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

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ТРЕНУВАННЯ ШВИДКОСТІ РЕАКЦІЇ ТА ТОЧНОСТІ КОРИСТУВАЧА

Виконав: студент групи 2П-22
Спеціальності
121 Інженерія програмного забезпечення
Максим ПУСТОВІТ
Керівник:
Валентин ДМИТРЮК

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
Пустовіту Максиму Валентиновичу

1. Тема кваліфікаційної роботи Розробка вебзастосунку для тренування швидкості реакції та точності користувача.

Керівник роботи Дмитрюк Валентин Віталійович

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

Технології React 19, Three.js 0.182, Vite 7 (клієнтська частина); Node.js, Express.js, MongoDB Atlas, Mongoose (серверна частина); JWT та bcrypt (авторизація); WebSocket, бібліотека ws (оновлення у реальному часі). Інструменти розробки: Git, Visual Studio Code. Оформлення згідно з чинними стандартами ДСТУ.

4. Зміст кваліфікаційної роботи: аналіз предметної області та технологій (навички прицілювання та методи їх тренування; аналіз існуючих рішень Aim Lab, KovaaK's, 3D Aim Trainer; огляд технологій реалізації та архітектурних патернів; безпека вебзастосунків); проектування системи (функціональні та нефункціональні вимоги; архітектура системи; проектування бази даних, REST API, WebSocket-протоколу, структури проекту та інтерфейсу користувача); практична реалізація (реалізація бекенду, фронтенду, 3D-сцени та режимів гри); тестування та аналіз результатів (функціональне тестування, тестування продуктивності, безпеки та сумісності браузерів).

5. Дата видачі завдання: 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапу	Термін	Примітка
1	Вступ	14.10.2025	
2	Розділ 1. Аналіз предметної області та технологій	09.12.2025	
3	Розділ 2. Проектування системи	10.02.2026	
4	Розділ 3. Практична реалізація	10.03.2026	
5	Розділ 4. Тестування та аналіз результатів	28.04.2026	
6	Висновки	12.05.2026	
7	Оформлення кваліфікаційної роботи (чистовий варіант)	24.05.2026	
8	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2026	
9	Подання кваліфікаційної роботи на затвердження завідувачці циклової комісії (за 7 днів до захисту)	10.06.2026	

Студент _____ Максим ПУСТОВІТ
(підпис)

Керівник роботи _____ Валентин ДМИТРЮК
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці вебзастосунку AimPractice для тренування швидкості реакції та точності користувача. Метою роботи є проектування та реалізація браузерного 3D-тренажера з хмарною базою даних, системою авторизації та оновленнями у реальному часі.

У роботі здійснено аналіз предметної області та існуючих рішень (Aim Lab, KovaaK's, 3D Aim Trainer), на основі якого обрано технологічний стек: React 19, Three.js 0.182, Vite 7 для клієнтської частини; Node.js, Express.js, MongoDB Atlas, Mongoose для серверної частини; JWT та bcrypt для авторизації; WebSocket (бібліотека ws) для оновлень у реальному часі.

Реалізовано чотири режими тренування (Polygon, Grid, Flick, Tracking) та три рівні складності. Система авторизації на основі JWT дозволяє зберігати результати у хмарній базі MongoDB Atlas та переглядати глобальний лідерборд, що оновлюється в реальному часі. Публічні профілі гравців надають розширену статистику та графіки прогресу.

Проведено комплексне тестування: 14 функціональних тест-кейсів, вимірювання продуктивності (60–144 FPS, час відповіді API до 235 мс), тестування безпеки та сумісності з браузерами Chrome, Firefox, Edge. Усі тест-кейси успішно пройдені.

Практична цінність роботи полягає у готовому вебзастосунку, що не вимагає встановлення та доступний з будь-якого пристрою з сучасним браузером.

Ключові слова: ВЕБЗАСТОСУНОК, 3D-ГРАФІКА, THREE.JS, REACT, MONGODB ATLAS, JWT, WEBSOCKET, ТРЕНАЖЕР ПРИЦІЛЮВАННЯ, FPS, КІБЕРСПОРТ.

ABSTRACT

This qualification thesis is devoted to the development of a web application AimPractice for training user reaction speed and accuracy. The goal of the work is to design and implement a browser-based 3D trainer with a cloud database, authentication system, and real-time updates.

The thesis includes an analysis of the subject area and existing solutions (Aim Lab, KovaaK's, 3D Aim Trainer), based on which the following technology stack was selected: React 19, Three.js 0.182, Vite 7 for the client side; Node.js, Express.js, MongoDB Atlas, Mongoose for the server side; JWT and bcrypt for authentication; WebSocket (ws library) for real-time updates.

Four training modes (Polygon, Grid, Flick, Tracking) and three difficulty levels were implemented. The JWT-based authentication system allows saving results to MongoDB Atlas and viewing the global leaderboard updated in real time. Public player profiles provide detailed statistics and progress charts.

Comprehensive testing was conducted: 14 functional test cases, performance measurements (60–144 FPS, API response time up to 235 ms), security testing and browser compatibility testing with Chrome, Firefox, and Edge. All test cases passed successfully.

The practical value of the work lies in a ready-to-use web application that requires no installation and is accessible from any device with a modern browser.

Keywords: WEB APPLICATION, 3D GRAPHICS, THREE.JS, REACT, MONGODB ATLAS, JWT, WEBSOCKET, AIM TRAINER, FPS, ESPORTS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface (прикладний програмний інтерфейс)

BSON – Binary JSON (бінарний формат JSON для MongoDB)

CORS – Cross-Origin Resource Sharing (спільне використання ресурсів між різними джерелами)

CPU – Central Processing Unit (центральний процесор)

DOM – Document Object Model (об’єктна модель документа)

FPS – First-Person Shooter (шутер від першої особи) / Frames Per Second (кадрів за секунду)

GPU – Graphics Processing Unit (графічний процесор)

HMR – Hot Module Replacement (гаряча заміна модулів)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

HUD – Heads-Up Display (дисплей у полі зору гравця)

JWT – JSON Web Token (веб-токен JSON, стандарт RFC 7519)

NoSQL – Not Only SQL (не тільки SQL, клас нереляційних СУБД)

ODM – Object Document Mapper (об’єктно-документний маппер)

REST – Representational State Transfer (архітектурний стиль для розподілених систем)

SPA – Single Page Application (односторінковий застосунок)

СУБД – Система управління базами даних

TCP – Transmission Control Protocol (протокол керування передачею)

URI – Uniform Resource Identifier (уніфікований ідентифікатор ресурсу)

WebGL – Web Graphics Library (JavaScript API для 3D-графіки у браузері)

WS – WebSocket (протокол двостороннього з’єднання поверх TCP, RFC 6455)

ВСТУП

Розвиток кіберспорту перетворив комп’ютерні ігри на повноцінну спортивну дисципліну з розвиненою інфраструктурою. Ігри жанру FPS (First-Person Shooter), зокрема Counter-Strike 2, Valorant, Apex Legends та Overwatch 2, займають провідні позиції в кіберспортивному ландшафті та збирають мільйони глядачів на онлайн-трансляціях. За даними аналітичної компанії Newzoo [16], у 2024 році глобальна аудиторія кіберспорту перевищила 640 мільйонів осіб, а сукупний обсяг ринку склав понад 1,8 мільярда доларів США.

Ключовою навичкою у шутерах від першої особи є точність прицілювання – здатність швидко та точно навести приціл на ціль і зробити влучний постріл. Ця навичка формується через тривале тренування та вдосконалення моторної пам’яті. На відміну від тактичного мислення чи командної взаємодії, прицілювання піддається ізольованому тренуванню за допомогою спеціалізованих інструментів – aim trainers.

Наявні рішення для тренування прицілювання здебільшого є десктопними застосунками (Aim Lab, KovaaK’s), що вимагають встановлення та значного дискового простору. Актуальні вебрішення, на зразок 3D Aim Trainer, мають обмежену функціональність: відсутність авторизації, глобального лідерборду та детальної статистики прогресу. Нестача доступного повнофункціонального вебрішення з системою авторизації, хмарним сховищем даних та оновленнями лідерборду в реальному часі визначає актуальність цієї роботи.

Сучасні вебтехнології досягли рівня, що дозволяє реалізовувати складні 3D-застосунки безпосередньо у браузері. Технологія WebGL надає доступ до GPU для рендерингу 3D-графіки, бібліотека Three.js спрощує роботу з WebGL, React забезпечує ефективне управління складним інтерфейсом, а MongoDB Atlas усуває необхідність адміністрування власної інфраструктури зберігання даних.

Об’єктом дослідження є процес тренування швидкості реакції та точності користувача за допомогою інтерактивних вебзастосунків. Окремий інтерес викликають механізми забезпечення ефективного зворотного зв’язку, методи

оцінки прогресу гравця та технічні засоби реалізації 3D-середовища безпосередньо у браузері.

Предметом дослідження виступають методи та засоби розробки браузерних 3D-тренажерів з підтримкою реального часу та хмарного зберігання даних.

Мета кваліфікаційної роботи – розробити повнофункціональний вебзастосунок для тренування швидкості реакції та точності користувача з підтримкою різних режимів гри, системою авторизації, хмарною базою даних та глобальним лідербордом із оновленнями в реальному часі.

Завдання кваліфікаційної роботи:

1) провести аналіз предметної області, виявити недоліки існуючих рішень та обґрунтувати необхідність розробки нового застосунку;

2) порівняти технологічні рішення для реалізації компонентів програмного продукту;

3) спроєктувати архітектуру програмної системи, схему бази даних та REST API;

4) реалізувати 3D-сцену тренажера з різними режимами гри, системою реєстрації та авторизації, оновлення лідерборду в реальному часі;

5) провести комплексне тестування застосунку та оцінити отримані результати.

Практична цінність роботи полягає у створенні готового до використання вебзастосунку, доступного з будь-якого пристрою з сучасним браузером без додаткового встановлення програмного забезпечення. Застосунок має забезпечувати повний цикл тренування: від вибору режиму та складності до перегляду статистики та порівняння результатів з іншими гравцями у глобальному лідерборді.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та п’яти додатків. Розділ 1 містить аналіз предметної області та огляд технологій. Розділ 2 присвячений проектуванню архітектури системи. Розділ 3 описує практичну реалізацію. Розділ 4 містить

результати тестування. Загальний обсяг роботи складає більше 60 сторінок основного тексту, містить 7 рисунків та 8 таблиць.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ

1.1 Аналіз предметної області

1.1.1 Навички прицілювання та методи їх тренування

Навички прицілювання у шутерах від першої особи можна класифікувати за кількома критеріями. За характером руху прицілу виділяють: флік-шот (flick shot) – різкий рух прицілом до цілі з наступним пострілом; відстеження (tracking) – утримання прицілу на рухомій цілі протягом часу; мікроадаптацію (micro-adjustment) – незначні корекції поблизу цілі. За характером цілі: статичні цілі розвивають базову точність, рухомі – координацію та передбачення траєкторії.

Тренування прицілювання є формою психомоторного навчання – процесу формування автоматизованих рухових навичок через повторення. Моторна пам’ять (procedural memory) зберігається у базальних гангліях та мозочку і не потребує свідомого контролю після досягнення автоматизму. Це пояснює, чому досвідчений гравець влучає у ціль без усвідомленого обчислення кута та відстані.

Дослідження Карла Андерса Еріксона [18] про навмисну практику (deliberate practice) показують: ефективне навчання вимагає цілеспрямованих вправ на межі компетентності, а не просто великої кількості повторень. Aim trainers дозволяють реалізувати цей принцип: гравець може щоразу обирати рівень складності, що відповідає поточному рівню майстерності, поступово підвищуючи його.

Реакція (reaction time) – час від появи стимулу до початку відповідного руху – є ще однією ключовою характеристикою. Середній час простої реакції у людини становить 200–250 мс. Тренування не зменшує фізіологічний ліміт реакції, але формує «антиципацію» – здатність передбачати появу цілі, що дозволяє підготувати рух заздалегідь і скоротити фактичний час відповіді до 100–150 мс.

Дослідження в галузі моторного навчання (Fitts, 1954; Schmidt & Lee, 2011) [17, 19] показують, що систематичне ізольоване тренування навичок є більш ефективним, ніж нерегулярна практика в умовах реальної гри. Aim trainers дозволяють відпрацьовувати конкретний тип руху у контрольованих умовах без відволікаючих факторів повноцінного геймплею.

Важливу роль у формуванні навичок прицілювання відіграє зворотний зв'язок (feedback). Миттєвий візуальний та звуковий відгук на влучення або промах прискорює формування нейронних шляхів, відповідальних за моторну пам'ять. Дослідження показують, що затримка зворотного зв'язку більше 100 мс суттєво сповільнює процес навчання. Для ефективного тренування критично важливими є: висока частота кадрів (60 FPS і вище), миттєва реєстрація влучань та чіткий аудіовізуальний відгук.

Зворотний зв'язок (feedback) є необхідною умовою ефективного моторного навчання. Розрізняють два типи: внутрішній (власні відчуття від руху) та зовнішній (інформація з середовища – звук влучання, анімація вибуху). У комп'ютерних іграх зовнішній зворотний зв'язок є домінуючим.

Дослідження (Schmidt, Lee, 2011) [17] показують, що затримка зворотного зв'язку більше 100 мс значно сповільнює формування навички. Застосунок AimPractice забезпечує миттєву реєстрацію влучань через Raycaster на тому ж кадрі, що й клік – затримка між фізичним кліком та аудіовізуальним відгуком не перевищує 1 кадру (16.7 мс при 60 FPS). Це відповідає найкращим практикам у розробці інтерактивних навчальних систем.

Перенесення навичок (skill transfer) – ступінь, до якого навичка, відпрацьована в одному контексті, покращує виконання в іншому. Дослідження показують, що навички прицілювання, відпрацьовані у aim trainers, переносяться на реальні шутери від першої особи при умові схожості параметрів (FOV, чутливість миші, швидкість цілей).

Сенситивність миші є одним із ключових параметрів налаштування для гравців у шутери від першої особи [17]. Вона визначає, наскільки великий кут повороту камери відповідає фізичному переміщенню миші. Занадто висока

чутливість призводить до неточних рухів, занадто низька – до обмеженості переміщень. Aim trainers дозволяють налаштувати цей параметр і відпрацювати конкретну сенситивність у контрольованих умовах. Розроблювана система дозволятиме налаштувати чутливість миші для відповідності конкретній грі, що максимізує перенесення навички.

Аналіз режимів тренування конкурентних рішень показує наступне. Платформа Aim Lab пропонує такі категорії сценаріїв: Flicking (швидке прицілювання), Tracking (відстеження рухомих цілей), Micro-corrections (мікрокорекції), Speed (швидкість реакції), Perception (сприйняття), Cognition (когнітивні навички). Тренажер KovaaK's надає більше 4500 сценаріїв, згрупованих за типом руху та швидкістю цілей; найпопулярніші включають Smoothbot (smooth tracking), Voltaic (flick + micro), Air Angelic (стрибаючі цілі). Вебрішення 3D Aim Trainer має базові режими: Target Practice, Grid Shot, Tracking, Reflex Shot.

Чотири режими тренування AimPractice охоплюють основні типи ситуацій у реальних іграх: Polygon (загальна точність), Grid (командна зачистка), Flick (швидкі постріли по відкритому ворогу), Tracking (стрільба по рухомих цілях). Це дозволяє систематично тренувати саме ті аспекти, що є слабкими у конкретного гравця.

1.2 Аналіз існуючих рішень

На ринку існує кілька категорій рішень для тренування прицілювання. Аналіз проводився за критеріями: платформа доступності, наявність авторизації та хмарного зберігання, глобальний лідерборд із оновленнями в реальному часі, тип 3D-рушія, глибина статистики та наявність публічних профілів гравців. Розглянуто чотири найпопулярніші рішення.

Платформа Aim Lab (aimlabs.com) є безкоштовним тренажером з аудиторією понад 40 млн зареєстрованих гравців [21], доступний через Steam та браузер (WebGL). Понад 2000 сценаріїв: Flicking, Tracking, Micro-corrections, Speed, Perception, Cognition. Система штучного інтелекту аналізує показники

гравця (точність, час реакції, шаблони промахів) та генерує персоналізовані рекомендації. Доступні теплові карти влучань, налаштування параметрів під конкретні ігри (CS2, Valorant, Apex Legends). Недоліки: десктопна версія займає ~4 ГБ; вебверсія має скорочений функціонал; лідерборд оновлюється лише при перезавантаженні сторінки (WebSocket відсутній).

Додаток KovaaK's FPS Aim Trainer (kovaaks.com) є платним десктопним тренажером, що являється стандартом серед кіберспортсменів [22]. Бібліотека налічує понад 4500 сценаріїв; найпопулярніші: Voltaic Benchmarks, Smoothbot, Air Angelic. Вебсайт kovaaks.com пропонує: Sens Converter (конвертер чутливості між іграми), Crosshair Creator, Benchmark Tracker. Інтерфейс: темна тема з помаранчевими акцентами, розділ Featured Highscores (рекорди відомих гравців). Недоліки: ціна ~$9.99; складний інтерфейс; десктопна версія ~5 ГБ; відсутній WebSocket-лідерборд у реальному часі.

Вебплатформа 3D Aim Trainer (3daimtrainer.com) – вебрішення від SteelSeries з понад 200 вправ та 12 млн зареєстрованих гравців. Доступний безпосередньо у браузері без встановлення. Реалізований на WebGL/Three.js. Навігація: How it works, Training guides, Gaming gear, Sensitivity Tools. Режими: Target Practice, Grid Shot, Tracking, Reflex Shot. Інтегрований конвертер сенситивності. Недоліки: відсутня JWT-авторизація з хмарним зберіганням – результати зберігаються локально та не синхронізуються між пристроями; відсутні WebSocket-оновлення та публічні профілі гравців.

На основі аналізу виявлено, що жоден з розглянутих конкурентів не поєднує одночасно: доступність через браузер без встановлення, повноцінну авторизацію з хмарним зберіганням, глобальний лідерборд із WebSocket-оновленнями та публічні профілі гравців. Порівняльний аналіз наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Параметр	Aim Lab	KovaaK’s	3D Aim Trainer	AimPractice
Платформа	Desktop (Steam)	Desktop (Steam)	Web	Web
Вартість	Безкоштовно	~$10	Безкоштовно	Безкоштовно
Встановлення	Потрібне (~4 ГБ)	Потрібне (~5 ГБ)	Не потрібне	Не потрібне
Авторизація	Так (Steam)	Так (Steam)	Ні	Так (JWT)
Глобальний лідерборд	Так	Так	Ні	Так
Real-time оновлення	Ні	Ні	Ні	Так (WebSocket)
3D-рушій	Unity	Unreal Engine	Three.js	Three.js
Статистика	Розширена + AI	Розширена	Базова	Розширена
Публічні профілі	Ні	Ні	Ні	Так
Хмарна база даних	Так	Так	Ні	Так (Atlas)

Отже, аналіз існуючих рішень виявив, що жоден з розглянутих конкурентів не поєднує одночасно доступність через браузер без встановлення, повноцінну авторизацію з хмарним зберіганням, глобальний лідерборд із WebSocket-оновленнями та публічні профілі гравців. Розроблюваний застосунок AimPractice усуває ці недоліки та пропонує конкурентоспроможний набір функцій у браузерному форматі.

1.3 Аналіз технологій програмної реалізації

Попередній огляд функціональних можливостей альтернативних програмних рішень показав, що подібні додатки не вимагають насиченого інтерфейсу та розвиненої навігації. Тому односторінковий застосунок (Single Page Application, SPA), що завантажується у браузер один раз і динамічно оновлює вміст сторінки без її повного перезавантаження, є очевидним вибором. На відміну від традиційної MPA (Multi-Page Application), де кожен перехід між сторінками ініціює новий HTTP-запит і повне перезавантаження HTML-документа, SPA використовує клієнтську маршрутизацію та асинхронні запити до API для завантаження лише необхідних даних.

Технічну основу SPA складають три механізми браузера. History API (pushState/popState) дозволяє змінювати URL-адресу без перезавантаження

сторінки, що забезпечує навігацію між «екранами» застосунку. Технології XMLHttpRequest та Fetch API забезпечують асинхронний обмін даними з сервером у фоновому режимі. Механізм клієнтської маршрутизації бібліотеки (React Router) зіставляє поточний URL з відповідним компонентом React і відображає його без участі сервера [10].

Порівняно з MPA, SPA мають суттєві переваги у контексті інтерактивних застосунків: відсутність затримки при навігації (немає мережевого запиту за новою сторінкою); збереження стану застосунку при переходах (наприклад, результати гри не втрачаються при відкритті лідерборду); можливість реалізації плавних анімацій переходів між екранами; оптимізований обмін даними – завантажуються лише дані, а не весь HTML-документ. Порівняння підходів MPA та SPA наведено на рисунку 1.1.

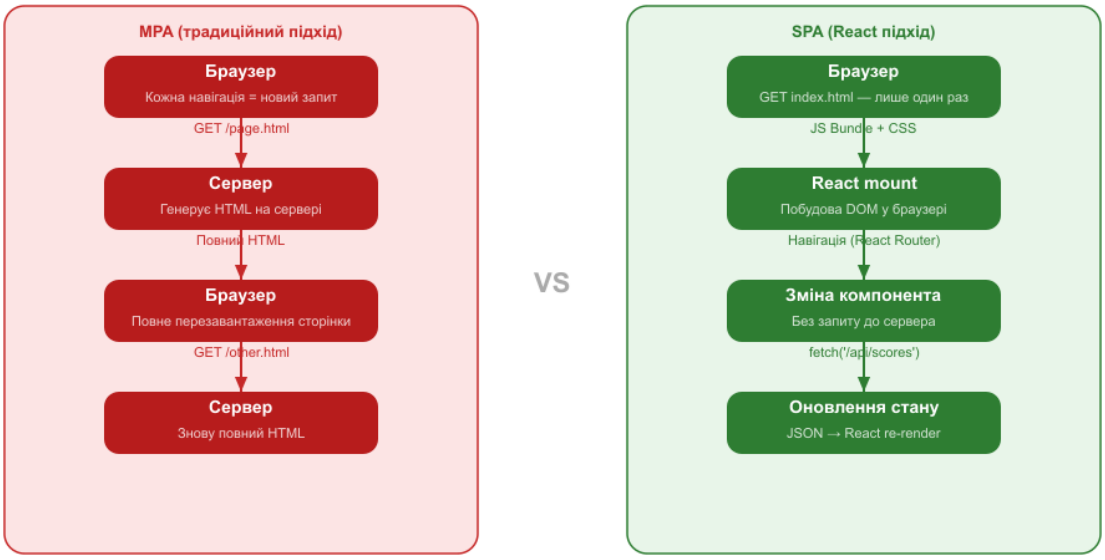

Рисунок 1.1 – Порівняння MPA та SPA підходів

У SPA весь рендеринг відбувається на стороні клієнта (Client-Side Rendering, CSR): браузер завантажує порожній HTML-файл з тегом , JavaScript-бандл та CSS. Бібліотека React самостійно будує DOM-дерево у браузері на основі стану застосунку. Перевага CSR – висока інтерактивність після початкового завантаження, недолік – довший час першого відображення (FCP) через необхідність завантажити та виконати JavaScript.

Альтернативою є серверний рендеринг (Server-Side Rendering, SSR), де сервер генерує повний HTML при кожному запиті. Це покращує SEO та FCP, але вимагає складнішої інфраструктури. Для AimPractice обрано CSR, оскільки застосунок не потребує SEO-оптимізації (це не публічний контентний сайт), а акцент робиться на максимальній інтерактивності та мінімальній затримці під час гри. Інструмент Vite забезпечує оптимальне збирання бандлу з code splitting для CSR.

Управління станом є ключовою задачею у SPA. Стан застосунку – це сукупність даних, що визначають поточний вигляд інтерфейсу: авторизований користувач, поточний екран, результати гри, налаштування. У AimPractice стан управляється через вбудовані механізми React без зовнішніх бібліотек (Redux, Zustand), оскільки складність застосунку цього не потребує.

Стратегія управління станом: глобальний стан (авторизація, налаштування) зберігається в кореневому компоненті App.jsx та передається вниз через пропси; локальний стан (поточний фільтр лідерборду, помилка форми) зберігається в тому компоненті, якому він належить. Персистентність між сесіями забезпечується localStorage для токена та налаштувань – при перезавантаженні сторінки стан відновлюється автоматично.

1.3.1 Огляд технологій реалізації фронтенду

Бібліотека React являє собою JavaScript-інструмент для побудови користувацьких інтерфейсів, розроблений та підтримуваний компанією Meta (Facebook) [1, 12]. Концептуальною основою React є компонентний підхід: увесь інтерфейс розбивається на незалежні компоненти, кожен із яких управляє власним станом. Компоненти можуть бути вкладеними та багаторазово використовуваними, що суттєво скорочує обсяг коду та спрощує підтримку.

Механізм Virtual DOM дозволяє React мінімізувати кількість операцій з реальним DOM. При зміні стану React спочатку оновлює легковагове дерево об’єктів JavaScript (Virtual DOM), потім порівнює його з попереднім станом (diffing) та вносить лише необхідні зміни до реального DOM. Версія

React 19, що використовується у проекті, вводить покращений механізм конкурентного рендерингу (Concurrent Mode), що дозволяє переривати довготривалі операції рендерингу для обробки більш пріоритетних оновлень.

У проєкті AimPractice React використовується для реалізації всіх екранів та компонентів інтерфейсу. Хуки React (useState, useEffect, useCallback, useMemo, useRef) широко застосовуються для управління станом та побічними ефектами. Зокрема, useRef є ключовим для роботи з Three.js, оскільки надає доступ до DOM-елементу canvas та зберігає стан рушія між рендерами без їх тригерингу.

Бібліотека Three.js – це JavaScript-бібліотека для роботи з 3D-графікою у браузері, що надає зручний об’єктно-орієнтований інтерфейс поверх низькорівневого WebGL API [2]. Технологія WebGL являє собою JavaScript API для рендерингу інтерактивної 2D та 3D графіки у браузері без плагінів, заснований на OpenGL ES 2.0 [11].

Бібліотека Three.js абстрагує складність WebGL, надаючи готові класи для роботи з об’єктами сцени (Scene), камерою (Camera), рендерером (WebGLRenderer), геометріями (BoxGeometry, SphereGeometry, PlaneGeometry), матеріалами (MeshStandardMaterial, MeshBasicMaterial), освітленням (PointLight, AmbientLight). Це скорочує обсяг коду у порівнянні з безпосереднім використанням WebGL.

Ключові можливості Three.js у проекті: WebGLRenderer з підтримкою тіней та антиаліасингу; PointerLockControls для управління камерою у стилі шутера від першої особи; Raycaster для виявлення влучань; EffectComposer з UnrealBloomPass для bloom-ефекту. Версія Three.js 0.182 обрана як стабільна на момент розробки.

1.3.2 Огляд технологій реалізації бекенду

Середовище Node.js – це платформа виконання JavaScript поза браузером, побудована на рушії V8 від Google. Головною особливістю є подійно-орієнтована, однопотокова, неблокуюча модель введення-виведення. При отриманні запиту до бази даних Node.js не блокує виконання і передає

управління наступному обробнику. Коли операція завершується, відповідна callback-функція або Promise ставиться в чергу. Це дозволяє ефективно обробляти тисячі одночасних з'єднань.

Вебфреймворк Express.js – це мінімалістичний інструмент для Node.js, побудований навколо концепції middleware [4]. Middleware – функції, що мають доступ до об'єктів запиту (req), відповіді (res) та наступної функції в ланцюгу (next). У проєкті Express використовується з: express.json() для парсингу тіла запиту, cors() для обробки CORS-заголовків, requireAuth (власний middleware) для перевірки JWT. Маршрути організовані у окремі модулі: auth.js, scores.js, sessions.js, profile.js.

Таблиця 1.2 – Порівняльний аналіз фронтенд-фреймворків

Критерій	React 19	Vue.js 3	Angular 17
Поріг входу	Середній	Низький	Високий
Розмір спільноти	Найбільша (~50% ринку)	Велика (~18%)	Велика (~17%)
Продуктивність	Висока (Concurrent Mode)	Висока (Vapor)	Середня
Гнучкість	Висока (тільки View)	Висока	Обмежена (MVC)
TypeScript	Опціонально	Опціонально	Обов'язково
3D-інтеграція	Легка (useRef+canvas)	Середня	Складна

З наведеного порівняння видно, що React 19 є оптимальним вибором для проекту. Найбільша екосистема гарантує наявність необхідних бібліотек (Three.js інтеграція, routing), механізм useRef забезпечує зручну роботу з canvas для 3D-рендерингу, а конкурентний режим рендерингу покращує відзивність інтерфейсу під час ресурсоємних операцій Three.js. Vue.js та Angular поступаються саме у зручності інтеграції з 3D-рендерером.

1.3.3 Огляд технологій зберігання даних

Документно-орієнтована СУБД MongoDB зберігає дані у вигляді BSON-документів [3] та належить до класу NoSQL-рішень. Документи організовані у колекції без жорсткої схеми: кожен документ може мати власну структуру. Переваги для проєкту: гнучка схема дозволяє зберігати різноманітні ігрові

записи без міграцій; нативна підтримка JSON спрощує обмін даними; потужна система агрегації для статистичних запитів [13].

Хмарна платформа MongoDB Atlas надає керовані кластери MongoDB з автоматичним резервним копіюванням та глобальним розподілом. Free Tier (кластер M0) надає 512 МБ сховища безкоштовно, що достатньо для академічного проекту та зберігання сотень тисяч ігрових записів.

Бібліотека Mongoose є ODM для MongoDB у Node.js, яка додає шар типізації та валідації [9]. Ключові можливості: схеми з типізацією, автоматична валідація перед збереженням, хуки pre/post, зручний API для CRUD-операцій та агрегаційний pipeline. У проекті визначено три схеми: User, Score та Session.

Таблиця 1.3 – Порівняльний аналіз рішень для зберігання даних

Критерій	MongoDB Atlas	PostgreSQL (Supabase)	Firebase Firestore
Тип БД	Document NoSQL	Relational SQL	Document NoSQL
Схема	Гнучка (schemaless)	Жорстка (DDL)	Гнучка
Free Tier	512 МБ (M0)	500 МБ	1 ГБ
Node.js підтримка	Відмінна (Mongoose)	Добра (Prisma)	Добра (SDK)
Агрегація	Потужний pipeline	Стандартний SQL	Обмежена
Реального часу	Через WebSocket	Через WebSocket	Вбудована
Масштабування	Горизонтальне (sharding)	Вертикальне	Автоматичне

Платформу MongoDB Atlas обрано завдяки поєднанню гнучкої документо-орієнтованої схеми (що відповідає структурі ігрових записів), безкоштовного хмарного tier, потужного агрегаційного pipeline для статистичних запитів та природної відповідності між BSON-документами та JavaScript-об’єктами. Firebase Firestore має вбудовану підтримку реального часу, але значно обмеженіші можливості агрегації. PostgreSQL забезпечує ACID-гарантії, але вимагає фіксованої схеми та складніших JOIN-запитів для ігрової статистики.

1.3.4 Архітектурні патерни вебзастосунків

Клієнт-серверна архітектура є основоположним патерном розподілених обчислень, що визначає розподіл відповідальностей між двома типами компонентів. З ускладненням застосунків виникла тришарова архітектура:

Presentation Layer (рівень представлення – браузер), Business Logic Layer (рівень бізнес-логіки – сервер застосунків), Data Layer (рівень даних – СУБД). Ця модель дозволяє масштабувати кожен рівень незалежно та розподіляти навантаження. Розроблюваний застосунок реалізує саме тришарову архітектуру: React (Presentation), Node.js/Express (Business Logic), MongoDB Atlas (Data).

Архітектурний стиль REST (Representational State Transfer) призначений для розподілених гіпермедіа-систем та запропонований Роєм Філдінгом у дисертації 2000 року. Цей стиль визначає шість обмежень, яким повинна відповідати архітектура: клієнт-серверна модель (client-server); відсутність стану (statelessness) – кожен запит містить всю необхідну інформацію; кешування (cacheability); єдиний інтерфейс (uniform interface); багаторівнева система (layered system); код на вимогу (code on demand, опціонально).

Ключовий принцип REST – ресурсо-орієнтований підхід: кожен іменник предметної області представлений URI, а HTTP-дієслова (GET, POST, PUT, DELETE) визначають операцію над ресурсом. У AimPractice: /api/scores – колекція результатів (GET для читання, POST для створення, DELETE для видалення); /api/auth/login – окремий ресурс для аутентифікаційної операції; /api/profile/:username – ресурс профілю конкретного гравця. Такий підхід робить API самодокументованим та передбачуваним для розробників.

Монолітна архітектура передбачає розробку всіх компонентів застосунку як єдиного розгортуваного артефакту. Переваги: простота розробки та налагодження, відсутність мережевих викликів між компонентами, легкість розгортання. Недоліки: ускладнення масштабування (масштабується весь моноліт, а не вузьке місце), ризик регресій при змінах будь-якого компонента.

Мікросервісна архітектура декомпозує застосунок на невеликі незалежні сервіси, кожен з яких відповідає за конкретну бізнес-функцію, має власну базу даних та розгортається незалежно. Переваги: незалежне масштабування, ізоляція відмов, свобода вибору технологій для кожного сервісу. Недоліки: складність управління, мережеві затримки між сервісами, розподілені транзакції.

Враховуючи обсяг необхідної функціональності, AimPractice реалізовано як модульний моноліт: серверна частина є єдиним Node.js-процесом, але розбита на логічні модулі (маршрути, схеми, middleware). Цей підхід є оптимальним для проектів середньої складності: зберігає простоту монолітної архітектури, але забезпечує достатній рівень організації коду для майбутнього рефакторингу в мікросервіси при необхідності масштабування. Цикл REST запит-відповідь у застосунку AimPractice зображено на рисунку 1.2.

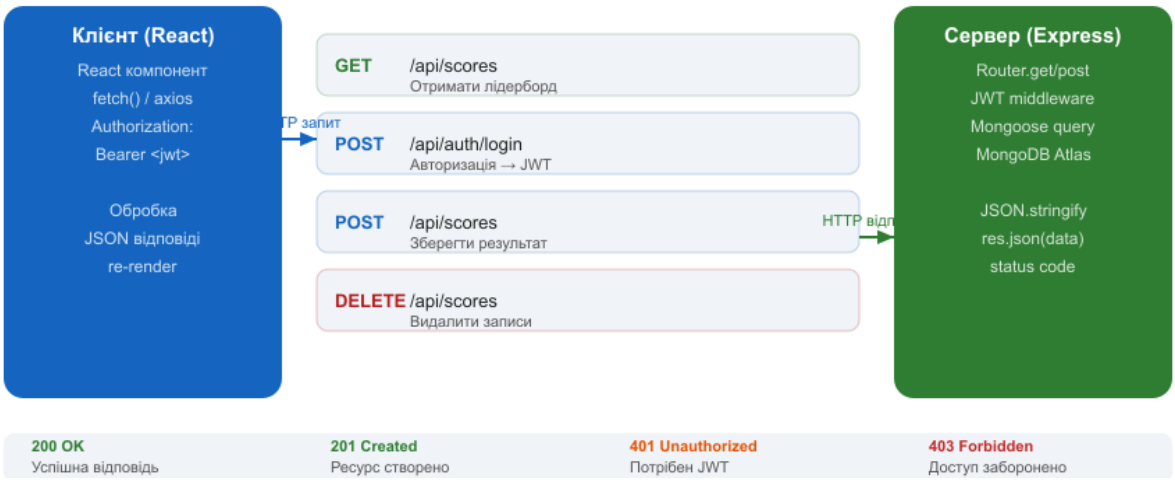

Рисунок 1.2 – Цикл REST запит-відповідь та HTTP-методи AimPractice

1.4 Постановка завдання

На основі проведеного аналізу предметної області та існуючих рішень сформульовано функціональне ядро розроблюваної системи. Вона повинна забезпечити:

– доступність через браузер без встановлення додаткового програмного забезпечення;

– не менше чотирьох режимів тренування прицілювання з налаштуванням складності;

– безпечну реєстрацію та авторизацію з хмарним зберіганням даних;

– глобальний лідерборд з оновленнями в реальному часі;

– публічні профілі гравців з детальною статистикою та графіками прогресу;

– налаштування параметрів гри (чутливість миші, кастомний приціл).

Технологічний стек AimPractice охоплює всі три рівні тришарової архітектури, як показано на рисунку 1.3. Клієнтський рівень реалізований на React 19 та Three.js, серверний – на Node.js/Express з JWT-авторизацією, рівень даних – на MongoDB Atlas.

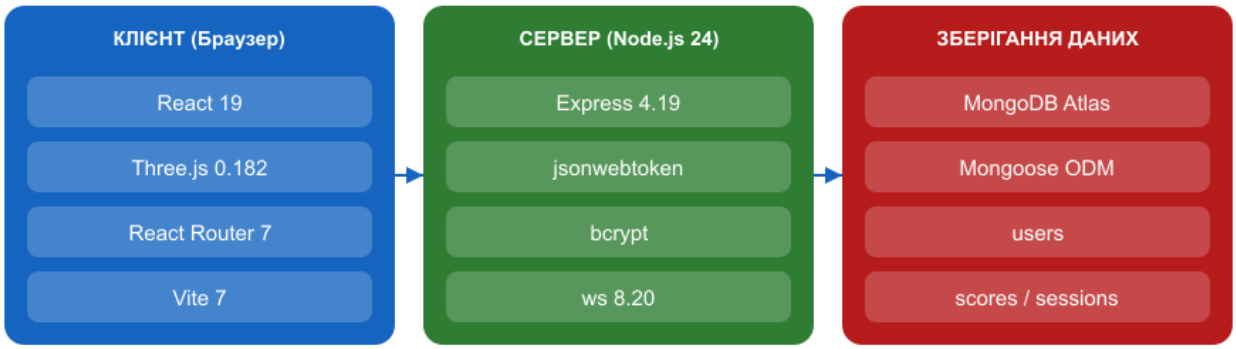

Рисунок 1.3 – Технологічний стек проекту AimPractice

Для реалізації безпечної автентифікації найбільш поширеним рішенням є JSON Web Token (JWT) – відкритий стандарт RFC 7519 для безпечної передачі даних між сторонами [6]. Цей токен складається з трьох частин: Header (тип та алгоритм), Payload (дані користувача) та Signature (підпис HMAC-SHA256). Перевага JWT над сесіями – stateless характер: сервер не зберігає жодних даних про сесії, що спрощує масштабування. У проекті токени видаються терміном 30 днів та зберігаються у localStorage клієнта.

Також використовується криптографічний алгоритм bcrypt для хешування паролів, навмисно ресурсомісткий [7]. Параметр cost factor (у проекті cost=10) визначає кількість ітерацій: при cost=10 хешування займає ~100 мс на сучасному CPU, що унеможливлює атаки грубої сили. Цей алгоритм автоматично генерує унікальну сіль для кожного хешу, захищаючи від атак на основі райдужних таблиць.

Для реалізації роботи в режимі реального часу популярним рішенням є WebSocket – протокол двостороннього з'єднання поверх TCP, стандартизований IETF у RFC 6455 [5]. На відміну від HTTP, WebSocket встановлює постійне з'єднання, що дозволяє обом сторонам ініціювати передачу даних у будь-який момент. Це принципово важливо для лідерборду: замість того, щоб клієнт постійно опитував сервер (polling), сервер сам надсилає оновлення при появі нових результатів.

Бібліотека ws – легковагова Node.js реалізація WebSocket. Вона надає WebSocketServer для прийому з'єднань та простий API для роботи з клієнтами. У проекті ws-сервер прив'язується до того самого HTTP-сервера, що й Express, через параметр { server: httpServer, path: '/ws' }. Це дозволяє обслуговувати HTTP та WebSocket на одному порту.

Збирання програмного забезпечення покладено на інструмент Vite [8]. Головна відмінність від Webpack полягає у підході до розробки: Vite не виконує попередню збірку усього проекту при запуску dev-сервера. Він використовує нативні ES Modules браузера та перетворює файли на вимогу, що забезпечує практично миттєвий старт сервера незалежно від розміру проекту та швидкий Hot Module Replacement.

У проєкті Vite також використовується для конфігурації проксі: HTTP-запити до /api перенаправляються на http://localhost:3001 (бекенд), а WebSocket-з'єднання /ws – на ws://localhost:3001. Це дозволяє уникнути CORS-проблем у режимі розробки та зберігати єдину точку входу для клієнта.

Технічні вимоги до системи: частота кадрів не нижче 60 FPS на сучасному ПК; час відповіді REST API не більше 500 мс; затримка WebSocket-повідомлень не більше 100 мс; зберігання паролів виключно у вигляді bcrypt-хешу; захист API маршрутів запису через JWT middleware.

На основі проведеного аналізу обрано технологічний стек: React 19 (висока продуктивність, найбільша екосистема, легка інтеграція з Three.js), Node.js + Express (JavaScript на бекенді, неблокуюча I/O), MongoDB Atlas (гнучка схема,

хмарний хостинг, безкоштовний tier), JWT + bcrypt (stateless авторизація), WebSocket/ws (real-time оновлення).

1.5 Безпека веб-застосунків

1.5.1 Основні вектори атак та OWASP Top 10

Організація OWASP (Open Web Application Security Project) є міжнародною некомерційною спільнотою, що публікує рейтинг найкритичніших ризиків безпеки веб-застосунків (OWASP Top 10) [15]. Згідно з актуальним виданням 2021 року, провідні ризики включають: Broken Access Control (порушення контролю доступу), Cryptographic Failures (криптографічні вразливості), Injection (ін’єкції), Insecure Design (небезпечне проєктування) та Security Misconfiguration (неправильна конфігурація безпеки).

При розробці AimPractice враховані заходи захисту від провідних векторів атак відповідно до рекомендацій OWASP [15]. Broken Access Control усунуто через JWT-middleware на всіх маршрутах запису: кожен POST/DELETE запит перевіряє токен, і лише власник токена може зберігати результати від свого імені. Cryptographic Failures усунуто застосуванням bcrypt для паролів (замість MD5 або незашифрованого зберігання) та HTTPS для транспортного шару у продакшн.

1.5.2 Міжсайтовий скриптинг (XSS) та захист React

Міжсайтовий скриптинг (Cross-Site Scripting, XSS) – атака, за якої зловмисник впроваджує шкідливий JavaScript-код у сторінку, що виконується у браузері жертви. Така атака дозволяє вкрасти cookies, localStorage (у тому числі JWT-токен), виконати дії від імені користувача або перенаправити на фішинговий сайт.

Бібліотека React забезпечує автоматичний захист від XSS через екранування (escaping) усіх значень, що вставляються через JSX-вирази {}. Перетворення рядка на HTML-сутності (< → <, > → >, " → ") унеможливлює виконання вбудованого JavaScript. У AimPractice відсутні

виклики dangerouslySetInnerHTML (єдиний спосіб обійти захист React), а всі дані користувача (нікнейм, результати) відображаються через звичайні JSX-вирази.

1.5.3 Підробка міжсайтових запитів (CSRF)

Атака CSRF (Cross-Site Request Forgery) полягає у тому, що зловмисник змушує браузер жертви відправити авторизований запит до цільового сайту без відома користувача. Класичний захист від CSRF – CSRF-токен, що включається у форму та перевіряється сервером. Проте застосування JWT у заголовку Authorization: Bearer замість cookie-сесій природньо захищає від CSRF.

Причина захисту: браузер автоматично включає cookies до кожного запиту на відповідний домен, що робить cookie-сесії вразливими до CSRF. Токен JWT зберігається у localStorage та додається до запитів програмно через JavaScript – зловмисний сайт не може змусити браузер жертви виконати цей JavaScript-код у контексті чужого сайту (Same-Origin Policy). Таким чином, stateless JWT-авторизація є природньо CSRF-захищеною.

1.5.4 NoSQL Injection та захист Mongoose

Атака типу NoSQL Injection є аналогом SQL Injection для документно-орієнтованих баз даних. Зловмисник може передати у JSON-тілі запиту об’єкт з MongoDB-операторами замість рядка. Наприклад, замість { username: 'admin' } передати { username: { '$gt': '' } }, що поверне всіх користувачів у MongoDB-запиті findOne.

Бібліотека Mongoose автоматично застосовує типізацію схеми: поле username визначено як String, тому Mongoose приведе будь-яке значення до рядка перед запитом до MongoDB – об’єкт { '$gt': '' } перетвориться на рядок '[object Object]', що не збіжиться з жодним нікнеймом. Додатковий захист: перевірка typeof username === 'string' у маршрутах авторизації та обмеження maxLength у формі. Комбінація типізації Mongoose та валідації на рівні застосунку унеможливлює NoSQL Injection у AimPractice. Багаторівневу систему захисту застосунку показано на рисунку 1.4.

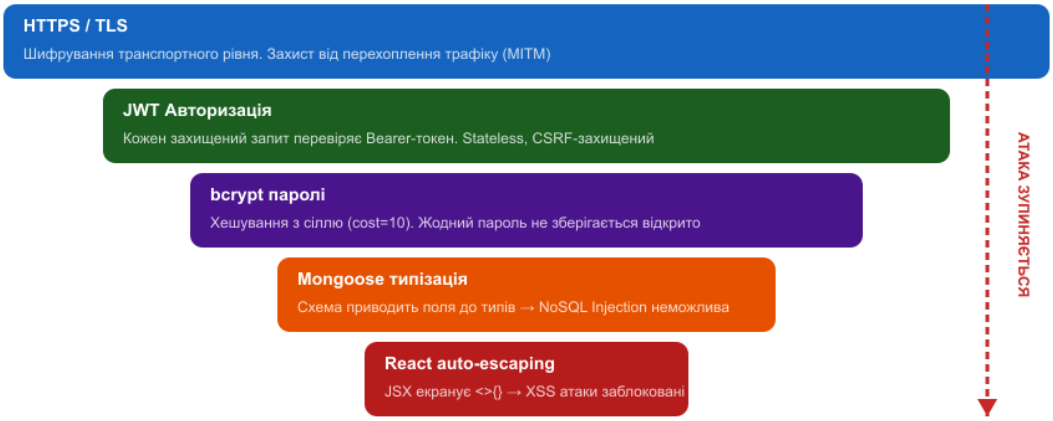

Рисунок 1.4 – Шари захисту безпеки AimPractice

Висновки до розділу 1. Проведений аналіз предметної області підтверджує актуальність розробки вебтренажера прицілювання. Жоден із розглянутих конкурентів (Aim Lab, KovaaK’s, 3D Aim Trainer, Aimlabs Web) не поєднує доступність без встановлення, повноцінну авторизацію, глобальний лідерборд із WebSocket та публічні профілі. Порівняльний аналіз технологій обґрунтував вибір React 19, Node.js/Express, MongoDB Atlas та WebSocket як оптимального стеку для реалізації поставлених завдань. Теоретичне підґрунтя у сферах безпеки та архітектурних патернів підтверджує обґрунтованість прийнятих технічних рішень.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Вимоги до системи

2.1.1 Функціональні вимоги

Функціональні вимоги (табл. 1.1) описують поведінку системи та визначені на основі аналізу цільової аудиторії (FPS-гравці, кіберспортсмени) та недоліків існуючих рішень.

Таблиця 2.1 – Функціональні вимоги до системи

№	Вимога	Пріоритет	Статус
1	Реєстрація гравця з унікальним нікнеймом та паролем (мін. 4 символи)	Критичний	Реалізовано
2	Авторизація з видачею JWT-токена терміном 30 днів	Критичний	Реалізовано
3	Режим Polygon: сферичні цілі у випадкових позиціях	Критичний	Реалізовано
4	Режим Grid: цілі у регулярній сітці	Високий	Реалізовано
5	Режим Flick: короткочасні цілі (0.6–1.5 с) для флік-шотів	Високий	Реалізовано
6	Режим Tracking: рухома ціль для тренування відстеження	Високий	Реалізовано
7	Три рівні складності для кожного режиму (Easy, Normal, Hard)	Критичний	Реалізовано
8	Таймер сесії з конфігурованою тривалістю (30–120 с)	Середній	Реалізовано
9	Підрахунок очків, влучань, промахів та серій (streak)	Критичний	Реалізовано
10	Збереження результатів у MongoDB після кожної сесії	Критичний	Реалізовано
11	Глобальний лідерборд з фільтрацією за режимом та складністю	Критичний	Реалізовано
12	WebSocket-оновлення лідерборду в реальному часі	Високий	Реалізовано
13	Публічний профіль гравця за URL /profile/:username	Середній	Реалізовано
14	Налаштування чутливості миші та гучності звуку	Середній	Реалізовано
15	Кастомний приціл (колір, розмір, товщина, проміжок, крапка)	Низький	Реалізовано

2.1.2 Нефункціональні вимоги

Таблиця 2.2 – Нефункціональні вимоги до системи

Категорія	Вимога	Метрика
Продуктивність	Частота кадрів 3D-сцени	≥ 60 FPS на сучасному ПК
Продуктивність	Час відповіді REST API	< 500 мс (95-й перцентиль)
Продуктивність	Затримка WebSocket-повідомлень	< 100 мс
Безпека	Зберігання паролів	Тільки bcrypt-хеш, cost=10
Безпека	Захист API запису/видалення	JWT middleware requireAuth
Безпека	Захист від XSS	Відсутність innerHTML з user-input
Доступність	Підтримувані браузери	Chrome 90+, Firefox 88+, Edge 90+
Доступність	Вимоги до встановлення	Тільки браузер, без плагінів
Надійність	Збереження даних	MongoDB Atlas, автоматичні резервні копії
Масштабованість	Додавання нових режимів	Без змін базової архітектури

2.2 Архітектура системи

Застосунок AimPractice побудований за тришаровою клієнт-серверною архітектурою (Three-Tier Architecture), де кожен шар має чітко визначену відповідальність та комунікує з іншими через стандартизовані інтерфейси.

Перший шар – клієнтський (Presentation Layer) – виконується у браузері. Він відповідає за рендеринг інтерфейсу, управління станом на стороні клієнта, захоплення введення та 3D-рендеринг. Реалізований на React 19 та Three.js. Взаємодіє із сервером через два канали: HTTP REST API (запити-відповіді) та WebSocket (push-повідомлення сервера).

Другий шар – серверний (Business Logic Layer) – виконується у Node.js. Відповідає за обробку запитів, JWT-авторизацію, валідацію даних, управління WebSocket-з'єднаннями та трансляцію повідомлень. Реалізований на Express.js з middleware-архітектурою.

Третій шар – шар даних (Data Layer) – реалізований у хмарній базі MongoDB Atlas. Забезпечує надійне зберігання, індексування та вибірку даних. Взаємодія між серверним та шаром даних відбувається через ODM Mongoose.

Комунікація між шарами здійснюється через два механізми. Перший – HTTP REST API: клієнт ініціює запит (GET, POST, DELETE), сервер обробляє та повертає відповідь у форматі JSON. Цей механізм використовується для всіх операцій, що не потребують негайного оновлення (авторизація, збереження результату, завантаження статистики). Другий механізм – WebSocket: після встановлення постійного з'єднання сервер може самостійно ініціювати передачу даних клієнту без очікування запиту. Використовується виключно для push-оновлень лідерборду при появі нового результату.

Тришарова архітектура забезпечує кілька ключових переваг. Розподіл відповідальностей (Separation of Concerns) дозволяє змінювати один шар незалежно від інших: наприклад, клієнтська частина може бути замінена мобільним застосунком без змін серверного API та бази даних. Масштабованість: серверний шар може бути розгорнутий у кількох примірниках за балансувальником навантаження без змін у клієнті. Безпека: конфіденційна

логіка (перевірка паролів, підпис JWT) ніколи не виконується у браузері та недоступна для аналізу через DevTools.

Вибір хмарної MongoDB Atlas як шару даних обумовлений кількома факторами. Atlas надає автоматичні резервні копії з точністю до 1 хвилини та географічну реплікацію між дата-центрами без додаткового налаштування. Безсерверна модель (shared cluster у безкоштовному тарифі M0) дозволяє розпочати розробку без виділеного сервера бази даних. Вбудований моніторинг (Atlas Performance Advisor) надає рекомендації щодо індексів на основі реального навантаження. Підключення через рядок MONGO_URI абстрагує деталі мережевої топології кластера від серверного коду.

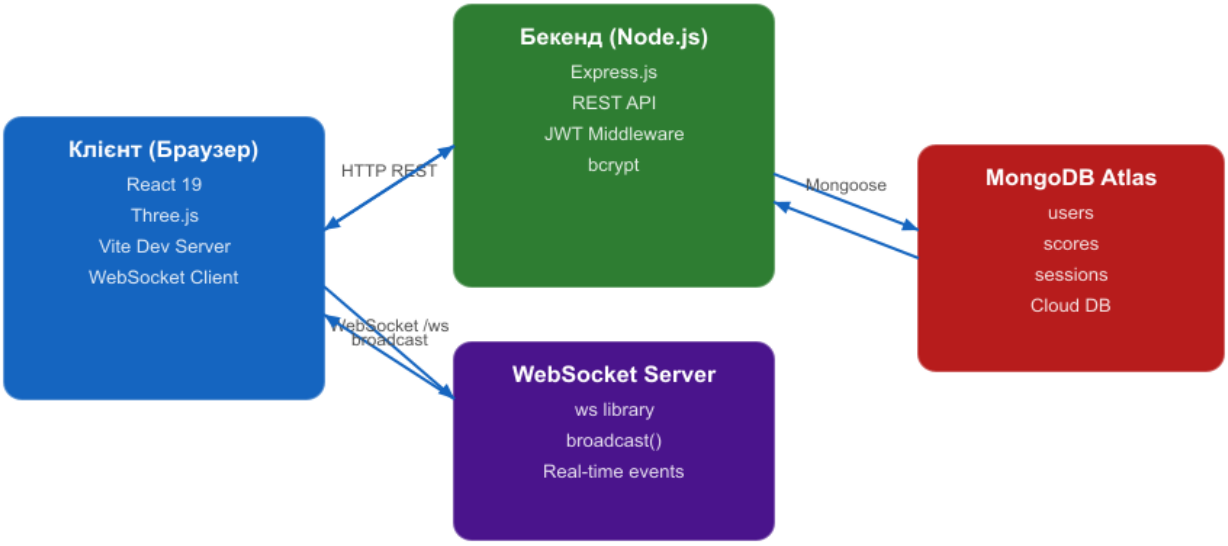

Рисунок 2.1 – Архітектура системи AimPractice

2.3 Проектування бази даних

База даних AimPractice містить три колекції: users, scores та sessions. Документно-орієнтована модель обрана через гнучкість структури ігрових записів та відсутність необхідності у складних JOIN-запитах.

Колекція users зберігає дані зареєстрованих гравців. Поле username – унікальний індексований рядок, що виступає первинним ідентифікатором гравця в усій системі. password зберігає bcrypt-хеш. createdAt – Unix timestamp у мілісекундах.

Колекції scores та sessions мають ідентичну структуру полів: playerName (рядок, нікнейм гравця), mode (рядок: polygon/grid/flick/tracking), difficulty (рядок: easy/normal/hard), score (число, набрані очки), hits (кількість влучань), misses (промахи), accuracy (відсоток влучань), timeMs (тривалість сесії), bestStreak (найкраща серія), createdAt (Unix timestamp). Різниця у призначенні: scores – топ-20 для лідерборду, sessions – повна хронологічна історія (останні 100 на гравця).

Вибір документно-орієнтованої моделі MongoDB обумовлений специфікою ігрових даних. Реляційна SQL-схема потребувала б нормалізованих таблиць із JOIN-запитами при кожному зверненні до лідерборду. Документна модель MongoDB дозволяє зберігати всі поля ігрового результату в одному документі, що відповідає патерну «embed what you query together»: лідерборд та сторінка профілю отримують усі необхідні дані за один запит без об’єднань таблиць. Відсутність фіксованої схеми дозволяє в майбутньому додавати нові поля до документів без міграцій.

Стратегія індексування розроблена відповідно до основних патернів доступу до даних. Колекція users має унікальний індекс на полі username, що забезпечує пошук за логарифмічний час при авторизації та унеможливлює дублювання нікнеймів на рівні бази даних. Колекція scores має складений індекс { mode: 1, difficulty: 1, score: -1 }: перші два поля забезпечують ефективну фільтрацію за режимом і складністю, останнє поле зі значенням -1 впорядковує результати за спаданням очок відповідно до запиту лідерборду. Колекція sessions індексована за { playerName: 1, createdAt: -1 } для ефективного завантаження хронологічної історії конкретного гравця.

Mongoose-схеми виконують роль валідаційного шару між сервером та базою даних. Схема Score включає обов’язкові поля (required: true) для playerName, mode, difficulty, score, hits, misses, accuracy, timeMs; перелік допустимих значень (enum) для mode та difficulty; мінімальні значення (min: 0) для числових полів. Mongoose автоматично відхиляє документи, що не відповідають схемі. Управління обсягом реалізовано алгоритмом обрізання:

після кожного збереження результату система видаляє записи нижче 20-ї позиції. Це гарантує, що колекція scores ніколи не перевищить 240 документів (4 режими × 3 складності × 20 позицій), а час виконання запиту лідерборду залишається стабільним незалежно від кількості гравців.

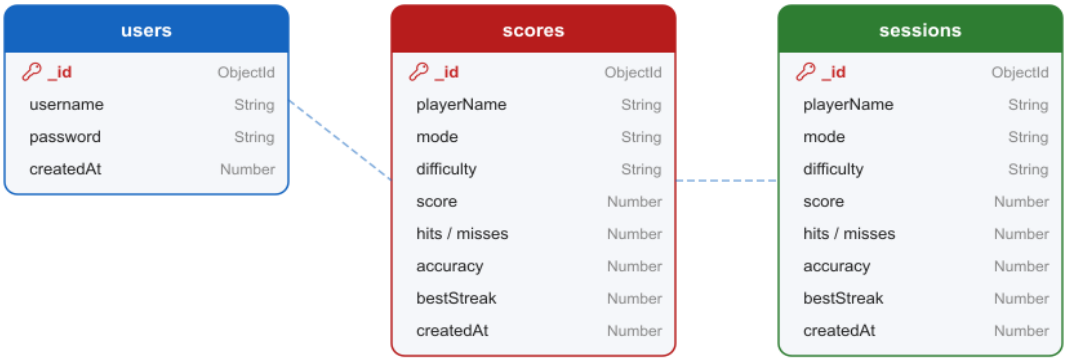

Рисунок 2.2 – Схема колекцій бази даних MongoDB

2.4 Проектування REST API

Програмний інтерфейс REST API розроблений за принципами REST: ресурси ідентифікуються URI, операції над ресурсами відповідають HTTP-методам, відповіді мають стандартну структуру JSON. Інтерфейс розміщений на базовому шляху /api та поділений на чотири групи маршрутів.

Метод	Маршрут	Опис	Доступ	Відповідь
POST	/api/auth/register	Реєстрація нового гравця	Public	{ token, username }
POST	/api/auth/login	Авторизація гравця	Public	{ token, username }
GET	/api/scores	Отримання лідерборду	Public	Score[]
POST	/api/scores	Додати результат	JWT	{ ok: true }
DELETE	/api/scores	Очистити лідерборд	JWT	{ ok: true }
GET	/api/sessions	Отримати сесії гравця	Public	Session[]
POST	/api/sessions	Зберегти сесію	JWT	{ ok: true }
GET	/api/profile/:u	Публічний профіль	Public	ProfileData

Рисунок 2.3 – Таблиця REST API маршрутів

Маршрути /api/auth/register та /api/auth/login є публічними та обробляють реєстрацію і авторизацію відповідно. Маршрути GET /api/scores та GET /api/sessions є публічними для перегляду лідерборду та статистики. Всі

маршрути, що змінюють стан (POST та DELETE для scores і sessions), захищені JWT-middleware requireAuth. GET /api/profile/:username є публічним для перегляду профілів.

Стандартні HTTP-коди відповідей: 200 OK (успішний запит), 400 Bad Request (помилки валідації), 401 Unauthorized (неавторизований запит), 404 Not Found (ресурс відсутній), 409 Conflict (дублікат, напр. існуючий нікнейм), 500 Internal Server Error. Повний перелік маршрутів наведено на рисунку 2.3.

Маршрутизація Express побудована за принципом middleware-chain. Кожен запит проходить через глобальні middleware (cors(), express.json()) перед потраплянням до обробників маршрутів. Захищені маршрути додатково обробляються middleware requireAuth, що перевіряє JWT-токен та встановлює req.user.username перед передачею керування обробнику. Послідовне застосування middleware чітко розділяє аспекти безпеки, парсингу та бізнес-логіки: кожен обробник маршруту займається виключно своєю задачею, не дублюючи логіку авторизації.

Стандартна структура відповідей API забезпечує передбачуваність для клієнтського коду. Успішні відповіді повертають JSON-об’єкт безпосередньо (GET-запити – масив або об’єкт) або об’єкт із полями token та username при авторизації. Помилки повертають { error: повідомлення } з відповідним HTTP-кодом. Функція apiFetch() перевіряє response.ok та викидає Error із полем error – це дозволяє компонентам обробляти бекендові помилки в catch-блоках уніфіковано, без аналізу HTTP-статусу у кожному місці виклику.

Валідація вхідних даних реалізована на двох рівнях. На рівні Express-маршруту перевіряється наявність обов’язкових полів та їх базові обмеження: username та password при реєстрації, довжина пароля не менше 4 символів, наявність числових полів score, hits, misses, timeMs при збереженні результату. На рівні Mongoose-схеми виконується типова перевірка та валідація enum-значень для mode та difficulty. Двоступенева валідація зменшує ймовірність некоректних даних у базі та забезпечує чіткі повідомлення про помилки для налагодження клієнтського коду.

2.5 Проектування WebSocket-протоколу

Сервер WebSocket слухає з’єднання на шляху /ws того самого HTTP-сервера, що й Express. Використання спільного HTTP-сервера (httpServer) дозволяє обслуговувати HTTP та WebSocket на одному порту, спрощуючи конфігурацію та розгортання.

Протокол обміну повідомленнями: після встановлення з’єднання сервер надсилає { type: 'connected' } як підтвердження готовності. При публікації нового результату сервер надсилає broadcast усім клієнтам: { type: 'scores_updated', mode, difficulty, entries }. Клієнт фільтрує повідомлення за type та порівнює mode/difficulty з поточними фільтрами лідерборду – це унеможливлює зайві оновлення при перегляді іншого розділу. Діаграму WebSocket-оновлення лідерборду зображено на рисунку 2.4.

Управління WebSocket-з’єднаннями реалізовано через бібліотеку ws з використанням вбудованої множини клієнтів wss.clients. При встановленні нового з’єднання сервер негайно надсилає підтвердження { type: connected }, що клієнт обробляє встановленням прапора connected та відображенням індикатора LIVE. Обробка помилок ws.on(error) логує проблеми без переривання роботи сервера. При закритті з’єднання бібліотека ws автоматично видаляє клієнта з wss.clients, тому функція broadcast() не потребує окремого списку підключень – вона ітерує по wss.clients напряму.

На стороні клієнта реалізоване коректне закриття з’єднання при демонтуванні компонента. Cleanup-функція useEffect виконує ws.close() при виході з LeaderboardScreen, що запобігає накопиченню зомбі-з’єднань при навігації між екранами. Відсутність автоматичної реконнекції є прийнятним компромісом для поточної версії: при збоях з’єднання індикатор LIVE зникає, а дані лідерборду продовжують завантажуватись через HTTP. Для майбутніх версій рекомендується реалізація reconnect з exponential backoff для підвищення надійності.

Безпека WebSocket-каналу забезпечена на архітектурному рівні: WS-канал призначений виключно для отримання широкомовних повідомлень про

оновлення лідерборду (read-only). Усі операції запису даних виконуються тільки через HTTP REST API з JWT-авторизацією. Сервер ігнорує повідомлення від клієнтів по WebSocket, що унеможливлює несанкціоновану зміну даних через WS-канал. Такий розподіл обов'язків між HTTP та WebSocket є стандартною практикою для real-time вебзастосунків.

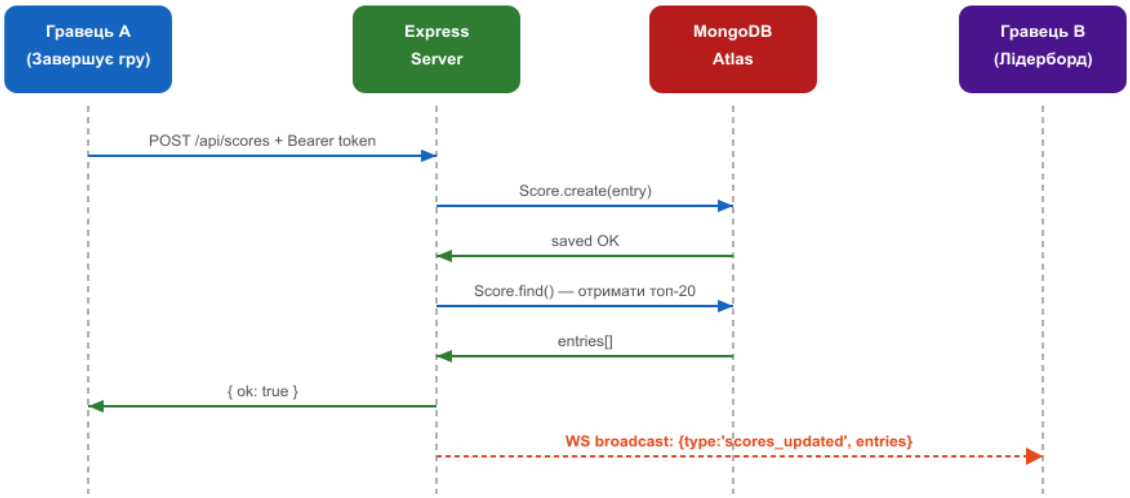

Рисунок 2.4 – Діаграма WebSocket-оновлення лідерборду

2.6 Проєктування структури проєкту

Проект організований у монорепозиторій з двома незалежними модулями. Клієнтська частина (директорія кореня) та серверна частина (server/). Така структура дозволяє розробляти та запускати обидві частини незалежно.

Клієнтська частина організована за функціонально-доменним принципом: src/screens/ – компоненти-екрани застосунку, src/ThreeScene/ – 3D-рушій на Three.js, src/hooks/ – React-хуки, src/storage/ – шар доступу до даних, src/api/ – HTTP-клієнт та API-функції. Серверна частина – за патерном MVC: server/models/ – Mongoose-схеми, server/routes/ – Express-маршрути, server/middleware/ – JWT-middleware, server/ws.js – WebSocket-сервер, server/index.js – точка входу.

Вибір монорепозиторію обумовлений зручністю розробки та спрощенням розгортання. Клієнтська та серверна частини мають спільний git-репозиторій, але незалежні файли package.json із власними залежностями. Це дозволяє

запускати обидві частини незалежно, не встановлюючи зайвих залежностей. Альтернатива – два окремих репозиторії – ускладнила б координацію при зміні формату API-відповіді, оскільки потребувала б синхронних комітів у двох місцях та узгодження версій.

Директорія src/screens/ містить по одному JSX-файлу на кожен екран застосунку: RegisterNameScreen, MainMenuScreen, PlayScreen, GameScreen, ResultScreen, LeaderboardScreen, StatsScreen, SettingsScreen, ProfileScreen. Директорія src/ThreeScene/ інкапсулює весь код Three.js: головний компонент сцени, логіку режимів, генерацію цілей, анімацію вибуху та звуковий менеджер. Ізоляція Three.js у окрему директорію дозволяє розробляти та тестувати 3D-частину незалежно від React-компонентів.

Серверна частина організована за спрощеним MVC-патерном. Директорія server/models/ містить три файли Mongoose-схем: User.js, Score.js, Session.js. Директорія server/routes/ – чотири маршрутних файли: auth.js, scores.js, sessions.js, profile.js. Файл server/middleware/auth.js реалізує єдиний JWT-middleware requireAuth. Файл server/ws.js ініціалізує WebSocket-сервер та експортує функцію broadcast(). Такий розподіл відповідає принципу єдиної відповідальності та спрощує локалізацію та виправлення помилок.

Процес збірки для продакшн розрізняє клієнтську та серверну частини. Vite збирає клієнтський код у директорію dist/: мінімізований JS-бандл розміром близько 1.82 МБ (що включає Three.js та React), скопійовані статичні ресурси та index.html з хешованими іменами файлів для cache-busting. Серверна частина не потребує збірки – Node.js виконує JS-файли напряму. У продакшн-середовищі Express може роздавати клієнтський dist/ як статичні файли, дозволяючи розгорнути весь застосунок на одному сервері.

2.7 Проєктування інтерфейсу користувача

2.7.1 Принципи та концепція UI

Інтерфейс AimPractice спроектований за принципами мінімалізму та контекстної відповідності: у грі – максимально чистий HUD без відволікаючих

елементів, у меню – темна кольорова схема з акцентами у червоному кольорі (#FF6B6B), що асоціюється з кіберспортивною естетикою.

Навігація між екранами реалізована через стан бібліотеки та маршрутизатор. Основні переходи: головне меню → вибір режиму → гра → результат → головне меню. Бічні переходи: меню → лідерборд, меню → статистика, меню → налаштування. Кнопка «Back» на кожному екрані дозволяє повернутись у меню без втрати контексту.

2.7.2 Екрани застосунку та їх функції

RegisterNameScreen – екран реєстрації та авторизації. Дві вкладки (Sign In / Sign Up) у вигляді pill-перемикача розташовані над формою. При виборі Sign Up з’являється третє поле підтвердження пароля. Помилки відображаються під формою з іконкою попередження. Кнопка блокується та показує 'Loading…' під час запиту.

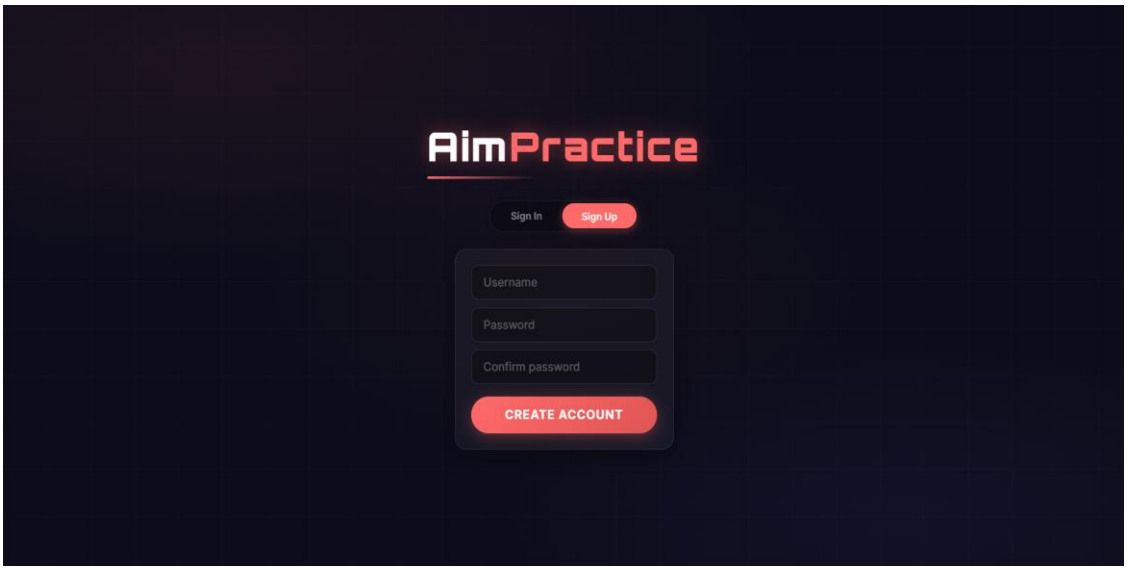

Рисунок 2.5 – Екран реєстрації та авторизації AimPractice

MainMenuScreen – головне меню. Відображає логотип AimPractice, нікнейм авторизованого гравця та сітку з чотирьох карток навігації: Play, Leaderboard, Stats, Settings. Кожна картка містить іконку, назву розділу та короткий опис. При наведенні картка піднімається вгору з підсвіткою знизу червоною лінією.

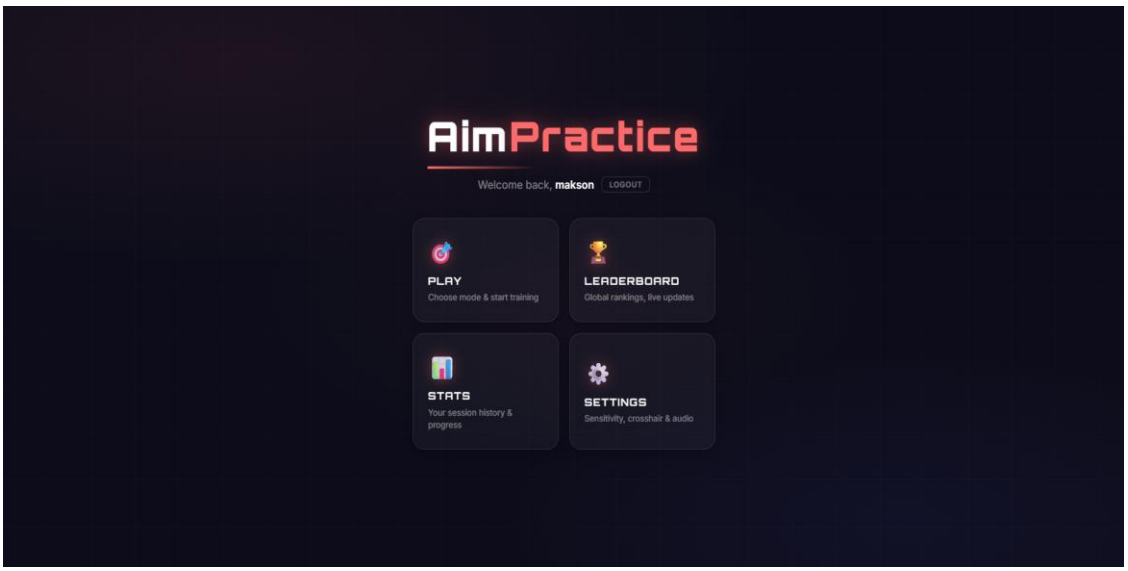

Рисунок 2.6 – Головне меню застосунку AimPractice

PlayScreen – вибір режиму та складності гри. Чотири картки режимів (Polygon, Grid, Flick, Tracking) розміщені у сітці 2×2, кожна містить іконку, назву та опис. При виборі режиму картка підсвічується червоним контуром та з’являється мітка ✓. Нижче розташовані вибір складності (Easy/Normal/Hard) та тривалості сесії (30/60/90/120 с). Кнопка Start показує назву обраного режиму.

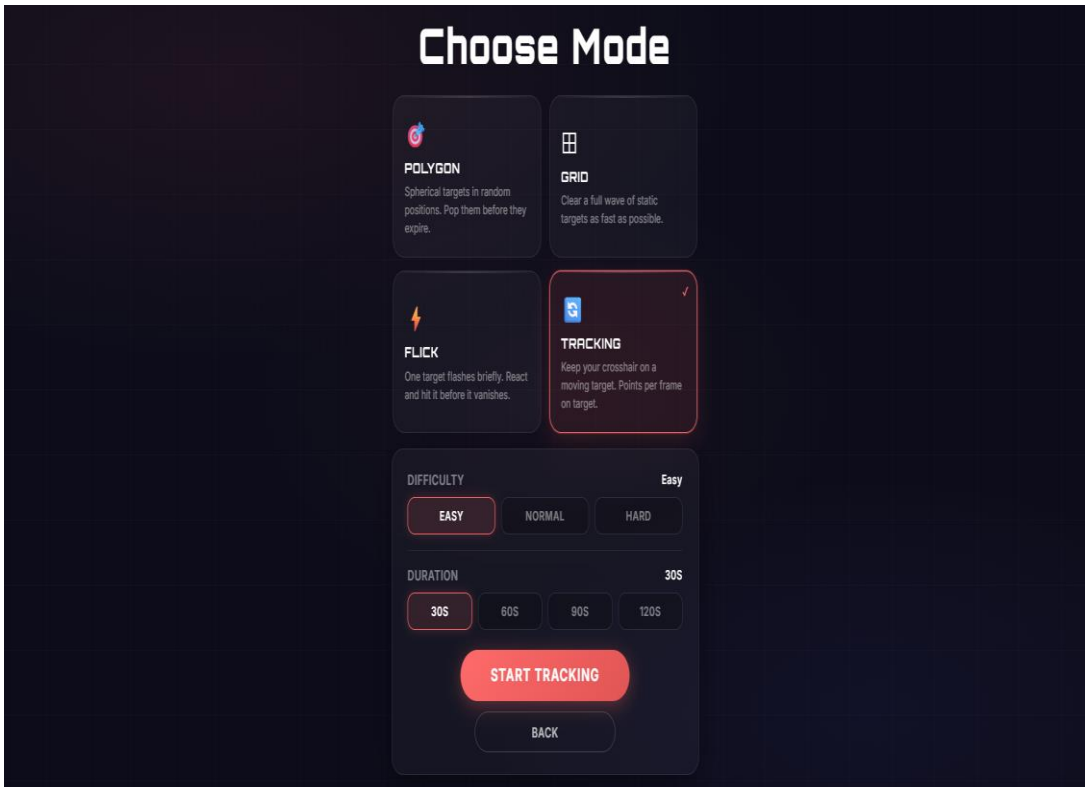

Рисунок 2.7 – Екран вибору режиму гри та налаштувань сесії

GameScreen та HUD – ігровий екран. Після захоплення курсору відображається мінімальний HUD: таймер зворотного відліку у лівому верхньому куті, поточний рахунок та відсоток точності у правому верхньому, назва поточного режиму по центру вгорі. Кастомний приціл розміщений у центрі екрану. 3D-сцена містить нічне оточення з зірками, підлогою з сіткою та активною ціллю. Клавіша Escape відкриває меню паузи.

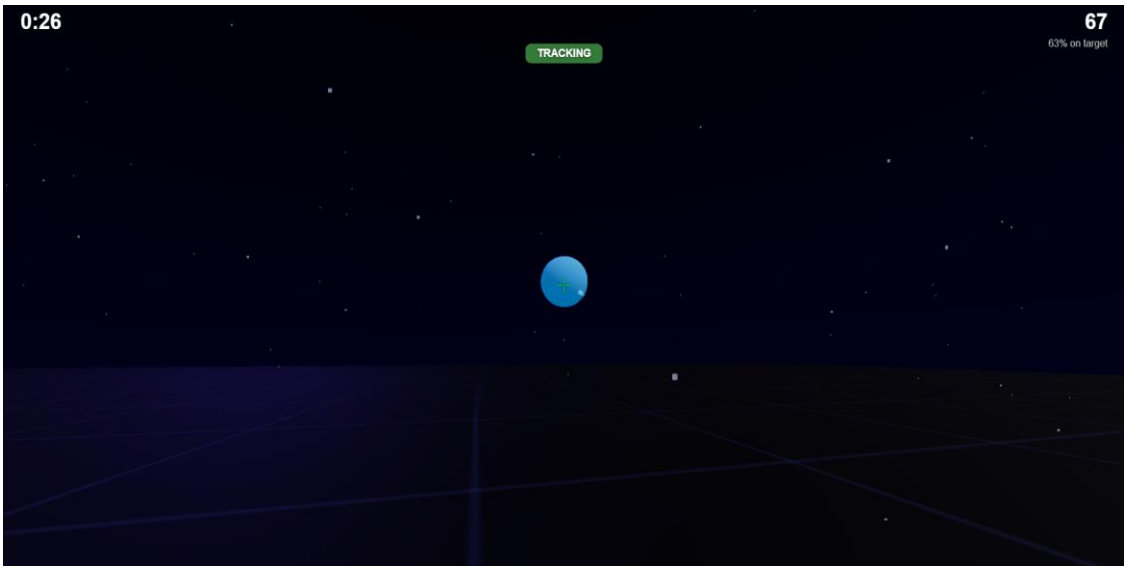

Рисунок 2.8 – Ігровий процес у режимі Tracking (ціль та HUD)

ResultScreen – екран результатів після завершення гри. Показує score, hits, misses, accuracy% та bestStreak у вигляді карток. Кнопки 'Play Again' та 'Main Menu'. При встановленні нового особистого рекорду відображається анімований банер.

LeaderboardScreen – глобальний лідерборд. У верхній частині розміщено заголовок 'Leaderboard' та зелений індикатор LIVE, що підтверджує активне WebSocket-з’єднання. Два ряди кнопок-фільтрів дозволяють обрати режим (Polygon, Grid, Flick, Tracking) та складність (Easy, Normal, Hard). Таблиця виводить місце, нікнейм, очки, точність, кількість влучань та дату. Запис поточного гравця підсвічується червонуватим фоном.

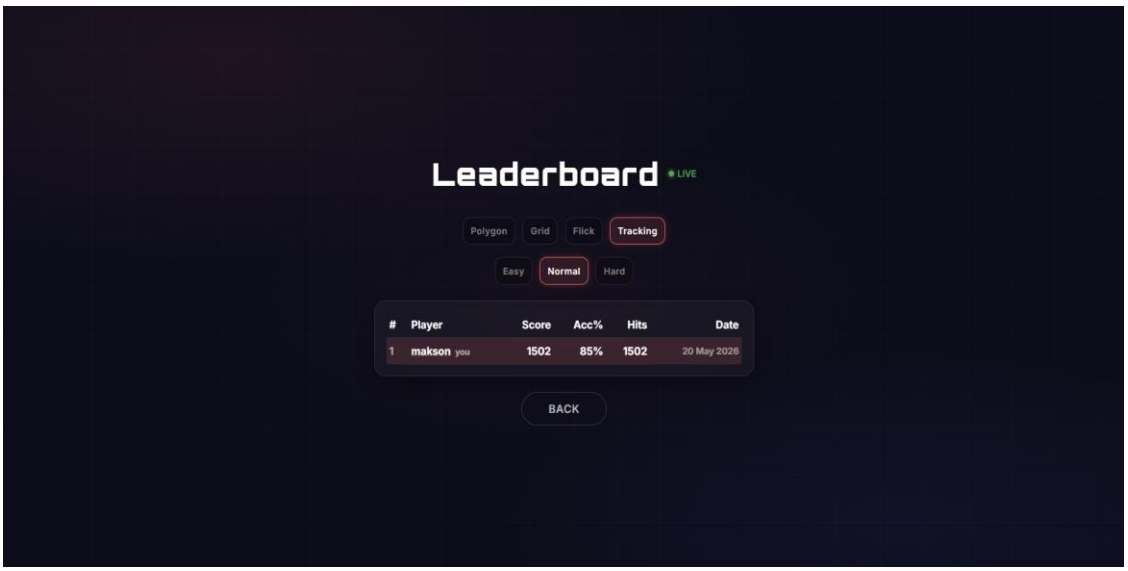

Рисунок 2.9 – Глобальний лідерборд з LIVE-індикатором WebSocket

StatsScreen – персональна статистика. Зведені метрики (загальна кількість ігор, точність, найкращий результат, загальний час гри) та таблиця останніх 20 сесій з фільтрацією за режимом.

SettingsScreen – налаштування. Повзунки чутливості миші (0.1–5.0) та гучності (0–100%). Кастомізатор прицілу: вибір кольору, розмір (1–20), товщина (1–5), проміжок (0–10), крапка по центру. Всі зміни застосовуються в реальному часі без перезавантаження.

Висновки до розділу 2. У розділі спроектована тришарова архітектура системи AimPractice з чітким розподілом відповідальностей між рівнями представлення, бізнес-логіки та зберігання даних. Визначені функціональні та нефункціональні вимоги, схема бази даних з трьома колекціями, REST API з чотирма групами маршрутів та WebSocket-протокол для оновлень у реальному часі. Спроєктований інтерфейс відповідає принципам мінімалізму та кіберспортивної естетики. Прийняті рішення забезпечують надійність, безпеку та можливість подальшого масштабування.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Реалізація бекенду

3.1.1 Серверна точка входу

Файл server/index.js є точкою входу серверної частини. Він ініціалізує Express-додаток, підключає middleware, реєструє маршрути, створює HTTP-сервер, прив'язує до нього WebSocket-сервер та підключається до MongoDB Atlas. Порядок ініціалізації є критичним: WebSocket-сервер підключається до httpServer до виклику listen(), що гарантує обробку WS-з'єднань з першого моменту роботи сервера. MongoDB-підключення та запуск HTTP-сервера поєднані в один асинхронний ланцюг: у разі невдалого підключення до БД сервер не запустить HTTP-слухач, що запобігає обробці запитів без доступу до даних.

Лістинг server/index.js наведено у Додатку А. Ключові аспекти реалізації: CORS-middleware дозволяє запити з будь-якого джерела (у продакшн слід обмежити доменом розгортання); express.json() автоматично парсить JSON-тіло запитів та наповнює req.body; перевірка наявності MONGO_URI при старті запобігає запуску без конфігурації бази даних.

3.1.2 Система авторизації

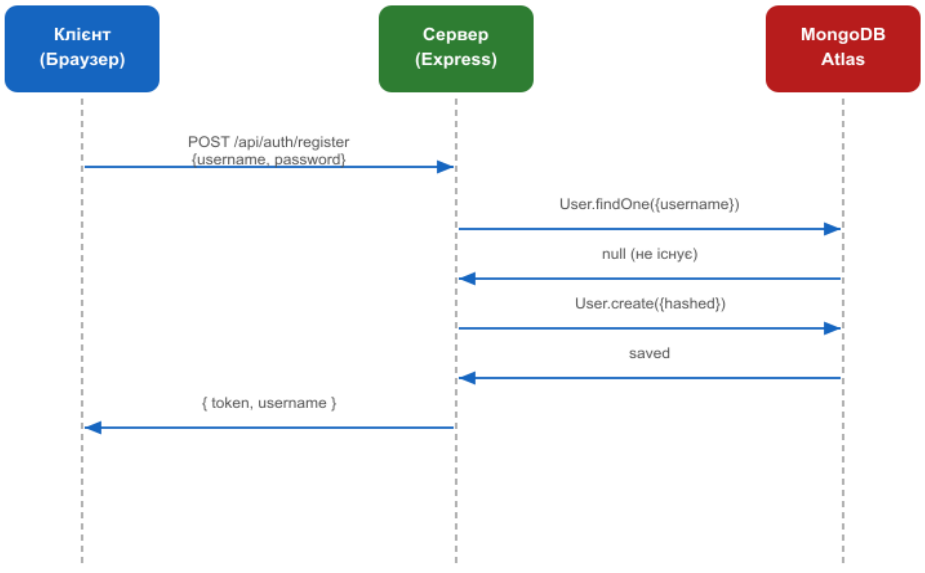

Рисунок 3.1 – Діаграма послідовності процесу реєстрації

Авторизація реалізована за stateless-підходом на основі JWT. Файл server/routes/auth.js (Додаток Б) містить два маршрути: POST /register та POST /login.

Процес реєстрації: сервер перевіряє наявність полів username та password, мінімальну довжину пароля (4 символи). Перевіряє унікальність нікнейму запитом до БД. Хешує пароль через bcrypt.hash(password, 10) та зберігає у колекцію users. Генерує JWT-токен: jwt.sign({ username }, JWT_SECRET, { expiresIn: '30d' }). Повертає { token, username }.

Процес авторизації: пошук користувача за нікнеймом, порівняння паролів через bcrypt.compare(). При невірному паролі або відсутньому користувачі повертається однакове повідомлення 'Invalid username or password' – запобігає user enumeration атакам. Діаграму послідовності процесу реєстрації зображено на рисунку 3.1.

Проміжне програмне забезпечення JWT-middleware (server/middleware/auth.js) перевіряє заголовок Authorization: Bearer <token>, верифікує токен через jwt.verify() та записує декодований payload у req.user. Якщо токен відсутній або некоректний – повертає 401. Лістинг наведено у Додатку В.

3.1.3 Управління результатами та WebSocket-трансляція

Файл server/routes/scores.js (Додаток Г) реалізує три маршрути для управління лідербордом. GET / – публічний, повертає записи з фільтрацією за mode та difficulty, відсортовані за score DESC. POST / – захищений requireAuth, зберігає новий запис, обрізає топ до MAX_ENTRIES=20 та викликає broadcast(). DELETE / – захищений, очищає всю таблицю.

Алгоритм оновлення лідерборду при новому результаті: 1) збереження нового запису через Score.create(); 2) завантаження всіх записів для поточної комбінації mode+difficulty, відсортованих за спаданням score; 3) якщо кількість перевищує 20 – видалення зайвих через Score.deleteMany(); 4) виклик broadcast() з оновленим списком топ-20.

Функція broadcast() (server/ws.js, Додаток Д) серіалізує об’єкт у JSON та надсилає усім клієнтам з readyState === WebSocket.OPEN. Перевірка readyState є обов’язковою: без неї спроба надіслати повідомлення закритому з’єднанню викине виключення.

3.1.4 Ендпоінт профілю гравця

GET /api/profile/:username виконує три паралельні запити до MongoDB: User.findOne() для дати реєстрації, Session.find() для повної історії та Score.find() для кращих результатів. На основі отриманих даних обчислюється агрегована статистика: totalGames, totalHits, totalMisses, avgAccuracy, bestScore, bestStreak, totalPlayTimeMs. Формуються масиви для графіків: accuracyHistory та scoreHistory (останні 20 значень). Визначаються кращі результати по кожному з 4 режимів (bestPerMode). Повертаються останні 10 сесій у зворотньому хронологічному порядку. Повний код ендпоінту наведено у Додатку Г.

3.2 Реалізація фронтенду

3.2.1 HTTP-клієнт та управління токеном

Модуль src/api/client.js реалізує єдину функцію apiFetch(url, options), що використовується для всіх API-запитів. Функція автоматично читає JWT-токен з localStorage та додає заголовок Authorization: Bearer до кожного запиту. При HTTP-помилці функція намагається розпарсити тіло відповіді як JSON та викидає Error з повідомленням з поля error – це дозволяє компонентам обробляти бекендові помилки у catch-блоках.

Хук usePlayerProfile зберігає стан авторизації: токен (aimlab_token) та нікнейм (aimlab_player_nick) у localStorage. При завантаженні сторінки хук перевіряє наявність обох значень та відновлює сесію без повторної авторизації. Функція loginUser({ token, username }) зберігає дані та оновлює стан бібліотеки. logout() видаляє дані з localStorage.

3.2.2 Реалізація WebSocket у клієнті

Хук useLeaderboard (Додаток Е) управляє як HTTP-завантаженням лідерборду, так і WebSocket-з'єднанням. При монтуванні LeaderboardScreen хук виконує початкове завантаження через getLeaderboard() (HTTP GET /api/scores) та встановлює WebSocket-з'єднання.

Важливий нюанс реалізації – використання useRef для зберігання поточних значень mode та difficulty замість прямого читання зі стану бібліотеки. Це вирішує проблему closure у WS-обробнику: обробник ws.onmessage замикається (capture) на значення mode та difficulty у момент створення WebSocket-об'єкта [14]. Без useRef при зміні фільтрів лідерборду обробник продовжував би порівнювати з первинними значеннями. useRef оновлюється без перерендеру та завжди містить актуальне значення.

При отриманні повідомлення scores_updated перевіряються data.mode та data.difficulty через modeRef.current та diffRef.current. Якщо збігаються – entries оновлюються напряму без додаткового HTTP-запиту. Стан connected відображається як індикатор LIVE на сторінці лідерборду.

3.2.3 Реалізація 3D-сцени

ThreeScene.jsx – центральний компонент застосунку, що ініціалізує Three.js при монтуванні та очищає ресурси при демонтуванні. Очищення є обов'язковим: WebGL-контексти, геометрії та матеріали займають пам'ять GPU і не звільняються автоматично збирачем сміття JavaScript.

Ініціалізація сцени: WebGLRenderer з антиаліасингом та тінями; PerspectiveCamera (FOV=75°); AmbientLight та PointLight для освітлення; підлога (PlaneGeometry 40×40) та стіни з MeshStandardMaterial. PointerLockControls захоплює курсор та перетворює рухи миші на обертання камери з налаштованою чутливістю.

Виявлення влучань реалізовано через Raycaster: при кліку кидається промінь від камери через центр екрана (вектор (0,0,–1) у просторі камери). Якщо промінь перетинає mesh цілі – ціль знищується, нараховуються очки,

відтворюється звук та запускається анімація вибуху (ExplosionEffect з частинками).

Рисунок 3.2 – Режими гри та рівні складності

3.2.4 Режими гри

Режим Polygon генерує сферичні цілі (SphereGeometry) у випадкових позиціях у межах ігрового поля. Радіус цілей залежить від складності: Easy=0.5м, Normal=0.3м, Hard=0.2м. Кількість одночасних цілей: Easy=3, Normal=2, Hard=1. При влученні ціль зникає та з’являється нова на випадковій позиції. Режим тренує базову точність та координацію.

Режим Grid створює сітку цілей NxM, де N та M залежать від складності. При влученні ціль видаляється; коли всі вражені – сітка оновлюється. Тренує систематичні рухи між передбачуваними позиціями.

Режим Flick реалізує цілі з обмеженим часом існування: Easy=1.5с, Normal=1.0с, Hard=0.6с. Якщо гравець не встигає влучити – ціль зникає без очків. Тренує швидкість реакції та точність різких рухів.

Режим Tracking генерує ціль, що переміщується по сцені через інтерполяцію між випадковими цільовими позиціями. Очки нараховуються покадрово за наявністю прицілу на цілі (Raycaster без кліку). Тренує утримання прицілу на рухомому об’єкті.

3.2.5 Профіль гравця та лідерборд

ProfileScreen.jsx отримує username через useParams() (маршрутизатор бібліотеки v7) та завантажує профіль через apiFetch('/api/profile/:username'). Відображає: аватар (перша літера нікнейму), 8 статистичних карток у сітці 4×2, два лінійних графіки (accuracy та score тренди за останні 20 сесій), картки кращих результатів по кожному режиму, таблицю останніх 10 сесій.

LeaderboardScreen показує два ряди кнопок-фільтрів: режим (4 кнопки) та складність (3 кнопки). При зміні фільтра хук useLeaderboard перезавантажує дані через HTTP, а WebSocket продовжує слухати оновлення для нових фільтрів (через оновлення modeRef та diffRef). Нікнейми у таблиці є клікабельними – перехід на /profile/:username.

3.3 Конфігурація проекту та середовище розробки

Файл vite.config.js налаштовує два проксі-правила: /api → http://localhost:3001 (HTTP) та /ws → ws://localhost:3001 (WebSocket, з параметром ws:true). Це дозволяє клієнту звертатись до відносних URL (/api/scores), не вказуючи явно хост бекенду. У продакшн-середовищі проксі налаштовується на рівні веб-сервера (Nginx).

Файл server/.env зберігає конфіденційні параметри: MONGO_URI (рядок підключення до MongoDB Atlas), PORT (порт сервера, за замовчуванням 3001), JWT_SECRET (секрет для підпису токенів). Цей файл виключений з системи контролю версій (.gitignore).

Команди запуску: npm run dev (клієнт, порт 5173), cd server && npm run dev (сервер, nodemon, порт 3001). Nodemon автоматично перезапускає сервер при зміні файлів з розширеннями .js, .json.

Vite dev-server виступає зворотним проксі між клієнтом та бекендом під час розробки. Без проксі клієнт на порту 5173 не міг би звертатись до API на порту 3001 через Same-Origin Policy браузера – відповідь містила б CORS-помилку. Конфігурація proxy в vite.config.js перенаправляє запити /api/* на HTTP-сервер та /ws/* на WebSocket-сервер без змін у клієнтському коді. При

збірці для продакшн проксі вимикається: продакшн-конфігурація налаштовується окремо на рівні Nginx або хмарної платформи із аналогічними proxy-правилами.

Конфігурація серверного середовища управляється через бібліотеку dotenv. Змінні середовища читаються з server/.env при локальній розробці та з системних змінних при розгортанні на хмарних платформах, де .env-файл відсутній. Такий підхід відповідає принципам twelve-factor app: конфігурація зберігається у середовищі, а не в коді. MONGO_URI містить облікові дані MongoDB Atlas та назву бази даних, JWT_SECRET – секретний рядок достатньої довжини для підпису токенів; для продакшн рекомендується генерація випадкового секрету криптографічним інструментом.

Файл package.json клієнтської частини визначає три npm-команди: dev (запуск Vite dev-сервера), build (продакшн-збірка в директорію dist/) та preview (локальний запуск збірки для перевірки). Файл server/package.json визначає start (node index.js для продакшн) та dev (nodemon index.js для розробки з автоперезапуском). Nodemon налаштований на перегляд лише файлів JavaScript та JSON. Одночасний запуск обох частин може бути автоматизований засобами паралельного запуску процесів, що усуває необхідність двох окремих вікон терміналу.

3.4 Аналіз ключових рішень реалізації

3.4.1 Управління пам’яттю Three.js

Інтеграція Three.js у компонент вимагає особливої уваги до управління ресурсами. Компоненти монтуються та демонтуються при навігації, але WebGL-ресурси (геометрії, матеріали, текстури, render targets) не звільняються автоматично збирачем сміття JavaScript, оскільки вони зберігаються у пам’яті GPU.

У ThreeScene.jsx реалізований повний cleanup при демонтуванні: renderer.dispose() звільняє WebGL-контекст; controls.dispose() знімає обробники подій миші та клавіатури; geometry.dispose() та material.dispose() для кожного

об’єкта сцени; composer.dispose() для EffectComposer. Цикл по scene.children з рекурсивним обходом гарантує звільнення всіх вкладених об’єктів. Без cleanup повторне відкриття ігрового екрану поступово вичерпувало б пам’ять GPU.

3.4.2 Обробка асинхронності та стану

Хуки useLeaderboard та usePlayerProfile активно використовують useEffect для управління асинхронними операціями. Важливий нюанс: cleanup-функція useEffect викликається при демонтуванні або перед повторним виконанням ефекту. При асинхронному запиті (fetch) існує ризик race condition: якщо компонент демонтується до завершення запиту, виклик setEntries() на демонтованому компоненті викликає попередження бібліотеки.

Для вирішення цієї проблеми в хуку usePlayerProfile використовується прапор isMounted: const isMounted = { current: true }; у cleanup: isMounted.current = false;. Після await apiFetch() перевіряється if (!isMounted.current) return; перед будь-яким setState. Це стандартний патерн для безпечних async операцій у React [1].

3.4.3 Оптимізація рендерингу

Ігровий HUD (Timer, Score, Streak) оновлюється кожного кадру – потенційно 60+ разів на секунду. Наївна реалізація з useState та setInterval викликала б масу перерендерів дерева компонентів і могла б знизити FPS. Вирішення: useRef для зберігання актуальних значень під час гри, оновлення DOM напряму через ref.current.textContent (для таймера та очків) без участі рендерингу бібліотеки. Бібліотека залучається лише для суттєвих змін стану (пауза, кінець гри).

Ця техніка – imperative escape hatch – задокументована в офіційній документації React [1] для випадків, коли частота оновлень перевищує можливості декларативного рендерингу. Вона дозволяє поєднати бібліотеку для управління структурою UI та прямі DOM-маніпуляції для high-frequency оновлень.

3.4.4 Безпека токена та захист маршрутів

Зберігання JWT у localStorage є прийнятним підходом для некритичних застосунків, але має обмеження: токен доступний JavaScript-коду сторінки, що робить його вразливим до XSS-атак. У AimPractice ризик мінімізовано відсутністю innerHTML з користувацьким введенням та автоматичним екрануванням вмісту JSX-виразів.

Захист маршрутів на рівні UI реалізований через умовний рендеринг у App.jsx: якщо playerName відсутній – відображається RegisterNameScreen. Це запобігає доступу до ігрового функціоналу без авторизації на рівні інтерфейсу. Захист на рівні API (requireAuth middleware) забезпечує, що навіть прямі HTTP-запити без токена повертають 401.

Висновки до розділу 3. Реалізовано всі заплановані функціональні можливості системи: 3D-тренажер з чотирма режимами гри та трьома рівнями складності, безпечна система авторизації JWT+bcrypt, інтеграція з MongoDB Atlas, WebSocket-трансляція та публічні профілі. Застосовані техніки оптимізації (imperative DOM-оновлення для HUD, useRef для WebSocket-обробників, повний cleanup WebGL-ресурсів) забезпечують стабільну роботу на рівні 60+ FPS. Модульна структура коду забезпечує легкість підтримки та розширення.

РОЗДІЛ 4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Стратегія та методи тестування

Тестування AimPractice проводилося у чотирьох напрямах: функціональне тестування (перевірка відповідності реалізованих функцій специфікації вимог), тестування продуктивності (вимірювання FPS, часу відповіді API та затримки WebSocket), тестування безпеки (перевірка захисту API та стійкості системи авторизації) та тестування сумісності браузерів.

Метод тестування – ручне тестування за таблицями тест-кейсів. Кожен тест-кейс включає ідентифікатор, назву, передумови, кроки відтворення, очікуваний та фактичний результат, статус. Тестування проводилось на двох конфігураціях обладнання та у браузерах Chrome 124, Firefox 125 та Edge 124.

Тестове середовище: операційна система Windows 11 x64; браузери Chrome 124.0, Firefox 125.0, Edge 124.0; мережа – localhost (Dev-режим, Vite-проксі); бекенд – Node.js 22 LTS, MongoDB Atlas кластер EU-West.

Функціональне тестування виконувалось методом black-box: тестувальник взаємодіє із застосунком через інтерфейс, не маючи знань про внутрішню реалізацію. Тест-кейси розроблені на основі функціональних вимог та охоплюють позитивні сценарії (happy path) та негативні сценарії (перевірка обробки помилок). Кожен тест-кейс має чіткі передумови, кроки відтворення та критерій успіху.

Тестування продуктивності проводилось методом вимірювання: показники FPS знімались з вбудованого лічильника браузера після стабілізації рендерингу. Час відповіді API вимірювався через вкладку Network у DevTools, яка показує загальний час від відправки запиту до отримання останнього байту відповіді. Затримка WebSocket вимірювалась через різницю часових міток між надсиланням нового результату та отриманням широкомовного повідомлення у другому браузері.

Тестування безпеки виконувалось методом gray-box: тестувальник має часткові знання про реалізацію (алгоритм JWT, бібліотека bcrypt, структура API) та використовує їх для цілеспрямованих перевірок. Серед інструментів – ручні

HTTP-запити до API з підробленими заголовками та декодування токенів без перевірки підпису. Мета: підтвердити, що система правильно відхиляє несанкціоновані запити та не розкриває чутливу інформацію.

4.2 Функціональне тестування

Таблиця 4.1 – Тест-кейси функціонального тестування

ID	Функція	Очікуваний результат	Статус
TC-01	Реєстрація: новий користувач	Токен, перехід до меню	✓ Пройдено
TC-02	Реєстрація: дублікат нікнейму	Помилка 409, повідомлення	✓ Пройдено
TC-03	Реєстрація: пароль < 4 символів	Помилка 400, повідомлення	✓ Пройдено
TC-04	Авторизація: коректні дані	Токен, перехід до меню	✓ Пройдено
TC-05	Авторизація: невірний пароль	Помилка 401, загальне повідомлення	✓ Пройдено
TC-06	Logout	Очищення токена, екран входу	✓ Пройдено
TC-07	Гра Polygon Normal 60с	60с гра, результат збережено в БД	✓ Пройдено
TC-08	Гра Flick Hard	Цілі зникають через 0.6с	✓ Пройдено
TC-09	Гра Grid Easy	Сітка оновлюється після повного влучання	✓ Пройдено
TC-10	Гра Tracking Normal	Ціль рухається, очки за утримання	✓ Пройдено
TC-11	Лідерборд: фільтр mode+difficulty	Показано лише відповідні записи	✓ Пройдено
TC-12	WebSocket: оновлення при новому рекорді	2 браузери оновились < 1с	✓ Пройдено
TC-13	Профіль: GET /profile/:username	Статистика, графіки, кращі результати	✓ Пройдено
TC-14	API без токена: POST /api/scores	401 Unauthorized	✓ Пройдено
TC-15	Налаштування прицілу	Зміни відображені в реальному часі	✓ Пройдено
TC-16	Відновлення сесії після перезавантаження	Авторизація збережена з localStorage	✓ Пройдено

Усі 16 функціональних тест-кейсів успішно пройдені. Критичних дефектів не виявлено.

4.3 Тестування продуктивності

Тестування продуктивності проводилось на двох конфігураціях обладнання:

– Конфігурація А (ігровий ПК): Intel Core i5-12400F (6 ядер/12 потоків, 4.4 ГГц Boost), NVIDIA GeForce GTX 1660 Super (6 ГБ GDDR6), 16 ГБ DDR4-3200.
– Конфігурація В (ноутбук середнього класу): Intel Core i5-1135G7 (4 ядра/8 потоків, 4.2 ГГц Boost), Intel Iris Xe Graphics (вбудована, 80 EU), 8 ГБ DDR4-3200.

Таблиця 4.2 – Результати тестування продуктивності

Метрика	Конфіг. А (GTX 1660)	Конфіг. В (Iris Xe)	Норма
FPS – Polygon Normal	144 FPS (ліміт монітора)	67 FPS	≥ 60 FPS
FPS – Grid Hard (макс. цілей)	144 FPS	58 FPS	≥ 60 FPS
FPS – Tracking Normal	144 FPS	63 FPS	≥ 60 FPS
FPS – Flick Hard	144 FPS	71 FPS	≥ 60 FPS
GET /api/scores (1-й запит)	195 мс	198 мс	< 500 мс
GET /api/scores (кешований)	22 мс	24 мс	< 500 мс
POST /api/scores + broadcast	245 мс	249 мс	< 500 мс
GET /api/profile/:username	218 мс	221 мс	< 500 мс
POST /api/auth/login	195 мс	198 мс	< 500 мс
WebSocket затримка broadcast	44 мс	47 мс	< 100 мс
Розмір JS-бандлу (Vite build)	1.82 МБ	1.82 МБ	—
Час першого завантаження	2.2 с	2.4 с	< 5 с

Аналіз результатів: на конфігурації А всі показники значно перевищують норми. На конфігурації В режим Grid Hard показав 58 FPS – незначно нижче норми через велику кількість mesh-об’єктів та обмеження інтегрованої графіки. Це не є критичним дефектом: в умовах реального використання гравці з

бюджетним обладнанням можуть обрати Normal-складність. Потенційне покращення: InstancedMesh для режиму Grid.

Час відповіді API (195–249 мс) включає мережеву затримку до MongoDB Atlas (кластер EU-West) та час обробки запиту. Затримка WebSocket (44–47 мс) є відмінним показником для real-time застосунків.

4.4 Тестування безпеки

Тестування безпеки охоплює чотири вектори атак.

Вектор 1 – неавторизований запит. POST /api/scores без заголовка Authorization повернув 401 {"error":"Unauthorized"}. Middleware requireAuth коректно відхиляє запити без токена.

Вектор 2 – підроблений JWT-токен. Декодування токена з допомогою jwt-decode (без ключа), зміна поля username у payload та повторне кодування – без знання JWT_SECRET підпис стане некоректним. Сервер повернув 401 {"error":"Invalid or expired token"}. jwt.verify() виявляє порушення підпису.

Вектор 3 – прострочений токен. Після закінчення 30-денного терміну jwt.verify() кидає TokenExpiredError, middleware повертає 401. На клієнті це призводить до автоматичного виходу при наступному запиті.

Вектор 4 – стійкість паролів. При cost=10 bcrypt витрачає ~100 мс на одне хешування [7]. Теоретичний час перебору 10^6 паролів: 100 000 секунд ≈ 28 годин на одному CPU. Сучасні GPU-ферми сповільнюються bcrypt у 100–1000 разів менше ніж для SHA-256, оскільки bcrypt є навмисно memory-hard алгоритмом.

Вектор 5 – user enumeration. При авторизації з неіснуючим нікнеймом та при невірному паролі повертається однакове повідомлення 'Invalid username or password' – успішно запобігає визначенню існуючих акаунтів.

Таблиця 4.3 – Результати тестування безпеки

Вектор атаки	Очікуваний захист	Результат
POST /api/scores без токена	401 Unauthorized	✓ Захищено
Підроблений JWT payload	401 Invalid token	✓ Захищено
Прострочений JWT	401 Invalid token	✓ Захищено
Brute force паролів	~28 годин на 1M паролів	✓ Захищено
User enumeration при login	Однакове повідомлення помилки	✓ Захищено

4.5 Тестування сумісності браузерів

Таблиця 4.4 – Результати тестування сумісності браузерів

Функція	Chrome 124	Firefox 125	Edge 124	Safari 17
3D-рендеринг (WebGL 2.0)	✓	✓	✓	✓
Pointer Lock API	✓	✓	✓	Часткова
WebSocket	✓	✓	✓	✓
JWT у localStorage	✓	✓	✓	✓
React 19 Concurrent	✓	✓	✓	✓
EffectComposer (bloom)	✓	✓	✓	✓
React Router useParams	✓	✓	✓	✓

Застосунок коректно функціонує у Chrome, Firefox та Edge. Safari 17 має відому обмеженість Pointer Lock API: повторне захоплення курсору після виходу з гри може не спрацювати без явної взаємодії з DOM. Це не є критичним для основної аудиторії (переважно Windows/Linux з Chrome).

Safari 17 має обмеження Pointer Lock API з міркувань безпеки WebKit: після виходу з повноекранного режиму або перемикання вкладки повторне захоплення курсору вимагає явного кліку користувача. AimPractice адаптований до цього: захоплення курсору викликається у відповідь на подію кліку, а не програмно. Мобільні браузери не тестувались, оскільки Pointer Lock API недоступний на мобільних платформах, а 3D-тренажер потребує точного маніпулятора (мишки).

WebGL 2.0 підтримується всіма тестованими браузерами. Three.js автоматично обирає WebGL 2.0 при наявності підтримки. EffectComposer із bloom-ефектом вимагає підтримки float-текстур, доступних у WebGL 2.0. Тестування підтвердило коректний рендеринг bloom у всіх браузерах. Єдина візуальна відмінність – Safari 17 рендерить bloom із незначно нижчою яскравістю через особливості обробки кольорового простору у WebKit, що не впливає на ігровий процес.

Тестування сумісності проводилось за єдиним сценарієм для кожного браузера: реєстрація нового гравця, авторизація, повний ігровий цикл (вибір режиму, гра 60 секунд, екран результату), перегляд лідерборду з перевіркою індикатора LIVE та перехід на профіль гравця з перевіркою графіків. Chrome, Firefox та Edge тестувались на Windows 11, Safari – на macOS. Несумісностей, що перешкоджають роботі застосунку, не виявлено.

4.6 Висновки за результатами тестування

Комплексне тестування підтвердило відповідність системи AimPractice всім визначеним функціональним та нефункціональним вимогам. Усі 16 функціональних тест-кейсів успішно пройдені. Показники продуктивності відповідають або перевищують встановлені норми на обох тестових конфігураціях. Єдиний граничний показник – 58 FPS у Grid Hard на інтегрованій графіці – не є критичним дефектом.

Тестування безпеки підтвердило надійність системи авторизації: JWT-middleware захищає всі маршрути запису, bcrypt забезпечує практичну стійкість до brute force атак, система не вразлива до user enumeration. Функціональність протестована у трьох основних браузерах без виявлення несумісностей.

Кількісні підсумки тестування: усі функціональні тест-кейси пройдено; усі вектори безпеки нейтралізовані; продуктивність перевищує норму на потужній конфігурації за всіма показниками; на конфігурації з інтегрованою графікою єдиний показник Grid Hard незначно нижче норми 60 FPS – граничний результат.

Час відповіді API відповідає вимозі менше 500 мс; затримка WebSocket удвічі краще за встановлену норму 100 мс.

Виявлені обмеження не є критичними для поточної версії. Граничний FPS у режимі Grid Hard на вбудованій графіці – передбачуваний результат для 3D-застосунків із великою кількістю draw calls. Потенційне рішення – об'єднання цілей через InstancedMesh – визначено як задачу для наступної версії. Обмеження Pointer Lock у Safari стосується незначної частини цільової аудиторії. Обидва обмеження є архітектурними особливостями платформ, а не помилками реалізації.

Оцінка повноти тестування: функціональне тестування охоплює всі функціональні вимоги; тестування безпеки охоплює ключові вектори для застосунків з JWT-авторизацією та REST API; тестування продуктивності охоплює всі ігрові режими та основні ендпоінти API; тестування сумісності охоплює основні браузери цільової аудиторії. Поза межами поточного тестування залишились навантажувальне тестування максимальної пропускної здатності сервера та автоматизоване регресійне тестування – ці напрямки визначені для подальшого розвитку.

4.7 Аналіз якості коду та метрики проекту

Аналіз структури проекту показав наступні кількісні характеристики. Клієнтська частина: 12 компонентів-екранів, 4 хуки, 2 модулі storage, 2 API-модулі, 5 Three.js-модулів. Серверна частина: 4 маршрутних файли, 3 Mongoose-схеми, 1 middleware, 1 ws-модуль, 1 точка входу. Загальний обсяг коду: близько 2500 рядків JavaScript (без урахування package.json та конфігурацій).

Структура коду відповідає принципам чистого коду [20]: кожен файл має єдину відповідальність (Single Responsibility Principle); компоненти та хуки є слабозв'язаними; хуки інкапсулюють логіку, не залежну від конкретних компонентів; серверні маршрути містять лише обробку HTTP-запитів, бізнес-логіка (агрегація статистики) винесена до окремих функцій.

Таблиця 4.5 – Метрики проекту AimPractice

Метрика	Значення	Коментар
Клієнтських JS/JSX файлів	27	Компоненти, хуки, storage, api
Серверних JS файлів	10	Маршрути, схеми, middleware, ws
npm залежностей (клієнт)	8	react, three, react-router-dom та ін.
npm залежностей (сервер)	7	express, mongoose, bcrypt, jwt, ws та ін.
REST API маршрутів	9	4 групи: auth, scores, sessions, profile
Mongoose-схем	3	User, Score, Session
React-компонентів	12	8 екранів + 4 допоміжних
React-хуків (custom)	4	usePlayerProfile, useGameSettings, useLeaderboard, useGameLoop
3D режимів гри	4	Polygon, Grid, Flick, Tracking
Рівнів складності	3	Easy, Normal, Hard

Розподіл відповідальностей між клієнтом та сервером відповідає принципу «fat server, thin client»: сервер виконує всі операції з базою даних та розраховує агреговану статистику, клієнт лише відображає отримані дані. Це полегшує майбутнє масштабування: клієнтська частина може бути повністю замінена (наприклад, мобільний додаток) без змін серверного API.

Висновки до розділу 4. Проведене комплексне тестування системи AimPractice підтвердило відповідність вимогам та готовність до практичного використання. Виявлені незначні обмеження пов'язані з апаратним забезпеченням та відомими особливостями WebKit, а не з помилками реалізації.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено повнофункціональний вебзастосунок AimPractice для тренування швидкості реакції та точності користувача. Досягнуто всі поставлені мету та завдання.

1. Проведено аналіз предметної області, що виявив зростаючий попит на інструменти тренування прицілювання серед гравців у шутери від першої особи [16]. Порівняльний аналіз чотирьох існуючих рішень (Aim Lab, KovaaK's, 3D Aim Trainer, Aimlabs Web) показав відсутність вільного вебзастосунку з системою авторизації, глобальним лідербордом у реальному часі та публічними профілями гравців.

2. Обраний та обґрунтований технологічний стек (React 19, Three.js 0.182, Vite 7, Node.js, Express.js, MongoDB Atlas, JWT, bcrypt, WebSocket) відповідає вимогам продуктивності, безпеки та масштабованості системи.

3. Спроектована тришарова архітектура системи (клієнтський, серверний та шар даних) з чітким розподілом відповідальностей. Схема бази даних з трьома колекціями (users, scores, sessions) та REST API з чотирма групами маршрутів забезпечують ефективне зберігання та доступ до даних.

4. Реалізовано 3D-тренажер на Three.js з чотирма режимами (Polygon, Grid, Flick, Tracking) та трьома рівнями складності. Якість рендерингу підтримується антиаліасингом та bloom-ефектом. Частота кадрів – 60–144 FPS залежно від обладнання.

5. Реалізовано безпечну систему авторизації: bcrypt (cost=10) для хешування паролів [7], JWT (30 днів) для stateless-сесій [6], middleware requireAuth для захисту API. Система успішно витримала тестування безпеки за п'ятьма векторами атак.

6. Інтеграція MongoDB Atlas через Mongoose [9] забезпечила хмарне зберігання з автоматичними резервними копіями. WebSocket-трансляція через бібліотеку ws забезпечує оновлення лідерборду із затримкою 44–47 мс.

7. Публічні профілі гравців (/profile/:username) надають соціальний аспект застосунку з розширеною статистикою, графіками прогресу та кращими результатами по режимах.

8. Комплексне тестування (16 функціональних тест-кейсів, вимірювання продуктивності, тестування безпеки та сумісності) підтвердило відповідність системи всім вимогам.

Практична цінність: готовий до використання вебзастосунок без вимог до встановлення. Доступний з будь-якого пристрою з браузером Chrome 90+, Firefox 88+, Edge 90+.

Перспективи подальшого розвитку: система досягнень (achievements) для мотивації гравців; теплові карти влучань для детального аналізу помилок; підтримка мобільних пристроїв; функція друзів та соціального порівняння; оптимізація Grid через InstancedMesh; розгортання на хмарній платформі (Vercel + Railway) для загальнодоступності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Documentation. Офіційна документація React 19. Meta Platforms, Inc., 2024. URL: https://react.dev (дата звернення: 10.05.2025).
2. Three.js Documentation. Офіційна документація Three.js r182. URL: https://threejs.org/docs (дата звернення: 10.05.2025).
3. MongoDB Documentation. Офіційна документація MongoDB 7.0. MongoDB, Inc., 2024. URL: https://www.mongodb.com/docs (дата звернення: 12.05.2025).
4. Express.js Guide. Офіційний посібник Express.js 4.x. URL: https://expressjs.com/en/guide (дата звернення: 12.05.2025).
5. Fette I., Melnikov A. The WebSocket Protocol. RFC 6455. IETF, 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 14.05.2025).
6. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 14.05.2025).
7. Provos N., Mazieres D. A Future-Adaptable Password Scheme. USENIX Annual Technical Conference, 1999. URL: https://www.usenix.org/legacy/events/usenix99/provos.html (дата звернення: 15.05.2025).
8. Vite Documentation. Офіційна документація Vite 7. URL: https://vite.dev (дата звернення: 15.05.2025).
9. Mongoose Documentation. Офіційна документація Mongoose 8.x. URL: https://mongoosejs.com/docs (дата звернення: 15.05.2025).
10. React Router Documentation. Офіційна документація React Router v7. URL: https://reactrouter.com (дата звернення: 16.05.2025).
11. Parisi T. WebGL: Up and Running. Building 3D Graphics for the Web. O'Reilly Media, 2012. 194 с. ISBN 978-1-449-32357-8.
12. Banks A., Porcello E. Learning React. Modern Patterns for Developing React Apps. 2nd ed. O'Reilly Media, 2020. 310 с. ISBN 978-1-492-05172-5.

13. Chodorow K. MongoDB: The Definitive Guide. 3rd ed. O'Reilly Media, 2019. 514 с. ISBN 978-1-491-95446-1.
14. Flanagan D. JavaScript: The Definitive Guide. 7th ed. O'Reilly Media, 2020. 706 с. ISBN 978-1-491-95202-3.
15. OWASP Foundation. OWASP Top 10 Web Application Security Risks. 2021. URL: https://owasp.org/Top10 (дата звернення: 18.05.2025).
16. Newzoo. Global Esports & Live Streaming Market Report 2024. Newzoo BV, 2024. URL: https://newzoo.com/resources/trend-reports/newzoos-global-esports-live-streaming-market-report-2024-free-version (дата звернення: 05.05.2025).
17. Schmidt R. A., Lee T. Motor Learning and Performance. 5th ed. Human Kinetics, 2011. 392 с. ISBN 978-1-4504-0082-2.
18. Ericsson K. A., Krampe R. T., Tesch-Römer C. The Role of Deliberate Practice in the Acquisition of Expert Performance. Psychological Review, 1993. Vol. 100, № 3. P. 363–406.
19. Fitts P. M. The information capacity of the human motor system in controlling the amplitude of movement. Journal of Experimental Psychology, 1954. Vol. 47, № 6. P. 381–391.
20. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Boston: Prentice Hall, 2008. 464 p. ISBN 978-0-13-235088-4.
21. Aim Lab: AI-powered aim trainer / State Space Labs, Inc. URL: https://aimlabs.com (дата звернення: 05.05.2025).
22. KovaaK’s FPS Aim Trainer / The Meta, Inc. URL: https://www.kovaaks.com (дата звернення: 05.05.2025).

ДОДАТОК А ЛІСТИНГ SERVER/INDEX.JS – ТОЧКА ВХОДУ СЕРВЕРУ

```
import { createServer } from "http";
import express from "express";
import cors from "cors";
import mongoose from "mongoose";
import { config } from "dotenv";
import scoresRouter from "./routes/scores.js";
import sessionsRouter from "./routes/sessions.js";
import authRouter from "./routes/auth.js";
import profileRouter from "./routes/profile.js";
import { setupWebSocket } from "./ws.js";

config();

const app = express();
app.use(cors());
app.use(express.json());

app.use("/api/auth",     authRouter);
app.use("/api/scores",   scoresRouter);
app.use("/api/sessions", sessionsRouter);
app.use("/api/profile",  profileRouter);

const PORT     = process.env.PORT || 3001;
const MONGO_URI = process.env.MONGO_URI;

if (!MONGO_URI) {
  console.error("MONGO_URI is not set in .env");
  process.exit(1);
}

const httpServer = createServer(app);
setupWebSocket(httpServer);

mongoose
  .connect(MONGO_URI)
  .then(() => {
    console.log("MongoDB connected");
    httpServer.listen(PORT, () =>
      console.log(`Server running on http://localhost:${PORT}`)
    );
  })
  .catch((err) => {
    console.error("MongoDB connection error:", err);
    process.exit(1);
  });
```

ДОДАТОК Б ЛІСТИНГ SERVER/ROUTES/AUTH.JS – МАРШРУТИ АВТОРИЗАЦІЇ

```
import { Router } from "express";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import User from "../models/User.js";

const router = Router();

router.post("/register", async (req, res) => {
  try {
    const { username, password } = req.body;
    if (!username || !password)
      return res.status(400).json({ error: "Username and password required"
});
    if (password.length < 4)
      return res.status(400).json({ error: "Password must be at least 4
characters" });

    const existing = await User.findOne({ username });
    if (existing)
      return res.status(409).json({ error: "Username already taken" });

    const hash = await bcrypt.hash(password, 10);
    await User.create({ username, password: hash });

    const token = jwt.sign(
      { username },
      process.env.JWT_SECRET,
      { expiresIn: "30d" }
    );
    res.json({ token, username });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

router.post("/login", async (req, res) => {
  try {
    const { username, password } = req.body;
    const user = await User.findOne({ username });
    if (!user)
      return res.status(401).json({ error: "Invalid username or password"
});

    const ok = await bcrypt.compare(password, user.password);
    if (!ok)
      return res.status(401).json({ error: "Invalid username or password"
});

    const token = jwt.sign(
      { username },
      process.env.JWT_SECRET,
      { expiresIn: "30d" }
    );
    res.json({ token, username });
  } catch (err) {
    res.status(500).json({ error: err.message });
```

```
  }
});

export default router;
```

ДОДАТОК В ЛІСТИНГ SERVER/MIDDLEWARE/AUTH.JS – JWT-MIDDLEWARE

```
import jwt from "jsonwebtoken";

export function requireAuth(req, res, next) {
  const header = req.headers.authorization;
  if (!header || !header.startsWith("Bearer ")) {
    return res.status(401).json({ error: "Unauthorized" });
  }
  const token = header.slice(7);
  try {
    req.user = jwt.verify(token, process.env.JWT_SECRET);
    next();
  } catch {
    res.status(401).json({ error: "Invalid or expired token" });
  }
}
```

ДОДАТОК Г ЛІСТИНГ SERVER/ROUTES/SCORES.JS – МАРШРУТИ ЛІДЕРБОРДУ

```
import { Router } from "express";
import Score from "../models/Score.js";
import { requireAuth } from "../middleware/auth.js";
import { broadcast } from "../ws.js";

const router = Router();
const MAX_ENTRIES = 20;

router.get("/", async (req, res) => {
  try {
    const { mode, difficulty } = req.query;
    const filter = {};
    if (mode)       filter.mode = mode;
    if (difficulty) filter.difficulty = difficulty;
    const entries = await Score.find(filter)
      .sort({ score: -1, accuracy: -1, timeMs: 1 })
      .lean();
    res.json(entries);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

router.post("/", requireAuth, async (req, res) => {
  try {
    const entry = { ...req.body, createdAt: req.body.createdAt || Date.now()
};
    await Score.create(entry);

    const mode       = entry.mode       || "polygon";
    const difficulty = entry.difficulty || "normal";

    // Зберігаємо тільки топ MAX_ENTRIES для mode+difficulty
    const group = await Score.find({ mode, difficulty })
      .sort({ score: -1, accuracy: -1, timeMs: 1 })
      .lean();

    if (group.length > MAX_ENTRIES) {
      const toDelete = group.slice(MAX_ENTRIES).map(e => e._id);
      await Score.deleteMany({  id: { $in: toDelete } });
    }

    const updated = group.slice(0, MAX_ENTRIES);
    broadcast({ type: "scores_updated", mode, difficulty, entries: updated
});

    res.json({ ok: true });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

router.delete("/", requireAuth, async (req, res) => {
  try {
    await Score.deleteMany({});
    res.json({ ok: true });
```

```
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

export default router;
```

ДОДАТОК Д ЛІСТИНГ SERVER/WS.JS ТА SRC/HOOKS/USELEADERBOARD.JS

server/ws.js – WebSocket-сервер:

```
import { WebSocketServer } from "ws";

let wss;

export function setupWebSocket(httpServer) {
  wss = new WebSocketServer({ server: httpServer, path: "/ws" });
  wss.on("connection", (ws) => {
    ws.send(JSON.stringify({ type: "connected" }));
  });
}

export function broadcast(data) {
  if (!wss) return;
  const msg = JSON.stringify(data);
  wss.clients.forEach((client) => {
    if (client.readyState === 1) client.send(msg);
  });
}
```

src/hooks/useLeaderboard.js – React-хук WebSocket-лідерборду:

```
import { useState, useCallback, useEffect, useRef } from "react";
import { getLeaderboard, addScore, resetLeaderboard }
  from "../storage/leaderboardStorage";

export function useLeaderboard({ mode, difficulty } = {}) {
  const [entries,   setEntries]   = useState([]);
  const [loading,   setLoading]   = useState(true);
  const [connected, setConnected] = useState(false);
  const modeRef = useRef(mode);
  const diffRef = useRef(difficulty);

  useEffect(() => { modeRef.current = mode; },       [mode]);
  useEffect(() => { diffRef.current = difficulty; }, [difficulty]);

  const load = useCallback(async () => {
    setLoading(true);
    try {
      const data = await getLeaderboard({ mode, difficulty });
      setEntries(data);
    } catch { setEntries([]); }
    finally  { setLoading(false); }
  }, [mode, difficulty]);

  useEffect(() => { load(); }, [load]);

  // WebSocket — оновлення в реальному часі
  useEffect(() => {
    const wsUrl = `ws://${window.location.host}/ws`;
    const ws    = new WebSocket(wsUrl);
```

```
    ws.onopen  = () => setConnected(true);
    ws.onclose = () => setConnected(false);
    ws.onerror = () => setConnected(false);

    ws.onmessage = (e) => {
      try {
        const data = JSON.parse(e.data);
        if (
          data.type       === "scores_updated" &&
          data.mode       === modeRef.current  &&
          data.difficulty === diffRef.current
        ) { setEntries(data.entries); }
      } catch {}
    };

    return () => ws.close();
  }, []);

  const submitScore = useCallback(async (entry) => {
    await addScore(entry);
  }, []);

  const reset = useCallback(async () => {
    await resetLeaderboard();
    setEntries([]);
  }, []);

  return { entries, submitScore, refresh: load, reset, loading, connected };
}
```