МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Розробка мобільного застосунку для вибору абітурєнтами закладу вищої освіти»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Смачило Богдан Андрійович

Керівник: ***Клебан Ю.В., старший викладач кафедри ЕММІТ***

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса*
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 8 від 06 березня 2025 р.

Острог, 2025

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Розробка мобільного застосунку для вибору абітурєнтами закладу вищої освіти

Автор: *Смачило Богдан Андрійович*
Науковий керівник: *Клебан Ю.В., старший викладач кафедри ЕММІТ*

Захищена «……»…… 2025 року.

Ключові слова: *Сервіс для абітурієнтів, вищі навчальні заклади, фільтрація, пошук, реляційна база даних, Node.js, React Native, RESTful API, мобільний додаток, ефективність, юзабіліті.*

Короткий зміст праці:

Кваліфікаційна робота присвячена розробці інноваційного сервісу, який має на меті надати ефективну допомогу абітурієнтам у виборі закладу вищої освіти. В умовах сучасного світу, коли кількість університетів, спеціальностей і можливостей для навчання значно зросла, вибір відповідного закладу вищої освіти стає складним завданням для багатьох молодих людей. Основна мета цієї роботи — створення сучасного цифрового інструменту, який забезпечить швидкий, інтуїтивно зрозумілий і зручний пошук закладів вищої освіти, а також їх фільтрацію за різними критеріями відповідно до потреб користувачів.

Сервіс буде орієнтований на різноманітну аудиторію, включаючи учнів старших класів, студентів, які бажають перевестися до іншого закладу вищої освіти, а також дорослих, які планують продовжити свою освіту. Головною функцією сервісу є можливість отримати доступ до бази даних університетів, яка містить детальну інформацію про доступні спеціальності, навчальні програми, рейтинги, вимоги до вступу та географічне розташування. Користувачі матимуть змогу сортувати заклади вищої освіти за ключовими параметрами, такими як вартість навчання, мовна програма, наявність стипендій, репутація закладу тощо.

Крім того, важливим аспектом сервісу є інтерактивність. Передбачено впровадження функціоналу для написання відгуків, обговорення питань і взаємодії між користувачами. Це дозволить абітурієнтам отримати об'єктивну інформацію від інших студентів, які вже навчаються в обраних закладах, а також отримувати поради щодо вступу. Окрему увагу буде приділено створенню зручного та естетичного інтерфейсу, який забезпечить позитивний користувацький досвід та полегшить навігацію в системі.

Технічна частина роботи охоплює розробку веб-додатку з використанням сучасних технологій. Для фронтенду буде застосовано React, що забезпечує швидку роботу додатку та гнучкість у реалізації інтерактивного інтерфейсу. Бекенд буде реалізовано за допомогою Node.js та Express, що дозволить ефективно обробляти запити користувачів і забезпечувати надійність роботи сервісу. Для зберігання даних про університети використовуватимуться бази даних SQL та NoSQL, що дозволить зберігати велику кількість структурованої та неструктурованої

інформації. Особливу увагу приділено оптимізації пошуку та фільтрації, що буде реалізовано за допомогою спеціалізованих алгоритмів для швидкого аналізу даних.

Ця робота також охоплює аналіз потреб цільової аудиторії, дослідження ринку подібних сервісів і розробку унікальних функцій, які зроблять продукт конкурентоспроможним. Планується проведення тестування сервісу з участю реальних користувачів, щоб виявити можливі недоліки та вдосконалити систему відповідно до отриманих відгуків. Таким чином, сервіс не лише спрощуватиме процес вибору університету, але й сприятиме підвищенню поінформованості абітурієнтів та їхньої готовності до вступу.

Розроблений інструмент стане корисним не лише для абітурієнтів, а й для самих закладів вищої освіти, які зможуть ефективніше комунікувати з потенційними студентами. У перспективі сервіс може стати міжнародною платформою, яка об'єднуватиме дані про університети з усього світу, забезпечуючи глобальну взаємодію між студентами та закладами вищої освіти.

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Development of the service for selecting higher education institutions*

Author: *Smachylo Bohdan*

Scientific supervisor: *Kleban Y., Senior Lecturer at DEMMIT*

Defensed «……»…… of 2025.

Keywords: *service for applicants, higher education institutions, filtering, search, relational database, Node.js, React Native, RESTful API, mobile application, efficiency, usability.*

Summary of the paper:

The qualification work is dedicated to the development of an innovative service aimed at providing effective assistance to prospective students in choosing a higher education institution. In today's world, where the number of universities, specializations, and educational opportunities has significantly increased, selecting the right educational institution becomes a challenging task for many young people. The primary goal of this work is to create a modern digital tool that ensures quick, intuitive, and convenient search for educational institutions, as well as filtering them according to various criteria based on users' needs.

The service will be targeted at a diverse audience, including high school students, individuals looking to transfer to another institution, and adults planning to continue their education. The main feature of the service is the ability to access a database of universities containing detailed information about available specializations, study programs, rankings, admission requirements, and geographic location. Users will be able to sort educational institutions based on key parameters such as tuition fees, language programs, availability of scholarships, reputation, and more.

Moreover, an important aspect of the service is interactivity. The implementation of functionality for writing reviews, discussing questions, and interacting among users is planned. This will allow prospective students to receive objective information from others who are already studying at selected institutions and gain advice on the admission process. Special attention will be given to creating a user-friendly and aesthetically pleasing interface that ensures a positive user experience and simplifies system navigation.

The technical part of the work involves the development of a web application using modern technologies. React will be used for the frontend, ensuring fast application performance and flexibility in implementing an interactive interface. The backend will be implemented using Node.js and Express, which will enable efficient handling of user requests and ensure the reliability of the service. Databases such as SQL and NoSQL will be used to store information about universities, enabling the storage of large amounts of structured and unstructured data. Particular emphasis will be placed on optimizing search and filtering,

which will be implemented using specialized algorithms for fast data analysis.

This work also includes analyzing the needs of the target audience, researching the market of similar services, and developing unique features that will make the product competitive. User testing with real participants is planned to identify possible shortcomings and refine the system based on feedback. Thus, the service will not only simplify the process of selecting a university but also contribute to increasing prospective students' awareness and readiness for admission.

The developed tool will be useful not only for prospective students but also for educational institutions themselves, which will be able to communicate more effectively with potential students. In the long run, the service could become an international platform that integrates data on universities worldwide, ensuring global interaction between students and educational institutions.

ЗМІСТ

ВСТУП

З розвитком інформаційних технологій та поширенням цифрових платформ, освіта активно інтегрує інноваційні рішення для забезпечення доступу до навчання. Одним із важливих аспектів цього процесу є створення сервісів для пошуку та організації навчальних курсів, що сприяють ефективній взаємодії між студентами, викладачами та закладами вищої освіти через цифрові платформи. У цьому контексті актуальним є створення веб-застосунку, який дозволяє користувачам шукати університети, фільтрувати курси та забезпечувати верифікацію учнів для покращення процесу навчання та забезпечення якості освіти.

Сучасні платформи часто не враховують потреби в організації пошуку конкретних університетів і курсів відповідно до індивідуальних вимог користувачів. Цей недолік стає особливо помітним для студентів, які шукають освітні можливості за кордоном, а також для викладачів та університетів, що прагнуть підвищити видимість своїх програм. Відсутність єдиної платформи, яка б поєднувала всі ці функції, ускладнює доступ до якісної освіти та гальмує процеси навчання.

Актуальність даного дослідження полягає в розробці веб-застосунку для пошуку університетів, фільтрації курсів та забезпечення верифікації учнів, що дозволить покращити доступність освітніх можливостей і забезпечить зручний процес навчання для користувачів.

Метою даного дослідження є створення веб-застосунку для пошуку університетів і курсів з можливістю фільтрації та верифікації учнів, який дозволить користувачам зручний доступ до інформації про освітні можливості та сприятиме підвищенню якості навчального процесу. Об’єктом дослідження є веб-застосунок для пошуку університетів і курсів, а предметом дослідження — методи розробки програмного забезпечення для створення таких систем, включаючи проєктування дизайну, серверної частини та бази даних.

Для досягнення поставленої мети визначено наступні завдання:

1. Аналіз ринку і існуючих рішень для пошуку університетів та курсів.
2. Проєктування дизайну інтерфейсу користувача для зручності пошуку, фільтрації та верифікації.
3. Проєктування бази даних для зберігання інформації про університети, курси та користувачів з урахуванням зручності та ефективності доступу до даних.
4. Вибір архітектурного рішення для реалізації бекенду на основі сучасних технологій, таких як Node.js, для ефективної обробки запитів.
5. Розробка технічного стеку для створення серверної частини, інтеграція сучасних бібліотек для розширення функціональності.
6. Розробка функціоналу для пошуку університетів, фільтрації курсів та верифікації учнів, включаючи процеси аутентифікації і авторизації користувачів.

Для виконання дослідження будуть застосовані методи аналізу існуючих платформ, побудови діаграм варіантів використання, проєктування бази даних, а також ручне тестування для перевірки коректності роботи компонентів системи.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ПРОЄКТУВАННЯ ТА РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ ДЛЯ ПОШУКУ УНІВЕРСИТЕТІВ І КУРСІВ: ДИЗАЙН СЕРВЕРНОЇ ЧАСТИНИ, БАЗИ ДАНИХ ТА ФРОНТЕНДУ

1.1. Опис предметного середовища

"UniSearch" — це веб-застосунок, що дозволяє користувачам шукати університети та фільтрувати курси за різними параметрами, забезпечуючи зручний доступ до освітніх можливостей. Платформа також інтегрує систему верифікації учнів, що сприяє покращенню якості навчання та організації освітнього процесу.
Функціональною моделлю проєкту UniSearch є:

1. Реєстрація та авторизація користувачів:
    - Студенти можуть створити обліковий запис на платформі, зареєструвавшись через email та пароль або сторонні сервіси, такі як Google.
    - Користувачі можуть увійти до системи за допомогою власного облікового запису.
2. Пошук університетів та програм:
    - Користувачі можуть здійснювати пошук університетів за різними критеріями, такими як країна, спеціальність, рівень навчання (бакалаврат, магістратура тощо).
    - Для кожного університету можна переглядати детальну інформацію про доступні програми та курси, такі як вимоги до вступу, тривалість, вартість та інші важливі параметри.
3. Фільтрація та сортування результатів пошуку:
    - Користувачі можуть фільтрувати результати за різними параметрами, такими як напрямок навчання, мова викладання, вартість, країна чи рейтинг університету.
    - Функція сортування дозволяє сортувати університети та програми за

рейтингом, популярністю чи іншими критеріями.

4. Збереження вибраних університетів та програм:
    - Користувачі можуть зберігати університети та програми, які їх цікавлять, для подальшого перегляду або порівняння.
    - Це дозволяє створити особистий список для легкого доступу до вибраних варіантів у майбутньому.
5. Реєстрація на програми та подача заявок:
    - Студенти можуть подати заявки на вибрані програми університетів через платформу, вказуючи необхідні документи та дані.
    - Інтерфейс для подачі заявок забезпечує простоту та зручність для користувачів, включаючи автоматичну перевірку на наявність всіх необхідних документів.
6. Модерація університетів та програм:
    - Адміністратори платформи здійснюють перевірку та модерацію інформації про університети та програми, гарантуючи, що дані є актуальними та достовірними.
    - Платформа надає можливість університетам оновлювати свої програми та курси для забезпечення актуальності.
7. Функціонал дизайну та фронтенду:
    - Платформа має інтуїтивно зрозумілий і адаптивний дизайн, який дозволяє користувачам швидко знаходити необхідну інформацію про університети та програми.
    - Дизайн інтерфейсу враховує зручність пошуку та фільтрації, з можливістю перегляду результатів на різних пристроях.
    - Використовуються сучасні фреймворки та бібліотеки для забезпечення ефективності і комфортного використання.

Процес діяльності проєкту UniSearch:

1. Пошук університетів та програм:

- Користувач вводить критерії пошуку (країна, спеціальність, рівень освіти тощо) для знаходження відповідних університетів та програм.
- Система надає список університетів, які відповідають заданим критеріям, з можливістю фільтрації та сортування результатів.

2. Перегляд інформації про університети та програми:
   - Користувач може детально ознайомитися з програмами, курсами та умовами вступу на сторінках кожного університету.
   - Інтерфейс дозволяє легко переглядати інформацію про університети, порівнювати програми та отримувати додаткові матеріали, якщо вони доступні.

1.2. Огляд наявних аналогів

У сфері підтримки абітурієнтів у виборі вищих закладів вищої освіти існує кілька сервісів, які надають подібні функції до розроблюваного UniSearch. Ці сервіси варіюються за функціональністю, географічним охопленням та підходами до представлення інформації. Нижче наведено аналіз основних з них:

1. Study.ua

Опис: Study.ua є одним із найпопулярніших українських порталів для абітурієнтів, що шукають інформацію про вищі заклади вищої освіти України та зарубіжжя.

Функціональні можливості:

- Пошук університетів за різними критеріями (місто, спеціальність, рівень навчання).
- Порівняння програм навчання.
- Відгуки студентів та випускників.
- Інформація про вступні іспити та вимоги до абітурієнтів.

Переваги:

- Велика база даних закладів вищої освіти.
- Інтерактивні інструменти для порівняння програм.
- Актуальні новини та статті про освіту.

Недоліки:

- Інтерфейс може бути не інтуїтивно зрозумілим для нових користувачів.
- Обмежені можливості персоналізації пошуку.

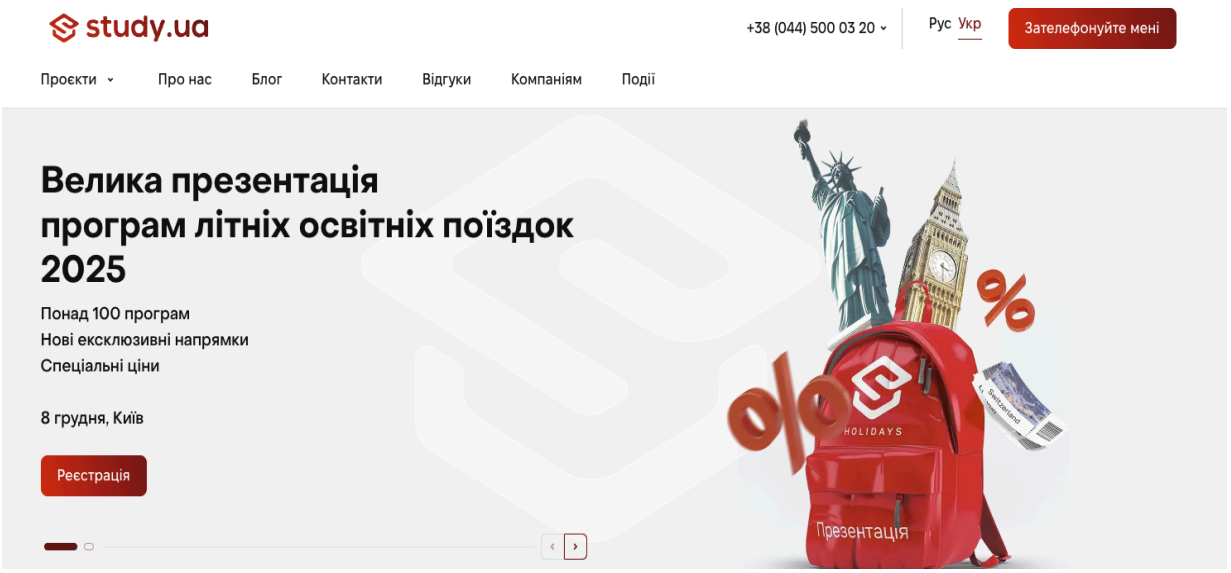

Рис.1.1 Інтерфейс на сайті Study.ua

2. UniRank

Опис: UniRank є глобальним рейтингом університетів, який надає інформацію про акредитовані заклади вищої освіти по всьому світу.

Функціональні можливості:

- Рейтинг університетів за країною, регіоном або світовим рівнем.
- Фільтрація за типом університету (публічний, приватний), рівнем навчання та формою власності.
- Детальна інформація про програми навчання, умови вступу та міжнародні партнерства.

Переваги:

- Величезна база даних університетів із понад 200 країн.

- Простий та зрозумілий інтерфейс.
- Постійно оновлюваний контент.

Недоліки:

- Відсутність інтерактивних інструментів для порівняння.
- Менш деталізовані дані про конкретні програми та відгуки студентів.

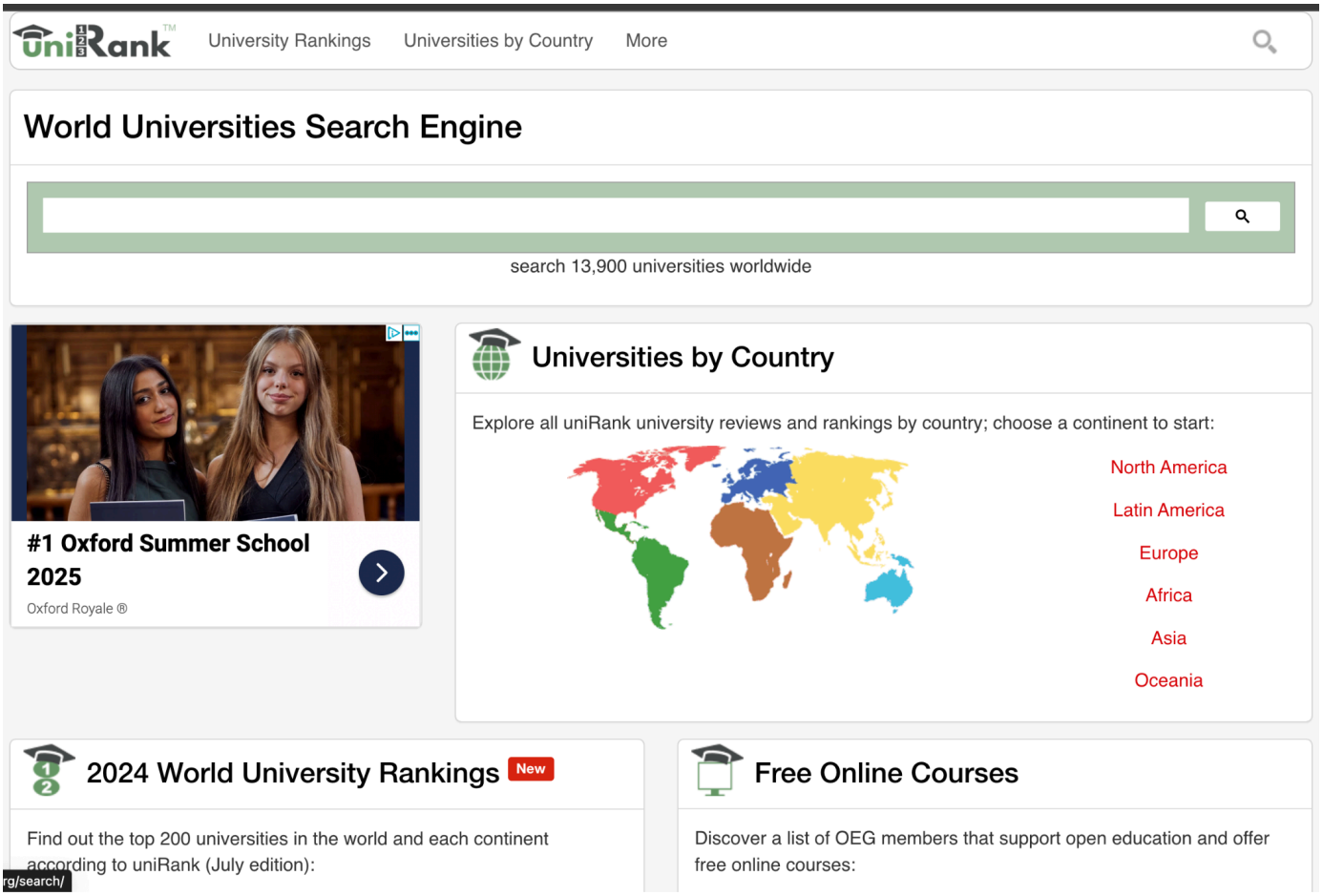

Рис. 1.2 Інтерфейс на сайті UniRank

3. QS World University Rankings

Опис: QS є провідним джерелом інформації про університети, їхні рейтинги та програми навчання. Сервіс орієнтований на абітурієнтів, які шукають найкращі заклади для навчання по всьому світу.

Функціональні можливості:

- Рейтинги університетів на глобальному та регіональному рівні.

- Фільтрація за спеціальностями, географією та рейтингами.
- Огляд дослідницьких програм, викладацької діяльності та академічного середовища.

Переваги:

- Висока репутація серед абітурієнтів та освітніх установ.
- Великий акцент на наукову діяльність і якість викладання.
- Можливість перегляду детальних даних про університети.

Недоліки:

- Більший акцент на рейтинги, ніж на персоналізований підхід до вибору університетів.
- Відсутність опції для порівняння конкретних програм між собою.

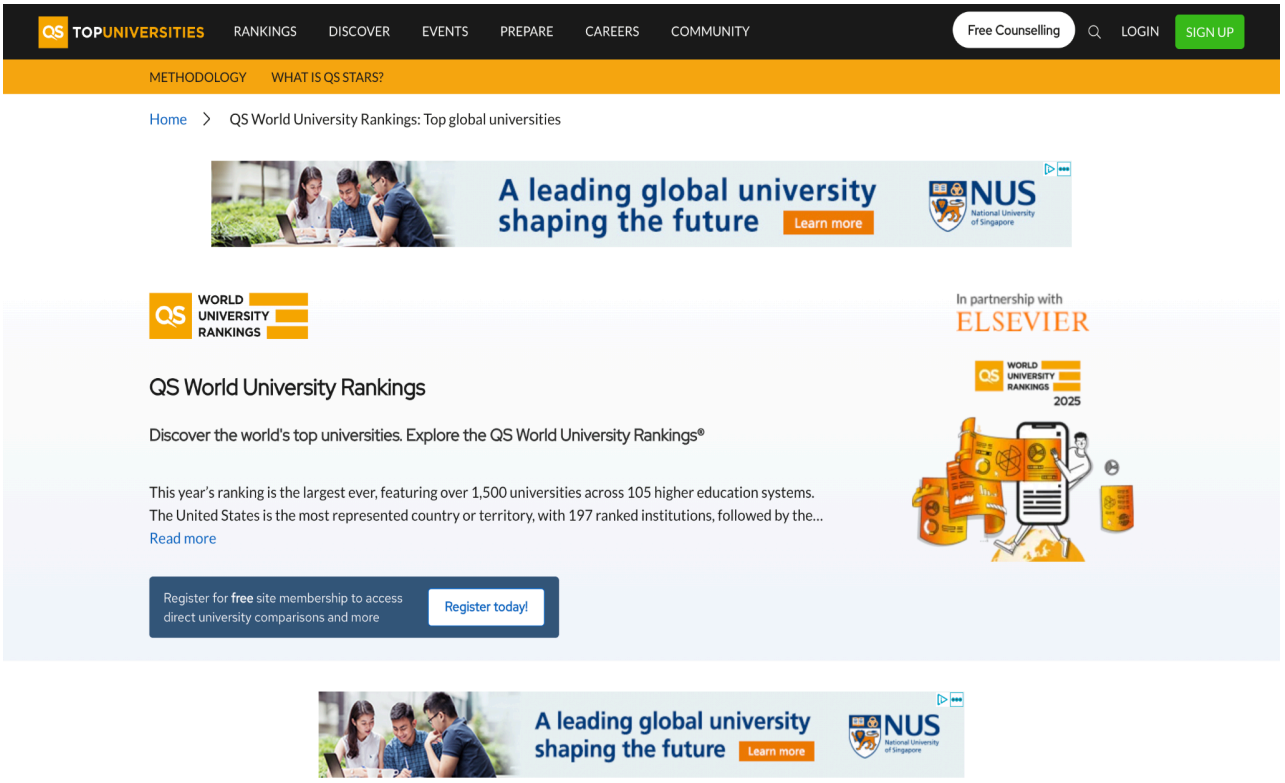

Рис. 1.3 Інтерфейс на сайті QS World University Rankings

Підсумок огляду:

Огляд аналогів UniSearch дозволяє виділити кілька ключових особливостей та недоліків, які притаманні існуючим платформам для абітурієнтів в Україні. Попри

широку функціональність і значний обсяг даних, такі сервіси, як Study.ua, UniRank та QS World University Rankings, не завжди відповідають усім потребам українських користувачів.

Головні спостереження:

1. Study.ua має велику базу університетів та інструменти для порівняння, але відсутність зручного та персоналізованого інтерфейсу ускладнює пошук відповідних програм.
2. UniRank надає багато загальної інформації про університети, проте бракує відгуків студентів та можливостей порівняння програм.
3. QS World University Rankings зосереджується на рейтингах і міжнародному аналізі, але мало враховує специфіку локальних потреб абітурієнтів в Україні.

Чого не вистачає наявним сервісам для українського ринку:

- Локалізованого підходу: немає достатньо даних про українські університети, їхні програми та специфіку вступу.
- Інтерактивних функцій: відсутність платформ для спілкування між абітурієнтами та можливості отримувати поради від студентів чи випускників.
- Верифікації абітурієнтів: недостатньо механізмів, які б дозволяли підтвердити достовірність заяв абітурієнтів або їхню відповідність вступним вимогам.
- Персоналізації пошуку: обмежені можливості для налаштування критеріїв пошуку відповідно до індивідуальних потреб і пріоритетів.

UniSearch має потенціал заповнити ці прогалини, створивши платформу, орієнтовану на українських користувачів, яка поєднуватиме зручність, інтерактивність та точність у наданні інформації.

1.3. Створення технічного завдання

При розробці веб-застосунку UniSearch, який допомагатиме абітурієнтам у виборі університетів, особлива увага приділяється зручності використання,

швидкому доступу до інформації та інтерактивності. Платформа складатиметься з серверної частини (Node.js + Express), бази даних (MongoDB) та клієнтської частини (React Native).

Ключовий функціонал системи:

1. Реєстрація та аутентифікація
    - Реєстрація користувачів із верифікацією email.
    - Аутентифікація через email і пароль.
    - Захист особистих даних через шифрування паролів.
2. Особистий кабінет
    - Редагування особистої інформації.
    - Перегляд історії активності: коментарі, поставлені питання.
    - Управління налаштуваннями профілю.
3. Пошук університетів
    - Інтерактивний пошук із фільтрами: географія, спеціальність, рейтинг.
    - Відображення списку університетів із ключовими параметрами.
    - Перегляд детальної інформації про університет (контакти, опис, рейтинги).
4. Коментарі та обговорення
    - Можливість залишати коментарі до профілів університетів.
    - Перегляд і відповідь на коментарі інших користувачів.
    - Система модерації для забезпечення якості обговорень.
5. Запитання та відповіді
    - Форма для постановки запитань про університети.
    - Інтерфейс для відповідей представників університетів або інших користувачів.
    - Відображення популярних запитань у профілях університетів.

Реалізація бекенду (Node.js + Express):

- Маршрути API:

- /auth: реєстрація, вхід, відновлення пароля.
- /users: отримання та редагування даних користувачів.
- /universities: пошук, перегляд інформації, фільтри.
- /comments: додавання, редагування, видалення коментарів.
- /questions: створення, перегляд, відповіді на запитання.

- База даних MongoDB:
  - Колекції:
    - Users: дані про користувачів.
    - Universities: інформація про університети.
    - Comments: коментарі до університетів.
    - Questions: запитання та відповіді.

Реалізація фронтенду (React):

1. Дизайн інтерфейсу
   - Простий і зручний UI/UX для реєстрації, пошуку та спілкування.
   - Компоненти для відображення університетів, фільтрів, форм коментарів і запитань.
2. Ключові сторінки
   - Головна сторінка: пошук університетів і популярні профілі.
   - Сторінка університету: детальна інформація, коментарі, популярні запитання.
   - Особистий кабінет: історія активності, редагування профілю.
3. Зв’язок із бекендом
   - Використання бібліотеки Axios або Fetch для роботи з API.
4. Стейт-менеджмент
   - Використання Redux для управління станом додатка.
5. Реалізація модулів
   - Форма реєстрації та входу: інтеграція з бекендом для авторизації.
   - Модуль коментарів: відображення й управління коментарями в

реальному часі.

- Форма запитань: можливість швидкого створення та перегляду відповідей.

Висновки до розділу 1

У процесі аналізу було визначено ключові аспекти та потреби абітурієнтів, які обирають заклади вищої освіти, а також досліджено існуючі сервіси, що надають подібний функціонал. Виявлено, що наявні рішення в Україні не повною мірою задовольняють потреби користувачів, зокрема через недостатню інтерактивність, обмежений набір функцій для взаємодії з іншими користувачами та відсутність централізованої платформи для обговорення університетів і поставлення запитань.

На основі проведеного аналізу було сформовано технічне завдання для розробки UniSearch. Основна мета платформи – надати абітурієнтам інструмент, який не лише полегшить пошук відповідних закладів вищої освіти, а й забезпечить можливість активного обговорення, залишення коментарів і обміну досвідом. Реалізація платформи з використанням сучасних технологій, таких як Node.js, MongoDB та React, дозволить створити інтерактивний, зручний і масштабований веб-застосунок, орієнтований на потреби українських користувачів.

Таким чином, розроблений концепт платформи UniSearch стане важливим кроком у вирішенні актуальних проблем вибору університетів, підвищивши доступність, якість і зручність інформаційної підтримки абітурієнтів.

РОЗДІЛ 2. РОЗРОБКА ДИЗАЙНУ ТА ПРОЄКТУВАННЯ СЕРВЕРНОЇ І КЛІЄНТСЬКОЇ ЧАСТИН ВЕБ-ЗАСТОСУНКУ

2.1 Проєктування та розробка дизайну веб-застосунку

2.1.1 Загальні принципи дизайну

При створенні дизайну веб-застосунку було дотримано ключових принципів, спрямованих на забезпечення зручності використання, естетичності та функціональності інтерфейсу. Основою для розробки стали підходи, які відповідають сучасним стандартам UX/UI дизайну.

Одним із головних принципів стала орієнтація на користувача. Інтерфейс було спроєктовано з урахуванням потреб цільової аудиторії, акцентуючи увагу на доступності та інтуїтивності використання. Усі функціональні елементи розташовані таким чином, щоб користувач міг швидко знайти необхідну інформацію чи виконати потрібну дію.

Важливу роль у проєктуванні відіграв принцип мінімалізму. Інтерфейс веб-застосунку розроблено без зайвих декоративних елементів, що дозволяє фокусуватися на основних функціях. Простота форм, чіткі межі елементів та обмежене використання кольорів створюють приємний візуальний досвід.

Окрім того, дизайн застосунку базується на принципі консистентності. Усі елементи інтерфейсу, від кнопок до шрифтів, виконано в єдиному стилі, що сприяє загальній гармонії системи та полегшує взаємодію користувача із застосунком.

Для забезпечення адаптивності дизайн було оптимізовано для різних типів пристроїв, включаючи комп’ютери, планшети та смартфони. Використання підходу responsive design дозволяє зберігати зручність та естетичність інтерфейсу незалежно від розміру екрана.

Додатково було враховано вибір типографіки та кольорової гами. Для тексту

обрано сучасний, легко читабельний шрифт, а кольорова схема підтримує нейтральні відтінки з акцентами, які привертають увагу до ключових елементів.

Загальний підхід до дизайну відповідає сучасним тенденціям веб-дизайну, інтегруючи такі концепції, як flat design і micro interactions, для забезпечення сучасного вигляду та зручності роботи з застосунком.

2.1.2. UX/UI-підхід до створення зручності користувача

Зручність використання є одним із найважливіших факторів, що визначають успіх веб-застосунку. У рамках розробки UniSearch застосовувалися передові принципи UX (User Experience) та UI (User Interface) дизайну, щоб забезпечити інтуїтивність взаємодії, візуальну привабливість і ефективність роботи з платформою. Особливу увагу було приділено тому, як користувачі орієнтуються на сайті, взаємодіють із його функціями та отримують доступ до потрібної інформації.

Основою UX-дизайну стало створення інтуїтивної навігації, яка забезпечує швидкий доступ до ключових функцій. Меню розташовано у верхній частині екрана, що є звичним для більшості користувачів, а кнопки виконання дій чітко виділяються кольором і розміром. Крім того, були враховані потреби користувачів у мінімізації дій для досягнення мети: наприклад, для написання коментаря достатньо натиснути на спеціальне поле, а фільтрація університетів здійснюється у кілька кліків.

UI-дизайн базується на принципах сучасності та естетичності. Було використано flat design, який забезпечує простоту форм, чіткість та легкість сприйняття інформації. Основний акцент зроблено на візуальній ієрархії: важливі елементи, як-от кнопки для основних дій, виділено за допомогою контрастних кольорів і більшого розміру. Крім того, типографіка була підібрана так, щоб текст залишався максимально читабельним на різних екранах, а кольорова схема підтримувала відчуття гармонії та зосередженості.

Головна сторінка є першим пунктом взаємодії користувача із застосунком. Вона має чітко структуровану область пошуку університетів та зрозумілі кнопки для переходу до ключових функцій платформи.

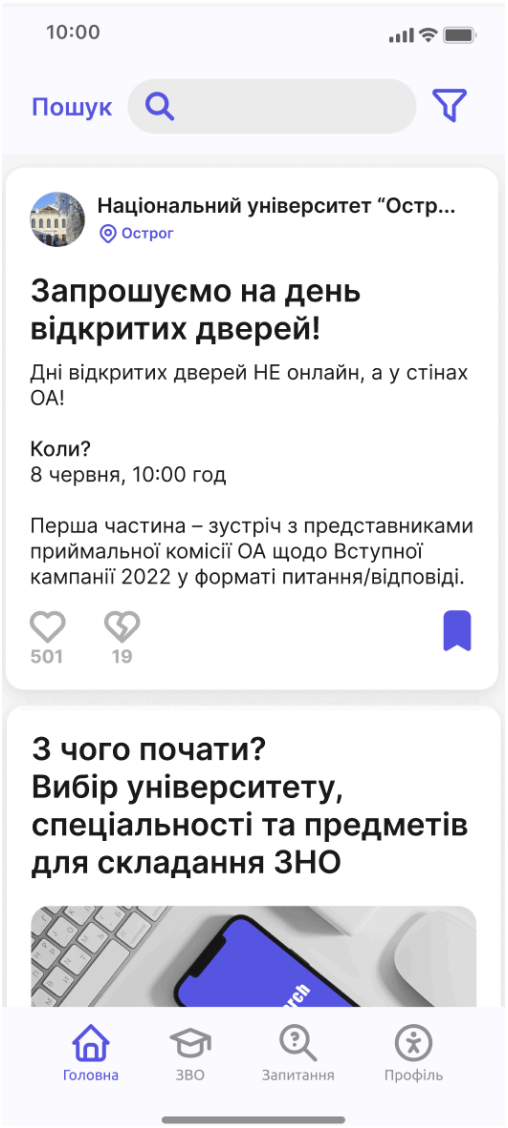

Рис. 2.1 Дизайн головної сторінки

Сторінка пошуку університетів дозволяє фільтрувати заклади за різними критеріями, як-от місцезнаходження, спеціальність чи рейтинг. Інформація про кожен університет представлена у вигляді карточок, які забезпечують легке сприйняття завдяки чіткій структурі.

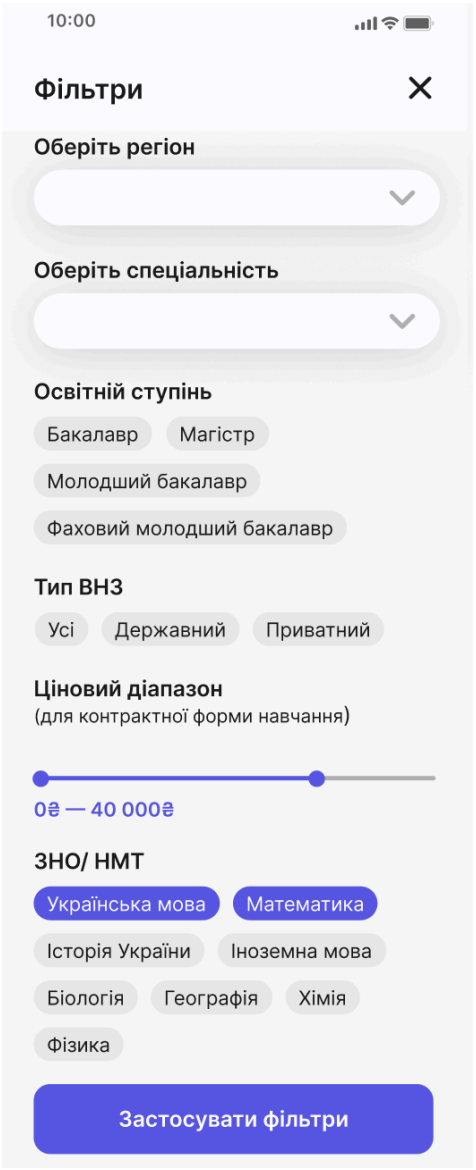

Рис. 2.2 Дизайн пошуку ВНЗ

Сторінка коментарів і запитань передбачає зручне поле для введення тексту, а також можливість переглядати відповіді інших користувачів. Нові відповіді чи активні обговорення відзначаються окремим маркуванням, що допомагає користувачам швидко знаходити актуальну інформацію.

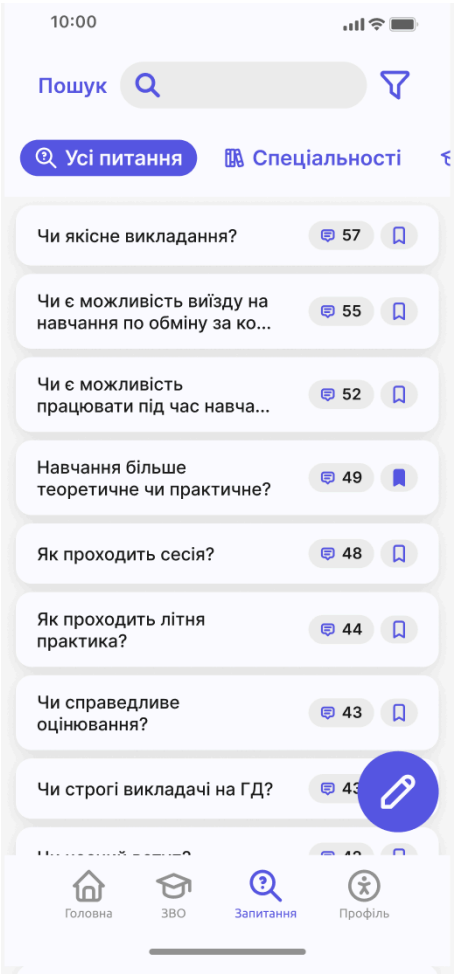

Рис. 2.3 Дизайн сторінки з запитаннями

Процес створення профілю користувача є ключовим етапом взаємодії із застосунком, адже саме через нього користувач отримує доступ до персоналізованих функцій платформи. Інтерфейс реєстрації був розроблений таким чином, щоб забезпечити максимальну простоту та зручність. Форми реєстрації містять лише необхідні поля, а підказки й валідація даних допомагають уникнути помилок під час введення інформації.

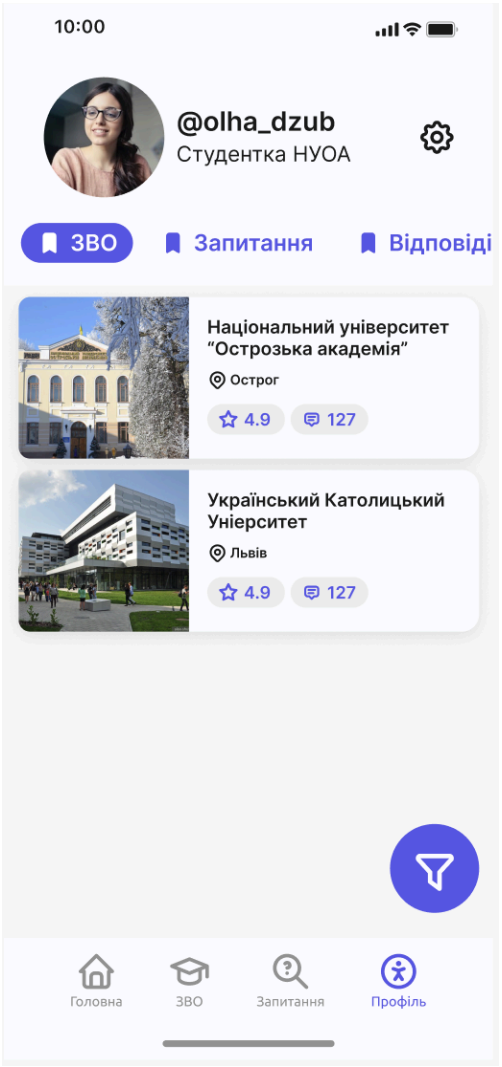

Рис. 2.4 Дизайн сторінки профілю

2.2. Розробка серверної частини веб-застосунку

Розробка серверної частини є важливим етапом у створенні веб-застосунку, оскільки саме вона забезпечує обробку запитів користувачів, взаємодію з базою даних, а також реалізацію бізнес-логіки. Для цього необхідно вибрати відповідні технології, спроектувати архітектуру серверної частини, впровадити механізми авторизації та аутентифікації, а також забезпечити ефективне збереження та обробку даних.

2.2.1. Вибір технологій для серверної частини

Для розробки серверної частини веб-застосунку обрано стек технологій, що дозволяє забезпечити високу продуктивність, безпеку та масштабованість. Вибір технологій включає:

- Node.js як платформу для розробки серверної частини, оскільки вона дозволяє ефективно обробляти асинхронні запити та підтримує велику кількість одночасних з'єднань.
- Express.js — фреймворк для Node.js, який забезпечує простоту створення RESTful API, управління маршрутами та обробку запитів.
- MongoDB як базу даних для зберігання даних про користувачів, курси, коментарі та іншу інформацію. MongoDB дозволяє зберігати структуровані та неструктуровані дані, а також забезпечує високу швидкість обробки запитів.

Цей стек технологій є оптимальним для веб-застосунків, що вимагають гнучкості, швидкої розробки та масштабованості.

2.2.2. Архітектура серверної частини (структура API, робота з базою даних)

Архітектура серверної частини побудована за принципом RESTful API, що забезпечує чітке розділення між клієнтською та серверною частинами. Ключові компоненти архітектури:

- API маршрути: ключовим елементом серверної архітектури – це забезпечення взаємодії між клієнтським інтерфейсом та сервером. Кожен маршрут (ендпоінт) відповідає за конкретну операцію, пов’язану з отриманням, обробкою або збереженням даних.
- Моделі даних: Взаємодія з базою даних здійснюється через моделі, які визначають структуру документів у MongoDB. Наприклад, модель для користувачів містить інформацію про ім’я, email, пароль, а модель для курсів

— дані про назву, опис, викладачів тощо.

- Міжшаровий підхід: Архітектура також включає бізнес-логіку, яка обробляє запити, взаємодіє з базою даних та формує відповіді для клієнта.

Ця архітектура дозволяє гнучко масштабувати застосунок і легко додавати нові функції.

2.2.3. Реалізація функціоналу авторизації та аутентифікації

Для забезпечення безпеки доступу до особистих даних користувачів була реалізована система авторизації та аутентифікації. Ключові аспекти реалізації:

- Паролі та хешування: Для зберігання паролів користувачів використовуються алгоритми хешування (наприклад, bcrypt), що дозволяє забезпечити безпечне зберігання паролів, не зберігаючи їх у відкритому вигляді.

Це гарантує захист даних користувачів і знижує ризик несанкціонованого доступу до їх акаунтів.

2.2.4. Забезпечення збереження та обробки даних

Дані користувачів, курси та інша важлива інформація зберігаються в MongoDB, що є документно-орієнтованою базою даних. Основні функціональні можливості:

- Зберігання інформації про користувачів: Кожен користувач має свій профіль, який містить інформацію про ім’я, email, пароль (у зашифрованому вигляді), список зареєстрованих курсів тощо.
- Інформація про курси та коментарі: Користувачі можуть додавати курси, коментувати їх та задавати питання. Усі ці дані зберігаються у відповідних колекціях у базі даних.
- Запити до бази даних: Використовуються ефективні запити для отримання необхідної інформації, наприклад, пошук курсів за певними критеріями або

відображення коментарів до курсу.

Це дозволяє швидко і ефективно зберігати та обробляти велику кількість даних, забезпечуючи зручність та ефективність роботи з платформою.

2.3. Розробка клієнтської частини веб-застосунку

Розробка клієнтської частини веб-застосунку забезпечує взаємодію користувача з системою через інтерфейс. Важливими аспектами цього процесу є вибір технологій для фронтенду, структура проєкту, розробка компонентів інтерфейсу, інтеграція з серверним API, а також забезпечення адаптивності та кросбраузерності застосунку. Всі ці етапи важливі для того, щоб забезпечити зручність та ефективність взаємодії користувача з веб-застосунком на різних пристроях.

2.3.1. Вибір технологій для фронтенд розробки

Для розробки клієнтської частини вибрано React Native як основну технологію для створення інтерактивних та зручних інтерфейсів для мобільних пристроїв. Це дозволяє одночасно розробляти застосунок для iOS та Android, зменшуючи час на розробку та забезпечуючи високу продуктивність.

- React — бібліотека для створення користувацьких інтерфейсів, яка дозволяє ефективно керувати станом додатку та створювати інтерактивні елементи.
- React Native — фреймворк, який дозволяє використовувати JavaScript та React для розробки мобільних застосунків. За допомогою React Native можна створювати мобільні додатки, які будуть працювати на різних платформах без необхідності писати окремий код для кожної з них.
- Redux — бібліотека для керування станом, що дозволяє централізувати збереження стану застосунку та полегшує його оновлення при зміні даних.

Вибір цих технологій забезпечує високу гнучкість та масштабованість

клієнтської частини, дозволяючи швидко вносити зміни і реалізовувати нові функціональні можливості.

2.3.2. Структура проєкту на React Native

Проєкт на React Native має свою структуру, яка дозволяє чітко організувати код та забезпечити його легке розширення в майбутньому. Основні компоненти структури проєкту:

- src — папка, що містить основний код застосунку. Вона поділяється на підкатегорії:
    - components — компоненти інтерфейсу, які відповідають за відображення окремих елементів інтерфейсу (кнопки, форми, списки тощо).
    - screens — екрани застосунку, які відповідають за відображення різних сторінок або вікон у додатку.
    - navigation — налаштування навігації між екранами додатку, використовуючи бібліотеки на зразок React Navigation.
    - redux — для зберігання стану додатку та його оновлення.
    - assets — зображення, іконки, шрифти та інші ресурси.

Структура проєкту допомагає зберігати порядок і полегшує роботу з великими проєктами, забезпечуючи чистоту та зрозумілість коду.

2.3.3. Реалізація компонентів інтерфейсу

Компоненти інтерфейсу є основою клієнтської частини веб-застосунку, і вони повинні бути зручними для користувачів, а також функціональними. Основні етапи реалізації:

- Створення кнопок, форм та полів введення – це базові елементи інтерфейсу, які дозволяють користувачам взаємодіяти з додатком. Наприклад, для реєстрації або авторизації користувачів використовуються форми введення

даних, а для навігації між екранами — кнопки.

- Списки та таблиці: Вони використовуються для відображення даних, таких як список курсів або коментарів. Компоненти для списків часто потребують додаткових функцій для пагінації або фільтрації.
- Модальні вікна та поп-апи: Використовуються для відображення сповіщень, помилок або для підтвердження дій користувача (наприклад, підтвердження видалення даних).

Ці компоненти мають бути адаптованими до різних розмірів екранів і забезпечувати зручний користувацький досвід.

2.3.4. Інтеграція з API

Для забезпечення функціональності застосунку необхідно здійснити інтеграцію з серверним API, яке відповідає за обробку запитів користувачів та взаємодію з базою даних. Основні етапи інтеграції:

- Запити до API: Клієнтська частина відправляє запити до серверу для отримання даних або для виконання операцій, таких як реєстрація або авторизація користувачів. Для цього використовуються HTTP методи (GET, POST, PUT, DELETE).
- Обробка відповіді від серверу: Після отримання даних від серверу застосунок обробляє їх і відображає на екрані користувача. Це може бути список курсів, повідомлення про помилку або сповіщення про успішне виконання дії.
- Управління помилками: Якщо запит до серверу не вдався, важливо відобразити користувачу відповідне повідомлення, яке пояснює причину помилки.

Інтеграція з API є критично важливою для забезпечення функціональності застосунку та його взаємодії з серверною частиною.

Висновки до розділу 2

У другому розділі було здійснено комплексне проектування дизайну, серверної та клієнтської частин веб-застосунку. Основними етапами розробки стали вибір технологій, створення архітектури та реалізація ключових функцій системи.

Було обґрунтовано вибір сучасних технологій для кожної з частин застосунку. Для клієнтської частини обрано React Native, що забезпечує ефективну розробку кросплатформних інтерфейсів. Для серверної частини використано Node.js та Express.js, які гарантують стабільну роботу API та зручну взаємодію з базою даних. Для збереження та обробки даних інтегровано MongoDB, яка забезпечує гнучкість у роботі з документами, масштабованість і надійність збереження інформації.

Розробка дизайну була спрямована на створення сучасного, адаптивного та інтуїтивно зрозумілого інтерфейсу. Особлива увага приділялася UX/UI-підходам для забезпечення зручності використання веб-застосунку. Реалізація компонентів інтерфейсу враховувала необхідність адаптивності для коректного відображення на мобільних і десктопних пристроях.

У рамках проектування серверної частини реалізовано функціонал авторизації та аутентифікації користувачів, а також розроблено механізми збереження та обробки даних. Забезпечено верифікацію користувачів для підтримки безпеки системи. Інтеграція клієнтської та серверної частин виконувалась через RESTful API, що дозволяє ефективно обробляти запити користувачів та забезпечувати їхню взаємодію із системою.

Загалом, виконане проектування створює надійний фундамент для подальшої реалізації функціоналу, тестування та вдосконалення веб-застосунку, забезпечуючи зручність користувачів, стабільність роботи і масштабованість системи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИН ВЕБ-ЗАСТОСУНКУ

3.1 Опис необхідних засобів для розробки

Для розробки веб-застосунку було використано сучасні технології, які забезпечують надійну роботу серверної та клієнтської частин, швидкість розробки та легкість інтеграції.

Основні технології:

- Node.js — платформа для створення серверної частини веб-додатків, що забезпечує швидку обробку запитів завдяки своїй асинхронній архітектурі.
- Express.js — фреймворк для Node.js, який дозволяє створювати API з мінімальними зусиллями.
- MongoDB — документоорієнтована база даних NoSQL, яка забезпечує гнучке зберігання даних із можливістю масштабування.
- React — JavaScript-бібліотека для створення інтерактивного та динамічного інтерфейсу користувача.

Основні бібліотеки:

- Mongoose — бібліотека для роботи з MongoDB, яка спрощує визначення моделей даних та взаємодію з базою.
- bcrypt.js — бібліотека для хешування паролів, що гарантує безпеку збережених даних користувачів.
- Axios — клієнт для HTTP-запитів, який використовується на фронтенді для комунікації з сервером.
- Formik та Yup — бібліотеки для валідації та управління формами на клієнтській частині, що забезпечують коректність введених даних.
- React Router — інструмент для реалізації маршрутизації в клієнтській частині, що дозволяє створювати багатосторінкові SPA-додатки.

3.2 Вибір програмного забезпечення

Для реалізації серверної та клієнтської частин веб-додатка було використано наступне програмне забезпечення, яке сприяло швидкому та ефективному процесу розробки.

- Visual Studio Code — легке, але потужне середовище розробки, яке підтримує інтеграцію з популярними мовами програмування, розширеннями та інструментами для Node.js, React, MongoDB. Завдяки своїй гнучкості VS Code став найкращим вибором для розробки веб-застосунку.
- MongoDB Atlas — хмарний сервіс для роботи з базою даних MongoDB, що забезпечує зручне управління базою, моніторинг продуктивності та легку інтеграцію з серверною частиною.
- Postman — інструмент для тестування API-запитів, який дозволив протестувати всі основні функції серверної частини, такі як реєстрація, аутентифікація, створення коментарів та отримання інформації про користувачів.
- Git та GitHub — система контролю версій та платформа для спільної роботи над проектом. Використання GitHub дозволило відстежувати всі зміни в коді, створювати окремі гілки для різних функціональностей та проводити злиття без ризику втрати даних.
- Figma — інструмент для проектування UX/UI дизайну. Використання Figma дозволило створити прототипи інтерфейсу користувача, які відображають логіку роботи веб-застосунку та забезпечують зручність навігації.

  Ці інструменти та технології забезпечили комплексний підхід до створення надійного, масштабованого та зручного веб-застосунку, що відповідає сучасним стандартам розробки.

3.3 Здійснення розробки функціоналу для серверної частини веб-додатку

У межах розробки веб-додатку було створено API для реєстрації та входу користувачів. Код використовує Node.js, Express та bcrypt для хешування паролів і забезпечення безпеки.

1. Реєстрація користувачів (Signup)

Метою є забезпечення безпечної реєстрації користувачів. Основні кроки:

1. Перевірка наявності всіх необхідних полів у запиті.
2. Валідація вхідних даних:
    - Поле `nickName` має містити лише літери.
    - Поле `email` перевіряється за допомогою регулярного виразу.
    - Пароль повинен мати щонайменше 8 символів.
3. Перевірка, чи користувач із зазначеним email уже існує в базі даних.
4. Хешування пароля за допомогою bcrypt.
5. Збереження нового користувача в базу даних.

Лістинг 3.1. Реалізація компонента реєстрації користувача у системі НАЗВА

```
router.post("/signup", async (req, res) => {
 const { nickName, email, password } = req.body;
 // Перевірка наявності даних
 if (!nickName || !email || !password) {
   return res.status(400).json({
     status: "FAILED",
     message: "Required fields are missing in the request body",
   });
 }
 // Очищення та валідація вхідних даних
 const trimmedNickName = nickName.trim();
 const trimmedEmail = email.trim().toLowerCase();
 const trimmedPassword = password.trim();
 if (!/^[a-zA-Z ]*$/.test(trimmedNickName)) {
   return res.status(400).json({ message: "Invalid nickname
entered" });
 }
 if
(!/^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/.test(trimmedEmail))
{
   return res.status(400).json({ message: "Invalid email entered"
});
 }
 if (trimmedPassword.length < 8) {
   return res.status(400).json({ message: "Password is too short!"
});
 }
 // Перевірка унікальності email
 const existingUser = await User.findOne({ email: trimmedEmail });
 if (existingUser) {
   return res.status(400).json({ message: "User already exists!"
});
```

```
  }
  // Хешування пароля
  const saltRounds = 10;
  const hashedPassword = await bcrypt.hash(trimmedPassword,
saltRounds);
  // Створення нового користувача
  const newUser = new User({
    nickName: trimmedNickName,
    email: trimmedEmail,
    password: hashedPassword,
  });
  // Збереження у базу даних
  try {
    const savedUser = await newUser.save();
    res.status(201).json({ message: "Signup successful", data:
savedUser });
  } catch (error) {
    res.status(500).json({ message: "Error saving user" });
  }
});
```

2. Вхід користувачів (Signin)

Метою є автентифікація користувачів на платформі. Основні кроки:

1. Перевірка введених даних (email, пароль).
2. Пошук користувача в базі даних.
3. Порівняння хешованого пароля з введеним за допомогою bcrypt.
4. Повернення успішного статусу після перевірки.

Код функції входу:

Лістинг 3.2 код виходу

```
    router.post("/signin", async (req, res) => {
     const { email, password } = req.body;
     const trimmedEmail = email.trim();
     const trimmedPassword = password.trim();
     if (!trimmedEmail || !trimmedPassword) {
       return res.status(400).json({
         message: "Missing email or password",
       });
     }
     const user = await User.findOne({ email: trimmedEmail });
     if (!user) {
       return res.status(404).json({ message: "User not found"
});
     }
     const isPasswordValid = await
bcrypt.compare(trimmedPassword, user.password);
     if (!isPasswordValid) {
       return res.status(400).json({ message: "Invalid
credentials" });
     }
     res.status(200).json({ message: "Signin successful", user
});
    });
```

3. Модель University (Mongoose)

Це схема, яка визначає, як зберігати інформацію про університети у базі даних MongoDB. Вона містить наступні поля:

- name: Назва університету.
- location: Інформація про місце розташування університету.
    - city: Місто.
    - region: Регіон.

- descriptions: Опис університету.
    - shortDescription: Короткий опис університету.
    - fullDescription: Детальний опис університету.
- universityImage: Зображення університету.
- faculties: Список факультетів університету.
    - Кожен факультет може мати:
        - name: Назва факультету.
        - departments: Відділи факультету.
            - Кожен відділ може мати:
                - name: Назва відділу.
                - programs: Програми, що пропонуються відділом.
                    - Кожна програма може включати:
                        - level: Рівень програми (наприклад, бакалавр, магістр).
                        - specialties: Спеціальності в межах програми.
                            - name: Назва спеціальності.
                            - description: Опис спеціальності.
                            - price: Вартість.
                            - requiredSubjects: Перелік предметів, які потрібні для вступу.
                            - image: Зображення.
- admissionCriteria: Критерії для вступу в університет.
    - generalCriteria: Загальні критерії вступу.
    - specificCriteria: Спеціальні критерії для кожної програми.
        - programName: Назва програми.
        - criteria: Критерії вступу.
        - deadlines: Дедлайни для подачі заявок.

        - requiredDocuments: Необхідні документи для вступу.
        - applicationProcess: Процес подачі заяви.
- scholarshipsAndFinancialAid: Інформація про стипендії та фінансову допомогу.
    - name: Назва стипендії.
    - description: Опис стипендії.
    - eligibilityCriteria: Критерії для отримання стипендії.
- campusFacilities: Інфраструктура університету.
    - libraries: Бібліотеки.
    - sportsFacilities: Спортивні споруди.
    - hostels: Гуртожитки.
    - diningServices: Їдальні.
- studentLife: Студентське життя.
    - clubsAndSocieties: Клуби та товариства.
    - events: Події університету.
- careerServices: Кар'єрні послуги.
    - internshipOpportunities: Можливості для стажувань.
    - jobPlacementServices: Послуги працевлаштування.
    - alumniNetworks: Мережі випускників.
- reviewsAndTestimonials: Відгуки студентів.
    - author: Автор відгуку.
    - year: Рік написання відгуку.
    - text: Текст відгуку.
- contactInformation: Контактні дані університету.
    - admissionsOffice: Офіс для прийому заявок.
    - studentSupport: Підтримка студентів.
    - academicDepartments: Академічні відділи.

Маршрути API

GET /all

Отримує список всіх університетів, що зберігаються в базі даних.

- Якщо університети знайдені, вони повертаються у форматі JSON.
- Якщо університети не знайдені, повертається повідомлення про помилку.

POST /add

Додає новий університет до бази даних за допомогою даних, що передаються в тілі запиту.

- Якщо додавання університету успішне, повертається повідомлення про успіх.
- Якщо сталася помилка, повертається повідомлення про невдачу.

GET /:name

Шукає університет за його назвою, використовуючи регулярні вирази для нечутливого до регістру пошуку.

- Якщо університет знайдений, повертається його інформація.
- Якщо університет не знайдений, повертається повідомлення про помилку.

Приклад запиту:

- GET /all
    - Повертає список всіх університетів.
- POST /add

Тіло запиту:

Лістинг 3.3 Тіло запиту для унівеситетів

```
{
 "name": "University of Example",
 "location": {
   "city": "Example City",
   "region": "Example Region"
 },
 "descriptions": {
   "shortDescription": "A leading university in Example City.",
   "fullDescription": "This university offers a wide range of
undergraduate and postgraduate programs."
 },
```

```
 "universityImage": "url-to-image.jpg",
 "faculties": [
   {
     "name": "Engineering",
     "departments": [
       {
         "name": "Computer Science",
         "programs": [
           {
             "level": "Undergraduate",
             "specialties": [
               {
                 "name": "Software Engineering",
                 "description": "This program focuses on the
development of software applications.",
                 "price": 10000,
                 "requiredSubjects": ["Math", "Physics"],
                 "image": "software-eng.jpg"
           }
         ]
       }
     ]
   }
 ],
 "admissionCriteria": {
   "generalCriteria": "A minimum GPA of 3.0 is required.",
   "specificCriteria": [
     {
       "programName": "Software Engineering",
       "criteria": "Completion of high school with strong
mathematics background.",
       "deadlines": "April 30th",
       "requiredDocuments": ["High School Diploma", "Transcript"],
       "applicationProcess": "Apply online."
```

```
      }
    ]
  }
}
```

Цей API дозволяє взаємодіяти з даними університетів, отримувати інформацію про факультети, програми та критерії вступу, а також додавати нові університети.

4. Модель Comment (Mongoose)

Ця схема визначає, як зберігати інформацію про коментарі користувачів до університетів. Кожен коментар пов'язаний з певним університетом і включає наступні поля:

Лістинг 3.4 Код коментарів

```
const mongoose = require('mongoose');

const commentSchema = new mongoose.Schema({
   universityId: {
       type: mongoose.Schema.Types.ObjectId,
       ref: 'University',  // Це поле вказує на університет, до
якого належить коментар
       required: true
   },
   author: {
       type: String,
       required: true
   },
   text: {
       type: String,
       required: true
   },
   date: {
```

```
        type: Date,
        default: Date.now  // Дата створення коментаря
    }
});

const Comment = mongoose.model('Comment', commentSchema);

module.exports = Comment;
```

Маршрути API для роботи з коментарями

GET /comments/:universityId

Отримує список всіх коментарів до конкретного університету за його ID.

- **Параметри**:
    - `universityId` - ID університету, до якого потрібно отримати коментарі.
- Якщо коментарі знайдені, вони повертаються у форматі JSON.
- Якщо коментарі не знайдені, повертається повідомлення про помилку.

Лістинг 3.5 Код запиту для коментарів

```
router.get('/comments/:universityId', async (req, res) => {
 try {
     const { universityId } = req.params;
     const comments = await Comment.find({ universityId: 
universityId });
     if (comments.length === 0) {
         return res.status(404).json({ status: "FAILED", message: 
"No comments found" });
     }
     res.json({ status: "SUCCESS", message: "Comments retrieved 
successfully", data: comments });
 } catch (error) {
```

```
    console.error(error);
    res.status(500).json({ status: "FAILED", message: "An error
occurred while fetching the comments" });
 }
});
```

POST /comments/add

Додає новий коментар до університету.

Тіло запиту:

Лістинг 3.6 Тіло запитів

```
{
 "universityId": "60c72b1f4f1a2b1c6c8e4d8a", // ID університету, до
якого додається коментар
 "author": "John Doe",
 "text": "This is a great university. The campus is beautiful and
the professors are very helpful."
}
```

- Якщо коментар успішно доданий, повертається повідомлення про успіх.
- Якщо сталася помилка, повертається повідомлення про невдачу.

Лістинг 3.7 Код добавлення коментарів

```
router.post('/comments/add', async (req, res) => {
 try {
    const { universityId, author, text } = req.body;
    const newComment = new Comment({
        universityId,
        author,
        text
    });
```

```
    const savedComment = await newComment.save();
    res.status(201).json({ status: "SUCCESS", message: "Comment
added successfully", data: savedComment });
 } catch (error) {
    console.error(error);
    res.status(500).json({ status: "FAILED", message: "An error
occurred while adding the comment" });
 }
});
```

DELETE /comments/:id

Видаляє коментар за його ID.

- Параметри:
    - `id` - ID коментаря, який потрібно видалити.
- Якщо коментар успішно видалено, повертається повідомлення про успіх.
- Якщо коментар не знайдено, повертається повідомлення про помилку.

Лістинг 3.8 Код видалення коментарів

```
router.delete('/comments/:id', async (req, res) => {
 try {
    const { id } = req.params;
    const deletedComment = await Comment.findByIdAndDelete(id);
    if (!deletedComment) {
        return res.status(404).json({ status: "FAILED", message:
"Comment not found" });
    }
    res.json({ status: "SUCCESS", message: "Comment deleted
successfully" });
 } catch (error) {
    console.error(error);
```

```
    res.status(500).json({ status: "FAILED", message: "An error
occurred while deleting the comment" });
 }
});
```

Приклад запиту:

1. GET /comments/:universityId
    - Параметри: `universityId` — це ID університету.

Приклад:

Лістинг 3.9 Приклад запиту отримання коментарів

```
GET /comments/60c72b1f4f1a2b1c6c8e4d8a
```

Відповідь:

Лістинг 3.10 Приклад відповіді отримання коментарів

```
{
 "status": "SUCCESS",
 "message": "Comments retrieved successfully",
 "data": [
   {
     "author": "John Doe",
     "text": "This is a great university. The campus is beautiful
and the professors are very helpful.",
     "date": "2024-12-06T12:34:56.789Z"
   },
   {
     "author": "Jane Smith",
     "text": "I had an amazing experience at this university,
```

```
highly recommend!",
     "date": "2024-12-05T14:22:33.123Z"
   }
 ]
}
```

2. POST /comments/add

Тіло запиту:

Лістинг 3.11 Приклад тіла запита добавлення коментарів

```
{
 "universityId": "60c72b1f4f1a2b1c6c8e4d8a",
 "author": "John Doe",
 "text": "This university is amazing!"
}
```

Відповідь:

Лістинг 3.12 Відвід на запит добавлення коментарів

```
{
 "status": "SUCCESS",
 "message": "Comment added successfully",
 "data": {
   "_id": "60c72b1f4f1a2b1c6c8e4d9a",
   "universityId": "60c72b1f4f1a2b1c6c8e4d8a",
   "author": "John Doe",
   "text": "This university is amazing!",
   "date": "2024-12-06T12:34:56.789Z"
 }
}
```

3. DELETE /comments/:id
    - Параметри: `id` — це ID коментаря.

Приклад:

Лістинг 3.13 Приклад запиту видалення коментарів

```
DELETE /comments/60c72b1f4f1a2b1c6c8e4d9a
```

Відповідь:

Лістинг 3.14 Відповід видалення коментарів

```
{
 "status": "SUCCESS",
 "message": "Comment deleted successfully"
}
```

Цей API дозволяє користувачам додавати коментарі до університетів, отримувати їх список і видаляти коментарі за ID. Це дозволяє створити взаємодію користувачів з університетами через їх відгуки та думки.

3.4 Здійснення розробки функціоналу для клієнської частини веб-додатку

. Розробка функціоналу для клієнтської частини веб-додатку

На клієнтській стороні веб-додатка ми використовуємо React Native для створення інтерфейсу користувача та Redux для управління станом додатка. Ось як реалізується основна структура та функціональність клієнтської частини:

Структура файлів:

1. App.js — Головний компонент додатка, який підключає Redux Store та навігацію.

2. Redux — Для управління станом користувача та його автентифікацією. Використовуємо Redux для зберігання та обробки даних користувача (наприклад, профілю та статусу автентифікації).
    - `store.js` — Налаштування Redux Store.
    - `userReducer.js` — Редуктор для збереження та оновлення даних користувача.
    - `userActions.js` — Дії, що змінюють стан користувача, наприклад, для входу чи виходу.
3. Навігація — Використовуємо бібліотеку `react-navigation` для керування екранами та їх переходами.
    - `AppNavigator.js` — Основна навігація додатка, яка визначає, які екрани доступні.
    - `AuthNavigator.js` — Навігація для екранів авторизації та реєстрації.
4. Реєстрація та авторизація — Екрани для входу та реєстрації, де використовується бібліотека `Formik` для керування формами.
    - `Login.js` — Екран для авторизації користувача.
    - `Register.js` — Екран для реєстрації нового користувача.
    - Використовуються компоненти `InputField.js`, `Button.js` для побудови форм.
5. Екрани — Компоненти для кожного окремого екрана додатка.
    - `Profile.js` — Екран для відображення профілю користувача (включає `ProfileHeader`).
    - `Question.js` — Екран для відображення питання.
    - `UniversitySearch.js` — Екран для пошуку університетів.
6. assets — Зображення, іконки та інші статичні ресурси.
7. index.js — Точка входу для додатка, де ініціалізується React Native додаток.

Опис компонентів та файлів:

1. App.js - Підключення Redux та навігації

Лістинг 3.15 redux в коді

```
import React from 'react';
import { Provider } from 'react-redux';
import RootStack from './navigators/RootStack';
import store from './redux/store';

const App = () => {
 return (
   <Provider store={store}>
     <RootStack />
   </Provider>
 );
};

export default App;
```

2. Redux Store та Reducer - Store обробляє дані користувача за допомогою userReducer.

Лістинг 3.16 Redux store

```
import { createStore, combineReducers } from 'redux';
import userReducer from './reducers/userReducer';

const rootReducer = combineReducers({
 user: userReducer,
});

const store = createStore(rootReducer);

export default store;
```

Лістинг 3.17 Redux reducer

```
const initialState = {
 user: null,
};
const userReducer = (state = initialState, action) => {
 switch (action.type) {
   case 'SET_USER':
     return {
       ...state,
       user: action.payload,
     };
   case 'LOGOUT_USER':
     return {
       ...state,
       user: null,
     };
   default:
     return state;
 }
};
export default userReducer;
```

3. RootStack.js - Навігація між екранами складається з кількох екранів, таких як екран авторизації, реєстрації та головна сторінка з вкладками.

Лістинг 3.18 Головна навігація

```
// navigators/RootStack.js
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
```

```
import { createNativeStackNavigator } from
'@react-navigation/native-stack';
import BottomTabNavigator from './BottomTabNavigator';

import Login from '../screens/auth/Login';

import Signup from '../screens/auth/Signup';
import ProfileSettings from '../components/ProfileSettings';
import { Notifications } from '../components/Notifications';
import  PersonalEdit  from '../components/PersonalEdit';
// import Main from '../screens/main/Main';
const Stack = createNativeStackNavigator();

const RootStack = () => {
   return (
       <NavigationContainer>
         <Stack.Navigator
               screenOptions={{
                   headerShown: false
               }}
               initialRouteName='Login'
           >
               <Stack.Screen name="Login" component={Login} />
               <Stack.Screen name="Signup" component={Signup} />
               <Stack.Screen name="Main"
component={BottomTabNavigator}/>
               <Stack.Screen
                   name="ProfileSettings"
                   component={ProfileSettings}
                   options={{ headerShown: true, title: 'Profile
Settings' }}
               />
               <Stack.Screen
                   name="Notifications"
```

```
                    component={Notifications}
                    options={{ headerShown: true, title:
'Notifications' }}
                />
                <Stack.Screen
                  name="PersonalEdit"
                  component={PersonalEdit}
                  options={{ headerShown: true, title: 'Edit'}}
                />
            </Stack.Navigator>
        </NavigationContainer>
    );
};
export default RootStack;
```

4. **BottomTabNavigator.js** - Вкладки включають головну сторінку, університети, питання та профіль. Вкладки мають свої власні компоненти та налаштування.

Лістинг 3.19 Навігація між сторінками

```
import { createBottomTabNavigator } from
'@react-navigation/bottom-tabs';
import React from 'react';
import { View, TextInput, StyleSheet, Text } from 'react-native';
import Icon from 'react-native-vector-icons/Ionicons';
import Main from '../screens/main/Main';
import Profile from '../screens/main/Profile';
import Question from '../screens/main/Question';
import Universities from '../screens/main/Universities';

const Tab = createBottomTabNavigator();

function CustomHeader() {
```

```
 return (
   <View style={styles.headerContainer}>
     <Text style={{color:'#5956E9'}}>
       Пошук
     </Text>
     <View style={styles.searchInput}>
       <Icon name="search-outline" size={25} color='#5956E9' />
       <TextInput placeholder="" style={styles.input} />
     </View>
     <Icon name="menu-outline" size={25} color="#5956E9" />
   </View>
 );
}

function BottomTabNavigator() {
 return (
   <Tab.Navigator
     screenOptions={({ route }) => ({
       tabBarIcon: ({ color, size }) => {
         let iconName;

         // Select an icon based on the route name
         if (route.name === 'Головна') {
           iconName = 'home-outline';
         } else if (route.name === 'ЗВО') {
           iconName = 'school-outline';
         } else if (route.name === 'Питання') {
           iconName = 'help-circle-outline';
         } else if (route.name === 'Профіль') {
           iconName = 'person-outline';
         }

         return <Icon name={iconName} size={size} color={color} />;
       },
```

```
        tabBarActiveTintColor: '#5956e9',
        tabBarInactiveTintColor: 'gray',
      })}
    >
      {/* CustomHeader shown on Main, Universities, and Question */}
      <Tab.Screen
        name="Головна"
        component={Main}
        options={{ header: () => <CustomHeader /> }}
      />
      <Tab.Screen
        name="ЗВО"
        component={Universities}
        options={{ header: () => <CustomHeader /> }}
      />
      <Tab.Screen
        name="Питання"
        component={Question}
        options={{ header: () => <CustomHeader /> }}
      />
      {/* No header on Profile */}
      <Tab.Screen
        name="Профіль"
        component={Profile}
        options={{ headerShown: false }}
      />
    </Tab.Navigator>
  );
}

const styles = StyleSheet.create({
  headerContainer: {
    flexDirection: 'row',
    alignItems: 'center',
```

```
    justifyContent: 'space-between',
    paddingHorizontal: 15,
    paddingVertical: 10,
    backgroundColor: 'white',
  },
  searchInput: {
    flex: 1,
    marginLeft: 10,
    marginRight: 10,
    flexDirection: 'row',
    alignItems: 'center',
    paddingHorizontal: 10,
    paddingVertical: 5,
    borderRadius: 20,
    backgroundColor: '#F2F3F5',
  },
  input: {
    marginLeft: 10, // Add some space between the search icon and
the text input
    flex: 1, // Ensure the input takes the remaining space
  },
});

export default BottomTabNavigator;
```

5. **Profile -** Цей компонент відображає інформацію профілю користувача. Він отримує дані зі стану Redux та передає їх дочірньому компоненту `ProfileHeader`, який відповідає за рендеринг голови профілю.

Лістинг 3.20 Profile компонент

```
import React from 'react';
import { View } from 'react-native';
import { useSelector } from 'react-redux';
import ProfileHeader from '../../components/ProfileHeader';
import { useNavigation } from '@react-navigation/native';

const Profile = () => {
 const user = useSelector((state) => state.user.user); // Отримання
даних користувача з Redux
 const navigation = useNavigation(); // Доступ до навігації

 return (
   <View>
     {/* Передача даних про користувача в ProfileHeader */}
     <ProfileHeader
       nickName={user?.nickName}
       status={user?.status}
       profilePicUrl={user?.profilePicUrl}
       navigation={navigation} // Передача навігації
     />
   </View>
 );
};

export default Profile;
```

6. **Question -** компонент створює екран для відображення питання. Він використовує `useState` для збереження тексту питання та `useEffect` для ініціалізації або запиту даних. Компонент також містить кнопку для переходу до наступного питання.

Лістинг 3.21 Question компонент

```
import React, { useState, useEffect } from 'react';
import { Button, View, Text } from 'react-native';

const Question = () => {
 const [question, setQuestion] = useState(''); // Стан для тексту
питання

 useEffect(() => {
   setQuestion('What is your favorite subject?');
 }, []);

 return (
   <View>
     <Text>{question}</Text>
     {/* Кнопка для дії "Next" */}
     <Button title="Next" onPress={() => alert('Next question!')}
/>
   </View>
 );
};

export default Question;
```

7. **UniversitySearch -** цей компонент дозволяє користувачам шукати університети за ключовими словами. Користувач вводить текст у пошукове поле, а список університетів фільтрується та відображається на основі введених даних.

Лістинг 3.22 UniversitySearch компонент

```
mport React, { useState } from 'react';
import { View, TextInput, Button, FlatList, Text, ActivityIndicator
} from 'react-native';
```

```
const UniversitySearch = () => {
 const [query, setQuery] = useState(''); // Стан для пошукового
запиту
 const [universities, setUniversities] = useState([]); // Стан для
списку університетів
 const [loading, setLoading] = useState(false); // Стан для
індикатора завантаження
 const [error, setError] = useState(null); // Стан для обробки
помилок

 const fetchUniversities = async () => {
   setLoading(true); // Початок завантаження
   setError(null); // Очистити попередні помилки

   try {
     const response = await
fetch(`http://localhost:3000/universities/${query}`);
     if (!response.ok) {
       throw new Error('Failed to fetch universities');
     }
     const data = await response.json();
     setUniversities(data); // Припускається, що ваш API повертає
масив університетів
   } catch (err) {
     setError(err.message);
   } finally {
     setLoading(false); // Завершення завантаження
   }
 };

 return (
   <View>
     <TextInput
       placeholder="Search for universities"
```

```
        value={query}
        onChangeText={setQuery} // Оновлення стану запиту
        style={{ borderWidth: 1, padding: 10, marginBottom: 10 }}
      />
      <Button title="Search" onPress={fetchUniversities} />

      {loading && <ActivityIndicator size="large" color="#0000ff"
/>}
      {error && <Text style={{ color: 'red' }}>{error}</Text>}

      <FlatList
        data={universities}
        renderItem={({ item }) => <Text>{item.name}</Text>}
        keyExtractor={item => item.id.toString()}
      />
    </View>
  );
};

export default UniversitySearch;
```

Висновки до Розділу 3

У третьому розділі було детально описано процес розробки функціоналу серверної та клієнтської частин веб-застосунку. У результаті виконаних робіт було досягнуто наступного:

1. Обґрунтовано вибір технологій та засобів розробки:

   Визначені інструменти та програмне забезпечення, такі як Node.js, Express.js, MongoDB, React, а також додаткові бібліотеки (bcrypt.js, Formik), забезпечили реалізацію сучасного, безпечного та інтерактивного застосунку. Вибір цих технологій був зумовлений їхньою продуктивністю, легкістю інтеграції та відповідністю задачам проєкту.

2. Реалізовано серверну частину веб-додатку:
    - Створено API з використанням Express.js, що забезпечує ефективну обробку запитів.
    - Реалізовано функціонал для роботи з базою даних MongoDB через Mongoose, включно з операціями реєстрації користувачів, автентифікації, зберігання та обробки даних.
    - Впроваджено систему безпеки для аутентифікації та bcrypt.js для захисту паролів.
3. Реалізовано клієнтську частину:
    - Розроблено інтерактивний інтерфейс на основі React, який забезпечує зручну взаємодію користувача із застосунком.
    - Використання бібліотек Formik та Yup дозволило ефективно реалізувати форми та забезпечити їх валідацію.
    - Реалізовано маршрутизацію з допомогою React Router, що сприяло організації SPA-структури веб-додатку.
4. Інтегровано системи тестування та прототипування:
    - Використання Postman забезпечило тестування та оптимізацію роботи API.
    - За допомогою Figma розроблено інтуїтивно зрозумілий дизайн інтерфейсу.
5. Застосовано інструменти спільної роботи та контролю версій:
    - Завдяки Git та GitHub вдалося організувати чіткий процес командної розробки із збереженням цілісності проєкту.

Розробка функціоналу веб-застосунку відповідає поставленим завданням і демонструє інтеграцію сучасних технологій для створення масштабованого та зручного рішення. Отримані результати створюють основу для подальшого розвитку додатку, включно з оптимізацією продуктивності, розширенням функціоналу та впровадженням нових інструментів.

ВИСНОВКИ

Метою даної дипломної роботи було розробити веб-застосунок UniSearch, який надає користувачам зручний інструмент для пошуку університетів, фільтрації за ключовими критеріями, комунікації з іншими користувачами, а також залишення коментарів і обговорення актуальних питань. Цей продукт призначений для спрощення вибору закладу вищої освіти та створення спільноти, яка сприяє обміну знаннями й досвідом. Завдяки інтеграції всіх функціональних компонентів веб-додатку, його кінцевий вигляд повністю відповідає поставленим цілям і задачам.

У процесі виконання роботи було реалізовано кілька ключових етапів, кожен з яких відігравав важливу роль у досягненні поставлених цілей. Спочатку було здійснено аналіз наявних рішень у сфері освітніх платформ, таких як UniRank, Study.ua та інших схожих сервісів. На основі цього аналізу вдалося виокремити потреби потенційних користувачів і сформулювати функціональні вимоги до майбутнього продукту. Основними аспектами, на яких було зосереджено увагу, стали зручність пошуку та фільтрації, інтерактивність взаємодії користувачів і можливість залишати та переглядати відгуки.

Наступним етапом стало формулювання завдань, які необхідно було виконати для досягнення мети проєкту. Було визначено такі основні задачі:

1. Реалізація системи реєстрації та автентифікації користувачів із підтримкою ролей (студенти, абітурієнти, випускники).
2. Створення бази даних університетів із можливістю зручного пошуку, фільтрації за параметрами (країна, спеціальність, рейтинг, вартість навчання тощо) та перегляду детальної інформації про заклади вищої освіти.
3. Інтеграція системи коментарів і питань, що дозволяє користувачам ділитися досвідом і отримувати відповіді на важливі для них питання.
4. Розробка гнучкого API, який забезпечує взаємодію між серверною та клієнтською частинами застосунку.

5. Розробка клієнської частини.

Було розроблено архітектуру застосунку, яка враховує можливості масштабування та забезпечує стабільну роботу системи. Для цього було обрано клієнт-серверну модель, що дозволяє розділити відповідальності між фронтендом і бекендом. Серверна частина була розроблена на базі Node.js з використанням Express.js для створення RESTful API. В якості бази даних було використано MongoDB, що забезпечує ефективне зберігання структурованих даних.

Проєктування бази даних включало детальну нормалізацію структури, що дозволило уникнути дублювання даних і забезпечити зручний доступ до інформації. Усі взаємодії з базою даних були оптимізовані за допомогою бібліотеки Mongoose.

Клієнтська частина була розроблена з використанням React Native. Цей фреймворк дозволив створити динамічний і зручний інтерфейс користувача. Застосування React Router забезпечило багатосторінкову структуру додатку, а бібліотеки Formik і Yup – зручну валідацію форм.

Для забезпечення зручного користувацького досвіду було впроваджено функціонал пагінації та динамічного завантаження даних, що дозволяє оптимізувати роботу з великими обсягами інформації. Крім цього, фільтри пошуку були адаптовані до потреб користувачів, щоб забезпечити швидкий доступ до потрібної інформації.

Окрему увагу було приділено тестуванню системи. За допомогою Postman було перевірено всі API-запити на коректність роботи, а Swagger забезпечив зручну документацію для серверної частини. Усі функціональні модулі були протестовані на відповідність технічним і функціональним вимогам.

Таким чином, результати роботи повністю відповідають поставленій меті. Веб-додаток UniSearch забезпечує зручний пошук закладів вищої освіти, можливість взаємодії між користувачами та спрощує процес прийняття рішень щодо вибору університету. Цей продукт може слугувати прикладом сучасного підходу до розробки веб-застосунків та бути основою для подальшого розвитку функціоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт Study.ua. URL: https://study.ua/ (дата звернення: 01.09.2024 р.).
2. UniRank – світовий рейтинг університетів. URL: https://www.4icu.org/ (дата звернення: 01.09.2024 р.).
3. QS World University Rankings 2024. URL: https://www.topuniversities.com/ (дата звернення: 01.09.2024 р.).
4. React.js – офіційна документація. URL: https://reactjs.org/ (дата звернення: 20.09.2024 р.).
5. Документація Node.js. URL: https://nodejs.org/ (дата звернення: 01.10.2024 р.).
6. Express.js – фреймворк для Node.js. URL: https://expressjs.com/ (дата звернення: 05.10.2024 р.).
7. Redux Toolkit – офіційний сайт. URL: https://redux-toolkit.js.org/ (дата звернення: 10.10.2024 р.).
8. GitHub – хостинг для контролю версій. URL: https://github.com/ (дата звернення: 20.10.2024 р.).
9. Документація MongoDB. URL: https://www.mongodb.com/docs/ (дата звернення: 25.10.2024 р.).
10. Visual Studio Code – редактор коду. URL: https://code.visualstudio.com/ (дата звернення: 01.11.2024 р.).
11. Figma – інструмент для дизайну UI/UX. URL: https://www.figma.com/ (дата звернення: 05.11.2024 р.).
12. Офіційна документація HTML5. URL: https://html.spec.whatwg.org/ (дата звернення: 20.09.2024 р.).
13. Bootstrap – CSS-фреймворк. URL: https://getbootstrap.com/ (дата звернення: 25.09.2024 р.).
14. Офіційний сайт Sass. URL: https://sass-lang.com/ (дата звернення: 01.10.2024 р.).

15. CSS – мова стилізації вебсторінок. URL: https://www.w3.org/Style/CSS/Overview.en.html (дата звернення: 18.10.2024 р.).
16. Google Docs – хмарний інструмент для створення документів. URL: https://www.google.com/docs/about/ (дата звернення: 20.10.2024 р.).
17. Git – система контролю версій. URL: https://git-scm.com/ (дата звернення: 22.10.2024 р.).
18. JavaScript (JS) – мова програмування для веброзробки. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 25.10.2024 р.).
19. Postman – інструмент для тестування API. URL: https://www.postman.com/ (дата звернення: 28.10.2024 р.).
20. Nodemon – утиліта для розробки Node.js. URL: https://nodemon.io/ (дата звернення: 02.11.2024 р.).
21. npm – менеджер пакетів для JavaScript. URL: https://www.npmjs.com/ (дата звернення: 04.11.2024 р.).
22. Wang, X., & Zhao, Y. (2020). *Еволюція систем рейтингування університетів: впливи та тенденції*. Journal of Higher Education Policy, 45(3), 123-140.
23. Smith, J., & Brown, K. (2019). *Цифрова трансформація в освіті: роль вебзастосунків*. Educational Technology Research and Development, 67(2), 89-104.
24. Bass, L. (2015). *Архітектура програмного забезпечення на практиці*. Addison-Wesley Professional.
25. Freeman, E., & Robson, E. (2021). *Програмування на JavaScript: підхід Head First*. O'Reilly Media.
26. OECD (2023). *Освіта у фокусі: показники OECD*. Париж: OECD Publishing.
27. World Economic Forum (2024). *Майбутнє роботи: вплив технологій на освіту*.
28. UNESCO (2023). *Глобальний звіт з моніторингу освіти: 2023*. URL: https://en.unesco.org/gem-report.

29. ResearchGate – статті, які стосуються вебзастосунків і систем рейтингування університетів.