МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«18» червня 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Конструктор особистих челенджів

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Олег БАБІЄНКО**

«18» червня 2026 р.

Керівник роботи
ст. викладачка

_____ **Світлана БОРОВЛЬОВА**

«18» червня 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____Євген ДАВИДЕНКО
«01» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну бакалаврську роботу здобувача
Бабієнко Олег

1. Тема кваліфікаційної роботи: Розробка вебплатформи «Конструктор особистих челенджів» з інтегрованими модулями відстеження прогресу та гейміфікації, затверджена наказом ректора ЧНУ ім. Петра Могили №349 від «26» грудня 2026 р.

2. Строк представлення кваліфікаційної роботи: «___» _____ 2026 р.

3. Очікуваним результатом є створена веборієнтована інформаційна система для персонального розвитку та мотивації користувачів, яка поєднує функції гнучкого налаштування цілей, челенджів, публічного представлення досягнень у соціальному середовищі та внутрішнього управління індивідуальними планами самовдосконалення.

4. Перелік питань, що підлягають розробці: аналіз предметної області та існуючих аналогів систем управління персональними цілями, проєктування мікросервісної архітектури та структури бази даних вебплатформи, реалізація механізмів динамічного створення челенджів, розробка публічної та закритої частин платформи для різних ролей користувачів, забезпечення захисту персональних даних під час їх передачі та зберігання у базі даних, а також розмежування прав доступу.

5. Перелік графічних матеріалів: Презентація

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «26» грудня 2026 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Конструктор особистих челенджів

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КБР	26.12.2025	26.12.2025	*Виконано*
2.	Розробка та затвердження технічного завдання на розробку SaaS-платформи	26.12.2025	29.12.2025	*Виконано*
3.	Огляд літератури та аналіз існуючих систем управління продуктивністю	02.01.2026	19.01.2026	*Виконано*
4.	Складання календарного плану КРБ	20.01.2026	21.01.2026	*Виконано*
5.	Визначення ролей користувачів, потоків даних та вимог до конструктора челенджів	22.01.2026	10.02.2026	*Виконано*
6.	Проєктування схеми бази даних та вибір технологічного стеку для мікросервісів	11.02.2026	05.03.2026	*Виконано*
7.	Моделювання та конструювання ядра системи: налаштування мікросервісу авторизації та механізму трекінгу прогресу	06.03.2026	25.03.2026	*Виконано*
8.	Кодування функціоналу конструктора, розробка інтерфейсу дашбордів та тестування безпеки	26.03.2026	15.04.2026	*Виконано*
9.	Попередній захист	27.05.2026	27.05.2026	*Виконано*
10.	Завершення оформлення КРБ та презентації	28.05.2026	28.05.2026	*Виконано*
11.	Рецензування	15.06.2026	18.06.2026	*Виконано*
12.	Захист кваліфікаційної роботи	23.06.2026	23.06.2026	

Здобувач _____ **Олег БАБІЄНКО**

«20» березня 2026 р.

Керівник роботи

ст. викладачка

_____ **Світлана БОРОВЛЬОВА**

«20» березня 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Конструктор особистих челенджів»

Здобувач 409 гр.: Бабієнко Олег

Керівник: ст. викладачка Боровльова Світлана

Актуальність роботи зумовлена зростаючою потребою у гнучких інструментах для саморозвитку, мотивації та систематичного досягнення особистих цілей за допомогою цифрових рішень.

Метою роботи є створення веборієнтованої інформаційної системи на базі мікросервісів для оптимізації процесів саморозвитку та автоматизації.

Об’єктом роботи є інформаційні процеси управління особистою ефективністю та механізми формування нових звичок.

Предметом роботи є технологічні методи та програмні інструменти створення масштабованих SaaS-систем для конструювання персоналізованих програм розвитку.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність обраної теми, визначено мету, основні завдання, об'єкт і предмет розробки, а також розкрито практичне значення отриманих результатів.

У першому розділі проведено комплексне дослідження цільової предметної галузі, а також виконано порівняльний огляд наявних на ринку цифрових інструментів для контролю звичок і персональних цілей.

Другий розділ містить вичерпну специфікацію функціональних та експлуатаційних вимог до створюваного продукту й присвячений глибокому архітектурному проєктуванню хмарної вебплатформи. У ньому розкрито етапи розробки реляційної моделі збереження даних, а також моделювання підсистеми ігрових заохочень. Окрему увагу приділено алгоритмізації процесів моніторингу успішності та проєктуванню багаторівневої ієрархії доступів, яка чітко розмежовує повноваження звичайних учасників, організаторів викликів і системних

адміністраторів. Поряд із цим наведено аргументоване технічне обґрунтування затвердженого стеку розробки, ядром якого виступають бібліотека React та мікрофреймворк FastAPI у поєднанні з реляційною системою PostgreSQL.

У третьому розділі описано процес програмної реалізації вебплатформи, створення клієнт-серверного застосунку, розробку конструктора завдань, модулів відстеження прогресу та публічної вітрини челенджів.

У четвертому розділі наведено результати тестування інтерфейсу користувача й системи відповідними даними.

У висновках узагальнено результати виконаної роботи, підтверджено виконання поставлених на початку завдань та визначено шляхи подальшого розширення функціоналу створеного вебзастосунку.

КРБ викладена на 78 сторінки, вона містить 4 розділи, 18 ілюстрацій, 14 таблиці, 25 джерел в переліку посилань.

Ключові слова: конструктор челенджів, вебплатформа, FastAPI, React, гейміфікація, відстеження прогресу, бази даних, мікросервісна архітектура.

ABSTRACT

to the bachelor's qualification work

"Personal challenge builder"

Student of group 409: Babiienko Oleg

Supervisor: Senior Lecturer Borovlyova Svitlana

The relevance of the work is due to the growing need for flexible tools for self-development, motivation and systematic achievement of personal goals using digital solutions.

The object of the study is the process of automating personal progress tracking and implementing game mechanics in everyday activities.

The subject of the study is methods, models and software for creating interactive web platforms with tracking and gamification modules.

The purpose of the work is to develop and implement an information system for creating, managing challenges and ensuring interaction between organizers and participants of the platform.

The bachelor's qualification work is presented on 78 pages, it contains 4 chapters, 18 illustrations, 14 tables, and 25 sources in the list of references.

In the introduction substantiates the relevance of the chosen topic, defines the goal, main tasks, object and subject of development, and also reveals the practical significance of the obtained results.

The first section provides a comprehensive study of the target subject area, as well as a comparative review of existing digital tools for controlling habits and personal goal markets.

The second section contains a comprehensive specification of functional and operational requirements for the product being created and is devoted to the in-depth architectural design of the cloud web platform. This reveals the stages of developing a relational data storage model, as well as modeling the game incentives subsystem. Special attention is paid to the algorithmization of success monitoring processes and the design of a multi-level access hierarchy that clearly distinguishes between the repetition of ordinary participants, challenge organizers, and system administrators. Along with this, a

reasoned technical justification of the approved development stack is provided, which features the React library and the FastAPI microframework in combination with the PostgreSQL relational system.

The third section describes the process of software implementation of the web platform, the creation of a client-server application, the development of a task designer, progress tracking modules and a public showcase of challenges.

In the fourth section presents the results of testing the user interface and the system with relevant data.

In the conclusions summarize the results of the work performed, confirm the fulfillment of the tasks set at the beginning, and identify ways to further expand the functionality of the created web application.

The bachelor's qualification work is presented on 78 pages, it contains 4 chapters, 18 illustrations, 14 tables, and 25 sources in the list of references.

Keywords: challenge builder, web platform, FastAPI, React, gamification, progress tracking, databases, microservice architecture.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних
КБР – кваліфікаційна бакалаврська робота
ПЗ – програмне забезпечення
СКБД – система управління базами даних

ACID – транзакційна цілісність даних (Atomicity, Consistency, Isolation, Durability)
API – інтерфейс програмування застосунків (Application Programming Interface)
ASGI – асинхронний інтерфейс (Asynchronous Server Gateway Interface)
CORS – спільне використання ресурсів з різних джерел (Cross-Origin Resource Sharing)
DOM – об'єктна модель документа (Virtual DOM)
HTTP / HTTPS – протокол передачі гіпертексту
JSON – об’єктна нотація JavaScript (JavaScript Object Notation)
JWT – вебтокен JSON (JSON Web Token)
MDA – фреймворк механік, динамік та естетик (Mechanics, Dynamics and Aesthetics Framework)
ORM – об'єктно-реляційне відображення (Object-Relational Mapping)
OWASP – відкритий проєкт безпеки вебзастосунків (Open Worldwide Application Security Project)
REST – передача стану представлення (Representational State Transfer)
SaaS – програмне забезпечення як послуга (Software as a Service)
SPA – односторінковий застосунок (Single-Page Application)
SQL – реляційні та нереляційні бази даних (Structured Query Language)
UX – користувацький досвід (User Experience)

ВСТУП

Попри надмірну кількість трендів на продуктивність, ринку критично бракує зручних україномовних інструментів для гнучкого трекінгу цілей. Оскільки більшість існуючих додатків мають жорстку структуру та високу вартість, процес саморозвитку залишається фрагментованим. Саме тому створення єдиної SaaS-платформи, яка поєднує конструктор челенджів, гейміфікацію та публічну вітрину спільноти, повністю вирішує цю проблему і є реальною необхідністю для сучасних користувачів.

Мета роботи – проєктування та розробка вебзастосунку «Конструктор особистих челенджів» для спрощення взаємодії між учасниками, організаторами програм та модераторами платформи.

Для досягнення визначеної мети необхідно вирішити завдання:

– аналіз ринку сервісів для саморозвитку та виявлення недоліків існуючих трекерів;

– деталізація функціональних вимог та побудова моделей взаємодії акторів;

– обґрунтування вибору стеку FastAPI та React для мікросервісної архітектури;

– проєктування реляційної структури бази даних у PostgreSQL;

– впровадження автентифікації на базі JWT-токенів та рольової моделі доступу;

– програмна реалізація логіки динамічного конструювання челенджів;

– розробка модулів гейміфікації (бали, стріки) та системи фотодоказів;

– реалізація публічного каталогу з алгоритмами підрахунку рейтингу успішності;

– створення адміністративного інструментарію для модерації контенту;

– проведення комплексного тестування стабільності системи перед релізом.

Об’єкт роботи – інформаційні процеси управління особистою ефективністю та механізми формування нових звичок.

Предмет роботи – технологічні методи та програмні інструменти створення масштабованих SaaS-систем для конструювання персоналізованих програм розвитку.

Обґрунтування необхідності розробки

Більшість сучасних сервісів для саморозвитку орієнтовані на західного користувача або пропонують занадто примітивні списки справ, які не дають реального відчуття росту. Дослідження сучасних вебплатформ показало, що більшість із них мають такі недоліки:

– відсутність якісної локалізації та адаптації під українського юзера;

– занадто висока ціна підписки, що відштовхує студентів та молодих спеціалістів;

– жорстка структура завдань, яка не дозволяє кастомізувати виклик під свої потреби;

– брак зручної системи фотодоказів для підтвердження реального прогресу.

Нинішні тренди в IT свідчать про те, що люди хочуть бути авторами власного розвитку, а не просто клацати галочки. Світова практика показує перехід до гнучких SaaS-платформ, де акцент зміщується на автоматизацію трекінгу, впровадження фанової гейміфікації та створення закритих ком’юніті для спільного проходження челенджів

Сфера застосування результатів

Розроблена платформа може бути використана у:

– маркетплейсах авторських програм саморозвитку для пошуку тематичних викликів;

– діяльності коучів та менторів для управління спільнотами та аналізу статистики;

– повсякденному житті користувачів як інструмент візуалізації особистого росту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Розробка ефективного інформаційного простору не може відбуватися без ґрунтовного аналізу сфери, де він буде використовуватися. Сучасна індустрія розвитку особистості та оптимізації особистої продуктивності активно трансформується в цифрову площину, де ключову роль відіграють SaaS-платформи, які об'єднують користувачів в амбіційні спільноти. Це глибоке дослідження даного напрямку, встановлення архітектурних недоліків програмних продуктів конкурентів та чітке розуміння очікувань цільової аудиторії дозволяють надійно визначити актуальні завдання. Це виступає як міцна основа для побудови ультрасучасної, ефективної платформи, призначеної для створення ігрових завдань та моніторингу досягнень.

1.1 Глобальні тренди та передумови формування ринку платформ персонального зростання

Підходи до того, як людина керує власним часом, зазнали суттєвих змін. Звичне лінійне планування крок за кроком втрачає свою актуальність, поступаючись місцем більш гнучким системам. У таких нових методиках головний акцент робиться на ігрових механіках та динамічному трекінгу результатів [1]. Завдяки технологічному стрибку тепер можна аналізувати свої щоденні звички буквально на ходу. Для цього більше не потрібна жодна прив'язка до конкретної локації чи жорсткого соціального графіка. Саме завдяки цьому виник такий формат, як спільні челенджі. У цьому інтерактивному просторі особиста дисципліна учасника підживлюється загальною енергією групи. Сучасні додатки для таких завдань вже давно ніхто не сприймає просто як віртуальні блокноти для записів. Наразі це скоріше комплексні середовища, головна задача яких – допомогти людині виробити стійкі навички і не покинути роботу над собою на півшляху [2].

Щодо українського сегменту, то тут попит на подібний софт зростає дуже швидко. Причини очевидні: люди почали більше дбати про свій психологічний

стан, плюс масовий перехід на віддалену роботу чи навчання змушує шукати нові формати самоорганізації через постійні стреси. У таких реаліях розроблюваний «Конструктор особистих челенджів» перетворюється на цілком практичний інструмент. Він дає змогу звичайним користувачам взяти щоденний хаос під контроль і розкласти розмиті цілі на конкретні, зрозумілі кроки. Але незважаючи на те, що інтернет буквально переповнений різноманітними порадами щодо тайм-менеджменту, на практиці люди постійно стикаються з банальною перешкодою. Їм просто бракує єдиного зручного майданчика для збереження своїх результатів. Головний бар'єр, який збиває мотивацію – це розпорошеність інструментів. Створення довгострокових планів, завантаження фотографій на підтвердження результату, побудова графіків успішності та обговорення з іншими учасниками зазвичай розкидані по різних месенджерах або базових нотатках у смартфоні.

Сучасні реалії диктують зовсім інші вимоги до сервісів, орієнтованих на особистісний розвиток. Сьогодні величезним попитом користуються майданчики, де можна конструювати колективні виклики. Причому цей формат уже давно вийшов за межі вузької аудиторії соло-ентузіастів і зараз дуже активно впроваджується безпосередньо в корпоративний сектор, стаючи частиною культури компаній [3].

Оскільки запити в аудиторії абсолютно різні – починаючи від соло-фрілансерів і закінчуючи професійними менторами чи цілими компаніями – критично важливо звести їхню взаємодію в одну систему. Розробка єдиного майданчика для генерації таких викликів нарешті вирішує проблему необхідності перемикатися між десятками різних додатків. Коли всі досягнення, графіки та рейтингові таблиці знаходяться поруч, процес стає прозорим і зрозумілим для кожного.

Щоб технічно реалізувати таку архітектуру, стандартних підходів зазвичай недостатньо. Тут потрібен сучасний стек технологій, здатний швидко обробляти потоки даних та видавати плавний інтерфейс без жодних затримок. Якщо грамотно поєднати компонентну логіку на фронтенді з надійною серверною

інфраструктурою, на виході отримаємо дійсно гнучкий продукт. Така база дозволяє системі легко масштабуватися і спокійно витримувати пікові напливи користувачів, що є критичною вимогою для виживання на сучасному ринку.

1.2 Аналіз предметної галузі та структурно-функціональні особливості об'єкта дослідження

Сама концепція перетворення рутинного саморозвитку на гру зародилася ще на початку нульових у середовищі перших IT-ентузіастів. Те, що спочатку виглядало як локальні експерименти на форумах, досить швидко масштабувалося до рівня світового тренду. Сьогодні фахівці сприймають такі цифрові рішення як цілком закономірний крок в еволюції інструментів тайм-менеджменту [4]. Фактично, ми отримали ефективну заміну звичайним паперовим щоденникам, де віртуальний простір органічно поєднується з живим відстеженням результатів.

Якщо розкласти цей ринок на базові складові з точки зору архітектури, то побачимо чіткий поділ на дві групи користувачів. Перша – це безпосередньо організатори. Їхня головна мета полягає в тому, щоб зібрати навколо себе активну аудиторію та зручно модерувати весь процес. Друга група – звичайні учасники, які приходять за вже готовим, чітко структурованим планом дій. Відповідно, успішна платформа має працювати як надійний місток між ними, закриваючи потреби обох сторін [5]. Вона мусить зробити правила взаємодії абсолютно прозорими та дати стовідсоткову гарантію, що весь накопичений прогрес надійно збережеться і нікуди не зникне [6].

Якщо розкласти систему на базові сутності, то отримаємо кілька ключових елементів. Центральною ланкою тут є безпосередньо сам челендж, який має власні жорсткі правила та критерії успіху. Далі йде організатор, на якому лежить уся відповідальність за створення контенту. Не менш важливими є профілі звичайних учасників із відповідними правами доступу. Проте чи не найважливішим гвинтиком у цій механіці виступає щоденна звітність. Найчастіше це завантажене фото, яке слугує фактичним доказом виконання

денної норми. Саме цей медіафайл безпосередньо впливає на підсумкову оцінку та позицію людини в загальній рейтинговій таблиці.

Логіка розмежування прав у додатку базується на чотирьох ключових ролях. Звичайний гість ресурсу може вільно гортати каталог доступних марафонів, але суто в режимі читання. Після створення облікового запису користувач отримує статус учасника, що відкриває йому повноцінний доступ до вибраних викликів та можливість відправляти свої звіти. Третя ланка – це організатори. На них тримається вся модерація створених ними ж проєктів та безпосередня перевірка надісланих учасниками доказів. Нарешті, на найвищому рівні знаходиться адміністратор, який зазвичай не втручається в сам ігровий процес, але тримає під контролем бекенд і стежить за технічною цілісністю всієї платформи.

Також, завжди треба враховувати людський фактор, адже користувачі часто намагаються обійти правила. Тому система має жорстко блокувати дублікати запитів, спираючись на ліміти конкретного завдання. Оскільки платформа постійно перетравлює великі масиви медіафайлів, для швидкої роботи з фотодоказами використовуються масштабовані хмарні об'єктні сховища. Ще один дієвий запобіжник від накруток – це логіка коментарів. Залишити відгук про марафон можна виключно після його успішного проходження до самого кінця. Такий підхід автоматично відсікає будь-які маніпуляції з оцінками та робить мотиваційне середовище дійсно чесним.

1.3 Аналіз існуючих реалізацій аналогічних систем

Якщо зараз подивитися на ринок додатків для контролю звичок, то вибір здається величезним. Але практично всі ці сервіси мають одну спільну проблему – вони залишають людину наодинці з її цілями. Наявний софт майже не містить нормального функціоналу для моделювання спільних активностей, де група людей могла б долати випробування разом. Творці таких продуктів чомусь часто забувають про соціальну складову. Вони роблять ставку на сухі персональні графіки, хоча саме підтримка та конкуренція найкраще тримають мотивацію

користувача на довгій дистанції і дають проєкту шанс на масштабування. Щоб наочно побачити ці архітектурні прогалини та зрозуміти сильні сторони конкурентів, для аналізу було відібрано три популярні платформи зі схожим стеком можливостей. Їхній детальний розбір та порівняння за ключовими параметрами систематизовані у таблицях 1.1–1.6.

Таблиця 1.1 – Характеристика та аналіз системи Habitica

Назва	OfficeRnD
Виробник	HabitRPG, Inc.
Архітектура	Cloud-based SaaS, веб-сайт та мобільний додаток
Мова реалізації	JavaScript, Vue.js, Node.js, MongoDB
Основні функції	Гейміфікація у стилі RPG; створення щоденних завдань; система віртуальних нагород; соціальна взаємодія через квести та гільдії.
Переваги	Висока ігрова мотивація; наявність відкритого API; активна спільнота.
Недоліки	Складний інтерфейс; відсутність механізмів реальної верифікації виконання (фото-звітів); застарілий дизайн веб-версії.
Джерело інформації	https://www.habitica.com

Хоча Habitica і вважається найвідомішим рішенням серед гейміфікованих трекерів, цей продукт має один помітний недолік. Занадто сильний акцент на віртуальних розвагах часто відволікає людей від їхніх реальних цілей. Якщо подивитися на це з інженерної точки зору, то вибір SQL бази даних, зокрема MongoDB, відверто обмежує функціональну гнучкість системи. Без нормальної реляційної моделі вкрай важко будувати складні зв'язки між багаторівневими завданнями та нестандартними розкладами.

Щоб чітко побачити розбіжності між Habitica та розроблюваним конструктором, їхні ключові параметри звели в єдиний порівняльний аналіз. Детальний розбір функціоналу дозволив одразу підсвітити сильні сторони нового проєкту, особливо коли йдеться про детальні налаштування правил та надійну верифікацію надісланих доказів. Завдяки цьому вдалося тверезо оцінити,

наскільки ефективними є закладені архітектурні рішення. Найбільше уваги під час порівняння приділялося саме методам перевірки результатів (табл. 1.2).

Таблиця 1.2 – Порівняння системи Habitica та проєктованого вебзастосунку

Критерій	**Habitica**	**Проєктований вебзастосунок**
Цільова аудиторія	Індивідуальні користувачі та малі групи	Організатори челенджів, ментори та мотивовані учасники
Мова інтерфейсу	Багатомовна базова	Повна українська локалізація
Публічний каталог	Доступний лише для учасників окремих спільнот	Відкрита публічна вітрина челенджів із фільтрацією
Верифікація прогресу	Самостійна відмітка виконання звичайною кнопкою	Завантаження та перевірка фотодоказів
Система відгуків	Відсутня для конкретних завдань	Рейтинги з суворою прив'язкою до успішного завершення

Підсумовуючи результати порівняння, можна стверджувати: незважаючи на сильний ігровий рушій Habitica, розроблений вебзастосунок виграє на рівні самої концепції. Його головна відмінність полягає у жорсткому фокусі саме на колективних соціальних челенджах. Крім того, нова платформа має повноцінну українську локалізацію, а будь-який прогрес обов'язково підтверджується реальними медіафайлами замість звичайного кліку (табл. 1.3).

Таблиця 1.3 – Характеристика та аналіз системи Streaks

Назва	Streaks
Виробник	Crunchy Bagel
Архітектура	Native Mobile App (iOS/Apple ecosystem)
Мова реалізації	Swift

Кінець таблиці 1.3

Основні функції	Відстеження до 24 завдань; інтеграція з Apple Health; візуалізація через графіки; система нагадувань.
Переваги	Мінімалістичний дизайн; автоматизація через датчики здоров'я; відсутність підписок.
Недоліки	Відсутність веб-версії та підтримки Android; обмежена кількість завдань; немає соціальної взаємодії.
Джерело інформації	https://www.streaksapp.com

Хоча додаток Streaks і може похвалитися еталонним UX-дизайном для мобільних пристроїв, його жорстка прив'язка до однієї екосистеми суперечить концепції відкритого SaaS-рішення. Щоб чітко простежити розбіжності між цим сервісом та новим проєктом, їх порівняли за критеріями, які є найбільш критичними для українських користувачів. Усі деталі цього зіставлення наведено нижче (табл. 1.4).

Таблиця 1.4 – Порівняння системи Streaks та проєктованого вебзастосунку

Критерій	**Streaks**	**Проєктований вебзастосунок**
Головне призначення системи	Індивідуальний моніторинг щоденних рутинних звичок	Соціальна екосистема для конструювання спільних викликів
Перегляд каталогу без реєстрації	Відсутній	Наявна відкрита публічна вітрина
Публічні рейтинги учасників	Відсутні	Наявні з урахуванням успішних підтверджень
Формат фіксації прогресу	Одинарне інтерактивне натискання кнопки	Завантаження мультимедійних фотодоказів
Модель доступу та поширення	Виключно платний мобільний застосунок	Безкоштовний базовий доступ через веббраузер

Як видно з аналізу, Streaks залишається досить вузьким інструментом виключно для персонального трекінгу. Натомість розроблений застосунок

робить ставку саме на соціальну взаємодію. Він пропонує абсолютно прозору систему оцінювання, а також дозволяє гостям вільно переглядати загальний каталог челенджів навіть без створення облікового запису.

Таблиця 1.5 – Характеристика та аналіз системи The Fabulous

Назва	The Fabulous
Виробник	The Fabulous
Архітектура	Cloud-based SaaS, Mobile Apps
Мова реалізації	Kotlin, Swift, Node.js
Основні функції	Науково обґрунтовані програми; голосовий супровід; трекер звичок; групові челенджі.
Переваги	Фокус на психології; якісний контент; гарний візуальний стиль.
Недоліки	Висока вартість підписки; низька гнучкість для створення власних структур; нав'язливі сповіщення.
Джерело інформації	https://www.thefabulous.co

Додаток The Fabulous здебільшого сфокусований на тому, щоб користувач просто споживав уже готовий експертний контент. Такий підхід сильно обмежує свободу дій, адже людина фактично позбавлена можливості самостійно створювати та модерувати власні виклики. Щоб чітко побачити різницю між цим сервісом та новим розроблюваним проєктом, їх порівняли за тими параметрами, які є найбільш критичними саме для української аудиторії. Усі деталі цього аналізу зібрано нижче (табл. 1.6).

Таблиця 1.6 – Порівняння системи The Fabulous та проєктованого вебзастосунку

Критерій	**The Fabulous**	**Проєктований вебзастосунок**
Тип контенту та програм	Попередньо встановлені куровані рутини	Створення власних публічних челенджів користувачами

Кінець таблиці 1.6

Система публічних відгуків	Відсутня	Наявна після успішного проходження
Локалізація інтерфейсу	Базова багатомовна	Повна українська локалізація
Механізм верифікації завдань	Самостійна відмітка прогресу кнопкою	Завантаження обов'язкових мультимедійних фотодоказів
Формування списку улюблених програм	Обмежено рамками одного поточного плану	Наявне збереження улюблених викликів без обмежень

Хоча The Fabulous і приваблює чудовим дизайном та приємним інтерфейсом, він не вирішує головного – там неможливо шукати відкриті челенджі від незалежних менторів. До того ж сервіс досі не має української локалізації. Новий вебзастосунок закриває ці слабкі місця, пропонуючи відкритий майданчик для гнучкого конструювання випробувань та ведення прозорих публічних рейтингів.

Якщо підсумувати весь цей аналіз ринку, тенденція цілком очевидна. Популярні конкуренти заточені виключно під ізольований трекінг звичок. Вони відверто ігнорують як запити локального українського користувача, так і потребу в зручних інструментах для організації саме колективних соціальних активностей.

Висновки до розділу 1

Підсумовуючи перший розділ, можна впевнено стверджувати: гейміфікація саморозвитку зараз перебуває на піку популярності. Проте цей попит стикається з проблемою розпорошеності інструментів, адже майже все наявне програмне забезпечення орієнтоване виключно під одиночне відстеження прогресу.

Детальний розбір логіки майбутнього проєкту довів, що системі критично необхідний жорсткий контроль термінів та автоматичне блокування однакових фотодоказів. Саме прозора обробка медіафайлів та об’єктивне формування

рейтингів є тією базою, на якій триматиметься довіра між авторами завдань та їхніми учасниками.

Огляд головних конкурентів, зокрема Habitica, Streaks та The Fabulous, чітко підсвітив вільну нішу: на ринку гостро бракує відкритих майданчиків саме для колективних викликів. Більше того, ці сервіси не мають повноцінної української локалізації, не вимагають реальних фотографій для підтвердження результатів і приховують свій каталог завдань за обов'язковою реєстрацією.

Виявлені функціональні прогалини існуючих аналогів прямо вказують на необхідність закладення масштабованої програмної архітектури. Оскільки майбутня система повинна безперебійно обробляти великі масиви візуальних звітів, підтримувати розгалужену рольову модель доступу для звичайних учасників, наставників та системних адміністраторів, а також забезпечувати миттєве оновлення статистики, це вимагає ретельного підходу до вибору сучасних технологічних інструментів та проєктування надійної клієнт-серверної бази.

Враховуючи зазначені недоліки наявних рішень, створення спеціалізованого вебзастосунку виступає логічним та обґрунтованим етапом. Проєктована інформаційна система має об'єднати інструменти спільної взаємодії, гнучкий конструктор персоналізованих викликів та прозору систему верифікації результатів. Такий комплексний підхід дозволить повністю задовольнити потреби цільової аудиторії у якісному інструментарії для самовдосконалення. Відповідно, подальша робота потребує переходу до системного моделювання, формалізації технічних вимог та детальної розробки інженерної архітектури програмного продукту.

2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ РОБОТИ

Щоб грамотно спроєктувати архітектуру майбутнього проєкту, спочатку потрібно визначитися з актуальним технологічним стеком та патернами розробки. Створення розподіленої хмарної SaaS-платформи для конструювання челенджів вимагає потужних інструментів. Стандартними рішеннями тут не обійтися: лише сучасні технології здатні гарантувати, що сервіс працюватиме без перебоїв, витримає будь-які системні збої та зможе спокійно масштабуватися, коли кількість активних користувачів почне стрімко зростати.

2.1 Дослідження актуальних програмних засобів архітектурних моделей та технологічних підходів

Перш ніж братися за безпосереднє проєктування системи, потрібно чітко визначитися з інструментами, архітектурними моделями та методами розробки. Створення масштабованої хмарної SaaS-платформи для конструювання челенджів вимагає сучасного технологічного стеку. Тільки так можна забезпечити високу відмовостійкість сервісу та його здатність без проблем витримувати постійне зростання кількості користувачів.

Аналіз інструментарію для створення клієнтського інтерфейсу

Сьогодні розробка сучасних вебплатформ здебільшого спирається на архітектуру SPA, яка дозволяє динамічно оновлювати контент без повного перезавантаження сторінки [7]. Наразі у цій ніші домінують кілька основних рішень: Angular, Vue.js та React [9]. Angular пропонує потужну екосистему, але має занадто високий поріг входу, тоді як Vue.js іноді поступається масштабованістю на великих проєктах. Натомість React, завдяки компонентному підходу та механізму Virtual DOM, виглядає найбільш збалансованим вибором навіть порівняно з новішими технологіями на кшталт Svelte [10]. Саме Virtual DOM гарантує високу швидкість рендерингу, що вкрай важливо для постійного оновлення графіків прогресу та рейтингових таблиць без затримок. До того ж фронтенд платформ для саморозвитку вимагає високої інтерактивності та

миттєвого візуального фідбеку. А використання компонентної архітектури дає змогу ізольовано тестувати кожен елемент інтерфейсу, що суттєво підвищує загальну стабільність застосунку [11].

Технології побудови серверної логіки

Бекенд бере на себе всю бізнес-логіку та обробку запитів. На сьогодні Python залишається одним із базових стандартів індустрії, а FastAPI справедливо вважається одним із найшвидших мікрофреймворків для створення REST API [13]. Завдяки нативній підтримці асинхронності, через ASGI, та автоматичній генерації документації в OpenAPI, процес розробки значно пришвидшується [13]. До того ж архітектура FastAPI дозволяє гнучко масштабувати обчислювальні потужності, коли платформа стикається з високими навантаженнями [14]. Окремий і вкрай важливий фокус на бекенді – це безпека. Захист API має будуватися з огляду на сучасні стандарти, зокрема спираючись на рекомендації OWASP [19]. Жорстка валідація вхідних даних та правильно налаштовані політики авторизації – це абсолютний мінімум, щоб закрити систему від базових мережевих вразливостей [20]. А щоб надійно працювати з конфіденційною інформацією користувачів, серверна частина обов'язково має проходити комплексне тестування на вразливості за списком OWASP Top 10 [21].

Дослідження систем керування базами даних та моделей організації інформації

Будь-яка система трекінгу досягнень тримається на грамотній архітектурі даних. Оскільки логіка управління цілями базується на чітких сутностях із жорсткими зв'язками між ними, тут найкраще працюють реляційні бази. Практика показує, що для таких завдань ідеально підходить PostgreSQL. Вона не лише випереджає багато аналогів за продуктивністю у складних вибірках, але й гарантує стовідсоткову транзакційну цілісність ACID [16].

Щоб платформа миттєво реагувала навіть тоді, коли тисячі учасників одночасно оновлюють свій прогрес, на рівні самої БД застосовується глибока оптимізація SQL-запитів [16]. Окрему увагу приділено індексуванню таблиць із

профілями та звітами. Якщо коректно налаштувати індекси [16], система взагалі не гальмуватиме під час побудови важкої аналітики. Завдяки цьому ментори та адміністратори можуть без жодних затримок отримувати зведену статистику по будь-якому активному челенджу [18].

Аналіз засобів обробки та зберігання мультимедійного контенту

Оскільки завантаження щоденних фотодоказів є важливою частиною механіки застосунку, зберігати такі великі бінарні файли безпосередньо в таблицях бази даних недоцільно, адже це суттєво знизить її продуктивність. Натомість увесь мультимедійний контент виноситься у спеціалізовані хмарні об'єктні сховища, наприклад, до інфраструктури AWS, що додатково спрощує інтеграцію інструментів для обробки зображень.

У реляційній БД при цьому зберігаються лише текстові посилання, що цілком відповідає сучасним архітектурним вимогам щодо роботи з великими масивами даних [18]. Такий розподіл завдань між основним сервером та об'єктним сховищем повністю нівелює ризик відмови системи під час масового завантаження звітів у пікові вечірні години.

Специфіка предметної галузі платформ самовдосконалення

Сучасні трекери звичок вже давно вийшли за межі звичайних електронних блокнотів і еволюціонували у повноцінні соціальні платформи. Сьогодні ринок гостро потребує продуманої гейміфікації, здатної формувати стійку мотивацію та допомагати користувачам системно досягати своїх цілей [14].

З архітектурної точки зору, для стабільної роботи такої платформи оптимально підходить мікросервісна методологія. Вона дозволяє грамотно розподіляти серверні ресурси – наприклад, ізолювати важкий блок авторизації від публічної вітрини з каталогом челенджів, на яку зазвичай припадає найбільше трафіку.

Предметна галузь систем управління цілями та гейміфікації

Щоб чітко сформувати вимоги до застосунку, потрібно розуміти, що саме драйвить користувачів. Якщо розібрати ігрові механіки та юзабіліті через призму фреймворку MDA, стає зрозуміло: прозорі правила, зручний інтерфейс і

продумана віртуальна система нагород – це фундамент будь-якого успішного інтерактивного продукту [15].

Виходячи з цього, архітектура проєкту має забезпечувати миттєвий відгук системи, особливо коли йдеться про завантаження фотозвітів. Швидкий і безперебійний фідбек від бекенду не лише формує загальну довіру до платформи, а й безпосередньо впливає на утримання аудиторії, мотивуючи користувачів щодня закривати свої таски [15].

2.2 Обґрунтування технологічного стеку та архітектурної моделі

Вибір технологічного стеку – це завжди фундаментальне рішення, від якого безпосередньо залежить життєздатність проєкту, зручність розробки та простота його подальшого масштабування. Враховуючи базові вимоги до функціоналу та огляд сучасних інструментів, фінальну архітектуру застосунку було поділено на три класичні шари: клієнтську частину, серверну логіку та базу даних. Нижче детально розібрано та обґрунтовано вибір конкретних технологій для кожного з цих рівнів.

Засоби реалізації клієнтського інтерфейсу

Для реалізації клієнтської частини було обрано архітектуру SPA [1]. Фундаментом виступає бібліотека React 18 у зв'язці з TypeScript [2]. Сувора типізація критично важлива для надійності коду, особливо коли логіка ускладнюється. Вона значно спрощує підтримку та подальше масштабування системи, що робить цей стек оптимальним вибором [3].

Щоб не марнувати час на повільні білди, замість класичного Webpack для збірки проєкту використовується Vite. Він забезпечує практично миттєве оновлення модулів, HMR, під час розробки. Управління глобальним станом – наприклад, даними профілю, кошиком товарів чи активними бронюваннями – реалізовано через компактний стейт-менеджер Zustand. А для візуального оформлення та верстки застосовано TailwindCSS, що дозволяє максимально швидко стилізувати компоненти та без проблем адаптувати інтерфейс під будь-які мобільні екрани.

Серверний рівень: FastAPI та Python

За бекенд відповідає Python та високоефективний фреймворк FastAPI [4]. Такий стек є галузевим стандартом для побудови швидких і масштабованих вебзастосунків [5]. Архітектура FastAPI чудово підходить для створення стабільного, надійного бекенду, який легко підтримувати [10].

Безпека реалізована за сучасними стандартами: аутентифікація працює через JWT-токени, а для зберігання паролів використовується стійке криптографічне хешування BCrypt через пакет passlib [12]. За валідацію всіх вхідних даних – від створення нових челенджів до текстових звітів – відповідає бібліотека Pydantic. Вона повністю закриває вимоги до оцінки вразливостей [13] та стандартів безпеки OWASP [14]. Окрім захисту, Pydantic автоматично приводить типи даних до потрібного формату, що суттєво зменшує кількість помилок та несподіваних збоїв на сервері.

Архітектура збереження даних: PostgreSQL

Для надійної реєстрації, зберігання та обробки даних [6] було обрано PostgreSQL – потужну реляційну систему управління базами даних [7]. Така модель ідеально підходить для предметної області, де ключовим є конструювання викликів та робота зі зв’язками між користувачами, менторами та звітами.

Використання зовнішніх ключів дозволяє ефективно структурувати базові сутності платформи, а транзакційна підтримка ACID гарантує цілісність даних та стабільну роботу навіть під час пікових навантажень [11]. Для взаємодії з БД використовується SQLAlchemy, ORM, у зв’язці з високопродуктивним асинхронним драйвером asyncpg. Це дозволяє розробникам працювати з даними як із об'єктами Python, зберігаючи при цьому максимальну швидкість виконання SQL-запитів.

Механізми взаємодії між архітектурними компонентами системи

Інтеграція фронтенду та бекенду реалізована через REST API з обов'язковим використанням захищеного протоколу HTTPS. Клієнтська частина

генерує запити за допомогою бібліотеки Axios, автоматично додаючи авторизаційні токени до кожного виклику.

Після того, як сервер опрацьовує бізнес-логіку та взаємодіє з базою даних, з можливим розгортанням у хмарі AWS [15], результат повертається клієнту у форматі JSON. Така архітектура повністю розділяє рівні представлення та серверне ядро. Усі помилки бекенду перехоплюються та трансформуються в уніфіковані статус-коди, що дає змогу фронтенду коректно обробляти винятки та забезпечує міцний фундамент для майбутньої інтеграції мобільних застосунків.

2.3 Специфікація вимог до програмного забезпечення

1. Визначення цільового призначення та функціональних меж системи

1.1 Цільове призначення продукту

Для звичайних учасників передбачено інструментарій, що забезпечує інтуїтивний трекінг. Доступні можливості створення особистих челенджів, відстеження власного прогресу в реальному часі та участь у відкритих колективних змаганнях, що суттєво підвищує рівень відповідальності.

Для менторів та авторів контенту розроблено окремий модуль управління. Він дає змогу не просто публікувати унікальні шаблони завдань, а й повноцінно модерувати командні проєкти. Завдяки розширеній аналітиці наставники можуть бачити реальну динаміку залученості своїх підопічних, що дозволяє вчасно коригувати стратегії навчання чи мотивації.

Це розділення ролей дозволяє платформі бути водночас простою для новачків та глибоко функціональною для тих, хто професійно займається наставництвом або коучингом.

1.2 Базові конвенції та документальні домовленості

Цей перелік технічних вимог відповідає настановам міжнародного стандарту ISO/IEC/IEEE 29148:2025. В основі архітектури лежить стек React 18, Vite та TypeScript на фронтенді, бекенд-ядро на базі FastAPI та реляційна база

даних PostgreSQL. Проєкт постачається з повною українською локалізацією, адаптивною верстkою під будь-які екрани та підтримкою перемикання між світлою і темною темами оформлення. Захист персональних даних реалізовано через криптографічні алгоритми хешування та шифрування, інтегровані безпосередньо в логіку серверного рівня.

1.3 Рамки та обмеження програмної реалізації

Запланований обсяг робіт охоплює створення динамічного вебклієнта та відмовостійкого серверного бекенда, необхідних для повноцінного управління мотиваційними викликами та ігровими механіками. У межах поточної ітерації розробки свідомо виключено створення нативних мобільних застосунків, імплементацію офлайн-режиму роботи, підключення зовнішніх платіжних систем для реалізації платних підписок, а також розробку спеціалізованих інструментів для проведення професійних психотерапевтичних сесій.

2. Загальний опис

2.1 Галузь експлуатації продукту

Розроблений вебзастосунок займає своє місце в сучасному інформаційному середовищі інструментів для підвищення персональної продуктивності та самовдосконалення. Запропоноване рішення об'єднує дві ключові групи споживачів: звичайних користувачів, які прагнуть сформувати корисні звички за допомогою ігрових механік, та наставників, які займаються проєктуванням комплексних курсів мотивації й адмініструванням масових соціальних викликів.

2.2 Специфікація цільової аудиторії та їхніх ролей

Користувацька модель платформи включає такі ролі: анонімний відвідувач, тобто особа без облікового запису, яка має доступ до вільного перегляду каталогу загальнодоступних викликів та ознайомлення з основними положеннями системи без необхідності реєстрації; активний учасник, тобто авторизований користувач, зацікавлений у пошуку мотиваційних ініціатив, який має доступ до ергономічного редактора для створення персональних цілей, можливості додавання візуальних підтверджень прогресу та персоналізованої

панелі аналітики досягнень; наставник або організатор, тобто автор контенту або лідер спільноти, який використовує адміністративну панель для управління макетами завдань, проведення командних турнірів та аналізу метрик активності підписників; головний модератор або адміністратор, тобто технічний фахівець, відповідальний за повний контроль публікацій, розподіл ієрархії дозволів та ведення журналів безпеки системи.

2.3 Архітектурна композиція та компонентна база

Фронтенд-модуль розроблено з використанням можливостей бібліотеки React 18, яка доповнюється інструментами Vite та TypeScript. Водночас обчислювальний бекенд-центр функціонує завдяки швидкодіючому середовищу FastAPI, тоді як роль фундаментального сховища для накопичення даних виконує реляційна СУБД PostgreSQL.

2.4 Глобальні ліміти та експлуатаційні умови

Застосунок коректно функціонує лише за умови стабільного інтернет-підключення через захищений канал із мінімальною пропускною здатністю від 1 Мбіт/с. Гарантується сумісність із браузерами: Safari версії 14 і вище, Edge та Chrome версії 90 і вище, а також Firefox версії 88 і вище. З метою раціонального використання серверного дискового простору обсяг мультимедійних підтверджень щоденної активності обмежено п'ятьма мегабайтами на один файл. Також, для уникнення інформаційного перевантаження та спаму сповіщеннями, кожному користувачеві дозволено одночасно брати участь максимум у десяти поточних викликах, тоді як куратори зі стандартним рівнем доступу обмежені можливістю розміщення лише п'яти відкритих макетів у глобальному реєстрі завдань.

3. Функціональні можливості програмного продукту

3.1 Механізми пошуку та ознайомлення із загальнодоступними викликами

3.1.1 Характеристика процесу

Система надає розвинений інструментарій для ефективного пошуку відкритих мотиваційних програм, забезпечуючи користувачам інтуїтивно

зрозумілу взаємодію з платформою. Пошуковий алгоритм підтримує фільтрацію за назвою або ключовою тематикою, а також пропонує можливості тонкого налаштування параметрів відбору, включаючи такі критерії, як загальна тривалість, рівень складності завдань та інтенсивність навантаження. Отримані результати пошуку інтегровані з інтелектуальною системою сортування за рейтингом, що базується на аналізі активності учасників та відгуках спільноти. Це дозволяє користувачам швидко ідентифікувати найактуальніші та найперспективніші проєкти, отримуючи миттєвий доступ до деталізованих описів кожного обраного турніру для прийняття зваженого рішення щодо участі.

3.1.2 Потоки вхідних та вихідних даних

— параметри на вході, текстовий запит, конфігурація фільтрів, алгоритм упорядкування;

— результати на виході, масив карток завдань, що містять заголовки, тематичні мітки, графічні прев'ю, поточну статистику залученості та агреговану оцінку складності.

3.1.3 Технічні критерії виконання

— алгоритм знаходження спрацьовує навіть при неповному збігу символів у заголовках;

— видача підвантажується динамічно за принципом нескінченної стрічки;

— швидкість відпрацювання звернення серверною частиною має вкладатися у ліміт до двох секунд.

3.2 Реєстрація участі у програмі та моніторинг індивідуальних досягнень

3.2.1 Характеристика процесу

Зареєстрованому користувачеві надається можливість долучитися до відкритого змагання або активувати персональний шаблон для старту нового циклу активностей. Система автоматично формує календарний план, розраховує

кінцеву дату завершення та створює спеціальні поля для регулярної публікації звітів про виконані завдання.

3.2.2 Потоки вхідних та вихідних даних

— параметри на вході, унікальний код обраного завдання, початкова часова мітка;

— результати на виході, згенерований запис про участь із маркуванням «у процесі», розгорнута сітка дедлайнів, актуалізовані метрики на персональному дашборді.

3.2.3 Технічні критерії виконання

— діє жорстке обмеження на паралельну активацію однакових мотиваційних курсів в одного юзера;

— алгоритм автоматично переводить змагання у стан «не виконано», якщо зафіксовано перевищення ліміту пропущених звітів;

— можливість долучення деактивується, якщо квота вільних позицій вичерпана.

3.3 Публікація підтверджень активності та акумулювання нагород

3.3.1 Характеристика процесу

Система надає інструменти для регулярної фіксації досягнень через завантаження фотоматеріалів та додавання супровідних текстових коментарів. Одразу після успішного збереження даних алгоритми оновлюють показники безперервності серії, а також ініціюють нарахування бонусних балів або розблокування відповідних цифрових значків за виконання завдань.

3.3.2 Потоки вхідних та вихідних даних

— параметри на вході, мультимедійний документ, текстове пояснення, ідентифікатор відповідного слота активності;

— результати на виході, зафіксований запис із точною датою, перерахований баланс рейтингу, екранне повідомлення щодо прогресу.

3.3.3 Технічні критерії виконання

— діє квота на прикріплення виключно одного доказу до конкретної цілі впродовж двадцяти чотирьох годин;

— вага візуальних файлів має чітко встановлені межі;

— внесена у базу хронологія результатів стає доступною лише для читання без права на редагування.

3.4 Проєктування мотиваційних шаблонів та адміністрування публікацій

3.4.1 Характеристика процесу

Кураторам надається можливість створювати нові шаблони змагань, налаштовувати частоту повторів, визначати шкалу складності та додавати зображення-обкладинку. Після завершення налаштування сформований курс відправляється на рецензію для подальшої публікації у загальній галереї платформи.

3.4.2 Потоки вхідних та вихідних даних

— параметри на вході, найменування, деталізована інструкція, цільова рубрика, налаштування розкладу, зображення-заставка;

— результати на виході, згенерована сутність курсу, що отримує маркування «очікує перевірки» чи «прихований».

3.4.3 Технічні критерії виконання

— стовідсоткове заповнення інформації в акаунті автора є критичною вимогою для виходу у глобальний пошук;

— стандартні права користувача лімітують максимальну кількість одночасних макетів до п'яти одиниць;

— відкриті ініціативи індексуються платформою виключно після схвалення модератором.

3.5 Глобальне керування системою та контроль якості контенту

3.5.1 Характеристика процесу

Фахівцям із відповідними привілеями надано інструменти для валідації, узгодження або відхилення авторських пропозицій, блокування облікових записів порушників, очищення системи від небажаного медіаконтенту, а також проведення детального аналізу журналів безпеки для забезпечення стабільної роботи платформи.

3.5.2 Потоки вхідних та вихідних даних

— параметри на вході, критерії фільтрації у таблиці логів, модераторські вердикти щодо публікацій;

— результати на виході, структурована вибірка подій безпеки, змінений стан відмодерованого макета чи облікового запису.

3.5.3 Технічні критерії виконання

— абсолютно всі значущі операції юзерів незворотно записуються до реєстру подій, фіксуючи хронометраж та електронну пошту виконавця;

— панель моніторингу відкривається лише для профілів із найвищим рівнем доступу і обов'язково оснащена засобами сортування та посторінкової розбивки.

4. Критерії до інформаційного забезпечення системи

4.1 Канали надходження та структура вхідних даних

Первинні масиви даних формуються через пряму взаємодію користувачів з інтерфейсом: під час реєстрації облікових записів, автентифікації, створення авторських турнірів та завантаження візуальних підтверджень виконання завдань. Супутні налаштувальні параметри зчитуються з URL-адрес, що забезпечує стабільну роботу модулів фільтрації, алгоритмів сортування та пагінації в каталозі загальнодоступних курсів.

4.2 Довідково-нормативна база

Архітектура застосунку передбачає використання вбудованих реєстрів, які охоплюють суворо стандартизовані переліки тематичних галузей, рівнів складності змагань, ієрархії прав користувачів, поточних статусів активності та типів цифрових винагород.

4.3 Специфікації щодо методів структурування, накопичення та адміністрування даних

Усі структуровані масиви даних зберігаються у високопродуктивній реляційній СУБД PostgreSQL. Облікові дані та критично важлива приватна інформація користувачів проходять обов’язкове криптографічне хешування ще до запису в базу. Персоналізовані налаштування акаунтів разом із динамічними

алгоритмами нарахування ігрового досвіду зберігаються як серіалізовані JSON-структури безпосередньо у відповідних колонках таблиць. Запропонована парадигма проєктування забезпечує високий ступінь гнучкості системи, дозволяючи уникати регулярних змін фундаментальної схеми відношень бази даних.

5. **Критерії до апаратно-технічної бази**

Для повноцінного функціонування програми телекомунікаційне середовище має гарантувати безперервний доступ до глобальної мережі з обов'язковим використанням криптографічного протоколу HTTPS, що є критично важливим для захищеного передавання облікових даних та мультимедійного контенту. Як термінали для взаємодії з платформою дозволяється використовувати настільні робочі станції або портативні смарт-пристрої, обов'язково оснащені актуальними версіями браузерів. Чітке дотримання зазначених експлуатаційних параметрів виступає фундаментом для коректного відображення динамічних графіків успішності, стабільної роботи інструментів створення викликів та швидкісного додавання візуальних підтверджень прогресу.

6. **Специфікація програмного забезпечення та технологічного оточення**

6.1 Структурна парадигма системи

В основі архітектури лежить традиційна трирівнева клієнт-серверна модель, у межах якої чітко розмежовано графічну оболонку користувача, фронтенд, обчислювальний центр бізнес-правил, бекенд, та модуль оперування даними, база даних. Транзит інформаційних пакетів між цими автономними вузлами реалізується винятково із залученням зашифрованого протоколу HTTPS.

6.2 Інфраструктурне та системне середовище

Як базові платформи для розгортання бекенду використовуються серверні дистрибутиви архітектури Linux, а саме Ubuntu версії 24.04 LTS або новіші версії, що працюють разом із вебсервером Nginx для організації безперебійного

зворотного проксіювання вхідного трафіку. Для забезпечення надійної ізоляції програмних компонентів та автоматизації процесів доставки коду додатково застосовується технологія контейнеризації на базі Docker.

6.3 Комунікаційні та мережеві модулі

Передбачено використання сучасних засобів криптографічного перетворення трафіку відповідно до специфікацій TLS 1.3, а також інтеграцію мережевих пакетів, що забезпечують вбудовану сумісність із високошвидкісним протоколом передачі даних HTTP/2.

6.4 Платформи адміністрування сховищ інформації

Для роботи з даними використовується потужна реляційна СУБД PostgreSQL. Взаємодія з нею відбувається не напряму, а через ORM-фреймворк SQLAlchemy в поєднанні з оптимізованим асинхронним драйвером asyncpg, що забезпечує високу продуктивність виконання запитів.

6.5 Мовні стандарти та інструментарій конструювання

Розробка серверної логіки базується на мові Python із використанням сучасного мікрофреймворку FastAPI та залученням супутніх бібліотек Alembic, для міграцій бази даних, PyJWT, для автентифікації, passlib, для хешування паролів, і pydantic, для валідації даних. Проєктування динамічної візуальної оболонки спирається на синтаксис TypeScript, екосистему React 18, швидкісний бандлер Vite 5 та утилітарний CSS-фреймворк TailwindCSS 4. Для забезпечення повноцінної взаємодії впроваджено контролер глобального стану Zustand, HTTP-клієнт Axios, а також низку допоміжних фронтенд-пакетів: Recharts, для візуалізації даних, React Hook Form, для управління формами, react-dropzone, для завантаження файлів, і framer-motion, для анімацій інтерфейсу.

7. Специфікації зовнішніх інтерфейсів

7.1 Користувацька графічна оболонка

Візуальний стиль проєкту розробляється з урахуванням принципів мінімалізму, пропонуючи користувачам вибір між темним та світлим режимами відображення. Завдяки впровадженню гнучких CSS-практик забезпечується коректне розташування елементів інтерфейсу на екранах будь-якого типу: від

компактних мобільних пристроїв із мінімальною роздільною здатністю 320x480 пікселів до широкоформатних десктопних моніторів з роздільною здатністю до 1920x1080 пікселів.

7.2 Апаратна взаємодія

Особливості роботи хмарної мотиваційної платформи не вимагають підключення жодних вузькоспеціалізованих фізичних приладів чи додаткових периферійних пристроїв.

7.3 Програмні комунікаційні інтерфейси

Обмін інформаційними потоками між клієнтським браузером та сервером реалізовано виключно відповідно до архітектурного стилю REST API. Для виконання операцій задіяно стандартні HTTP-методи, при цьому тіла відповідей форматуються як серіалізовані JSON-об'єкти з використанням стилю іменування camelCase.

За результатами опрацювання запитів бекенд генерує загальноприйняті HTTP-статуси: 200 OK та 201 Created для успішних транзакцій; 400, 401, 403 та 404 у разі помилок клієнта або обмежень доступу; 429 при перевищенні ліміту запитів, а також 500 для індикації критичних збоїв на стороні сервера. Важливою особливістю є автоматизована генерація технічної API-документації: завдяки вбудованим механізмам FastAPI формується інтерактивна панель Swagger UI згідно зі специфікацією OpenAPI, що значно спрощує процес тестування та інтеграції.

7.4 Телекомунікаційні стандарти зв'язку

Передача будь-яких мережевих пакетів між вузлами системи допускається винятково через зашифрований транспортний канал HTTPS, що є безальтернативною умовою забезпечення конфіденційності даних.

8. Атрибути якості та експлуатаційні характеристики системи

8.1 Рівень доступності

Цільовий показник аптайму, безперебійної роботи, встановлено на рівні 99,9%. Такий норматив означає, що сукупна тривалість запланованих технічних

пауз або непередбачуваних відключень сервісу не повинна перевищувати ліміт у вісім годин 45 хвилин протягом одного календарного року.

8.2 Придатність до обслуговування

Сувора декомпозиція програмних модулів як на фронтенді, так і на бекенді створює сприятливі умови для безперешкодного масштабування платформи. Крім того, такий підхід значно полегшує процеси рефакторингу та впровадження нових алгоритмів до механізмів ігрових заохочень.

8.3 Портативність

Розроблена архітектура застосунку повністю адаптована для міграції між різними хмарними сервісами. Завдяки цьому забезпечується гнучке налаштування обчислювального вузла та сховища даних PostgreSQL у будь-якому сучасному віртуальному оточенні.

8.4 Показники швидкодії

Ключовою вимогою до системи є забезпечення відгуку серверної частини тривалістю менше однієї секунди. Цей стандарт має беззаперечно виконуватися навіть у моменти пікових навантажень, коли близько тисячі користувачів одночасно ініціюють надсилання мультимедійних підтверджень досягнення своїх цілей.

8.5 Відмовостійкість

Регламентований період відновлення серверів після виникнення критичних помилок обмежено п'ятьма хвилинами. Для уникнення втрати даних налаштовано циклічне створення резервних копій, які охоплюють усю хронологію активності та метрики успішності всієї спільноти.

8.6 Критерії кібербезпеки

Сесійна автентифікація базується на генерації JWT-токенів із добовим життєвим циклом тривалістю 1440 хвилин. Паролі користувачів проходять незворотне хешування за алгоритмом BCrypt, а приватні дані підлягають надійному шифруванню. Від SQL-ін'єкцій інфраструктуру захищають вбудовані механізми ORM SQLAlchemy. Поруч із цим реалізовано лімітування запитів для

запобігання перевантаженням та жорстко налаштовано правила CORS для відсікання шкідливого крос-доменного трафіку.

9. **Додаткові критерії та обмеження**

Програмні алгоритми управління клієнтською інформацією розроблені з повним дотриманням чинного законодавства, зокрема нормативних актів про захист персональних даних. Це гарантує законність та надійність на всіх етапах збору, обробки та зберігання приватної інформації користувачів. На рівні глобального управління платформою впроваджено механізм реалізації права на забуття: за першим запитом учасника відповідальний технічний фахівець має доступ до інструментів, що дозволяють остаточно видалити обліковий запис із безповоротним стиранням усієї історії індивідуальних досягнень та прикріплених медіафайлів.

Водночас у закритому системному журналі постійно накопичується інформація про всі важливі модераторські дії в екосистемі. Процес перевірки працює автоматично, детально реєструючи IP-адресу, електронну пошту ініціатора та точний час кожної операції. Крім того, закладена архітектурна основа дозволяє легко розширювати логіку мотиваційних турнірів та впроваджувати нові правила гейміфікації, уникаючи потреби в масштабній перебудові початкової реляційної моделі бази даних.

Висновки до розділу 2

Підбиваючи підсумки матеріалів другого розділу, варто відзначити успішне проведення всебічного аналізу сучасних засобів розробки та докладне проєктування цільової предметної сфери. Спираючись на сформовану специфікацію програмних вимог, було визначено чіткі рамки майбутнього продукту, ідентифіковано основні категорії цільової аудиторії та описано головні алгоритми їхньої взаємодії із системою. Також ґрунтовно розкрито ключовий функціонал проєкту: від механізмів створення індивідуальних чи колективних завдань до інструментів фіксації досягнень через додавання

регулярних візуальних доказів, включно з модулями ігрових заохочень та панеллю адміністративного управління.

Поряд із цим, у межах цього етапу наведено аргументовану доказову базу щодо доцільності впровадження конкретних технологічних рішень у контексті класичної трирівневої клієнт-серверної моделі. Фундаментом для візуальної оболонки обрано екосистему React 18, яка доповнюється статичним типізатором TypeScript та середовищем збирання Vite. Роль обчислювального бекенд-ядра відведено мікрофреймворку FastAPI, тоді як відповідальність за надійне зберігання даних покладено на реляційну СУБД PostgreSQL. У тексті деталізовано сильні сторони кожної ланки, серед яких: оптимізація рендерингу завдяки алгоритмам Virtual DOM, високий рівень захисту та адаптивності серверної логіки через використання middleware-компонентів, а також стовідсоткова консистентність транзакцій на рівні інформаційного сховища.

Крім того, значну увагу було приділено проєктуванню моделі даних та логіки гейміфікації, які складають основу мотиваційної платформи. Детально опрацьовано механізми нарахування балів досвіду, принципи формування безперервних серій активності та умови розблокування ігрових досягнень. Оскільки ці процеси тісно пов'язані з обробкою щоденних фотозвітів і вимагають постійної синхронізації між клієнтом та сервером, обрана масштабована архітектура дозволить системі миттєво оновлювати персональну статистику учасників без затримок інтерфейсу.

Отже, напрацювання цієї частини роботи формують деталізований перелік технічних умов. Вони закладають методологічну та архітектурну основу, необхідну для старту практичної розробки відмовостійкого, продуктивного та захищеного цифрового середовища для конструювання мотиваційних викликів.

3 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ

Перехід до практичного кодування вебплатформи вимагає детального опрацювання її внутрішньої структури та логіки взаємодії окремих елементів. Чітка формалізація системи за допомогою уніфікованої мови моделювання UML створює міцний інженерний фундамент для написання програмного коду. Своєю чергою, поетапне об'єднання серверної бізнес-логіки та клієнтської частини забезпечує повну відповідність готового продукту сучасним індустріальним вимогам щодо надійності, інформаційної безпеки та продуктивності.

3.1 Розробка UML-діаграм

Для вебплатформи «Конструктор особистих челенджів» UML-діаграми застосовуються як універсальний візуальний «словник», що узгоджує бачення замовника, розробників і майбутніх користувачів. Вони сприяють формальній фіксації функціональних сценаріїв, внутрішньої структури програмних модулів та послідовності їхньої інтеграції із зовнішнім середовищем. Стандарт UML охоплює різні рівні абстракції – від високорівневих бізнес-процесів до технічних деталей реалізації, тому дозволяє поступово деталізувати модель, зберігаючи її цілісність.

Таблиця 3.1 – Перелік UML-діаграм, запланованих для вебплатформи

Тип діаграми	**Мета використання**	**Нотація / інструмент**	**Охоплення в системі**
Діаграма використання	Фіксація сценаріїв і акторів	UML Use Case / StarUML	Гість, Учасник, Ментор, Адміністратор
Діаграми взаємодії	Послідовність викликів між об’єктами	UML Sequence / StarUML	Реєстрація, звіт, статистика

Кінець таблиці 3.1

Діаграма класів	Статична структура сутностей	UML Class / StarUML	User, Challenge, ProgressReport, Achievement
Діаграми діяльності	Алгоритмічні потоки	UML Activity / StarUML	Конструювання челенджу, валідація фото
Діаграма компонентів	Логічні модулі й залежності	UML Component / StarUML	UI (React), API (FastAPI), DB (SQLAlchemy)
Діаграма розгортання	Фізичне розміщення вузлів	UML Deployment / StarUML	Клієнт, Вебсервер, Сервер БД

Наведений набір схем забезпечує послідовне уточнення моделі впродовж життєвого циклу проєкту. На практиці робота відбувається ітеративно: спочатку формується «каркас» системи, далі уточнюються класи, а після стабілізації логіки фіксуються стани й схеми розгортання.

3.1.1 Діаграма використання

Діаграма відображає чотири головні категорії користувачів: гість, зареєстрований користувач, наставник та адміністратор, які взаємодіють із системою для виконання визначених бізнес-завдань. Центральним елементом схеми є сценарій «Авторизуватися у вебдодатку», який неодмінно пов'язаний з усіма іншими функціями через зв'язок «include», що гарантує перевірку прав доступу перед початком будь-якої дії. Така структура та спосіб зв'язків забезпечують гнучкість побудови та надійний розподіл повноважень між різними групами користувачів сервісу.

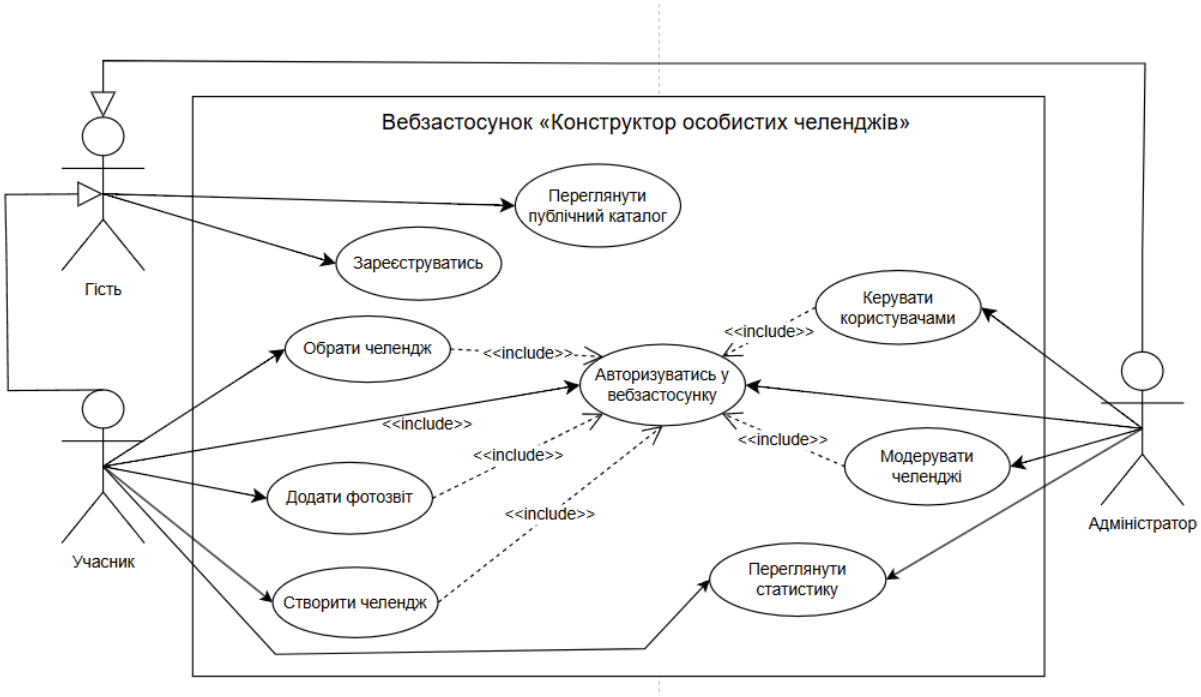

Рисунок 3.1 – Діаграма використання

Наведена схема чітко відображає ієрархію користувачів за допомогою зв'язків узагальнення. Згідно з цією ієрархією, адміністратор має повний набір можливостей наставника, який, своєю чергою, успадковує всі права звичайного учасника. Учасник розширює базовий функціонал неавторизованого гостя. Варіанти використання також пов'язані між собою зв'язками «extend», що дозволяє відображати додаткові та необов'язкові функції, які активуються лише за певних умов. Це робить архітектуру платформи надзвичайно гнучкою та легкорозширюваною.

3.1.2 Діаграма взаємодії

Представлена діаграма послідовності детально візуалізує процес надсилання користувачем мультимедійного доказу досягнення мети. Сценарій описує логіку взаємодії між актором Учасник, клієнтською частиною React, серверним ядром FastAPI та базою даних PostgreSQL. Процес ініціюється вибором та відправленням фотографії через інтерфейс React, після чого клієнтська частина формує POST-запит, що містить JWT-токен та файл. Отримавши запит, сервер FastAPI виконує послідовність перевірок: валідує

токен безпеки та перевіряє формат отриманого зображення. Після успішної верифікації сервер ініціює запис шляху до файлу в базу даних PostgreSQL, отримуючи підтвердження про успішне збереження. Наступним етапом відбувається оновлення ігрових балів досвіду користувача в БД, що супроводжується фіксацією оновленої статистики успішності. Завершується сценарій передачею від сервера до інтерфейсу React статусу успішності та актуальних балів, що призводить до відображення оновленого графіка прогресу в особистому кабінеті учасника.

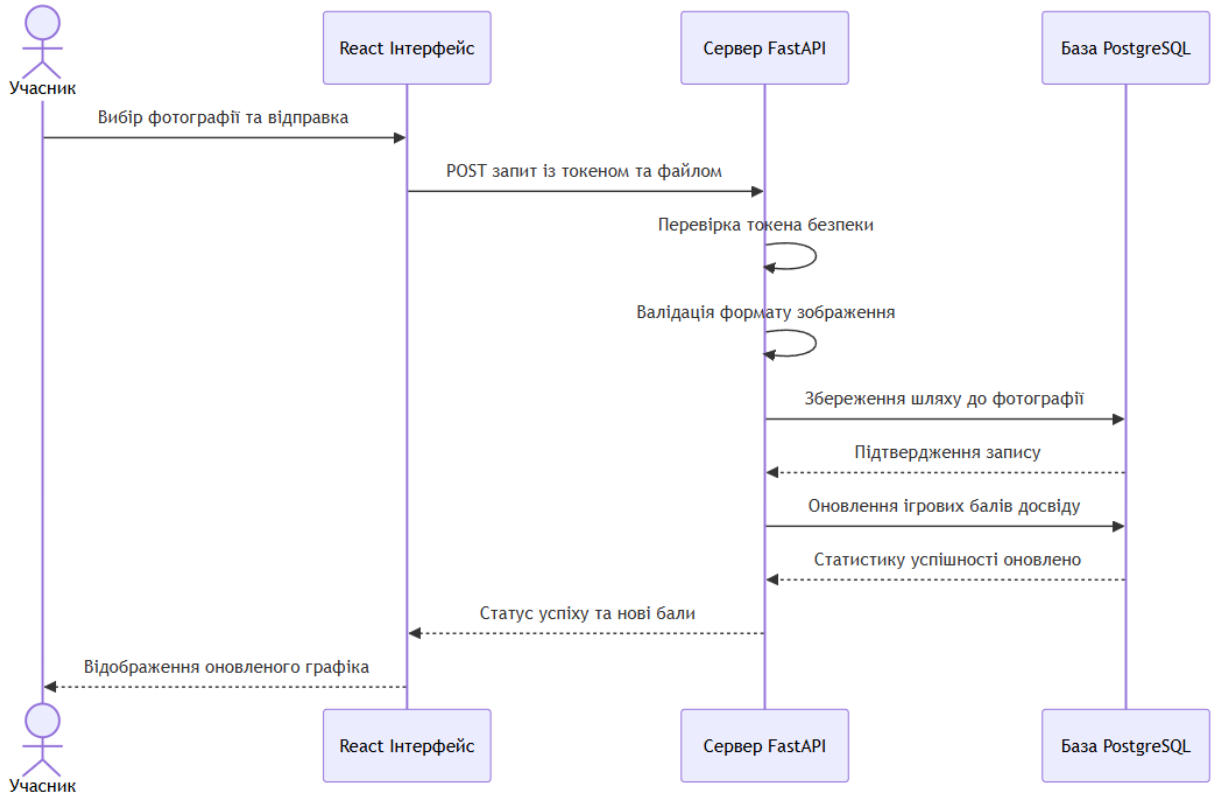

Рисунок 3.2 – Діаграма взаємодії

Діаграма демонструє складний процес завантаження щоденного підтвердження від зареєстрованого гравця. Вона включає всі необхідні кроки від вибору файлу на пристрої до фінального оновлення ігрової статистики. Тут відзначено перевірку токена доступу, запис шляху до картинки в таблицю даних та обчислення нових очок досвіду. Діаграма наочно демонструє асинхронну

роботу між клієнтським інтерфейсом, сервером обробки та реляційною базою даних.

3.1.3 Діаграма класів

Діаграма класів показує основну статичну структуру моделі домену нашого застосунку. Вона містить усі сутності, їхні внутрішні властивості, функції та зв'язки між ними. Вона слугує планом даних та поведінки для розробників, значно спрощуючи узгодження назв полів, типів змінних і бізнес-правил ще до створення реальних таблиць та контрактів обміну інформацією. У межах поточного проєкту діаграма класів надійно закріплює ядро системи, яке охоплює класи користувача, виклику, категорії, участі, звіту та досягнень.

Класи системи побудовано таким чином, щоб повноцінно забезпечити відповідність усім заданим функціональним вимогам застосунку. На схемі використано різні типи зв'язків між класами. Наявна композиція між записом про активність та звітами, оскільки звіт не може існувати окремо без фактичної участі користувача у конкретному виклику. Також реалізовано агрегацію між категоріями та челенджами. Кожен змодельований клас містить виключно необхідні характеристики, для яких було визначено відповідні суворі типи даних, що зменшує ймовірність виникнення помилок.

Окрему увагу під час проєктування діаграми було приділено механізмам успадкування та розширюваності структури, зокрема для розмежування прав доступу між звичайними користувачами та адміністраторами. Використання поліморфних зв'язків та чітке визначення інтерфейсів для методів керування дозволяє легко додавати нові типи активностей або ігрових механік у майбутньому без потреби в кардинальній зміні існуючої структури бази даних. Такий підхід забезпечує гнучкість системи, гарантуючи, що розроблена модель даних залишатиметься актуальною навіть у разі суттєвого ускладнення бізнес-логіки та розширення функціональних можливостей вебплатформи.

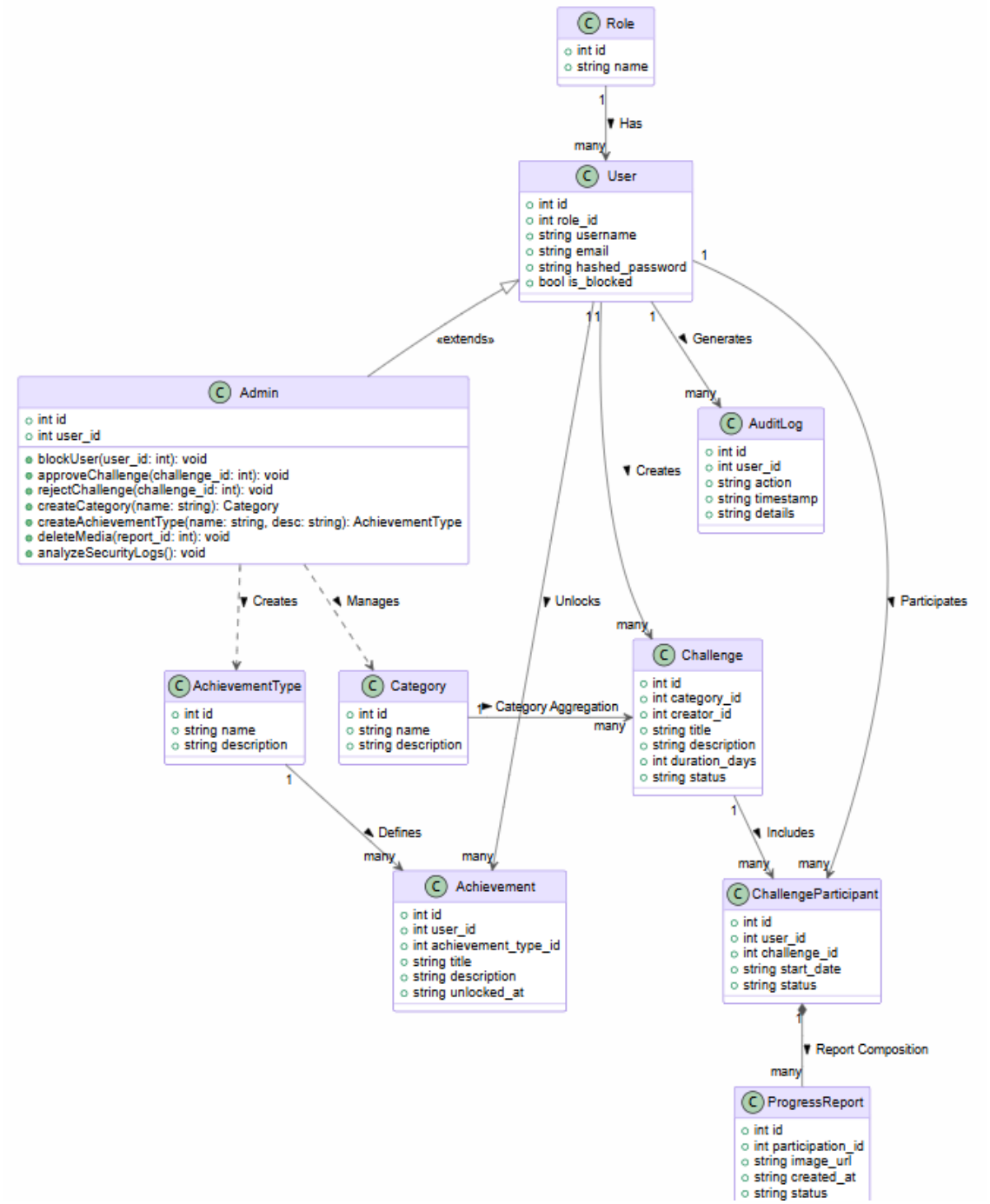

Рисунок 3.3 – Діаграма класів

Класи системи побудовано таким чином, щоб повноцінно забезпечити відповідність усім заданим функціональним вимогам застосунку. На схемі використано різні типи зв'язків між класами. Наявна композиція між записом про активність та звітами, оскільки звіт не може існувати окремо без фактичної участі користувача у конкретному виклику. Також реалізовано агрегацію між категоріями та челенджами. Кожен змодельований клас містить виключно

необхідні характеристики, для яких було визначено відповідні суворі типи даних, що зменшує ймовірність виникнення помилок.

3.1.4 Діаграма діяльності

Діаграма діяльності демонструє детальний алгоритмічний потік операцій, який потрібен для реалізації певного бізнес-процесу в системі. Наприклад, вона чудово ілюструє процес створення та збереження нового челенджу в базі даних. Ця схема відображає послідовність кроків, логічні розгалуження умов, паралельні шляхи виконання та моменти синхронізації. Це дає змогу аналітику дуже легко виявити залежності та потенційні вузькі місця.

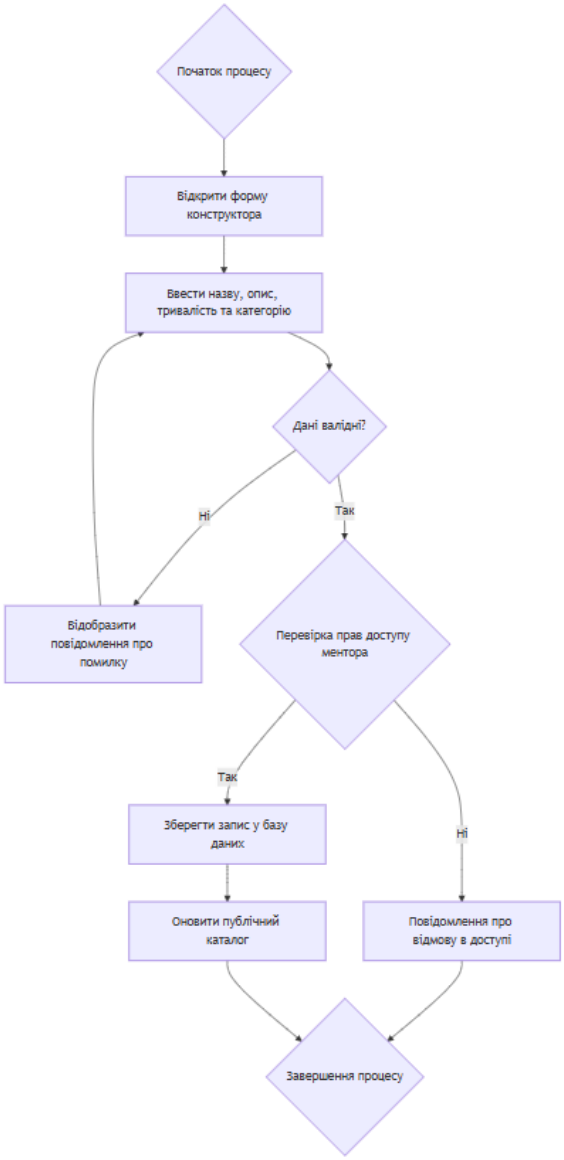

Рисунок 3.4 – Діаграма діяльності "Створення нового челенджу"

Ментор запускає форму конструювання, вписує назву, докладний опис, задає тривалість та тип майбутньої програми. На цьому етапі система здійснює ретельну перевірку коректності внесених даних. Якщо дані правильні, процес успішно переходить до перевірки дозволів поточного користувача. У випадку помилки користувачеві негайно виводиться сповіщення з пропозицією внести виправлення у введену інформацію. Тому в схемі застосовано кілька вузлів для прийняття рішень, які ефективно керують переміщенням між головними станами системи.

3.1.5 Діаграма компонентів та розгортання

Схема компонентів дуже детально показує логічне розділення системи на цілком автономні модулі. Основними архітектурними вузлами є клієнтський інтерфейс, створений на React, серверна бізнес-логіка, побудована на FastAPI, та окремий шар доступу до даних, реалізований через SQLAlchemy. Вона встановлює чіткі інтерфейси, через які ці компоненти взаємодіють між собою. Це дає змогу розробникам завчасно оцінити ступінь впливу майбутніх змін у коді та визначити оптимальні межі для можливого мікросервісного підходу. Завдяки цій схемі команда може впевнено планувати поетапний рефакторинг проєкту.

Діаграма компонентів ілюструє логіку розбиття системи на незалежні частини: клієнтську оболонку на React, серверу бізнес-логіку на FastAPI та шар роботи з даними SQLAlchemy. Визначені ясні інтерфейси взаємодії між частинами дають змогу попередньо оцінити рівень впливу прийдешніх змін і запланувати поступове покращення проєкту.

Відображений послідовний шлях обробки запитів показує початок взаємодії через HTTP REST, де серверний рівень втілює послідовну передачу завдань: від відповідного розділу FastAPI до служб бізнес-логіки та кінцевого рівня ORM SQLAlchemy, котрий перетворює сутності на запити до бази даних PostgreSQL. Така багатоступенева структура гарантує повне приховування даних, де кожна частина відповідає за свій етап обробки, що зменшує залежність та помітно полегшує налагодження всієї конструкції.

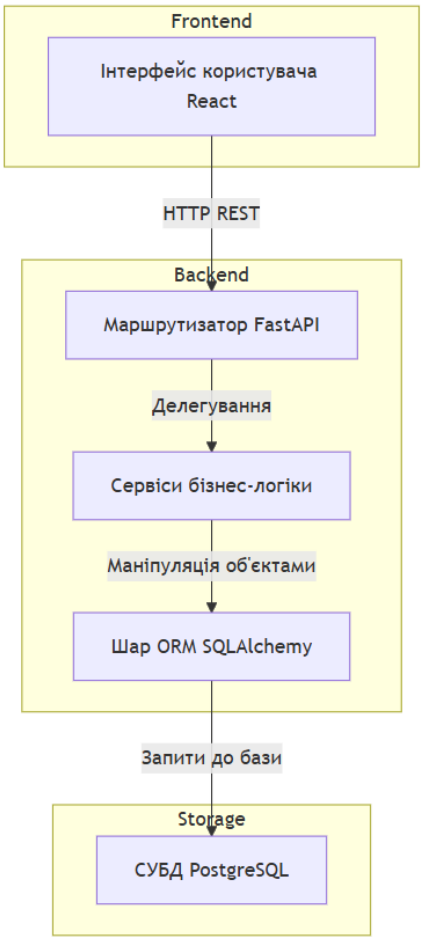

Рисунок 3.5 – Діаграма компонентів

Діаграма розгортання показує фактичне фізичне розміщення всіх елементів програми у промисловому мережевому оточенні. На стороні клієнта виступає звичайна робоча станція або мобільний телефон із встановленим сучасним веббраузером, який отримує інтерфейс і надсилає мережеві запити. У центрі схеми знаходиться потужний вебсервер, де розміщено сервісну обробку даних. За надійне зберігання даних відповідає абсолютно окремий сервер реляційної бази даних PostgreSQL.

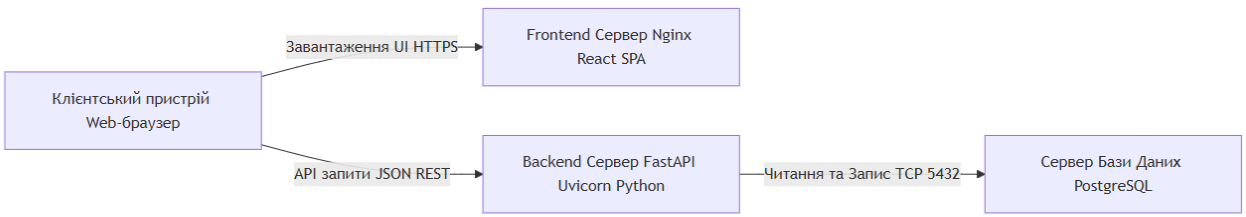

Рисунок 3.6 – Діаграма розгортання

Зв'язок між браузером користувача та вебсервером відбувається виключно через захищений криптографічний канал HTTPS. Внутрішній трафік між застосунком та сервером бази даних здійснюється в ізольованому мережевому сегменті. Така конфігурація демонструє класичний та перевірений часом тришаровий розподіл. Це забезпечує безперервну та стабільну роботу застосунку від моменту клієнтського запиту до успішного збереження інформації у постійне сховище.

3.2 Варіанти використання системи

Розділ варіантів використання описує взаємодію зовнішніх суб'єктів із платформою через приклади Use Cases, які фіксують очікувану поведінку системи без деталей технічної реалізації. Кожен приклад задає послідовність кроків для досягнення цільового результату, чітко визначаючи функціональні межі проєкту. Основні суб'єкти платформи: гість, учасник, ментор та адміністратор – взаємодіють із системою через веббраузер. Вони ініціюють ключові дії, зокрема реєстрацію облікового запису (таблиця 3.2), конструювання викликів (таблиця 3.3), відправку фотозвітів та модерацію контенту. Змодельовані приклади слугують надійною базою для деталізації бізнес-вимог, побудови тест-кейсів та проєктування внутрішньої логіки серверних сервісів.

Таблиця 3.2 – Реєстрація користувача та налаштування цілей

Use Case Section	**Comment**
Use Case Name	Зареєструвати нового користувача та налаштувати цілі
Scope	SaaS-платформа для створення та відстеження челенджів
Level	Успішно створити персональний акаунт та підготувати систему до трекінгу.
Primary Actor	Користувач

Кінець таблиці 3.2

Stakeholders and interests	Користувач – отримання інструменту для саморозвитку та відстеження прогресу. Головний адмін – залучення активної аудиторії.
Preconditions	Користувач має доступ до Інтернету та діючу електронну пошту.
Success guarantee	Створено запис в БД users та user_preferences. Користувач авторизований, обрано початкові категорії інтересів.
Main Success Scenario	1) користувач відкриває сторінку реєстрації; 2) користувач вводить нікнейм, email, пароль; 3) користувач натискає кнопку «зареєструватися»; 4) система створює профіль користувача та видає JWT-токен; 5) користувач переходить у розділ «налаштування профілю»; 6) користувач обирає пріоритетні категорії челенджів (напр. Спорт, Навчання); 7) система зберігає персональні налаштування.
Extensions	Користувач вже існує: 1) користувач вводить дані; 2) система виявляє, що email вже зареєстрований; 3) система видає повідомлення про помилку «користувач з таким email вже існує»; 4) користувач вводить інші дані або відновлює пароль.
Special Requirements	Пароль має бути хешований. Захищене з'єднання.
Technology and Data Variations List	Веб-інтерфейс, реляційна БД (PostgreSQL), JWT-авторизація.
Frequency of Occurrence	100%
Miscellaneous	Чи потрібно впроваджувати реєстрацію через Google/Apple ID?

Після успішного завершення процесу реєстрації гість одразу отримує статус авторизованого користувача та базову роль у системі, що надає можливість повноцінного відстеження звичок. Усі введені дані обов'язково перевіряються на сервері, де пароль проходить криптографічну обробку перед збереженням у базу даних PostgreSQL. Ретельна перевірка електронної пошти на стороні бекенду забезпечує унікальність адреси та актуальність профілю. Якщо введений email вже був використаний раніше, програмний інтерфейс безпечно зупиняє реєстрацію, надаючи відповідне текстове сповіщення з пропозицією здійснити авторизацію або відновити забутий пароль.

Таблиця 3.3 – Створення нового челенджу

Use Case Section	**Comment**
Use Case Name	Створити новий челендж
Scope	SaaS-платформа для створення та відстеження челенджів (Модуль «Конструктор»)
Level	Створити цифровий запис про челендж із налаштуваннями та обкладинкою в базі даних.
Primary Actor	Користувач
Stakeholders and interests	Користувач – планування діяльності та візуалізація мети. Підписники (якщо челендж публічний) – перегляд інформації про новий виклик.
Preconditions	Користувач авторизований. Система має доступ до хмарного сховища для завантаження зображень.
Success guarantee	Запис створено в таблиці challenges, фото-обкладинку завантажено в хмару, URL фото збережено в БД.

Кінець таблиці 3.3

Main Success Scenario **Main Success Scenario**	1) користувач натискає кнопку «Створити челендж»; 2) користувач заповнює поля (назва, категорія, термін виконання, опис цілі); 3) користувач обирає файл зображення для обкладинки челенджу; 4) користувач натискає «Зберегти»; 5) система відправляє фото у хмарне сховище; 6) система зберігає дані в БД та оновлює список активних челенджів у профілі.
Extensions	**Некоректний файл зображення:** 1) користувач завантажує файл > 5Мб або невідповідного формату; 2) система видає помилку валідації; 3) запис не створюється до виправлення помилки. **Незаповнені обов'язкові поля:** 1) користувач не ввів назву або термін; 2) система підсвічує відповідні поля червоним кольором.
Special Requirements	Підтримка форматів зображень JPG, PNG, WEBP.
Technology and Data Variations List	Хмарне сховище зображень, інтерактивна веб-форма, REST API.
Frequency of Occurrence	50%
Miscellaneous	—

Успішне створення нового макета виклику запускає його негайне відображення у глобальному публічному каталозі платформи, роблячи його повністю доступним для пошуку та приєднання іншими учасниками. Всі текстові дані, включаючи назву, детальний опис правил та тривалість програми, проходять ретельну типізацію та перевірку на стороні сервера за допомогою

спеціальних схем валідації. У разі виникнення будь-яких логічних помилок під час заповнення форми, наприклад встановлення нульової тривалості або пропуску обов'язкового поля назви, система безпечно скасовує операцію запису до бази даних. Замість цього користувачеві надається чітке повідомлення про помилку валідації, що дозволяє швидко виправити введену інформацію та успішно повторити спробу публікації без жодної втрати набраного тексту.

3.3 Проєктування архітектури та вибір технологічного стеку

Після формалізації функціональних вимог архітектура платформи «Конструктор особистих челенджів» була спроєктована за принципом чистого тришарового поділу: клієнтський інтерфейс, сервісний рівень та рівень зберігання даних. Такий підхід дає змогу мінімізувати зв'язування між модулями, спростити тестування та забезпечити гнучке масштабування окремих рівнів у разі зростання навантаження.

Вибір інструментів програмування ґрунтується на вимогах високої продуктивності та надійності в усіх частинах проєкту. Саме тому для бекенду обрано мову Python та мікрофреймворк FastAPI, що дозволяє розробляти високопродуктивні вебзастосунки та програмні інтерфейси з мінімальними затримками. Для розробки фронтенду використовується бібліотека React, яка забезпечує гнучку архітектуру односторінкових застосунків. Для сховища даних застосовано реляційну систему PostgreSQL версії 15. Ця система управління базами даних підтримує транзакції за стандартом ACID та складні запити, гарантуючи високу продуктивність і надійну оптимізацію.

При виборі компонентів розробник керувався критеріями зрілості, активності спільноти та простоти заміни. На клієнті React доповнюється інструментами управління глобальним станом та фреймворком Tailwind CSS для швидкої стилізації. На сервері FastAPI пропонує зручну систему впровадження залежностей та модульність, а швидку обробку запитів забезпечує сервер Uvicorn. Доступ до бази даних реалізовано через систему об'єктно-реляційного

відображення SQLAlchemy, що автоматично генерує міграції та синхронізує схему під час розробки.

Таблиця 3.4 – Порівняльний аналіз функціональних можливостей аналогів

Категорія	**Обраний компонент**	**Причина вибору та роль у проєкті**
Інтерфейсна бібліотека	React 18	Компонентна модель та підтримка односторінкових застосунків
Стан керування	Zustand	Централізований кеш та керування станом клієнта
Стилізація інтерфейсу	Tailwind CSS	Утилітарні класи для гнучкого макетування сторінок
Фреймворк бекенду	FastAPI	Надвисока продуктивність та генерація документації OpenAPI
Сервер додатків	Uvicorn	Швидка обробка мережевих запитів за стандартом ASGI
Шар доступу до даних	SQLAlchemy 2.0	Декларативні моделі та автоматичні міграції бази даних
База даних	PostgreSQL 15	Підтримка ACID та надійне зберігання інформації
Валідація даних	Pydantic	Строга типізація та перевірка даних без ручних втручань

Вибрана комбінація технологій та бібліотек формує цілісну, але ненавантажену інфраструктуру. Вона повністю покриває базові потреби першого релізу платформи та залишає широкі можливості для безболісного додавання нових модулів, наприклад системи рекомендацій чи черги сповіщень, без радикальної зміни початкового дизайну.

Додатково до наведеного стеку особливу увагу приділено шаблонам організації бізнес-логіки. У внутрішніх сервісах застосовується підхід

використання ізольованих об'єктів передачі даних. Це полегшує оптимізацію складних вибірок, наприклад агрегації статистики успішності, без загрози порушити транзакційну цілісність у сценаріях запису. Валідацію вхідних даних повністю реалізовано через бібліотеку Pydantic. Таким чином, формат об’єктів автоматично перевіряється на рівні маршрутизатора, а бізнес-сервіси отримують уже гарантовано коректну інформацію.

Для оброблення помилок використовується централізований механізм глобальних обробників виключень FastAPI. Він перехоплює винятки з будь-якого шару, перетворює їх на стандартизовані коди відповідей протоколу HTTP й формує єдину структуру відповіді. Це суттєво спрощує клієнтську обробку, адже React отримує передбачуваний формат незалежно від джерела проблеми: чи то база даних, чи помилка авторизації, чи збій стороннього сервісу.

Тестування будується за багаторівневою моделлю. Інструмент Pytest покриває сервіси та контролери бекенду, а інструменти екосистеми React перевіряють візуальні компоненти безпосередньо у браузері. Така стратифікація дає змогу швидко локалізувати помилки на ранніх стадіях і водночас зберегти прийнятний час виконання автоматичних перевірок. Усі тести можуть запускатися ізольовано, тому якість коду перевіряється перед кожним оновленням системи.

Якщо навантаження почне перевищувати можливості одного сервера, конфігурація готова до простого горизонтального масштабування шляхом додавання ще одного вузла обробки запитів та підключення резервної репліки PostgreSQL. У найпростішому варіанті це реалізується за лічені години без зміни коду, оскільки всі змінні середовища зберігаються у безпечних конфігураційних файлах і зчитуються під час старту процесу.

Під час вибору технологій було відхилено деякі альтернативні рішення. Наприклад, нереляційна база даних MongoDB розглядалася як кандидат для зберігання каталогу челенджів, але відсутність надійних транзакцій на рівні кількох документів ускладнює цілісність ігрової статистики учасників. Аналогічно, використання повноцінного оркестратора Kubernetes відкладено до

етапу, коли обсяг трафіку потребуватиме автоматичного масштабування. Наразі гнучкості стандартних інструментів контейнеризації більш ніж достатньо.

Таким чином, обрана архітектура лишається свідомо лаконічною. Вона покриває поточні бізнес-потреби платформи «Конструктор особистих челенджів» та не прив'язує команду розробників до надмірної інфраструктурної складності. У разі розширення функціоналу чи стрімкого зростання аудиторії система природно еволюціонує. Додавання нового мікросервісу, системи кешування або черги повідомлень відбуватиметься без руйнації вже сформованого коду і без зміни основних принципів проєкту.

3.4 Мокапи інтерфейсів

Для візуалізації основних екранів платформи на ранньому етапі було створено кілька низькодеталізованих та середньодеталізованих макетів. Вони слугують прототипами, які дозволяють швидко протестувати логіку розміщення елементів, сценарії навігації та відповідність дизайну сучасним стандартам доступності вебконтенту, не витрачаючи ресурсів на повну програмну верстку. Кожен дизайн ретельно узгоджено з майбутніми компонентами бібліотеки React, тому під час реалізації кожен блок інтерфейсу користувача перетворюється на відповідний функціональний компонент без необхідності повторного проєктування.

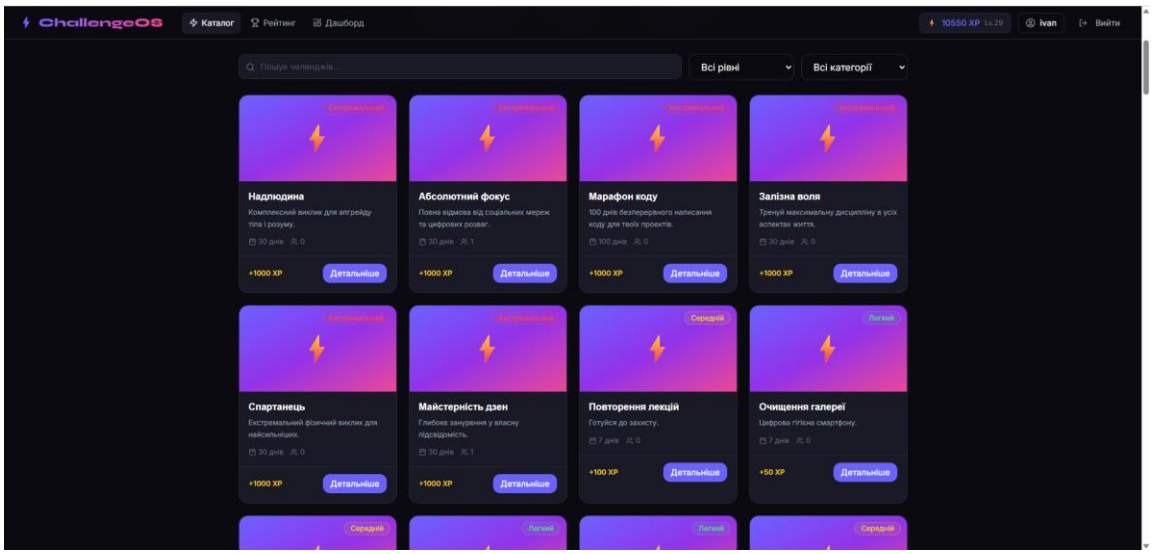

Рисунок 3.7 – Мокап інтерфейсу публічної вітрини

На сторінці каталогу відображається перелік доступних програм саморозвитку у вигляді інтерактивної сітки. Кожна картка містить обкладинку виклику, його назву, рівень складності та повну тривалість. У верхній частині розміщено рядок пошуку та фільтри для швидкого доступу до необхідної активності.

Функціональні елементи сторінки:

— заголовок сторінки та панель навігації;
— поле текстового пошуку та фільтри за категоріями;
— сітка карток програм;
— картка челенджу із зображенням, назвою та кнопкою перегляду.

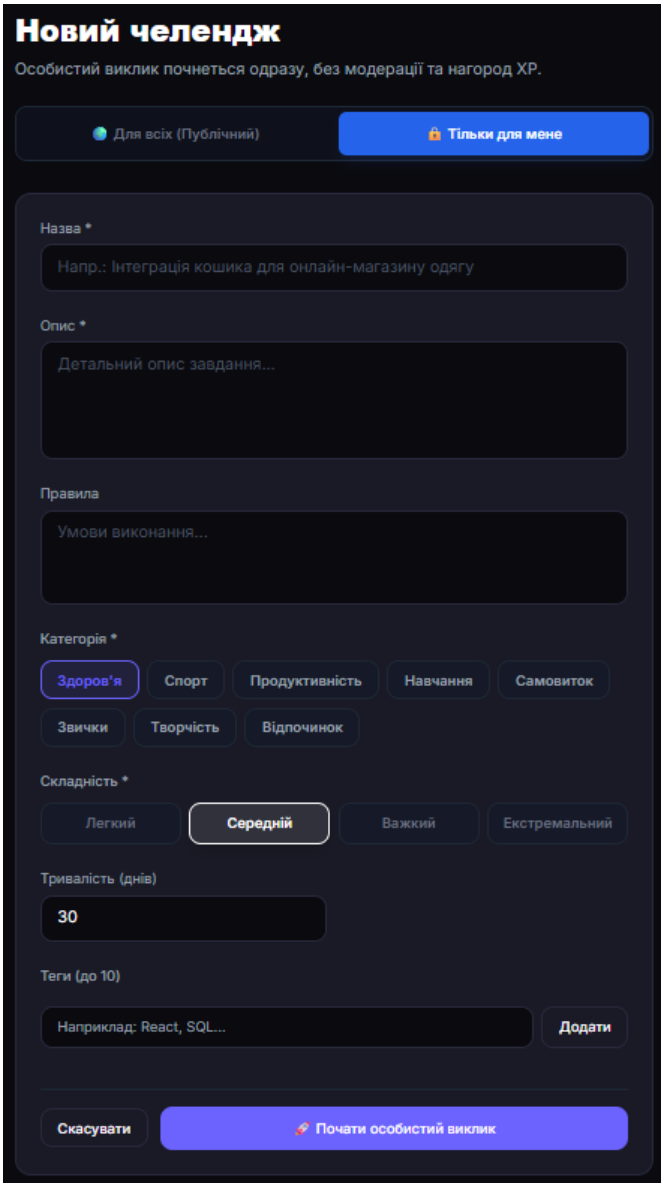

Рисунок 3.8 – Мокап інтерфейсу конструктора челенджів

Сторінка створення нового виклику містить зручний інтерфейс для введення налаштувань майбутньої програми. Користувач із правами ментора може вказати назву, детальний опис правил, обрати категорію та встановити тривалість у днях. Внизу розташована кнопка для збереження та публікації макета у загальний каталог. Функціональні елементи:

— поля для введення назви та опису правил;
— випадаючі списки для вибору категорії та рівня складності;
— лічильник кількості днів тривалості;
— кнопка збереження та публікації.

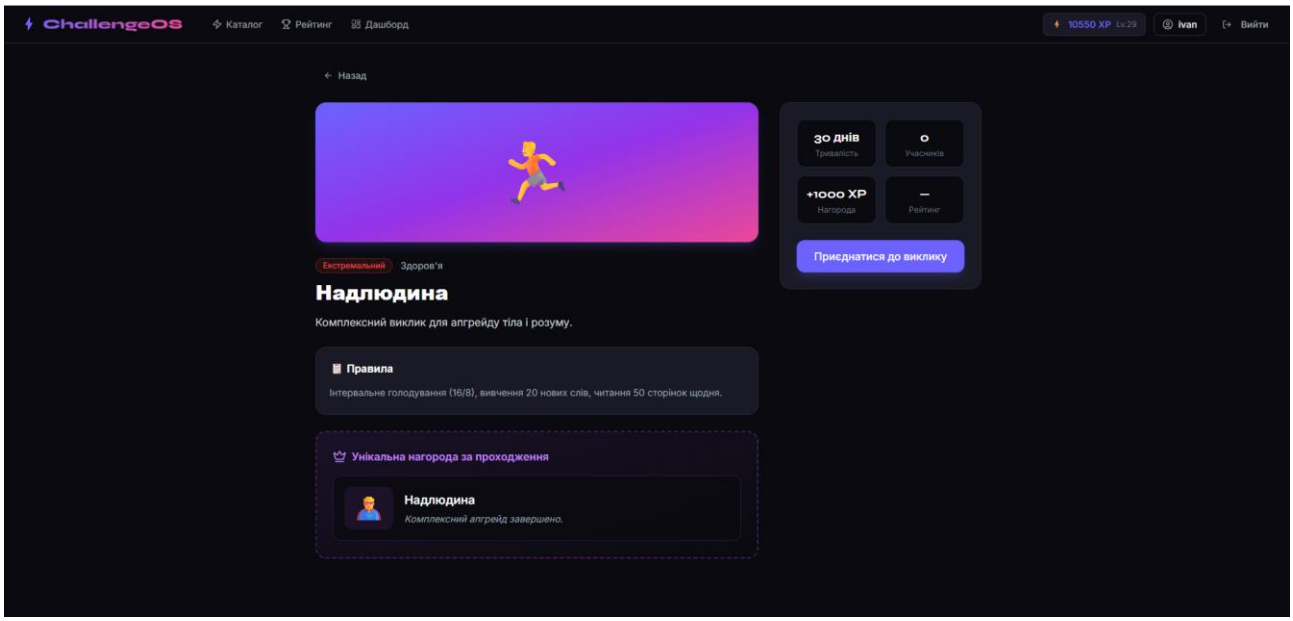

Рисунок 3.9 – Мокап інтерфейсу сторінки деталей челенджу

Сторінка виклику містить обкладинку, опис цілей, статистику учасників та бали досвіду. Мінімалістичний дизайн зосереджений на мотивації користувача приєднатися до програми за допомогою виділеної кнопки.

Функціональні елементи:

— зображення обкладинки челенджу;
— назва та мотиваційний опис;
— блок статистики із кількістю учасників та балами досвіду;
— велика кнопка приєднання до виклику.

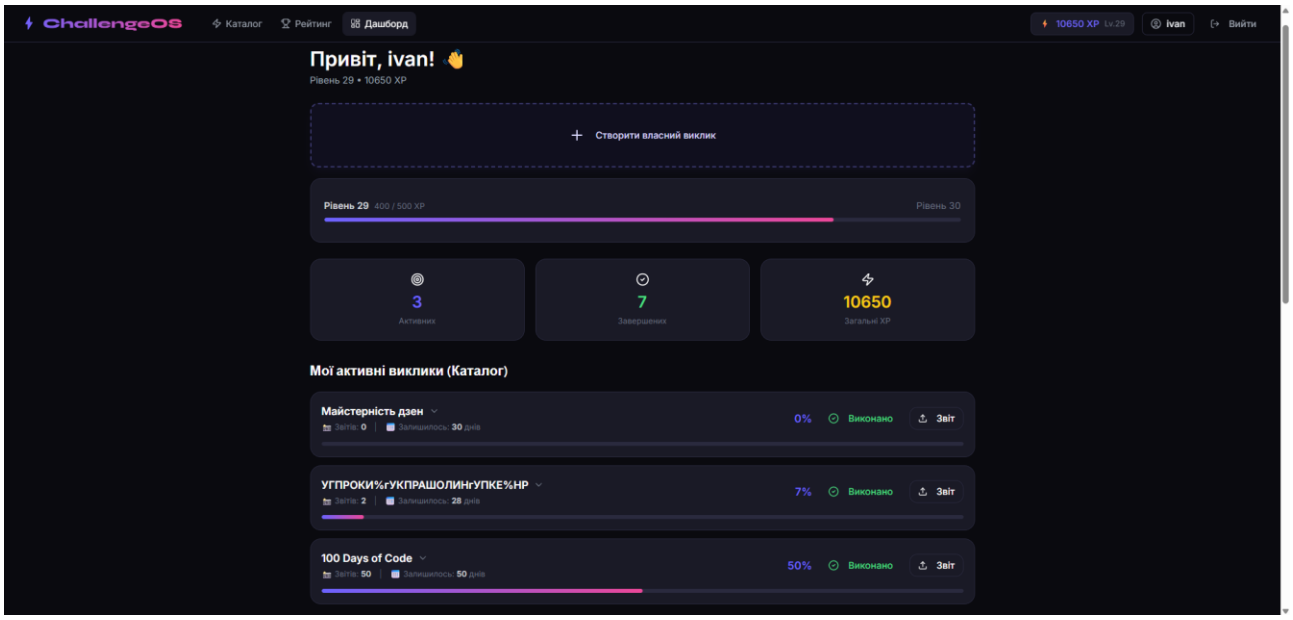

Рисунок 3.10 – Мокап інтерфейсу персонального дашборду

Головний екран авторизованого користувача демонструє зведену статистику його особистого прогресу. Тут показано поточний ігровий рівень, кількість накопичених балів досвіду та безперервну серію днів активності. Нижче розташовано інтерактивний блок для швидкого завантаження щоденного фотозвіту за активними викликами.

Окрім статистичних даних, інтерфейс екрана містить стрічку активних челенджів, що дає змогу переглянути деталі завдань та прогрес. Візуалізація успіхів представлена інтуїтивними шкалами, які оновлюються після підтвердження звітів на сервері. Така побудова елементів створює відчуття руху вперед та підсилює мотивацію через наочне відображення прогресу до наступного рангу чи винагороди.

Функціональні елементи:

— панель статистики з рівнем та балами досвіду;
— індикатор серії безперервних днів;
— список активних викликів користувача;
— форма для завантаження щоденного фотодоказу.

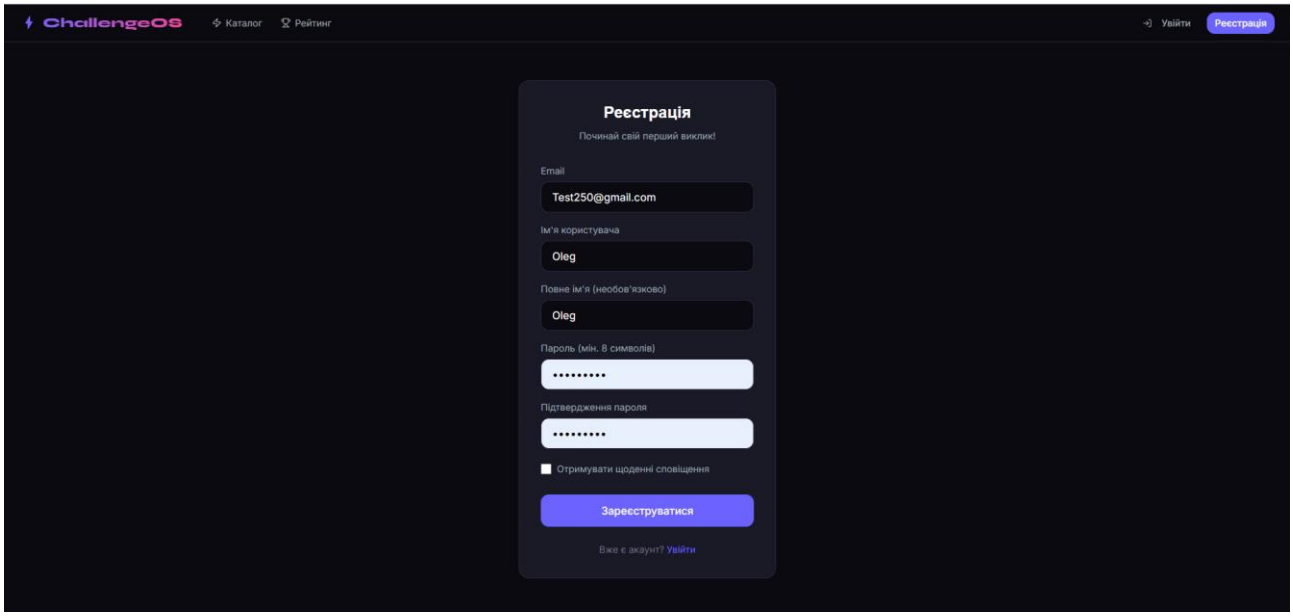

Рисунок 3.11 – Мокап інтерфейсу створення нового профілю користувача

Сторінка автентифікації містить дві окремі вкладки для входу та створення нового профілю. У вкладці входу є поля для введення пошти та пароля, кнопка для входу, а також посилання для відновлення втраченого доступу. Вкладка реєстрації дозволяє створити новий акаунт, ввівши базові контактні дані та двічі підтвердивши надійний пароль.

Функціональні елементи:

— вкладки перемикання між входом та реєстрацією;
— текстові поля введення електронної пошти та пароля;
— поле підтвердження пароля для нових користувачів;
— головна кнопка підтвердження дії;
— посилання на сторінку відновлення доступу.

Висновки до розділу 3

Таким чином, у третьому розділі сформовано цілісне бачення програмної архітектури розробленої вебплатформи та спосіб її поетапного втілення. Обрано тришаровий мінімалістичний стек на базі React з TypeScript для клієнтської частини, FastAPI з Python для серверного ядра та PostgreSQL для зберігання даних, що забезпечує ідеальний баланс між швидким стартом розробки й

можливістю подальшого масштабування. Для формалізації прийнятих інженерних рішень розроблено вичерпний набір UML-діаграм, що охоплюють усі рівні абстракції від прецедентів користувача до фізичного розгортання на серверах. Таблиці технологій та компонентів наочно демонструють прозорий розподіл відповідальностей і суттєво спрощують подальше технічне обслуговування. Передбачено використання сучасних декларативних фреймворків, що значно полегшить інтеграцію сторонніх сервісів у майбутньому.

Передусім у системі реалізуються базові сценарії, такі як реєстрація та авторизація користувачів, перегляд публічної вітрини, конструювання нових викликів та завантаження щоденних фотозвітів, але сама структура цілком готова до безболісного додавання додаткових мікросервісів та модулів аналітики. Усі обрані програмні інструменти мають надзвичайно широку підтримку світової IT-спільноти, тому ризики технологічного глухого кута є мінімальними. У результаті третій розділ закладає дуже міцну методологічну й технічну основу, на якій може впевнено розбудовуватися весь подальший життєвий цикл програмного проєкту.

Також спроєктована архітектура беззаперечно сприяє підвищенню загальної надійності та безпеки застосунку завдяки чіткому розмежуванню функціональних модулів і суворому використанню сучасних стандартів захисту даних. Застосування статичної типізації за допомогою TypeScript та строгих схем Pydantic забезпечує надійний контроль вхідних потоків інформації, що кардинально зменшує кількість помилок на ранніх етапах розробки. З іншого боку, інтеграція мікрофреймворку FastAPI та об'єктно-реляційної системи SQLAlchemy дозволяє впроваджувати високопродуктивні асинхронні рішення з урахуванням найкращих практик індустрії. Таким чином, розроблена програмна архітектура не лише повністю відповідає поточним вимогам проєкту, але й залишається максимально гнучкою для адаптації до майбутніх технологічних викликів і безперервного розвитку бізнес-логіки.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

Практичний етап передбачає розробку розподіленого вебзастосунку, де клієнтська частина повністю відокремлена від серверного ядра. Логіка платформи базується на гейміфікації та відстеженні прогресу для трьох категорій користувачів. Учасники фіксують свої досягнення через фотозвіти, наставники конструюють авторські програми та керують ними, а адміністратори контролюють цілісність бази даних і модерують контент. Такий розподіл обов'язків гарантує високий захист, надійне збереження даних та гнучкість управління всією екосистемою.

4.1 Організація інтерфейсу користувача та макети сторінок

Для детальної візуалізації взаємодії користувача з елементами системи на етапі проєктування було створено серію інтерактивних макетів сторінок. Основне завдання цих прототипів полягало у перевірці зручності навігації та оптимізації розміщення графічних блоків до моменту написання реального коду верстки. Для неавторизованих відвідувачів початковою точкою входу виступає головна сторінка, на якій розміщено базові відомості про можливості платформи та форми для реєстрації облікового запису.

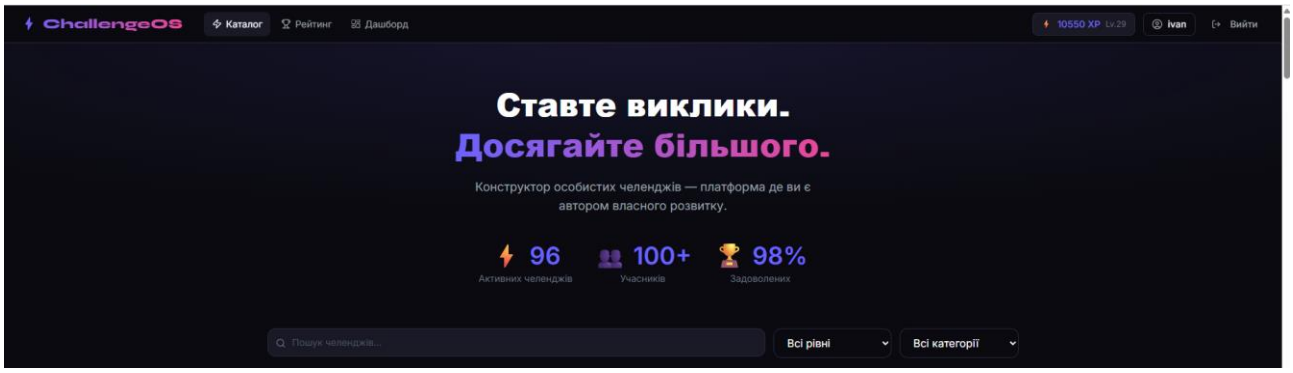

Рисунок 4.1 – Макети головного екрана платформи

На початку сторінки користувачу представлені блоки з коротким описом логіки роботи челенджів, а також інтерактивні елементи для швидкого переходу

до реєстрації. Дизайн цієї частини програми зроблено у мінімалістичному стилі для підвищення конверсії нових користувачів системи.

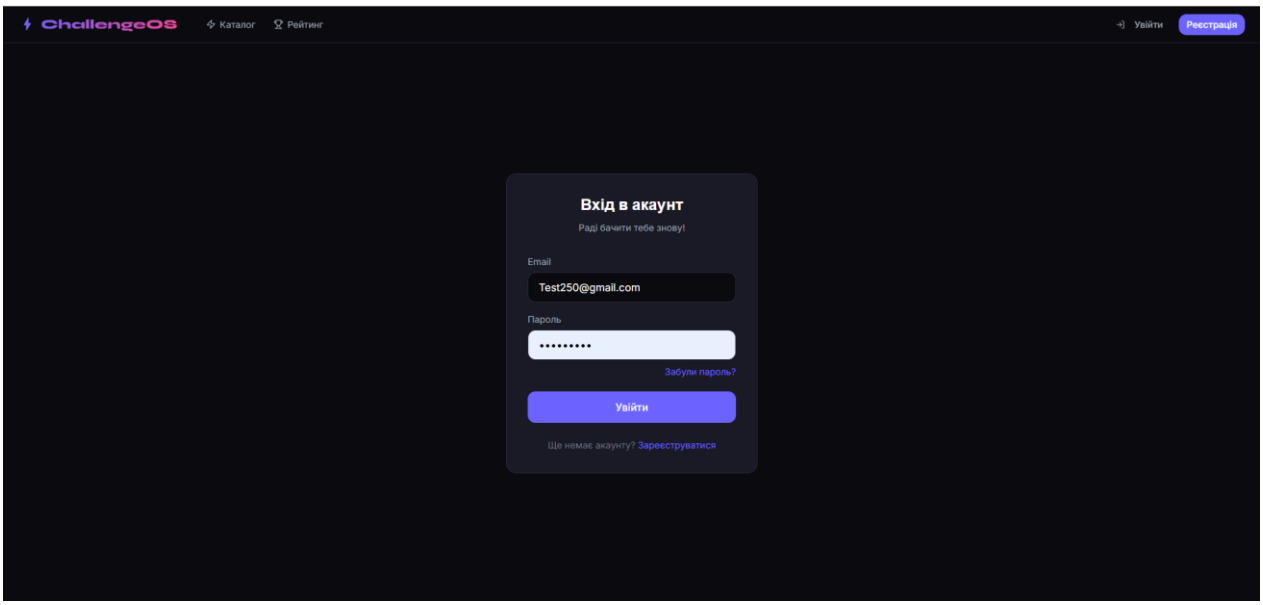

Рисунок 4.2 – Інтерфейс авторизації користувача

Форма реєстрації максимально спрощена: користувач вводить унікальний нікнейм, пошту, пароль та за бажанням активує щоденні сповіщення. Інтерактивний логотип дозволяє швидко повернутися на головну сторінку. Після успішної авторизації система генерує динамічне бічне меню навігації для швидкого доступу до персонального дашборда зі статистикою, публічної вітрини завдань, панелі конструктора та сторінки здобутих досягнень.

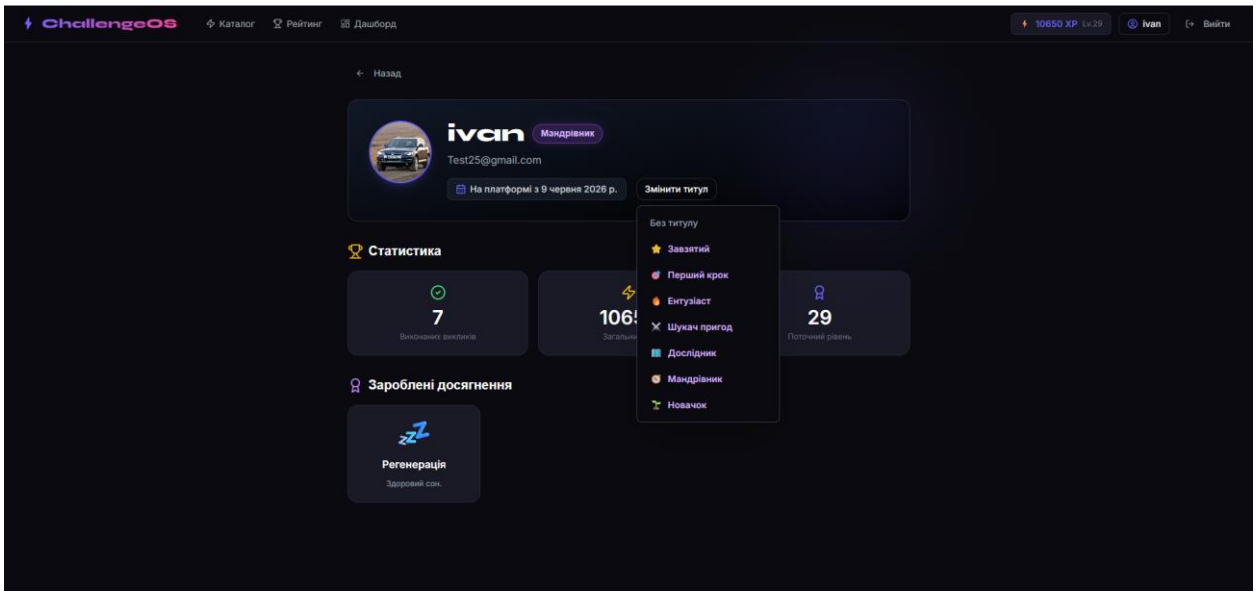

Рисунок 4.3 – Головна панель навігації авторизованого учасника

Персональна панель показує прогрес за активними викликами, візуальні індикатори успішності, лічильники серій діяльності та модуль швидкого завантаження фотозвітів. Така інтеграція робочого простору дозволяє миттєво оцінювати стан виконання завдань, значно оптимізує час на звітність та стимулює регулярне використання платформи.

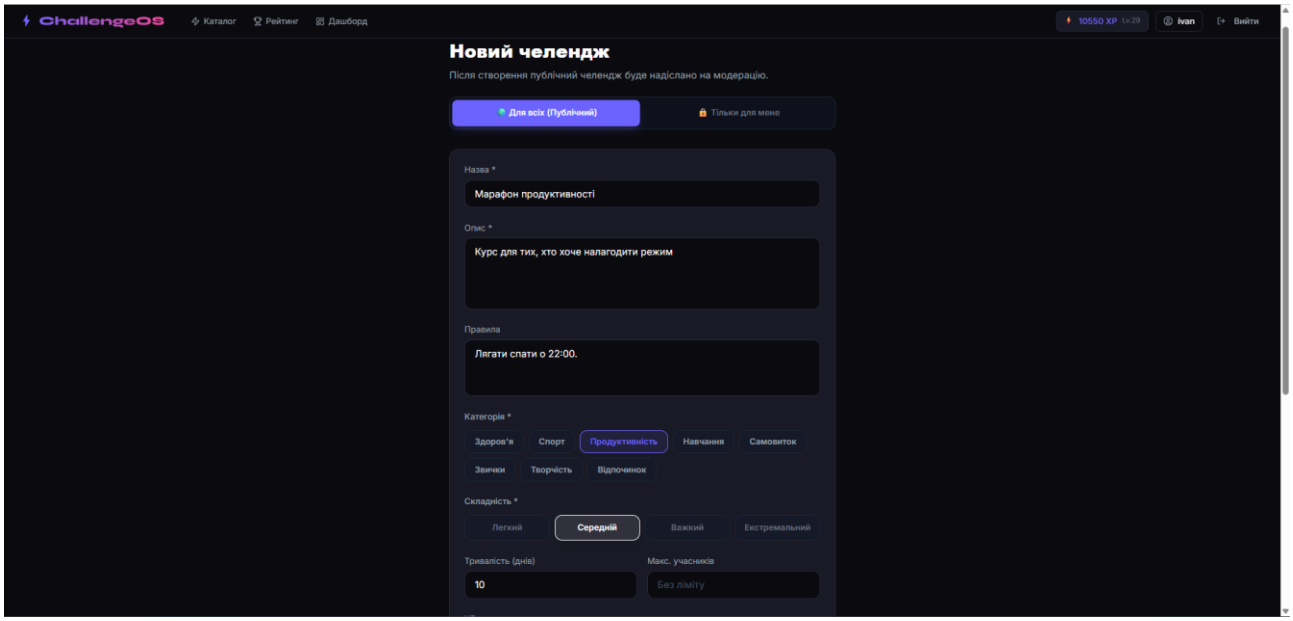

Рисунок 4.4 – Інтерфейс додавання програми в конструкторі

Панель розробника надає наставникам зручну форму для створення викликів, що містить обов'язкові текстові поля для назви й опису правил, цифровий лічильник тривалості у днях та випадаючий список для вибору тематичної категорії.

4.2 Реалізація серверної частини та взаємодія з базою даних

Створення серверної інфраструктури ґрунтується на засадах високої продуктивності та безпеки, де основою є мікрофреймворк FastAPI, що забезпечує ефективну асинхронну обробку запитів, а взаємодія з реляційною СУБД PostgreSQL реалізована через ORM-систему SQLAlchemy, яка гарантує цілісність транзакцій даних та надійне управління бізнес-логікою.

4.2.1 Асинхронна обробка запитів та автентифікація

Бекенд платформи розроблено на Python та мікрофреймворку FastAPI для досягнення швидкої асинхронної обробки запитів. Захист реалізовано за допомогою безсесійних токенів доступу та криптографічного хешування паролів методом BCrypt. За маршрутизацію, перевірку дозволів і валідацію відповідають компоненти серверного ядра, де вхідні дані автоматично аналізуються схемами Pydantic, що повністю унеможливлює передачу невірних даних до бази.

```
28
29  @router.post("/register", response_model=TokenResponse, status_code=201)
30  async def register(
31      data: UserRegister, request: Request, db: AsyncSession = Depends(get_db)
32  ):
33      # Автоматична генерація username, якщо фронтенд його не передав
34      if not getattr(data, "username", None):
35          base_username = data.email.split("@")[0]
36          data.username = base_username
37
38      if await get_user_by_email(db, data.email):
39          raise HTTPException(400, "Цей email вже зареєстровано")
40
41      existing = (await db.execute(
42          select(User).where(User.username == data.username)
43      )).scalar_one_or_none()
44
45      # Якщо згенерований юзернейм вже зайнятий, додаємо випадкові цифри
46      if existing:
47          data.username = f"{data.username}{random.randint(1000, 9999)}"
48
49      try:
50          user = await create_user(db, data)
51      except ValueError as e:
52          raise HTTPException(400, str(e))
53
54      # Підтягуємо role_ref для відповіді
55      await db.refresh(user, ["role_ref"])
56      await write_audit(db, "register", user.id, "user", user.id,
57                        request.client.host)
58
59      token = create_access_token({"sub": str(user.id)})
60      return TokenResponse(access_token=token, user=_build_user_out(user))
61
```

Рисунок 4.5 – Фрагмент програмного коду маршрутизатора реєстрації

Адаптивність користувацького інтерфейсу забезпечується допоміжними класами Tailwind CSS. На великих екранах навігаційне меню відображається повністю розгорнутим, а на мобільних пристроях автоматично трансформується

у стислу панель, що випадає, приховуючи другорядні елементи без жодної втрати основного функціоналу.

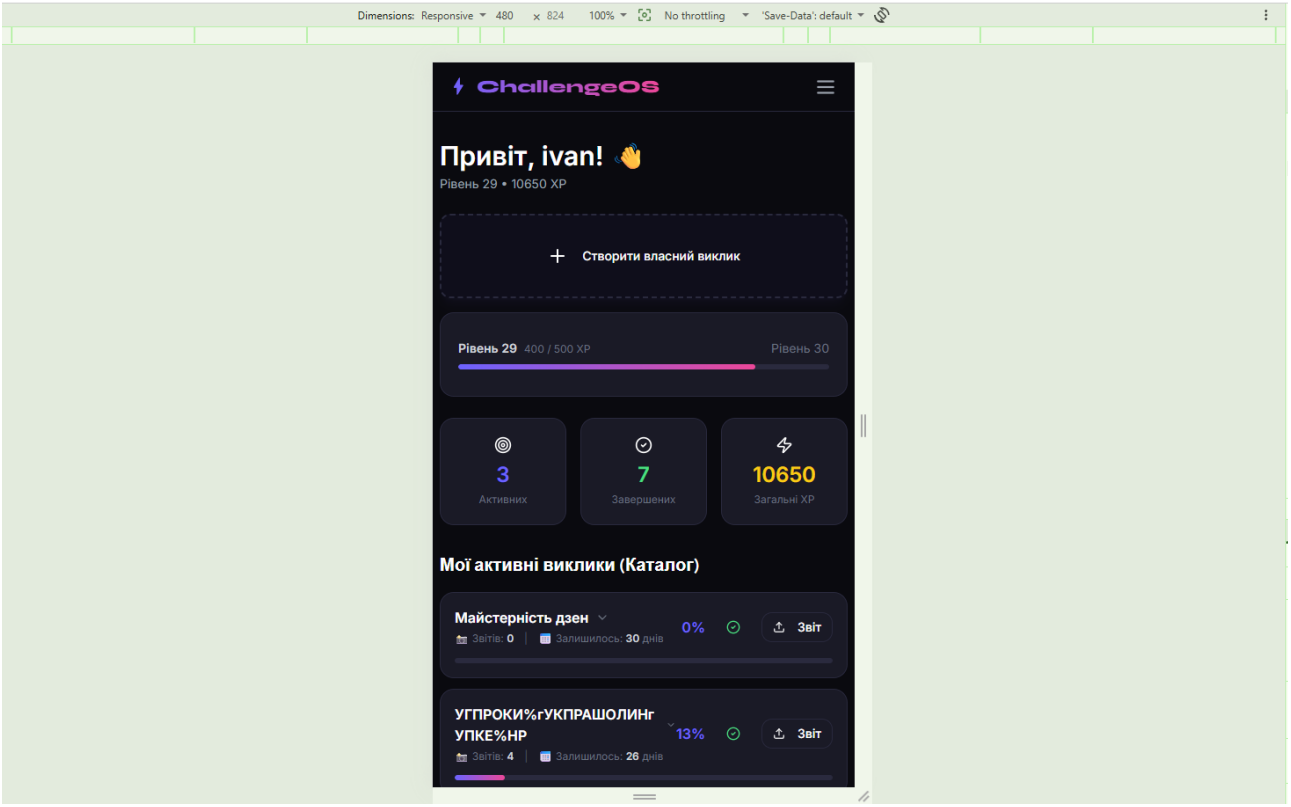

Рисунок 4.6 – Адаптивне відображення інтерфейсу на різних екранах

Така динамічна зміна візуального макета відбувається абсолютно непомітно для користувача та гарантує стабільно високий рівень зручності як при роботі за стаціонарним комп'ютером, так і при використанні смартфона.

4.2.2 Керування челенджами та фіксація прогресу

Основне технологічне ядро системи відповідає за виконання базових операцій управління сутностями викликів та звітів учасників. Взаємодія з реляційною базою даних PostgreSQL реалізована через шар об'єктно-реляційного відображення SQLAlchemy шістнадцятої версії, що дозволяє виконувати складні транзакції з дотриманням принципів цілісності та ізольованості даних.

```python
    # Якщо користувач завантажив реальне фото:
    if photo and photo.filename:
        upload_dir = "uploads"
        os.makedirs(upload_dir, exist_ok=True)

        file_ext = photo.filename.split('.')[-1] if '.' in photo.filename else 'jpg'
        file_name = f"{uuid.uuid4()}.{file_ext}"
        file_path = os.path.join(upload_dir, file_name)

        with open(file_path, "wb") as buffer:
            buffer.write(await photo.read())

        photo_url = f"http://localhost:8000/uploads/{file_name}"

    # Якщо користувач просто натиснув "Виконано" (без фото):
    else:
        # Генерується красива картинка з галочкою для галереї
        photo_url = "https://ui-avatars.com/api/?name=%E2%9C%93&background=10B981&color=fff&size=400&font-size=0.6"

    try:
        # Виклик CRUD-сервісу для збереження звіту в БД з використанням транзакції
        report = await create_report(db, current_user.id, participation, photo_url, note)
    except ValueError as e:
        raise HTTPException(400, str(e))

    await write_audit(db, "submit_report", current_user.id,
                      "participation", participation_id,
                      details={"day_number": getattr(report, "day_number", 1)})

    return ReportOut.model_validate(report)
```

Рисунок 4.7 – Реалізація функцій обробки бізнес-сервісів

Під час подання запиту сервер перевіряє дані, створює запис у базі даних та оновлює кеш вітрини. Завантаження фотозвіту включає збереження медіафайлу та транзакційне оновлення досвіду користувача.

4.3 Моделювання тестових сценаріїв та аналіз надійності

Перед випуском проведено комплексне тестування безпеки та продуктивності системи [22]. Працездатність перевірялася автоматизованими сценаріями [23], які охоплюють бекенд-модулі та наскрізну взаємодію з інтерфейсом [24]. Результати перевірок подано у специфікаціях нижче [25].

Таблиця 4.1 – Тестовий сценарій процесу створення профілю

Use Case Section	**Comment**
Primary Actor	Гість
Level	Успішно створити персональний акаунт та підготувати систему до трекінгу.

Кінець таблиці 4.1

Preconditions	Користувач має стабільний доступ до мережі Інтернет та діючу електронну пошту.
Success guarantee	Створено запис у базі даних користувачів. Користувач успішно авторизований, видано токен доступу.
Main Success Scenario	1) гість відкриває сторінку реєстрації через навігаційне меню; 2) гість вводить унікальний нікнейм, електронну пошту та безпечний пароль; 3) гість натискає кнопку підтвердження реєстрації; 4) sistema перевіряє валідність введених даних через схеми безпеки; 5) система криптографічно хешує пароль за допомогою стійкого алгоритму; 6) система створює профіль користувача та генерує токен доступу; 7) гість автоматично перенаправляється на головний персональний дашборд.
Extensions	Користувач вже існує: 1) гість вводить дані у форму реєстрації; 2) система виявляє дублювання електронної пошти в реляційній базі; 3) система видає візуальне повідомлення про помилку валідації; 4) запис не створюється до моменту введення унікальної пошти.
Special Requirements	Пароль має бути обов'язково хешований перед збереженням. Захищене мережеве з'єднання за протоколом HTTPS.
Technology and Data Variations List	Візуальний веб-інтерфейс React, реляційна база PostgreSQL, асинхронний маршрутизатор FastAPI.

Наведений вище сценарій є важливим для забезпечення безпечного доступу до сервісу. Після успішної реєстрації користувач отримує доступ до можливостей конструктора, де реалізовано створення програм саморозвитку, що докладно показано у наступній таблиці. Саме цей механізм перетворює звичайного учасника на активного наставника, готового формувати власні мотиваційні спільноти.

Таблиця 4.2 – Створити новий челендж

Use Case Section	**Comment**
Primary Actor	Ментор
Level	Створити цифровий запис про програму із налаштуваннями та обкладинкою в базі даних.
Preconditions	Користувач авторизований та має підтверджені права ментора. Система має доступ до файлового сховища.
Success guarantee	Запис успішно створено в таблиці челенджів, фотографію завантажено на сервер, посилання збережено в базу.
Main Success Scenario	1) ментор відкриває форму створення челенджу на дашборді; 2) ментор заповнює назву, опис, категорію, тривалість та завантажує обкладинку; 3) ментор натискає кнопку збереження макета; 4) ментор завантажує файл зображення для обкладинки програми; 5) ментор натискає фінальну кнопку збереження макета; 6) система валідує інформацію через схеми Pydantic та оптимізує фото; 7) сервер зберігає дані в базу та миттєво оновлює публічний каталог.
Extensions	Некоректний файл зображення: 1) ментор завантажує файл розміром понад п'ять мегабайт; 2) sistema миттєво видає помилку валідації розміру на екрані; 3) запис не створюється до моменту завантаження валідного файлу. Незаповнені обов'язкові поля: 1) ментор ігнорує поле назви або тривалості; 2) система підсвічує відповідний порожній блок червоним кольором.
Special Requirements	Повна підтримка сучасних форматів зображень JPG, PNG, WEBP.
Technology and Data Variations List	Клієнтська інтерактивна веб-форма, маршрутизатор FastAPI, об'єктно-реляційне відображення SQLAlchemy.

Після створення челенджу головною метою учасника стає постійний моніторинг прогресу та засвідчення виконання завдань, що є взагалі основою мотиваційного компонента системи. Ця процедура показана у наступній таблиці.

Таблиця 4.3 – Завантажити щоденний фотозвіт виконання

Use Case Section	**Comment**
Primary Actor	Учасник
Level	Підтвердження виконання щоденного завдання та транзакційне оновлення ігрової статистики.
Preconditions	Учасник успішно приєднаний до обраного активного челенджу.
Success guarantee	Графічний файл безпечно збережено на сервері, статус поточного дня змінено на виконано, бали досвіду успішно нараховано.
Main Success Scenario	1) учасник відкриває сторінку активної програми на дашборді; 2) учасник натискає кнопку додавання щоденного візуального звіту; 3) учасник вибирає відповідну фотографію через системне вікно пристрою; 4) учасник натискає кнопку відправки даних на серверну перевірку; 5) сервер приймає запит та суворо перевіряє формат медіафайлу; 6) система транзакційно оновлює лічильник успішності користувача; 7) інтерефейс миттєво відображає оновлений графік виконання завдань.
Extensions	Невірний формат звіту: 1) учасник намагається завантажити текстовий документ замість фотографії; 2) серверна логіка валідації миттєво відхиляє мережевий запит; 3) система виводить повідомлення про абсолютно недопустимий формат файлу; 4) ігровий прогрес не зараховується до моменту правильного завантаження.

Кінець таблиці 4.3

Special Requirements	Обов'язкова програмна оптимізація розміру фотографії перед фізичним збереженням на жорсткий диск сервера.
Technology and Data Variations List	Серверна файлова система, надійні транзакції бази даних PostgreSQL, компонентна модель інтерфейсу React.

Останнім етапом взаємодії, що вимагає перевірки надійності, є механізм пошуку доступних програм. Для забезпечення відповідності результатів було розроблено систему фільтрації, специфікацію якої наведено нижче.

Таблиця 4.4 – Знайти челендж за допомогою системи фільтрів

Use Case Section	**Comment**
Primary Actor	Гість або Учасник
Level	Отримання релевантного пагінованого списку програм за заданими користувачем складними критеріями.
Preconditions	У базі даних існують коректно опубліковані менторами публічні програми.
Success guarantee	На екрані пристрою успішно відображено список карток, що стовідсотково відповідають заданим умовам пошуку.
Main Success Scenario	1) відвідувач відкриває розділ публічної вітрини через навігаційне меню; 2) відвідувач активує бічну інтерактивну панель налаштування фільтрів; 3) відвідувач обирає бажані тематичні категорії та потрібний рівень складності; 4) клієнтський додаток формує параметризований запит до серверного бекенду; 5) маршрутизатор FastAPI виконує оптимізований пошук у базі даних; 6) сервер повертає повністю відформатований список знайдених програм; 7) інтерфейс плавно оновлює сітку результатів без перезавантаження сторінки.

Кінець таблиці 4.4

Extensions	Відсутність результатів пошуку: 1) відвідувач встановлює занадто вузькі критерії пошуку на панелі; 2) сервер не знаходить жодного відповідного запису в таблицях бази; 3) клієнтський інтерфейс отримує абсолютно порожній масив даних; 4) система коректно відображає графічну заглушку замість порожнього екрана.
Special Requirements	Свідоме використання спеціальних індексів бази даних для забезпечення надвисокої швидкості обробки пошукових запитів.
Technology and Data Variations List	Менеджер глобального стану Zustand, складні запити мовою SQL, асинхронна архітектура бекенду.

Представлені сценарії охоплюють найбільш важливі вузли системи. Усі зазначені випадки, включно з обробкою непередбачених ситуацій, були успішно відтворені під час етапу програмного тестування. Це підтверджує високу функціональну готовність застосунку до експлуатації та його здатність правильно обробляти помилки введення в динамічних умовах навантаження.

Висновки до розділу 4

У четвертому розділі втілено практичний аспект розробки інноваційної вебплатформи «Конструктор особистих викликів», що стало логічним завершенням етапу архітектурного проєктування. Створено сучасну, масштабовану та надійно розподілену програмну структуру, де клієнтська складова, побудована на основі компонентного підходу бібліотеки React, повністю та безпечно відділена від серверного ядра. Серверна частина реалізована за допомогою сучасного мікрофреймворку FastAPI, який забезпечив строгу валідацію вхідних даних та високу швидкість розробки. Завдяки застосуванню технології асинхронного опрацювання мережевих запитів та об'єктно-реляційного відображення SQLAlchemy у тісній взаємодії з потужною

реляційною базою даних PostgreSQL, було досягнуто максимально високої продуктивності системи, гарантованої цілісності відомостей та стійкості до потенційних перевантажень.

З метою забезпечення найвищої якості програмного продукту та його повної відповідності технічному завданню, було проведено комплексне та багаторівневе тестування функціональних можливостей програми. Результати цього критично важливого етапу було ретельно зафіксовано у формі деталізованих сценаріїв взаємодії та тестових специфікацій. Це дозволило на практиці перевірити безпомилковість та правильність виконання всіх ключових бізнес-процесів: від безпечної реєстрації користувачів із використанням криптографічного захисту даних, безперебійного конструювання авторських програм наставниками, до стабільного завантаження мультимедійних фотозвітів та швидкого динамічного пошуку актуальних викликів через багатокритеріальну систему фільтрів.

Крім того, значним досягненням практичного етапу стало успішне запровадження адаптивного інтерфейсу користувача та інтеграція елементів гейміфікації безпосередньо в робочу область системи. Візуалізація особистих здобутків, зручна панель обліку для менторів та інтерактивний каталог завдань сприяли формуванню повноцінного мотиваційного середовища. Підбиваючи підсумки розділу, можна впевнено констатувати, що створений програмний продукт є цілісним, надійним і придатним до практичного використання рішенням, а розроблена архітектурна основа надає широкі перспективи для подальшого розширення платформи.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі проведено комплексне науково-практичне дослідження, результатом якого стало проєктування, програмна реалізація та тестування високопродуктивної вебплатформи «Конструктор особистих викликів». Створена інформаційна система успішно вирішує актуальну проблему автоматизації процесів саморозвитку та вдосконалює взаємодію між організаторами програм розвитку й учасниками, котрі прагнуть до систематичного досягнення особистих цілей.

У ході роботи було розроблено архітектурну модель системи на базі технологічного стеку React, FastAPI та PostgreSQL, яка забезпечує високу швидкодію, масштабованість та цілісність даних при значних навантаженнях. Сформовано детальну специфікацію функціональних вимог, що охоплює логіку взаємодії чотирьох типів користувачів, алгоритми автоматизованого відстеження прогресу та механізми гейміфікації. Окрім того, впроваджено комплекс архітектурних рішень із безпеки, зокрема систему надійного шифрування персональних даних та механізми автентифікації на основі криптографічних вебтокенів. Розроблені проєктні рішення, змодельовані функціональні блоки та програмні компоненти повністю відповідають завданням переддипломної практики, а отримані результати створюють міцну методологічну базу для подальшої експлуатації вебзастосунку.

Проведені дослідження та результати тестування підтверджують високу функціональну готовність платформи до впровадження. Завдяки модульній архітектурі, створеній у межах кваліфікаційної роботи, система забезпечує можливість подальшого розширення функціоналу, зокрема впровадження інтелектуальної аналітики прогресу та нових механізмів соціальної взаємодії, що гарантує її конкурентоспроможність та ефективність використання в умовах сучасного ринку цифрових інструментів для самовдосконалення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Zainuddin Z., Chu S. K. W., Shujahat M., Perera C. J. The impact of gamification on learning and instruction: A systematic review of empirical evidence. Educational Research Review. 2020. Vol. 30. P. 100326. URL: https://doi.org/10.1016/j.edurev.2020.100326 (Accessed: 02.05.2026).

2. Krath J., Schürmann L., von Korflesch H. F. Revealing the theoretical basis of gamification: A systematic review and analysis of theory in research on gamification. Computers in Human Behavior. 2021. Vol. 125. P. 106963. URL: https://doi.org/10.1016/j.chb.2021.106963 (Accessed: 03.05.2026).

3. Novo C., Zanchetta C., Goldmann E., Carvalho C. V. de. The use of gamification and web-based apps for sustainability education. Sustainability. 2024. Vol. 16, № 8. P. 3197. URL: https://doi.org/10.3390/su16083197 (Accessed: 03.05.2026).

4. Gulzar S., Ansari N. A. Understanding Usability Challenges in Educational Games Through the Lens of the Mechanics, Dynamics and Aesthetics Framework. ProScholar Insights. 2025. Vol. 4, № 1. P. 185–197. URL: https://doi.org/10.1007/s11528-024-00984-2 (Accessed: 04.05.2026).

5. Lopez C. E., Tucker C. S. The effects of player type on performance in gamified environments. Computers in Human Behavior. 2020. Vol. 108. P. 106311. URL: https://doi.org/10.1016/j.chb.2020.106311 (Accessed: 04.05.2026).

6. Inayat I., Salim S. S., Marczak S. Agile requirements engineering practices and challenges: an empirical study. Journal of Software: Evolution and Process. 2022. Vol. 34, № 2. P. e2434. URL: https://doi.org/10.1002/smr.2434 (Accessed: 04.05.2026).

7. Scott Jr E. A. SPA Design and Architecture: Understanding single-page web applications. Simon and Schuster, 2015. 300 p. ISBN 1638353506.

8. Meredova A. Comparison of Server-Side Rendering Capabilities of React and Vue. 2023. P. 8–25. URL: https://urn.fi/URN:NBN:fi:amk-2023052915191 (Accessed: 02.05.2026).

9. Emmanni P. S. Comparative analysis of angular, react, and Vue. Js in single page application development. International Journal of Science and Research. 2023. Vol. 12, № 6. P. 2971–2974. URL: https://dx.doi.org/10.21275/SR24401230015 (Accessed: 02.05.2026).

10. Dubaj S., Pańczyk B. Comparative of React and Svelte programming frameworks for creating SPA web applications. Journal of Computer Sciences Institute. 2022. Vol. 25. C. 345–349. URL: https://doi.org/10.35784/jcsi.3020 (Accessed: 02.05.2026).

11. Rippon C. Learn React with TypeScript. Packt Publishing, 2023. 474 p. ISBN 9781803233849.

12. Пасічник С., Кунанець Н. Особливості формування front-end технологій у хмарних середовищах: клієнт–серверна архітектура та часове моделювання. Системи та технології. 2025. Вип. 18. С. 151–162. URL: https://doi.org/10.23939/sisn2025.18.2.151 (Accessed: 02.05.2026).

13. Luca G. De. FastAPI Cookbook: Develop high-performance APIs and web applications with Python. Packt Publishing Ltd, 2024. P. 20–47. ISBN 1805127853.

14. Lathkar M. High-Performance Web Apps with FastAPI. California: Apress Berkeley. 2023. P. 29–64. ISBN 9781484291726.

15. Lubanovic B. FastAPI. «O'Reilly Media, Inc.», 2023. P. 27–191. ISBN 1098135474.

16. Dombrovskaya H., Novikov B., Bailliekova A. PostgreSQL Query Optimization. Springer, 2021. P. 175–183. ISBN 1484268849.

17. Martins P., Tomé P., Wanzeller C., Sá F., Abbasi M. Comparing oracle and postgresql, performance and optimization. Springer, 2021. P. 481–490. URL: https://doi.org/10.1007/978-3-030-72657-7_46 (Accessed: 05.05.2026).

18. Do T.-T.-T., Mai-Hoang T.-B., Nguyen V.-Q., Huynh Q.-T. Query-based performance comparison of graph database and relational database. Proceedings of Conference. 2022. P. 375–381. URL: https://doi.org/10.1145/3568562.3568648 (Accessed: 05.05.2026).

19. Casola V., De Benedictis A., Rak M., Villano U. A novel security assessment methodology for web applications based on OWASP Top 10. Journal of Network and Computer Applications. 2022. Vol. 204. P. 103423. URL: https://doi.org/10.1016/j.jnca.2022.103423 (Accessed: 07.05.2026).

20. Idris M., Syarif I., Winarno I. Web application security education platform based on OWASP API security project. EMITTER international journal of engineering technology. 2022. P. 246–261. URL: https://doi.org/10.24003/emitter.v10i2.721 (Accessed: 07.05.2026).

21. Willberg M. Web application security testing with owasp top 10 framework. 2019. P. 7–32. URL: https://urn.fi/URN:NBN:fi:amk-2019121326449 (Accessed: 07.05.2026).

22. Varela-Vaca A. J., Gasca R. M. Automated Security Testing in Modern Web Frameworks and REST APIs. Computers & Security. 2023. Vol. 128. P. 103175. URL: https://doi.org/10.1016/j.cose.2023.103175 (Accessed: 08.05.2026).

23. Tverdokhlib O., Kovalenko O. Static and Dynamic Application Security Testing for Python-based Web APIs. Advances in Science, Technology and Engineering Systems Journal. 2025. Vol. 10, № 2. P. 112–125. URL: https://doi.org/10.25046/aj100211 (Accessed: 08.05.2026).

24. Rzepka C., Berger O. User Experience Evaluation of Single Page Applications: Frameworks and Methodologies. Applied Sciences. 2024. Vol. 14, № 3. P. 1205. URL: https://doi.org/10.3390/app14031205 (Accessed: 05.05.2026).

25. Li Y., Wang X., Liu Z. Automated Testing Techniques for Web Applications: A Comprehensive Review. IEEE Transactions on Software Engineering. 2021. Vol. 47, № 8. P. 1528–1545. URL: https://doi.org/10.1109/TSE.2021.3051428 (Accessed: 07.05.2026).