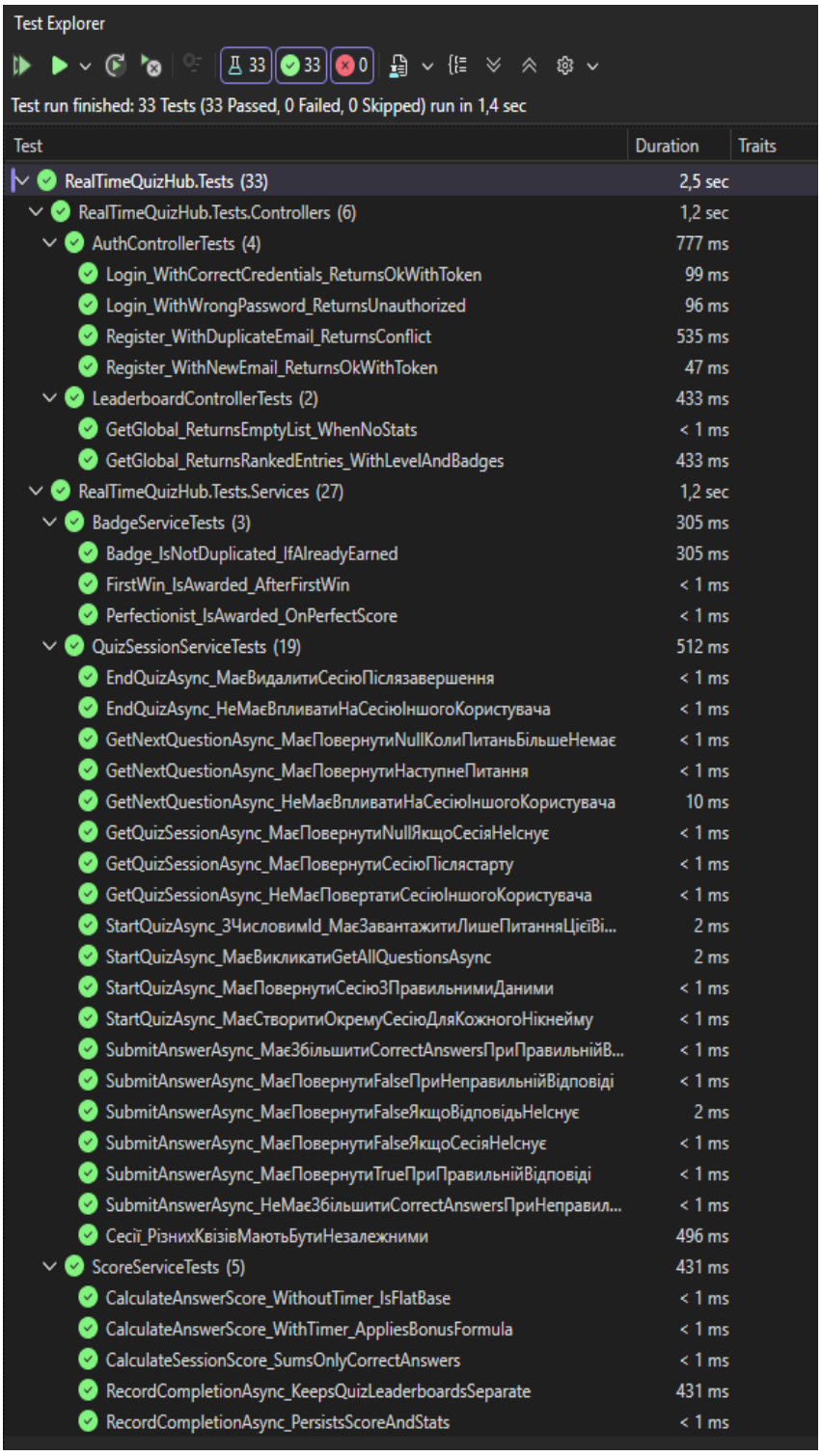

Рисунок 3.1 – Результати запуску модульних тестів у xUnit

3.3 Функціональне тестування REST API

Функціональне тестування REST API платформи Real-Time Quiz Hub виконано засобами вбудованого інтерактивного інтерфейсу Swagger UI, що автоматично генерується бібліотекою Swashbuckle.AspNetCore. Swagger UI підключається у режимі розробки (Development) і стає доступним за адресою https://localhost:{port}/swagger після запуску застосунку командою dotnet run.

Перевагою Swagger UI є відсутність потреби у сторонніх клієнтах: усі HTTP-запити виконуються безпосередньо у браузері. Для тестування захищених ендпоінтів передбачено кнопку «Authorize», через яку вводиться JWT Bearer-токен; після авторизації кожен наступний запит автоматично містить заголовок Authorization: Bearer {token}. Це дозволяє в межах єдиного сеансу перевіряти як захищені, так і публічні ендпоінти.

Тестування проводилося на локальному середовищі: ОС Windows, .NET 8 SDK, PostgreSQL 16, браузер Google Chrome. Базу даних попередньо підготовлено командою dotnet ef database update (застосування EF Core-міграцій). Послідовно виконувалися сценарії реєстрації, авторизації, адміністрування питань, ігрового процесу, роботи з каталогом вікторин і лідербордами відповідно до тест-кейсів ТС-01-ТС-15. Рис. 3.2 відображає загальний вигляд Swagger UI зі списком доступних REST-ендпоінтів (26 ендпоінтів у семи контролерах).

Для функціонального тестування розроблено 15 тест-кейсів (ТС-01–ТС-15), що охоплюють п’ять груп функціональності: автентифікацію й авторизацію (ТС-01–ТС-04), адміністрування питань (ТС-05–ТС-07), ігровий процес (ТС-08–ТС-10), каталог вікторин і лідерборди (ТС-11–ТС-14) та завершення гри з нарахуванням досвіду (ТС-15). Детальний опис кожного тест-кейсу наведено у табл. 3.4.

Тест-кейси ТС-01–ТС-15 підтверджено фактичними запусками. По-перше, вимога FR-01 (автентифікація та авторизація) підтверджена тест-кейсами ТС-01 і ТС-02: реєстрація повертає 200 OK, вхід із коректними даними повертає JWT

Bearer-токен. Тест-кейс TC-03 підтверджує, що за невірного пароля система повертає 401 Unauthorized.

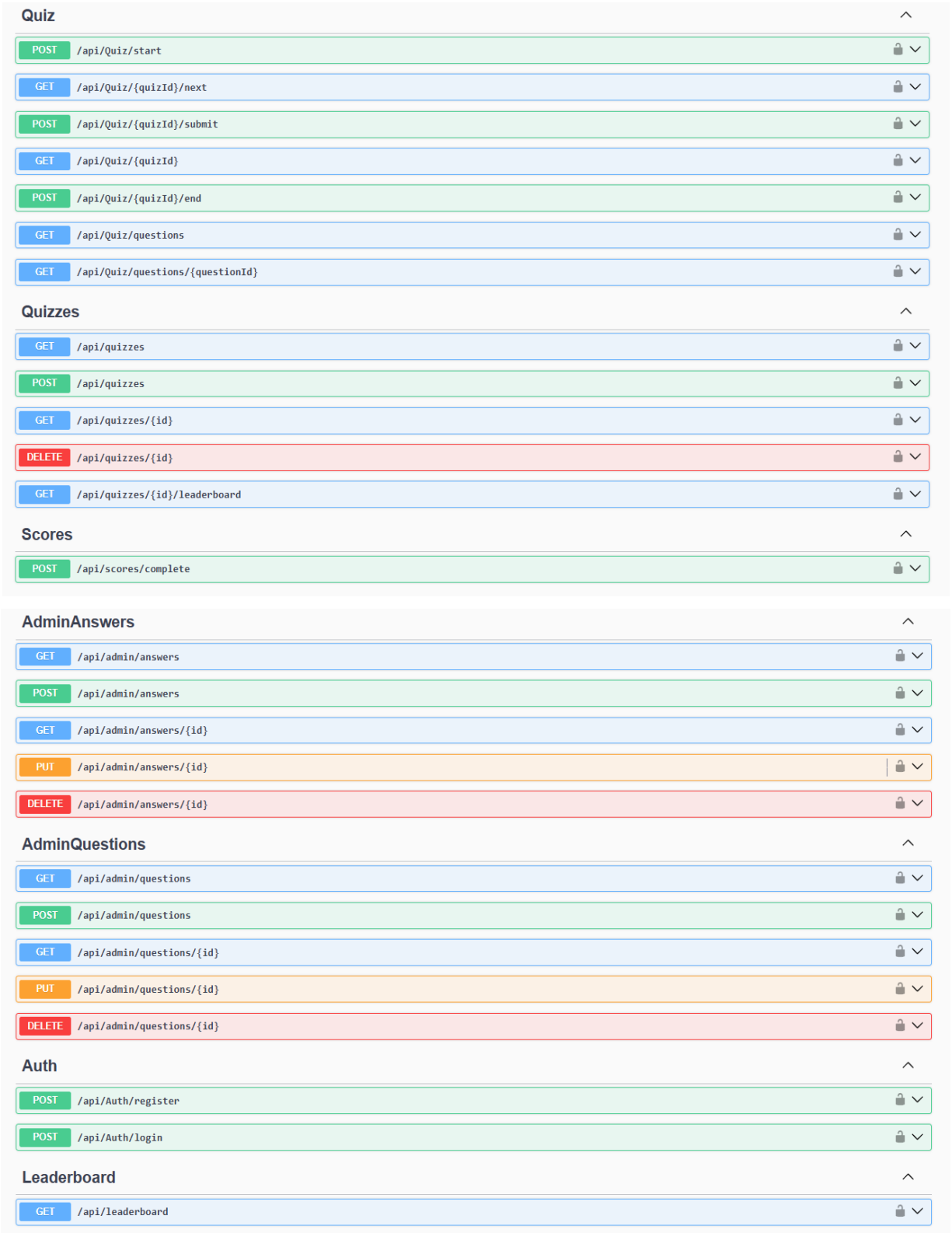

Рисунок 3.2 – Swagger UI платформи Real-Time Quiz Hub

Таблиця 3.4 – Тест-кейси функціонального тестування REST API

ID	Ендпоінт	Передумова	Вхідні дані	Очікуваний результат	Фактичний результат
TC-01	POST /api/Auth/register	Сервер запущено	Валідні email + пароль	200 OK, акаунт створено	200 OK ✓
TC-02	POST /api/Auth/login	Акаунт зареєстровано	Коректні email + пароль	200 OK + JWT у відповіді	200 OK, токен ✓
TC-03	POST /api/Auth/login (невірний пароль)	Акаунт зареєстровано	email + невірний пароль	401 Unauthorized	401 ✓
TC-04	GET /api/admin/questions (без токена)	Сервер запущено	Запит без Authorization	401 Unauthorized	401 ✓
TC-05	POST /api/admin/questions	JWT-токен отримано	Bearer + JSON тіло питання	201 Created	201, додано ✓
TC-06	GET /api/admin/questions	Питання додано (TC-05)	Bearer-токен	200 OK + масив питань	200 OK ✓
TC-07	DELETE /api/admin/questions/{id}	Питання існує	Bearer + валідний id	200/204 No Content	Видалено ✓
TC-08	POST /api/Quiz/start	Питання наявні; JWT отримано	Bearer + quizId	200 OK + перше питання	Сесія ✓
TC-09	GET /api/Quiz/{quizId}/next	Сесія запущена	Валідний quizId	200 OK + наступне питання	200 OK ✓
TC-10	POST /api/Quiz/{quizId}/submit	Сесія запущена; питання є	quizId + варіант відповіді	200 OK, результат перевірки	Бали нараховано ✓
TC-11	GET /api/quizzes	Сервер запущено; вікторини наявні	Запит без параметрів	200 OK + список вікторин	200 OK ✓
TC-12	GET /api/quizzes/{id}	Вікторина існує	Валідний id	200 OK + дані вікторини	200 OK ✓
TC-13	GET /api/quizzes/{id}/leaderboard	Є результати вікторини	Валідний id	200 OK + лідерборд вікторини	200 OK ✓
TC-14	GET /api/leaderboard	Є статистика гравців	Запит без параметрів	200 OK + рейтинг (ранг, рівень, бейджі)	200 OK ✓
TC-15	POST /api/scores/complete	Сесія завершена; JWT отримано	Bearer + результат сесії	200 OK + бал, XP, рівень, бейджі	200 OK ✓, бали пораховані

По-друге, вимога NFR-03 (безпека: JWT-захист) підтверджена тест-кейсом TC-04: запит до захищеного ендпоінту без заголовка Authorization повертає 401 Unauthorized без даних у відповіді, що засвідчує коректне налаштування JWT Middleware у конвеєрі ASP.NET Core.

По-третє, вимога FR-03 (CRUD питань) підтверджена тест-кейсами TC-05–TC-07: питання додається (201), зчитується (200) і видаляється (204). Аналогічний CRUD для відповідей (FR-02) перевірено за тією самою схемою.

По-четверте, вимоги FR-04, FR-05, FR-06 і FR-10, що стосуються ігрового процесу, підтверджені тест-кейсами TC-08–TC-10: сесія ініціалізується, повертає наступне питання та коректно обробляє відповідь з нарахуванням балів.

По-п’яте, нові ендпоінти каталогу вікторин (GET /api/quizzes, GET /api/quizzes/{id}) і лідербордів (GET /api/quizzes/{id}/leaderboard, GET /api/leaderboard) забезпечують реалізацію FR-11 (вибір вікторини), а ендпоінт POST /api/scores/complete – FR-12 (нарахування балів, досвіду, рівнів і бейджів). Очікувані результати цих сценаріїв описано у TC-11-TC-15, їх фактичне підтвердження виконується запуском відповідних запитів у Swagger UI.

Отже, функціональне тестування через Swagger UI підтвердило коректну роботу десяти базових сценаріїв REST API (TC-01-TC-10); ще п’ять сценаріїв (TC-11-TC-15) для нових ендпоінтів каталогу, лідербордів і завершення гри додано до плану перевірки. Вимоги FR-01–FR-12 та NFR-03 верифікуються на рівні HTTP-взаємодії в обсязі, доступному для функціонального тестування.

RTM-матриця платформи Real-Time Quiz Hub охоплює всі 12 функціональних вимог (FR-01–FR-12) та 7 нефункціональних вимог (NFR-01–NFR-07). Для кожної вимоги зазначено пов’язані тест-кейси, метод тестування та підсумковий результат. Повна матриця наведена у табл. 3.5. Усі 12 функціональних вимог (FR-01–FR-12) підтверджено: автентифікація, CRUD-операції, ігровий процес, нарахування балів, вибір вікторини й механіки гейміфікації верифіковані комбінацією unit-тестів і функціонального тестування через Swagger UI. Вимогу FR-07 (SignalR-лідерборд) перевірено ручним

тестуванням у браузері, оскільки автоматизація WebSocket-з'єднань виходить за межі обсягу поточного MVP.

Нові вимоги FR-11 (вибір вікторини) і FR-12 (механіки постійної гейміфікації) покрито на двох рівнях: модульними тестами (ізоляція лідербордів вікторин – UT-17; формула балів – UT-14; нарахування бейджів – UT-18-UT-20; глобальний рейтинг із рангом, рівнем і бейджами – UT-21, UT-22) та функціональною перевіркою нових ендпоінтів API (TC-11-TC-15).

З семи нефункціональних вимог чотири (NFR-02–NFR-05) підтверджено повністю. NFR-02 (ізоляція сесій) верифіковано чотирма xUnit-тестами (UT-02, UT-07, UT-11, UT-12). NFR-03 (безпека) підтверджено тест-кейсами TC-03 і TC-04, що перевіряють відхилення невалідних облікових даних і запитів без JWT-токена.

Вимога NFR-06 (тестованість) перебуває у статусі часткового підтвердження: реалізовано й успішно пройдено 33 xUnit-тести, проте інтегральне покриття коду не досягає типових цільових порогів.

Вимога NFR-07 (масштабованість кешування) також підтверджена частково: на поточному етапі проміжний стан реального часу зберігається у пам'яті процесу (потокобезпечні структури в межах singleton-сервісів), без окремої абстракції IMemoryCache та без сервісу Redis у docker-compose. Перехід на Redis як спільне розподілене сховище зафіксовано в архітектурних рішеннях (ADR) як заплановану еволюцію під вимоги горизонтального масштабування.

3.4 Розгортання та супроводження програмного продукту

Розгортання платформи Real-Time Quiz Hub реалізовано засобами контейнеризації на основі Docker і Docker Compose [30; 31], що є основним способом постачання продукту. Локальний запуск через dotnet run збережено як допоміжний варіант для розробки. Контейнерний підхід забезпечує відтворюваність середовища, ізоляцію залежностей і спрощення розгортання на будь-якій машині з Docker без ручного встановлення .NET SDK і PostgreSQL.

Таблиця 3.5 – Матриця відстежуваності вимог (RTM) платформи Real-Time Quiz Hub

ID вимоги	Опис	Тип	Пов’язані тест-кейси	Метод тестування	Результат
FR-01	Автентифікація та авторизація користувача	FR	TC-01, TC-02, TC-03, TC-04	Swagger UI	Підтверджено
FR-02	CRUD відповідей (адміністратор)	FR	TC-05–TC-07 (аналог)	Swagger UI	Підтверджено
FR-03	CRUD питань (адміністратор)	FR	TC-05, TC-06, TC-07	Swagger UI	Підтверджено
FR-04	Запуск ігрової сесії	FR	TC-08; UT-01, UT-02	Swagger UI + xUnit	Підтверджено
FR-05	Перехід до наступного питання	FR	TC-09; UT-03, UT-05, UT-06	Swagger UI + xUnit	Підтверджено
FR-06	Надсилання відповіді гравцем	FR	TC-10; UT-08, UT-09	Swagger UI + xUnit	Підтверджено
FR-07	SignalR-лідерборд у реальному часі	FR	Ручне тестування браузера	Ручний (Chrome)	Підтверджено
FR-08	Завершення ігрової сесії	FR	UT-06, UT-10, UT-11	xUnit	Підтверджено
FR-09	Відображення інформації про вікторину	FR	TC-08, TC-12	Swagger UI	Підтверджено
FR-10	Нарахування балів за відповіді	FR	UT-08, UT-09, UT-15; TC-10	xUnit + Swagger UI	Підтверджено
FR-11	Вибір вікторини (каталог і лідерборди)	FR	TC-11–TC-14; UT-17, UT-21, UT-22	Swagger UI + xUnit	Підтверджено
FR-12	Механіки постійної гейміфікації (бали, XP, рівні, бейджі)	FR	TC-14, TC-15; UT-14, UT-16, UT-18–UT-21	Swagger UI + xUnit	Підтверджено
NFR-01	Надійність: ізоляція стану сесій	NFR	UT-02, UT-07, UT-11, UT-12	xUnit	Підтверджено
NFR-02	Безпека: JWT, bcrypt, захист від SQL-ін’єкцій	NFR	TC-03, TC-04	Swagger UI	Підтверджено
NFR-03	Зручність: вхід ≤ 3 кроки, адаптивність ≥ 320 px	NFR	Ручне UI-тестування	Ручний (DevTools)	Підтверджено
NFR-04	100 % україномовна локалізація	NFR	Ручний UI-огляд	Ручний (Chrome)	Підтверджено
NFR-05	Тестованість: достатнє покриття unit-тестами	NFR	UT-01–UT-24 (33 тести)	xUnit + звіт покриття	Частково
NFR-06	Масштабованість кешу: можливість переходу на Redis	NFR	Архітектурний огляд; ADR	Код-рев’ю	Частково

Стек розгортання описано у файлі docker-compose.yml, що визначає два сервіси: app і db. Сервіс db ґрунтується на офіційному образі postgres:16-alpine, має визначений healthcheck для перевірки готовності СУБД і монтований

том (volume) pgdata, який забезпечує збереження стану бази даних між перезапусками контейнера. Сервіс app – бекенд на ASP.NET Core, що збирається з первинного коду за допомогою Dockerfile і має залежність depends_on від сервісу db, чим упорядковується послідовність запуску.

Складання образу застосунку виконується багатоетапним Dockerfile (multi-stage build) з двома стадіями. Стадія build використовує образ mcr.microsoft.com/dotnet/sdk:8.0: спершу копіюється лише файл проєкту RealTimeQuizHub.csproj і виконується dotnet restore (винесено окремим кроком, щоб шар відновлених залежностей кешувався і не перебудовувався, доки не змінюється .csproj), після чого копіюється решта вихідного коду і виконується dotnet publish із параметром /p:UseAppHost=false. Стадія final ґрунтується на полегшеному образі mcr.microsoft.com/dotnet/aspnet:8.0 (лише ASP.NET Core runtime, без SDK), куди копіюється тільки результат публікації; статичні файли фронтенду (wwwroot) і Data/questions.json входять до результату публікації й автоматично потрапляють у фінальний образ. Контейнер обслуговує застосунок по HTTP на порту 8080 (EXPOSE 8080), а точкою входу є dotnet RealTimeQuizHub.dll. Такий підхід суттєво зменшує розмір кінцевого образу порівняно з однофазовою збіркою [30].

Безпека конфігурації забезпечується винесенням секретів у файл .env, який додано до .gitignore і тому не потрапляє до системи контролю версій. У .env задаються змінні POSTGRES_DB, POSTGRES_USER, POSTGRES_PASSWORD (обліковий запис і база PostgreSQL) та JWT_SECRET (ключ підпису JWT, не коротший за 32 символи). Файл docker-compose.yml зчитує ці значення через підстановку і формує похідні змінні всередині контейнерів: для сервісу db передаються POSTGRES_DB, POSTGRES_USER, POSTGRES_PASSWORD, а для сервісу app встановлюються ASPNETCORE_ENVIRONMENT=Production, ASPNETCORE_HTTP_PORTS=8080, рядок підключення ConnectionStrings__DefaultConnection (складається динамічно з облікових даних PostgreSQL) і JwtSettings__Secret (зі значення JWT_SECRET). Шаблон необхідних змінних наведено у файлі .env.example.

Решта параметрів JWT (Issuer, Audience, ExpiresInMinutes) у Docker через змінні середовища не передаються і беруться зі значень appsettings; за потреби їх можна перевизначити змінними JwtSettings__Issuer, JwtSettings__Audience, JwtSettings__ExpiresInMinutes (подвійне підкреслення – стандартний роздільник ієрархії конфігурації .NET).

Для потреб розробки й налагодження збережено можливість локального запуску без контейнерів. Він потребує встановленого .NET 8 SDK і доступного екземпляра PostgreSQL та виконується у кілька кроків: клонування репозиторію (git clone), застосування міграцій командою dotnet ef database update (схема БД формується з EF Core-міграцій, включених до репозиторію), запуск застосунку командою dotnet run (Kestrel-сервер прослуховує порт із launchSettings.json) і відкриття Swagger UI за адресою https://localhost:{port}/swagger. У цьому режимі змінні середовища не є обов’язковими: рядок підключення до БД і параметри JWT беруться безпосередньо з appsettings.json.

Журналювання у платформі Real-Time Quiz Hub реалізовано засобами вбудованого механізму ASP.NET Core – інтерфейсу ILogger<T>, що не потребує підключення зовнішніх бібліотек і доступний через вбудований контейнер ін’єкції залежностей. ILogger<T> ін’єктується у конструктори контролерів і сервісів, що забезпечує структуроване журналювання з автоматичним зазначенням джерела події (простір імен і назва класу). У режимі Development (dotnet run) записи виводяться у консоль у вигляді структурованих рядків з міткою часу, рівнем і повідомленням; у контейнерному (Production) розгортанні консольний вивід перехоплюється середовищем виконання Docker, а провайдер виведення конфігурується через appsettings.

Визначено чотири рівні журналювання відповідно до семантики Microsoft.Extensions.Logging [36]. DEBUG-записи використовуються для деталізованого налагодження (наприклад, перехід між питаннями) і активні лише в режимі Development. INFO-записи фіксують нормальні бізнес-події: старт і завершення сесій, успішну автентифікацію. WARNING-записи сигналізують про потенційні проблеми безпеки, зокрема спроби доступу без

JWT-токена. ERROR-записи фіксують виключення у шарі Services/, що свідчать про некоректні стани системи. Повний перелік типів подій наведено у табл. 3.6.

Таблиця 3.6 – Типи подій для журналювання платформи

Рівень	Подія	Де фіксується	Мета
INFO	Запуск ігрової сесії	QuizSessionService.StartQuizAsync	Трасування ігрового циклу, аудит активності
INFO	Завершення ігрової сесії	QuizSessionService.EndQuizAsync	Аудит очищення пам’яті; виявлення незавершених сесій
INFO	Успішна реєстрація / вхід	AuthController	Безпека; моніторинг активності користувачів
WARNING	401 Unauthorized (запит без токена)	JWT Middleware	Виявлення спроб несанкціонованого доступу (NFR-03)
ERROR	Виключення у методах Services/	Сервіси шару Services/ (зокрема QuizSessionService)	Діагностика збоїв; виявлення неочікуваних станів
DEBUG	Перехід між питаннями	QuizSessionService.GetNextQuestionAsync	Деталізоване налагодження у режимі Development

Наявність структурованого журналювання через ILogger<T> забезпечує мінімально необхідний рівень спостережуваності для MVP: усі критичні події безпеки та збої сервісного шару фіксуються і доступні для аналізу без додаткової інфраструктури.

Для платформи Real-Time Quiz Hub визначено чотири ключові метрики, вибір яких безпосередньо обґрунтований нефункціональними вимогами. Метрика quiz_sessions_active (кількість активних ігрових сесій у пам’яті) відображає поточне навантаження на in-memory сховище активних сесій (in-process) і є сигналом для планування переходу до Redis (NFR-07). Різке зростання цього показника без відповідного зростання завершень сесій може свідчити про витік пам’яті.

Метрика quiz_answer_latency_ms (час від отримання відповіді гравця до трансляції оновленого лідерборду через SignalR) є прямим вимірником NFR-01: якщо 95-й перцентиль цієї метрики перевищує 2 000 мс при 100 одночасних гравцях, це є однозначним сигналом для оптимізації.

Метрика api_request_duration_ms (час відповіді REST API по ендпоінтах) забезпечує моніторинг стабільності HTTP-шару і дозволяє виявляти деградацію продуктивності при зростанні навантаження.

Метрика signalr_connections_count (кількість активних WebSocket-з'єднань) відображає реальну кількість підключених гравців і є корисною для оцінки пікових навантажень та планування масштабування.

Загалом, третій розділ продемонстрував хорошу тестованість запропонованого рішення та достатній ступінь покриття тестами для здійснення висновку щодо задовільної верифікації. Для забезпечення повноцінної спостережуваності у виробничому середовищі може підключатись стек Prometheus + Grafana [37; 38]. Prometheus збирає числові метрики за моделлю pull (періодично звертається до ендпоінту /metrics застосунку), Grafana забезпечує їх візуалізацію у вигляді дашбордів.

ВИСНОВКИ

У кваліфікаційній роботі було розв’язано практичне завдання проєктування та реалізації вебплатформи Real-Time Quiz Hub для проведення онлайн-вікторин у реальному часі з механіками гейміфікації. Розроблена система орієнтована на використання в освітньому та корпоративному середовищі, де важливими є швидке залучення учасників, синхронна взаємодія, автоматизоване оцінювання, підтримка різних типів запитань, таблиця лідерів і наявність адміністративних інструментів.

У першому розділі було досліджено предметну область онлайн-вікторин і гейміфікованого оцінювання знань. Визначено, що інтерактивні вікторинні платформи поєднують педагогічні, мотиваційні та технологічні аспекти: вони забезпечують оперативний зворотний зв’язок, підтримують залученість учасників і дають змогу організовувати як навчальні, так і розважальні або корпоративні сценарії. Було проаналізовано типові сценарії використання таких платформ: синхронну вікторину в аудиторії, дистанційну синхронну вікторину, асинхронне завдання, корпоративне тестування, інтерактивну лекцію та самооцінювання. За результатами цього аналізу встановлено, що повноцінна платформа повинна підтримувати як гостьову участь, так і авторизований доступ, забезпечувати швидку взаємодію в реальному часі, накопичення результатів і масштабованість для значної кількості одночасних користувачів.

У другому розділі було виконано проєктування системи. Сформульовано функціональні та нефункціональні вимоги, визначено ролі користувачів, побудовано модель прецедентів, описано основні сценарії роботи системи та спроєктовано архітектуру програмного продукту. Було розроблено гібридну модель даних платформи. Постійні дані – користувачі, вікторини, запитання, відповіді, результати, статистика, бейджі та зв’язки між ними – зберігаються у PostgreSQL. Оперативний стан активної ігрової сесії підтримується в пам’яті процесу, що зменшує навантаження на базу даних під час одночасного надходження відповідей від багатьох учасників. У моделі передбачено сутності для реалізації не лише базового тестування, а й постійної гейміфікації:

накопичення XP, обчислення рівня, ведення статистики користувача та присвоєння бейджів.

Практичним результатом роботи стала реалізована MVP-версія вебплатформи Real-Time Quiz Hub. Серверну частину реалізовано на ASP.NET Core із використанням REST API, SignalR, Entity Framework Core, JWT-авторизації та PostgreSQL. Клієнтську частину створено на основі HTML, CSS і JavaScript без застосування складного фронтенд-фреймворку, що спрощує розгортання і підтримку системи. Реалізовано ключові екрани: автентифікацію та гостьовий вхід, ігровий екран, лідерборд, екран результату та адміністративну панель для керування контентом.

У третьому розділі було проведено тестування, перевірку відповідності вимогам і розгортання програмного продукту. Для перевірки системи застосовано багаторівневу стратегію тестування: модульні тести бізнес-логіки за допомогою xUnit, функціональне тестування REST API через Swagger UI, ручну перевірку real-time-взаємодії в браузері та аналіз відповідності вимогам за допомогою RTM-матриці. Результати тестування підтвердили працездатність основних функціональних можливостей системи: автентифікації та авторизації, CRUD-операцій для запитань і відповідей, запуску і завершення ігрової сесії, переходу до наступного запитання, надсилання відповіді, нарахування балів, роботи таблиці лідерів, вибору вікторини та механік постійної гейміфікації. Було підтверджено відповідність системи основним функціональним і нефункціональним вимогам, зокрема щодо ізоляції стану сесій, супроводжуваності, зручності використання та готовності до подальшого масштабування.

Розгортання платформи реалізовано із застосуванням Docker і Docker Compose, що забезпечує відтворюваність середовища, ізоляцію залежностей і спрощення запуску системи на іншій машині. Контейнеризація дає змогу уникнути ручного встановлення окремих компонентів середовища, зокрема .NET SDK і PostgreSQL, та створює основу для подальшого впровадження CI/CD-процесів.

Отже, поставлену мету кваліфікаційної роботи досягнуто: спроєктовано та реалізовано вебплатформу для проведення онлайн-вікторин у реальному часі з механіками гейміфікації, адміністративними функціями та підтримкою інтерактивної взаємодії учасників. Розроблена система демонструє практичну придатність для освітнього й корпоративного використання, має зрозумілу архітектуру, підтримує подальше розширення функціональності та може бути адаптована до потреб конкретної установи.

Подальший розвиток Real-Time Quiz Hub доцільно спрямувати на реалізацію повноцінного асинхронного режиму проходження вікторин із дедлайнами, розширення аналітики для викладача або адміністратора, удосконалення адміністративної панелі, підтримку імпорту й експорту запитань, впровадження Redis для розподіленого зберігання стану активних сесій, додавання централізованого моніторингу та журналювання, а також розгортання CI/CD-процесу для автоматизованої перевірки й постачання нових версій програмного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Li M., Ma S., Shi Y. Examining the effectiveness of gamification as a tool promoting teaching and learning in educational settings: a meta-analysis. *Frontiers in Psychology*. 2023. Vol. 14. Article 1253549. DOI: 10.3389/fpsyg.2023.1253549.
2. Kahoot! Research. Independent research on Kahoot! in education. URL: https://kahoot.com/research/ (дата звернення: 2.06.2026).
3. Kudubes A. A. The effect of using Kahoot in pediatric emergency nursing lessons on students' success and motivation levels: A randomized controlled study. Nurse Education in Practice. 2025. Article 104265. DOI: 10.1016/j.nepr.2025.104265.
4. Маренич Ф. А., Марченко С. В. Архітектура вебплатформи проведення онлайн-вікторин у режимі реального часу. Матеріали XVIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених за тематикою «Тенденції розвитку ІТ-технологій в Україні», м. Черкаси, 23–24 квітня 2026 р. Черкаси : Черкаський державний фаховий бізнес-коледж, 2026. С. 177–180.
5. ДСТУ 2226-93. Автоматизовані системи. Терміни та визначення. Київ : Держстандарт України, 1993. 89 с.
6. Maraza-Quispe B., Traverso-Condori L. C., Torres-Gonzales S. B., Reyes-Arco R. E., Tinco-Túpac S. T., Reyes-Villalba E., Carpio-Ventura J. R. Impact of the Use of Gamified Online Tools: A Study with Kahoot and Quizizz in the Educational Context. *International Journal of Information and Education Technology.* 2024. Vol. 14, No. 1. P. 132–140. DOI: 10.18178/ijiet.2024.14.1.2033.
7. Martin F., Bolliger D. U. Engagement matters: Student perceptions on the importance of engagement strategies in the online learning environment. Online Learning. 2018. Vol. 22, No. 1. P. 205–222. DOI: 10.24059/olj.v22i1.1092.

8. Johnson K. L., Nazer D. When it’s all fun and games: gamification of child abuse medical education. *International Journal of Pediatrics and Adolescent Medicine*. 2024. Vol. 11, No. 1. P. 13–17. DOI: 10.4103/ijpam.ijpam_67_24.
9. Research School Network. Using digital quiz platforms in the classroom. URL: https://www.researchschool.org.uk/ (дата звернення: 2.06.2026).
10. Căpățînă A., Juarez-Varon D., Micu A., Micu A. E. Leveling up in corporate training: Unveiling the power of gamification to enhance knowledge retention, knowledge sharing, and job performance. *Journal of Innovation & Knowledge*. 2024. Vol. 9, No. 3. Article 100530. DOI: 10.1016/j.jik.2024.100530.
11. Black P., Wiliam D. Assessment and classroom learning. *Assessment in Education: Principles, Policy & Practice*. 1998. Vol. 5, No. 1. P. 7–74. DOI: 10.1080/0969595980050102.
12. Moodle Documentation. Question types. URL: https://docs.moodle.org/en/Question_types (дата звернення: 12.06.2026).
13. Bloom B. S., Engelhart M. D., Furst E. J., Hill W. H., Krathwohl D. R. Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain. New York : Longmans, Green, 1956. 207 p.
14. MathJax Documentation. Writing Mathematics for MathJax. URL: https://docs.mathjax.org/en/latest/basic/mathematics.html (дата звернення: 2.06.2026).
15. Kahoot! Pricing. URL: https://kahoot.com/pricing/ (дата звернення: 2.06.2026).
16. Quizizz Pricing. URL: https://quizizz.com/pricing/ (дата звернення: 2.06.2026).
17. AhaSlides Pricing. URL: https://ahaslides.com/pricing/ (дата звернення: 2.06.2026).
18. Sailer M., Homner L. The gamification of learning: a meta-analysis. Educational Psychology Review. 2020. Vol. 32. P. 77–112. DOI: 10.1007/s10648-019-09498-w.

19. Ruiz J. J. R., Sanchez A. D. V., Figueredo O. R. B. Impact of gamification on school engagement: a systematic review. *Frontiers in Education*. 2024. Vol. 9. Article 1466926. DOI: 10.3389/feduc.2024.1466926.
20. Kivetz R., Urminsky O., Zheng Y. The goal-gradient hypothesis resurrected: Purchase acceleration, illusionary goal progress, and customer retention. *Journal of Marketing Research*. 2006. Vol. 43, No. 1. P. 39–58. DOI: 10.1509/jmkr.43.1.39.
21. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16/ (дата звернення: 2.06.2026).
22. Redis Ltd. Redis Documentation. URL: https://redis.io/docs/latest/ (дата звернення: 12.06.2026).
23. Microsoft. Cache in-memory in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/performance/caching/memory (дата звернення: 2.06.2026).
24. Fette I., Melnikov A. The WebSocket Protocol : RFC 6455. Internet Engineering Task Force (IETF), 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 2.06.2026).
25. Microsoft. ASP.NET Core SignalR documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction (дата звернення: 2.06.2026).
26. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 p.
27. Microsoft. Configure JWT bearer authentication in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/configure-jwt-bearer-authentication (дата звернення: 2.06.2026).
28. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley, 2021. 464 p.
29. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.
30. Docker Documentation. Docker Inc. URL: https://docs.docker.com/ (дата звернення: 2.06.2026).

31. Docker Compose Documentation. Docker Inc. URL: https://docs.docker.com/compose/ (дата звернення: 2.06.2026).
32. Microsoft. Entity Framework Core documentation. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 2.06.2026).
33. Meszaros G. xUnit Test Patterns: Refactoring Test Code. Boston : Addison-Wesley, 2007. 883 p.
34. xUnit.net : documentation. URL: https://xunit.net/ (дата звернення: 2.06.2026).
35. Swashbuckle.AspNetCore : документація проєкту. GitHub. URL: https://github.com/domaindrivendev/Swashbuckle.AspNetCore (дата звернення: 2.06.2026).
36. Microsoft. Logging in .NET. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/logging (дата звернення: 2.06.2026).
37. Prometheus Documentation. URL: https://prometheus.io/docs/ (дата звернення: 12.06.2026).
38. Grafana Documentation. Grafana Labs. URL: https://grafana.com/docs/grafana/latest/ (дата звернення: 2.06.2026).

ДОДАТКИ

Додаток А – Репозиторій з код програмного продукту

Код програмної реалізації розробленого в межах кваліфікаційної роботи продукту розташований за адресою https://github.com/Ururu221/Real-Time-Quiz-Hub.