МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА НАВЧАЛЬНОЇ ПЛАТФОРМИ З ВИКОРИСТАННЯМ CLOJURE»**

Виконав: студент 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
О.О. Чернов
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.пед.н. Пшенична О.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Чернову Олександру Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка навчальної платформи з використанням Clojure

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження предметної області та постановка задачі.
2. Проєктування навчальної платформи.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	13.01.2025	
2.	Збір вихідних даних.	27.01.2025	
3.	Обробка методичних та теоретичних	17.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	31.03.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2025	

Студент _____ О.О. Чернов
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка навчальної платформи з використанням Clojure»: 46 с., 27 рис., 8 джерел.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, СИСТЕМА, ФРЕЙМВОРК, ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ, API, BACK-END, CLOJURE, CLOJURESCRIPT, FRONT-END, LUMINUS.

Об’єкт дослідження – процес розробки навчальної платформи.

Предмет дослідження – мова програмування Clojure, фреймворк Luminus.

Мета роботи – спроєктувати, розробити і протестувати навчальну платформу з використанням Clojure.

Методи дослідження – аналіз літературних джерел, вивчення аналогів, методи проєктування програмного забезпечення, вебпрограмування, тестування.

У ході виконання роботи було проведено аналіз навчальних матеріалів та офіційної документації використаних інструментів. Було проаналізовано популярні застосунки, що використовуються для вивчення основ функціонального програмування та мови Clojure: 4ever-clojure, ClojureDocs, Try Clojure, Exercism. Здійснено такі кроки для проєктування системи: аналіз функціональних і нефункціональних вимог, концептуальне проєктування, логічне проєктування та фізичне проєктування. Спроєктована система була реалізована з використанням мови програмування Clojure і фреймворку Luminus для задач серверної частини, ClojureScript та BulmaCSS – для клієнтської частини застосунку.

Розроблена система є прикладом застосування Clojure та функціонального підходу програмування у створенні навчального програмного забезпечення.

SUMMARY

Bachelor's qualifying paper "Developing a Learning Platform Using Clojure": 46 pages, 27 figures, 8 references.

API, BACK-END, CLOJURE, CLOJURESCRIPT, DATABASE, FRAMEWORK, FRONT-END, FUNCTIONAL PROGRAMMING, LUMINUS, SYSTEM, WEB APPLICATION.

The object of the study is the process of developing an educational platform.

The subject of the research is the Clojure programming language and the Luminus framework.

The aim of the study is to design, develop, and test an educational platform using Clojure.

The research methods include analysis of literature sources, study of existing solutions, software design methods, web development, and software testing.

During the course of the project, an analysis of educational materials and official documentation for the chosen tools was conducted. Several popular applications for learning the basics of functional programming and the Clojure language were examined, including 4ever-clojure, ClojureDocs, Try Clojure, and Exercism. The following steps were carried out in the system design process: analysis of functional and non-functional requirements, conceptual design, logical design, and physical design. The designed system was implemented using the Clojure programming language and the Luminus framework for backend development, along with ClojureScript and BulmaCSS for the client-side of the application.

The developed system serves as an example of applying Clojure and the functional programming paradigm to the development of educational software.

ЗМІСТ

Завдання на кваліфікаційну роботу……..……..……...2
Реферат 4
Summary 5
Вступ..... 7
1 Дослідження предметної області..... 9
1.1 Огляд аналогічних рішень..... 9
1.2 Опис програмних засобів 10
1.2.1 Огляд фреймворку Luminus 10
1.2.2 Використання Clojurescript у фронтенд-розробці. 13
1.2.3 Використання бібліотек Reagent та Re-Frame..... 14
2 Проєктування навчальної платформи 17
2.1 Аналіз вимог 17
2.2 Використання UML під час розробки системи..... 18
2.3 Концептуальне проєктування 19
2.4 Логічна структура системи 25
2.5 Фізична структура системи..... 27
2.5.1 Діаграма компонентів..... 27
2.5.2 Діаграма розгортання 28
3 Реалізація та тестування 31
3.1 Керівництво користувача 31
3.1.1 Підготовка до роботи..... 31
3.1.2 Реєстрація та вхід до системи..... 32
3.1.3 Перегляд навчальних статей..... 37
3.1.4 Взаємодія з інтерактивними уроками..... 38
3.1.5 Взаємодія з компілятором виразів clojure 40
3.2 Unit-тестування 41
Висновки 45
Перелік посилань..... 46

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та цифрової трансформації всіх сфер суспільного життя особливої ваги набуває якість та доступність освітніх ресурсів. Освітній процес дедалі більше інтегрується з IT-рішеннями, що дозволяють забезпечити високий рівень взаємодії між студентом і навчальним матеріалом. У зв’язку з цим зростає попит на інтерактивні, адаптивні та ефективні навчальні платформи, здатні забезпечити індивідуалізацію навчання, гнучкий доступ до інформації, а також підтримку різноманітних освітніх підходів.

Традиційні форми подання навчального матеріалу – такі як друковані посібники або статичні вебресурси часто не задовольняють вимоги сучасних користувачів. Вони поступаються місцем динамічним технологічним рішенням, що дозволяють реалізовувати інтерактивність, зворотній зв’язок і високий рівень залучення користувачів. Саме тому створення нових освітніх платформ із залученням сучасних мов програмування та архітектур є одним із пріоритетних напрямів у сфері освітніх технологій.

Метою даної роботи є створення онлайн-платформи у вигляді вебзастосунку, за допомогою якої користувач зможе ознайомитися з основами функціонального програмування, а також опанувати одну з найпотужніших мов у цій галузі – Clojure. Clojure є діалектом Lisp, що поєднує концепції функціонального програмування із можливостями платформи Java, що робить її перспективною для створення надійних, масштабованих і підтримуваних програмних рішень.

Задля досягнення поставленої мети було поставлено низку завдань:

- провести аналіз сучасних навчальних платформ та схожих програмних рішень;
- вивчити літературні джерела й документацію до обраних інструментів та фреймворків;

- спроєктувати архітектуру вебзастосунку та його основні функціональні компоненти;
- реалізувати функціонал клієнтської та серверної частин;
- провести тестування застосунку з метою перевірки його стабільності та відповідності вимогам.

Практична значущість даного проєкту полягає в тому, що запропоноване рішення може бути використане як допоміжний інструмент для самостійного вивчення функціонального програмування, а також як демонстраційна платформа для ознайомлення з основами мови Clojure. Крім того, архітектура розробленого застосунку може слугувати прикладом побудови сучасного вебпрограмного забезпечення із застосуванням функціональних парадигм.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд аналогічних рішень

Світ функціонального програмування, хоча і має у своєму арсеналі велику кількість потужних інструментів, усе ж таки поступається за популярністю та кількістю готових рішень традиційним імперативним мовам програмування, таким як Java, Python чи JavaScript. Це створює певні труднощі для початківців, які прагнуть опанувати нові технології на кшталт Clojure, оскільки доступ до навчальних матеріалів, інтерактивних середовищ та практичних інструментів є обмеженим.

Одним з найвідоміших застосунків для вивчення мови Clojure є проєкт 4ever-clojure [1]. Цей вебзастосунок призначений для інтерактивного вивчення функцій базової бібліотеки clojure.core. Користувачам пропонується вирішувати задачі, реалізуючи конкретні функції, після чого система автоматично перевіряє правильність розв’язків. Онлайн-компілятор дозволяє в реальному часі писати та тестувати код, що сприяє глибшому засвоєнню синтаксису та парадигм функціонального програмування.

Ще одним корисним ресурсом є ClojureDocs [2] – спільнотна документація з відкритим доступом. Вона є неофіційним довідником по стандартній бібліотеці мови, де кожна функція супроводжується прикладами використання, коментарями та варіантами реалізацій, наданими самими користувачами. Це зручний інструмент як для новачків, так і для досвідчених розробників, який дозволяє швидко знаходити необхідну інформацію без заглиблення у повну технічну документацію.

Крім того, варто згадати Try Clojure [3] – мікросервіс, що надає можливість експериментувати з базовими виразами Clojure прямо у веббраузері. Хоча він обмежений лише простими прикладами, його роль як вступного інструменту для новачків є важливою.

Окремо слід звернути увагу на платформу Exercism [4], яка, хоча й не спеціалізується виключно на Clojure, включає її у свій список підтримуваних мов. Exercism пропонує серію завдань, які перевіряються автоматично, а також систему наставництва, де ментори можуть коментувати рішення студентів.

1.2 Опис програмних засобів

У процесі створення навчальної платформи будуть використані сучасні програмні засоби, що відповідають вимогам до функціональності, гнучкості та підтримки функціонального програмування. Основу серверної частини складатиме мова Clojure разом із фреймворком Luminus, який забезпечує ефективну організацію структури застосунку. Для реалізації клієнтської частини буде застосовано ClojureScript, а також бібліотеки Reagent і Re-Frame для побудови реактивного інтерфейсу. Комбінація цих інструментів дозволить реалізувати продуктивну та зручну у використанні платформу.

1.2.1 Огляд фреймворку Luminus

Одним із найпопулярніших фреймворків для розробки вебзастосунків на мові програмування Clojure є фреймворк Luminus [5]. Luminus є високорівневим каркасом, що базується на бібліотеках екосистеми Clojure і орієнтований на швидку розробку сучасних вебзастосунків з чистою архітектурою, підтримкою REST API, інтеграцією з базами даних та засобами безпеки.

Luminus не є окремим вебфреймворком у класичному розумінні (як, наприклад, Django чи Ruby on Rails), а радше узгодженим набором перевірених бібліотек і шаблонів конфігурацій, що допомагають розробнику ефективно організовувати проєкт.

Серед основних переваг Luminus можна виділити такі як:

– забезпечення структурування проєкту відповідно до найкращих практик функціонального програмування;
– використання перевірених бібліотек екосистеми Clojure, таких як Compojure/Reitit (маршрутизація), Ring (обробка HTTP-запитів), Reagent (інтерфейс користувача), Next.jdbc/HugSQL (робота з базами даних), Buddy (безпека);
– підтримка реактивної побудови клієнтської частини за допомогою ClojureScript і Re-Frame;
– розширювана модульна структура, що дозволяє адаптувати проєкт до конкретних вимог;
– стартовий шаблон із готовими рішеннями для логування, конфігурації, управління сесіями та REST API.

Однією з ключових переваг Luminus є його розширюваність і модульність: розробник може вибирати та налаштовувати необхідні компоненти відповідно до вимог конкретного проєкту. Важливо й те, що Luminus підтримує інтеграцію з frontend-технологіями, такими як ClojureScript та бібліотекою Reagent, що забезпечує реактивний підхід до побудови інтерфейсу користувача.

Крім того, стартовий шаблон Luminus містить усе необхідне для швидкого початку роботи: структуру каталогів, систему конфігурації, логування, управління сесіями, безпекові посередники (middleware), а також базову реалізацію REST-інтерфейсу.

У межах розробки навчальної платформи Luminus був обраний як основа серверної частини проєкту завдяки своїй гнучкості, активній спільноті, підтримці сучасних практик функціонального програмування та можливості легко масштабувати рішення в майбутньому.

Кожен додаток, створений за допомогою фреймворку Luminus, поділений на окремі модулі, кожен з яких відповідає за конкретний аспект роботи застосунку. Такий підхід дозволяє забезпечити розділення

відповідальностей, гнучкість у розробці та зручність у супроводі програмного коду. Розглянемо основні модулі, з яких складається типовий застосунок з використанням Luminus.

Вебсервер відповідає за запуск застосунку та обробку вхідних HTTP-запитів. Luminus зазвичай використовує http-kit або jetty як сервер. Маршрутизація (routing) реалізується за допомогою бібліотеки reitit або compojure, які дозволяють описувати шляхи обробки HTTP-запитів (маршрути), їхні методи (GET, POST тощо) та відповідні функції-обробники.

База даних (database) підключається через бібліотеку hikari-cp, а запити виконуються через бібліотеки next.jdbc або conman (з підтримкою SQL-файлів). Модуль запитів (queries) представлений окремими просторами імен (namespace), що містять SQL-запити до бази даних, які завантажуються з окремих .sql файлів. Це забезпечує чистоту коду та гнучкість при зміні запитів.

Шаблони представлення (templates) відповідають за відображення HTML-сторінок, і Luminus підтримує шаблонізатори Selmer, Hiccup або інші засоби виведення (rendering). Посередники (middleware) – це набір функцій для обробки запитів і відповідей між клієнтом і сервером, наприклад: логування, обробка помилок, захист від CSRF, CORS, стиснення, авторизація тощо.

Налаштування (configuration) здійснюється через конфігураційні файли (config.edn, .env) та бібліотеку cprop для зчитування параметрів залежно від середовища (розробка, продакшн тощо). Обробники (handlers/controllers) – це функції, які реалізують бізнес-логіку обробки запитів, таких як створення користувача, авторизація, валідація форм тощо.

Компонентна система (component) використовується для структурування та управління життєвим циклом додатку. Кожен модуль (сервер, база даних, маршрути тощо) описується як компонент і ініціалізується в єдину систему за допомогою бібліотеки mount або integrant. Клієнтська частина (frontend), якщо проєкт є SPA (Single Page Application), включає інтеграцію з клієнтськими фреймворками: ClojureScript (reagent, shadow-cljs) або іншими мовами через API.

Модуль безпеки (security) охоплює автентифікацію, авторизацію користувачів, шифрування паролів (наприклад, з використанням buddy або bcrypt), захист від атак (CSRF, XSS тощо). Тестування (testing) у Luminus підтримується структурою для юніт- та інтеграційного тестування з використанням бібліотек clojure.test, expectations, kaocha тощо [6].

1.2.2 Використання ClojureScript у фронтенд-розробці

Однією з важливих особливостей архітектури розроблюваної платформи є застосування мови ClojureScript для створення клієнтської частини вебзастосунку. ClojureScript є діалектом мови Clojure, що компілюється у JavaScript та дозволяє будувати сучасні, інтерактивні та масштабовані вебінтерфейси з використанням парадигми функціонального програмування.

Розглянемо цю мову програмування більш детально. ClojureScript є функціональною мовою програмування і логічним продовженням самої Clojure. Оскільки ми компілюємо код у JavaScript, ми отримуємо доступ до функціоналу браузера і можемо так само ефективно писати логіку для застосунку. Компілятором ClojureScript виступає Google Closure. Це програмне рішення від компанії Google було створено з ціллю оптимізації вебзастосунків і підвищення швидкості їх компіляції та роботи.

Компілятор Closure Compiler трансформує JavaScript-код у компактний та високопродуктивний формат. Він автоматично виконує видалення «мертвого» коду (тобто такого, що не використовується), оптимізує та мінімізує решту, що сприяє швидшому завантаженню та виконанню вебзастосунків. Окрім цього, компілятор здійснює перевірку синтаксису, коректності посилань на змінні, а також типів даних, своєчасно попереджаючи про поширені помилки, характерні для мови JavaScript. Такі механізми аналізу та оптимізації значно підвищують надійність програмного коду, зменшують кількість помилок у розробці та полегшують подальший супровід і масштабування програмних рішень [7].

Проте висока продуктивність і швидкість роботи є хоча і значним плюсом роботи з цією мовою, проте не є головною мотивацією її використання. Для професійних розробників однією з головних переваг є поширення коду між сервером і клієнтом. Оскільки ми використовуємо споріднені мови програмування для бек- і фронт-енду застосунку, ми маємо доступ до одних і тих самих типів даних, як, наприклад, хеш-мапи, вектори, атоми та багатьох інших. Розробник не має вручну серіалізувати запити і відповіді, а може одразу працювати з ними як зі звичайними даними мови Clojure. Це дуже корисно при втіленні таких завдань, як аутентифікація користувачів, авторизація, сесія та обмін статичними даними між клієнтом і сервером [8].

Проте, як і будь-який додаток на JavaScript не пишеться в сучасному світі без використання великих фреймворків, як, наприклад, React, Angular або Redux – так і програми на ClojureScript потребують сторонніх біліотек для розв’язання комплексних завдань. Одними із найпопулярніших інструментів для функціональної веброзробки є бібліотеки Reagent та Re-Frame. Розглянемо їх більш детально у наступному розділі.

1.2.3 Використання бібліотек Reagent та Re-Frame

Для ефективної реалізації клієнтської частини застосунків на мові ClojureScript розробники, як правило, вдаються до використання сторонніх бібліотек, які забезпечують зручне управління інтерфейсом користувача та станом програми. Найпоширенішими серед таких інструментів є бібліотеки Reagent та Re-Frame, які значною мірою визначають архітектурний стиль сучасних ClojureScript-додатків.

Reagent є обгорткою над React.js, що дозволяє писати компоненти інтерфейсу користувача у вигляді чистих функцій на ClojureScript з використанням гнучкої та зручної синтаксичної форми – Hiccup. Завдяки

цьому, розробка UI-компонентів стає прозорою, декларативною та легкою для тестування. Роглянемо типовй приклад такого компоненту. Для створення UI-компоненту потрібно реалізувати функцію, що повертатиме вектор зі структурою нашого компоненту. Слід зазначити, що перетворення вектору на чистий HTML відбувається за допомогою бібліотеки Hiccup, що являє собою DSL (мова спеціалізована для певної предметної області). На рисунку 1.1 представлений такий компонент.

```
(ns components.server-response
  (:require [re-frame.core :as rf]))

(defn server-response-component []
  (if-let [server-errors @(rf/subscribe [:server-errors])]
    [:div.notification.is-danger.my-2
     [:p "Server errors:"]
     (for [err (vals server-errors)]
       ^{:key err} [:p err])]
    (if-let [server-success @(rf/subscribe [:server-success])]
      [:div.notification.is-success.my-2
       [:p "Message from the server:"]
       (for [suc (vals server-success)]
         ^{:key suc} [:p suc])
       [:a {:href "/" :target "_blank"} "Go to the home page ->"]])))
```

Рисунок 1.1 – Приклад UI-компонента з використанням Reagent

Re-Frame, у свою чергу, базується на бібліотеці Reagent і надає більш високорівневу архітектуру, що орієнтована на шаблон event-driven programming. Ця бібліотека впроваджує чіткий поділ на події, ефекти, підписки та подання, що значно спрощує управління станом застосунку, забезпечує його передбачуваність та масштабованість. Re-Frame сприяє кращій організації коду, полегшує його розширення та тестування, що є особливо важливим у проєктах зі зростаючою складністю.

Таким чином, використання Reagent та Re-Frame є рекомендованою практикою при створенні продуктивних, модульних та підтримуваних фронтенд-застосунків на ClojureScript.

2 ПРОЄКТУВАННЯ НАВЧАЛЬНОЇ ПЛАТФОРМИ

2.1 Аналіз вимог

Першим кроком у проєктуванні застосунку є визначення його основних функціональних та нефункціональних вимог. Від правильного формулювання цих вимог залежить не лише відповідність розробленого продукту очікуванням користувачів, а й ефективність його роботи, можливість масштабування та подальшого супроводу.

Функціональні вимоги описують, що саме повинна робити система, тобто її основну поведінку. У випадку розробки навчальної платформи на базі Clojure до функціональних вимог належать:

- можливість реєстрації нових користувачів;
- автентифікація користувачів (вхід у систему);
- відображення навчальних матеріалів у структурованому вигляді;
- виконання інтерактивних завдань;
- автоматична перевірка відповідей;
- взаємодія з навчальним контентом у реальному часі (без перезавантаження сторінки).

Нефункціональні вимоги визначають якісні характеристики системи, що впливають на її продуктивність, стабільність та зручність користування. Для розроблюваної платформи вони включають:

- високу швидкість завантаження (не більше 2 секунд при нормальному навантаженні);
- сумісність з актуальними версіями сучасних веббраузерів;
- модульну структуру коду з чітким розділенням на завдання клієнта та серверу;
- компонентну систему елементів користувацького інтерфейсу, що можуть бути модифіковані незалежно один від одного;
- можливість масштабування системи без значних змін її арзітектури.

Вказані вимоги слугували основою для формування технічного завдання, у якому деталізовано ключові модулі, способи взаємодії між компонентами та обрані інструменти розробки. У наступних підрозділах ці аспекти будуть розглянуті детальніше.

2.2 Використання UML під час розробки системи

UML (Unified Modeling Language, уніфікована мова моделювання) є стандартною мовою для візуалізації, специфікації, конструювання та документування програмних систем. Вона широко застосовується в інженерії програмного забезпечення для опису структури та поведінки системи в уніфікованій графічній формі. UML не залежить від конкретної мови програмування чи технологічної платформи, що робить її універсальним інструментом для планування та документування архітектури програмного забезпечення.

У процесі розробки навіть невеликого застосунку створення моделей UML допомагає краще осмислити внутрішню структуру системи, встановити логічні зв’язки між її компонентами та спроєктувати основні сценарії взаємодії. UML забезпечує чіткість у формулюванні вимог, дозволяє на ранньому етапі виявити суперечності або логічні помилки в проєкті та спрощує перехід від технічного задуму до безпосередньої реалізації.

UML включає набір діаграм, які поділяються на структурні (наприклад, діаграми класів, компонентів, розгортання) та поведінкові (діаграми прецедентів, послідовності, активностей). Кожна з них використовується для моделювання різних аспектів системи: від загального уявлення про функціональні можливості до деталей реалізації бізнес-логіки.

У межах даної роботи UML застосовувалась як засіб формалізації проєктних рішень. Було побудовано діаграми прецедентів і послідовності, що описують ключові сценарії взаємодії користувача із системою, а також

реалізовано структурні діаграми, які демонструють логіку організації коду та бази даних. Завдяки цьому вдалося забезпечити більш послідовне та обґрунтоване проєктування програмної архітектури.

2.3 Концептуальне проєктування

Діаграма варіантів використання (діаграма прецедентів, Use Case Diagram) – це один із базових типів діаграм у UML, який дозволяє описати функціональні вимоги до системи у формі взаємодії зовнішніх акторів із її можливостями. Вона демонструє, хто саме (актор) і як саме (прецедент) взаємодіє з системою. Такий тип моделі особливо зручний для початкового етапу проєктування, коли важливо зрозуміти загальну структуру функціональності, не заглиблюючись у технічні деталі реалізації.

Діаграма прецедентів допомагає виявити основні сценарії використання системи, визначити ролі користувачів та межі відповідальності системи. Вона є корисною як для постановки завдання, так і для перевірки повноти реалізованого функціоналу на фінальних етапах.

У межах розробки навчальної платформи така діаграма дала змогу чітко сформулювати перелік ключових дій, які має підтримувати застосунок – зокрема, реєстрацію, авторизацію, перегляд контенту, виконання завдань тощо.

На рисунку 2.1 представлена діаграма варіантів використання. На діаграмі представлений актор «Користувач», що має доступ до наступних прецедентів у застосунку: реєстрація, авторизація, перегляд інтерактивних уроків, взаємодія з онлайн-компілятором, виконання завдань уроків та перевірки їх результатів. На кожну з цих дій передбачено відповідне оновлення користувацького інтерфейсу, запити до сервера та бази даних, а також відповіді сервера і обробку помилок. Розглянемо кожний прецедент детальніше.



Рисунок 2.1 – Діаграма варіантів використання

Прецедент «Реєстрація».

Призначення: даний варіант використання забезпечує реєстрацію нового користувача у системі зі збереженням його до бази даних.

Основний потік подій: даний варіант використання починає виконуватись, коли користувач натискає кнопку “Sign up”, заповнює дані для реєстрації та натискає кнопку “Submit”.

Альтернативний потік подій: даний варіант використання не є доступним для авторизованих користувачів, замість кнопки реєстрації користувачу повертається привітання з його іменем.

Передумова: система не має авторизованого користувача на момент реєстрації.

Виняткова ситуація: користувач вводить дані, що не проходять перевірку на клієнті та сервері – система відобразить відповідне повідомлення, користувач може змінити дані у формі і спробувати ще раз.

Прецедент «Авторизація».

Призначення: даний варіант використання дає можливість користувачу авторизуватися в систему, використовуючи ім’я користувача і пароль, що порівнюється з записами у базі даних.

Основний потік подій: даний варіант використання починає виконуватись, коли користувач натискає на кнопку “Log in” та вводить необхідні дані у форму, після успішної авторизації користуавача переносить на домашню сторінку.

Альтернативний потік подій: якщо введеного імені користувача не існує в базі даних або введено неправильний пароль – замість авторизації користувач отримує повідомлення про помилку.

Передумова: система не має авторизованого користувача.

Виняткова ситуація: користувач вводить дані, що не проходять перевірку на клієнті та сервері – система відобразить відповідне повідомлення, користувач може змінити дані у формі і спробувати ще раз.

Прецедент «Перегляд навчальних статей».

Призначення: даний варіант використання дає можливість користувачу переглядати список усіх існуючих статей у базі даних.

Основний потік подій: даний варіант використання починає виконуватись, коли користувач натискає на кнопку “Articles”. Після натискання користувачу відображається компонент графічного інтерфейсу, що містить перелік статей.

Альтернативний потік подій: якщо статті перебувають у стані завантаження, користувачу відображається компонент із текстом “Articles are being loaded”; у разі помилки завантаження статей користувач бачить відповідне повідомлення про неможливість завантаження вмісту.

Передумова: користувач натиснув кнопку “Articles” та перейшов на відповідну сторінку; наявне підключення до бази даних.

Прецедент «Перегляд інтерактивних уроків».

Призначення: даний варіант використання дає можливість користувачу переглядати список усіх наявних інтерактивних уроків.

Основний потік подій: даний варіант використання починає виконуватись, коли користувач натискає на кнопку “Lessons”. Після натискання відображається компонент графічного інтерфейсу, що містить перелік уроків.

Альтернативний потік подій: якщо уроки перебувають у стані завантаження, користувачу відображається компонент із текстом “Lessons are being loaded”; у разі помилки завантаження статей користувач бачить відповідне повідомлення про неможливість завантаження вмісту.

Передумова: користувач натиснув кнопку “Lessons” та перейшов на відповідну сторінку; наявне підключення до бази даних.

Прецедент «Взаємодія з онлайн-компілятором».

Призначення: даний варіант використання предбачає взаємодію користувача з онлайн-компілятором мови Clojure.

Основний потік подій: даний варіант використання починає виконуватись, коли користувач натискає на кнопку “Compiler”. Після натискання користувачу відображається компонент графічного інтерфейсу, що містить поле для вводу виразів, які мають бути скомпільовані, а також повідомлення про результат їх компіляції.

Альтернативний потік подій: якщо компілятор перебуває у стані завантаження користувачу відображається відповідне повідомлення.

Передумова: користувач натиснув кнопку “Compiler” та був направлений до відповідної сторінки.

Прецедент «Виконання завдань».

Призначення: даний варіант використання призначений для виконання користувачем інтерактивних уроків у форматі питань із множинним вибором.

Основний потік подій: даний варіант використання починає виконуватись, коли користувач натискає на посилання із назвою уроку при перегляді списку уроків.

Альтернативний потік подій: якщо урок перебуває у стані завантаження користувачу відображається відповідне повідомлення.

Передумова: користувач натиснув на кнопку “Lessons” та перебуває на відповідні сторінці зі списком уроків.

Прецедент «Перевірка результатів».

Призначення: даний варіант використання призначений для перевірки користувачем правильності відповідей на запитання інтерактивного уроку.

Основний потік подій: даний варіант використання починає виконуватись, коли користувач натискає на кнопку “Show results”.

Альтернативний потік подій: у разі помилки обробки результатів користувачу відображається відповідне повідомлення.

Передумова: користувач натиснув на кнопку “Show results”, на кожне завдання уроку вибраний варіант відповіді.

Діаграма послідовності використовується на етапі проєктування для формалізації логіки взаємодії між основними компонентами системи у хронологічному порядку. У контексті навчальної платформи побудована діаграма дозволяє описати типову послідовність подій, що відбуваються під час взаємодії користувача із застосунком на рівні автентифікації та авторизації. Побудована діаграма представлена на рисунку 2.2.

Основними учасниками діаграми виступають: користувач, клієнтська частина застосунку (браузер), серверна частина, а також система зберігання користувацьких даних (база даних). Діаграма охоплює чотири ключові сценарії: реєстрація нового користувача, вхід до системи, звернення до захищених ресурсів та вихід із облікового запису.

У процесі реєстрації користувач заповнює форму із необхідними персональними даними. Після підтвердження введеної інформації клієнтська частина застосунку надсилає запит на сервер. Сервер перевіряє, чи є дані задовільними і чи не існує в системі облікових записів із вказаним ім’ям або електронною адресою. У разі успішної перевірки відповідна інформація зберігається у базі даних, після чого користувач отримує підтвердження про завершення реєстрації.

Під час автентифікації користувач передає облікові дані (логін та пароль) за допомогою форми. Серверна частина застосунку, отримавши запит,

здійснює перевірку достовірності введених даних шляхом порівняння з наявною інформацією у базі. Якщо перевірка проходить успішно, створюється сесія або токен, що підтверджує особу користувача та дозволяє отримувати доступ до функціоналу, передбаченого для авторизованих осіб.

Усі подальші звернення до захищених маршрутів супроводжуються перевіркою наявності чинної сесії або токена. У разі, якщо користувач є авторизованим, сервер дозволяє виконання запиту. В іншому випадку повертається відповідь про відсутність доступу. Завдяки цьому реалізується контроль за рівнем доступу до внутрішніх ресурсів системи.

Процес завершення сеансу роботи передбачає надсилання запиту на вихід. У відповідь сервер анулює сесію або токен, після чого користувач втрачає можливість взаємодії із захищеним функціоналом без повторної автентифікації.

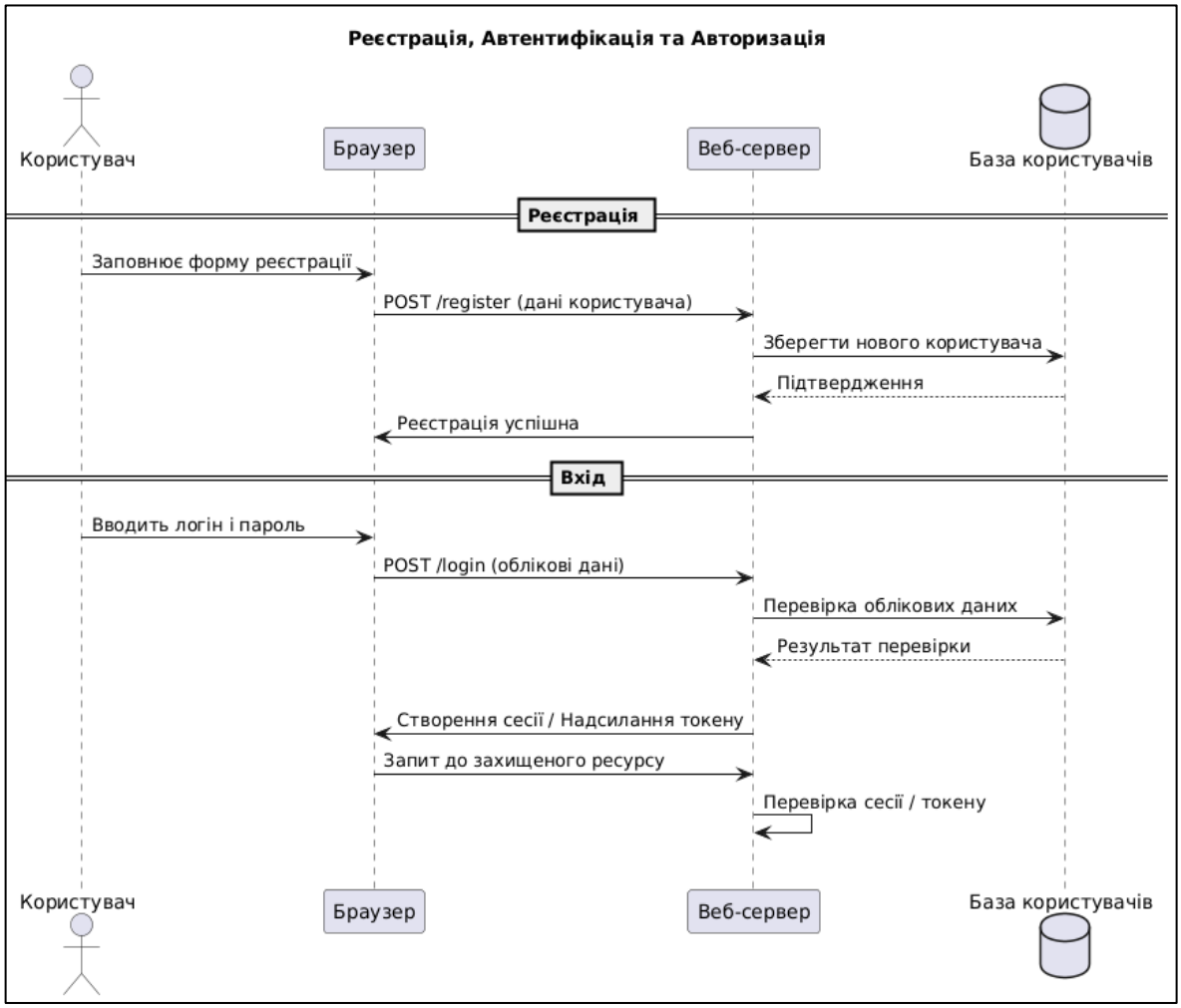


Рисунок 2.2 – Діаграма послідовності

Таким чином, побудована діаграма послідовності дозволяє чітко зафіксувати порядок дій між окремими компонентами системи, встановити логіку доступу до ресурсів, а також сформувати базу для подальшої деталізації механізмів безпеки, обробки помилок та збереження стану користувача в рамках вебзастосунку.

2.4 Логічна структура системи

На етапі логічного проєктування особливу увагу приділено побудові діаграми класів, яка дозволяє формалізувати структуру програмної системи до її безпосередньої реалізації. Дана діаграма відображає основні логічні сутності навчальної платформи, включаючи користувачів, уроки, статті, запитання до уроків та службові компоненти обробки запитів. Вона демонструє зв’язки між класами, їх атрибути та основні методи, що забезпечують взаємодію всередині системи.

Використання діаграми класів дозволяє краще уявити логічну організацію даних і залежностей між компонентами, спростити процес реалізації архітектури, забезпечити підтримку принципів модульності, повторного використання та розширюваності. Представлена на рисунку 2.3 модель стала базою для реалізації маршрутизації, взаємодії з базою даних та обробки HTTP-запитів у відповідних модулях системи.

Розглянемо діаграму класів більш детально. Спершу опишемо серверну логіку застосунка, представлену на діаграмі.

UserHandler, **LessonHandler**, **ArticleHandler** – HTTP-контролери, що обробляють відповідні маршрути /api/login, /api/lessons, /api/articles тощо. Вони викликають відповідні сервіси (AuthService, LessonService, ArticleService).

LessonService та **ArticleService** – логіка бізнес-рівня, що отримує або змінює навчальний контент у базі даних.

Middleware – обробники запитів (наприклад, JSON-десереалізація, автентифікація, обробка параметрів із запиту), які застосовуються до всіх маршрутів через систему Ring.

Routes – конфігурація маршрутів, яка об’єднує всі обробники запитів в єдину структуру URL-адрес.

Database – абстракція над доступом до бази даних, що дозволяє виконувати SQL-запити з параметрами.

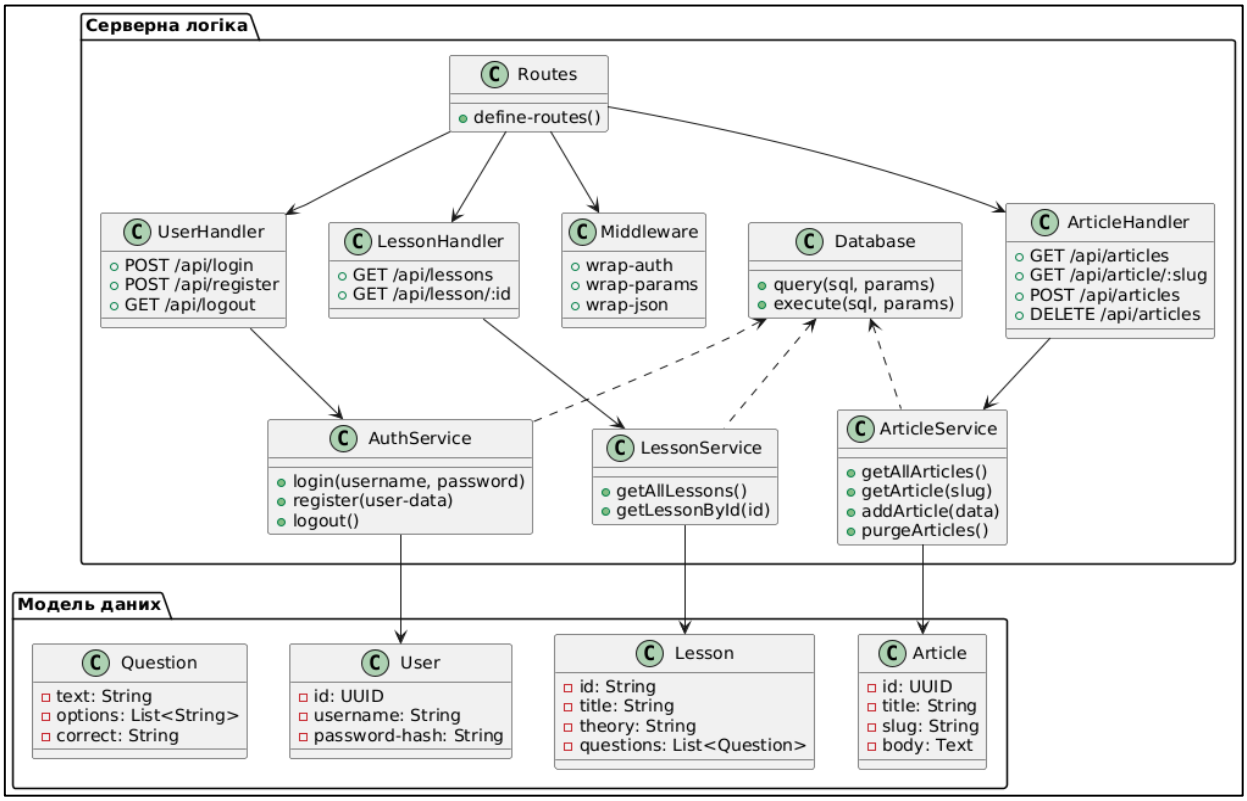


Рисунок 2.3 – Діаграма класів

Проведемо опис моделі даних.

User – представляє користувача системи. Містить унікальний id, ім’я користувача та хеш паролю. Його створення та автентифікація обробляються через AuthService.

Article – сутність, що відображає статтю. Має поля id, title, slug (унікальний ідентифікатор у URL), body. Обробляється через ArticleService.

Lesson – описує навчальний урок із заголовком, теоретичною частиною та набором запитань.

Question – зв’язана з уроками структура, яка містить текст запитання, список варіантів відповідей та правильну відповідь.

2.5 Фізична структура системи

2.5.1 Діаграма компонентів

Діаграма компонентів слугує засобом представлення архітектурної структури програмної системи на вищому рівні абстракції. Вона дозволяє візуалізувати взаємозв’язки між ключовими модулями застосунку, які відповідають за різні аспекти його функціонування – від обробки HTTP-запитів до логіки авторизації, навчального контенту, доступу до даних та рендерингу інтерфейсу користувача.

У розроблюваній навчальній платформі застосовано клієнт-серверну архітектуру з чітким розділенням відповідальностей між фронтендом і бекендом (див. рис. 2.4).

Фронтенд реалізовано як односторінковий застосунок (SPA) з використанням бібліотеки Reagent (обгортки над React для ClojureScript). Клієнтська частина взаємодіє з сервером через REST API, який реалізовано у вигляді набору маршрутів, побудованих за допомогою маршрутизатора Ring у зв’язці з бібліотекою Reitit.

На серверному боці система структурована за принципами багаторівневої архітектури. Вебрівень обробляє HTTP-запити та передає їх до відповідних сервісів, які реалізують бізнес-логіку (реєстрація, обробка вправ, перевірка відповідей тощо).

Окремі компоненти відповідають за взаємодію з базою даних через пул з’єднань та централізований репозиторій.

Інфраструктурні модулі реалізують сесійну підтримку, логування, безпеку та зчитування конфігурацій.

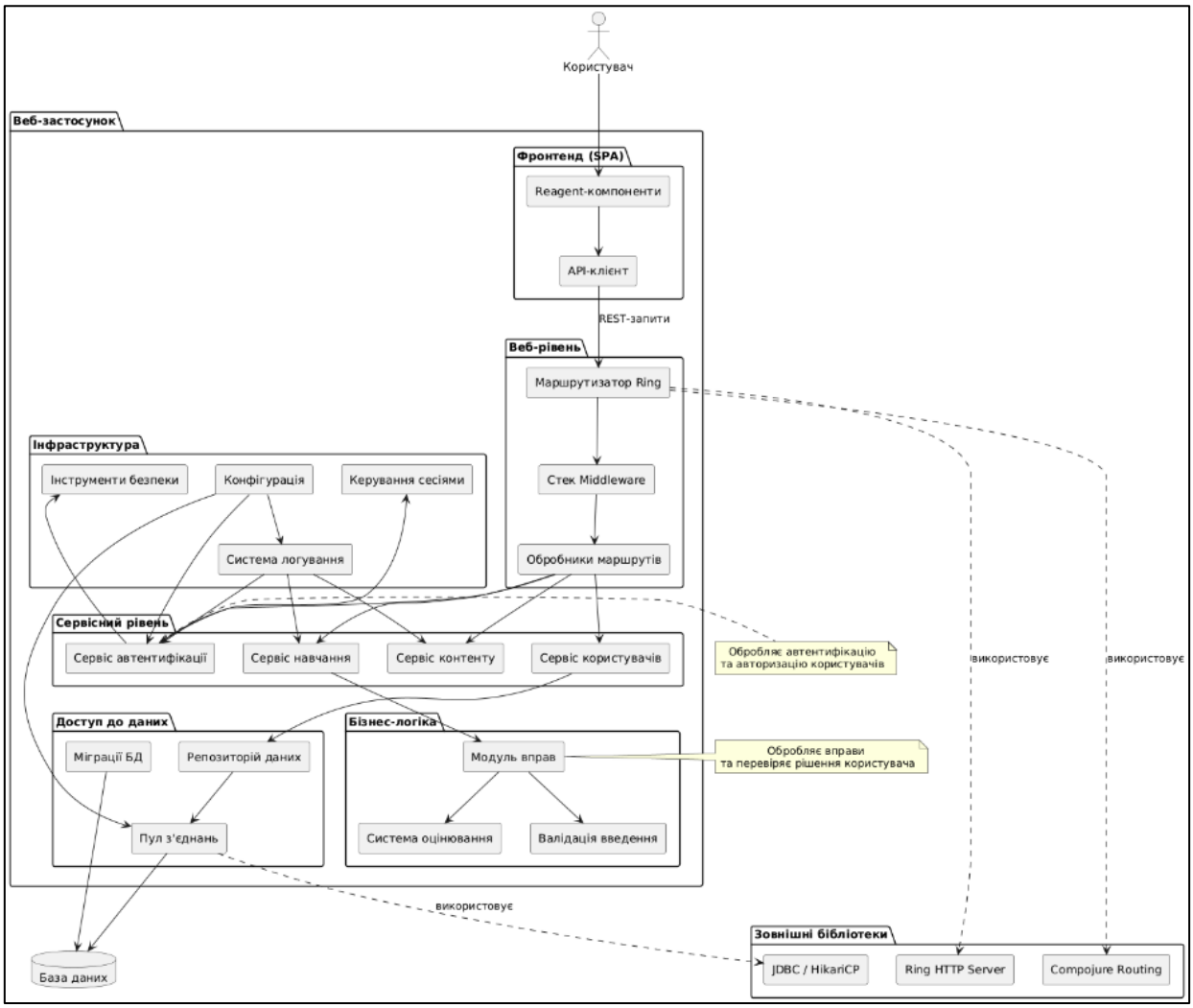


Рисунок 2.4 – Діаграма компонентів

Така модульна структура забезпечує масштабованість, полегшує супровід та дозволяє ізольовано працювати над окремими компонентами системи. Представлена на рисунку 2.4 діаграма компонентів відображає ключові логічні блоки системи, їх залежності та канали взаємодії.

2.5.2 Діаграма розгортання

Діаграма розгортання відображає фізичне розміщення компонентів програмного забезпечення на вузлах обчислювальної інфраструктури. Вона дозволяє побачити, як логічні частини системи розподіляються між клієнтською, серверною та інфраструктурною частинами. Така діаграма

особливо корисна для розуміння архітектури взаємодії між компонентами під час виконання, а також для планування розгортання і масштабування. Діаграма розгортання представлена на рисунку 2.5.

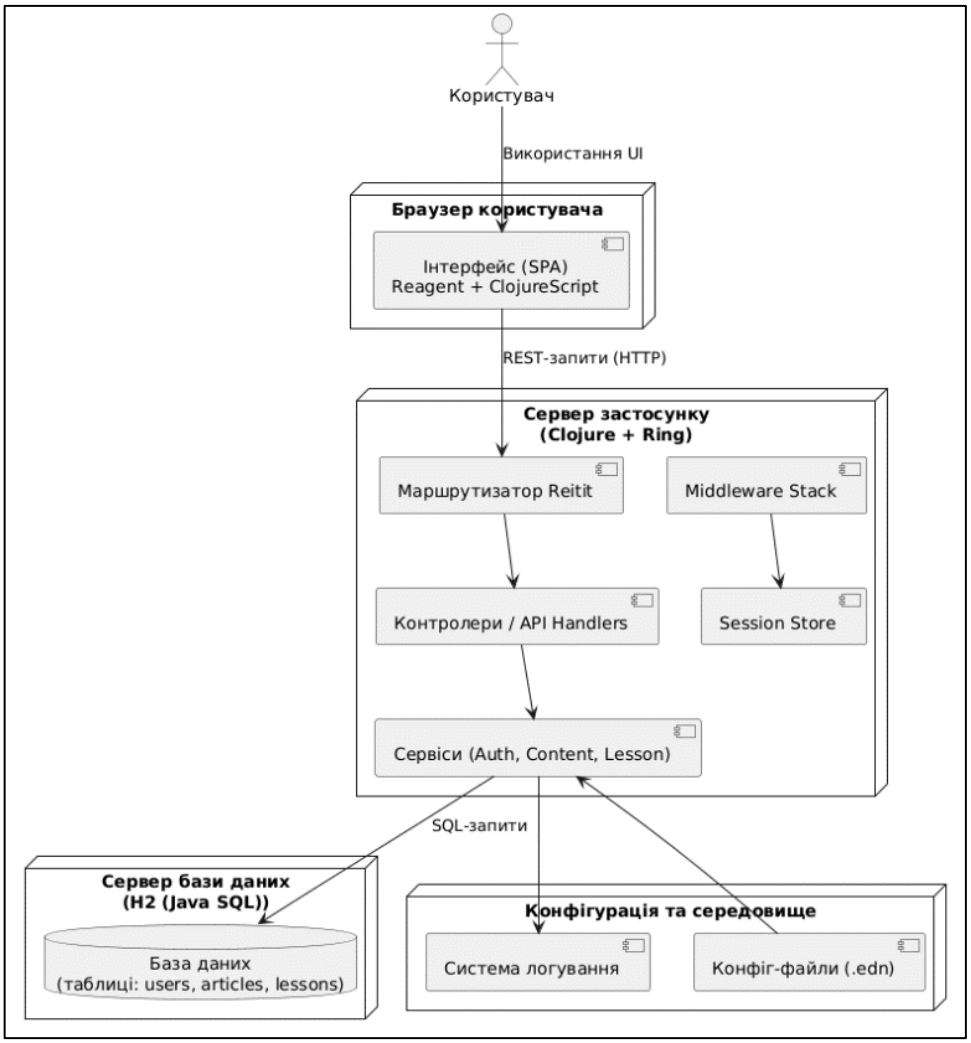


Рисунок 2.5 – Діаграма розгортання

У навчальній платформі клієнтська частина реалізується у вигляді односторінкового застосунку (SPA) з використанням Reagent – бібліотеки, що забезпечує реактивний інтерфейс користувача, компільований у JavaScript за допомогою ClojureScript. Цей інтерфейс завантажується в браузері користувача, який ініціює HTTP-запити до бекенд-серверу.

Бекенд реалізується на Clojure та використовує фреймворк Ring разом із маршрутизатором Reitit для обробки REST-запитів. Запити передаються до відповідних контролерів, які взаємодіють із сервісами, що реалізують бізнес-

логіку авторизації, управління контентом і навчальними матеріалами. Проміжне ПЗ (middleware) обробляє сесії та форматування запитів/відповідей.

Дані зберігаються у вбудованій базі даних H2, яка використовується як реляційна СУБД для зберігання таблиць users, articles та lessons. Система логування та конфігураційні файли .edn інтегруються безпосередньо з сервером і забезпечують централізовану підтримку та налаштування застосунку.

Усі компоненти взаємодіють між собою у типовому сценарії: користувач взаємодіє з SPA, яка надсилає запити до API, що обробляється сервером, результати зчитуються з бази даних і повертаються на клієнт у вигляді transit/JSON.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

У цьому розділі розглянемо програмну реалізацію спроєктованих модулів та проведемо тестування їх коректної роботи. Проаналізуємо і покажемо поетапну роботу основних функцій застосунку: реєстрація нових користувачів, їх автентифікація та авторизація; перегляд статичного контенту у вигляді навчальних статей; взаємодія з інтерактивними елементами застосунку такими як компілятор виразів Clojure та інтерактивними уроками.

3.1 Керівництво користувача

3.1.1 Підготовка до роботи

Доступ до системи здійснюється за допомогою мережі Інтернет з використанням веббраузера. Для коректної роботи всіх функцій застосунку має використовуватись сучасний браузер, як наприклад, Google Chrome, Mozilla Firefox, Brave тощо. Наповнення застосунку виконано виключно англійською мовою, що полегшує сприйняття матеріалів для найбільш широкого кола користувачів. Для повноцінного використання платформи користувач має володіти базовою англійською мовою і специфічною для комп’ютерних наук та зокрема функціонального програмування термінологією.

При вході на сайт користувачу повертається домашня сторінка застосунку, що містить основну інформацію про платформу та її функції. За замовчуванням користувач відноситься до групи анонімних користувачів, допоки не здійснить реєстрацію або вхід в систему.

Стартова сторінка складається з меню навігації, що містить основні посилання на сторінки, а також із головного блоку (hero section), що містить

заголовок та інформацію про переваги функціонального програмування та мови Clojure (див. рис. 3.1).

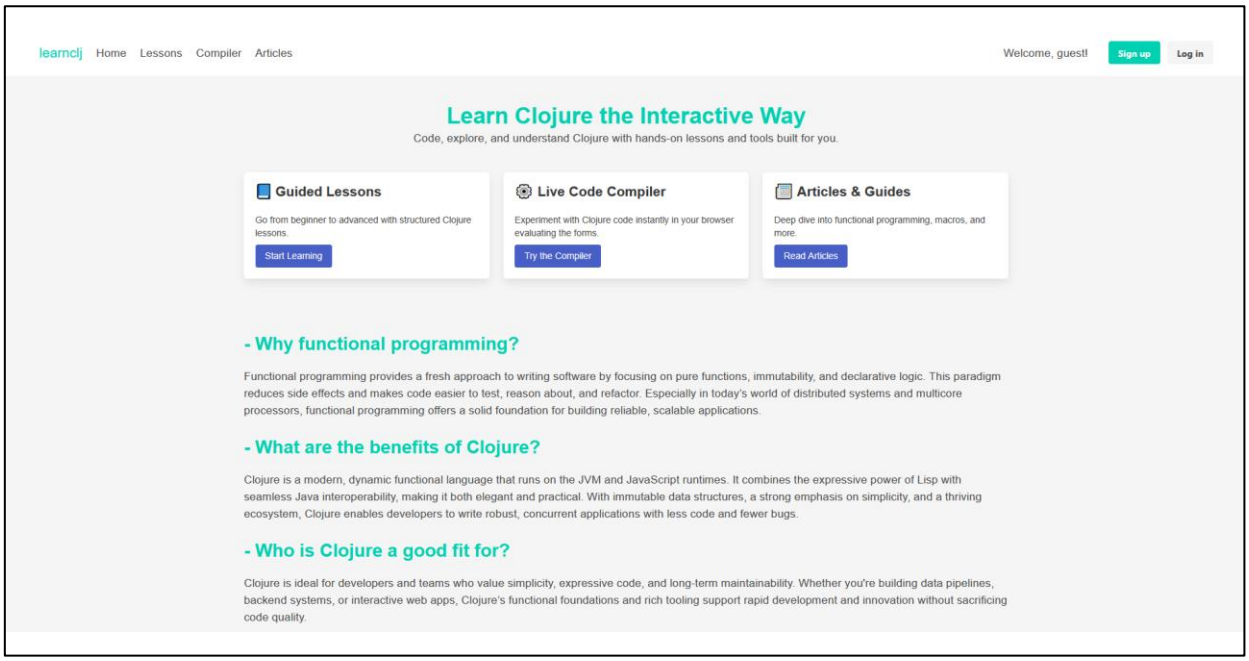


Рисунок 3.1 – Стартова сторінка застосунку

Кожна сторінка представляє собою окремий компонент користувацького інтерфейсу, що повертається користувачу залежно від URI (уніфікованого ідентифікатора ресурсу) у адресному рядку браузера. На кожен запит користувача сервер завжди повертає одну і ту саму сторінку home.html, оскільки SPA (односторінковий застосунок) передбачає логіку маршрутизації реалізовану на клієнті.

У наступних розділах будуть описані основні прецеденти програми, описаний чіткий алгоритм дій користувача, а також відображені можливі виняткові ситуації на реакція системи на них.

3.1.2 Реєстрація та вхід до системи

При вході на сайт користувачу доступні дві кнопки у меню навігації для реєстрації та входу у систему (див. рис. 3.2). За замовчуванням вони є

активними, коли в системі немає авторизованого користувача. Кнопки відображаються разом із стандартним повідомленням для гостя.

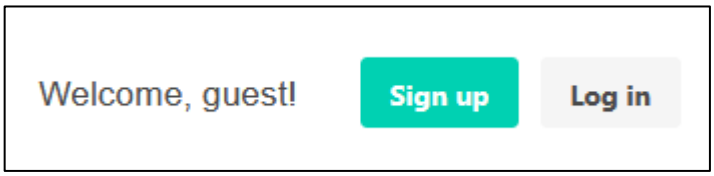


Рисунок 3.2 – Кнопки для створення облікового запису та входу в систему

Якщо користувач не має облікового запису у системі, потрібно натиснути кнопку “Sign up”, що переносить користувача на відповідну сторінку реєстрації із формою. Форма представлена на рисунку 3.3.

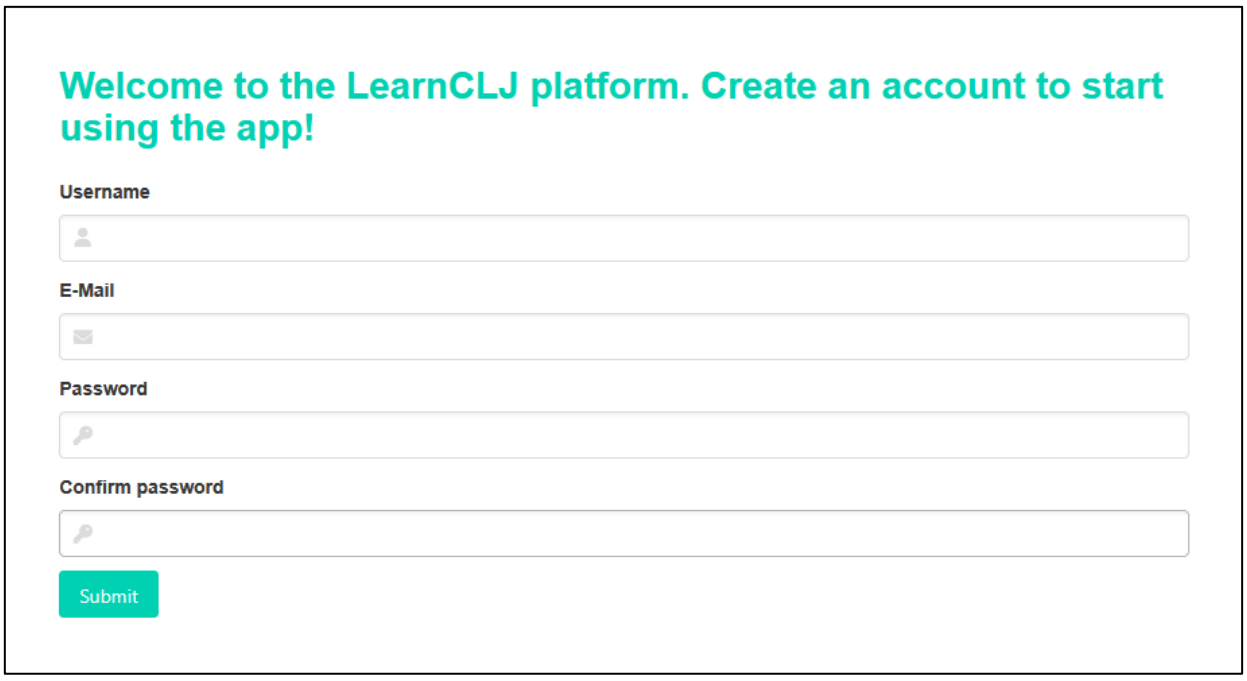


Рисунок 3.3 – Форма реєстрації користувача

Під час заповнення форми система автоматично перевіряє правильність введення даних. Для обох форм реєстрації та входу існує дві системи валідації. Перша – на стороні клієнта, вона не дозволить відправити невалідні дані на сервер, про що повідомить користувача. На рисунку 3.4 представлена відповідна ситуація.

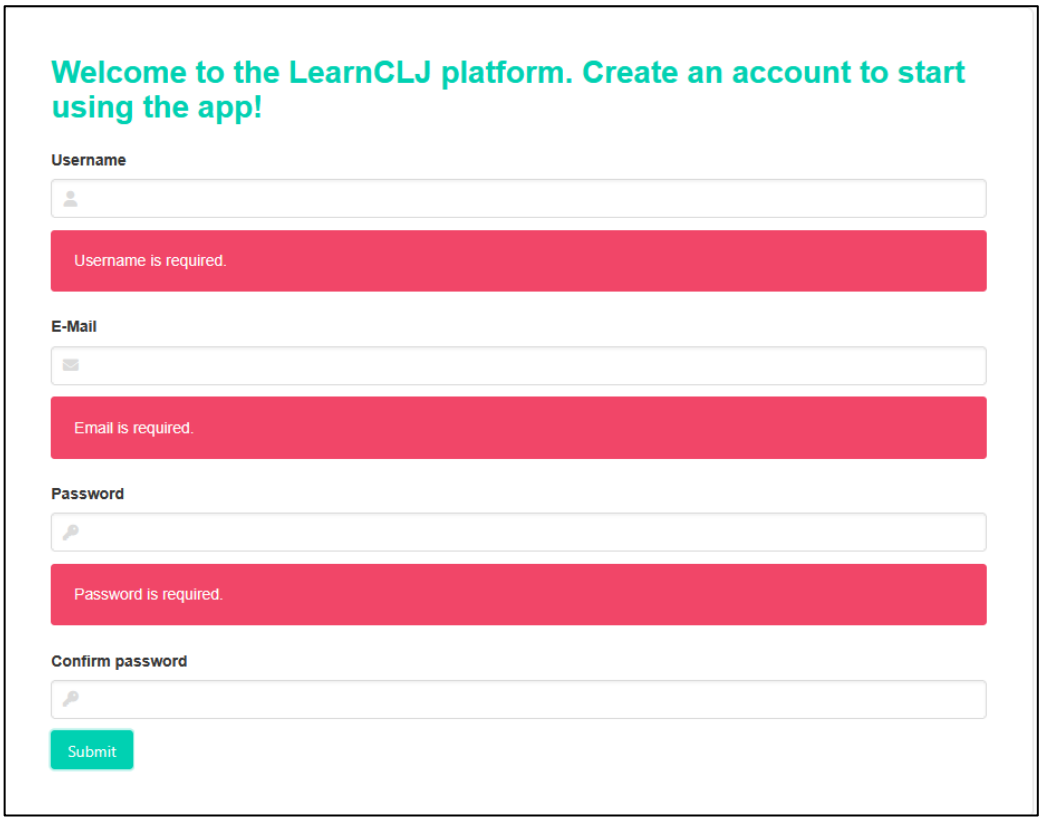


Рисунок 3.4 – Повідомлення про помилку при обробці на стороні клієнта

На рисунку 3.5 представлений відповідний фрагмент коду, що відповідає за валідацію даних форми.

```
(defn validate-fields [fields]
  (let [f @fields]
    (cond-> {}
      (str/blank? (:username f))
      (assoc :username "Username is required.")
      (str/blank? (:email f))
      (assoc :email "Email is required.")
      (and (not (str/blank? (:email f)))
           (not (re-matches #".+@.+\..+" (:email f))))
      (assoc :email "Email is invalid.")
      (str/blank? (:password f))
      (assoc :password "Password is required.")
      (and (not (str/blank? (:password f))) (> 8 (count (:password f))))
      (assoc :password "Password should be at least 8 charachters long.")
      (not= (:password f) (:confirm-password f))
      (assoc :confirm-password "Passwords do not match."))))
```

Рисунок 3.5 – Валідація даних форми реєстрації

При успішній валідації форми на стороні клієнта та відправки запиту на сервер дані мають пройти серверну валідацію. Залежно від цього сервер формує відповідь, яку бачить користувач. До типових помилок серверної валідації відносяться: ім’я користувача або електрона пошта зареєстровані у системі, дані спотворено при передачі на сервер і програма не може їх обробити тощо. На рисунку 3.6 зображена відповідь сервера під час неуспішної валідації даних після відправки запиту.

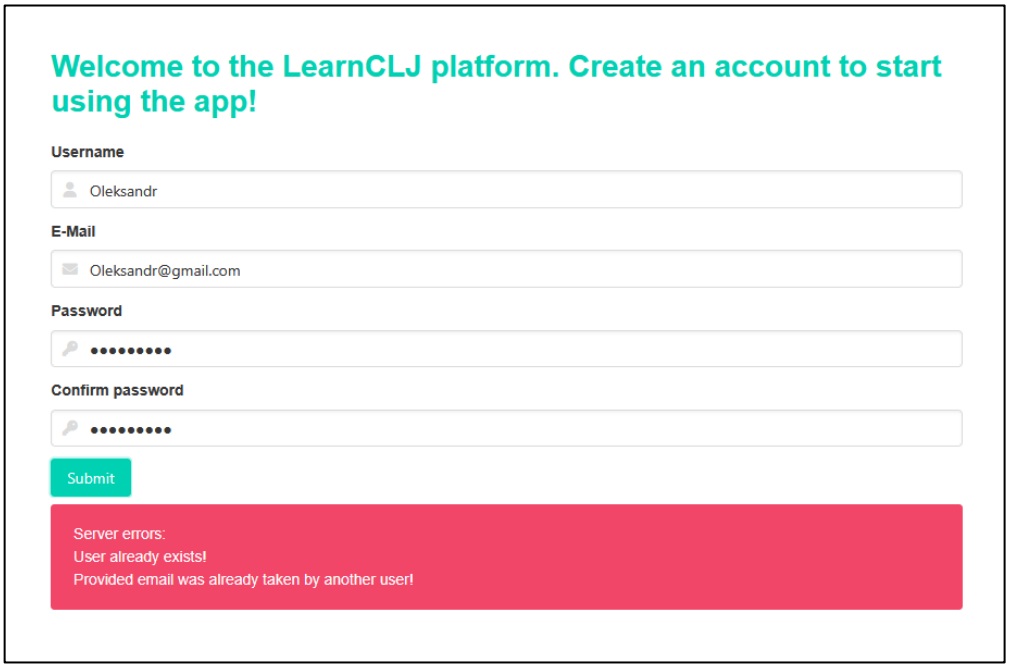


Рисунок 3.6 – Повідомлення про помилку на сервері

На рисунку 3.7 представлений фрагмент коду, що відповідає за валідацію даних на сервері.

```
(defn validate-registration [fields]
  (let [username (:username fields)
        email (:email fields)]
    (cond-> {}
      (auth/user-exists? username)
      (assoc :already-exists-error "User already exists!")
      (auth/email-exists? email)
      (assoc :email-taken-error "Provided email was already taken by another user!"))))
```

Рисунок 3.7 – Валідація даних форми реєстрації на сервері

У разі успішної валідації даних на клієнті та сервері створюється новий запис у базі даних у таблиці users. Задля безпеки пароль користувача зберігається у вигляді хешу, що генерується за допомогою криптографічної бібліотеки для мови Clojure buddy/hashers. На рисунку 3.8 представлена відповідь сервера при успішній реєстрації.

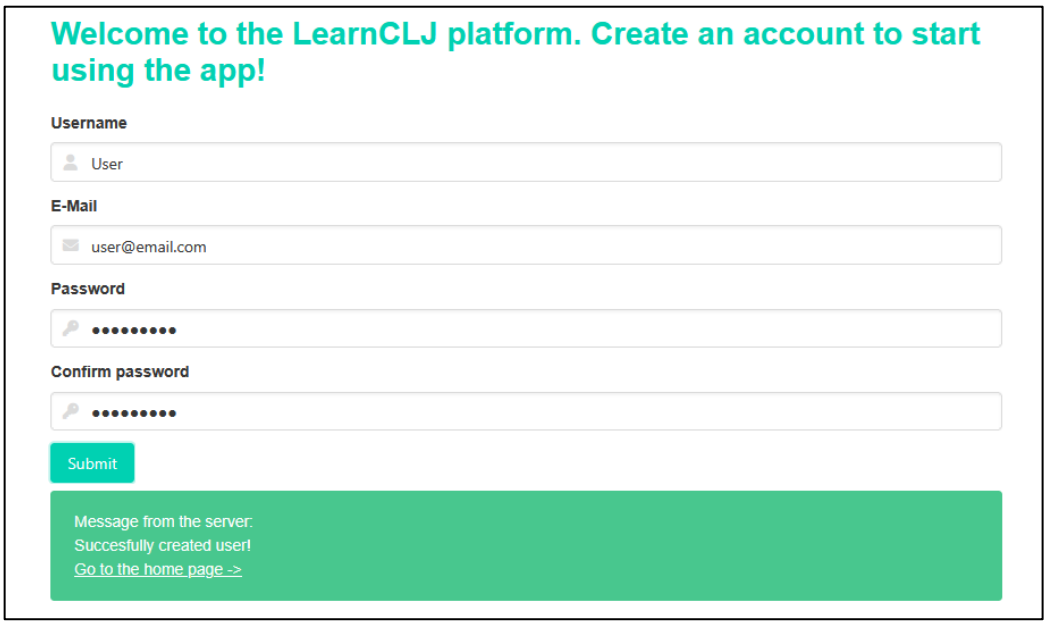


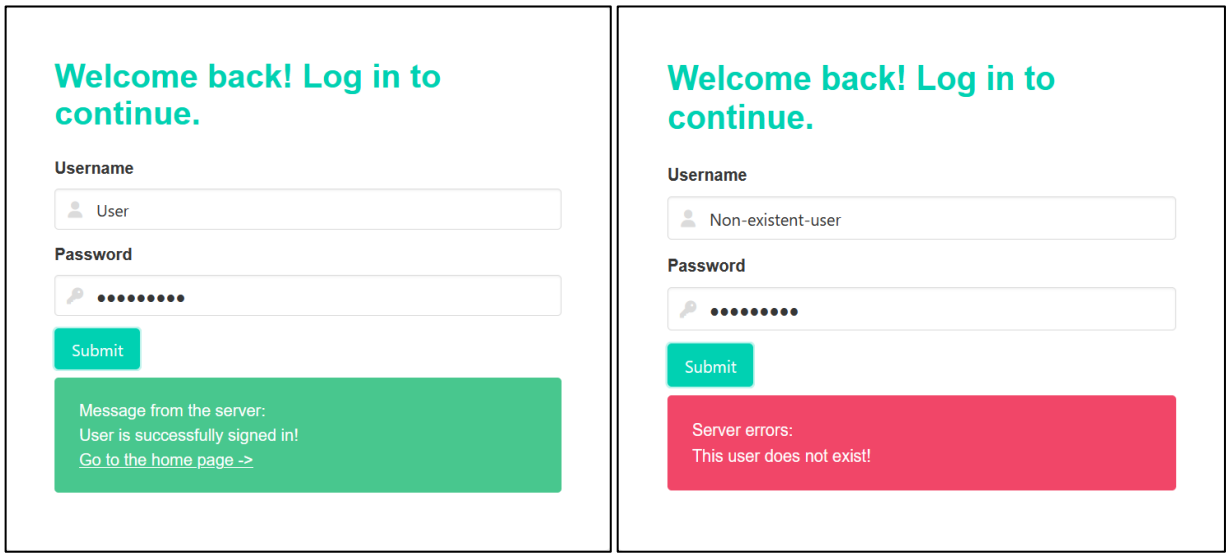
Рисунок 3.8 – Успішна реєстрація користувача

Якщо користувач має обліковий запис, він має здійснити вхід у систему. Для цього необхідно натиснути кнопку “Log in”, що переносить користувача на сторінку із формою для входу. Дані проходять валідацію на стороні клієнта та сервера, аналогічно до попередньої форми реєстрації. На рисунку 3.9 представлений фрагмент коду валідації входу користувача.

```
(defn validate-log-in [fields]
  (let [username (:username fields)
        password (:password fields)]
    (if-not (auth/user-exists? username)
      {:no-user-found "This user does not exist!"}
      (if-not (auth/check-password-hash (:pass (auth/get-user-hash username)) password)
        {:wrong-credentials "The password is wrong!"}
        {}))))
```

Рисунок 3.9 – Валідація даних форми входу

На рисунку 3.10 представлена форма входу із повідомленнями про помилку і успіх входу користувача.



а) повідомлення про успіх б) повідомлення про помилку

Рисунок 3.10 – Форма входу користувача

3.1.3 Перегляд навчальних статей

Після авторизації користувач отримує доступ до перегляду навчальних статей. На рисунку 3.11 представлений компонент графічного інтерфейсу програми, що містить перелік статей.

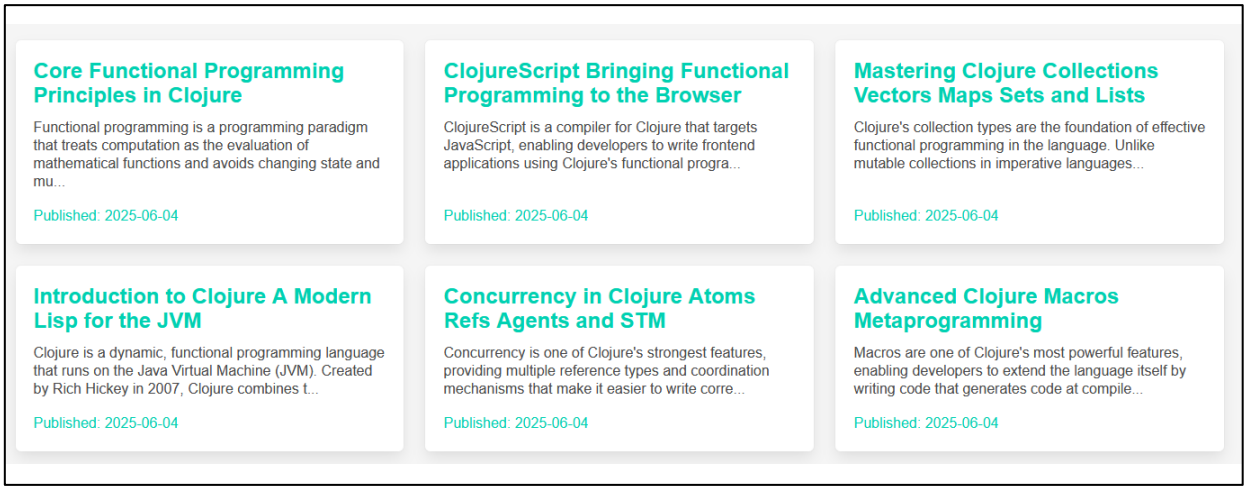


Рисунок 3.11 – Перелік доступних користувачу статей

Навчальні статті зберігаються у базі даних та мають спільну структуру. Кожна стаття має заголовок, дату публікації та вміст, що представляє собою текст статті та фрагменти коду з коментарями. Для перегляду повного списку наявних статей користувач має перейти на відповідну сторінку, натиснувши кнопку “Articles”.

Кожна стаття зі списку має гіперпосилання на свій вміст. При натисканні на заголовок користувачу повертається повний текст статті. На рисунку 3.12 представлений приклад відображення вмісту статті.

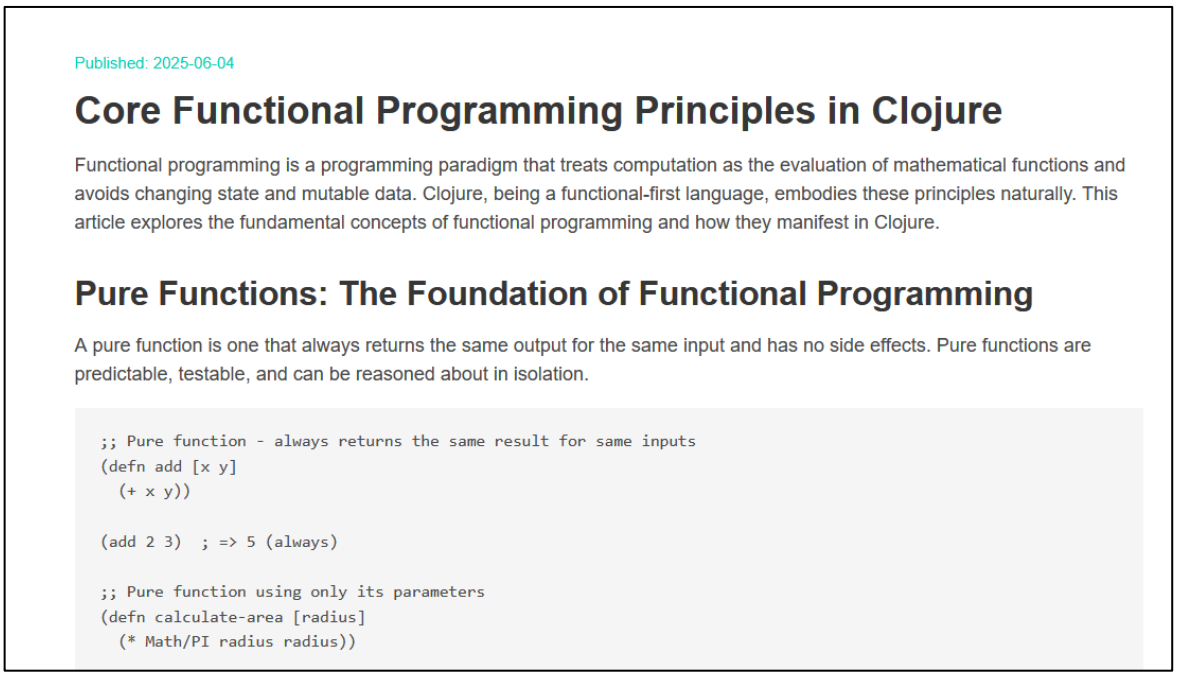


Рисунок 3.12 – Вміст статті

3.1.4 Взаємодія з інтерактивними уроками

Кожен користувач має доступ до інтерактивних уроків, які складаються з двох блоків – теоретичного та практичного. Перший блок містить інформацію на задану тему, а другий представляє собою інтерактивний тест з множинним вибором. На основі теорії поданій в уроці користувач має пройти завдання. Щоб обрати будь-який урок користувач натискає на кнопку “Lessons”. На рисунку 3.13 зображений перелік всіх доступних уроків.

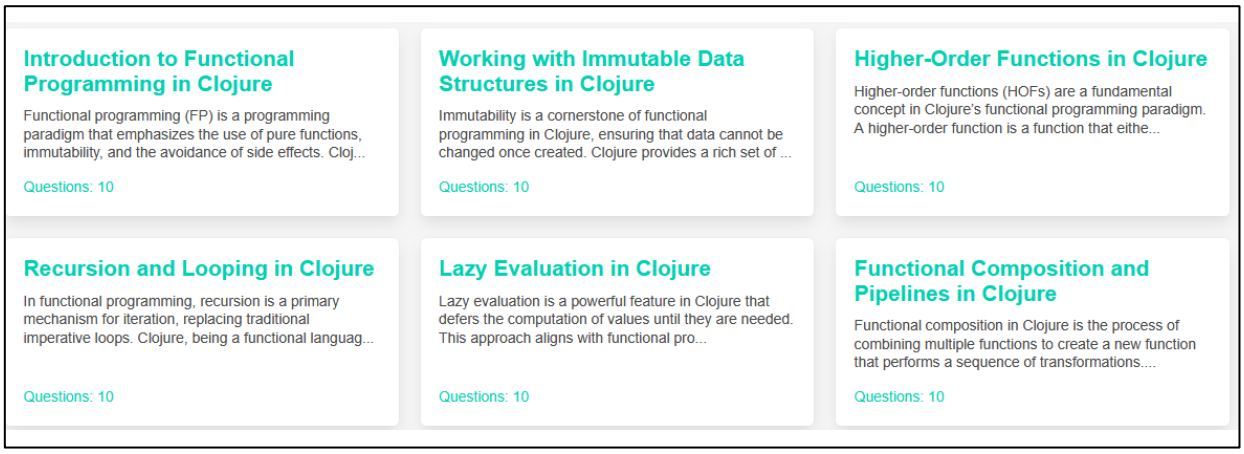


Рисунок 3.13 – Перелік доступних користувачу уроків

При проходженні кожного уроку передбачена автоматична перевірка результатів, про що користувач отримує відповідне повідомлення.

Після проходження уроку користувач бачить кожну свою відповідь, що помічена як правильна або неправильна.

Користувачу також повертається кількість набраних ним балів. За бажанням, користувач може пройти тест повторно.

На рисунку 3.14 та 3.15 зображені відповідні елементи користувацького інтерфейсу.

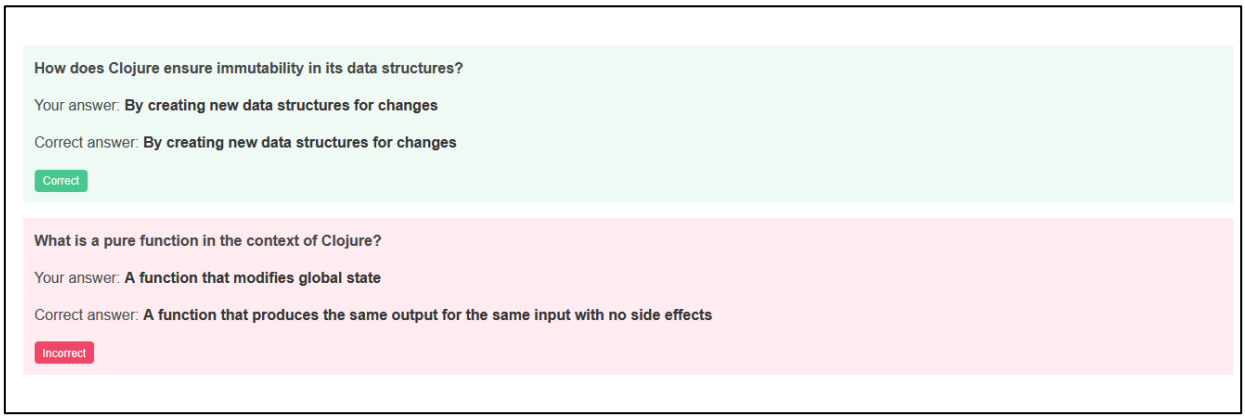


Рисунок 3.14 – Валідація відповідей користувача



Рисунок 3.15 – Загальний результат уроку

3.1.5 Взаємодія з компілятором виразів Clojure

Користувач також має доступ до онлайн-компілятора виразів Clojure, що працює у браузері. Цей компонент дозволяє користувачам експериментувати з базовими конструкціями мови, перевіряти синтаксис і логіку програмного коду в реальному часі, що є надзвичайно важливим для вивчення функціонального підходу.

Компілятор представляє собою компонент користувацького інтерфейсу, що складається з поля для вводу виразів, поля для виводу повідомлень та помилок, а також із підказки, в якій стисло описаний синтаксис типового виразу мови Clojure.

На рисунку 3.16 зображений відповідний елемент користувацького інтерфейсу.

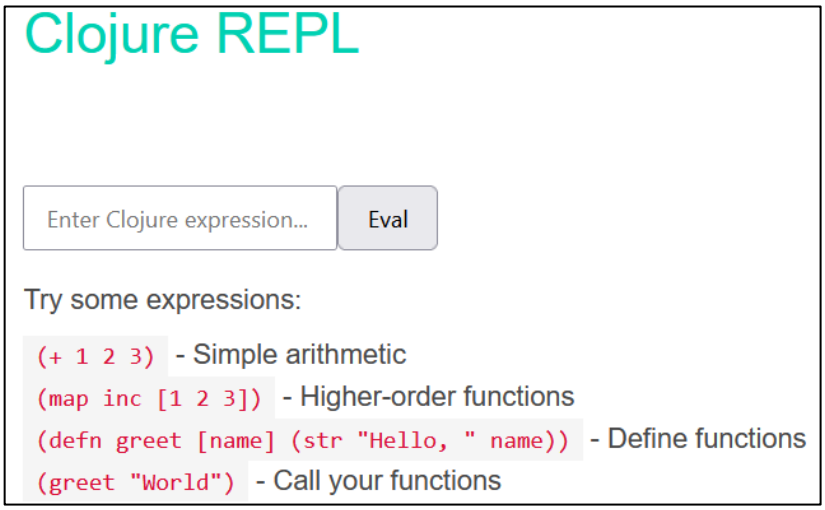


Рисунок 3.16 – Компілятор виразів Clojure

Користувач вводить вираз у поле, натискає кнопку “Eval”, після чого отримує повідомлення про результат компіляції виразу включно з можливими помилками.

Браузерний компілятор має той самий функціонал, що і звичайний компілятор Clojure в терміналі. Приклад роботи компілятора продемонстрований на рисунку 3.17.

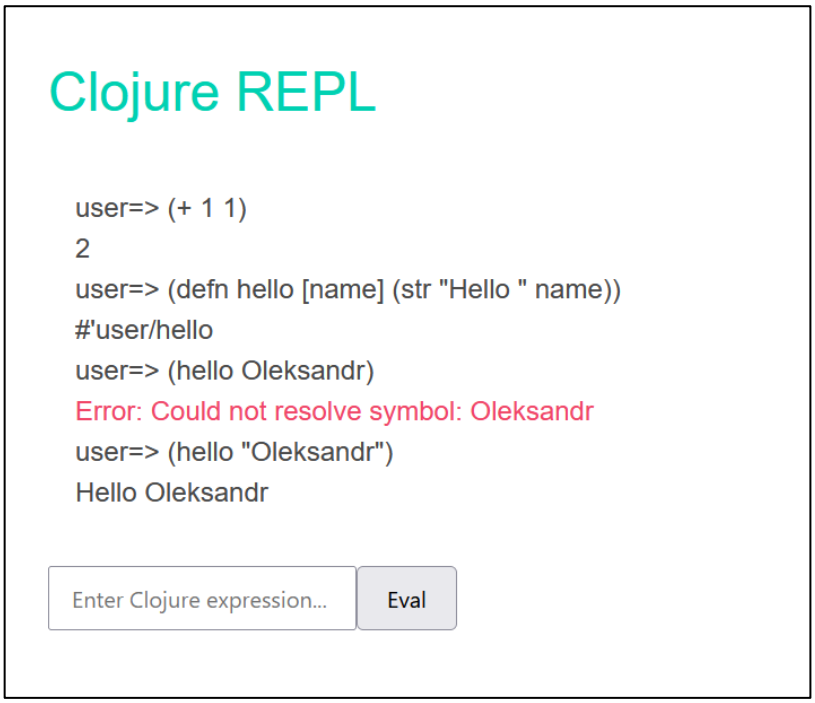


Рисунок 3.17 – Приклад роботи компілятора виразів Clojure

3.2 Unit-тестування

Юніт-тестування є важливою складовою процесу забезпечення якості програмного забезпечення. Його мета полягає у перевірці окремих модулів (юнітів) системи на предмет коректності їх функціонування в ізольованому середовищі.

У межах даної роботи юніт-тести були реалізовані з використанням стандартної бібліотеки clojure.test та інструментів бібліотеки next.jdbc, яка забезпечує взаємодію з базою даних. Для уникнення впливу на реальні дані під час тестування було використано механізм транзакцій з параметром {:rollback-only true}, що гарантує скасування всіх змін після завершення тесту.

Кожен тест реалізовано згідно з принципом Arrange-Act-Assert:

– Arrange (підготовка): визначення вхідних даних та початкового стану системи;
– Act (дія): виклик функції або методу, що тестується;
– Assert (перевірка): порівняння фактичного результату з очікуваним.

Перевірка створення нового користувача. Код тесту відображено на рисунку 3.18.

Мета тесту: метою тесту є переконатись у тому, що функція create-user! створює новий обліковий запис, а get-users дозволяє зчитати відповідну інформацію.

Передумова: активне підключення до БД.

Arrange: підготувати дані користувача: username, email, pass.

Act: виклик db/create-user! повинен повернути 1 – кількість рядків; виклик db/get-users: переконатись, що один запис існує.

Assert: перевірити, що ключі першого запису відповідають вхідним даним.

```
(deftest test-create-and-get-user
  (jdbc/with-transaction [t-conn *db* {:rollback-only true}]
    (let [user {:username "Test"
                :email "testuser@example.com" :pass "testpass"}]
      (is (= 1 (db/create-user! t-conn user)))
      (is (= (select-keys user [:username :email])
             (-> (db/get-users t-conn {}) first (select-keys [:username :email])))))))
```

Рисунок 3.18 – Перевірка створення нового користувача

Перевірка унікальності електроної пошти. Код тесту відображено на рисунку 3.19.

Мета тесту: переконатись, що створення двох користувачів з однаковою електронною адресою викликає помилку (порушення унікальності).

Передумова: БД містить одного користувача з певною email-адресою.

Arrange: підготувати два набори даних користувача з однаковим email.

Act: створити першого користувача; спробувати створити другого користувача з тим самим email.

Assert: перевірити, що другий виклик create-user! викликає виняток (Exception).

```
(deftest test-user-already-exists
  (jdbc/with-transaction [t-conn *db* {:rollback-only true}]
    (db/create-user! t-conn {:username "one"
                             :email "a@example.com"
                             :password "abc"})
    (is (thrown? Exception
          (db/create-user! t-conn {:username "two"
                                   :email "a@example.com"
                                   :password "xyz"})))))
```

Рисунок 3.19 – Перевірка унікальності електроної пошти

Перевірка створення нової статті. Код тесту відображено на рисунку 3.20.

Мета тесту: переконатись, що функція add-article правильно створює новий запис із статею у базі даних.

Передумова: наявне активне підключення до БД, таблиця статей існує.

Arrange: підготувати структуру нової статті.

Act: виклик функції add-article! з підготовленими даними, зчитати список наявних статей.

Assert: перевірити, що серед зчитаних статей є принаймні одна з очікуваним значенням поля slug.

```
(deftest test-create-article
  (jdbc/with-transaction [t-conn *db* {:rollback-only true}]
    (let [article {:title "Introduction to FP"
                   :slug "intro-fp"
                   :body "This article explains functional programming concepts."}]
      (is (= 1 (db/add-article! t-conn article)))
      (is (some #(= "intro-fp" (:slug %)) (db/get-all-articles t-conn))))))
```

Рисунок 3.20 – Перевірка створення нової статті

Результати запуску юніт-тестів продемонстровані на рисунку 3.21.

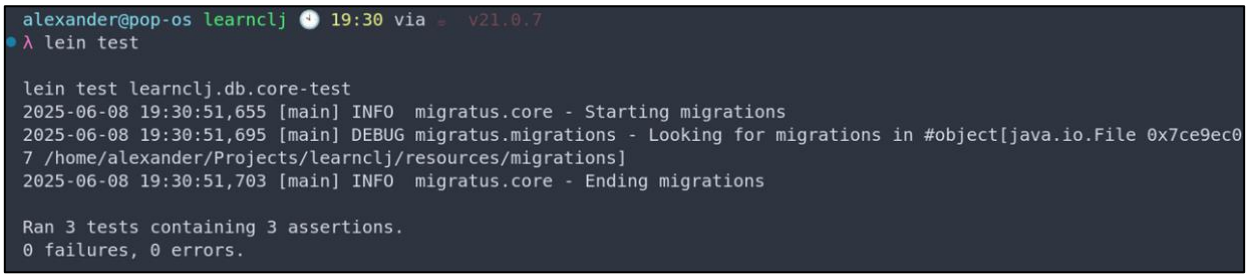


Рисунок 3.21 – Запуск тестів

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було здійснено комплексне дослідження функціонального підходу до програмування, зосереджене на мові Clojure, та розроблено навчальну вебплатформу для інтерактивного засвоєння її основ. У теоретичній частині проєкту проаналізовано сучасні тенденції використання функціональної парадигми у програмуванні, її ключові переваги, а також здійснено порівняння наявних онлайн-ресурсів, таких як 4Clojure, ClojureDocs, Exercism, та Try Clojure. Недоліки і переваги проаналізованих ресурсів були враховані при проєктуванні та розробці навчальної платформи.

У практичній частині було реалізовано вебзастосунок за архітектурою «клієнт-сервер» із використанням фреймворку Luminus для серверної частини та бібліотеки Reagent з Re-frame для клієнтської частини. Розроблено функціонал аутентифікації користувачів, збереження інформації в базі даних, взаємодії з вмістом уроків та онлайн-компілятором виразів мови Clojure. Здійснено тестування основних функцій, зокрема: створення користувачів, перевірки унікальності електронної пошти та збереження навчальних статей. Тести показали стабільну та коректну роботу базової логіки системи. Проєкт реалізовано із врахуванням вимог безпеки та розширюваності.

Розроблене програмне рішення є прикладом того, як сучасні функціональні технології можуть застосовуватись у сфері реалізації програмних рішень для онлайн освіти. Створена платформа може використовуватись як допоміжний інструмент для студентів та викладачів при вивченні функціонального стилю програмування, а також як основа для подальшого розширення і оптимізації функціоналу для розробників.

ПЕРЕЛІК ПОСИЛАНЬ

1. 4ever-clojure. *4clojure*. URL: https://4clojure.oxal.org/ (дата звернення: 10.03.2025).

2. Community-powered documentation and examples repository for the Clojure programming language. *ClojureDocs*. URL: https://clojuredocs.org/ (дата звернення: 12.03.2025).

3. Scotto E. Try clojure. URL: https://tryclojure.org/ (дата звернення: 14.03.2025).

4. Clojure : Want to learn and master Clojure? *Exercism*. URL: https://exercism.org/tracks/clojure (дата звернення: 15.03.2025).

5. Luminus – a Clojure web framework. *Luminus*. URL: https://luminusweb.com/ (дата звернення: 15.04.2025).

6. Luminus Documentation. *Luminus*. URL: https://luminusweb.com/docs/guestbook (дата звернення: 25.04.2025).

7. Closure tools | Google for developers. *Google for Developers*. URL: https://developers.google.com/closure (дата звернення: 25.04.2025).

8. Sotnikov D., Brown S. Web Development with Clojure: Build Large, Maintainable Web Applications Interactively. Raleigh, NC : Pragmatic Bookshelf, 2021. 470 p.