Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«29»травня 2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем» спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Система моніторингу аналітичних даних навчального закладу»

Виконав (-ла):
слухач (-ка) IV курсу, групи ЗПІ-зп01
Корнієнко Микита Валерійович _____

Керівник:
ст. викладач кафедри ІСТ,
Корнага В.І. _____

Рецензент:
зав. кафедри, д.т.н., проф.
Мухін Вадим Євгенович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Слухач (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр Ролік

«15» травня 2023 р.

ЗАВДАННЯ

На дипломний проєкт студенту

Корнієнко Микиті Валерійовичу

1. Тема роботи «Система моніторингу аналітичних даних навчального закладу», керівник проекту доц., к.т.н., с.н.с. Корнага В.І., затверджені наказом по університету від 22.05.2023 № 1873-с.
2. Термін подання студентом роботи 13.06.2023
3. Вихідні дані по роботи: наукова література з теми, існуючі реалізації системи на різних платформах, технічне завдання на розробку комп'ютерної програми для створення програми моніторингу аналітичних даних навчального закладу.
4. Зміст пояснювальної записки:
   Перелік скорочень
   Вступ

1) Опис предметної області
2) Методичне та інструментальне забезпечення
3) Реалізація системи
4) Висновки

5. Графічний матеріал, а саме: Схема архітектури застосунку (А3); Схема бази даних (А3); Діаграма діяльності (А3); Діаграма класів (А3); Процес роботи із застосунком (А3); Схеми графіків статистичних даних застосунку (А3).
6. Дата видачі завдання «01.03.2023».

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Обрання теми дипломного проекту	15.10.2022	
2	Збір інформації про предметну область	11.01.2023	
3	Аналіз наявної проблеми	13.02.2023	
4	Огляд наявних рішень	23.02.2023	
5	Огляд методів розробки веб-застосунків	15.03.2023	
6	Розробка алгоритму роботи рішення	28.03.2023	
7	Проектування архітектури	13.04.2023	
8	Реалізація спроєктованого рішення	24.04.2023	
9	Тестування створеного рішення	04.05.2023	
10	Оформлення дипломної роботи	11.05.2023	
11	Попередній захист	29.05.2023	

Слухач Микита Корнієнко

Керівник роботи Василь Корнага

АНОТАЦІЯ

Зростаюча кількість даних, що генерується навчальним закладом, ставить перед його адміністрацією виклики щодо ефективного управління цими даними та отримання висновків, щодо них.

Для полегшення роботи з персональними даними, візуалізації багатьох статистичних показників, що їх генерує будь який навчальний заклад я і розробив свій проєкт – електронну систему моніторингу.

Із статистичних показників, що поступатимуть кожен навчальний період, таких як оцінки, персональні дані студентів, фінансова звітність, облік студентів іноземців, тощо – мій проєкт дозволяє побудувати корисну візуалізацію у вигляді зведених таблиць, графіків, ріе-чартів, діаграм.

Ці дані будуть корисними абітурієнтам, батькам студентів, професорам, керівництву навчального закладу, ну і звісно ж – самим студентам, які зможуть краще відслідковувати свою успішність, рейтинг та свою фінансову звітність, щодо навчального закладу.

Додаток може бути використаний для аналізу роботи будь-якого навчального закладу, однак нашим тестовим навчальним закладом буде коледж, де вивчаються переважно технічні дисципліни, а також бухгалтерія.

Ключові слова: Аналітичні дані, статистика, візуалізація, агрегація даних, MongoDB, React, Express.js, Node.js, Redux, Compass.

Розмір пояснювальної записки – 58 аркуш, містить 24 ілюстрації, 4 додатки.

SUMMARY

The increasing amount of data generated by educational institutions presents challenges for their administration in terms of effectively managing and deriving insights from this data. To facilitate the handling of personal data and visualize various statistical indicators generated by any educational institution, I have developed an electronic monitoring system as my project.

By incorporating statistical indicators received each academic period, such as grades, students' personal data, financial reports, international student records, and more, my project allows for the construction of useful visualizations in the form of consolidated tables, charts, pie charts, and diagrams.

These data will be valuable to prospective students, parents, professors, institution management, and, of course, the students themselves, enabling them to better track their performance, ranking, and financial reports related to the educational institution.

While the application can be used to analyze the operations of any educational institution, our testing institution will be a technical college with a focus on technical disciplines and accounting.

Keywords: Analytical data, statistics, visualization, data aggregation, MongoDB, React, Express.js, Node.js, Redux, Compass.

The size of the explanatory note is 58 pages, including 24 illustrations, and 4 appendices.

Пояснювальна записка

до дипломної роботи

на тему: «Система моніторингу аналітичних даних навчального закладу»

Київ – 2023 року

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

UI - Інтерфейс користувача (User Interface)

UX - Взаємодія користувача (User Experience)

API - Інтерфейс програмного забезпечення (Application Programming Interface)

CRUD - Операції створення, читання, оновлення та видалення (Create, Read, Update, Delete)

JWT - Токен аутентифікації JSON (JSON Web Token)

DB - База даних (Database)

ORM - Об'єктно-реляційне відображення (Object-Relational Mapping)

REST - Представлення стану передачі (Representational State Transfer)

SPA - Односторінкова програма (Single-Page Application)

CLI - Інтерфейс командного рядка (Command Line Interface)

NPM - Менеджер пакетів Node.js (Node Package Manager)

JSX - Розширена синтаксис JavaScript (JavaScript XML)

SEO – Оптимізація у пошуку

ВСТУП

В останні роки у всьому світі сфера освіти змінюється – все більшої ваги набуває онлайн навчання. Причиною цього є розвиток засобів комунікації – все більше поширюється швидкісний інтернет, який стає доступним навіть у віддалених куточках світу. Також онлайн-освіта дозволяє ознайомитись з певним навчальним курсом без відвідування, що робить її набагато більш доступною, ніж класична очна освіта, однак такий вид освіти має і свої неочікувані недоліки, а саме: контакт з викладачем у реальному житті, відвідування лекції чи семінару безпосередньо, а не віддалено, дають можливість студенту краще засвоїти навчальний матеріал, менше відволікатися, більше зосередитись на предметі вивчення.

Крім того, класичні навчальні заклади будь-якого рівня акредитації дозволяють студенту зустріти однодумців, знайти людей зі спільними захопленнями, конкурувати за рейтинг, а також – краще проводити час.

Саме тому, навчальні заклади є ключовими установами, що забезпечують освіту та розвиток суспільства. Ми можемо із впевненістю стверджувати, що вони з нами надовго, і, що навчання у них завжди буде привілеєм.

Якість навчального процесу та ефективність роботи навчальних закладів є питаннями, які стоять перед освітніми системами та органами управління освітою. Оцінка роботи навчальних закладів та аналіз статистичних даних про їхню діяльність є важливими інструментами для забезпечення якості освіти та прийняття обґрунтованих управлінських рішень.

Ця дипломна робота присвячена розробці "Системи статистичної оцінки роботи навчального закладу", та має на меті створити зручний та ефективний інструмент для збору, аналізу та візуалізації статистичних даних, пов'язаних з роботою навчального закладу.

Основною метою цієї системи є забезпечення доступу до об'єктивних та надійних даних про різні аспекти роботи навчального закладу, таких як академічні показники, фінансові дані, аналіз успішності учнів, відвідуваність, статистика

викладачів та інші ключові показники. Ця система дозволить адміністраторам, вчителям, батькам та іншим зацікавленим сторонам отримувати об'єктивну інформацію та здійснювати аналіз даних для покращення якості освіти.

Для реалізації даної системи будуть використані сучасні технології, зокрема фреймворк React для розробки користувацького інтерфейсу, платформа Node.js для створення серверної частини, а також нереляційна база даних Mongo.db для зберігання та управління великим обсягом статистичних даних.

У даній дипломній роботі будуть розглянуті основні принципи проектування та реалізації системи статистичної оцінки роботи навчального закладу. Буде проведено аналіз вимог до системи, розроблено архітектуру, реалізовано необхідні функціональні модулі та проведено тестування системи на практиці.

Остаточним результатом цієї дипломної роботи буде працездатна та ефективна система статистичної оцінки роботи навчального закладу, яка допоможе удосконалити процес прийняття управлінських рішень, забезпечить об'єктивний аналіз та покращення якості освіти.

Цей проект має значний потенціал для використання в різних типах навчальних закладів, починаючи від шкіл і до університетів. Він дозволить забезпечити об'єктивну оцінку роботи навчального закладу, зробити аналітичні висновки та приймати обґрунтовані рішення, спрямовані на покращення якості навчання та досягнення найкращих результатів у педагогічній сфері.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний огляд проблеми

У сучасному освітньому середовищі, ефективне управління навчальними закладами та забезпечення якості освіти стають вирішальними завданнями. Адміністрація навчальних закладів, вчителі, батьки та студенти потребують доступу до достовірної та об'єктивної інформації щодо роботи закладу для прийняття обґрунтованих управлінських рішень та покращення якості навчання.

Також студентам могло б бути цікаво відслідковувати свої оцінки то оцінки своїх товаришів онлайн, що, без наявності спеціальної системи, буде неможливо.

Система оцінок має показувати загальний прогрес наших учнів, динаміку їх успішності, різницю в успішності по факультетам, за статтю, за курсом і за країною.

Крім того, батьки студентів та самі студенти часто хочуть висловити свою думку, стосовно того чи іншого питання. Я ставлю собі за мету реалізувати таку можливість у моєму додатку, за допомогою системи опитування, що доступна кожному зареєстрованому користувачу.

Невід’ємною частиною будь-якого навчального закладу є фінансова звітність. Не дивлячись на те, що більша її частина є, авжеж, конфіденційною інформацією, в інтересах відкритості та прозорості, навчальний заклад залишає за собою право ознайомити всіх зацікавлених осіб у деяких аспектах своєї фінансової діяльності, для чого у нашій системі є загальнодоступна фінансова інформація, реалізована у вигляді графіків.

У багатьох навчальних закладах є учні з інших країн світу. Так, до нас часто приїздять вчитись студенти з Африки, Близького Сходу, країн колишнього СРСР. Наші студенти, в свою чергу, нерідко вступають до навчальних закладів країн Східної та Центральної Європи, Німеччини, Франції і США.

Для контролю за кількістю іноземців, їх фінансовими транзакціями та країнами походження я маю намір у даній роботі створити інтерактивну мапу, де буде видно, з якої країни до нашого навчального закладу приїхала певна кількість людей.

Було б також непогано, щоб можна було побачити, які саме країни є, скажімо, найкращими фінансовими партнерами нашого навчального закладу. Для цього у нас в застосунку буде також специфічний графік.

Таким чином, у даному застосунку ми проаналізуємо велику кількість статистичних показників, які є характерними для будь-якого навчального закладу.

Однією з основних проблем, які вирішує цей проект, є складність збору, обробки та аналізу великого обсягу даних, пов'язаних з роботою навчального закладу. Такі дані можуть включати інформацію про відвідуваність, академічні досягнення учнів, успішність викладачів, витрати та бюджет закладу та інші показники. Ці дані можуть бути розподілені у різних системах та форматах, що робить їхнє зіставлення та аналіз важким завданням.

Архітектура системи "система статистичної оцінки роботи навчального закладу" може бути описана за допомогою шарової або клієнт-серверної моделі.

Шарова архітектура:

Користувацький інтерфейс: Верхній шар системи, що забезпечує взаємодію користувачів з системою. Включає в себе веб-інтерфейс, за допомогою якого користувачі можуть переглядати статистику, вводити дані, отримувати звіти.

Бізнес-логіка: Шар, що містить логіку обробки та аналізу даних. Включає в себе компоненти для обробки даних про студентів, викладачів, курси, оцінки та інші важливі аспекти навчального закладу. Забезпечує розрахунок статистичних показників, генерацію звітів та інформаційних матеріалів.

Доступ до даних: Шар, що відповідає за збереження та доступ до даних. Включає в себе базу даних (наприклад, MongoDB або SQL базу даних) та модулі для комунікації з нею. Забезпечує збереження та отримання інформації про студентів, курси, оцінки, викладачів тощо.

1.2 Огляд наявних рішень застосунків для аналізу статистичних даних навчальних закладів

Нумерація таблиць в межах розділу. Підсумковий аналіз наведено у Існує досить багато рішень для навчальних закладів, що дозволяють демонструвати та аналізувати показники їх роботи.

Переважно такі рішення є платними та робляться під замовлення навчального закладу.

Демонстрація статистичних даних є у таких системах, зазвичай, лише частиною функціоналу. У них є також, як правило, є функціонал дистанційного навчання, кабінет студента із завданнями, чат із вилкадачем чи адвайзером. У своїй роботі я не ставлю собі таких цілей і фокусуюсь переважно на статистичних даних, їх зборі та демонстрації.

Статистичні дані у інших застосунках показуються, але я ставлю собі за мету, дослідити якомога більше їх різновидів і продемонструвати якнайбільшу кількість різних показників. Але розглянемо і інші застосунки із подібними можливостями.

Цілі огляду наявних рішень:

– Вивчення ринку: Аналіз аналогів дозволить нам ознайомитися з наявними рішеннями на ринку. Ми можемо дізнатися про існуючі продукти, їх функціональність, особливості та популярність серед користувачів

– Виявлення сильних і слабких сторін: Аналіз аналогів допоможе виявити сильні і слабкі сторони існуючих програм. Ми зможемо проаналізувати їхні функції, інтерфейс, продуктивність, надійність та інші параметри;

– Вдосконалення нашого продукту: Огляд аналогів може дати нам ідеї для вдосконалення нашого продукту. Ми також зможемо виявити недоліки існуючих рішеннях і знайти способи їх усунути;

– Визначення унікальних функцій: Аналізуючи аналогічні програми, ми зможемо визначити, які функції вже існують на ринку і є стандартними, а які можуть бути унікальними нашого вашого рішення, так звані «killer features».

1.2.1 Застосунок від PowerSchool

PowerSchool – це освітня технологічна платформа, власником якої є PowerSchool Group LLC.

Ця платформа є однією із перших подібних платформ в історії, вона написана на мові Java у 1999 році та має останній реліз у 2017 році.

У даному застосунку широко реалізовано отримання та аналіз статистичної інформації, переважно, по успішності учнів. У своїй роботі вони спираються на Microsoft Power BI.

Аналізу у цьому застосунку підлягає широкий спектр даних щодо успішності студентів (рисунок 1.2).

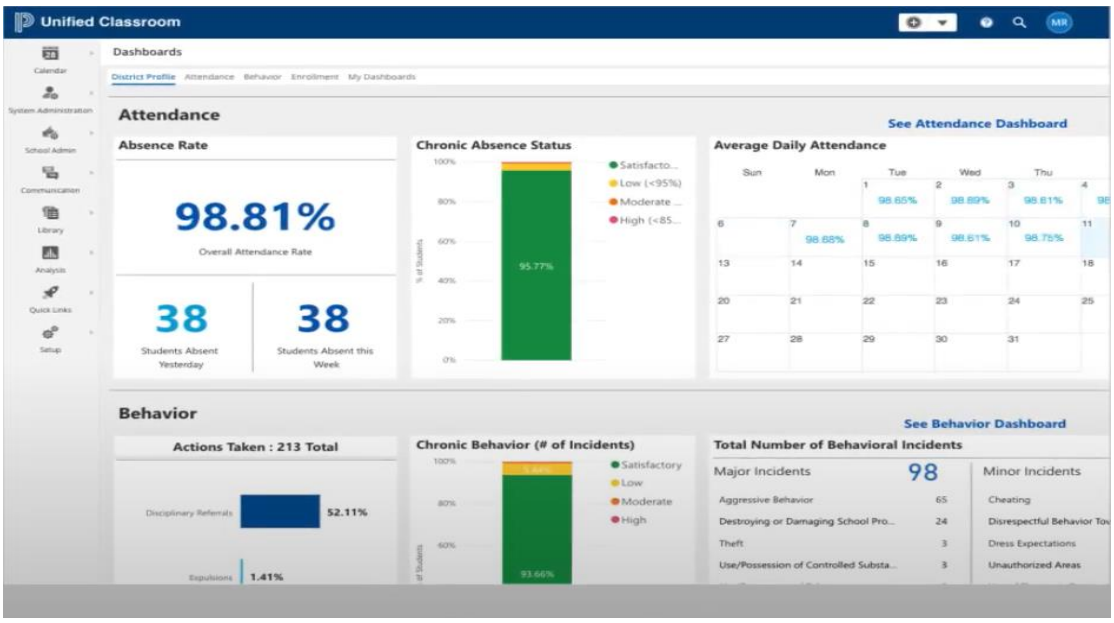

Рисунок 1.2 – PowerSchool. Інтерфейс учня

Недоліками даного застосунку є певна архаїчність технологій, відсутність стабільного оновлення, висока ціна, присутність тільки на освітньому ринку США.

1.2.2 Застосунок Jenzabar

Ця система спрямована на навчальні заклади різних рівнів та надає функціонал для збору, аналізу та відображення статистичних даних.

Jenzabar Analytics має в собі три основних компоненти: Program Insights Model, Financial Health Model, і Data Cloud.

The Program Insights Model – дає інформацію про успішність студента, його фінансовий стан, також про його здоров'я.

The Financial Health Model – робить можливим дізнатись все про фінансову складову роботи навчального закладу, зокрема розглядаючи такі показники, як Composite Financial Index, net operating revenue, та інші індикатори (рисунок 1.3).

The Data Cloud – збирає, агрегує та зберігає інші дані навчального закладу.

Рисунок 1.3 – Jenzabar. The Financial Health Model

1.2.3 Застосунок Ellucian Banner

Ellucian Banner – освітня система, що дозволяє покращити роботу будь-якого навчального закладу.

Вона є комплекною і включає в себе широкий набір різноманітних функцій управління навчальним закладом і є корисною як студентам, так і керівникам.

Студентам вона дозволяє отримати велику кількість даних про свій навчальний прогрес, керівникам – про фінансову частину і роботу персоналу.

Використовується в основному – в США, розповсюджується по моделі підписки для навчальних закладів, є платною (рисунок 1.4).

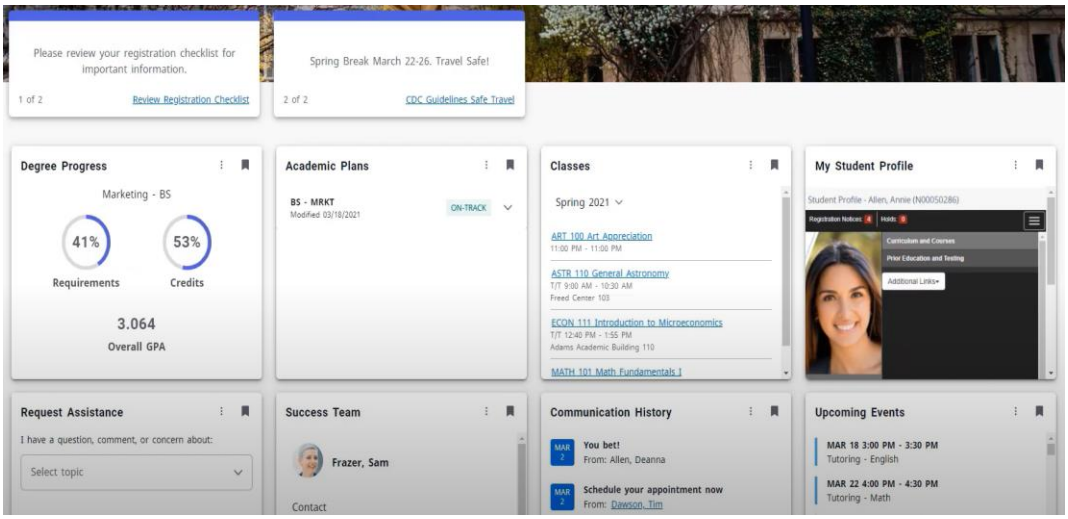

Рисунок. 1.4 – Ellucian Banner. Особистий кабінет студента

Має веб-інтерфейс, та мобільний застосунок.

Недоліки має ті ж – закритість моделі розповсюдження, висока ціна.

1.2.3 Застосунок HeRo Study Space

HeRo Study розробляє SaaS рішення для цифрової трансформації ВНЗ з фокусом на автоматизацію ключових бізнес-процесів.

Компанія працює у сфері EdTech з 2016 року, коли її засновник Антон Сіленко, разом із невеликою командою розробників створив застосунок Hero Study Space.

Цей застосунок є комплексною освітньою програмою, де представлено багато модулів для комунікації між студентом та навчальним закладом, є блок домашніх завдань, блок комунікації з власним едвайзером та інше (рисунок 1.5).

Статистичні дані доступні як для студентів, так і для керівників навчального закладу. Переважно компанія співпрацює з ВНЗ Казахстану, України та інших країн колишнього СРСР.

Тривалий час я і сам був Front-end розробником у даній компанії.

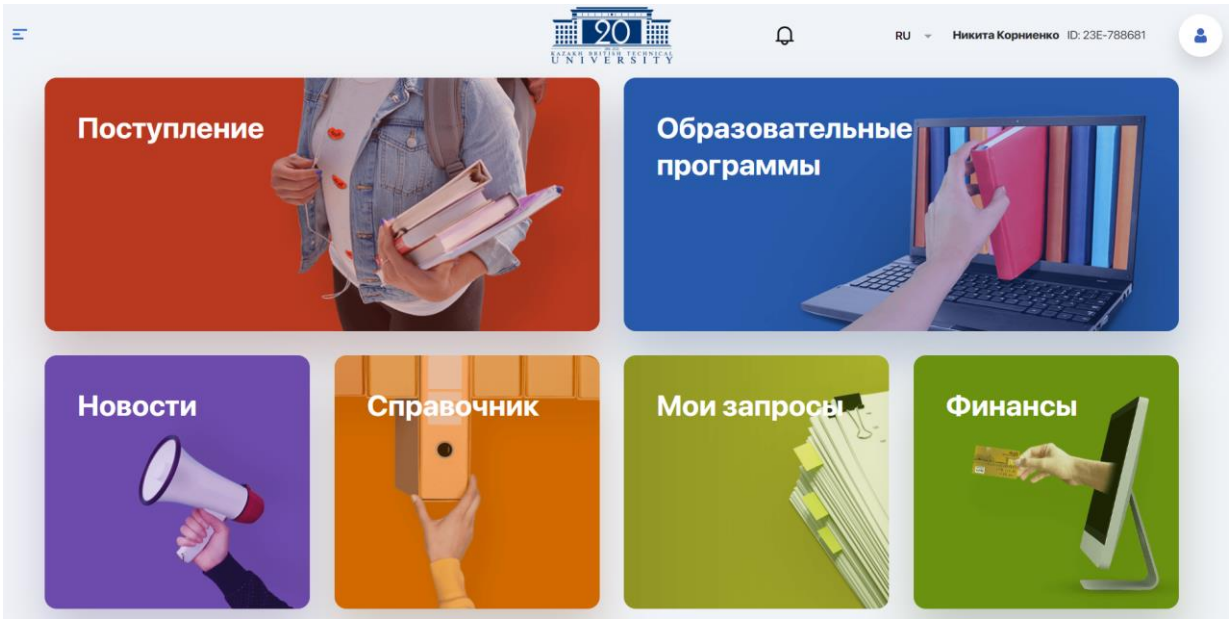

Рисунок 1.5 – Hero Study Space. Особистий кабінет абітурієнта

1.3 Постановка задачі

Після того, як я детально проаналізував зарубіжні та вітчизняні рішення, я вирішив розробити своє.

Мій застосунок я назвав «EduStatsTracker». Він також збиратиме достатньо великий обсяг даних, які генерує навчальний заклад та демонструватиме аналітику у вигляді простих, доступних графіків та чартів.

Вимоги до застосунку:

– Має надавати доступ до великого набору статистичних даних навчального закладу;

– Повинен містити свій кабінет, з авторизацією та різними рівнями доступу;

– Повинен бути mobile-responsive;

– Повинен бути універсальним;
– Мови інтерфейсу – Українська і Англійська.

Висновки до розділу

На сьогоднішній день існує ряд рішень, які використовуються для статистичної оцінки роботи навчальних закладів. Деякі з них базуються на традиційних підходах, в той час як інші використовують передові технології для автоматизації та покращення процесу збору та аналізу даних.

Я розглянув деякі наявні на ринку рішення застосунків, подібних до того, що я створив. Розглянув їх переваги та недоліки та визначив той напрям, у якому я буду рухатись для створення свого унікального рішення, що буде враховувати функціонал схожих продуктів та намагатиметься повторити та покращити його.

2. МЕТОДИЧНЕ ТА ІНСТРУМЕНТАЛЬНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Опис задач застосунку

Після огляду всіх подібних застосунків, було проаналізовано завдання, що стоятимуть при розробці застосунку «EduStatsTracker» і було визначено набір дій, що їх він буде виконувати:

– Авторизація і аутентикація; Власний кабінет;

– Перегляд різноманітної статистики всього університету;

– Перегляд власних успіхів та досягнень;

– Перегляд досягнень інших студентів;

– Перегляд фінансової звітності;

– Перегляд інформації по іноземним студентам та країнам, з яких вони прибули.

Оскільки такі завдання ставлять нас перед вибором певних технологій, які буде висвітлено у наступному розділі даної записки.

2.2 Сучасні технології веб-додатків

Оскільки я хочу зробити програму якомога доступнішою для широкого кола користувачів (абітурієнтів, студентів, персоналу, керівництва, батьків), я вирішив розробляти її у вигляді веб-застосунку, який завжди можна розширити на інші платформи (десктоп, мобільний додаток).

Так як ми не маємо у наших планах великої кількості взаємодій у реальному часі, що потребували б швидких, динамічних відповідей, а всі наші дії у застосунку будуть досить стандартизованими, ми не потребуємо нічого більше, крім стандартної REST-архітектури, яку я і вважаю оптимальною для побудови даного додатку.

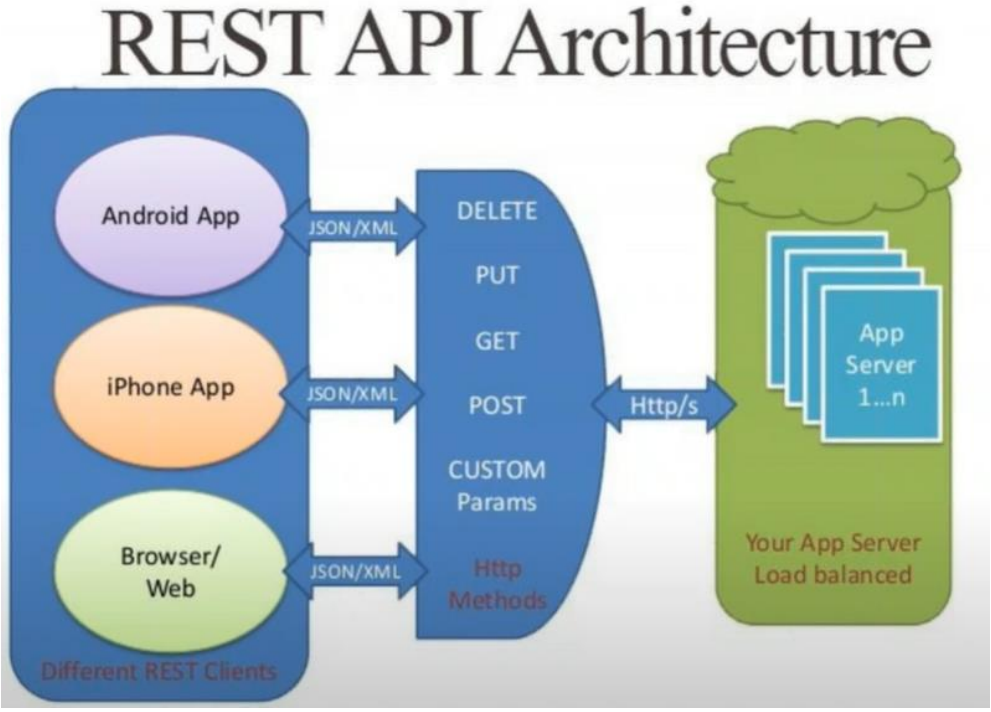

Рисунок 2.1 – Схема стандартної REST API

Наша система також матиме front-end, back-end і базу даних.

Front-end створений на фреймворку React. Back-end – монолітний і виконаний на Node.js із фреймворком Express. База даних – нереляційна Mongo.db. Причини вибору цих технологій – далі.

MongoDB надає розширений набір функцій, таких як текстовий пошук, геопросторові запити, агрегація даних та інші, що дозволяє забезпечити потрібну функціональність у нашому застосунку.

2.3 Сучасні підходи до рендерінгу веб-додатків

Рендерінг – процес перетворення даних та програмного коду у кінцевий html, що його буде бачити користувач.

Цей процес може відбуватись як на сервері, так і на стороні клієнта, або поєднувати певним чином ці варіанти.

Розглянемо спочатку класичну стратегію – Статичний вебсайт. Такі вебсайти були популярними з часів Web 1.0, але й зараз такі сайти мають місце у Web.

При роботі зі статичним веб-сайтом, кожна сторінка генерується на сервері заздалегідь і повертається користувачу для подальшої взаємодії.

На даний час існують фреймворки, що допомагають у створенні таких вебсайтів – наприклад Jekyll, Hugo та 11ty.

Такий тип рендерінгу не дуже придатний для веб-сайтів, де дані часто змінюються. Такі сайти не є дуже динамічними, кожен клік означає нове завантаження сторінки, що погіршує UX.

Multi-page rendering – стратегія рендерінгу, при якій html та дані об’єднуються на сервері в момент надходження нового запиту від користувача.

Такий спосіб використовується сьогодні у багатьох e-commerce проєктах, наприклад – на сайті компанії Amazon, де ми можемо побачити, як на кожен наш клік виводиться нова, динамічно-згенерована сторінка.

На сьогодні існує велика кількість фреймворків, що дають можливість створювати додатки – це і Ruby on Rails, і Django, і PHP-фреймворк Laravel.

Такий підхід був розповсюджений до появи Iphone, коли користувачі збагнули, що завантаження після кожного кліку не є оптимальним з точки зору UX.

Однак, слід зазначити, що перевагою даного підходу є досить високий рівень SEO.

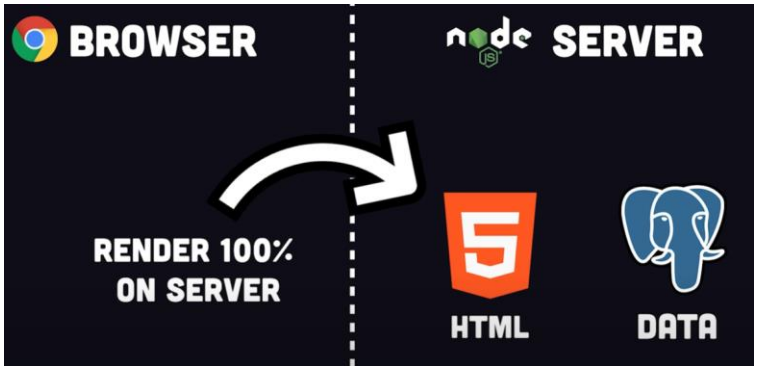

Рисунок 2.2 – Схема рендерінгу багатосторінкового застосунку

Односторінковий застосунок, або SPA(Single page application) – Роком народження даної парадигми рендерінгу вважається 2010, коли вийшла перша версія Angular.js, і, згодом, - у 2011 – React.js.

Сайти, чий рендерінг відбувається у цей спосіб, починають роботу з лише однієї html-сторінки – так званої «обгортки» і, далі, використовують мову Javascript для рендерінгу UI-елементів, а також – для HTTP-запитів.

Основні принципи SPA:

– Більш зручний користувацький досвід: SPA надає більш плавну та швидку навігацію, оскільки не потрібно завантажувати повну сторінку при кожному переході;

– Висока продуктивність: Оскільки лише необхідні дані та компоненти завантажуються асинхронно, це зменшує навантаження на сервер та забезпечує більш швидку відповідь додатку; – Легше розширення та підтримка: SPA дозволяє розробникам розширювати функціональність окремих компонентів без впливу на інші частини додатку. Це полегшує розвиток та підтримку проекту.

– Не дивлячись на те, що це лише один документ, у ньому можуть бути «маршрути», але вони не вказують на шлях до серверу – їх обробляє Javascript.

– У такий спосіб ми робимо перехід між «сторінками» нашого застосунку непомітним для користувача.

– Недоліками даного підходу є досить великий розмір сторінки «обгортки», що знижує швидкість першого завантаження. Також, пошукові двигуни поки що не навчились адекватному пошуку по таким «віртуальним» сторінкам, тому SEO є ускладненим і ці два недоліки знижують привабливість таких застосунків для комерційних проєктів (хоча, все ж, інтернет-магазинів, зроблених на базі SPA існує достатньо багато).

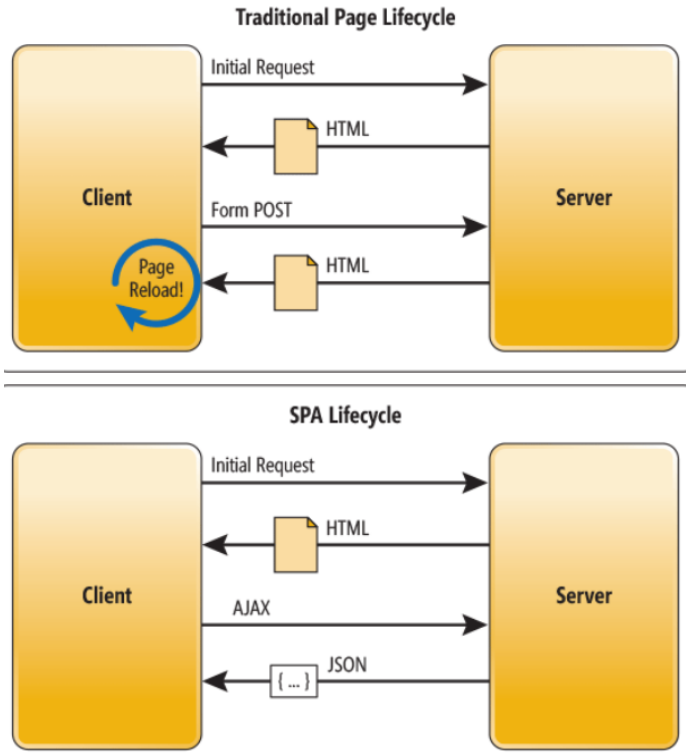

Рисунок 2.3 – Схема рендерінгу односторінкового застосунку

Рендерінг на стороні сервера із «зволожуванням» - «зволожування», або hydration – процес наповнення об'єкту або екземпляру класа – даними.

У зв’язку з веб-розробкою та рендерінгом, термін використовується для описання заповнення «обгортки» – функціоналом SPA.

У цій парадигмі – перший запуск застосунку, на відміну від SPA, не створює «обгортку» на рівні клієнта, а повертає її із сервера, а вже Javascript, що у ній наявний, заповнює її контентом на функціоналом.

Цей підхід є «кращим із двох світів», бо він запобігає повільному першому завантаженню, що є характерним для SPA, а також при такому підході, пошукові двигуни набагато краще знаходять подібні сторінки.

Цей підхід зараз є найбільш популярним і використовується сучасними «мета-фреймворками», такими як Nuxt.js, Next.js, SvelteKit та Angular Analog.

Варіацією минулого підходу є Статична генерація сайтів зі «зволоженням».

Така парадигма дозволяє створити всі необхідні сторінки сайту заздалегідь, помістити їх на сервер і віддавати користувачу за необхідністю, всі ці сторінки

будуть перетворені на Javascript і будуть змінюватись відповідно до дій користувача.

Такі вебсайти називаються JamStack. Їх створюють на тих самих метафреймворках, що і попередній тип, і, як і для минолого виду рендерінгу, недоліком є необхідність мати потужний сервер, тільки на цей раз – сервер має бути із великим запасом пам’яті, щоб вмістити сторінки для усіх юзерів.

Поступова статична регенерація – як і в попередніх видах рендерінгу сервер повертає первинну сторінку «обгортку», але на цей раз Javascript слідкує за завантаженнями на сторінці, їх швидкістю та часом, коли кеш стає не валідним. Якщо сайт завантажується занадто довго, або інвалідизується кеш – повертається нова сторінка-«обгортка».

Часткове «зволоження» – на відміну від усіх попередніх видів рендерінгу, включно з SPA, при цьому варіанті, ми проводимо наповнення сторінки даними не одразу, а частково – коли користувач робить якісь дії, наприклад – рухається по сторінці.

«Острови» – при цьому варіанті ми проводимо «зволоження» лише тих ділянок веб-сайту, з якими користувач активно взаємодіє, створючи таким чином «острови» гідратації. Це один із найефективніших видів рендерінгу, оскільки заповнення даними відбувається динамічно тільки тих компонентів застосунку, з якими взаємодіють тут і зараз. Таким чином, якщо ви створили статичну сторінку, в якій немає взаємодій, не буде відбуватись зайвої роботи Javascript, на відміну від SPA.

Фреймворк Astro працює таким чином і демонструє чудові показники швидкості роботи застосунків.

Потокова SSR – новий вид рендерінгу, що доступний у Next.js 13-ї версії.

«Зволоження» відбувається у вигляді інтерактивних потоків, що надходять постійно від серверу.

А чи можна не заповнювати сторінку при первинному завантаженні взагалі, тобто – обійтись без «зволоження»?

Так. Остання парадигма рендерінгу – продовжуваність (Resumability) – піонером у даному виді рендерінгу є фреймворк qwik. За його розробку відповідають розробники Angular.

При роботі цього фреймворку, все, що відбувається у веб-застосунку, серіалізується у вигляді HTML, а Javascript розбивається на велику кількість фрагментів. Тому перше завантаження – це завантаження порожньої статичної сторінки, а увесь javascript завантажується поступово у фоновому режимі.

Ми ознайомились з усіма наявними на даний час парадигмами рендерінгу контенту у веб. Я впевнений, що таких варіантів згодом буде ще більше, однак, обираючи зараз, я зупинюсь на SPA.

Такий вибір я роблю тому, що у даному застосунку час першого завантаження (так званий FCP – перше малювання контенту) не є дуже важливим і при наявних розмірах застосунку навряд чи змінитися в «червоний» бік.

Також, ми не ставимо собі на меті піднятись у перші щаблі пошуку Google. Якщо наш застосунок колись прийме вигляд промислового продукту він все одно буде лише для підписників (як і розглянуті раніше аналоги), тому ефективного SEO він не потребує.

Розроблятимемо Front-end на фреймворку React. Зберемо за домомогою Vite. React вибраний з-поміж інших фреймворків через швидкість розгортання, простий синтаксис, та особисті преференції.

2.4 Огляд бібліотеки React

Бібліотека, а потім і повноцінний фреймворк, розроблений компанією Facebook у 2013 році.

Його можна назвати найвпливовішою бібліотекою сучасності.

Основним принципом його роботи є поділ UI на обмежені логічно та функціонально компоненти. Кожен такий компонент являє собою ніщо інше, аніж Javascript-функцію.

Така функція має специфічне значення, що обов’язково повертається із return – а саме – JSX. Це об’єкт із специфічним синтаксисом, що візуально є

схожим на HTML, а також може містити певні функції та методи, що будуть обробляти події, що відбуватимуться з цим компонентом після рендерінгу.

Також, ключовими для даного фреймворку є поняття State та Hooks.

State – стан компоненту, який відслідковується екземпляром React, а зміни відбуваються через функції – хукі. При зміні стану є повторний рендерінг компоненту.

```
1  import React, { Component } from 'react';
2  import ems from './ems.png';
3  export class Instructions extends Component {
4    render() {
5      return(
6      <div>
7      <h1>Employee Mern Project</h1>
8      <p>This is an Employee Management System</p>
9      <img src={ems} alt="ems" />;
10       </div>
11     )
12   }
13 }
14 export default Instructions;
```

Рисунок 2.4 – Схема типового класового JSX-компоненту

На даній схемі зображений класовий React-компонент. Вони є більш застарілими, хоча і використовуються інколи у різних проектах, переважно для написання бібліотек.

Функціональні реакт-компоненти більш розповсюджені та легші для розуміння. Вони не використовують метод render, а просто повертають JSX.

Їх hooks, або хукі життєвого циклу – функції, що змінюють стан компоненту. Можуть бути, вбудованими в бібліотеці React, запозиченими із бібліотек, та кастомними – написаними користувачем.

Найпоширенішими є UseState та UseEffect. UseState – дозволяє змінювати стан компоненту, UseEffect – робити сторонні виклики.

2.5 Підходи до менеджмента стану застосунку. Redux.

Стан застосунку – набір значень змінних у різних компонентах застосунка, у одного користувача у конкретний момент часу.

При користуванні програмою користувач може виконувати багато різних дій, результати яких мають бути візуально відображені, або збережені у застосунку. Це і переключення різноманітних режимів, колорових тем, мови, відкриття різноманітних сайдбарів, модальних вікон, повідомлень, також це можуть бути дії на кшталт додавання в корзину товару або збереження їх у список базових покупок.

Всі ці дії як правило відбуваються у різних частинах застосунку і в різних же частинах може відображатись результат. Таким чином, слідкування за усіма змінними є досить складним та потребує передачі даних між різними гілками дерева компонентів.

Систему менеджменту стану, такі як Redux, Mobx, Zustand полегшують дану задачу.

Redux – бібліотека, яку я використав у своєму застосунку, слугує «single source of truth» – єдиним джерелом істини. Вона зберігає дані про те, чи вікритий у нас боковий блок – сайдбар, чи має користувач дані про логін, яку роль має користувач, та багато іншого.

Я використовую Redux разом із бібліотекою Redux-toolkit, яка зменшує кількість додаткового коду для ініціалізації стану і роботи з ним.

Як же працює Redux?

З точки зору Redux, у застосунку є один незмінний State. Будь яка дія, яку виконує користувач, створює новий state, а не змінює існуючий стан.

Стан поділється на «шари», або slices, відповідно до функції у застосунку.

Щоб дістатися до значення у слайсі використоввуються селектори – на які можна підписатись та відстежувати кожен новий стейт у різних частинах застосунку, що ніяк не пов'язані між собою, окрім як через Redux state.

Кожен раз, для отримання нового state, користувач створює дію – action, який redux реєструє, оцінює, що в собі action несе (чи є об’єкт payload) і, відповідно до типу дії – повертає новий state (рисунок 2.4).

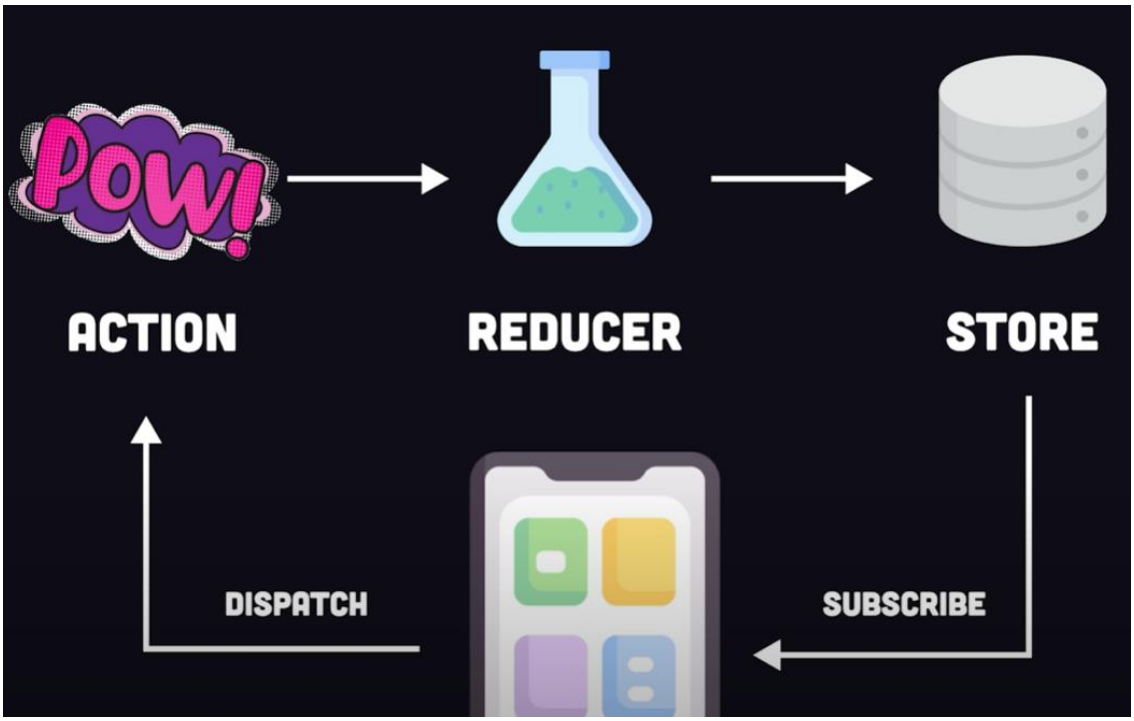

Рисунок 2.5 – Схема роботи бібліотеки Redux

2.6 Архітектура баз даних. Вибір бази даних для застосунку

Щоб зрозуміти, яку, із великої кількості баз даних, ми будемо використовувати у нашому проекті, нам необхідно знати їх різновиди.

Баз даних є велика кількість, та й навіть парадигм, за якими вони влаштовані, є близько десятку.

Найбільш розповсюдженими ж є SQL та NoSql бази даних.

Про них та про причини вибору саме тієї бази даних, що використана у засосунку – далі.

2.6.1 SQL, або Structured Query language

SQL – мова, розроблена для роботи з базами даних, що виникла на основі алгебри відношень, за участі Дональда Камбертіна та Реймонда Бойса у 1974 році.

У цій мові є досить великий набір команд для роботи з таблицями.

Додавання у таблицю, видалення, об’єднання таблиць, запит певної інформації із окремої таблиці чи із багатьох – це все можна робити за допомогою SQL-синтаксису.

Сама база даних підтримує відношення. Базуються вони на чітких інструкціях, які є частиною мови і для реалізації цих інструкцій нам потрібні схеми бази даних, або Schema.

SQL дозволяє створювати, змінювати та отримувати дані з баз даних, а також виконувати різноманітні операції з ними. Основна сила SQL полягає в його декларативному підході, що означає, що ви описуєте, які дані потрібно отримати або змінити, а не як це зробити. SQL визначає набір команд, за допомогою яких ви можете працювати з базами даних

SQL є мовою з великою кількістю реалізацій, включаючи такі популярні СУБД, як MySQL, PostgreSQL, Oracle Database та Microsoft SQL Server. Використання SQL дозволяє розробникам ефективно та зручно працювати з даними у реляційних базах даних та виконувати різноманітні операції для забезпечення ефективності та цілісності даних.

SQL базується на засадах реляційної моделі даних, де дані зберігаються у вигляді таблиць зі стовпцями (полями) та рядками (записами). SQL дозволяє створювати, змінювати та видаляти такі таблиці, а також встановлювати зв'язки між ними.

У Schema є чітко детерміновані поля – fields.

Кожен новий запис у таблицю буде являти собою рядок, що чітко вписується у дану схему – має всі поля і значення, або містить значення null.

Неможливо, щоб запис мав додаткові поля. Якщо один запис має поле – всі інші також матимуть його.

Наступним «будівельним каменем» бази даних такого типу є те, що ми зазвичай працюємо не з одною, а з багатьма таблицями, які є пов'язаними. Тобто – мають відношення між собою.

Відношення бувають різними. Якщо у нас одна таблиця зв'язана з іншою таблицею, то такий зв'язок називають зв'язком «один до одного» (рисунок 2.5).

Рисунок 2.5 – Відношення виду «один до одного»

Прикладом такого відношення може бути таблиця користувача, та таблиця його контактів. Одному користувачу належить лише одна таблиця контактів – і навпаки. Якщо нам треба зробити запит на контакти користувача – ми не будемо отримувати їх із таблиці самого користувача, ми їх об’єднаємо через відношення.

Сортування та групування: SQL дозволяє сортувати дані у таблицях за певними стовпцями з використанням команди ORDER BY. Також можна групувати дані за певними стовпцями з використанням команди GROUP BY.

Наступним видом зв’язків є зв’язок типу «один до батьох».

Такий зв’язок є характерним, коли ми маємо справу із однією таблицею, яка об’єднується із багатьма – наприклад, коли ми маємо одного клієнта, що робить багато замовлень (рисунок 2.6).

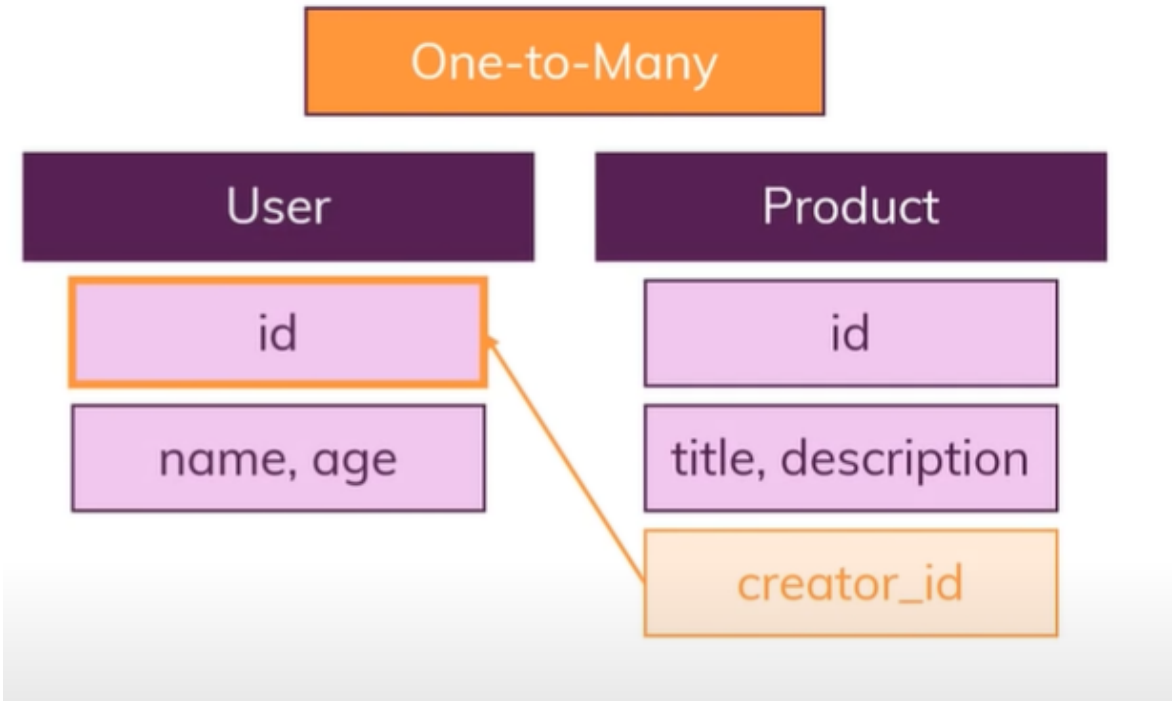

Рисунок 2.6 – Відношення виду «один до багатьох»

Id у даному випадку є первинним ключом (primary key), creator_id у таблиці Product – foreign key.

Через таку зв’язку ключів і відбувається взаємодія.

Останнім видом зв’язків є «many-to-many» відношення.

Такий вид зв’язку використовується, коли обидві таблиці можуть належати багатьом іншим таблицям.

Наприклад, автор може написати багато книг, а одна книга, в свою чергу, може бути написаною декількома авторами.

Виникає необхідність в створенні третьої таблиці author_book, яка буде поєднувати обидві попередні таблиці через ключі (рисунок 2.7).

Рисунок 2.7 – Відношення виду «багато до багатьох»

Таким чином, через відношення між таблицями, що називаються joins, можна об’єднувати декілька таблиць, при наявності чіткої схеми та ключів, які повторюються у декількох таблицях (primary – foreign key),

Таким чином, у SQL-світі, ми матимемо нормалізовані, систематизовані таблиці, які завжди будуть мати поля, по яким ми зможемо зробити запит. І саме така модель взаємодії робить SQL ідеальним для багатьох проектів.

2.6.2 NoSql бази даних. MongoDB

Існує декілька баз даних цього типу, одним із прикладів була б DynamoDb, але найпопулярнішою NoSql базою даних зараз можна із впевненістю назвати – MongoDb.

Назва цієї бази походить від англійського слова «humongous», тобто – величезний. Її так назвали тому, що вона мала вміщювати багато даних і ефективно обробляти їх.

Вона була створена у 2009 році на мові C++.

Як же вона працює?

У такій базі даних немає схем і таблиць – лише так звані «колекції».

В кожній колекції є документи. Кожен документ, що лежить у колекції має подібний до JSON формат, але, при цьому – немає жодної схеми, що її необхідно дотримуватись!

Рисунок 2.8 – Документи у базі даних без схеми

При роботі із NoSql базою даних, немає жодного контракту, якого ми маємо дотримуватись. Ми можемо покласти у одну колекцію абсолютно різні документи.

Такий підхід може бути не ідеальним, якщо ми працюємо з існуючими даними, але якщо у нас є більш розширений набір даних, то ми можемо і не хотіти врешті решт притримуватись схеми, якщо ми точно знаємо, що шукана інформація у колекції є.

Також у такій базі даних немає відношень. Ми, звісно, можемо їх встановити вручну, через такі самі пари ключів у різних таблицях, проте NoSql бази даних набагато менше покладаються на них у своїй роботі.

Ідея роботи таких баз даних в іншому – ми згуртовуємо всі дані разом, у великі колекції і робимо декілька рівнів вкладеності, отримуючи таким чином у кожній колекції – великі за розміром документи, де є вся нам потрібна інформація.

Недоліком є те, що якщо у нас відбувається оновлення інформації у одній колекції, у нас також буде можливим оновлення у багатьох інших.

Такий підхід як раз підходить для застосунків у яких є багато операцій читання із бази даних, і не так багато операцій запису, оновлення та оновлення зв'язаних документів або таблиць.

Переглялянемо ще раз переваги та недоліки двох найрозповсюдженіших підходів до побудови баз даних.

2.7 Переваги різних типів баз даних

Зразу можна сказати, що абсолютного переможця тут немає. Все залежить від тих завдань, які ставить перед собою розробник, та того проекту, який з базою даних пов'язаний. Часто також можуть використовуватись різні бази даних для різних потреб, якщо застосунок має мікросервісну архітектуру, та все ж, для чого краще підійде кожен тип баз даних?

Спочатку поговоримо про SQL.

Наявність у таких базах даних схеми може бути перевагою, а може і недоліком – в залежності від того, наскільки суворо типізованим є застосунок.

Якщо застосунок містить багато пов’язаних між собою даних, які часто змінюються, варто задуматись про використання SQL, так як зміни даних відбуваються в різних таблицях, що спрощує до них доступ та робить доступ більш швидким.

Також треба згадати про розширення (Scaling). Що це таке?

Розширення – це процес у якому змінюються потужності накопичуючих систем у бік збільшення.

Горизонтальне розширення – додавання нових серверів.

Вертикальне розширення – збільшення потужності існуючого серверу (накопичувача).

Якщо вертикальне розширення можливе для SQL, горизонтальне є нереальним, через неможливість розділити базу даних. Це може бути проблемою, якщо нам потрібно додати потужності.

Для SQL також ми маємо деякі обмеження в секунду на кількість запитів читання/запису.

У NoSql все навпаки – там немає схеми, немає (або дуже мало) відношень, можна з легкістю розділити базу даних на різні сервери, колекцій як правило менше, ніж таблиць і швидкодія набагато краща, ніж в SQL, але тільки коли йдеться про запити типу Read.

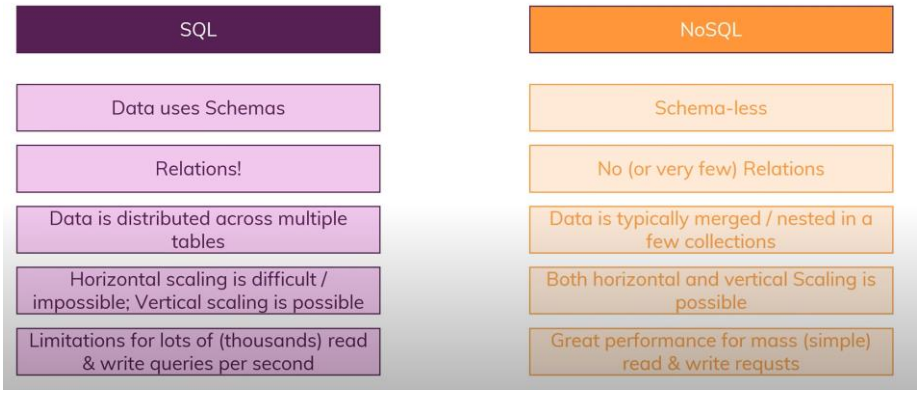

Рисунок 2.9 – SQL vs NoSql

Я вибрав для свого додатку саме NoSql базу даних через те, що у мене немає багато операцій запису, а є багато операцій читання та об’єднання даних.

Висновки до розділу

Я дослідив методи розробки веб-додатків, стратегії рендерінгу, наявні фреймворки, та архітектуру баз даних і визначився з тими технологіями, що їх я буду використовувати у своєму застосунку.

Я обрав React з-поміж інших фреймворків через легкість розгортування, простий синтаксис, компонентну архітектуру, гарну підтримку та велику кількість бібліотек, що їх можна використати для демонстрації матеріалу у застосунку «EduStatsTracker».

Також я вирішив використати Node.js з фреймворком Express, через простий та зрозумілий підхід до створення бекенду, з використанням MVC паттерну розробки.

Також я взяв за основу базу даних Mongo.db, через те, що вона більше підходить для потреб застосунку «EduStatsTracker», а саме – зберігання великої кількості документів, що не завжди пов’язані між собою, та більшу потребу у читанні, ніж у записі нових даних.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектура програми

Наш застосунок «EduStatsTracker» був реалізований на фреймворку React із застосуванням React-router-dom для маршрутизації, Redux – для менеджменту state та Redux-toolkit для зменшення кількості boilerplate-коду.

Наш застосунок можна запустити, виконавши збірку за допомогою утіліти Vite, яка є заміною стандартного create-react-app.

На бекенді у нас використовується Node.js та Express.js, як і було вже зазначено. Ми під'єднуємося до бази даних за допомогою ORM Mongoose.

Бекенд у нас має монолітну архітектуру, і ми виконуємо вимоги шаблону MVC, тобто у нас є контроллер – виконавець запитів, модель – містить бізнес-логіку та View - яким можуть слугувати маршрутизатори, запит до яких повертає дані (Додаток Б).

Пакет concurrency збирає нам увесь додаток – і фронтенд і бекенд, для комфортної розробки.

Ми також використали npm пакет faker для створення mock-даних.

За показ графіків відповідає бібліотека nivo, що надає доступ до великої кількості чартів та графіків. Вона є «обгорткою» над існуючою і перевіреною бібліотекою D3.js.

Хешування паролей відбувається у bcrypt.js.

Використання JavaScript як мови програмування для даного застосунку являється перевагою, оскільки розробники ми можемо використовувати одну мову для розробки як клієнтської, так і серверної частини застосунку.

Комбінація React, Node.js та MongoDB дозволяє нам розробляти наш веб-застосунок з високою продуктивністю, гнучкістю та швидкістю розробки. Ці технології мають активну спільноту та розширений екосистему, що дозволить при розробці швидко отримати необхідну допомогу у разі виявлення проблем.

Наведемо приклад точки входу у застосунок та конфігурації middleware (рисунок 3.1).

Рисунок 3.1 – Конфігурація точки входу застосунок

3.2 Структура бази даних

Оскільки була вибрана нереляційна база даних, у нас немає чіткої схеми, та, все ж, ми маємо задати певну модель кожного із документів, що їх ми матимемо у даному застосунку.

Для цього у нас є Mongoose, який дозволяє нам не тільки визначити всі необхідні поля у базі даних, а й задати валідацію за типом для кожного з полів.

У нас будуть наявні такі п’ять моделей (далі можна додати і інші):

– Студент;
– Професор;
– Користувач;
– Предмет;
– Оцінка.

Розглянемо модель бази даних «Студент». Ми маємо її представлення у MongoDb Compass та представлення із валідацією у ORM Mongoose.

– Ім’я (name) – тип String;
– Прізвище (last_name) – тип String;
– Пошта (email) – тип String;
– Телефон (Phone) – тип String (в подальшому можна зробити валідацію по regexp);

– Ідентифікатор (_id) – тип String – унікальний ідентифікатор;

– Чи виключений (is_expelled) – тип Boolean;

– Чи є стипендія (has_scholarship) – тип Boolean;

– Факультет (faculty) – Тип enum. Обмеження по значенням: ["Pr", "Pl", "Buh", "Obr"];

– Курс (course) – Тип enum. Обмеження по значенням: [1, 2, 3, 4];

– Група (group) – Тип String;

– Дисципліни (disciplines) – Тип Array. Масив foreign keys – дисциплін;

– Транзакції (transactions) – Тип Array. Також є масивом foreign keys;

– Чи є іноземцем? (is_foreign) – Тип Boolean;

– Оплата за семестр (semester_fee) – тип Number, відсутня у стипендіатів, у іноземців – різна, а у не іноземців – однакова;

– Країна (country) – Тип String, має відповідати стандарту неймінгу у бібліотеці Nivo;

– Чи є випускником? (Is_alumni) – тип Boolean;

– Стать (gender) – Тип Boolean. Зараз, як відомо, є більше ніж дві статі, але ми будемо враховувати для простоти лише базові варіанти;

– Рік закінчення (year_of_graduation) – Тип Number. Лише для випускників;

У майбутньому можна цю модель змінити, чи доповнити, в залежності від специфіки задач навчального закладу.

```
_id: "0xBeAe5d6Dde2B0xC66801dBE2D2"
name: "Steve"
last_name: "Rath"
email: "Steve89@gmail.com"
phone: "+380666951584"
is_expelled: false
has_scholarship: false
faculty: "Pl"
course: 2
group: "211-Pl"
▸ disciplines: Array
▸ transactions: Array
is_foreign: true
semester_fee: 3650
country: "JOR"
is_alumni: false
gender: true
year_of_graduation: null
__v: 0
```


Рисунок 3.2 – Схема студента у базі даних

А так виглядає Mongoose object із схемою студента (рисунок 3.2).

```
const StudentSchema = new mongoose.Schema(
    definition: {
        name: {
            type: String,
        },
        last_name: String,
        email: String,
        phone: String,
        _id: {
            type: String,
        },
        is_expelled: {
            type: Boolean,
        },
        has_scholarship: {
            type: Boolean,
        },
        faculty: {
            type: String,
            enum: ["Pr", "Pl", "Buh", "Obr"],
        },
        course: {
            type: Number,
            enum: [1, 2, 3, 4],
        },
        group: String,
        disciplines: {
            type: Array,
        },
        transactions: {
            type: Array,
        },
        is_foreign: Boolean,
        semester_fee: Number,
        country: String,
        is_alumni: Boolean,
        gender: Boolean,
        year_of_graduation: Number,
    },
);
```

Рисунок 3.3 – Mongoose об’єкт Student

Такий об’єкт є типовим для ORM Mongoose, ця технологія дозволяє стандартизувати доступ до бази даних та зробити первинну валідацію даних, що надходять з бекенду.

Основним принципом роботи моєї програми є створення різноманітних графіків із різних видів даних. Досягаю я цього, в умовах NoSql, за допомогою агрегацій.

Агрегація – спосіб запиту даних з у нереляційних базах даних. Представляє собою набір інструкцій, що виконуються одна за одною, послідовно змінюючи вивід даних.

У даній роботі ми сильно спираємося на цей інструмент. Його ми реалізуємо через код, і отримуємо змінені дані об'єднаних таблиць.

Приклад отримання всіх оцінок одного студента (рисунок 3.4)

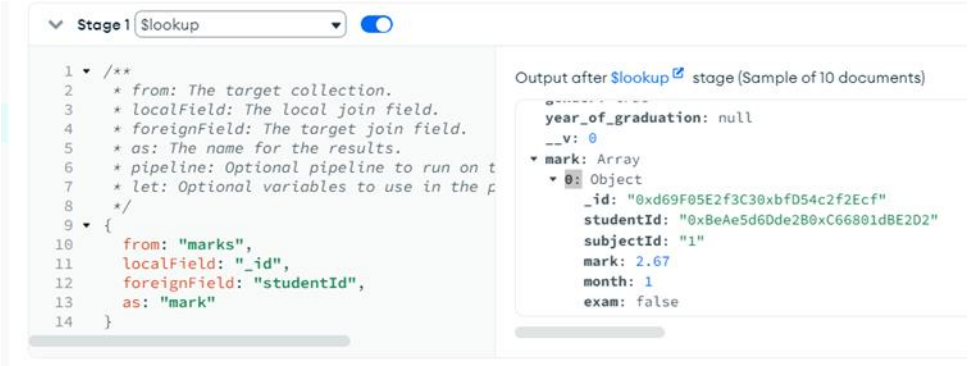

Рисунок 3.3 – Mongoose об’єкт Student

Так, із колекції marks ми робимо пошук (lookup) і отримуємо всі оцінки певного студента. Далі нам залишається лише сортувати їх за семестрами, роками, успішністю, екзаменами, ітд.

Взаємодія з базою даних: Mongoose надає потужні методи для звернення до MongoDB бази даних, такі як створення, зчитування, оновлення та видалення

документів. Вона спрощує виконання різноманітних операцій з даними та запитів до бази даних.

3.3 Авторизація у системі

У системі проводиться авторизація за допомогою JWT токенів і протоколу OAUTH.

Протокол OAuth (Open Authorization) є стандартом авторизації, який використовується для дозволу користувачам надавати обмежений доступ до своїх ресурсів третім сторонам без необхідності надавати їм свої облікові дані. OAuth широко використовується веб-сервісами та додатками для авторизації та доступу до захищених ресурсів.

Принциповою ідеєю OAuth є забезпечення доступу до ресурсів за допомогою виданого токену доступу (access token), який представляє обмежений набір дозволів, наданих користувачем. В процесі авторизації за протоколом OAuth використовуються наступні основні сторони:

– Власник ресурсу (користувач): Це особа, яка має ресурси, до яких треба надати доступ. Власник ресурсу вирішує, до яких даних він дозволяє доступ і керує цим доступом;

– Клієнт (додаток): Це додаток або сервіс, який бажає отримати доступ до ресурсів від імені користувача. Клієнт реєструється у провайдера авторизації та отримує ідентифікатори (Client ID) та секрети (Client Secret) для аутентифікації;

- Сервер авторизації (провайдер): Це сервер, який обробляє запити на авторизацію та видає токени доступу. Він перевіряє правомочність запиту, аутентифікує користувача та видає токен доступу клієнту;

Процес авторизації за протоколом OAuth включає такі основні кроки:

1) Реєстрація клієнта: Клієнт реєструється на сервері авторизації, отримує Client ID та Client Secret.

2) Перенаправлення користувача: Клієнт направляє користувача на сервер авторизації для аутентифікації.

3) Підтвердження доступу: Користувач надає свою згоду на доступ до своїх ресурсів клієнту.

4) Видача авторизаційного коду: Сервер авторизації видає авторизаційний код клієнту.

5) Отримання токена доступу: Клієнт обмінює отриманий авторизаційний код на токен доступу від сервера авторизації.

6) Використання токена доступу: Клієнт використовує токен доступу для отримання захищених ресурсів на сервері ресурсів.

За допомогою протоколу OAuth можна реалізувати безпечну та стандартизовану систему авторизації, яка дозволяє користувачам контролювати доступ до своїх ресурсів та дозволяє додаткам отримувати доступ до ресурсів без необхідності розкривати облікові дані користувачів.

Це загальний опис процесу авторизації з використанням JWT на React. Конкретні реалізації можуть варіюватися в залежності від вибраної бібліотеки або фреймворку для управління станом та роботи з JWT.

Загалом, аутентифікація та авторизація: JWT використовується для перевірки ідентичності користувача та надання прав доступу до ресурсів. Після успішної аутентифікації, сервер генерує JWT, який містить дані про користувача та наділяє його правами доступу. Клієнт зберігає JWT і включає його у кожний запит до сервера для перевірки автентичності та отримання необхідних дозволів.

У нашій програмі авторизація проводиться за допомогою middleware – проміжного компоненту у pipeline Node.js (рисунок 3.4).

Ми повертаємо токен користувачу, який читається відповідною бібліотекою, і з якого ми отримуємо усі дані користувача на фронтенді (рисунки 3.5, 3.6).

Рисунок 3.4 – Авторизаційне Middleware

Рисунок 3.5 – JWT – токен у Cookies

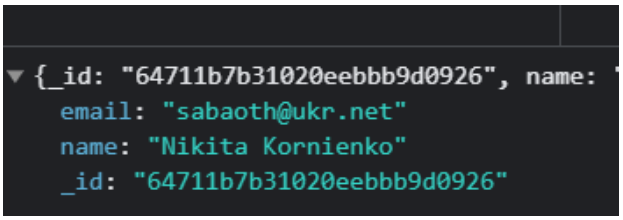

Рисунок 3.6 – User у LocalStorage

Зберігаючи токен у локальному сховищі, ми завжди можемо знати авторизаційний статус користувача і, в залежності від цього, надавати йому доступ.

3.4 Графічний інтерфейс користувача

У проектуванні графічного середовища ми використовуємо фронтенд бібліотеку MUI, також користуємось модульним CSS, та SCSS для нанесення стилів. Маємо такі вікна веб-застосунку:

– Сторінка входу в систему;
– Сторінка реєстрації;
– Сторінка Dashboard;

– Сторінка всіх студентів;
– Сторінка одного студента;
– Сторінка Країн;
– Сторінка Фінансової звітності;
– Сторінка Викладачів.

Подальші сторінки будуть доповнюватись, якщо застосунок набере якоїсь «ваги» та я буду мати намір доводити його до вигляду стартапу.

Зовнішній вигляд також може змінюватись. Наразі має місце стандартна кольорова схема MUI бібліотеки, дещо змінена локальним CSS.

Також виконано зміна кольорової гами на темний та світлий стиль, про що зберігається інформація у slice нашого Redux-store.

Є також можливість сховати Sidebar, про що інформація зберігатиметься у LocalStorage.

Сторінка входу виконана у сучасному стилі Neumorphism. Анімація «вдавлювання кнопки» виглядає симпатично (рисунок 3.7).

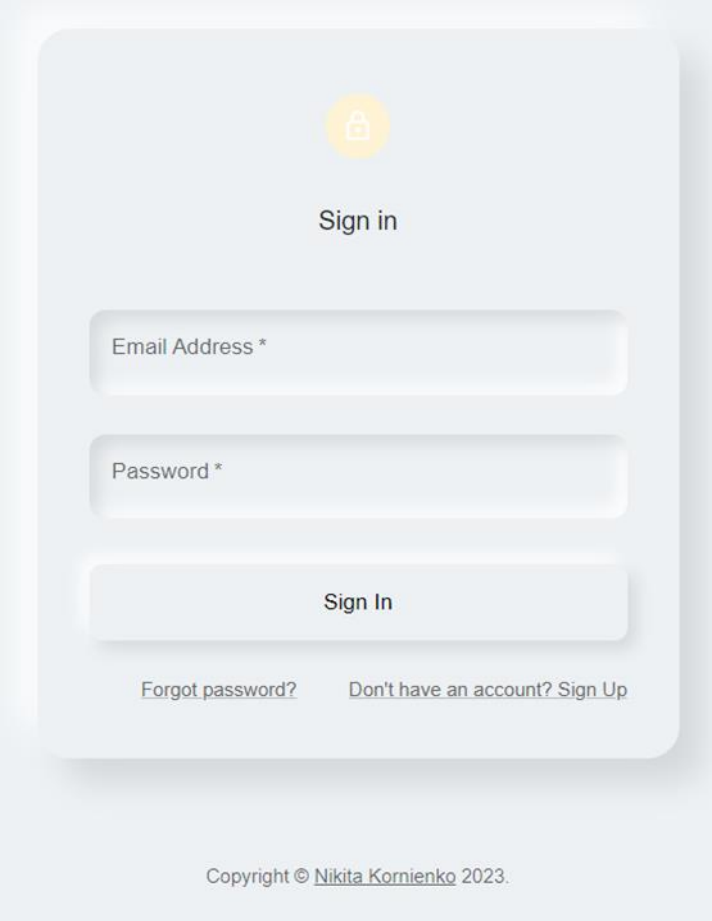

Рисунок 3.7 – Сторінка входу

Сторінка реєстрації абсолютно ідентична, окрім додаткових полів.

Сторінка входу містить форму, в якій користувач може ввести свої облікові дані для авторизації. Форма може включати наступні поля:

– Поле введення електронної пошти або ім'я користувача: Користувач вводить свою електронну пошту або ім'я користувача для ідентифікації;

– Поле введення пароля: Користувач вводить свій пароль для авторизації;

– Кнопка "Увійти": Кнопка, за допомогою якої користувач ініціює процес авторизації;

– Посилання "Забули пароль", що дозволяє користувачам скинути або відновити свій пароль.

Сторінка Dashboard відображатиме графіки успішності, кількість студентів, наявні дані про кожного студента, кількість студентів за категоріями та інше.

Різні віджети вбудовані в Grid систему на основі MUI (рисунок 3.8)

Рисунок 3.8 – Dashboard (темна тема)

Компонент "Dashboard" в нашому застосунку може містити різноманітну інформацію та функціонал, спрямований на надання користувачеві зручного та цілеспрямованого інтерфейсу.

Візуалізація даних у вигляді графіків, діаграм або інших візуальних елементів для наглядної презентації розподілу даних, трендів або аналізу продуктивності навчального закладу.

Таблиця студентів.

На цій сторінці можна подивитись всіх студентів та їх особисті дані, успішність, групу, курс ітд (рисунок 3.9).

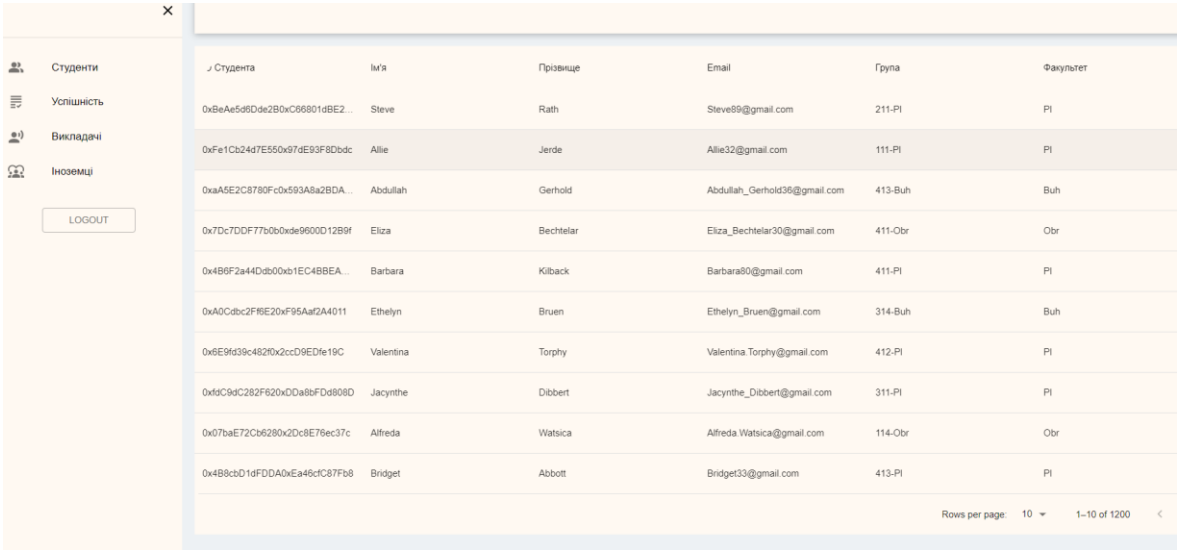

Рисунок 3.9 – Сторінка студентів (світла тема)

Таблиця студентів: Відображає список всіх зареєстрованих студентів. У таблиці можуть бути такі колонки:

– Прізвище та ім'я: Інформація про повне ім'я студента.

– Група: Інформація про групу або клас, до якого належить студент.

– Середній бал: Статистика про середній бал студента за певний період або за певні предмети.

Дії: Додаткові дії, які можуть бути виконані з кожним студентом, наприклад, перегляд деталей, редагування або видалення.

Пошук та фільтрація: Можливість пошуку та фільтрації студентів за різними критеріями, такими як група, прізвище або середній бал. Це дозволяє користувачу швидко знаходити потрібних студентів великого списку.

Сторінка одного студента

На цій сторінці є дані самого студента, його оцінки, успішність за місяцями (рисунок 3.10).

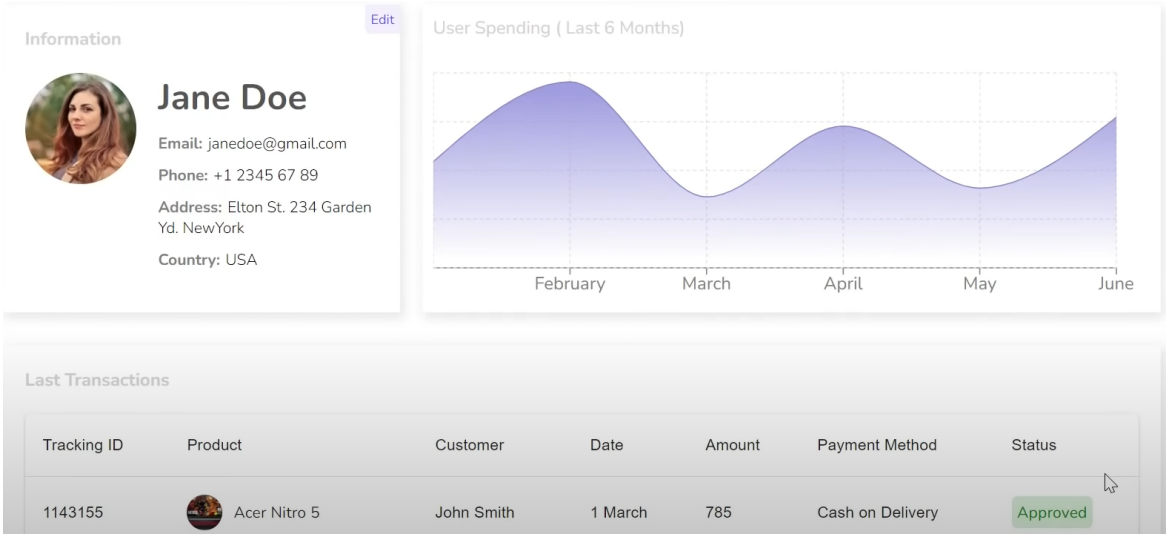

Рисунок 3.10 – Сторінка одного студента

Містить такі дані:

Особисті дані: Відображення особистих даних студента, таких як прізвище, ім'я, дата народження, стать, адреса тощо.

Контактна інформація: Показ контактної інформації студента, наприклад, електронної пошти, телефонного номера, адреси.

Інформація про групу: Відображення деталей про групу, до якої належить студент, такі як назва групи, рік навчання, керівник групи тощо.

Успішність та оцінки: Відображення інформації про успішність студента, наприклад, середній бал, оцінки за предмети, список завершених курсів чи проектів.

Фотографія: Показ фотографії студента, що дозволяє візуально ідентифікувати студента.

Сторінка географії

Відображає ті країни, з яких до нас приїхали студенти, їх кількість (рисунок 3.11).

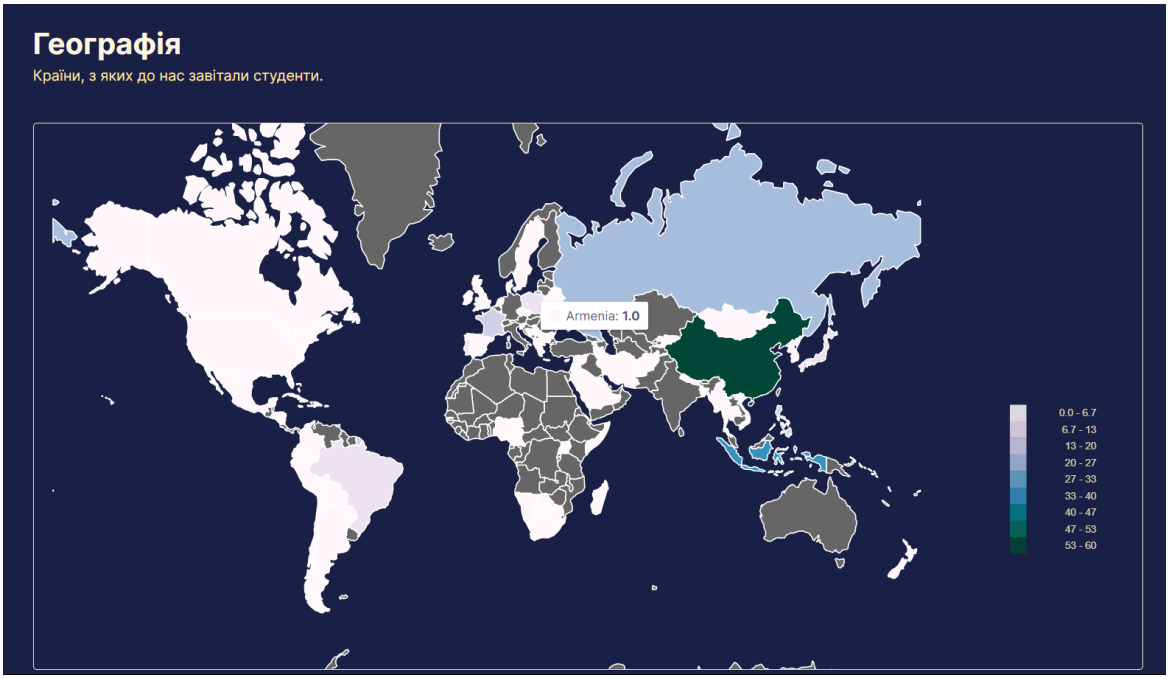

Рисунок 3.11 – Сторінка географії

Карта: Відображення картографічного інтерфейсу, на якому відображаються географічні об'єкти, такі як міста, країни, регіони, точки цікавості тощо. Карта може бути інтерактивною, дозволяючи користувачу наводити, масштабувати та переміщатися по ній.

Маркери та позначки: Показ різних маркерів або позначок на карті, що відображають певні географічні об'єкти, наприклад, місця розташування користувачів, важливі місця, події або інші цікаві об'єкти.

Інформація про місця: Відображення деталей про місця на карті. Це може включати назву місця, адресу, опис, фотографії, рейтинг, відгуки користувачів тощо.

Сторінка фінансів

Показує фінансові поступлення коледжу за останні роки по місяцям.

Сторінка має бути прикрита авторизацією (рисунок 3.11).

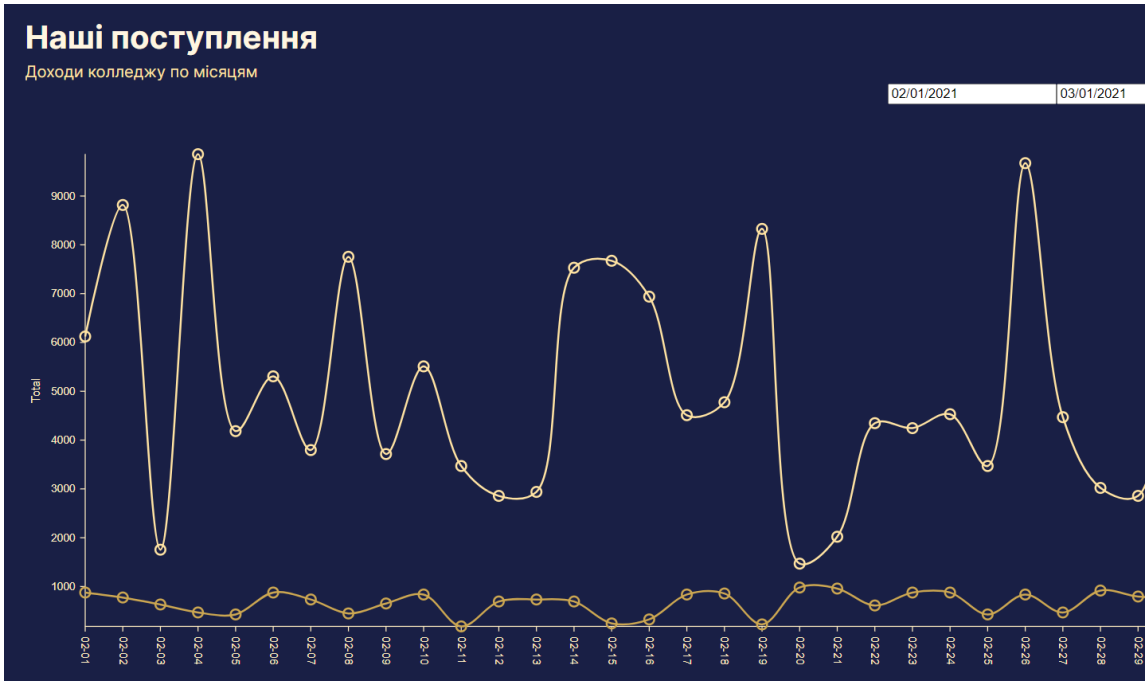

Рисунок 3.12 – Сторінка фінансів

У цьому модулі може бути зображено:

– Загальний огляд фінансового стану: Відображення загальної фінансової ситуації, такої як баланс, прибуток та витрати, активи та зобов'язання. Це може бути у вигляді графіків, діаграм або числових показників, які показують загальну картину фінансів;

– Бюджетування та планування: Можливість створення та керування бюджетом, встановлення фінансових цілей та планів. Користувач може створювати категорії витрат, встановлювати ліміти, слідкувати за виконанням бюджету;

– Витрати та доходи: Показ деталей про витрати та доходи. Це може включати списки транзакцій, дату, опис, категорію, суму тощо. Користувач може додавати нові транзакції, редагувати або видаляти існуючі, та інше.

3.5 Керівництво користувача

Даний застосунок є демонстративним варіантом системи для навчального закладу. Він має інтуітивно-зрозумілий принцип роботи і стандартне оформлення. Може бути розширений додатковими функціями, але наразі має лише демонструвати показники навчального закладу, що і було поставлено в задачі.

3.5.1 Область застосування

Може бути застосований у будь-яких навчальних закладах, що мають різні факультети, в тому числі факультет іноземців.

3.5.2 Основні можливості

Додаток демонструє дані навчального закладу, збираючи їх з бази даних Mongo.db за допомогою агрегацій. Має аутентикацію, реєстрацію та авторизацію, містить багато графіків та таблиць, що є корисним для розуміння показників роботи навчального закладу.

Висновки до розділу

У даному розділі я описав процес розробки додатку «EduStatsTracker». Під час реалізації я створив нереляційну базу даних Mongo.db з декількома таблицями, куди помістив згенеровані дані про студентів, викладачів, оцінки та користувачів. Дані я використав у візуальній частині, зробивши багато запитів, що необхідні для демонстрації функціоналу застосунку.

ВИСНОВКИ

У рамках даного дипломного проекту була розроблена система статистичної оцінки роботи навчального закладу, яка базується на використанні технологій React, Node.js та нереляційної бази даних Mongo.db. Основною метою системи є збір, аналіз та відображення статистичних даних, які дозволять навчальному закладу отримати об'єктивну оцінку своєї роботи та здійснювати ефективне управління.

У процесі розробки було проведено дослідження предметної області, вивчено показники навчального закладу, які варто відстежувати. Було виконано проектування та реалізацію системи, включаючи функціонал збору даних, обробки та відображення статистики. Важливою складовою розробленої системи є інтерфейс користувача, який забезпечує зручну навігацію та відображення результатів.

Застосування системи статистичної оцінки роботи навчального закладу може допомогти збільшити ефективність управління, виявити сильні та слабкі сторони навчального процесу, забезпечити об'єктивну оцінку якості навчання та забезпечити підґрунтя для прийняття обґрунтованих рішень щодо вдосконалення роботи навчального закладу.

На основі проведених досліджень та розробленої системи можна зробити висновок, що система статистичної оцінки роботи навчального закладу є потужним інструментом для збору та аналізу даних, що дозволяє отримати об'єктивну картину роботи навчального закладу.

Можливість подальшого розвитку системи я вбачаю у розширенні функціоналу, вдосконаленні алгоритмів аналізу даних та впровадженні додаткових модулів для різних аспектів оцінки роботи навчального закладу. Також варто звернути увагу на питання безпеки та захисту даних, забезпечення масштабованості та оптимізації системи для роботи з великими обсягами даних.

Також, розроблений бекенд API дає можливість приєднання іншого виду клієнту, такого як мобільний додаток, чи десктопна програма, звідки можна отримати ті ж самі дані, що і у веб-застосунку та зробити окремий клієнт на іншому виді платформи.

Намагаючись використати сучасний стек технологій, я ознайомився з багатьма парадигмами веб розробки, видами рендерінгу, фреймворками, видами баз даних, ORM-системами, сторонніми програмами та навчальними курсами, все це дозволило мені застосувати отримані знання у створенні даного дипломного проекту.

Загалом, розроблена система статистичної оцінки роботи навчального закладу має потенціал стати важливим інструментом для управління та покращення навчального процесу. Вона дозволяє збирати та аналізувати дані, що допомагає зробити об'єктивні рішення та покращити якість навчання в навчальних закладах.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. И. Браун. Веб-разработка с применением Node и Express (2021): 218-230.
2. Eberharn Wolff. Microservices. Flexible Software Architectures (2016) 344-410.
3. Luca Mezzalira. Front-End Reactive Architectures (2018) 129-149.
4. Anthony Accomazzo, Nate Murray, Ari Lerner, Clay Allsopp, David Guttman, and Tyler McGinnis. Fullstack React the Complete Guide to ReactJS and Friends (2020) 547-654.
5. Hari Narayn. Just React! (2022) 299–341.
6. Бэнкс Алекс, Порселло Ева. React: современные шаблоны для разработки приложений. (2021) 297–317.
7. Hargittai, E. (2018). Accessible App Design: Connecting Users to Inclusive Software. 75–92.
8. Maximillian Schwartzmueller URL: https://www.youtube.com/watch?v=ZS_kXvOeQ5Y.
9. Maximillian Schwartzmueller URL: https://www.youtube.com/watch?v=VELru-FCWDM.
10. MongoDB. URL: https://www.mongodb.com/docs/manual/aggregation/.
11. Платформа для створення діаграм. URL: https://app.diagrams.net/
12. Платформа для створення діаграм баз даних. URL: https://dbdiagram.io/home
13. React. URL: https://legacy.reactjs.org/docs/getting-started.html.
14. Udemy. URL: https://udemy.com/%20.
15. Vite. URL: https://vitejs.dev/guide/

ДОДАТКИ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ЗПІ-зп01.120БАК.007 ПЗ	Пояснювальна записка	61	
3	А3	ЗПІ-зп01.120БАК.007 ПЗ	Додаток А. Схема бази даних	1	
4	А3	ЗПІ-зп01.120БАК.007 ПЗ	Додаток Б. Схема взаємодії	1	
5	А3	ЗПІ-зп01.120БАК.007 ПЗ	Додаток В. Діаграма класів	1	
6	А3	ЗПІ-зп01.120БАК.007 ПЗ	Додаток Г. Процес роботи із застосунком	1	
7	А3	ЗПІ-зп01.120БАК.007 ПЗ	Додаток Ґ. Архітектурна схема застосунку	1	