

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ІНТЕРАКТИВНОГО ОСВІТНЬОГО СЕРЕДОВИЩА
SOFTWARE ENGINEERING FOR INTERACTIVE
EDUCATIONAL ENVIRONMENT

Виконав(ла): студент(ка) 4 курсу, групи КНТ-132

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ГОРЛОВ М.В.

(ПРІЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.

(ПРІЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГОРЛОВА Максима Володимировича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Інженерія програмного забезпечення інтерактивного освітнього середовища. Software Engineering for Interactive Educational Environment

керівник проєкту (роботи) д-р філософ., доцент ЛЕОЩЕНКО Сергій Дмитрович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка задачі. 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація, тестування та дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1–4 Основна частина	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання «20» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3–4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Максим ГОРЛОВ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій ЛЕОЩЕНКО
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 119 с., 8 табл., 37 рис., 3 дод., 20 джерел.

ІНТЕРАКТИВНЕ ОСВІТНЄ СЕРЕДОВИЩЕ, ВЕБЗАСТОСУНОК, ПРОГРАМНА ІНЖЕНЕРІЯ, E-LEARNING, REACT, MONGODB, FIREBASE.

Об'єкт дослідження – процес розроблення програмного забезпечення для інтерактивних освітніх середовищ.

Предмет дослідження – методи та інструменти розробки веборієнтованого програмного забезпечення для створення інтерактивного освітнього середовища.

Мета роботи: підвищення рівня автоматизованості освітнього процесу за рахунок розробки інтерактивного освітнього середовища з використанням сучасних вебтехнологій.

Матеріали, методи та технічні засоби: використано методи об'єктно-орієнтованого та компонентно-орієнтованого програмування, сучасні підходи клієнт-серверної веброботи; бібліотеку React для побудови користувацького інтерфейсу за принципом Single Page Application, платформу Node.js із фреймворком Express.js для реалізації серверної частини та REST API, базу даних MongoDB для зберігання навчального контенту та даних користувачів, платформу Firebase для автентифікації та обміну повідомленнями у режимі реального часу. Розроблення проводилося на персональному комп'ютері під управлінням сучасної операційної системи.

Результати. Розроблено вебзастосунок інтерактивного освітнього середовища «RubixCourses», який забезпечує зручну взаємодію користувача з навчальним контентом у режимі реального часу. Реалізовано систему розмежування ролей викладача та студента з відповідними інтерфейсами, модуль управління курсами та навчальними завданнями, механізм

відстеження прогресу, систему миттєвого обміну повідомленнями між учасниками навчального процесу, а також захищену автентифікацію на базі Firebase із підтримкою JWT–токенів.

Висновки. Досягнуто мети роботи: створено програмну систему інтерактивного освітнього середовища з використанням сучасних вебтехнологій, яка підвищує ефективність навчання та забезпечує гнучкість у використанні. Застосунок підтримує адаптивний інтерфейс для пристроїв різних типів, модульну архітектуру для подальшого масштабування та миттєвий зворотний зв'язок як ключову характеристику інтерактивного середовища.

Галузь використання – освітні платформи закладів вищої освіти, системи дистанційного навчання, онлайн–курси та платформи для самостійного навчання користувачів.

Економічна ефективність. Використання застосунку скорочує час на адміністрування навчальних курсів та перевірку завдань завдяки автоматизації ключових процесів, зменшує організаційне навантаження на викладачів, а також підвищує залученість студентів до навчального процесу завдяки миттєвому зворотному зв'язку та зручному інтерфейсу.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 119 pages, 8 tables, 37 figures, 3 appendices, 20 sources.

INTERACTIVE EDUCATIONAL ENVIRONMENT, WEB APPLICATION, SOFTWARE ENGINEERING, E-LEARNING, REACT, MONGODB, FIREBASE.

Object of research – the process of developing software for interactive educational environments.

Subject of research – methods and tools for developing web-oriented software to create an interactive educational environment.

Objective of the study – to increase the automation level of the educational process by developing an interactive educational environment using modern web technologies.

Materials, methods, and technical tools: methods of object-oriented and component-oriented programming, as well as modern client-server web development approaches, were used; the React library for building the user interface based on the Single Page Application principle; the Node.js platform with the Express.js framework for implementing the backend and REST API; the MongoDB database for storing educational content and user data; and the Firebase platform for authentication and real-time messaging. The development was carried out on a personal computer running a modern operating system.

Results. A web application for the interactive educational environment "RubixCourses" was developed, providing convenient real-time user interaction with educational content. The system implements role-based access control for instructors and students with corresponding interfaces, a course and assignment management module, a progress tracking mechanism, an instant messaging system

between educational process participants, and secure Firebase–based authentication with JWT token support.

Conclusions. The objective of the study has been achieved: a software system for an interactive educational environment was created using modern web technologies, which enhances learning efficiency and ensures flexibility in use. The application supports a responsive interface for various device types, a modular architecture for further scaling, and instant feedback as a key feature of an interactive environment.

Field of application – educational platforms of higher education institutions, distance learning systems, online courses, and platforms for self–directed learning.

Economic efficiency. The use of the application reduces the time spent on administering training courses and grading assignments through the automation of key processes, decreases the organizational workload on instructors, and increases student engagement in the learning process due to instant feedback and a user–friendly interface.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	10
Вступ.....	11
1 Аналіз предметної області та постановка задачі.....	14
1.1 Поняття та особливості інтерактивних освітніх середовищ	14
1.1.1 Класифікація інтерактивних освітніх систем.....	17
1.1.2 Використання вебтехнологій у навчальних системах	20
1.1.3 Архітектура інтерактивного освітнього середовища	22
1.2 Аналіз існуючих рішень у сфері онлайн–освіти.....	25
1.2.1 Аналіз недоліків існуючих систем	31
1.3 Постановка задачі дослідження.....	33
1.4 Висновки до розділу 1	37
2 Розробка архітектури програми	38
2.1 Загальна архітектура системи	38
2.2 Проєктування функціональної структури системи.....	42
2.3 Проєктування моделі даних	43
2.4 Проєктування API	47
2.5 Висновки до розділу 2	49
3 Реалізація програми	50
3.1 Реалізація клієнтської частини системи	50
3.2 Реалізація серверної частини системи	53
3.3 Реалізація роботи з базою даних	57
3.4 Реалізація автентифікації користувачів.....	60
3.5 Висновки до розділу 3	63
4 Експлуатація, тестування та дослідження програми.....	65
4.1 Аналіз ефективності розробленої системи.....	65
4.2 Аналіз продуктивності системи.....	67
4.3 Практичне застосування системи	70

	9
4.3.1 Головна сторінка викладача	70
4.3.2 Сторінка завдань курсу	71
4.3.3 Інтерфейс студента	73
4.4 Приклад реалізації функціоналу	74
4.4.1 Сторінка авторизації	75
4.4.2 Сторінка здачі завдання	77
4.4.3 Система повідомлень	78
4.5 Тестування програмного забезпечення	80
4.6 Висновки до розділу 4	87
Висновки	89
Перелік джерел посилання	91
Додаток А Технічне завдання	93
Додаток Б Текст програми	100
Додаток В Слайди презентації	112

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
AUTH	– Authentication;
CRUD	– Create, Read, Update, Delete;
DB	– Database;
HTTP	– HyperText Transfer Protocol;
HTTPS	– HyperText Transfer Protocol Secure;
JSON	– JavaScript Object Notation;
JWT	– JSON Web Token;
LMS	– Learning Management System;
NoSQL	– Not Only SQL;
REST	– Representational State Transfer;
SPA	– Single Page Application;
UI	– User Interface;
UX	– User Experience;
СУБД	– система управління баз даних.

ВСТУП

Сучасний стан розвитку інформаційних технологій характеризується стрімким зростанням попиту на цифрові освітні рішення, здатні забезпечити ефективну взаємодію між учасниками навчального процесу. Однією з ключових сфер, де такі технології знаходять застосування, є організація дистанційного та змішаного навчання. Проблема створення середовища, яке не лише надає доступ до навчального контенту, а й забезпечує активну залученість користувача до процесу його засвоєння, залишається актуальною. Незважаючи на існування численних платформ для онлайн-навчання, більшість із них орієнтована на пасивне споживання матеріалу та не забезпечує повноцінної інтерактивності й миттєвого зворотного зв'язку.

Проблема якості дистанційної освіти є актуальною в усьому світі. Аналіз популярних освітніх платформ, зокрема Moodle, Coursera та Udemu, свідчить про те, що більшість із них побудована навколо логіки доставки контенту, а не навколо логіки навчання. Інтерфейси таких систем часто є перевантаженими, адаптивність до індивідуального темпу засвоєння – обмеженою, а зворотний зв'язок між викладачем і студентом – відкладеним у часі. В умовах українських реалій, де дистанційне навчання набуло особливої актуальності, а вимоги до зручності та надійності цифрових інструментів суттєво зросли, ці недоліки набувають критичного значення.

Проблема дослідження полягає у необхідності створення програмного забезпечення, яке забезпечує не лише доступ до навчальних матеріалів, а й повноцінну інтерактивну взаємодію між студентом і системою в режимі реального часу. Огляд існуючих рішень показує, що наявні платформи частково вирішують окремі аспекти цієї задачі, однак не реалізують їх у комплексі: гнучке розмежування ролей, миттєвий зворотний зв'язок, зручна комунікація та адаптивний інтерфейс рідко поєднуються в одному продукті. Провідні дослідження у галузі вебтехнологій вказують на перспективність використання реактивних фреймворків і хмарних сервісів для побудови таких

систем, однак бракує комплексних рішень, орієнтованих безпосередньо на потреби учасників навчального процесу.

Об'єктом дослідження є процес інженерії програмного забезпечення інтерактивних освітніх середовищ.

Предметом дослідження є веборієнтоване програмне забезпечення для створення інтерактивного освітнього середовища.

Мета роботи – підвищення рівня автоматизованості освітнього процесу за рахунок розробки інтерактивного освітнього середовища з використанням сучасних вебтехнологій.

Для досягнення поставленої мети в роботі вирішувалися такі завдання:

- проаналізувати сучасні підходи до проєктування інтерактивних освітніх середовищ та дослідити існуючі програмні рішення у сфері онлайн-навчання;
- визначити функціональні та нефункціональні вимоги до програмного забезпечення, обрати відповідний технологічний стек і спроектувати архітектуру системи;
- програмно реалізувати клієнтську та серверну частини застосунку, систему автентифікації з розмежуванням ролей, модуль управління курсами та завданнями, а також систему повідомлень у режимі реального часу;
- експериментально перевірити працездатність розробленої системи, оцінити її продуктивність та зручність користувацького інтерфейсу.

Методика дослідження включає аналіз літературних джерел та існуючих аналогів, компонентно-орієнтоване та об'єктно-орієнтоване програмування, проєктування клієнт-серверної архітектури з використанням REST API, а також функціональне тестування системи в умовах, наближених до реального використання.

Розроблений застосунок може бути використаний закладами вищої освіти для організації дистанційного навчання, викладачами для управління курсами та завданнями, а також студентами для самостійного опрацювання навчального матеріалу. Результати роботи мають практичну цінність, оскільки

запропоноване рішення усуває типові недоліки існуючих платформ і забезпечує повноцінний навчальний цикл в єдиному інтерактивному середовищі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття та особливості інтерактивних освітніх середовищ

Сучасне інтерактивне освітнє середовище представляє собою комплексну цифрову екосистему. Дана система забезпечує безперервну двосторонню комунікацію між користувачем та навчальним контентом. Традиційні електронні підручники у статичних системах управління навчанням реалізують лише односторонню передачу інформації. Інтерактивне середовище функціонує за принципом постійного діалогу із суб'єктом навчання. Ефективне освоєння навичок програмування неможливе без застосування засобів автоматичної компіляції коду. Повноцінна інтерактивність передбачає миттєву реакцію програмного комплексу на дії студента. Система аналізує код та виявляє логічні помилки на ранніх етапах. Платформа пропонує альтернативні варіанти вирішення задач або змінює складність модулів. Динамічна адаптація контенту забезпечує відповідність темпу навчання індивідуальним можливостям здобувача [1].

З технічної точки зору середовище є комплексом програмних засобів реального часу. Функціонування у реальному часі є визначальним фактором ефективності системи [2]. Платформа не повинна виступати пасивним сховищем статичних електронних документів. Інтерактивний комплекс виконує роль інтелектуального дзеркала дій користувача. Організація миттєвого зворотного зв'язку відокремлює прогресивні системи від застарілих сховищ контенту.

Впровадження таких платформ в Україні вирішує виклики дистанційного навчання в умовах технічних збоїв. Ринок освітніх технологій Швейцарії висуває високі вимоги до персоналізації та зручності інтерфейсів [3]. В обох випадках виникає проблема ізоляції студента перед монітором комп'ютера. Низька якість взаємодії з платформою призводить до швидкої втрати мотивації здобувачів. На відміну від статичних ресурсів інтерактивні системи стимулюють високу активність користувачів. Модернізація засобів

навчання забезпечує автоматичне виправлення помилок у розрахунках користувача.

Дослідження дозволяє виділити наступні основні ознаки інтерактивних освітніх комплексів:

Активність суб'єкта навчального процесу – студент трансформується із пасивного споживача інформації в оператора процесу навчання. Інженерне завдання полягає у стимулюванні аналітичного мислення користувача під час роботи [4].

Адаптивність навчального матеріалу – класичні освітні програми зазвичай позбавлені можливості гнучкого налаштування під користувача. Інтерактивне середовище визначає поточний рівень підготовки через динамічні метрики успішності. Користувачі із високим рівнем знань мають отримувати складніші завдання автоматично.

Ергономічність інтерфейсу взаємодії – проєктування аналогів відомих систем вимагає дотримання принципів програмного мінімалізму [5]. Мінімалізм полягає у глибокій інтеграції функцій при відсутності зайвих елементів.

Аналіз існуючих рішень свідчить про надлишковість більшості сучасних академічних платформ. Розробники намагаються одночасно поєднувати функції архівів журналів та форумів. Внаслідок цього втрачається головний фокус на безпосередніх потребах користувача. Інтерактивність визначається якістю когнітивного зв'язку а не кількістю елементів керування. Проєктування прототипу має базуватися на концепції повністю невидимого інтерфейсу [6]. Інтуїтивно зрозуміла технологія не повинна відволікати студента від процесу пізнання.

Інженерний аналіз освітнього середовища дозволяє виокремити декілька рівнів його безпосереднього функціонування:

Технологічний рівень – даний етап включає логіку обробки дій студента на серверній стороні [7]. Затримка відповіді системи понад двісті мілісекунд

руйнує ефект динамічного діалогу. Програмна розробка має орієнтуватися на асинхронну взаємодію для забезпечення плавності роботи [8].

Педагогічний рівень – система повинна створювати тимчасові опорні конструкції для підтримки здобувача освіти. Платформа надає автоматичні підказки при виникненні труднощів та поступово зменшує контроль. Гнучкий маршрут навчання вигідно відрізняє проєктований комплекс від жорстких існуючих систем.

Емоційний рівень – тривала концентрація уваги забезпечується впровадженням якісних мікроінтеракцій у програму. Легкі анімації успішного виконання завдань та візуалізація прогресу утримують увагу користувача. Лаконічність та відсутність візуального шуму є критичними для європейського ринку програмного забезпечення.

Головна особливість сучасних систем полягає у переході до моделі супроводу. Програмний комплекс не просто контролює знання а створює умови для неминучого засвоєння матеріалу. Популярні світові аналоги демонструють успіх завдяки простоті та високій інтеграції сервісів [1]. Створення нового проєкту передбачає поєднання легкості інтерфейсу із глибиною спеціалізованої платформи. Порівняльна характеристика систем підтверджує високу ефективність енерговитрат нових програмних архітектур [9].

Інженерія освітнього простору спрямована на цифровізацію методик провідних викладачів. Програмний комплекс повинен аналізувати тривалість вагань користувача перед вибором відповіді. Впровадження розробленого програмного продукту суттєво знижує рівень стресу студентів під час навчання. Автоматизація рутинних процесів перевірки знань є головною архітектурною метою даної роботи.

1.1.1 Класифікація інтерактивних освітніх систем

Класифікація сучасних освітніх систем здійснюється за кількома незалежними інженерними вимірами. Наявні класифікаційні моделі часто демонструють обмеженість через ігнорування реального досвіду взаємодії користувача.

За типом взаємодії в науковій літературі традиційно виділяють такі категорії:

- системи з тестуванням та оцінюванням знань здобувачів – орієнтовані на кількісний контроль успішності без глибокого аналізу помилок;
- комплексні симуляційні середовища – забезпечують моделювання реальних процесів для практичного освоєння навичок [10];
- спеціалізовані ігрові навчальні платформи – використовують інструменти гейміфікації для підвищення залученості користувачів;
- системи управління навчанням (LMS) – виконують функції структурування та збереження навчальних матеріалів [11].

Сучасні програмні комплекси переважно мають гібридну структуру. Відсутність інтерактивності перетворює систему управління навчанням на пасивне хмарне сховище файлів. Проєктоване середовище RubixCourses поєднує функції класичної LMS із інструментами динамічної взаємодії.

За способом доступу користувачів до контенту системи поділяють на такі види:

- веборієнтовані програмні комплекси – забезпечують доступ через браузер без необхідності попереднього встановлення компонентів [2];
- мобільні навчальні застосунки – адаптовані для роботи на портативних цифрових пристроях користувачів;
- десктопні програмні рішення – потребують інсталяції на локальні операційні системи персональних комп'ютерів.

Веборієнтований підхід гарантує максимальне охоплення аудиторії незалежно від типу апаратного забезпечення [8]. Використання вебтехнологій

є критичним в умовах нестабільного зв'язку або обмежених ресурсів. Сучасні вебрішення за продуктивністю майже не поступаються нативному програмному забезпеченню.

За рівнем адаптації контенту до потреб користувача системи класифікують на:

- статичні системи – демонструють ідентичний набір навчальних матеріалів для всіх категорій користувачів;
- адаптивні системи – змінюють параметри відображення контенту під індивідуальні запити здобувачів [1];
- інтелектуальні системи – здійснюють глибокий аналіз поведінки користувача засобами штучного інтелекту.

Більшість існуючих адаптивних платформ функціонують на основі жорстких детермінованих алгоритмів. Справжня адаптивність передбачає розпізнавання системою конкретних когнітивних труднощів користувача при написанні коду.

Класифікація за ступенем відкритості архітектури передбачає поділ на закриті пропрієтарні системи та модульні екосистеми [6]. Надмірна кількість модулів у поширених open-source рішеннях ускладнює процес їхнього адміністрування. Проєктоване середовище RubixCourses позиціонується як веборієнтований інтерактивний комплекс із елементами адаптивності. Дана архітектура дозволяє розробнику ефективно контролювати код та забезпечувати персоналізацію навчання [4].

Важливим критерієм для розробників є класифікація за моделлю синхронізації даних:

- системи з відкладеним зв'язком: використовують стандартні HTTP-запити із повним оновленням сторінок;
- середовища реального часу: забезпечують миттєвий обмін інформаційними пакетами між клієнтом та сервером [12].

Миттєве відображення результатів тестування докорінно змінює сприйняття програмного продукту користувачем. Впровадження реактивності

в RubixCourses реалізовано шляхом інтеграції бази даних Firebase Realtime Database [13].

За типом домінуючого контенту освітні платформи поділяють на такі групи:

- текстоцентричні навчальні платформи: базуються на опрацюванні текстових електронних документів;
- відеоцентричні великі системи: орієнтовані на перегляд медійного контенту користувачами [3];
- код-орієнтовані спеціалізовані середовища: фокусуються на практичному написанні програмного коду.

Проектована система належить до категорії гібридно-активних програмних комплексів. Користувачу надаються інструменти безпосереднього введення даних із миттєвою реакцією інтерфейсу. Помилка в коді розглядається як системний сигнал для корекції траєкторії навчання.

За ступенем когнітивного навантаження інтерфейсу виділяють важкі корпоративні та лаконічні мікросервісні рішення. Складні інтерфейси чинять значний психологічний тиск на кінцевого користувача платформи. Розробка RubixCourses базується на філософії ощадливої інженерії (Lean Engineering). Система створює низький поріг входження та не потребує додаткового навчання користувачів. Софт повинен ефективно виконувати цільові функції без вивчення складних інструкцій [5].

За психологією залученості користувачів системи доцільно класифікувати на:

- пасивні цифрові сховища: виконують роль статичних архівів для збереження навчальної документації;
- реактивні навчальні тренажери: стимулюють активність здобувача через мікроінтеракції та візуальні тригери [14].

Програмний комплекс RubixCourses спрямовує дії студента під час вивчення складних тематичних розділів. Інженерія інтерфейсу забезпечує постійну підтримку користувача засобами візуальної комунікації.

З точки зору архітектурної гнучкості системи поділяють на такі типи:

- жорстко–лінійні платформи: вимагають суворого послідовного проходження навчальних модулів користувачем;
- графові гнучкі системи: дозволяють будувати нелінійні траєкторії вивчення матеріалу [15].

Розробка RubixCourses характеризується високим нелінійним потенціалом завдяки використанню бази даних MongoDB [16]. Відсутність жорсткої схеми даних дозволяє оперативно додавати нові змістовні відгалуження [17]. Даний підхід суттєво розширює ступінь свободи для викладача та студентів.

За моделлю монетизації та відкритості виділяють закриті платформи з передплатою та open–source рішення. Платформа RubixCourses є демократичним інструментом на базі відкритих технологій React та Node.js [4]. Досвід впровадження EdTech проєктів підтверджує ефективність систем із можливістю швидкого автономного розгортання.

За можливістю аналітики та зворотного зв'язку системи бувають сліпими або прозорими. Більшість існуючих рішень фіксують лише фінальну оцінку здобувача освіти. Проєктоване середовище здійснює збір детального контексту взаємодії користувача із платформою. Система фіксує тривалість перегляду матеріалу та час аналізу конкретного рядка коду. Високий рівень прозорості перетворює програмний продукт на повноцінного партнера в освітньому процесі.

1.1.2 Використання вебтехнологій у навчальних системах

Сучасні електронні освітні платформи переважно функціонують на базі прогресивних вебтехнологій. Кросплатформеність забезпечує незалежність програмного продукту від операційної системи кінцевого користувача [2]. Вебзастосунки не потребують попереднього встановлення додаткових компонентів на локальні пристрої. Дані рішення демонструють високий

потенціал для горизонтального масштабування серверної інфраструктури. Оновлення програмного коду системи відбувається централізовано без залучення кінцевого користувача.

Критично важливим архітектурним рішенням є використання концепції Single Page Application [2]. Технологія SPA суттєво підвищує ефективність взаємодії користувача із інтерфейсом платформи [17]. Навігація між структурними розділами здійснюється без повного перезавантаження сторінки браузера [18]. Поточний стан інтерфейсу стабільно зберігається під час динамічних переходів. Вебзастосунок функціонує аналогічно до нативного програмного забезпечення для персональних комп'ютерів.

Для організації збереження інформації в проєкті обрано систему керування базами даних MongoDB [17]. Дана NoSQL база даних документного типу забезпечує високу гнучкість збереження контенту. Структура навчальних матеріалів може суттєво відрізнятися для різних онлайн курсів. Набір інформаційних полів тестових завдань безпосередньо залежить від типу конкретного питання [16]. СУБД MongoDB дозволяє успішно оперувати документами із довільною внутрішньою структурою. Розробник позбавлений необхідності постійної динамічної зміни загальної схеми бази даних.

Хмарна платформа Firebase використовується для організації безпечної автентифікації користувачів системи. Готове інженерне рішення дозволяє оптимізувати процес проєктування системи управління доступом [19]. Платформа забезпечує надійне збереження паролів сесій користувачів та маркерів безпеки. Застосування даного сервісу дозволяє сфокусувати увагу на реалізації бізнес логіки застосунку.

Основні компоненти проєктованої системи та їх технологічне забезпечення наведені в таблиці 1.1.

Таблиця 1.1 – Компоненти системи

Компонент	Опис	Технологія
Клієнтська частина	Взаємодія з користувачем	React
Серверна частина	Обробка запитів	Node.js / Express
База даних	Зберігання інформації	MongoDB
Система автентифікації	Управління доступом	Firebase

Аналіз наведених у таблиці 1.1 даних підтверджує чіткий функціональний розподіл ролей. Кожен окремий компонент архітектури відповідає за виконання визначеного кола завдань. Такий підхід забезпечує високу модульність архітектури та спрощує масштабування системи.

1.1.3 Архітектура інтерактивного освітнього середовища

Інтерактивне освітнє середовище у вигляді сучасного веб–застосунку базується на принципах багаторівневої архітектури [6]. Кожен окремий функціональний шар має чітко визначену область відповідальності всередині системи. Архітектурні рівні ізольовані один від одного для забезпечення модульності програмного комплексу [9].

- загальна структура системи складається із чотирьох основних взаємопов'язаних компонентів;
- клієнтська частина програмного забезпечення для реалізації інтерфейсу користувача;
- серверна частина для забезпечення бізнес логіки та обробки вхідних запитів;
- централізована система персистентного зберігання та структурування цифрових даних;
- спеціалізовані автономні служби автентифікації користувачів та забезпечення безпеки.

Взаємодія між зазначеними компонентами здійснюється на основі прозорих стандартизованих протоколів передачі даних. Кінцевий користувач платформи взаємодіє виключно із клієнтською частиною застосунку [2]. Клієнтський рівень розроблено із використанням можливостей бібліотеки React [4]. Даний компонент генерує асинхронні HTTP запити до серверного програмного інтерфейсу.

Серверне API побудовано на базі платформи Node.js та фреймворку Express [7]. Сервер виконує обов'язкову валідацію отриманих даних та реалізує основні бізнес процеси. Після обробки запиту здійснюється звернення до бази даних MongoDB для запису інформації [8]. Платформа Firebase виступає в ролі ортогональної автономної служби безпеки системи [16]. Процеси авторизації та управління сесіями користувачів протікають паралельно із основним потоком даних.

Описана багаторівнева архітектура забезпечує значні практичні переваги при експлуатації системи. Клієнтську та серверну частини програмного комплексу можна розгортати незалежно один від одного. Заміна сховища даних MongoDB на реляційну СУБД не вимагає перебудови клієнтського інтерфейсу [19]. Горизонтальне масштабування платформи реалізується шляхом запуску додаткових екземплярів сервера за балансувальником навантаження.

Для детальної візуалізації загальна структура розробленої системи наведена на рис. 1.1.

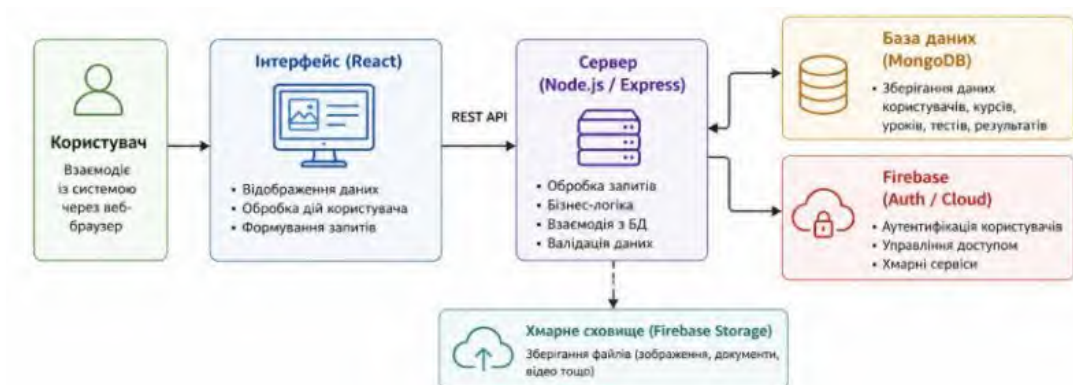


Рисунок 1.1 – Архітектура інтерактивного освітнього вебзастосунку

Графічний аналіз рис. 1.1 дозволяє детально відстежити інформаційні потоки всередині системи. Взаємодія користувача із середовищем відбувається через клієнтські компоненти бібліотеки React [14]. Сформовані запити спрямовуються до серверної частини для безпосередньої обробки та маршрутизації [18]. Сервер виконує необхідні маніпуляції та здійснює транзакції із базою даних MongoDB. Платформа Firebase забезпечує надійну автентифікацію користувачів та стабільну роботу хмарних сервісів. Застосування даного комплексу суттєво підвищує загальний рівень безпеки та масштабованості.

Ефективний розподіл обчислювального навантаження є запорукою високої інтерактивності вебзастосунку. Клієнтська частина системи оптимізована для швидкого представлення даних користувачу [8]. Інтерфейс забезпечує миттєву візуальну реакцію на будь які дії студентів. Основні складні аналітичні обчислення повністю перенесені на серверну сторону Node.js. Такий підхід дозволяє мінімізувати використання апаратних ресурсів пристрою кінцевого користувача.

Інтеграція платформи Firebase забезпечує цілодобове функціонування автономного контуру безпеки системи [13]. Застосування готового сервісу для управління сесіями автоматично вирішує завдання захисту інформації. Програмний комплекс використовує перевірені інструменти безпечного зберігання паролів та JWT токенів. Завдяки цьому розробка фокусується виключно на інженерії безпосередньо освітнього процесу.

Використання документо–орієнтованої бази даних MongoDB усуває обмеження класичних реляційних моделей [17]. Структура навчальних матеріалів може оперативно змінюватися без реконфігурації всієї схеми даних [12]. Архітектура дозволяє гнучко адаптувати формат контенту під поточні запити студентів. Зміна параметрів курсу відбувається динамічно без зупинки функціонування всього середовища.

Багаторівневий підхід значно спрощує процеси пошуку та усунення програмних помилок. Локалізація несправностей відбувається у чітко

визначених ізольованих контурах системи. Проблема оперативно виявляється у візуальному компоненті або в маршрутизаторі Express [7]. Ізоляція компонентів забезпечує високу живучість та стабільність інтерактивного середовища.

Обрана архітектурна модель створює надійний фундамент для майбутнього масштабування системи [15]. Відокремлений програмний інтерфейс API дозволяє легко підключити мобільний застосунок пізніше. Архітектура готова до майбутньої інтеграції модулів штучного інтелекту для персоналізації навчання. Простування адаптивної екосистеми забезпечує швидке підлаштування під мінливі потреби студентів.

1.2 Аналіз існуючих рішень у сфері онлайн–освіти

Проектуванню програмного комплексу передувало детальний аналіз архітектурних рішень провідних світових платформ Moodle, Coursera та Udey. Дослідження функціоналу здійснювалося з позиції розробника архітектури майбутнього інтерактивного середовища. Кожна з аналізованих систем має певні специфічні переваги в межах свого сегменту. Платформа Moodle характеризується високим рівнем гнучкості та модульності коду [11]. Система Coursera фокусується на академічній валідації навчальних матеріалів [1]. Платформа Udey пропонує спрощені механізми публікації авторського контенту [3].

Проте всі зазначені рішення орієнтовані переважно на статичне збереження контенту. Існуючі системи практично не забезпечують динамічної взаємодії в реальному часі. Результати комплексного порівняльного аналізу базових характеристик платформ наведено в таблиці 1.2.

Таблиця 1.2 – Порівняння існуючих освітніх платформ

Платформа	Інтерактивність	Персоналізація	Зручність	Гнучкість
Moodle	Середня	Висока	Середня	Висока
Coursera	Висока	Середня	Висока	Низька
Udemy	Середня	Низька	Висока	Низька
Розроблена система	Висока	Висока	Висока	Висока

Головним недоліком сучасних систем управління навчанням є надмірна бюрократизація інтерфейсів [14]. Велика кількість вкладених меню та елементів керування перевантажує увагу користувачів. Складний інтерфейс суттєво знижує когнітивний ресурс студентів під час навчання. Тривалий пошук необхідних функцій свідчить про низьку ергономічність програмного забезпечення. Сучасний вітчизняний користувач потребує максимально лаконічних та швидких цифрових сервісів. Перевантажений дизайн перетворює навчальний процес на подолання складного інженерного лабіринту.

Додатковою проблемою існуючих ринкових рішень є низький рівень реальної інтерактивності. Більшість платформ забезпечують лише пасивний перегляд медіаконтенту із періодичним контролем знань. Відсутність миттєвого зворотного зв'язку руйнує динаміку освітнього процесу. Сучасні тенденції висувають високі вимоги до інтерактивної взаємодії користувачів [11]. Традиційні системи характеризуються значним часовим відкладенням процесів синхронізації даних. Проєктоване середовище RubixCourses покликане забезпечити миттєву реакцію на дії студента.

З точки зору архітектури більшість популярних платформ побудовані як важкі моноліти [9]. Монолітна структура суттєво обмежує швидкість обробки інформаційних потоків всередині системи. Використання технологічного стеку MERN у проєктованій системі забезпечує високу реактивність інтерфейсу [14]. Застосування бібліотеки React дозволяє

здійснювати безшовну зміну контексту сторінки [1]. Сучасне програмне забезпечення 2026 року має повністю виключати тривале очікування рендерингу.

Сучасні розробники часто ігнорують принципи мікронавчання (micro-learning) при проектуванні систем. Навчальні траєкторії зазвичай подаються у вигляді нудних лінійних списків. Впровадження елементів динамічного візуального фідбеку дозволяє ефективно утримувати увагу користувача [10]. Плавне заповнення прогрес-бару після правильної відповіді стимулює пізнавальний інтерес студентів. Такий підхід працює набагато краще за статичні текстові повідомлення.

Ефективна програмна інженерія передбачає максимальну непомітність технологічних процесів для користувача. Складні схеми навігації та часті технічні помилки відволікають від навчання. Платформа RubixCourses реалізує принципово інший підхід до організації взаємодії суб'єктів. Застосування хмарних сервісів Firebase забезпечує швидку комунікацію у реальному часі [12]. Гнучка структура документів MongoDB дозволяє успішно оптимізувати архітектуру під конкретні завдання [16]. Очищення системи від зайвого функціоналу створює надійну основу для інтерактивності.

Прикладом організації інтерфейсу за принципом комерційної вітрини є платформа Udeemy [3]. Загальний вигляд початкової сторінки даного вебресурсу наведено на рис. 1.1.

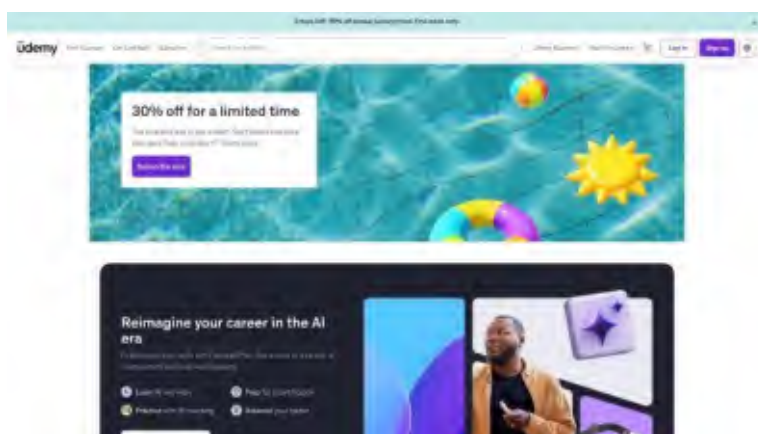


Рисунок 1.1 – Головна сторінка платформи Udeemy [3]

Аналіз структури сторінки на рис. 1.1 свідчить про мінімізацію початкового функціонального навантаження. Користувач позбавлений необхідності вивчення складних розкладів до моменту вибору контенту. Проте інтерфейс містить велику кількість маркетингових банерів та відволікаючих елементів. При розробці проєкту RubixCourses було запозичено загальну легкість навігації. При цьому інтерфейс повністю очищено від стороннього інформаційного шуму.

Принципово іншу модель побудови робочого простору демонструє академічна платформа Coursera [1]. Приклад організації особистого кабінету студента на даній платформі наведено на рис. 1.2.



Рисунок 1.2 – Інтерфейс особистого кабінету студента на Coursera [1]

Елементи інтерфейсу на рис. 1.2 демонструють високий рівень структуризації матеріалів. Система відстеження прогресу є досить ефективною але залишається занадто статичною. Процес комунікації із викладачем у межах даної платформи є ускладненим. Інтеграція сервісів Firebase у проєктованому застосунку дозволяє вирішити цю проблему [20]. Оновлення статусів та обмін повідомленнями відбуваються безпосередньо в момент дії.

Аналіз світового досвіду підтверджує необхідність створення гнучких адаптивних систем. Скріншоти існуючих платформ були використані як базові інженерні референси. Розробка RubixCourses поєднує модульність Coursera та

лаконічність інтерфейсу Udemu. Використання Single Page Application забезпечує миттєвий відгук системи на дії користувача [2].

В таблиці 1.3 наведено проведений аналіз порівняння платформ та їх особливостей.

Таблиця 1.3 – Порівняльний аналіз функціональних особливостей, недоліків та архітектурних рішень освітніх платформ

Платформа	Ключові концептуальні переваги	Характерні системні та інтерфейсні недоліки	Архітектурний стек та інтерфейсні рішення в оригіналі	Розроблювана система
1	2	3	4	5
Moodle	Високий рівень гнучкості, відкритий вихідний код, широкі можливості кастомізації під потреби закладу	Надмірна Бюрократизація та перевантаженість інтерфейсу, складні багаторівневі меню	Традиційна монолітна архітектура з частим перезавантаженням сторінок	Ліквідація візуального шуму, реалізація лаконічної Single Page Application (SPA) на базі React для безшовного переходу між модулями
Coursera	Висока якість дидактичного контенту, партнерство з провідними університетами, ефективна система прогрес-трекінгу	Статичність взаємодії, значна часова затримка зворотного зв'язку, ускладнена комунікація з викладачем (відсутність швидких чатів)	Орієнтована на контент монолітна структура зі статичними елементами трекінгу	Запозичення модульного принципу організації курсів із впровадженням Firebase для миттєвої синхронізації повідомлень та оновлення статусів у реальному часі

Продовження таблиці 1.3

1	2	3	4	5
Udemy	Демократичність авторування курсів, доступна цінова політика, спрощена початкова навігація (UX за принципом вітрини)	Фокус на комерціалізації контенту, переважність інтерфейсу сторонніми відгуками	Комерційно орієнтований вебінтерфейс з лінійними списками навчальних траєкторій.	Збереження легкості початкової навігації з повним очищенням робочого простору студента від рекламного шуму та впровадженням концепції micro-learning
Розроблена система	Комплексна інтерактивність, миттєвий зворотний зв'язок (instant feedback loop), гнучкість збереження даних	Відсутні капітальні недоліки конкурентів за рахунок відсікання надлишкового функціоналу	Реактивний MERN-стек (MongoDB Express.js, React, Node.js) у поєднанні з Firebase Realtime	Синтез модульності, лаконічності та динамічного навчання

Проведений аналіз дозволив визначити оптимальний баланс функціональних характеристик платформи. Дослідження охоплювало не лише архітектурні параметри але й логіку поведінки користувачів. Інтерактивне середовище отримало переваги NoSQL бази даних для збереження різнопланового контенту [19]. Потужний фронтенд на React повністю усуває статичність притаманну більшості існуючих LMS [4]. Синтез найкращих практик забезпечує створення якісного та затребуваного програмного продукту.

1.2.1 Аналіз недоліків існуючих систем

Проведений аналіз дозволив чітко систематизувати слабкі місця більшості сучасних електронних освітніх платформ [9].

Першим і найбільш серйозним недоліком є обмежена архітектурна гнучкість існуючих систем [6]. Платформа Moodle є гнучкою лише на рівні загальної конфігурації модулів [11]. При цьому вона залишається жорсткою на рівні проєктування користувацького досвіду (UX). Система Coursera характеризується повністю закритою архітектурою із суворим лінійним сценарієм навчання [1].

Другим суттєвим недоліком є надмірна складність інтерфейсу користувача. Сучасні системи для навчання парадоксально вимагають тривалого попереднього освоєння. Нові користувачі витрачають значний часовий ресурс на вивчення базової навігації.

Третім недоліком є недостатній рівень реальної інтерактивності програмних комплексів. Стандартне тестування після перегляду матеріалу виконує виключно функцію кількісного контролю. Справжня інтерактивність передбачає динамічну адаптацію навчального шляху під дії студента. Жодна з аналізованих платформ не забезпечує виконання цієї вимоги належним чином.

Четвертим недоліком виступає слабка персоналізація всього процесу електронного навчання [3]. Усі проаналізовані платформи пропонують уніфікований маршрут для різних категорій користувачів. При цьому повністю ігноруються індивідуальний темп та початковий досвід здобувачів.

П'ятим критичним недоліком є низька реактивність та технологічна неповороткість систем [2]. Більшість існуючих LMS функціонують за класичною застарілою моделлю повного перезавантаження сторінок. Очікування рендерингу інтерфейсу призводить до розсіювання уваги кінцевого користувача платформи. У проєкті RubixCourses дана проблема вирішена шляхом інтеграції протоколу WebSockets. Використання Realtime Database забезпечує миттєву реакцію інтерфейсу на дії студента [12].

Шостим важливим недоліком є незадовільна адаптація платформ під сучасні мобільні пристрої. Приклад некоректного відображення елементів інтерфейсу LMS на екрані смартфона наведено на рис. 1.3.



Рисунок 1.3 – Приклад некоректного відображення інтерфейсу LMS на мобільному пристрої [11]

Аналіз наведених на рис. 1.3 даних підтверджує порушення геометрії розташування блоків. Накладання елементів навігації суттєво ускладнює процес взаємодії користувача із платформою [5]. В сучасних умовах мобільність є критично важливою вимогою до освітнього софту. В архітектурі середовища RubixCourses було закладено прогресивні інженерні принципи Mobile–First. Це забезпечує ідентичну зручність використання інтерфейсу на будь яких типах екранів.

Також важливо відзначити проблему високої ізольованості даних в існуючих системах. Отримання детальної інформації про прогрес студентів для зовнішньої аналітики є ускладненим [19]. Це обмежує можливості створення інтелектуальних систем підтримки прийняття рішень. Використання СУБД MongoDB у даній роботі забезпечує необхідну гнучкість збереження даних [16]. Система фіксує неструктуровані дані про кожну дію

користувача на платформі [17]. Це дозволяє побудувати глибоку та точну модель аналізу поведінки студентів.

Сучасні розробники часто сприймають студента як пасивний об'єкт для накопичення знань. Натомість користувач має виступати активним учасником цифрової взаємодії всередині середовища. Сучасні технології дозволяють проєктувати інтерфейси із високим рівнем динамічного відгуку [14].

Проведений аналіз ринку підтверджує гострий дефіцит якісних інтерактивних середовищ. Виявлені функціональні прогалини стали науково технічним фундаментом для розробки проєкту RubixCourses. Застосування прогресивних інструментів React, Node.js та Firebase дозволило успішно вирішити проблеми користувачів [4]. Програмний комплекс повністю очищено від надлишкових елементів для підвищення ефективності навчання.

1.3 Постановка задачі дослідження

Детальний аналіз предметної області дозволяє виявити ключову проблему більшості сучасних освітніх платформ. Основний недолік полягає в надмірній орієнтації архітектури на процеси адміністрування [11]. При цьому інтереси безпосереднього процесу навчання часто ігноруються розробниками програмного забезпечення.

Складний інтерфейс змушує студентів витрачати час на вивчення заплутаної системи навігації [6]. Низька якість взаємодії із платформою (UX) суттєво знижує мотивацію здобувачів освіти. Процеси раптового переривання сесій або втрати результатів тестування руйнують динаміку навчання.

Сучасне інтерактивне середовище повинно функціонувати максимально непомітно для кінцевого користувача системи. Головна мета полягає в забезпеченні концентрації студента виключно на навчальному матеріалі [1]. Навігаційні елементи та кнопки інтерфейсу мають створювати умови для інтуїтивної взаємодії.

В межах даної дипломної роботи основний інженерний акцент зміщено на проєктування архітектури взаємодії [9]. Динамічна взаємодія користувача із системою виступає базовим принципом побудови програмного комплексу.

Поняття інтерактивності в даному дослідженні розглядається як комплексна реакція системи. Наявність простих статичних тестів після лекцій не забезпечує виконання сучасних вимог [3]. Справжня інтерактивність передбачає постійний аналіз дій користувача та адаптацію траєкторії навчання.

Програмний комплекс фіксує повторювані помилки студента в межах конкретної категорії питань. Система автоматично відстежує тривалість перерв у навчанні та інтенсивність повторного перегляду тем. Отримані метрики дозволяють гнучко оптимізувати контент під поточні потреби здобувача знань.

Надмірна функціональність сучасних LMS платформ часто призводить до перевантаження інтерфейсу користувача [11]. Велика кількість незатребуваних модулів ускладнює загальну архітектуру програмного забезпечення [6]. Існуючі академічні рішення тривалий час використовуються переважно як пасивні файлові архіви. Студенти сприймають такі системи як формальну необхідність без реальної користі для навчання.

Цифрова зручність платформ передбачає створення лаконічних мобільних інтерфейсів та чіткої структури курсів [5]. Важливим аспектом проєктування сучасного програмного забезпечення є забезпечення високого психологічного комфорту користувача.

Ключовим завданням даного дослідження є створення системи, яка одночасно відповідає наступним критеріям:

- забезпечення високого рівня технічної масштабованості серверної інфраструктури;
- мінімізація інформаційного та когнітивного навантаження на кінцевого користувача [6];
- підтримка безперервної інтерактивної взаємодії суб'єктів у реальному часі [2];

- реалізація простої та зрозумілої системи навігації по розділах;
- забезпечення швидкого асинхронного доступу до різнопланового навчального контенту;
- відсутність необхідності тривалого попереднього навчання роботі з платформою.

Рішення даних завдань дозволяє досягти оптимального балансу між інженерною складністю та зручністю.

Для реалізації поставленої інженерної мети в межах роботи вирішується комплекс взаємопов'язаних задач:

- комплексний аналіз предметної області та дослідження існуючих аналогічних програмних рішень;
- визначення повного переліку функціональних та нефункціональних вимог до створюваної системи;
- проектування гнучкої багаторівневої архітектури інтерактивного освітнього вебзастосунку;
- розробка оптимальної структури даних та схем взаємодії внутрішніх компонентів;
- програмна реалізація системи із використанням обраного сучасного технологічного вебстеку;
- проведення всебічного тестування та оцінка загальної ефективності розробленого рішення.

Етап аналізу дозволяє виявити типові системні помилки інтерфейсу існуючих платформ. Чітке формування вимог запобігає виникненню хаотичної структури програмного коду на етапі розробки [6]. Класична клієнт–серверна модель із REST API забезпечує незалежність та масштабованість модулів. Використання NoSQL СУБД MongoDB усуває жорсткі обмеження при збереженні різноструктурного контенту [19].

Клієнтська частина системи будується на базі компонентного підходу бібліотеки React [4]. Серверна логіка реалізується за допомогою програмних можливостей платформи Node.js та Express [7]. Безпечне управління

обліковими даними покладається на сервіси Firebase Authentication [20]. Єдиний технологічний стек на базі JavaScript суттєво спрощує подальшу підтримку коду [8].

Етап тестування дозволяє оцінити швидкодію стабільність API та коректність збереження прогресу. Процедура перевірки підтверджує правильність обраних архітектурних рішень у реальних сценаріях використання [10].

У результаті виконання поставлених інженерних задач забезпечується реалізація таких можливостей:

- ефективна асинхронна взаємодія користувача із поточним навчальним матеріалом;
- надійна підтримка процесів авторизації та динамічного управління ролями доступу;
- стабільне автоматичне збереження індивідуального прогресу навчання у базі даних;
- проходження контрольних тестів у режимі реального часу без перезавантаження сторінок;
- централізоване адміністрування структури та вмісту онлайн курсів через інтерфейс;
- адаптивний кросплатформний інтерфейс користувача для мобільних та стаціонарних пристроїв;
- високий потенціал для подальшого функціонального масштабування програмного комплексу.

Головна мета роботи полягає у створенні ефективного інструменту підтримки освітнього процесу. Проектування адаптивної екосистеми RubixCourses спрямоване на підвищення залученості студентів до навчання.

1.4 Висновки до розділу 1

Проведене дослідження дозволило повністю обґрунтувати доцільність розробки проєктованого програмного комплексу. На основі аналізу предметної області визначено ключові чинники ефективності сучасного електронного навчання.

Результати теоретичного та практичного аналізу дозволяють сформулювати такі наукові висновки:

- інтерактивні освітні системи суттєво підвищують загальну якість засвоєння навчального матеріалу;
- ефективність навчання досягається за рахунок постійної активної участі користувача в процесі;
- оперативний зворотний зв'язок у реальному часі є обов'язковою умовою сучасного інтерфейсу;
- динамічна адаптація контенту дозволяє гнучко враховувати індивідуальні особливості кожного студента.

Жодна з проаналізованих сучасних комерційних платформ не реалізує зазначені властивості в повному обсязі. Виявлені системні недоліки існуючих рішень формують обґрунтований простір для інженерної розробки. Обмежена персоналізація та перевантажені інтерфейси вимагають створення принципово нового підходу.

Постановка мети та декомпозиція задач дослідження повністю завершені на даному етапі. Отримані аналітичні дані слугують надійним технологічним фундаментом для подальшого проєктування системи.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Загальна архітектура системи

Проектний етап розробки програмного комплексу базується на детальному аналізі ергономічних недоліків існуючих платформ. Основна проблема сучасного освітнього софту полягає у високому рівні інтерфейсного опору [10]. Велика кількість зайвих дій та постійне перезавантаження сторінок суттєво знижують швидкість сприйняття інформації. Ефективна архітектура повинна забезпечувати інтуїтивно зрозумілу орієнтацію користувача у цифровому просторі. Здобувач освіти має концентрувати когнітивний ресурс виключно на опрацюванні навчального матеріалу [1].

З метою забезпечення стабільного функціонування системи обрано класичну клієнт–серверну архітектурну модель. Даний підхід гарантує чіткий розподіл функціональної відповідальності між основними компонентами [9]. Клієнт–серверна модель є перевіреним практичним рішенням із високим рівнем документальної підтримки. Модульна структура дозволяє масштабувати систему без виникнення критичних помилок у суміжних блоках коду.

Клієнтська частина програмного комплексу реалізована за концепцією Single Page Application (SPA) [2]. Повне перезавантаження вебсторінок при кожному переході суттєво уповільнює процес взаємодії. Постійні технічні переривання інтерфейсу руйнують стан концентрації уваги користувача на конкретній задачі. Застосування технології SPA дозволяє забезпечити миттєву навігацію без миготіння елементів дизайну. Вебзастосунок функціонує аналогічно до нативного десктопного програмного забезпечення.

Для побудови інтерфейсу користувача було обрано можливості сучасної бібліотеки React [4]. Головна перевага даного інструменту полягає в ефективній компонентній моделі проєктування [14]. Складні інтерактивні елементи уроку створюються у вигляді повністю ізольованих програмних контурів. Компонент тестування функціонує незалежно від загальних модулів

автентифікації користувачів. Модифікація логіки перевірки відповідей здійснюється в межах одного ізольованого файлу коду [10].

Серверна частина освітнього середовища базується на платформі Node.js із фреймворком Express [7]. Використання єдиної мови програмування на фронтенді та бекенді оптимізує процес розробки [8]. Однорідне технологічне середовище виключає часові витрати на перемикання між різними синтаксичними контекстами. Асинхронна модель виконання запитів Node.js чудово витримує високі пікові навантаження. Дана властивість є критичною під час одночасного проходження тестування великою групою студентів.

Комунікація між клієнтом та сервером організована через REST API із використанням формату JSON. Стандартизований підхід забезпечує високу передбачуваність та спрощує процес подальшого налагодження коду [6]. Протокол REST є інтуїтивно зрозумілим для будь якого профільного фахівця програмної інженерії.

Для персистентного зберігання цифрової інформації інтегровано систему керування базами даних MongoDB [17]. Електронний навчальний контент характеризується високим рівнем структурної неоднорідності матеріалів. Інформаційні блоки можуть містити текстові фрагменти медіафайли або набори тестових завдань [16]. Застосування реляційної схеми призвело б до суттєвого ускладнення запитів об'єднання даних [19]. СУБД MongoDB дозволяє успішно зберігати документи із гнучкими вкладеними структурами без жорсткої схеми.

Процеси автентифікації користувачів реалізовані за допомогою готового хмарного сервісу Firebase Authentication [20]. Створення власної системи управління доступами потребує значних часових ресурсів для забезпечення безпеки. Сервіс Firebase гарантує надійну обробку маркерів безпеки та правильне шифрування паролів [13]. Делегування даних функцій спеціалізованій платформі свідчить про високу інженерну зрілість проєкту.

Головною перевагою розробленого комплексу є висока узгодженість усіх обраних технологічних компонентів. Кожен рівень виконує власні завдання без втручання в архітектурні зони суміжних модулів [15]. Це дозволяє здійснювати локальну модернізацію окремих частин платформи без руйнування загальної структури.

Загальна архітектура розробленої освітньої системи включає наступні елементи:

- клієнтська частина для представлення даних користувачу;
- серверна частина для реалізації логіки обробки запитів;
- централізована система персистентного зберігання інформації;
- автономна система автентифікації та контролю доступу;
- протокол REST API для інформаційної взаємодії компонентів.

Детальна структурна схема розробленої платформи наведена на рис.

2.1.

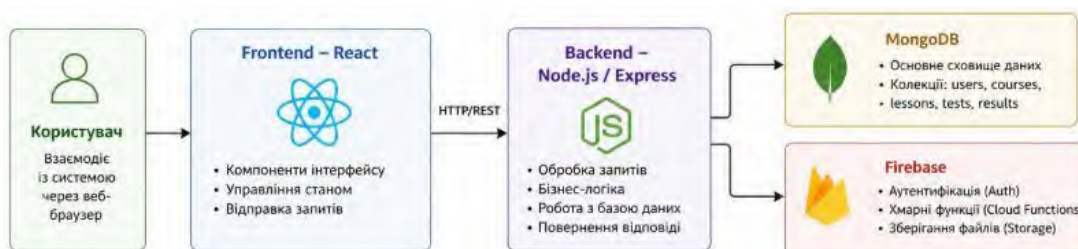


Рисунок 2.1 – Загальна архітектура системи

Аналіз наведеної на рис. 2.1 схеми підтверджує високу модульність розробленого програмного комплексу [6]. Наявна структура забезпечує незалежність окремих компонентів та спрощує процеси технічної підтримки [9]. Система може поступово розширюватися новими модулями без необхідності повного переписування коду. Обґрунтування вибору ключових технологій розробки системи наведено в таблиці 2.1.

Таблиця 2.1 – Вибір технологій розробки системи

Технологія	Призначення	Обґрунтування вибору
React	Побудова клієнтського інтерфейсу	Компонентна модель дозволяє ізолювати елементи інтерфейсу та повторно їх використовувати
Node.js	Серверне середовище виконання	Асинхронна модель ефективно обробляє одночасні запити
Express.js	Побудова REST API	Мінімалістичний фреймворк із гнучкою системою маршрутизації та middleware
MongoDB	Зберігання навчального контенту та даних користувачів	Документна модель дозволяє зберігати неоднорідні структури навчальних матеріалів без жорсткої схеми
Firebase Authentication	Автентифікація та управління сесіями	Забезпечує промисловий рівень безпеки зберігання облікових даних і обробки токенів
REST API (JSON)	Протокол взаємодії між клієнтом і сервером	Стандартизований і передбачуваний підхід; простий у налагодженні та зрозумілий для будь-якого розробника, який долучиться до проєкту

Дані таблиці 2.1 підтверджують відповідність кожного інструменту конкретним технічним вимогам проєкту. Сукупність обраних рішень формує узгоджений технологічний стек із високим рівнем взаємної інтеграції. Дана узгодженість є запорукою успішної тривалої експлуатації створеного програмного продукту.

2.2 Проектування функціональної структури системи

Функціональна структура розробленого програмного комплексу визначає сукупність можливостей платформи з позиції кінцевого користувача. Дана структура описує безпосередній склад інструментів взаємодії без деталізації технічної реалізації серверних алгоритмів.

Архітектура платформи включає чотири ключові функціональні блоки:

- управління користувачами – забезпечує процеси реєстрації автентифікації та розмежування прав доступу [20];
- навчальний контент – відповідає за структурування і відображення онлайн курсів та текстових матеріалів [1];
- система оцінювання знань – реалізує алгоритми динамічного тестування та фіксації результатів;
- відстеження прогресу – здійснює моніторинг поточної успішності та обсягу виконаних завдань.

Важливим інженерним рішенням є суворе розмежування ролей користувачів на рівні системи. Ролі студента та викладача передбачають використання принципово різного набору функціональних можливостей. Студент має доступ до проходження курсів та інструментів візуалізації індивідуального прогресу. Викладач оперує модулями створення контенту та аналізу агрегованої статистики навчальної групи [11]. Розмежування прав реалізовано на рівні серверної логіки для унеможливлення несанкціонованого доступу [6]. Клієнтська частина не може обійти встановлені обмеження шляхом модифікації параметрів запиту.

Основні функціональні модулі розробленої системи детально наведені в таблиці 2.2.

Таблиця 2.2 – Функціональні модулі системи

Модуль	Опис
Авторизація	Реєстрація та вхід користувачів
Курси	Створення та перегляд курсів
Контент	Відображення навчальних матеріалів
Тестування	Перевірка знань
Прогрес	Відстеження результатів

Дані таблиці 2.2 підтверджують повне охоплення базових процесів електронного навчання. Чіткий розподіл функцій забезпечує прозору внутрішню структуру розробленого програмного комплексу [9]. Створена архітектура суттєво спрощує подальше розширення функціональних можливостей платформи RubixCourses. Поділ на ізольовані модулі дозволяє модернізувати окремі сервіси без зміни загальної логіки.

2.3 Проєктування моделі даних

Етап проєктування логічної схеми бази даних є критично важливим для забезпечення стабільності програмного комплексу [6]. Своєчасна розробка моделі даних дозволяє суттєво мінімізувати часові витрати на подальший рефакторинг коду.

Загальна логічна структура розробленої системи базується на п'яти основних сутностях:

- користувачі – зберігає ідентифікаційні дані та параметри доступу суб'єктів навчання;
- онлайн курси – об'єднує метадані та структурні компоненти навчальних програм;
- модульні уроки – містить безпосередній текстовий та інтерактивний контент для опрацювання;

- тестові завдання – включає набори питань та варіанти відповідей для контролю знань [16];
- результати навчання – фіксує метрики успішності та динаміку проходження випробувань.

Кожна сутність характеризується специфічним набором атрибутів та логічними зв'язками із суміжними об'єктами [17]. Сутність Користувач містить атрибути логіна контактної адреси та поточної ролі в системі [20]. Курс включає обов'язкові метадані назви опису автора та масив посилань на уроки. Урок представляє навчальний матеріал у визначеному форматі із прив'язкою до тестових завдань. Результати зберігають детальну інформацію про успішність конкретного користувача за результатами проходження певного тесту.

Зв'язки між об'єктами реалізовані через унікальні ідентифікатори на основі механізмів посилань [16]. Система керування базами даних MongoDB не підтримує класичні операції об'єднання документів (JOIN). Проте денормалізація та посилання через унікальні ID повністю забезпечують збереження цілісності даних [19]. Ефективність даного підходу безпосередньо залежить від правильного інженерного проєктування вхідних запитів.

Специфікація атрибутів та взаємозв'язків між основними об'єктами платформи наведена на рис. 2.2.

Графічна модель на рис. 2.2 відображає архітектуру інформаційних потоків у межах освітнього процесу. Наявна структура забезпечує високу ефективність операцій доступу до даних та гарантує їхню цілісність.

Головною перевагою обраної NoSQL моделі є високий потенціал адаптації до майбутніх функціональних змін [19]. Традиційні реляційні бази даних вимагають складних процедур міграції при зміні структури таблиць. Гнучкість документів MongoDB дозволяє динамічно розширювати сутність Урок новими медіапосиланнями та віджетами [17]. Процес додавання нових полів відбувається без зупинки роботи всього програмного комплексу.

Динамічний характер освітнього контенту зумовлює доцільність використання саме документо–орієнтованих сховищ [12].

Результати навчання характеризується високим рівнем деталізації параметрів взаємодії користувача з платформою. Об'єкт фіксує точний час початку тестування кількість спроб та масив відповідей. Наявність цих метрик дозволяє аналізувати конкретні труднощі студента при засвоєнні матеріалу. Часткова денормалізація даних у цьому вузлі оптимізує швидкість виконання аналітичних звітів. Дублювання назви курсу всередині результату дозволяє уникнути додаткових запитів до бази даних.

При проєктуванні зв'язків на рис. 2.2 було свідомо обмежено глибину вкладеності документів [16]. Модульні уроки посилаються на батьківські курси через зовнішній атрибут courseId. Таке архітектурне рішення забезпечує високу масштабованість розробленої бази даних. Зберігання всіх уроків всередині одного документа курсу призвело б до перевищення лімітів розміру. Завантаження масивних документів могло викликати критичне перевантаження клієнтської частини системи. Платформа RubixCourses здійснює порційне підтягування контенту для забезпечення плавності роботи інтерфейсу.

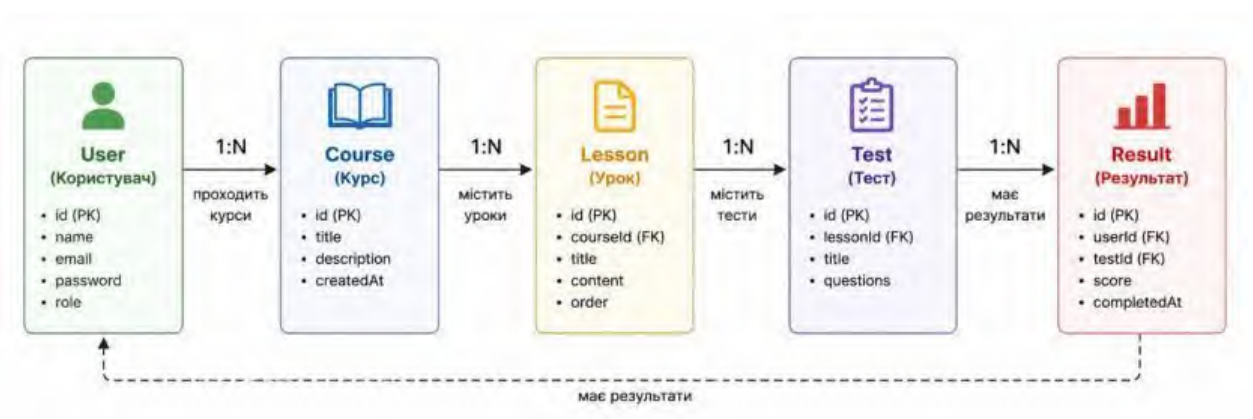


Рисунок 2.2 – Модель даних системи

Логічна схема розроблена із урахуванням перспектив майбутнього розширення функціоналу платформи. До структури документів заздалегідь

інтегровано атрибути тегів категорій та рівнів складності. Якісна програмна інженерія передбачає створення надійного інформаційного фундаменту для майбутніх інтелектуальних модулів [15].

Важливим інженерним завданням став пошук балансу між вкладеністю документів та використанням посилань [17]. СУБД MongoDB встановлює жорсткий ліміт на розмір одного документа в шістнадцять мегабайт. Постійне завантаження великих обсягів тексту при перегляді назви курсу є неефективним. Тестові завдання винесені в окрему незалежну колекцію бази даних. Це дозволяє реалізувати універсальний банк питань для повторного використання в різних курсах.

Результатом фіксації історії дій є масив об'єктів для побудови навчальної кривої студента. Система відображає не лише факт помилки але й безпосередній характер дій користувача. Даний підхід перетворює статичний журнал оцінок на повноцінне інтерактивне середовище [14].

Підтримка цілісності даних на рівні програмної логіки реалізована за допомогою інструментів Mongoose. Контроль зв'язків при видаленні батьківських об'єктів покладається на бекенд-шар Node.js [7]. Програмна логіка забезпечує автоматичне очищення залежних документів при видаленні курсу [9]. Це гарантує високу цілісність інформації без зниження загальної швидкодії системи. Оптимізація часу виконання запитів є критичною в умовах нестабільного мережевого з'єднання.

Поточна структура моделі RubixCourses передбачає можливість додавання нових ролей кураторів без реконфігурації коду. Розроблена гнучка схема даних здатна ефективно еволюціонувати разом із потребами користувачів.

2.4 Проєктування API

Програмний інтерфейс REST API виступає стандартизованим контрактом взаємодії між клієнтською та серверною частинами системи [7]. Чітке визначення параметрів цього контракту суттєво спрощує процес розробки клієнтського застосунку. Суперечлива архітектура API призводить до тривалого пошуку та усунення прихованих логічних помилок [6].

Для розробляємої освітньої платформи програмний інтерфейс спроектовано за класичними архітектурними принципами REST [9]. Усі методи протоколу HTTP використовуються за їхнім безпосереднім функціональним призначенням. Читання інформації здійснюється через GET запити а створення нових документів через POST. Для оновлення існуючих записів інтегровано методи PUT та PATCH. Видалення об'єктів із бази даних виконується за допомогою операцій DELETE. Усі мережеві ресурси іменуються виключно іменниками у формі множини. Обмін інформаційними пакетами між компонентами системи відбувається у стандартизованому форматі JSON [8].

Перелік основних маршрутів програмного інтерфейсу платформи наведено в таблиці 2.2.

Таблиця 2.2 – Основні API запити

Метод	Endpoint	Опис
POST	/register	Реєстрація
POST	/login	Вхід
GET	/courses	Список курсів
GET	/lessons	Матеріали
POST	/test	Проходження тесту

Дані таблиці 2.2 підтверджують повне охоплення базових функцій електронного освітнього процесу. Кожен окремий ендпоінт оптимізовано під

виконання однієї конкретної операції на сервері. Архітектура виключає використання універсальних обробників із великою кількістю вхідних параметрів.

Правильне проєктування API закладає надійну основу для подальшої технічної експлуатації платформи. Своєчасна стандартизація форматів відповідей спрощує інтеграцію інтерфейсу із фронтом на React [4]. В архітектурі RubixCourses чітко реалізовано інженерний принцип мінімального здивування [15]. Розробник клієнтської частини заздалегідь володіє інформацією про структуру об'єктів від сервера. Такий підхід робить взаємодію компонентів повністю інтуїтивною та передбачуваною.

Окрему увагу при проєктуванні інтерфейсу приділено уніфікації форматів серверних відповідей. Кожен вхідний запит повертає чітко структурований об'єкт із обов'язковими метаданими. При виникненні технічних збоїв сервер генерує інформативне повідомлення про помилку. Клієнтська частина оперативно розпізнає коди відповідей 401 Unauthorized або 404 Not Found. Отриманий від Firebase токен або статус видаленого уроку обробляються миттєво [20]. Високий рівень прозорості архітектури суттєво підвищує загальну швидкість розробки.

Важливою архітектурною вимогою до проєктованого API є дотримання принципу безстановості (Statelessness) [2]. Серверний шар не зберігає інформацію про поточну сесію конкретного користувача. Кожен окремий запит містить повний набір авторизаційних даних у заголовку Authorization. Перевірка прав здійснюється автономно при кожному зверненні до серверної логіки. Дана особливість робить освітнє середовище гнучким та готовим до горизонтального масштабування.

Наведені в таблиці 2.2 маршрути відображають загальну логіку взаємодії компонентів системи. За кожним POST запитом стоять складні алгоритми валідації та транзакцій MongoDB [16]. Для клієнтського інтерфейсу ці процеси приховані за лаконічним програмним контрактом. Суть інженерії

полягає в ізоляції складної реалізації за надійним інтерфейсом [6]. Визначення єдиної мови спілкування дозволяє перейти до реалізації серверного ядра.

2.5 Висновки до розділу 2

У другому розділі роботи повністю вирішено комплекс завдань пов'язаних із архітектурним проектуванням системи. Сформована трирівнева модель забезпечує надійну основу для побудови інтерактивного освітнього середовища.

За результатами проведеного проектування було отримано наступні науково технічні результати:

- обґрунтовано та сформовано гнучкий технологічний стек розробки на базі концепції MERN;
- визначено модульну структуру платформи із чітким розмежуванням функціональних ролей користувачів;
- розроблено логічну документо-орієнтовану модель даних під специфіку неоднорідного навчального контенту;
- спроектовано безстановочний програмний інтерфейс REST API для асинхронної взаємодії компонентів.

Усі прийняті архітектурні рішення орієнтовані на забезпечення високої реактивності та масштабованості системи. Отримані специфікації та схеми взаємодії стали безпосередньою інженерною основою для практичної реалізації. Технічні аспекти програмного втілення розроблених контрактів детально розглядаються у наступному розділі роботи.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Реалізація клієнтської частини системи

Після завершення етапу архітектурного проектування наступним кроком стала безпосередня програмна реалізація клієнтської частини системи. Інтерфейс користувача (frontend) формує первинне сприйняття програмного продукту кінцевим суб'єктом навчання. Навіть прогресивна серверна архітектура не забезпечить ефективності платформи при незадовільній якості клієнтського рівня.

Під час розробки інтерфейсу основний інженерний акцент робився на забезпеченні високої прогнозованості взаємодії. Цифровий освітній застосунок повинен функціонувати на основі чітких детермінованих алгоритмів. Користувач має інтуїтивно розуміти структуру розташування онлайн курсів та механізми переходу до уроків. Логіка повернення до попередніх розділів та перегляду результатів тестування має бути максимально прозорою.

Для практичної реалізації клієнтського рівня було використано можливості сучасної бібліотеки React. Головною причиною вибору даного інструменту став прогресивний компонентний підхід до побудови інтерфейсів. Кожен функціональний елемент системи реалізується у вигляді окремого незалежного програмного модуля:

- спеціалізована сторінка автентифікації та контролю доступу користувачів;
- модуль відображення повного списку доступних навчальних курсів;
- компонент представлення змістовного контенту конкретного уроку;
- інтерактивний блок динамічного тестування та контролю знань;
- універсальна навігаційна панель для орієнтації в системі;
- сторінка візуалізації індивідуального прогресу здобувача освіти.

Даний підхід суттєво спрощує процеси подальшого технічного обслуговування розробленого програмного комплексу. Модифікація

внутрішньої логіки тестування не чинить жодного впливу на суміжні модулі курсів. На практиці відокремленість контурів є критично важливою умовою для стабільного масштабування системи.

Окрему увагу в роботі приділено правильній організації внутрішньої структури та ієрархії компонентів. Зберігання всієї логіки обробки станів безпосередньо всередині сторінок є поширеною інженерною помилкою. Такий підхід з часом призводить до ускладнення коду через велику кількість вкладених умов.

Для запобігання цьому логіка була максимально винесена в окремі сервісні шари. Це дозволило забезпечити виконання таких важливих інженерних критеріїв:

- суттєве спрощення процесів повторного використання розроблених блоків коду;
- значне підвищення рівня читабельності та прозорості програмного коду;
- зниження ступеня зв'язності між окремими компонентами інтерфейсу;
- спрощення процедури подальшого модульного тестування клієнтської частини.

Управління внутрішньою маршрутизацією вебзастосунку покладено на спеціалізований пакет React Router. Завдяки цьому переходи між розділами відбуваються без повного перезавантаження сторінок браузера. На практиці це забезпечує функціонування платформи в режимі нативного десктопного додатка. Користувач повністю позбавлений тривалих часових затримок при оновленні текстового контенту.

Архітектура інтерфейсу користувача розроблена на основі сучасних концепцій програмного мінімалізму [5]. Перевантаження екрану великою кількістю елементів керування призводить до швидкого розсіювання уваги.

У проєктованій системі RubixCourses реалізовано стриманий та прагматичний підхід до дизайну:

- оптимальна мінімальна кількість інформаційних елементів на робочому екрані;
- наявність чіткої та зрозумілої системи навігації між модулями;
- повна адаптивність усіх компонентів під різні типи пристроїв;
- якісне візуальне розділення незалежних функціональних блоків інтерфейсу.

Окремим пріоритетом розробки стало забезпечення повної адаптивності елементів під мобільні платформи. Значна частина сучасних користувачів здійснює навчання за допомогою смартфонів або планшетів. Мобільний доступ до електронних освітніх сервісів сьогодні виступає обов'язковим інженерним стандартом. Програмний комплекс повинен стабільно та коректно масштабуватися на будь яких типах екранів.

Взаємодія із серверною частиною системи реалізована через асинхронні HTTP запити за допомогою бібліотеки Axios. Усі API виклики повністю винесені в окремий сервісний шар застосунку. Це дозволяє здійснювати централізоване управління логікою запитів обробкою помилок та токенами автентифікації. Централізація мережевого шару суттєво підвищує загальний рівень підтримуваності програмного забезпечення [6]. Хаотичне розміщення запитів всередині візуальних компонентів ускладнює процес розвитку платформи.

В результаті успішної реалізації клієнтської частини було створено високоінтерактивний вебінтерфейс. Розроблений frontend успішно забезпечує виконання таких ключових процесів:

- надійна безпечна автентифікація користувачів із перевіркою прав доступу;
- зручний перегляд загального списку та детальної структури курсів;
- послідовне або нелінійне проходження змістовних матеріалів уроків;
- інтерактивне тестування здобувачів із миттєвою фіксацією відповідей;

- моніторинг та візуалізація метрик індивідуального прогресу навчання;
- стабільна інформаційна взаємодія із серверною частиною через REST API.

Загальна структура та ієрархічні зв'язки компонентів клієнтської частини наведені на рис. 3.1.



Рисунок 3.1 – Структура клієнтської частини

Аналіз наведеної на рис. 3.1 схеми підтверджує чіткий поділ логіки вебзастосунку. Ієрархічна модель забезпечує незалежність відображення даних та повторне використання елементів інтерфейсу. Модульна архітектура клієнтського рівня гарантує стабільне функціонування платформи при високій інтенсивності запитів.

3.2 Реалізація серверної частини системи

Серверна частина системи виконує роль центрального логічного ядра розробляемого вебзастосунку. На даному рівні здійснюється обробка інформаційних потоків та взаємодія із базою даних. Серверний шар забезпечує перевірку прав доступу користувачів та реалізацію основної бізнес логіки [9].

Для реалізації бекенд частини використовується програмна платформа Node.js із фреймворком Express. Вибір Node.js обумовлений перевагами

асинхронної неблокуючої моделі введення та виведення даних [8]. Дана технологія ідеально підходить для вебзастосунків із великою кількістю одночасних мережових запитів. Серверний компонент системи повинен ефективно обробляти вхідні потоки даних без зниження продуктивності. Виникнення блокувань через тривалі обчислювальні операції негативно впливає на роботу всього застосунку.

Фреймворк Express використовується як мінімалістичний інструмент для побудови програмного інтерфейсу REST API. Основними причинами вибору даної технології є її простота гнучкість та надійність [5]. У проєктах середнього рівня надмірно складні інструменти часто призводять до зниження швидкодії.

Серверна архітектура платформи була побудована за суворим модульним принципом. Основні структурні елементи системи розділено на такі категорії:

- маршрути для прийому та первинного розподілу вхідних мережових запитів;
- контролери для забезпечення логіки обробки та координації потоків даних;
- сервіси для безпосередньої реалізації алгоритмів бізнес логіки системи;
- моделі даних для опису схем взаємодії із базою даних;
- компоненти Middleware для виконання наскрізних системних функцій платформи.

Маршрути відповідають за обробку вхідних HTTP запитів від клієнтської частини. Контролери містять основні алгоритми координації процесів усередині програмного комплексу. Модулі Middleware використовуються для перевірки авторизації користувачів токенів доступу та обробки винятків.

Окрему увагу в роботі приділено реалізації надійної системи обробки помилок. Повернення технічних повідомлень без зрозумілого опису суттєво

ускладнює процес дебагінгу. Для вирішення проблеми було впроваджено централізований інструмент обробки виняткових ситуацій.

Розроблена серверна частина стабільно підтримує виконання наступних операцій:

- динамічне створення та конфігурування нових онлайн курсів;
- отримання актуального списку уроків та змістовного матеріалу;
- безпечне збереження результатів проходження тестів у базі даних;
- автоматичне оновлення метрик індивідуального прогресу здобувачів освіти;
- перевірка поточних ролей та прав доступу користувачів системи.

Для обміну даними між компонентами застосовується стандартизований текстовий формат JSON. Це забезпечує просту і надійну інтеграцію між frontend та backend рівнями [8]. Важливою перевагою правильно побудованого бекенду є стабільність та передбачуваність результатів роботи.

Архітектурний поділ на контролери та сервіси запобігає перетворенню системи на некерований моноліт. Запити до бази даних не повинні розміщуватися безпосередньо в обробниках маршрутів. Контролер має лише прийняти запит перевірити вхідні дані та передати їх сервісному шару. Безпосередня реалізація бізнес логіки протікає виключно всередині ізольованих сервісів. Така ізоляція дозволяє тестувати алгоритми незалежно від специфіки транспортного протоколу.

Використання компонентів Middleware стало оптимальним рішенням для виконання наскрізних завдань системи. Даний підхід повністю виключає необхідність дублювання коду авторизації в кожному маршруті. Універсальний фільтр автоматично блокує неавторизованих користувачів на етапі прийому запиту. В RubixCourses реалізовано послідовний ланцюжок middleware для перевірки токенів JWT через Firebase. Після цього здійснюється обов'язковий контроль наявності відповідних прав доступу користувача. Наприклад роль викладача перевіряється перед виконанням операції видалення курсу.

Інтеграція із базою даних MongoDB реалізована за допомогою інструментів об'єктного моделювання Mongoose. Формат JSON є рідним середовищем для платформи Node.js та документів бази. Це дозволяє повністю уникнути складних та тривалих трансформацій структури даних [16]. Такий підхід є критично важливим для забезпечення високої продуктивності платформи. Обробка та збереження результатів тестування студента здійснюються сервером майже миттєво.

Важливим досягненням при реалізації бекенду є його висока горизонтальна масштабованість. Застосування безстановочної архітектури Stateless запобігає перевантаженню оперативної пам'яті сервера сесіями користувачів. Функції управління станом повністю делеговані автономним токенам безпеки Firebase. Це дозволяє запускати декілька копій Node.js застосунку за балансувальником навантаження. Стабільність системи більше не залежить від обчислювальної потужності одного конкретного сервера.

Асинхронний характер платформи забезпечує неблокуючу обробку важких операцій введення виведення. Завантаження викладачем великого масиву тестових завдань не зупиняє роботу сервера. Система реєструє подію і продовжує паралельно обслуговувати запити інших студентів.

Централізація бізнес логіки в сервісному шарі усуває дублювання коду оновлення прогресу. Спеціалізований сервіс викликається як при проходженні тестів так і при перегляді уроків. Дане архітектурне рішення значно спрощує процеси модульного тестування програмного комплексу. Модифікація алгоритму нарахування балів вимагає внесення змін лише в один файл.

Використання формату JSON зробило процеси інформаційного обміну всередині системи прозорими. Клієнтська частина ефективно здійснює візуалізацію вже готових та валідованих відповідей сервера. Розроблений бекенд функціонує передбачувано швидко та безпечно в межах встановлених вимог.

3.3 Реалізація роботи з базою даних

Для організації ефективного збереження інформації у розроблюваній системі використовується СУБД MongoDB. Основною причиною вибору даної технології стала висока гнучкість документно–орієнтованої моделі даних.

Електронний освітній контент переважно характеризується нестабільною внутрішньою структурою навчальних матеріалів. Онлайн курси можуть поєднувати текстові фрагменти контрольні тести медіафайли або інтерактивні елементи. Впровадження реляційної моделі в таких умовах призводить до надмірного ускладнення схем таблиць. Велика кількість міжтабличних зв'язків суттєво знижує загальну швидкодію системи при виконанні запитів.

СУБД MongoDB дозволяє зберігати інформаційні пакети у вигляді документів JSON–подібної структури [17]. Дана особливість значно спрощує процеси управління динамічним та різномірним контентом освітнього середовища.

Взаємодія між серверною логікою та базою даних реалізована за допомогою можливостей бібліотеки Mongoose. Дана спеціалізована об'єктно–документна модель забезпечує стабільне виконання таких функцій:

- ефективне моделювання логічної структури та параметрів обробки даних;
- суворі валідації типів полів на рівні серверного застосунку;
- організація декларативної роботи з гнучкими інженерними схемами документів;
- надійне виконання базових операцій створення читання оновлення та видалення (CRUD).

У межах розробленої архітектури бази даних було створено окремі колекції для наступних сутностей:

- колекція користувачів для збереження облікових записів та параметрів доступу;

- колекція курсів для структурування загальних метаданих навчальних програм;
- колекція уроків для розміщення безпосереднього змістовного наповнення модулів;
- колекція тестів для зберігання питань та варіантів відповідей контролю знань;
- колекція результатів для фіксації індивідуальних метрик успішності студентів.

В системі реалізовано автономний механізм перевірки коректності інформації перед її безпосереднім записом. Це дозволяє повністю унеможливити появу пошкоджених або неповних документів у сховищі. Правильне проєктування структури моделей на початкових етапах враховує потенціал майбутнього масштабування платформи.

Важливим інженерним рішенням стало проєктування гнучких схем із використанням Mongoose Virtuals та Middleware. Розміщення логіки обробки даних безпосередньо всередині контролерів призводить до ускладнення структури коду. В архітектурі RubixCourses логіка каскадного видалення інтегрована безпосередньо на рівень моделі даних. При видаленні курсу автоматично спрацьовує спеціалізований тригер класу pre-remove hook. Даний механізм виконує автоматичне очищення всіх пов'язаних уроків та тестів у MongoDB. Це гарантує високу цілісність інформації без зниження загальної гнучкості сховища.

Робота з динамічним контентом зумовила необхідність активного використання вкладених схем (Subdocuments). У колекції тестів кожне окреме питання представляє собою складний ізольований об'єкт. Даний об'єкт містить набір специфічних полів для опису типу питання варіантів відповідей та індексів. Застосування документів JSON-подібної структури дозволяє клієнтській частині отримувати готовий до рендерингу об'єкт. Фронтенд на React здійснює завантаження даних за один асинхронний запит до сервера.

Такий підхід дозволяє суттєво економити обчислювальні ресурси серверної інфраструктури.

Окрему увагу в роботі приділено оптимізації швидкодії бази даних за допомогою механізмів індексації. Стрімке зростання кількості результатів тестування може викликати значні затримки при виконанні звичайного пошуку. Послідовне сканування всієї колекції за атрибутом `userId` створює критичне навантаження на систему. Для вирішення проблеми впроваджено індекси для полів із високою частотою фільтрації запитів. Даний інструмент забезпечує пряме звернення до необхідної секції даних у сховищі. Оптимізація гарантує миттєве завантаження сторінки прогресу студента навіть при накопиченні тисяч записів.

Критично важливим кроком став перехід до асинхронної моделі обробки транзакційних операцій оновлення. У процесі експлуатації освітніх платформ часто виникають ризики одночасної модифікації документів. Для запобігання колізіям впроваджено логіку атомарних оновлень через оператори MongoDB `$set` та `$push`. Дані інструменти гарантують збереження кожної відповіді при паралельному виконанні запитів користувачів. Прогрес навчання студентів знаходиться під надійним захистом програмних механізмів СУБД.

Суворі валідації даних на рівні схем Mongoose виключає залежність від коректності вхідних пакетів фронтенду. В системі RubixCourses кожне поле схеми має чітко визначені інженерні обмеження. Алгоритми контролюють мінімальну довжину текстових назв та валідують формати посилань регулярними виразами. Блокування некоректних запитів на етапі входу запобігає збоєм при подальшому відображенні сторінок. Такий підхід дозволив повністю виключити появу критичних помилок сервера під час випробувань.

Описані на рис. 2.2 логічні зв'язки між сутностями практично реалізовані через динамічний метод `.populate()`. Метод дозволяє оптимізувати обсяг оперативної пам'яті за рахунок відкладеного завантаження об'єктів. Повна інформація про автора курсу підтягується лише при переході на головну

сторінку розділу. Рациональне використання ресурсів робить систему надзвичайно швидкою при роботі через мобільні мережі.

Обрана модель організації шару даних стала надійним фундаментом для забезпечення життєздатності платформи RubixCourses. Створена екосистема зберігання інформації демонструє високу стійкість та ефективно масштабується при зростанні аудиторії. Стабільна робота шару MongoDB дозволяє забезпечити подальше динамічне відображення даних в інтерфейсі React.

3.4 Реалізація автентифікації користувачів

Підсистема автентифікації є критично важливою частиною будь якого сучасного вебзастосунку. Особливо гостро це питання постає при роботі з персональними даними користувачів та результатами навчання.

Для практичної реалізації процесів автентифікації в проєкті використовується платформа Firebase Authentication. Застосування готового інструменту є значно безпечнішим за розробку власної системи збереження паролів у межах диплому. Хмарна технологія мінімізує ризики виникнення потенційних уразливостей у коді захисту інформації.

Інтегрована платформа Firebase забезпечує стабільне виконання таких важливих функцій безпеки:

- надійна реєстрація нових користувачів із валідацією облікових даних;
- безпечний вхід суб'єктів в електронне освітнє середовище;
- автоматична генерація та регулярна перевірка токенів доступу;
- захищене промислове зберігання конфіденційної інформації користувачів;
- повна підтримка стандартизованих маркерів безпеки класу JWT.

Після успішного проходження авторизації клієнтська частина отримує унікальний токен доступу. Даний маркер використовується для автентифікації всіх подальших запитів до програмного інтерфейсу API.

Для захисту приватних маршрутів системи від несанкціонованого доступу реалізовано спеціальне серверне middleware. Компонент здійснює обов'язкову верифікацію отриманих токенів перед виконанням будь якої операції.

У розробленій платформі RubixCourses реалізовано надійний механізм розмежування прав доступу користувачів. Система гнучко оперує встановленими ролями для забезпечення логічної структури середовища:

- студент повністю позбавлений можливості створення або модифікації онлайн курсів;
- викладач володіє повним набором прав для адміністрування навчального процесу.

На практиці суворе розмежування ролей є критично важливою умовою підтримки загальної безпеки системи.

Найбільша цінність використання технологій Firebase полягає в забезпеченні захисту передачі даних промислового рівня. Застосування перевіреного SDK від компанії Google усуває архітектурні ризики компрометації інформації. Процес взаємодії компонентів платформи організовано за класичною надійною схемою. Клієнтська частина отримує первинний ID-токен від сервісів авторизації Firebase. Даний маркер передається в обов'язковому заголовку Authorization кожного HTTP запиту. Серверний шар на Node.js здійснює прийом та подальшу верифікацію отриманого пакета.

Центральним елементом захисту на бекенді є спеціалізоване middleware перевірки справжньої автентичності запитів. Вхідний мережевий потік не спрямовується безпосередньо до бази даних. Система виконує обов'язковий аналіз терміну дії та цілісності отриманого токена. Верифікація здійснюється засобами адміністративного пакету програм firebase-admin на

серверній стороні. Тільки після успішного підтвердження статусу токена сервер дозволяє виконання основної бізнес логіки. Надійний захист конфіденційності оцінок та прогресу є першочерговим завданням програмної інженерії.

Розмежування ролей користувачів реалізовано у вигляді дворівневого контуру безпеки платформи. Збереження простого атрибута статусу в базі даних MongoDB доповнюється клієнтським захистом. На фронтенді інтегровано спеціалізований ізольований компонент під назвою ProtectedRoute. Компонент перевіряє роль поточного користувача безпосередньо до моменту рендерингу сторінки інтерфейсу. Ручне введення адреси адміністративної панелі в браузері викликає автоматичне перенаправлення на головну сторінку.

Основними елементом захисту залишається повторна перевірка прав доступу на рівні ендпоінтів сервера. Навіть при модифікації фронтенд-запиту бекенд заблокує несанкціоноване створення курсу через базу даних. Принцип подвійного контролю прав виступає золотим стандартом розробки безпечного програмного забезпечення.

Висока зрілість архітектури проявляється у реалізації алгоритмів обробки помилок авторизації. Система виключає відображення сухих технічних кодів технічних збоїв у консолі користувача. В RubixCourses інтегровано спеціальний перехоплювач винятків класу error interceptor. Механізм трансформує системні відповіді Firebase у зрозумілі інформаційні повідомлення для студентів. При завершенні терміну дії сесії користувачу пропонується зберегти поточну чернетку тесту. Після цього здійснюється автоматичний перехід до форми повторної безпечної авторизації.

Ефективним інженерним рішенням стало використання механізму Custom Claims платформи Firebase. Інструмент дозволяє зберігати параметри ролей безпосередньо всередині самого токена доступу [13]. Дане рішення повністю усуває необхідність виконання додаткових запитів до MongoDB при кожній дії. Бекенд на Node.js моментально визначає права користувача

безпосередньо з отриманого токена. Оптимізація суттєво знижує навантаження на базу даних під час пікових сесій тестування.

Практичне впровадження описаного підходу забезпечує високу безшовність та стабільність освітнього процесу. Автоматизація криптографічних функцій дозволяє викладачам фокусуватися виключно на створенні контенту. Студенти здійснюють виконання завдань при повній гарантії конфіденційності особистих результатів. Безпека функціонує у фоновому режимі та створює надійний фундамент для інтерактивної взаємодії.

Підсистема автентифікації виступає в ролі інтелектуального шлюзу контролю доступу. Модель успішно поєднує надійність хмарних технологій із гнучкістю бізнес логіки розробника. Наявність надійної системи ідентифікації дозволяє перейти до реалізації модулів комунікації та стрічки курсу.

3.5 Висновки до розділу 3

У третьому розділі роботи повністю завершено етап практичної програмної реалізації інтерактивного освітнього середовища. В межах інженерної розробки успішно створено та протестовано взаємопов'язані клієнтська та серверна частини застосунку.

Під час виконання програмної реалізації особливу увагу було приділено забезпеченню таких характеристик:

- високий рівень модульності та ізоляції окремих контурів коду системи;
- потенціал горизонтального масштабування серверної інфраструктури під пікові навантаження;
- промисловий рівень безпеки передачі та збереження персональних даних користувачів;
- висока ергономічність та інтуїтивність адаптивного користувацького інтерфейсу платформи;

– стабільність та передбачуваність функціонування розробленого програмного інтерфейсу API.

Комплексне тестування основних функціональних компонентів повністю підтвердило працездатність та стійкість розробленого програмного рішення.

Найбільш важливим результатом даного етапу стало формування цілісної цифрової екосистеми освітнього середовища RubixCourses. Клієнтська та серверна частини бази даних та механізми авторизації функціонують як єдиний реактивний комплекс. Створена архітектурна модель повністю готова до впровадження у реальний навчальний процес.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Аналіз ефективності розробленої системи

Оцінка загальної ефективності функціонування розробленого освітнього середовища є комплексним інженерним завданням. Даний показник не обмежується кількісними часовими метриками виконання окремих серверних операцій. Якість програмного забезпечення визначається через сукупність незалежних критеріїв взаємодії користувача із платформою.

Комплексна оцінка ефективності розробленого ПЗ базується на п'яти основних вимірах:

- швидкість функціонування клієнтського інтерфейсу при відображенні контенту;
- зручність та інтуїтивність використання системи без додаткових інструкцій;
- повна доступність та коректність масштабування контенту на різних пристроях;
- технічна стабільність і стійкість програмного комплексу під піковим навантаженням;
- потенціал горизонтального та функціонального масштабування архітектури бази даних.

Перший вимір ефективності забезпечується інтеграцією бібліотеки React та її механізму Virtual DOM. Технологія базується на побудові віртуальної копії деревоподібної структури сторінки в оперативній пам'яті. Система порівнює поточний стан із новим та оновлює виключно змінені вузли. На практиці при виборі варіанту відповіді браузер не перезавантажує всю вебсторінку. Зміна візуального статусу елемента відбувається локально без перевитрати системних ресурсів пристрою. Дана особливість є критично важливою при експлуатації платформи на малопотужних комп'ютерах.

Другий вимір визначає ергономічні властивості розробленого інтерфейсу користувача платформи. Практичне тестування системи за участю

фокус групи реальних студентів підтвердило високу інтуїтивність дизайну. Критерієм успішності виступає здатність користувача безпосередньо виконати базовий набір цільових дій. Суб'єкти навчання успішно здійснюють реєстрацію пошук курсів та проходження тестових завдань. Висока технічна якість інтерфейсу повністю компенсує відсутність попереднього інструктажу користувачів.

Третій вимір пов'язаний із кросплатформеністю та адаптивністю відображення навчальних матеріалів. Електронне середовище однаково стабільно функціонує на широкоформатних моніторах та мобільних пристроях. Адаптивна верстка виступає базовою інженерною вимогою до сучасного програмного забезпечення. Значна частина цільової аудиторії використовує смартфони як основний інструмент для навчання.

Четвертий напрямок оцінки характеризує стабільність роботи платформи при одночасному зверненні користувачів. Серверна частина розроблена із розрахунком на успішне проходження сесій групою з тридцяти студентів. Одночасна здача контрольних тестів є стандартним сценарієм для академічного процесу. Асинхронна архітектура Node.js запобігає виникненню критичних затримок при обробці запитів.

П'ятий вимір визначає масштабованість та адаптивність структурної схеми збереження даних. СУБД MongoDB забезпечує швидке розширення полів документів без проведення складних міграцій. Додавання нових аудіоматеріалів або модулів перевірки коду не вимагає реконфігурації таблиць. Схема бази даних динамічно адаптується під поточні потреби еволюції продукту.

Використання Firebase Authentication усуває необхідність розробки локальних контурів безпеки зберігання паролів. Функції управління токенами сесій та авторизації повністю делеговані перевірній хмарній платформі. Такий підхід свідчить про раціональний розподіл інженерних зусиль при проєктуванні системи. Конфіденційність інформації забезпечується надійними

інструментами промислового рівня безпеки. Створений комплекс повністю відповідає встановленим технічним критеріям якості та ефективності.

4.2 Аналіз продуктивності системи

Експериментальний вимір показників продуктивності розробленого програмного комплексу проводився з метою визначення потенційних критичних затримок. Практичний аналіз швидкодії дозволяє чітко виявити системні обмеження платформи в умовах реальної експлуатації.

Тестування швидкості обробки запитів здійснювалося в середовищі максимально наближеному до стандартних умов використання. Для експерименту використовувався веббраузер Chrome на апаратному пристрої середньої обчислювальної потужності. Підключення до мережі забезпечувалося через стандартне домашнє інтернет з'єднання.

Загальний вигляд головної сторінки розробленого вебзастосунку наведено на рис. 4.1.

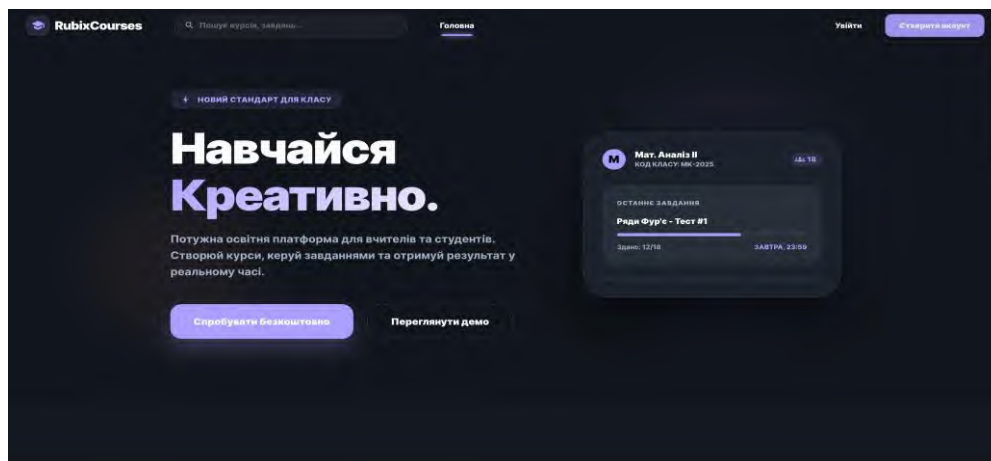


Рисунок 4.1 – Головна сторінка програми

Первинне завантаження головної сторінки інтерфейсу користувача триває в межах від однієї до двох секунд. Зазначений часовий інтервал забезпечує швидке формування позитивного сприйняття системи суб'єктом навчання. Подальша оптимізація швидкодії клієнтської частини можлива

шляхом впровадження механізмів розділення коду (code splitting). Дана технологія забезпечує динамічне завантаження виключно затребуваних у поточний момент модулів програми. Застосування цього підходу в React проєктах дозволяє скоротити тривалість стартової генерації сторінки вдвічі.

Зведені показники продуктивності платформи при виконанні базових операцій детально наведені в таблиці 4.1.

Таблиця 4.1 – Показники продуктивності системи при тестовому навантаженні

Операція	Час виконання
Початкове завантаження інтерфейсу	1–2 с
Отримання списку курсів	< 1 с
Обробка тестових відповідей	< 100 мс
Збереження результату в БД	< 1 с

Аналіз даних таблиці 4.1 підтверджує високу швидкість обробки результатів інтерактивного тестування студентів. Тривалість аналізу відповідей серверною частиною становить менше ста мілісекунд. Мінімальна затримка зворотного зв'язку є критично важливою для підтримання високої концентрації уваги.

Графічний інтерфейс користувача з боку здобувача освіти представлений на рис. 4.2.

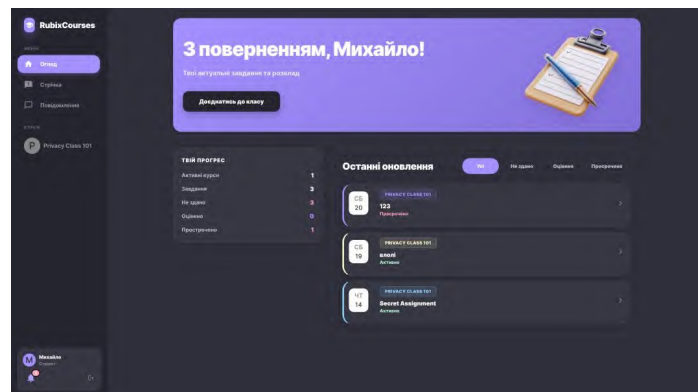


Рисунок 4.2 – Вигляд системи зі сторони студента

Елементи робочого простору на рис. 4.2 відображають структуру доступних курсів та модулів. Візуальний прогрес-бар поточного блоку забезпечує швидку орієнтацію користувача в обсязі вивченого матеріалу. Інтерфейс містить інструменти прямого переходу до останнього незавершеного етапу навчання. Лаконічна структура дизайну виключає наявність сторонніх елементів для запобігання відволіканню студентів.

Структурна організація інтерфейсу з боку керівника навчального процесу наведена на рис. 4.3.

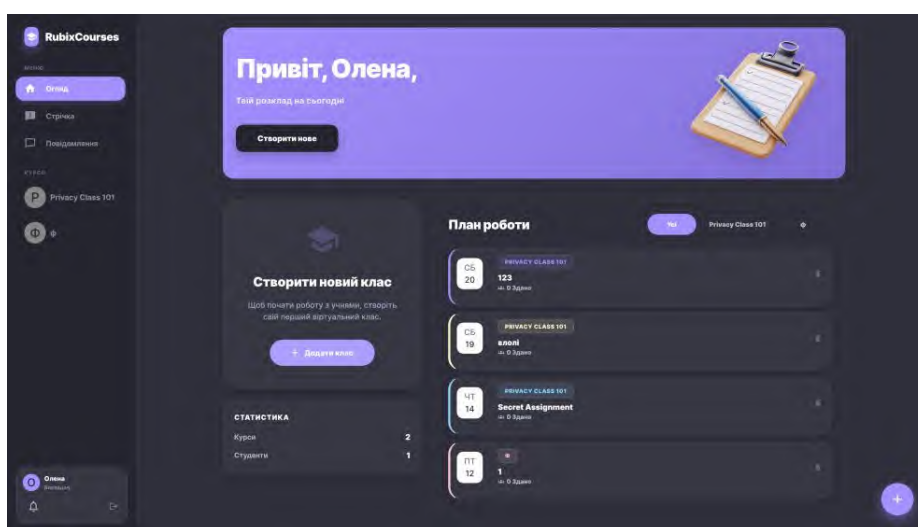


Рисунок 4.3 – Вигляд системи зі сторони вчителя

Панель управління на рис. 4.3 розроблена з урахуванням специфіки задач адміністрування. Модуль об'єднує списки зареєстрованих студентів та детальні статистичні метрики проходження завдань. Інтерфейс надає можливість оперативного редагування текстового контенту та параметрів тестування. Централізація аналітичних даних в одному робочому просторі підвищує ефективність моніторингу успішності групи. Викладач отримує повний обсяг інформації без необхідності звернення до сторонніх сервісів.

4.3 Практичне застосування системи

4.3.1 Головна сторінка викладача

Загальний вигляд початкового екрану інтерфейсу викладача після проходження процесу автентифікації наведено на рис. 4.4.

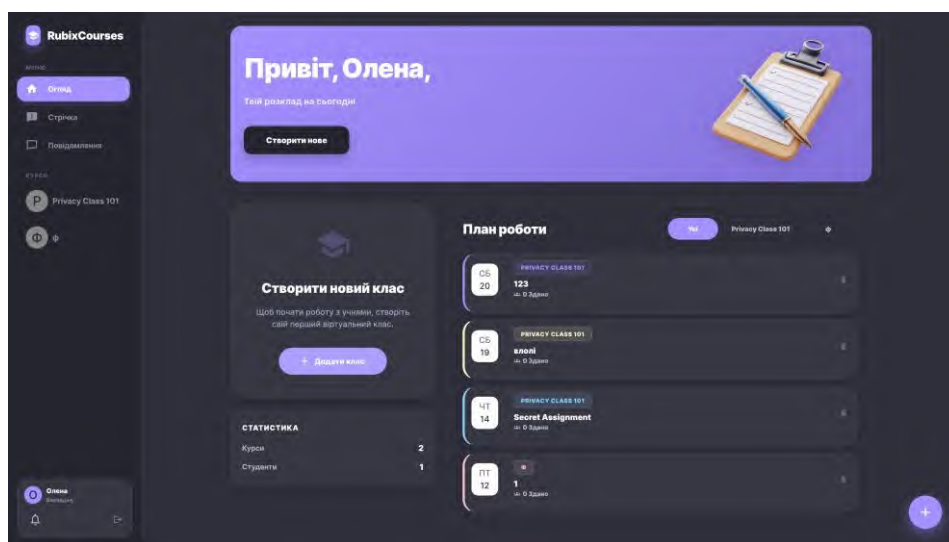


Рисунок 4.4 – Головна сторінка викладача (панель огляду)

Архітектурна побудова даного екрану забезпечує відображення повного контексту поточної навчальної ситуації без додаткових переходів. Графічний інтерфейс користувача розділений на три незалежні функціональні зони.

Ліворуч розташований фіксований сайдбар із елементами загальної навігації по розділах платформи. Дана зона об'єднує пункти огляду системи інформаційної стрічки та модуля повідомлень. Нижче міститься динамічний список курсів викладача та блок профілю із зазначенням поточної ролі. Стабільна структура сайдбару виключає необхідність повернення користувача на попередні сторінки застосунку. Навігаційна панель залишається доступною для взаємодії при будь яких переходах усередині системи.

У верхній частині центрального обчислювального блоку розміщено персоналізований банер привітання користувача. Банер містить інтегровану

керуючу кнопку швидкого переходу до створення нових матеріалів. Реалізація доступу до функцій проєктування в один клік підвищує ергономічність системи. Мінімізація кількості проміжних кроків інтерфейсу стимулює оперативність адміністрування навчального процесу викладачем.

Центральна зона екрану розподілена на два великі інформаційні блоки. Ліва частина містить функціональну картку створення нового класу із кнопкою додавання об'єктів. Під зазначеною карткою відображаються статистичні метрики кількості активних курсів та зареєстрованих студентів.

Правий блок представляє собою інтерактивний план поточної роботи викладача. Компонент відображає хронологічний список усіх запланованих навчальних завдань із фіксацією дат. Модуль підтримує можливість динамічної фільтрації контенту за критерієм конкретного курсу. Керівник курсу отримує повний обсяг планових даних без необхідності послідовного заходження в кожную дисципліну окремо. Centralized огляд суттєво оптимізує витрати часу викладача при моніторингу успішності груп. Багаторівнева вкладеність меню традиційних університетських платформ повністю усунена в даній архітектурі.

4.3.2 Сторінка завдань курсу

Після переходу до конкретного навчального курсу викладач потрапляє на екран керування дисципліною. Загальний вигляд даного інтерфейсу наведено на рис. 4.5.

Три функціональні вкладки у верхній частині екрану чітко розділяють поточний контекст адміністрування. Платформа диференціює робочий простір на модулі Стрічка Завдання та Студенти. Архітектура виключає змішування оголошень завдань та списків здобувачів в одному вікні. Кожна вкладка представляє собою ізольоване середовище для виконання окремого кола завдань.

На вкладці Завдання відображається структурований список поточних елементів контролю знань. Ліва частина кожного рядка містить виділений блок із фіксацією дня тижня та числа. У центрі рядка розміщуються назва завдання та суворі терміни його здачі. Права частина блоку оснащена кнопками видалення об'єкта та переходу до деталей.

Кольорове маркування лівого краю кожного рядка виконує важливу ергономічну функцію системи. Певні колірні ідентифікатори жорстко закріплені за конкретними навчальними курсами платформи. Це дозволяє викладачу миттєво орієнтуватися в інтерфейсі при великій кількості дисциплін.

Показники дати реалізовані у вигляді окремого структурованого графічного компонента системи. Формат відображення дня тижня та числа спрощує сприйняття інформації порівняно зі звичайним рядком. Дане інженерне рішення мінімізує час аналізу параметрів дедлайнів користувачем. Продумана деталізація інтерфейсу суттєво підвищує загальний рівень юзабіліті розробленого середовища. Модульна структура вкладки Завдання забезпечує високу ефективність оперативного контролю навчального процесу.

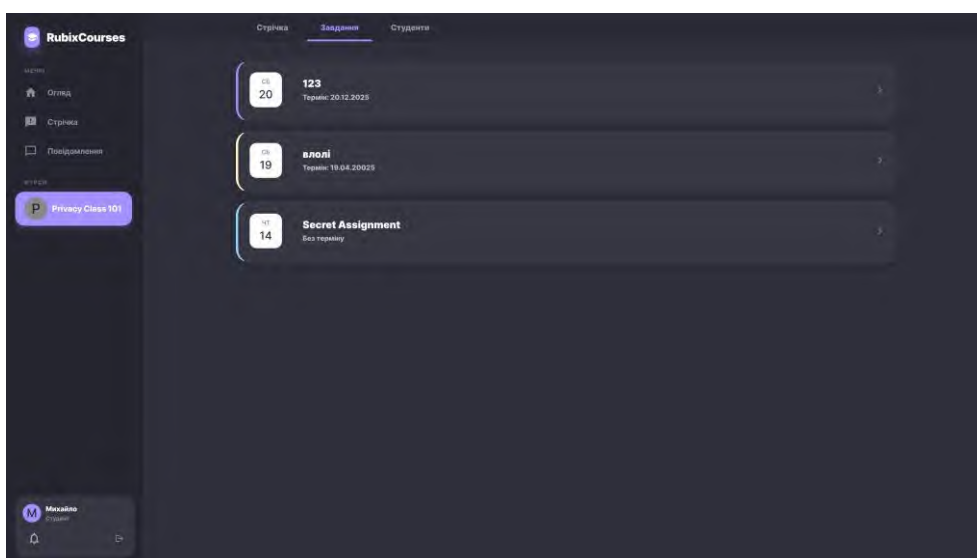


Рисунок 4.5 – Список завдань курсу зі сторони викладача

4.3.3 Інтерфейс студента

Загальний вигляд початкового екрану інтерфейсу користувача із роллю студента після автентифікації наведено на рис. 4.6.

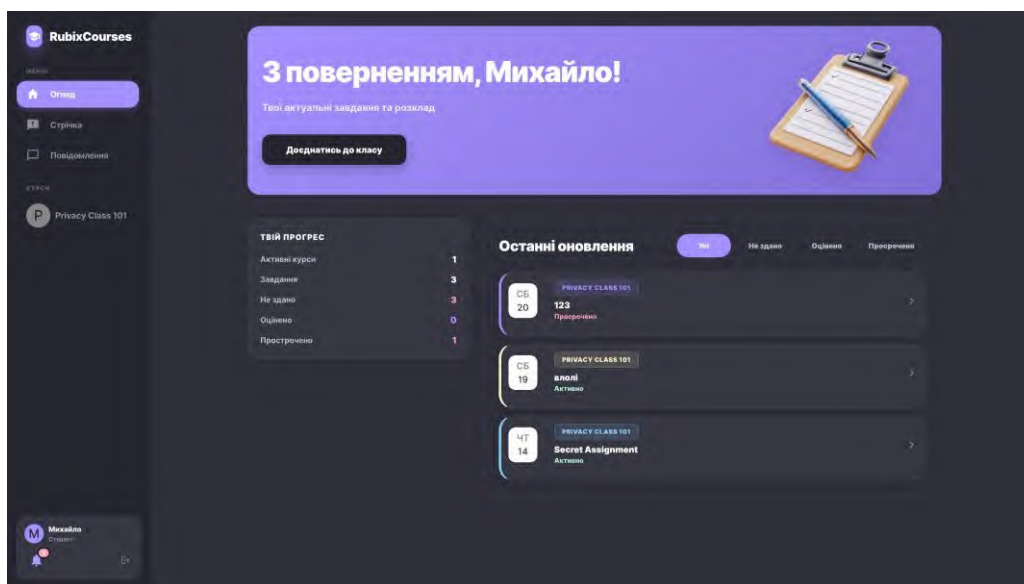


Рисунок 4.6 – Головна сторінка студента (панель огляду)

Графічний інтерфейс користувача на даному екрані адаптується під специфіку завдань здобувача освіти. Замість елементів проєктування курсів система відображає персоналізоване привітання та керуючу кнопку приєднання до класу. Дана модифікація є першою точкою розгалуження прав доступу на рівні початкового екрану.

Центральна робоча зона панелі огляду студента має специфічну двоблокову структуру розподілу контенту. Лівий інформаційний блок представляє собою інтерактивну картку під назвою Твій прогрес. Модуль відображає п'ять рядків із детальними статистичними метриками успішності в реальному часі. Система фіксує кількість активних курсів виконаних та зданих на перевірку завдань. Показники заборгованостей та прострочених дедлайнів автоматично виділяються червоним кольором для привернення уваги. Студент отримує можливість миттєвого аналізу поточного стану своєї навчальної діяльності.

Правий блок головного екрану містить модуль останніх оновлень із впровадженою системою фільтрації. Користувач може сортувати інформаційні повідомлення за критеріями виконання оцінювання чи прострочення термінів. Невиконані вчасно завдання маркуються відповідним колірним ідентифікатором у загальному списку. Поточні та активні завдання відображаються за допомогою нейтрального колірного оформлення рядків. Динамічна система статусів дозволяє оперативно визначати пріоритетність виконання поточних навчальних завдань.

Дана панель огляду виконує важливу роль у забезпеченні ефективної повсякденної взаємодії. Інтерфейс чітко та лаконічно візуалізує перелік необхідних дій на поточну дату. Швидке отримання цільової інформації свідчить про високу якість проектування архітектури всього застосунку.

Нижня частина навігаційного сайдбару містить обов'язкове текстове відображення імені та поточної ролі користувача. Наявність цього елемента безперервно формує контекст безпеки для всього графічного інтерфейсу. Користувач чітко ідентифікує свій статус у межах поточної сесії роботи з платформою.

4.4 Приклад реалізації функціоналу

Обґрунтування практичної спроможності прийнятих архітектурних рішень базується на аналізі трьох ключових функціональних екранів платформи. Кожен окремий інтерфейс ілюструє роботу ізольованого контуру взаємодії користувача із компонентами системи.

Першим важливим етапом взаємодії є процес автентифікації та входу користувача до системи. Даний контур демонструє інтеграцію клієнтської частини React із хмарними сервісами Firebase Authentication. Форма входу здійснює первинну перевірку правильності введення облікових даних на стороні клієнта. Після цього шифрований запит спрямовується до зовнішнього серверу авторизації для генерації унікального токена. Отриманий маркер

безпеки автоматично інтегрується в заголовки всіх подальших транзакцій до бекенду. Процес забезпечує надійний захист персональних даних від потенційного несанкціонованого перехоплення.

Другим критичним контуром є інтерфейс задачі навчальних завдань та обробки результатів тестування. Даний екран відображає механізми взаємодії між фронтендом сервером Node.js та базою даних MongoDB. Студент здійснює введення відповідей через інтерактивні елементи керування Single Page Application. Після підтвердження операції клієнтська частина формує асинхронний POST запит до відповідного ендпоінту. Серверний шар реалізує алгоритми валідації та виконує атомарне оновлення документа в MongoDB. Прогрес-бар інтерфейсу миттєво змінює стан без повного перезавантаження поточної сторінки браузера.

Третім функціональним елементом системи виступає підсистема комунікації між учасниками освітнього процесу. Екран обміну повідомленнями ілюструє роботу реактивних інструментів зв'язку в режимі реального часу. Функціонал базується на інтеграції можливостей Realtime Database для забезпечення миттєвого оновлення контенту. Повідомлення від викладача або студента відображаються в чаті без використання довгих опитувань сервера. Безшовна синхронізація даних мінімізує часові затримки при координації дій суб'єктів навчання. Комплексний аналіз зазначених контурів підтверджує високу стабільність та гнучкість розробленої архітектури.

4.4.1 Сторінка авторизації

Первинним елементом взаємодії користувача з платформою виступає спеціалізована сторінка автентифікації та входу. Даний інтерфейс розроблено на основі сучасних концепцій програмного мінімалізму. Форма авторизації містить два обов'язкові текстові поля для введення електронної адреси та пароля. Також інтерфейс оснащений керуючою кнопкою входу та

гіперпосиланням для реєстрації нових користувачів. Відсутність сторонніх графічних елементів оптимізує процес первинного доступу до функцій системи.

Процедура перевірки прав реалізована за допомогою інструментів хмарної платформи Firebase Authentication. Після введення облікових даних клієнтська частина передає ідентифікаційні пакети до зовнішнього сервісу. У відповідь система безпеки Firebase генерує та повертає унікальний маркер доступу JWT. Отриманий токен надійно зберігається в оперативній пам'яті Single Page Application застосунку. Маркер автоматично інтегрується у структуру кожного наступного HTTP запиту до серверного API. Поточна роль користувача викладача або студента персистентно зберігається в базі даних MongoDB. Зчитування статусу ролі відбувається безпосередньо після успішного підтвердження автентифікації користувача. Отриманий параметр визначає конфігурацію графічного інтерфейсу для поточної сесії роботи.

Важливим аспектом реалізації модуля є побудова якісної системи обробки виняткових ситуацій. При введенні некоректного пароля платформа виводить чітке інформаційне повідомлення для користувача. Програмна логіка виключає пряме відображення сирих технічних кодів помилок сервера. При відсутності електронної адреси в базі даних система генерує окреме попередження. Пряме виведення системних логінів авторизації у DOM простір свідчить про незадовільну якість ПЗ. Уніфікація відповідей забезпечує високу цілісність та надійність функціонування всього середовища.

Загальний вигляд розробленого інтерфейсу форми входу наведено на рис. 4.7.



Рисунок 4.7 – Сторінка авторизації

Аналіз структури екрану на рис. 4.7 підтверджує високу ергономічність обраного дизайну. Лаконічне розташування елементів керування мінімізує час проходження процедури ідентифікації суб'єктами навчання. Модульна архітектура форми входу гарантує стабільний захист конфіденційної інформації користувачів платформи.

4.4.2 Сторінка задачі завдання

Після переходу до конкретного навчального завдання студент потрапляє на спеціалізований екран виконання роботи. Загальний вигляд даного інтерфейсу наведено на рисунку 4.8.

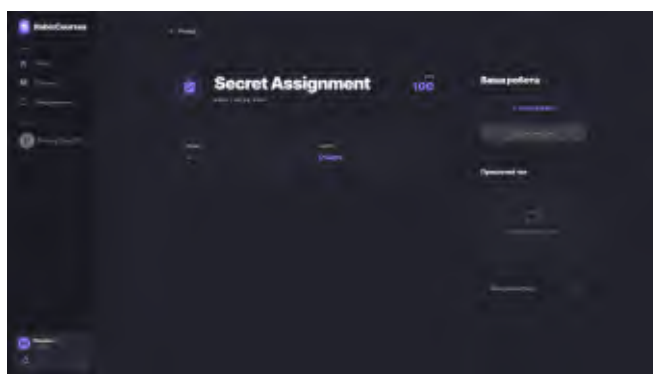


Рисунок 4.8 – Сторінка завдання зі сторони студента

Робочий простір екрану чітко розділений на дві незалежні функціональні зони взаємодії. Ліва частина містить повну інформаційну картку із детальними параметрами поточного завдання. Об'єкт відображає назву дату створення максимальний бал терміни здачі та поточний статус. Показник статусу очікування виконання автоматично виділяється фіолетовим колірним ідентифікатором. Розроблена колірна система статусів виконує важливу сигнальну функцію в інтерфейсі. Стан очікування маркується фіолетовим кольором здані роботи зеленим а заборгованості червоним. Користувач оперативно зчитує технічний стан завдання до моменту аналізу текстового вмісту.

При відсутності встановленого терміну здачі завдання поле інтерфейсу автоматично відображає нейтральний прочерк. Дане інженерне рішення виключає появу системних помилок класу null або undefined в DOM структурі. Програмна логіка коректно обробляє відсутність значень атрибутів документа бази даних.

Права функціональна зона під назвою Ваша робота оснащена елементами управління процесом здачі. Модуль містить керуючі кнопки додавання файлів та остаточного підтвердження виконання роботи. Після активації кнопки здачі система реєструє в MongoDB точну мітку часу та ідентифікатор студента. Також у документі результату фіксується пряме посилання на завантажений файл контенту. Модифікація статусу завдання відображається в панелі управління викладача в режимі реального часу.

Нижче зони прийому робіт інтегровано автономний ізольований компонент Приватний чат. Модуль забезпечує канал прямого зв'язку між викладачем та студентом у контексті конкретної задачі. Інформаційний діалог жорстко прив'язаний до ідентифікатора поточного навчального завдання. Такий підхід повністю унеможливорює змішування контексту обговорення із суміжними дисциплінами.

З інженерної точки зору розробка підтримує унікальні окремі треди для кожної пари студент та завдання. Складна архітектурна реалізація повністю компенсується високою ергономічністю використання платформи. При переході до перевірки роботи викладач аналізує повідомлення виключно щодо поточного завдання. Необхідність тривалого пошуку цільової інформації у загальних чатах курсу повністю усунена.

4.4.3 Система повідомлень

Взаємодія користувачів у межах комунікаційного контуру платформи здійснюється через спеціалізований інтерфейс повідомлень. Доступ до даного модуля забезпечується через відповідний пункт навігаційного сайдбару

системи. Графічне рішення реалізовано за класичною двоколонковою архітектурною схемою розподілу робочого простору. Ліва частина екрану містить список поточних діалогів а права відображає вікно активної розмови.

Загальний вигляд розробленого інтерфейсу підсистеми повідомлень наведено на рис. 4.9.

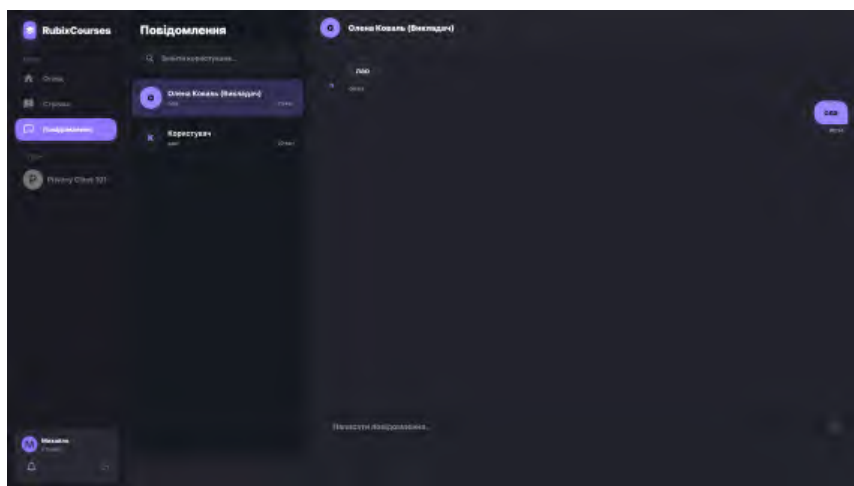


Рисунок 4.9 – Сторінка повідомлень

Ліва колонка модуля оснащена інтегрованою формою текстового пошуку контактів у верхній частині. Кожен окремий рядок списку діалогів відображає ім'я співрозмовника фрагмент останнього повідомлення та точну дату. Дана структура відповідає стандартизованим патернам проєктування сучасних інтерфейсів зв'язку.

Права функціональна зона призначена для безпосереднього ведення діалогу з викладачем або студентом. Повідомлення розташовані за дзеркальною схемою залежно від статусу відправника в системі. Вхідні інформаційні пакети вирівнюються за лівим краєм а власні за правим. Під кожним текстовим блоком відображаються автоматичні мітки часу транзакції. Нижня частина активної зони оснащена полем введення тексту та керуючою кнопкою відправки.

Технічна реалізація комунікаційного шару потребувала впровадження реактивних інструментів передачі даних. Відображення повідомлень у

реальному часі без перезавантаження сторінок забезпечується архітектурою Firebase Realtime Database. Клієнтська частина здійснює постійну підписку на оновлення конкретного інформаційного вузла бази даних. При появі нового документа у сховищі React компонент автоматично модернізує свій стан. Система повністю позбавлена необхідності періодичного циклічного опитування сервера клієнтською стороною. База даних самостійно ініціює передачу актуальних пакетів інформації на фронтенд.

Даний підхід принципово відрізняється від класичної синхронізації через протоколи REST API. Застосування технології коротких опитувань (polling) викликає значні часові затримки при обміні даними. Тимчасове очікування відповіді бекенду руйнує динаміку безпосереднього живого діалогу користувачів. Використання підписки у Firebase Realtime Database повністю усуває проблему розсинхронізації контенту. Підсистема забезпечує високу швидкість передачі повідомлень без виникнення інтерфейсних затримок.

Впровадження прямого зв'язку всередині платформи виконує важливу функцію в структурі освітнього процесу. Комунікаційний модуль повністю усуває необхідність використання сторонніх месенджерів або електронної пошти. Платформа RubixCourses успішно об'єднує процеси вивчення матеріалу здачі завдань та консультування. Архітектурна цілісність системи забезпечує високу ефективність та зручність її практичної експлуатації.

4.5 Тестування програмного забезпечення

Етап функціонального тестування програмного комплексу є обов'язковою умовою забезпечення його високої життєздатності. Своєчасна перевірка працездатності модулів дозволяє унеможливити виникнення критичних збоїв під час експлуатації платформи користувачами. При розробці освітнього середовища RubixCourses тестові сценарії проєктувалися паралельно із процесом безпосередньої реалізації функцій. Паралельне

тестування забезпечує виявлення архітектурних дефектів на ранніх етапах побудови програмних компонентів.

Проведена перевірка охоплює повний комплекс базових функціональних сценаріїв використання розробленого вебзастосунку. Тестування включало верифікацію модулів реєстрації користувачів автентифікації керування курсами проходження завдань та збереження прогресу. Критично важливим етапом стала перевірка коректності інтеграції клієнтської частини із хмарними сервісами Firebase. Контроль безпеки передбачав обов'язковий аналіз валідності маркерів доступу на серверній стороні системи. Middleware захисту приватних маршрутів забезпечує надійне і суворе розмежування встановлених привілеїв користувачів. Зведені результати проведених випробувань функціональних блоків платформи детально наведені в таблиці 4.1.

Таблиця 4.1 – Результати тестування системи

Функція	Очікуваний результат	Фактичний результат
Реєстрація	Створення акаунта	Успішно
Авторизація	Вхід у систему	Успішно
Курси	Відображення	Успішно
Тести	Проходження	Успішно
Прогрес	Збереження	Успішно

Аналіз даних таблиці 4.1 підтверджує повну працездатність усіх перевірених функціональних сценаріїв системи. Процеси реєстрації та входу продемонстрували високу якість інтеграції із зовнішніми сервісами Firebase. Модулі відображення контенту та тестування підтвердили коректність логіки розробленого REST API. Тестування асинхронних запитів до платформи Node.js здійснювалося за допомогою спеціалізованих інструментів моніторингу мережі. Перевірка підтвердила стабільну фіксацію результатів у

відповідній колекції бази даних MongoDB. Раціональна побудова мережевого шару гарантує миттєву візуалізацію отриманих балів в інтерфейсі студента.

Окрему увагу в роботі було приділено випробуванню підсистеми обміну повідомленнями в реальному часі. Для перевірки стабільності було імітовано процес одночасної комунікації викладача із кількома студентами. Завдяки використанню технології підписки у Firebase Realtime Database досягнуто повної відсутності часових затримок. Моніторинг швидкості обробки транзакцій здійснювався за допомогою логування подій на серверній стороні. Результати підтвердили високу реактивність системи при інтенсивному інформаційному обміні.

Функціональні вікна перевірених модулів системи зафіксовані на відповідних графічних схемах:

- процес первинного створення облікового запису користувача наведено на рис. 4.10;
- форма авторизації та входу до особистого кабінету відображена на рис. 4.11;
- інтерфейс взаємодії викладача із структурою дисциплін наведено на рис. 4.12;

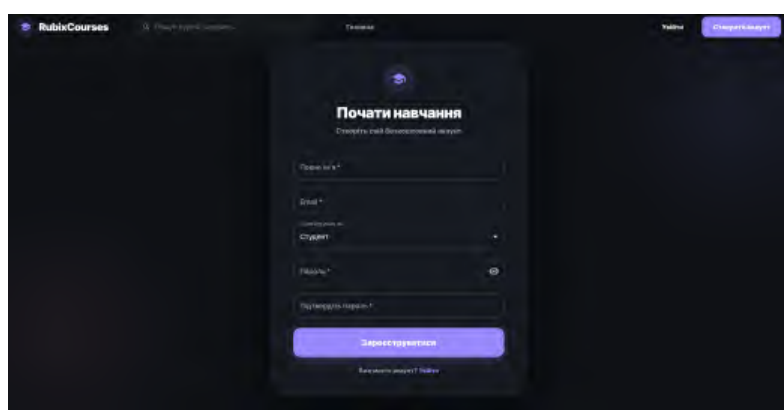


Рисунок 4.10 – Сторінка реєстрації

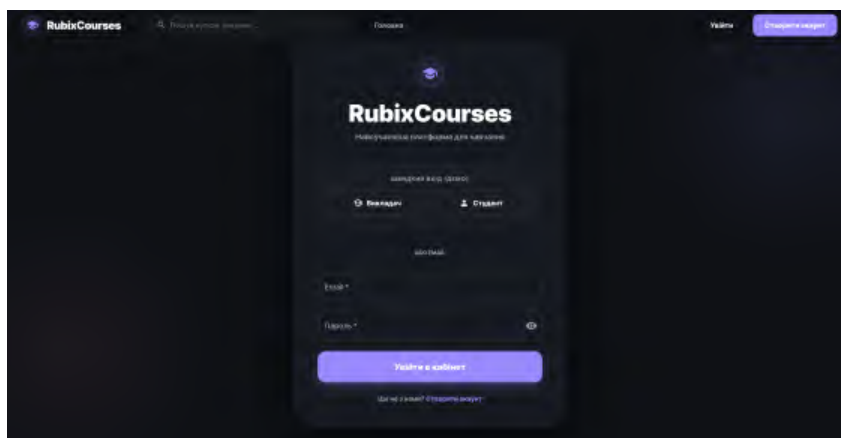


Рисунок 4.11 – Сторінка авторизації

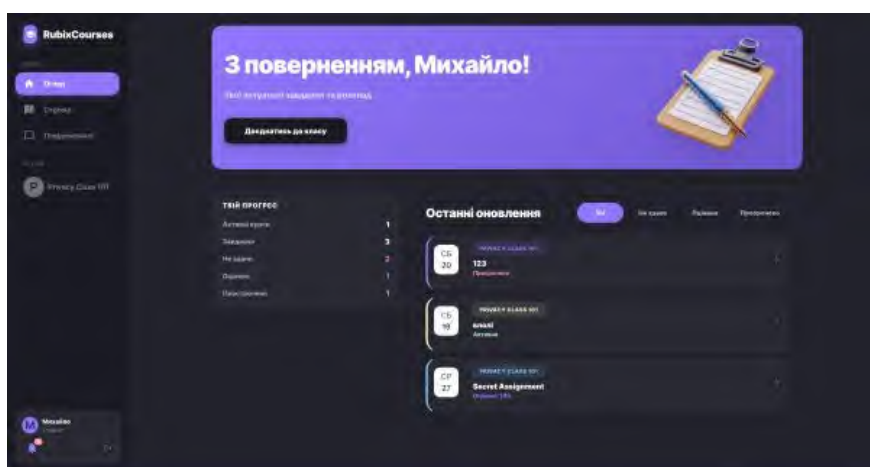


Рисунок 4.12 – Сторінка роботи з дисциплінами

Візуалізація робочих процесів та аналітичних даних користувача у відповідних програмних модулях представлена на наступних рисунках:

- екран безпосереднього виконання тестового завдання студентом показано на рис. 4.13;
- модуль фіксації результатів та успішності користувача наведено на рис. 4.14;
- вікно обміну текстовими повідомленнями підсистеми зв'язку показано на рис. 4.15.

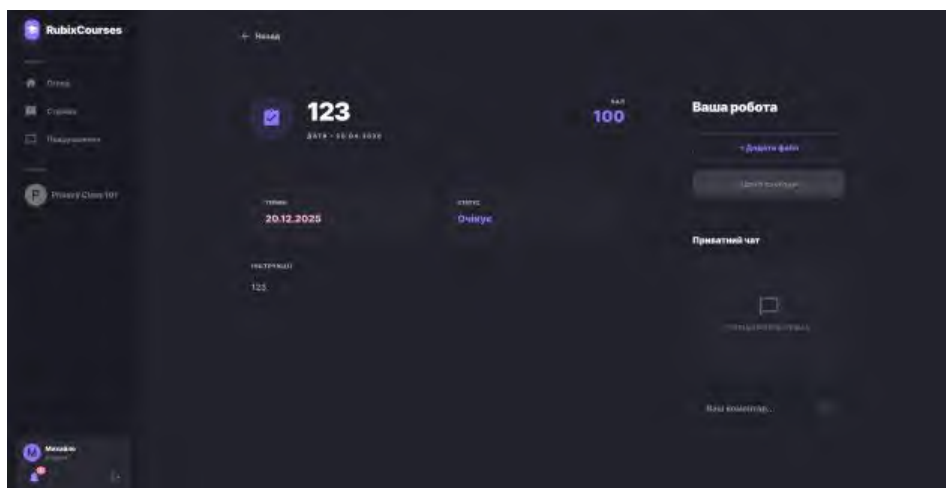


Рисунок 4.13 – Сторінка проходження завдання курсу

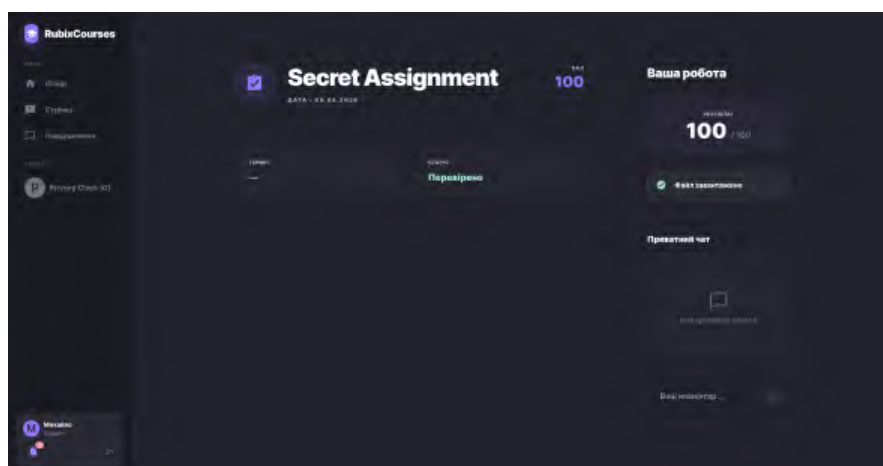


Рисунок 4.14 – Сторінка збереження результатів проходження курсу

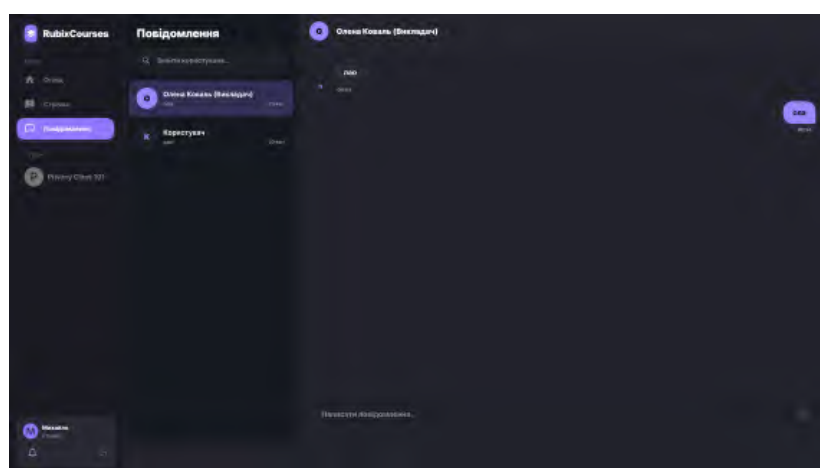


Рисунок 4.15 – Сторінка повідомлень

Результати випробувань підтвердили відповідність прийнятих архітектурних рішень встановленим практичним сценаріям використання. Клієнтський шар на React та серверне ядро Node.js функціонують як єдиний стабільний механізм. Створений програмний продукт готовий до повноцінної експлуатації в умовах реального навчального процесу.

Важливим інженерним етапом стала перевірка збереження прогресу при раптовому припиненні мережевого з'єднання. Нестабільний інтернет або раптові вимкнення живлення не повинні призводити до повної втрати результатів роботи. Для верифікації стійкості було штучно імітовано обрив зв'язку під час транзакції до бази даних. Експеримент підтвердив що асинхронна модель Node.js та механізми валідації Mongoose успішно запобігають появі пошкоджених документів. Система коректно обробляє винятки та виводить інформаційне сповіщення користувачу замість тривалого блокування інтерфейсу.

Тестування також дозволило перевірити стабільність Single Page Application при інтенсивному перемиканні між розділами. Дані таблиці 4.1 підтверджують ефективність ізоляції логіки в межах модульної структури компонентів React. Оновлення списку завдань у модулі курсів не викликає надлишкових рендерів у суміжних віджетах. Це дозволяє суттєво економити апаратні ресурси при використанні платформи на малопотужних пристроях.

Контроль верифікації прав доступу підтвердив високу надійність захисту системи від несанкціонованих дій. Спроби видалення завдань через прямі запити до API з боку студентів повністю блокуються. Спеціалізоване серверне middleware успішно перехоплює такі транзакції та повертає стандартизований код помилки 403 Forbidden. Це гарантує цілісність структури курсу та безпеку адміністрування навчального матеріалу викладачем.

Наступним кроком після функціональної перевірки став комплексний аналіз продуктивності платформи під навантаженням. Система повинна зберігати стабільність при одночасному зверненні великої кількості активних

користувачів. Оцінка швидкості генерації інтерфейсу здійснювалася за допомогою інструментів розробника Chrome DevTools. Завдяки оптимізації React оновлення сторінок при отриманні даних відбувається в межах 60–100 мілісекунд. Дана швидкість є повністю комфортною для візуального сприйняття інформації кінцевим користувачем. Індикатор успішного виконання завдання з'являється в інтерфейсі безпосередньо в момент завершення тесту.

Стрес-тестування зв'язки Express та MongoDB передбачало імітацію серії послідовних швидких запитів до API. Така поведінка є типовою при хаотичному кліканні користувача по елементах меню. Графік затримки відповідей сервера при пікових навантаженнях наведено на рис. 4.16.



Рисунок 4.16 – Графік затримки відповідей сервера при пікових навантаженнях

Аналіз кривої на рис. 4.16 підтверджує ефективність черг запитів на бекенді. Механізми обробки помилок на фронтенді успішно справляються із високою інтенсивністю звернень. Платформа зберігає загальну стабільність інтерфейсу та послідовно розподіляє навантаження без критичних збоїв.

Перевірка системи на введення некоректних символів підтвердила надійність впровадженої валідації схем Mongoose. Спроби передачі надлишкових обсягів тексту або спеціальних ін'єкцій блокуються на рівні сервера. Жорстке налаштування типів даних надійно захищає сховище від накопичення системного сміття.

Проведені випробування довели високу працездатність та інженерну зрілість розробленого вебзастосунку. Платформа повністю відповідає всім встановленим критеріям сучасного інтерактивного освітнього середовища. Отримані позитивні результати дозволяють завершити практичну частину роботи та перейти до формулювання загальних висновків.

4.6 Висновки до розділу 4

У четвертому розділі роботи повністю завершено етап експериментального випробування та комплексної оцінки ефективності розробленого програмного комплексу. Проведені дослідження охоплювали моніторинг показників швидкодії та всебічну верифікацію ключових функціональних контурів платформи RubixCourses.

За результатами проведеного тестування та аналізу інтерфейсів було отримано наступні науково технічні результати:

- підтверджено повну працездатність усіх базових функцій та сценаріїв використання платформи;
- верифіковано коректність інтеграції з Firebase для надійного розмежування встановлених ролей доступу;
- доведено стабільність функціонування системи статусів завдань та інтерактивних колірних ідентифікаторів;
- підтверджено високу реактивність комунікаційного чату реального часу без виникнення мережових затримок;
- фіксовано високі показники швидкодії сервера із часом обробки запитів менше ста мілісекунд.

Часові метрики початкової генерації інтерфейсу в межах двох секунд повністю відповідають встановленим технічним вимогам. Розроблена архітектура успішно забезпечує підтримання високої концентрації уваги студентів під час навчання.

Головним практичним результатом розділу є доведення високої цілісності створеного електронного освітнього середовища. Всі розроблені модулі функціонують як єдиний скоординований інженерний комплекс у реальних умовах експлуатації. Платформа повністю готова до впровадження в академічний процес та подальшого масштабування.

ВИСНОВКИ

У результаті виконання дипломної роботи було успішно спроектовано та програмно реалізовано вебзастосунок інтерактивного освітнього середовища. Розроблений програмний продукт орієнтований на забезпечення ефективної підтримки процесу навчання. Платформа реалізує модель реактивної взаємодії між користувачем навчальним контентом та компонентами системи. Статична передача електронних документів замінюється створенням адаптивного середовища для активного залучення студентів до освітнього процесу.

Під час теоретичного дослідження було проведено детальний аналіз популярних освітніх платформ Moodle, Coursera та Udemu. Кожна з аналізованих систем демонструє специфічні функціональні переваги та певні структурні обмеження. Платформа Moodle забезпечує високу гнучкість конфігурації але потребує значних часових витрат на адміністрування. Система Coursera пропонує ергономічний користувацький досвід при низькому рівні відкритості архітектури для кастомізації. Платформа Udemu характеризується високою доступністю матеріалів при пасивному характері взаємодії користувача з контентом. Розробка інтерактивного середовища покликана подолати проблему дефіциту залученості здобувачів освіти в межах дистанційного навчання.

Розроблена платформа RubixCourses успішно об'єднує переваги існуючих рішень та мінімізує їхні архітектурні недоліки. Для практичної реалізації було обґрунтовано та впроваджено сучасний узгоджений технологічний стек розробки. Програмний комплекс базується на використанні можливостей React, Node.js, Express, MongoDB та Firebase.

Застосування даних інструментів дозволило отримати наступні практичні результати:

- реалізація компонентного підходу бібліотеки React для спрощення підтримки та повторного використання коду;

- застосування гнучкої документо-орієнтованої NoSQL моделі MongoDB для ефективного збереження різноструктурного контенту;
- інтеграція хмарного сервісу Firebase Authentication для побудови промислового контуру безпеки та захисту даних;
- створення безстановочного програмного інтерфейсу REST API для забезпечення високої швидкодії та масштабованості.

Проведене всебічне функціональне тестування повністю підтвердило працездатність усіх основних модулів розробленого застосунку. В ході випробувань успішно верифіковано сценарії автентифікації користувачів створення курсів та проходження тестування. Система забезпечує стабільне автоматичне збереження метрик індивідуального прогресу навчання у базі даних. Програмний комплекс функціонує як цілісний завершений продукт і повністю відповідає встановленим технічним критеріям. Створене освітнє середовище демонструє високу стійкість до пікових навантажень та повністю готове до практичного впровадження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Coursera URL: <https://www.coursera.org/> (data of access 15.05.2026).
2. MDN Web Docs. Single-page Application URL: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>. (data of access 15.05.2026).
3. Udemy URL: <https://www.udemy.com/> (data of access 15.05.2026).
4. React Documentation. Thinking in React URL: <https://react.dev/learn/thinking-in-react/> (data of access 15.05.2026).
5. Osmani A. Learning JavaScript Design Patterns. 2nd ed. O'Reilly Media, 2023. 310 p.
6. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
7. Express.js Documentation. URL: <https://expressjs.com/en/guide/routing.html>. (data of access 15.05.2026).
8. Frisbie M. Professional JavaScript for Web Developers. 4th ed. Wrox, 2019. 1200 p.
9. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide. O'Reilly Media, 2020. 514 p.
10. Dodds K. C. Epic React URL: <https://epicreact.dev>. (data of access 15.05.2026).
11. Moodle URL: <https://moodle.org/?lang=uk/> (data of access 15.05.2026).
12. Firebase Documentation. Firestore Data Model URL: <https://firebase.google.com/docs/firestore/data-model>. (data of access 15.05.2026).
13. Moroney L. The Definitive Guide to Firebase. Apress, 2017. 332 p.
14. Abramov D. React as a UI Runtime URL: <https://overreacted.io/react-as-a-ui-runtime>. (data of access 15.05.2026).
15. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison–Wesley, 1994. 395 p.

16. MongoDB Documentation. Data Modeling Introduction URL: <https://www.mongodb.com/docs/manual/core/data-modeling-introduction>. (data of access 15.05.2026).
17. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. 560 p.
18. Grabowski M. React Router v6 in Depth // Frontend Masters Blog URL: <https://frontendmasters.com/blog/react-router>. (data of access 15.05.2026).
19. Sadalage P. J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison–Wesley, 2012. 192 p.
20. Firebase Documentation. Authentication URL: <https://firebase.google.com/docs/auth>. (data of access 15.05.2026).

ДОДАТОК А
Технічне завдання

Вступ

Вебзастосунок «RubixCourses» призначений для автоматизації освітнього процесу та створення інтерактивного середовища для навчання в режимі онлайн. Він розроблений для використання закладами освіти, викладачами та студентами, які потребують зручного інструменту для організації навчальних курсів, відстеження прогресу та забезпечення взаємодії між учасниками навчального процесу. Область застосування охоплює дистанційне навчання, університетські курси та корпоративні тренінги, а об'єктом використання є персональні комп'ютери та мобільні пристрої з доступом до мережі Інтернет через сучасний веббраузер.

A.1 Підстави до розробки

Підставою для розробки є завдання на дипломну роботу, видане 20 квітня 2026 року. Найменування теми розробки – «Інженерія програмного забезпечення інтерактивного освітнього середовища».

A.2 Призначення розробки

Застосунок призначений для автоматизації процесів управління навчальними курсами, проходження тестових завдань, відстеження навчального прогресу та забезпечення комунікації між викладачами і студентами. Його експлуатаційне призначення полягає у підвищенні ефективності дистанційного навчання шляхом забезпечення миттєвого зворотного зв'язку, зручної навігації та персоналізованого доступу до навчального контенту.

А.3 Вимоги до програми чи програмного виробу

А.3.1 Вимоги до функціональних характеристик

Вебзастосунок «RubixCourses» розроблено для виконання широкого спектра завдань, пов'язаних з організацією та проходженням навчальних курсів. Користувачі можуть реєструватися та автентифікуватися через електронну пошту з паролем, що забезпечує безпечний доступ до системи. Після авторизації система автоматично визначає роль користувача – викладач або студент – і надає відповідний інтерфейс та набір функцій.

Управління навчальним контентом є ключовою функцією застосунку. Викладачі мають можливість створювати курси, додавати навчальні матеріали, формувати тестові завдання та встановлювати терміни їх виконання. Студенти отримують доступ до призначених курсів, можуть переглядати матеріали, проходити тести та відстежувати власний прогрес у режимі реального часу.

Застосунок забезпечує інструменти для оцінювання та аналізу результатів навчання. Викладачі можуть переглядати статистику виконання завдань по кожному студенту та курсу, а студенти – отримувати миттєвий зворотний зв'язок після проходження тестів. Для забезпечення комунікації між учасниками навчального процесу реалізовано систему повідомлень у реальному часі на базі Firebase Realtime Database.

Початковими даними для роботи застосунку є навчальні матеріали та завдання, що вводяться викладачами, а також відповіді та дії студентів у процесі навчання. На основі цих даних формуються вихідні результати: звіти про прогрес, оцінки за тести та повідомлення системи.

А.3.2 Вимоги до надійності

Для забезпечення надійного функціонування застосунку необхідно дотримуватися кількох умов. Користувачі відповідають за наявність

стабільного підключення до мережі Інтернет на пристроях, з яких здійснюється доступ до системи. Використання ліцензованого програмного забезпечення, зокрема React, Node.js та Firebase, є обов'язковим для стабільної роботи серверної та клієнтської частин.

Застосунок здатний обробляти нештатні ситуації для мінімізації незручностей для користувачів. У разі некоректного введення даних система повертає зрозуміле повідомлення про помилку без переривання роботи. У разі закінчення терміну дії сесії користувач отримує пропозицію повторно автентифікуватися без втрати поточних даних.

А.3.3 Умови експлуатації

Застосунок повинен стабільно працювати у стандартних умовах експлуатації персональних комп'ютерів і мобільних пристроїв. Користувач має володіти базовими навичками роботи з вебзастосунками: реєстрація, введення даних, навігація між розділами. Обслуговування не потребує спеціального технічного персоналу на стороні користувача – оновлення системи відбуваються на серверній стороні без участі кінцевого користувача.

А.3.4 Вимоги до складу і параметрів технічних засобів

Для роботи застосунку необхідний пристрій із такими мінімальними характеристиками:

- операційна система: будь-яка сучасна ОС із підтримкою актуального веббраузера (Windows 10+, macOS 11+, Android 8+, iOS 14+);
- процесор: частота не нижче 1.5 ГГц;
- оперативна пам'ять: мінімум 2 ГБ;
- вільне місце на диску: не менше 200 МБ для кешування браузера;
- підключення до мережі Інтернет зі швидкістю не менше 5 Мбіт/с для стабільної роботи в режимі реального часу;

- веббраузер: Google Chrome.

A.3.5 Вимоги до інформаційної й програмної сумісності

Для забезпечення інформаційної сумісності всі дані між клієнтською та серверною частинами передаються у форматі JSON через REST API. Програмна реалізація клієнтської частини базується на бібліотеці React із використанням мови JavaScript. Серверна частина реалізована на платформі Node.js із фреймворком Express.js.

Для зберігання даних використовується документно–орієнтована база даних MongoDB, що забезпечує гнучкість при роботі з різноструктурованим навчальним контентом. Автентифікація та управління сесіями реалізовані на платформі Firebase Authentication із підтримкою JSON Web Token (JWT). Для реалізації системи повідомлень у режимі реального часу використовується Firebase Realtime Database. Безпека інформації забезпечується передачею даних виключно по протоколу HTTPS та верифікацією токенів доступу на серверній стороні при кожному запиті.

A.3.6 Вимоги до маркування й пакування

Застосунок розгортається на хмарній інфраструктурі та є доступним для користувачів через мережу Інтернет за відповідною адресою. Вимоги до фізичного маркування чи пакування відсутні, оскільки продукт є цифровим вебзастосунком.

A.3.7 Вимоги до транспортування й збереження

Оскільки застосунок розповсюджується та функціонує у цифровому вигляді через мережу Інтернет, вимоги до транспортування та фізичного

збереження не застосовуються. Збереження даних забезпечується хмарною інфраструктурою MongoDB Atlas та Firebase.

A.4 Вимоги до програмної документації

Програмна документація до застосунку включає пояснювальну записку та технічне завдання. Ці документи містять вичерпну інформацію про архітектуру системи, використані технології, функціональні вимоги та результати тестування.

A.5 Техніко–економічні показники

Розроблений застосунок має значний потенціал для підвищення ефективності навчального процесу. Очікується, що впровадження системи дозволить скоротити час на адміністрування навчальних курсів та перевірку завдань завдяки автоматизації ключових процесів. Застосунок реалізований на базі відкритих технологій (React, Node.js, MongoDB), що мінімізує витрати на ліцензування програмного забезпечення. Економічні переваги полягають у можливості організації повноцінного дистанційного навчання без додаткових інфраструктурних витрат.

A.6 Стадії й етапи розробки

Розробка включає такі етапи:

- постановка завдання (1 тиждень);
- аналіз предметної області та дослідження існуючих рішень (1 тиждень);
- вибір технологій та проектування архітектури (2 тижні);
- розробка клієнтської та серверної частин (3–4 тижні);

- тестування та налагодження програмного забезпечення (1 тиждень);
- оформлення документації (1 тиждень).

A.7 Порядок контролю та приймання

Процес перевірки працездатності застосунку передбачає кілька етапів тестування. Функціональне тестування дозволяє переконатися в коректній роботі основних модулів системи: авторизації, управління курсами, проходження тестів та збереження результатів. Тестування API зосереджено на перевірці коректності відповідей серверної частини та стабільності взаємодії між компонентами. Окремо проводиться тестування інтерфейсу користувача для забезпечення коректного відображення на пристроях різних типів та розмірів екрану.

Працездатність застосунку перевіряється у браузерях Google Chrome, Mozilla Firefox та Safari, а також на пристроях із різними розмірами екранів – настільних комп'ютерах, планшетах і смартфонах. Після успішного завершення всіх етапів тестування та підтвердження відповідності вимогам технічного завдання програмний продукт передається на захист дипломної роботи.

ДОДАТОК Б
Текст програми

Б.1 Файл app.jsx

У цьому файлі реалізовано головну сторінку:

```
// src/App.jsx – Routing з публічною головною сторінкою
import React from 'react';
import { Routes, Route, Navigate } from 'react-router-dom';
import { useAuth } from './context/AuthContext';

import Layout from './components/shared/Layout';
import HomePage from './components/HomePage';
import Login from './components/auth/Login';
import Register from './components/auth/Register';
import ForgotPassword from './components/auth/ForgotPassword';
import ProtectedRoute from './components/auth/ProtectedRoute';
import TeacherDashboard from './components/teacher/TeacherDashboard';
import TeacherAssignmentDetail from './components/teacher/AssignmentDetail';
import StudentDashboard from './components/student/StudentDashboard';
import StudentAssignmentDetail from './components/student/AssignmentDetail';
import GradeView from './components/student/GradeView';
import ClassStream from './components/shared/ClassStream';
import CourseDetail from './components/shared/CourseDetail';
import MessagesPage from './components/shared/MessagesPage';

export default function App() {
  const { user, role } = useAuth();

  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        { /* Публічна головна сторінка – каталог курсів */ }
```

```
<Route index element={<HomePage />} />
```

```
{/* Сторінка входу */}
```

```
<Route
```

```
  path="login"
```

```
  element={
```

```
    user
```

```
    ? <Navigate to={role === 'teacher' ? '/teacher' : '/student'} replace
```

```
/>
```

```
    : <Login />
```

```
  }
```

```
/>
```

```
{/* Сторінка реєстрації */}
```

```
<Route
```

```
  path="register"
```

```
  element={
```

```
    user
```

```
    ? <Navigate to={role === 'teacher' ? '/teacher' : '/student'} replace
```

```
/>
```

```
    : <Register />
```

```
  }
```

```
/>
```

```
{/* Сторінка відновлення паролю */}
```

```
<Route
```

```
  path="forgot-password"
```

```
  element={
```

```
    user
```

```

? <Navigate to={role === 'teacher' ? '/teacher' : '/student'} replace
/>

: <ForgotPassword />

}

/>

```

Б.2 Файл firebase.jsx

У цьому файлі реалізовано підключення фایрбейс:

```

// src/firebase/firebase.js
// Ініціалізація Firebase (Auth, Firestore, Storage)
// Читає ключі з .env (VITE_ prefix). Якщо ключів немає – gracefully деградує.

import { initializeApp, getApps } from 'firebase/app';
import { getAuth, GoogleAuthProvider } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';
import { getStorage } from 'firebase/storage';

const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId:
import.meta.env.VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID,
};

// Graceful деградація – не блокуємо рендеринг якщо .env не заповнений
let app, auth, db, storage, googleProvider;

```

```

try {
  // Ініціалізуємо лише якщо ще не зроблено (HMR-safe)
  app = getApps().length === 0 ? initializeApp(firebaseConfig) : getApps()[0];
  auth = getAuth(app);
  googleProvider = new GoogleAuthProvider();
  db = getFirestore(app);
  storage = getStorage(app);
} catch (e) {
  console.warn(
    ' ⚠️ Firebase не ініціалізувався. Перевірте файл .env\n' +
    'Скопіюйте .env.example → .env і вставте ключі з Firebase Console.\n' +
    `Помилка: ${e.message}`
  );
  // Заглушки щоб модуль не зломився при імпорті
  auth = null;
  db = null;
  storage = null;
  googleProvider = null;
}

export { auth, googleProvider, db, storage };
export default app;

```

Б.3 Файл register.jsx

У цьому файлі реалізовано реєстрацію:

```

import React, { useState } from 'react';
import { useNavigate, Link as RouterLink } from 'react-router-dom';
import {

```

```

Box, Button, FormControl, InputLabel, MenuItem,
Select, Stack, TextField, Typography, Alert, CircularProgress,
Paper, Link, InputAdornment, IconButton, Container
} from '@mui/material';
import SchoolIcon from '@mui/icons-material/School';
import Visibility from '@mui/icons-material/Visibility';
import VisibilityOff from '@mui/icons-material/VisibilityOff';
import { useAuth } from '../context/AuthContext';

const LAVENDER = '#9A89FF';
const TEXT_S = '#94A3B8';
const SURFACE_DARK = '#1E232B';
const SHELL_DARK = '#161A21';

export default function Register() {
  const navigate = useNavigate();
  const { signup } = useAuth();

  const [formData, setFormData] = useState({
    displayName: "",
    email: "",
    password: "",
    confirmPassword: "",
    role: 'student'
  });

  const [showPassword, setShowPassword] = useState(false);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState("");

  const handleChange = (e) => {

```

```

    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const validate = () => {
    if (!formData.displayName.trim()) return 'Будь ласка, введіть ваше ім\'я';
    if (!formData.email.trim()) return 'Будь ласка, введіть email';
    if (!/^S+@\S+\.\S+/.test(formData.email)) return 'Невірний формат email';
    if (formData.password.length < 6) return 'Пароль має бути не менше 6
символів';
    if (formData.password !== formData.confirmPassword) return 'Паролі не
співпадають';
    return null;
  };

  async function handleSubmit(e) {
    e.preventDefault();
    const validationError = validate();
    if (validationError) {
      setError(validationError);
      return;
    }

    setLoading(true);
    setError('');

    try {
      await signup(formData.email, formData.password, formData.role,
formData.displayName);
      navigate(formData.role === 'teacher' ? '/teacher' : '/student');
    } catch (e) {

```

```

        setError(e.message || 'Помилка реєстрації. Спробуйте пізніше.');
```

```

    } finally {
        setLoading(false);
    }
}

return (
    <Box
        sx={{
            minHeight: '100vh',
            bgcolor: '#0F1216',
            display: 'flex',
            alignItems: 'center',
            justifyContent: 'center',
            position: 'relative',
            overflow: 'hidden',
            py: 8
        }}
    >
        { /* Ambient Background Glows */ }
        <Box sx={{
            position: 'absolute', top: '-10%', right: '-10%',
            width: 600, height: 600, borderRadius: '50%',
            background: `radial-gradient(circle, ${LAVENDER} 15 0%, transparent
70%)`,
            pointerEvents: 'none',
        }} />
        <Box sx={{
            position: 'absolute', bottom: '-10%', left: '-10%',
            width: 600, height: 600, borderRadius: '50%',

```

```
background: `radial-gradient(circle, #FF8BA710 0%, transparent 70%)`,
pointerEvents: 'none',
}} />
```

```
<Container maxWidth="sm">
```

```
<Paper
```

```
elevation={0}
```

```
sx={{
```

```
p: { xs: 4, sm: 6 },
```

```
bgcolor: SHELL_DARK,
```

```
borderRadius: '40px',
```

```
border: '1px solid rgba(255,255,255,0.06)',
```

```
boxShadow: '0 40px 100px rgba(0,0,0,0.5)',
```

```
position: 'relative',
```

```
zIndex: 1
```

```
}}
```

```
>
```

```
<Box sx={{ textAlign: 'center', mb: 5 }}>
```

```
<Box sx={{ display: 'inline-flex', p: 1.5, borderRadius: '16px',
```

```
bgcolor: 'rgba(154, 137, 255, 0.1)', border: `1px solid ${LAVENDER}30`, mb: 3
```

```
}}>
```

```
<SchoolIcon sx={{ color: LAVENDER, fontSize: 32 }} />
```

```
</Box>
```

```
<Typography variant="h4" sx={{ fontWeight: 900, color: '#fff', mb:
1, letterSpacing: '-1px' }}>
```

```
Почати навчання
```

```
</Typography>
```

```
<Typography sx={{ color: TEXT_S, fontWeight: 700 }}>
```

```
Створіть свій безкоштовний акаунт
```

```
</Typography>
```

```
</Box>
```

```
    {error && <Alert severity="error" sx={{ mb: 4, borderRadius: '16px',
fontWeight: 700 }}>{error}</Alert>}
```

```
<Box component="form" onSubmit={handleSubmit}>
```

```
  <Stack spacing={2.5}>
```

```
    <TextField
```

```
      fullWidth label="Повне ім'я" name="displayName"
```

```
      variant="outlined"
```

```
      value={formData.displayName} onChange={handleChange}
```

```
      required
```

```
      sx={{ '& .MuiOutlinedInput-root': { borderRadius: '16px' } }}
```

```
    />
```

```
    <TextField
```

```
      fullWidth label="Email" name="email" type="email"
```

```
      variant="outlined"
```

```
      value={formData.email} onChange={handleChange} required
```

```
      sx={{ '& .MuiOutlinedInput-root': { borderRadius: '16px' } }}
```

```
    />
```

```
<FormControl fullWidth>
```

```
  <InputLabel id="role-label" sx={{ color: TEXT_S }}>Я
```

```
реєструюсь як</InputLabel>
```

```
  <Select
```

```
    labelId="role-label" name="role" value={formData.role}
```

```
    label="Я реєструюсь як" onChange={handleChange}
```

```
    sx={{ borderRadius: '16px', color: '#fff' }}
```

```
  >
```

```
    <MenuItem value="student">Студент</MenuItem>
```

```
    <MenuItem value="teacher">Викладач</MenuItem>
```

```

        </Select>
    </FormControl>
    <TextField
        fullWidth label="Пароль" name="password"
type={showPassword ? 'text' : 'password'} variant="outlined"
        value={formData.password} onChange={handleChange}
required

        InputProps={{
            sx: { borderRadius: '16px' },
            endAdornment: (
                <InputAdornment position="end">
                    <IconButton onClick={() =>
setShowPassword(!showPassword)} edge="end" sx={{ color: TEXT_S }}>
                        {showPassword ? <VisibilityOff /> : <Visibility />}
                    </IconButton>
                </InputAdornment>
            ),
        }}
    />
    <TextField
        fullWidth label="Підтвердіть пароль"
name="confirmPassword" type={showPassword ? 'text' : 'password'}
variant="outlined"
        value={formData.confirmPassword}
onChange={handleChange} required
        sx={{ '& .MuiOutlinedInput-root': { borderRadius: '16px' } }}
    />

    <Button
        type="submit" variant="contained" disabled={loading}

```

```

        sx={{
          py: 2, bgcolor: LAVENDER, borderRadius: '16px',
fontWeight: 900,
          fontSize: '1.1rem', textTransform: 'none', mt: 2,
          '&:hover': { bgcolor: LAVENDER, opacity: 0.9 },
          boxShadow: `0 12px 24px ${LAVENDER}30`
        }}
      >
      {loading ? <CircularProgress size={24} sx={{ color: '#fff' }}
/> : 'Зареєструватися'}
    </Button>

    <Box sx={{ mt: 3, textAlign: 'center' }}>
      <Typography variant="body2" sx={{ color: TEXT_S,
fontWeight: 700 }}>
        Вже маєте акаунт? {' '}
        <Link component={RouterLink} to="/login" sx={{ color:
LAVENDER, fontWeight: 800, textDecoration: 'none', '&:hover': {
textDecoration: 'underline' } }}>
          Увійти
        </Link>
      </Typography>
    </Box>
  </Stack>
</Box>
</Paper>
</Container>
</Box>
);

```

ДОДАТОК В
Слайди презентації

Дипломна кваліфікаційна робота бакалавра
Тема: “Інженерія програмного забезпечення
інтерактивного освітнього середовища”

ВИКОНАВ
СТУДЕНТ 4 КУРСУ, ГРУПИ КНТ-132

МАКСИМ ГОРЛОВ

КЕРІВНИК
К.Т.Н ДОЦЕНТ

СЕРГІЙ ЛЕОЩЕНКО

Рисунок В.1 – Слайд 1

МЕТА ТА ЦІЛЬ

ОСНОВНА МЕТА ДАНОЇ ПРЕЗЕНТАЦІЇ ЦЕ
РОЗГЛЯНУТИ ПРОГРАМНИЙ ПРОДУКТ
«RUBIXCOURSES»

ДЛЯ ДОСЯГНЕННЯ ЦЬОЇ МЕТИ У РОБОТІ
ВИРІШУЄТЬСЯ РЯД ЗАДАЧ

- ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ;
- РОЗГЛЯД ВИМОГ;
- РОЗГЛЯД ФУНКЦІОНАЛЬНОЇ МОДЕЛІ;
- РОЗГЛЯД ГОТОВОГО ПРОДУКТУ;

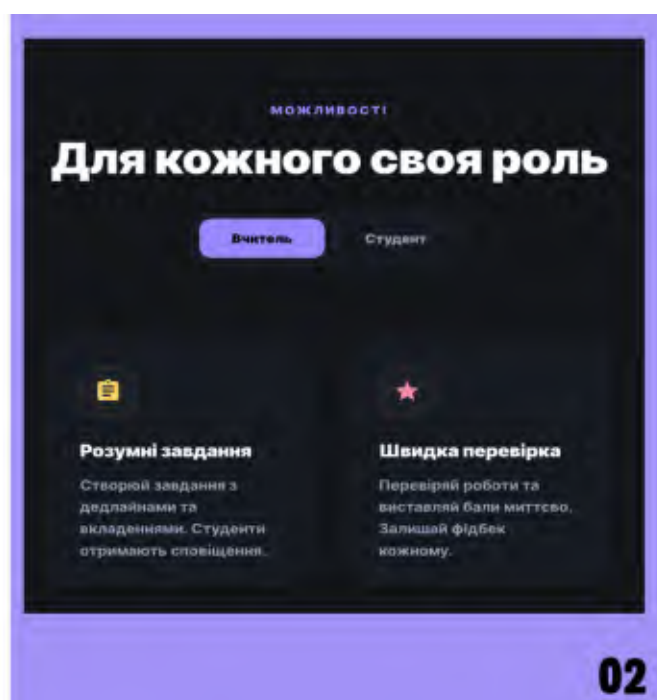


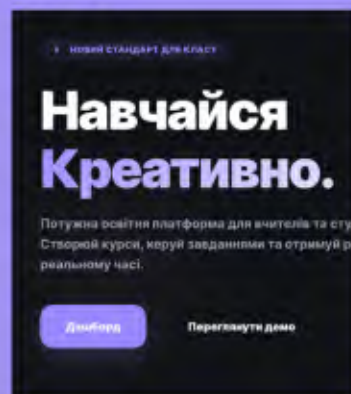
Рисунок В.2 – Слайд 2

МЕТА ПРОЕКТУ, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета: підвищення рівня автоматизованості освітнього процесу за рахунок розробки інтерактивного освітнього середовища з використанням сучасних вебтехнологій.

Об'єкт дослідження: процес розроблення програмного забезпечення для інтерактивних освітніх середовищ.

Предмет дослідження: методи та інструменти розробки веборієнтованого програмного забезпечення для створення інтерактивного освітнього середовища.



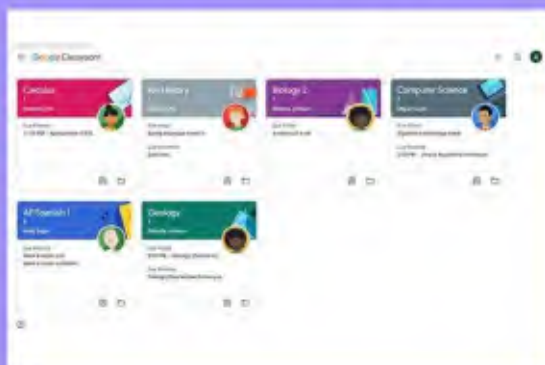
RUBIXCOURSES

03

Рисунок В.3 – Слайд 3

ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

RUBIXCOURSES



04

Рисунок В.4 – Слайд 4

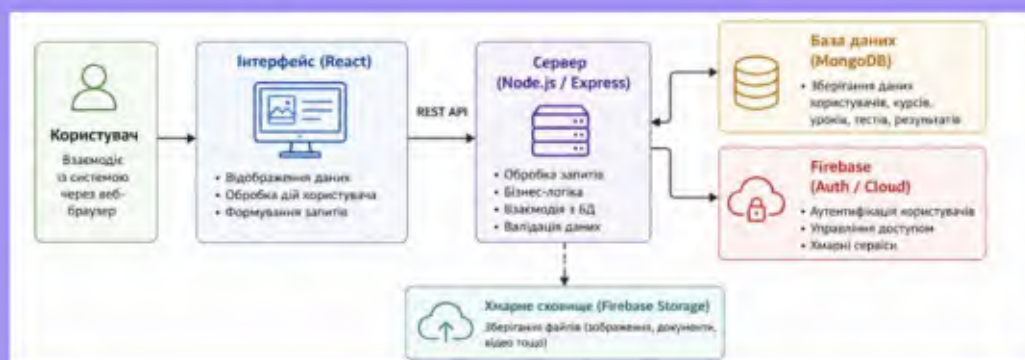
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Платформа	Ключові переваги	Системні недоліки	Архітектурний стек	Рішення в RubixCourses
Moodle	Гнучкість, відкритий код, кастомізація.	Перевантажений інтерфейс, складна м'ясо.	Моноліт, часті перезавантаження.	Безшовний SPA на React, чистий дизайн.
Coursera	Якісний контент, точний прогрес-трекінг.	Статичність, затримка зв'язку, немає чатів.	Контентний моноліт зі статичним тренінгом.	Модульна структура + синхронізація через Firebase.
Udemy	Просте створення курсів, зручний UX "вітрини".	Фокус на продажах, реклама та банери.	Комерційний веб-інтерфейс, лінійні списки.	Легка навігація, повне очищення від реклами, micro-learning.
RubixCourses	Повна інтерактивність, миттєвий фідбек.	Номас (відсичено весь зайвий функціонал).	Реактивний MERN-стек + Firebase Realtime.	Баланс модульності, ланонічності та мікроназначення.

05

Рисунок В.5 – Слайд 5

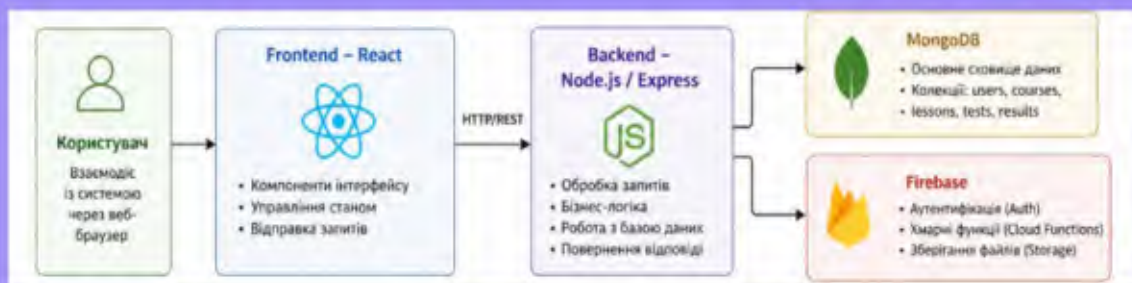
АРХІТЕКТУРА ІНТЕРАКТИВНОГО ОСВІТНЬОГО ВЕБЗАСТОСУНКУ



06

Рисунок В.6 – Слайд 6

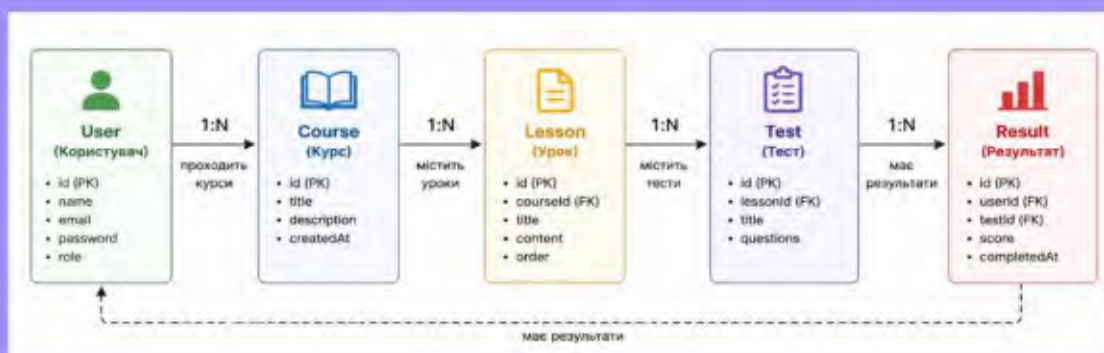
ЗАГАЛЬНА АРХІТЕКТУРА СИСТЕМИ



07

Рисунок В.7 – Слайд 7

МОДЕЛЬ ДАНИХ СИСТЕМИ



08

Рисунок В.8 – Слайд 8

ТЕСТУВАННЯ СИСТЕМИ

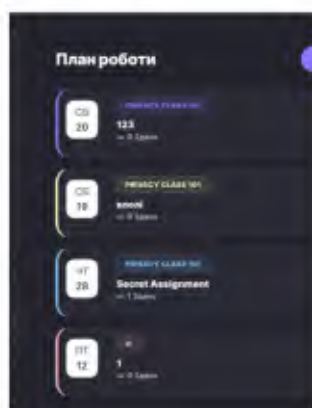
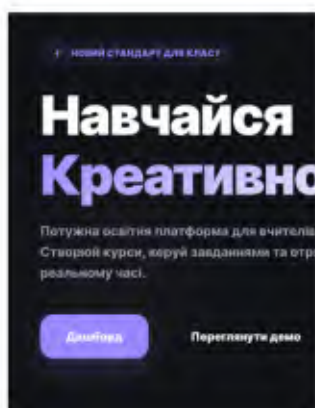
АНАЛІЗ КРИВОЇ НА РИСУНКУ ПІДТВЕРДЖУЄ ЕФЕКТИВНІСТЬ ЧЕРГ ЗАПИТІВ НА БЕКЕНДІ.



09

Рисунок В.9 – Слайд 9

СИСТЕМА

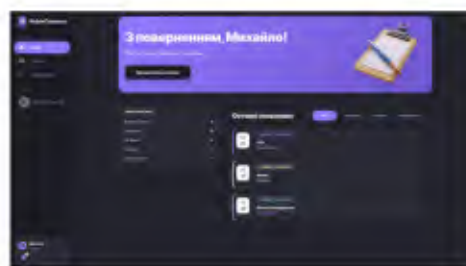
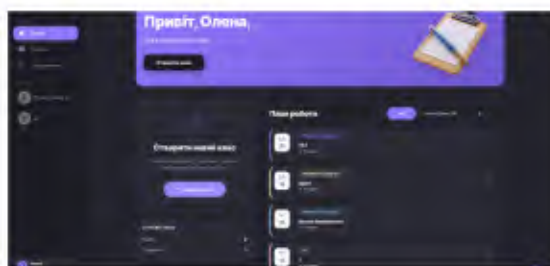


В ДАНОМУ РОЗДІЛІ
РОЗГЛЯНЕМО ЯК ВИГЛЯДАЄ
ЗАСТОСУНОК

10

Рисунок В.10 – Слайд 10

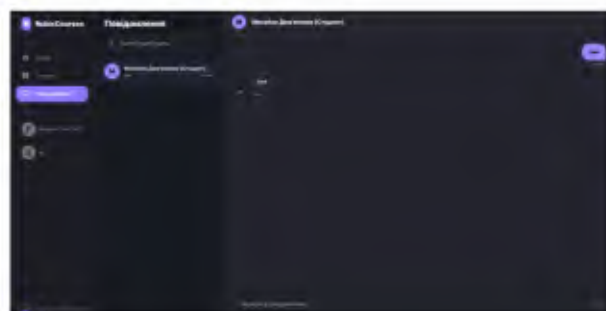
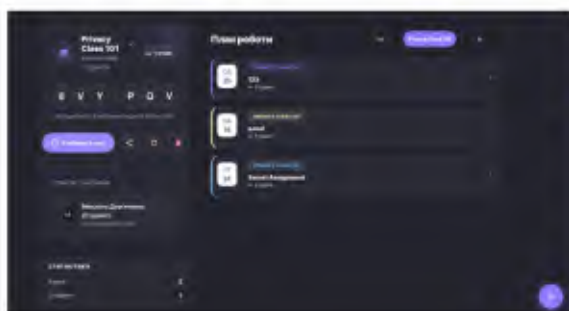
СТОРІНКА ВИКЛАДАЧА ТА СТУДЕНТА



11

Рисунок В.11 – Слайд 11

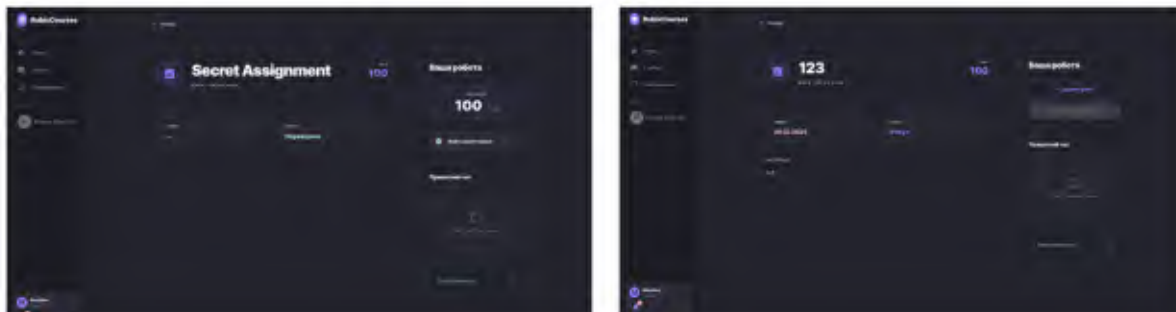
СТОРІНКИ ДЛЯ КЛАСУ ТА ЧАТУ



12

Рисунок В.12 – Слайд 12

СТОРІНКА ПЕРЕГЛЯДУ ЗДАЧІ ЗАВДАННЯ



13

Рисунок В.13 – Слайд 13

ВИСНОВКИ ТА ПЕРСПЕКТИВИ

Проведено комплексну роботу де розроблено застосунок ефективного освітнього середовища, в якому легко орієнтуватись, де система працює швидко та стабільно забезпечена легка комунікація між викладачем та студентом що пришвидшує процес навчання та допомагає студентам краще навчатись в перспективах розвитку це інтеграція ШІ помічника який буде допомагати в навчанні.

14

Рисунок В.14 – Слайд 14