МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КОМПЛЕКСНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система автоматичного складання розкладу. Блок редагування розкладу»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Шелехов І.В.**

Студента групи ІН.м-92 — **Марюха В.В.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Марюсі Віталієві Васильовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система автоматичного складання розкладу. Блок редагування розкладу*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Постановка задачі 2) Інформаційно-екстремальна інтелектуальна технологія 3) Інформаційна та програмна реалізація ситеми*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми та постановка задачі*		
2.	*Інформаційно-екстремальна інтелектуальна технологія*		
3.	*Інформаційна та програмна реалізація ситеми*		
4.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 49 стор., 24 рис., 2 табл., 2 додаток, 12 джерел.

Об’єкт дослідження — створення автоматичної системи складання розкладу.

Мета роботи — розробка інформаційного та програмного забезпечення системи автоматичного формування та контролю розкладу.

Методи дослідження — методи аналізу і синтезу інформаційних систем, методи теорії розкладів.

Результати —створено мобільний додаток для Android девайсів, який дозволяє користувачеві продивлятися розклад занять. Також була розроблена панель адміністрування для управління інформацією розкладу. Додатки виконані з використання фреймворків на мові JavaScript.

РОЗКЛАД ЗАНЯТЬ, ФРЕЙМВОРК, МОБІЛЬНИЙ ДОДАТОК, ANDROID, JAVASCRIPT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

JS – JavaScript

API – Application Programming Interface

MVC – Model-View-Controller

MVP – Model-View-Presenter

MVVM – Model – View – View – Model

SQL – Structured query language

UI – User interface

SDK – Software development kit

ВСТУП

Одним з найважливіших на сьогодні питань людства є розвиток. Адже цей процес ніколи не стоїть на місці, і в наші дні це стосується всіх сфер діяльності, а не тільки комп'ютерних технологій. Людство вигадало немало способів розвинутися. Одними із популярних способів розвитку в наші дні є онлайн- конференції, курси лекцій.

Кількість людей які цікавляться такими заходами досягає декількох сотень. Але виникає питання , як забезпечити всіх графіком лекцій так, щоб цей графік завжди був поряд. А також можливість повідомити відвідувачів про зміни в графіку або попередити про щось важливе.

На допомогу в таких випадках приходять сервіси, які допомагають систематизувати інформацію . Завдяки таким додаткам людина може контролювати свій час та бути повідомленою про зміни в розкладі.

Інформаційні додатки є дуже актуальними для мобільних телефонів, адже ці гаджети в наш час є в кожного і куди б ми не зібралися завжди поряд.

Кінцевим результатом роботи будуть системи формування та відображення розкладу з можливістю адміністрування. Головний адміністратор матиме змогу додавати , редагувати , видаляти розклад.

Таким чином , було запропоновано дослідити можливість створення системи для гнучкого управління розкладом, як додаток на мобільні пристрої та реалізувати систему до якої входить серверна частина та два клієнтських додатки.

1 ПОСТАНОВКА ЗАДАЧІ ДО СИСТЕМИ УПРАВЛІННЯ РОЗКЛАДОМ

1.1 Постановка задачі

Метою роботи є розробка гнучкої системи розкладу на базі React. В ході роботи було визначено наступні задачі:

– обрання архітектурної платформи для зв'язку між двома клієнтами;

– проєктування та розробка серверної частини системи;

– проєктування та розробка мобільного додатку користувача;

– проєктування та розробка додатку для адміністратора.

Для вирішення поставлених задач виділено декілька підзавдань:

– обрати фреймворк на якому буде виконаний проєкт ;

– обрати набори бібліотек для розробки додатків;

– провести дослідження методів розробки серверних систем для мобільних додатків

1.2 Загальні положення по взаємодію додатків з сервером

Найважливіша частина роботи - це взаємодія додатків з нашим сервером.

Клієнт – це програмовий компонент, який надсилає запити на сервер та отримує звідти відповіді. При роботі клієнта та серверу використовується спеціальний протокол. Наш додаток адміністрування матиме змогу вносити зміни , редагувати та видаляти розклад з таблиці. Тим часом ,як додаток клієнта на мобільному телефоні матиме змогу тільки продивлятися заданий адміном розклад.

Архітектура клієнт-серверу є одним із шаблонів архітектурного програмного забезпечення та є важливою концепцією у створенні розподілених мережевих додатків і передбачає взаємодію та обмін даними між ними.[7] Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та серверами. На рисунку 1.1 зображено базове представлення клієнт-серверної архітектури

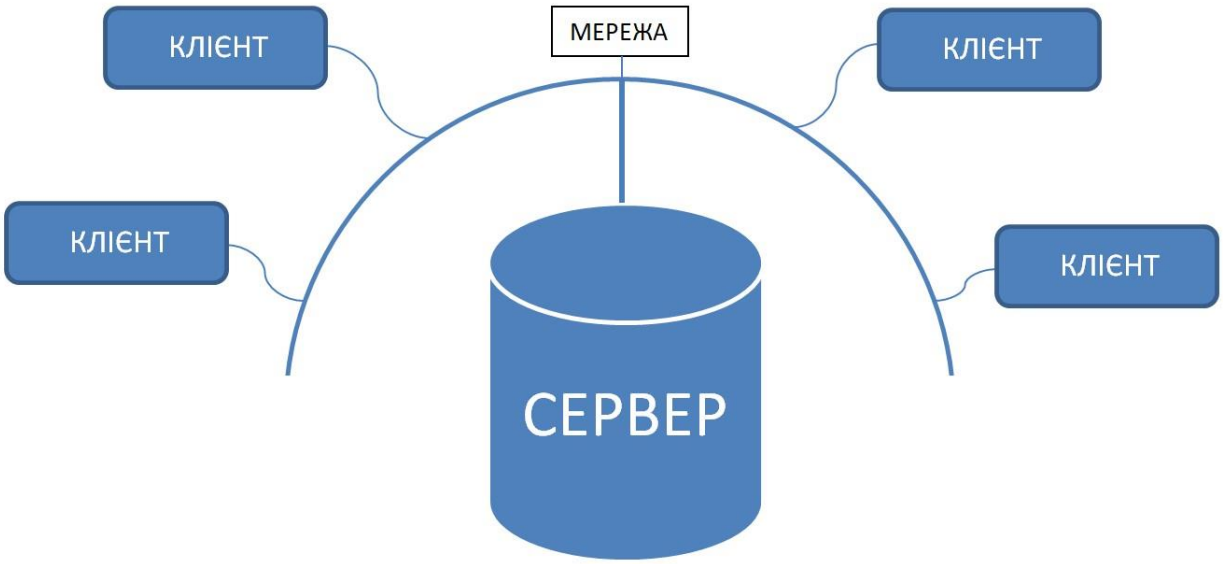

Рисунок 1.1 – Клієнт-серверна архітектура

REST – це підхід до архітектури інтернет протоколів, які забезпечують доступ до ресурсів інформації. В основі REST закладено принципи Всесвітньої павутини і, зокрема, можливості HTTP. Це означає що дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML чи JSON). Будь-який REST протокол в тому числі і HTTP, повинен підтримувати кешування, і не повинен залежати від мережевого шару, також не повинен зберігати інформації про стан. Стверджується, що такий підхід забезпечує масштабність системи і дозволяє їй еволюціонувати з новими вимогами. [6]

Клієнт-серверні додатки є дуже поширеними і в той же час дуже складними в розробці. Дії, які можна виконувати над ресурсом, визначаються повідомленнями, які визначено стандартним протоколом. В системі WWW це

протокол - HTTP, але існують інші REST-архітектури на основі деяких інших протоколів [3].

Найчастіше використовують 4 типи запитів:

– POST – додати новий ресурс;

– GET – отримати вже існуючий ресурс;

– PUT – замінити стан обраного ресурсу;

– DELETE – видалити ресурс.

На рисунку 1.2 зображено взаємодію клієнта та сервера за допомогою REST API.

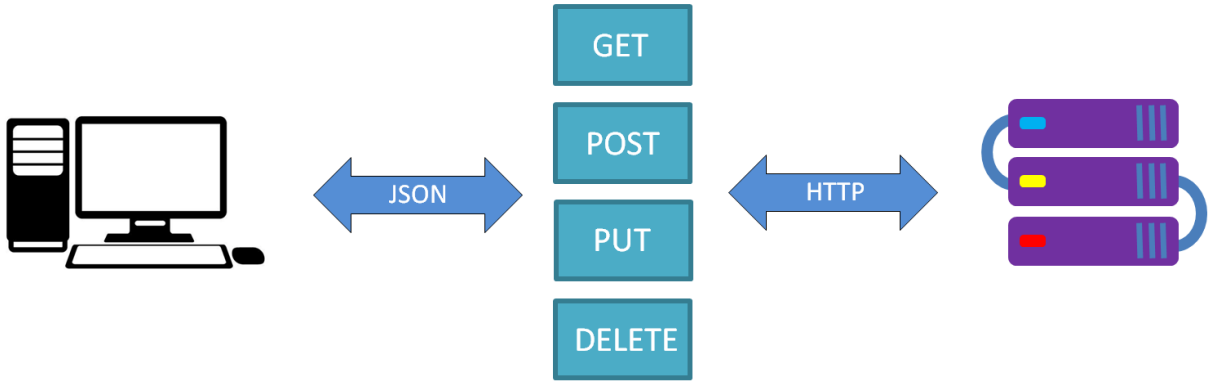

Рисунок 1.2 – взаємодію клієнта та сервера за допомогою REST API

Також для взаємодії з сервером можна використати технологію push сповіщень. За допомогою неї сервер може відправляти повідомлення на клієнта без його запиту.

Створення застосунків на подібній архітектурі є частим явищем. А також клієнт-серверна архітектура є актуальною в часі.

2 ІСНУЮЧІ ПРОГРАМИ РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ

В інтернеті існує безліч програм, які дозволяють розпланувати свій час на майбутнє. Деякі програми орієнтовані на використання для самого себе , такі програми не потребують серверу і всі задані записи зберігаються в пам’яті телефону. Інші програми орієнтовані на використання в гуртках , тренінгах , курсах та інших направленнях та навпаки потребують наявності серверу. Такі програми потребують контролю зі сторони власників гуртків . Всю потрібну інформацію власники заносять до адмінпанелі ,звідки вона потрапляє до бази даних, а потім на екрани клієнтів.

Розглянемо деякі готові варіанти додатків ,щоб розуміти ,які варіанти рішень є актуальними, а також, щоб розуміти, які функції нам потрібні.

Вивчивши ринок технологій розглянемо декілька схожих за функціонал додатків для управління розкладом заходів , тренінгів і т .д. Розглянемо наступні додатки: Timetable , Google Calendar.

2.1 Timetable

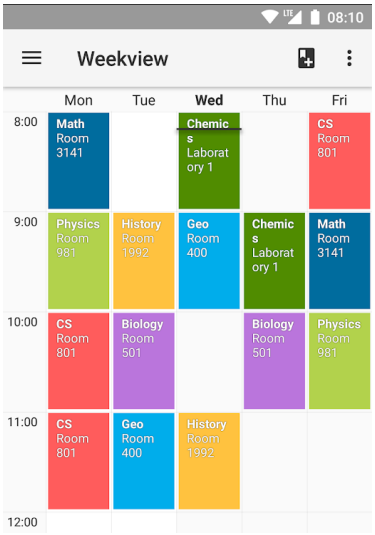

Рисунок 2.1 – Користувацький інтерфейс Timetable

Цей додаток дозволяє розпланувати розпорядок вашого шкільного або університетського життя. Після заповнення розкладу всі данні зберігаються на сервері ,що дозволяє синхронізуватися з любого пристрою, де ви ввійшли

до облікового запису. Також додаток має декілька простих, але дуже важливих функцій :

– автоматичне відключення звуку телефону, коли триває заняття;

– перегляд розкладу як списком так і сіткою;

– в додатку присутні віджети на робочий стіл вашого телефону;

– нагадування про домашнє завдання за день до його здачі;

2.2 Google Calendar

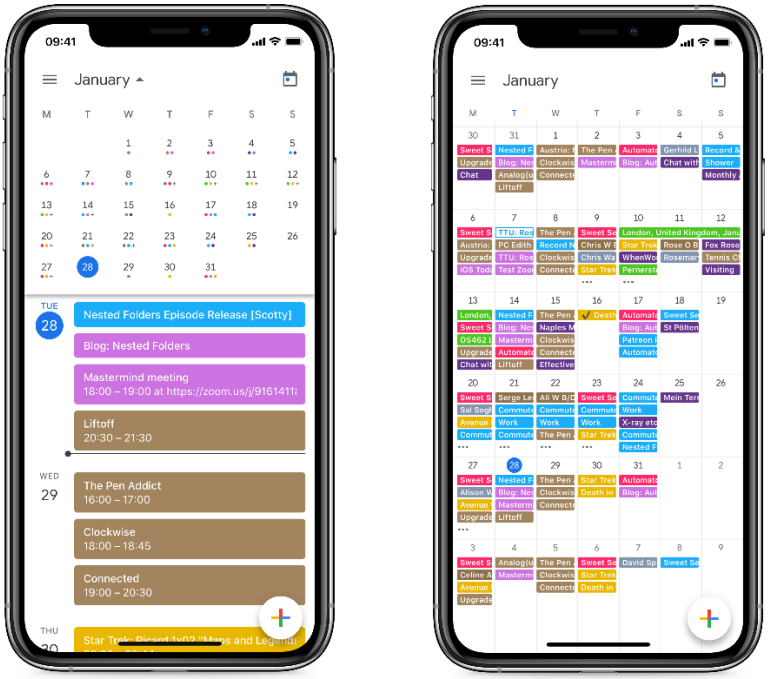

Рисунок 2.2 – Користувацький інтерфейс Google Calendar

В основі цього додатку лежить гнучка система управління розкладом. Цей аналог можна розглядати як приклад того, як повинен виглядати розклад. Додаток Google Calendar надає своєму користувачеві можливість дуже зручно та просто керувати своїм розкладом, а також ділитися або об’єднуватися з іншими користувачами, щоб бачити спільний розклад своїх записів.

Така система відрізняється від нашої системи розкладу. В нашій системі розкладу можливість додавати або змінювати розклад має тільки

адміністратор . А наш клієнт може тільки продивлятися розклад та слідкувати за його змінами.

2.3 SumDU Schedule

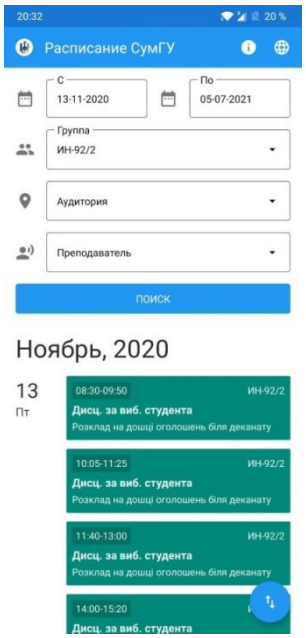

Рисунок 2.3 – Користувацький інтерфейс SumDU Schedule

Це зручний додаток, який дозволяє студентам слідкувати за початком, а також переносом лекцій. Також додаток має зручний фільтр для сортування лекцій по даті, аудиторії або за викладачем який проводить заняття.

2.4 Висновки до розділу

Було розглянуто дві системи гнучкого управлінні розкладом . З чого ми розуміємо, що кожна з них має свої переваги та недоліки .

Timeline зручний додаток який має свої цікаві функції, але не зовсім відповідає нашій ідеї розкладу.

Google Calendar дуже зручний та багато функціональний додаток, який надає великі можливості для самоорганізації свого часу, але в нашому випадку додатку для звичайного розкладу не потрібний такий функціонал.

SumDU Schedule додаток, який найбільш наближений до нашої ідеї тому по функціоналу будемо спиратися частково на нього. Додаток дозволяє всім користувачам переглядати розклад, але редагувати може тільки адміністратор.

Але в нашому випадку ми розуміємо, що додаток SumDU Schedule є найбільш наближений до нашої ідеї , розроблюваного нами розкладу.

3 ЗАСОБИ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ РОЗКЛАДОМ

Одним із головних завдань при розробці програмного забезпечення є вибір спеціальних засобів, які б полегшили та скоротили час розробки. Для реалізації додатків було використано редактор коду Visual Studio Code, який має зручний інтерфейс та багато плагінів, які спрощують роботу при розробці проєктів. Великим плюсом VSCode є зручне використання Git та Node.js.

Для розробки панелі адміністратора було використано React , який являє собою JavaScript бібліотеку для створення користувацьких інтерфейсів.

Додаток для мобільного телефону був розроблений на фреймворкові React Native , який має базу звичайного React. В проєкті було використано не звичайний React Native ,а набір інструментів Expo, який містить готові конфігурації Android Studio / XCode , надає змогу управляти сертифікатами в Apple & Google та push-повідомлень.

3.1 React

React — або інша назва React.js відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка створена вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими зустрічаються в розробці односторінкових застосунків.

React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, які змінюються з часом, без перезавантаження сторінки. Головна мета цієї бібліотеки бути швидким та гнучким рішенням для вирішення потрібних задач. React обробляє тільки користувацький інтерфейс у застосунках. Робота відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків. Як

бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.[9]

Model-View-Controller (MVC) — схема поділу даних програми, призначених для користувача інтерфейсу і керуючої логіки, має три окремих компоненти: модель, подання і контролер - таким чином, що кожна модифікація виконується незалежно. Структура MVC показана на рисунку 3.1.

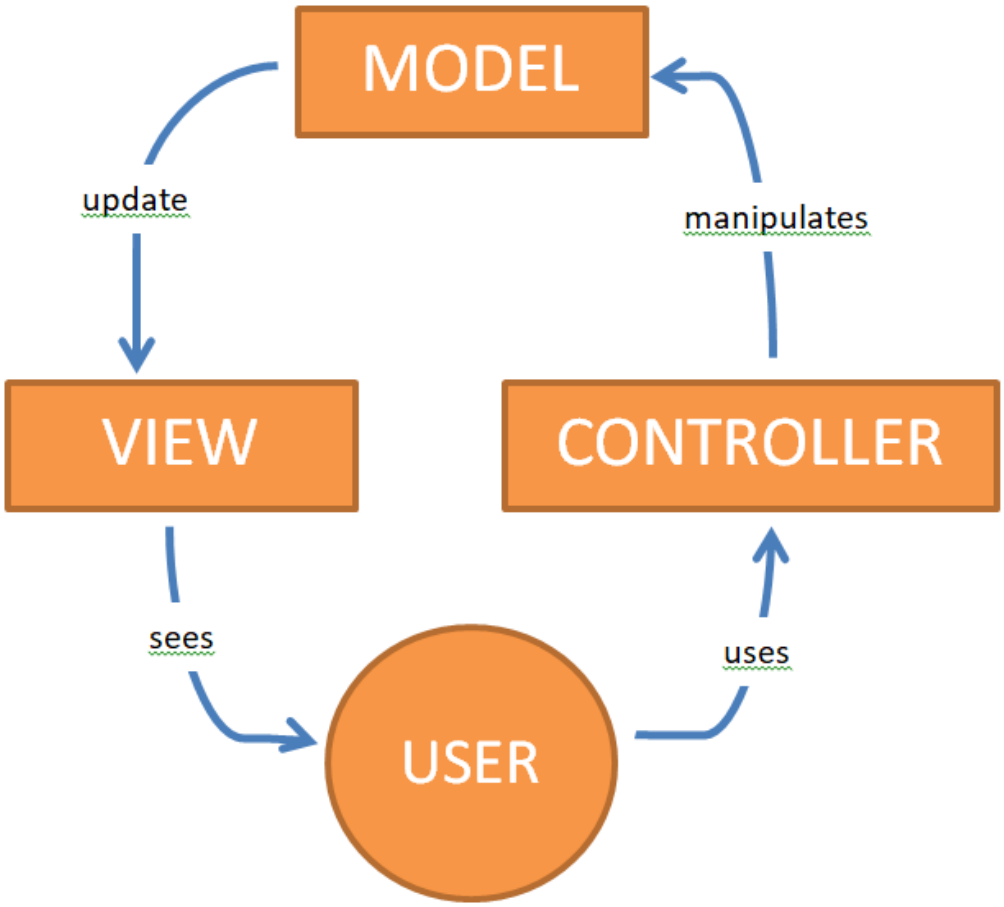

Рисунок 3.1 – Структура MVC

3.2 React Native

Що таке React Native?

React Native був представлений компанією Facebook у 2015р. що застосовує React архітектуру до нативних IOS, Android та UWP додатків.

Це сучасний фреймворк для написання кроссплатформенних мобільних додатків. Він дозволяє створювати єдиний проєкт під IOS і Android. Facebook, Instagram, Skype, Pinterest, Uber і інші великі компанії використали цей фреймворк для розробки своїх мобільних додатків за допомогою цієї технології. Дуже величезним плюсом є використання однієї бази ,писати один код, а отримати результат, який буде працювати відразу на двох платформах, iOS і Android. Цей фреймворк швидко розвивається та покращується , підтримується компаніями Facebook та Microsoft. Саме головне ,що цей продукт поширюється під ліцензією відкритого програмного забезпечення ,що дозволяє повністю бескоштовно реалізувати свої мобільні нативні додатки.

3.3 Середовище розробки додатків

Для написання обох клієнтських додатків було використано редактор коду Visual Studio Code.

Visual Studio Code — засіб для створення, редагування сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code є повністю безкоштовною і працює на Windows, Linux і OS X. За основу Visual Studio Code Microsoft використала напрацювання вільного проекту Atom, що розвивається компанією GitHub.

Редактор містить інструменти для роботи з Git і засоби рефакторингу, а також навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки.

Графічний інтерфейс програмного середовища Visual Studio Code можемо побачити на Рисунку 3.2.

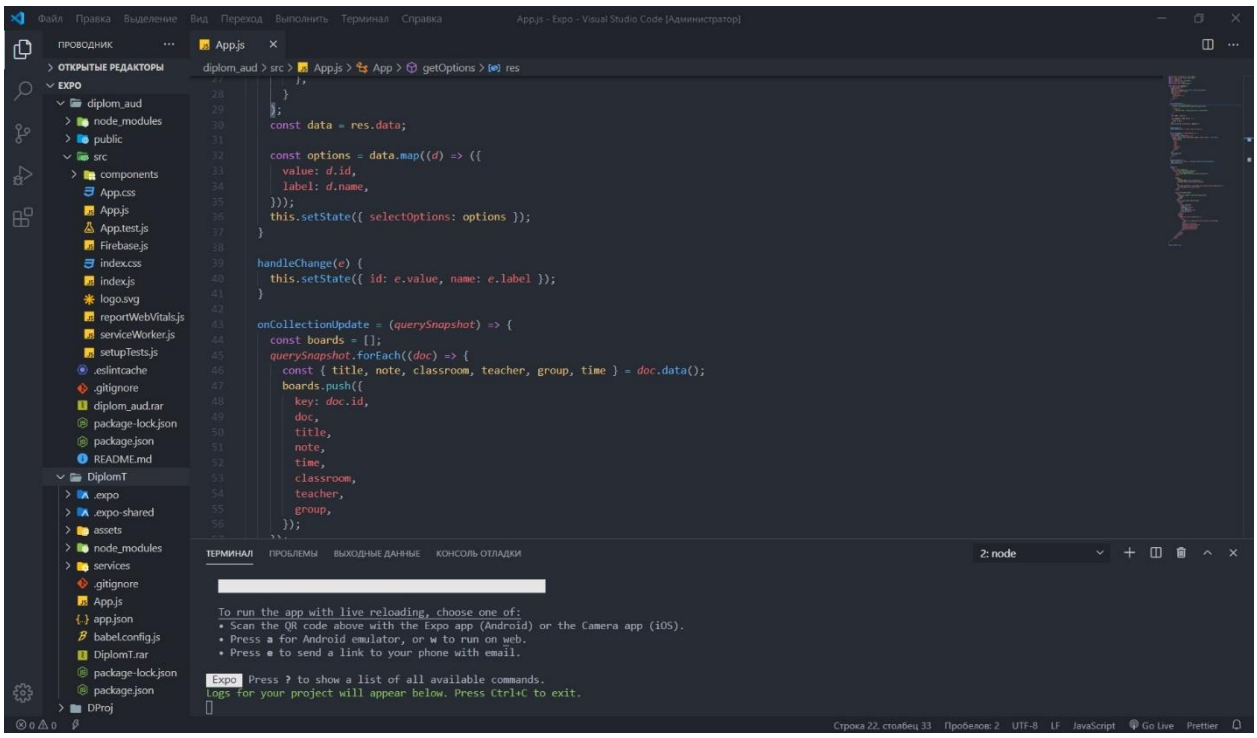

Рисунок 3.2 – Інтерфейс Visual Studio Code

Редактор VSCode має дуже багато розширень редактора, які полегшують життя розробника. Під час розробки були використані такі розширення:

– Prettier – полегшує читабельність коду в редакторі, дозволяє вільно писати код та під час зберігання приводить його в нормальний вигляд.

– Expo Tools – при роботі з файлами конфігурацій допомагає автоматично підтягнути потрібний пакет під час початку введення імені пакету.

– Simple React Snippets – містить готові сніпети та команди, які використовуються в React.

3.4 Вигляд структури файлів проєктів в редакторі коду

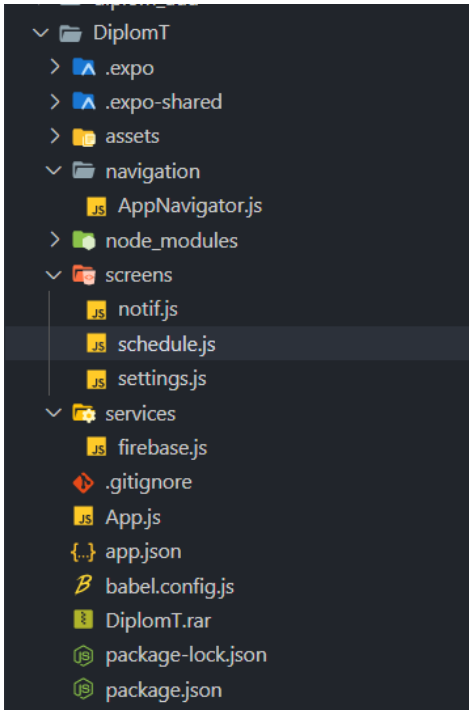

Рисунок 3.3 Вигляд структури мобільного додатку в редакторі коду VSCode

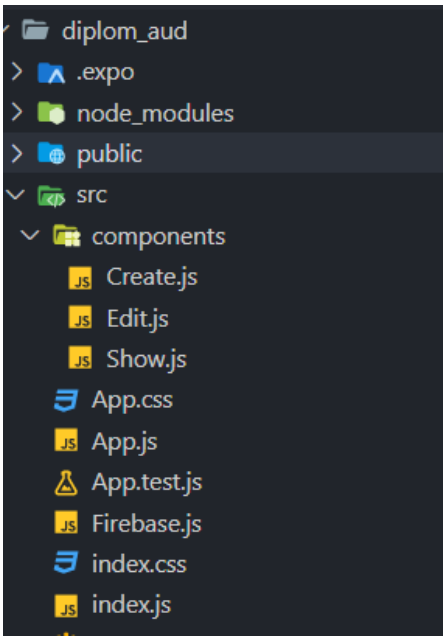

Рисунок 3.4 Вигляд структури адмінпанелі в редакторі коду VSCode

3.5 Архітектура розроблюваних додатків

В новій архітектурі React Native виконується поступова відмова від функцій моста, на зміну якій приходить новий механізм, званий JavaScript

Interface (JSI).Застосування JSI відкриває нові можливості для деяких чудових поліпшень в розробці.

Перше таке поліпшення полягає в тому, що JS-бандл більше не покладається на JSC. Іншими словами, двигун JSC тепер легко можна замінити на щось інше, цілком можливо, що відрізняється більш високою продуктивність.

Друге поліпшення - це те, що лежить в основі нової архітектури React Native. Воно полягає в тому, що, завдяки використанню JSI, в JavaScript можна зберігати посилання на C ++ - об'єкти (Host Objects) і викликати методи цих об'єктів. В результаті JavaScript-частина програми та нативні механізми будуть знати один про одного набагато більше, ніж раніше.[10]

На Рисунку 3.5 ми можемо побачити структуру нової архітектури React Native

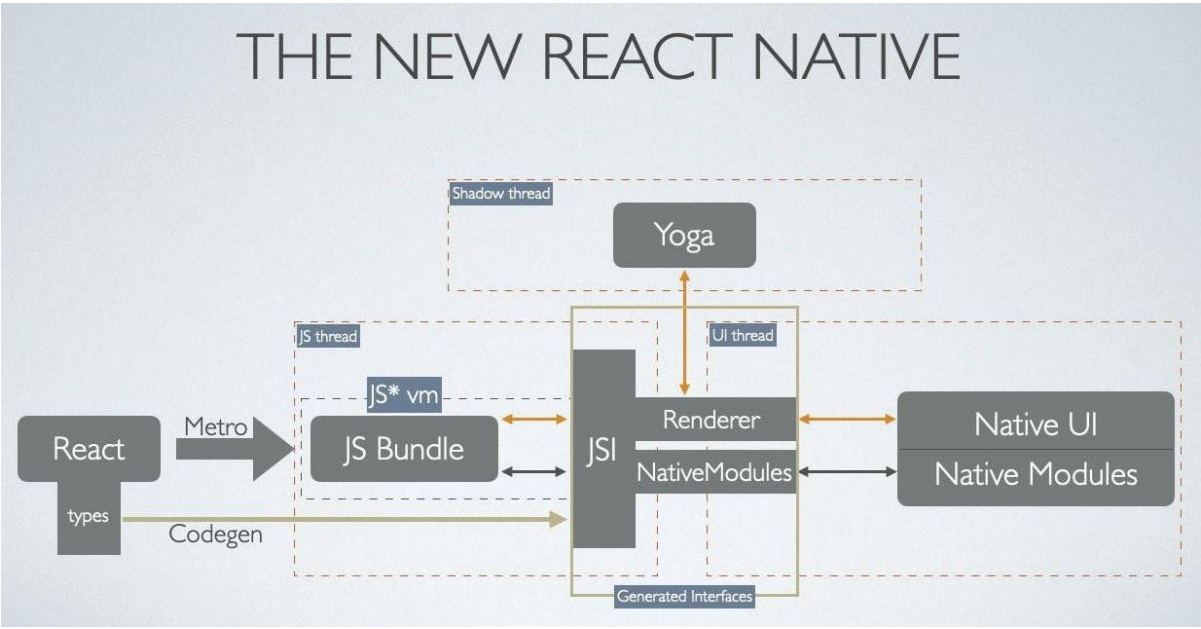

Рисунок 3.5 – Нова архітектура React Native

Іншими словами, JSI дозволяє забезпечити повну взаємодію між всіма потоками. Завдяки використанню концепції спільного володіння (shared ownership), JavaScript-код може запускати нативні методи безпосередньо з JS-потоку. При цьому немає необхідності в тому, щоб використовувати JSON

для передачі даних. Це дозволяє позбутися від проблем, характерних для використання моста, пов'язаних з можливим переповненням черги і з асинхронною передачею даних.

3.6 Інструменти розробки

Взаємодія наших двох додатків побудована на FireBase. Це дає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам мобільних чи веб-додатків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарному середовищі Firebase в виді таблиць. Приклад таблиці можна побачити на Рисунку 3.6.

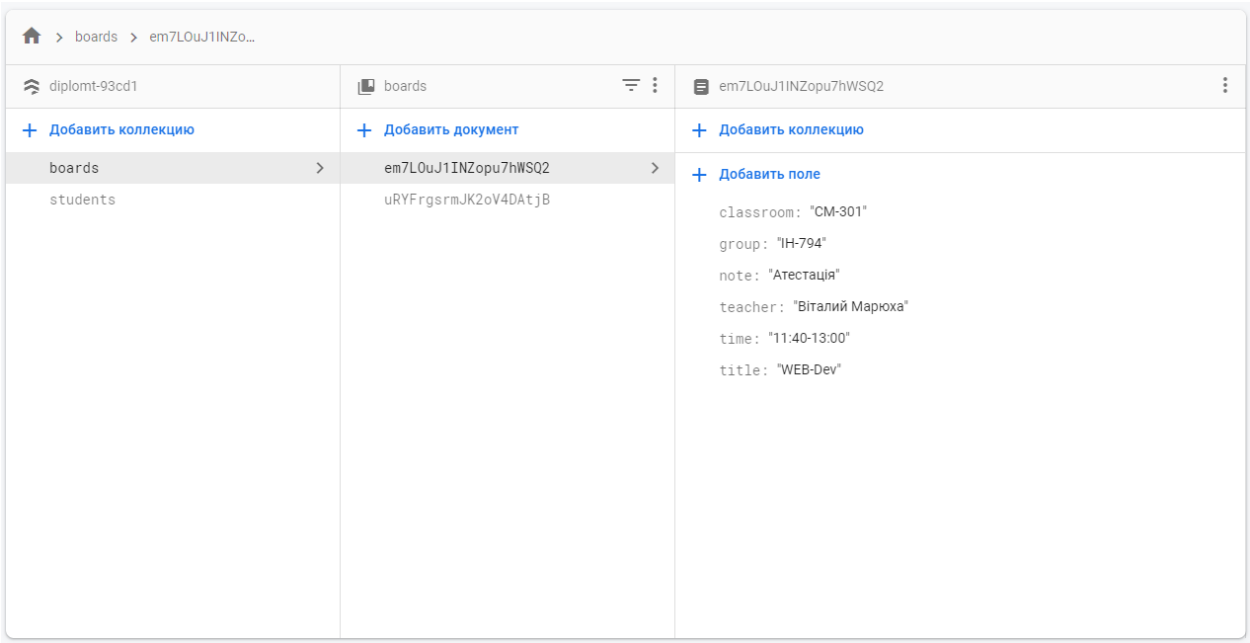

Рисунок 3.6 – Приклад таблиці бази даних Cloude Firestore

Компанія надає всі потрібні клієнтські бібліотеки, які дозволяють інтегрувати Firebase в додатки на Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. База даних також доступна через REST API та прив'язки до декількох сценаріїв JavaScript, таких як AngularJS, React, Ember.js та Backbone.js.

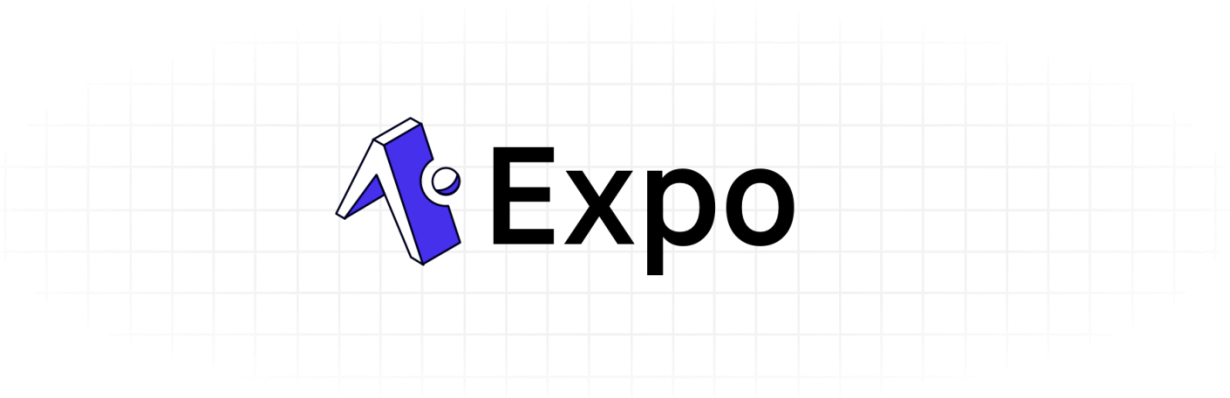

Рисунок 3.7 – Логотип Expo

В розробці мобільного додатку було використано набір інструментів Expo тому, що він має переваги при розробці.

Основні переваги:

– Expo розширює можливості роботи з API React Native Звичайний React Native не дає вам усіх можливостей при роботі з API в JS, а лише примітивні функції. В той час, коли Expo забезпечує всіми можливими для роботи з API та навіть більше.[12]

– Expo надає змогу оновлювати JS код в режимі реального часу, що набагато пришвидшує розробку додатків.

– Клієнт Expo надає можливість використовувати будь -які додатки сумісні з ним.

– Надає можливість зібрати фінальний результат в працюючий додаток однією командою в терміналі.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ РОЗКЛАДОМ

Проаналізувавши поставлені задачі ,почнемо розробку наших додатків . Перше з чого ми почнемо -це буде панель управління адміністратора , так як вона служить елементом, який наповнює нашу базу даних FireBase, звідки дані будуть зчитуватися та виводитися на додаток мобільного телефону.

4.1 Опис функціональної системи адмінпанелі

Адмінпанель реалізована на Reac.js , що дозволило нам використати вже готові деякі доповнення ,які скоротили час розробки . Використання доповнень - це хороша практика , адже вони не несуть ніякої шкоди , а також є повністю безкоштовними. Доповнення інсталюються за допомогою команд npm менеджера. Основними використаними доповненнями є :

```
"bootstrap": "^4.5.3",
"firebase": "^8.1.1",
```

Всі використані доповнення:

```
    "@testing-library/jest-dom": "^5.11.6",
    "@testing-library/react": "^11.2.2",
    "@testing-library/user-event": "^12.2.2",
    "axios": "^0.21.0",
    "bootstrap": "^4.5.3",
    "firebase": "^8.1.1",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.1",
    "react-select": "^3.1.1",
    "web-vitals": "^0.2.4"
```

Перейдемо до функцій ,які може виконувати адміністратор в адмін-панелі:

– має можливість бачити всі раніше додані заняття , у вигляді таблиці для зручності;

– можливість детально переглядати окремо додані заняття;

– можливість редагувати окремі заняття , та зберігати зміни;

– змога повністю видалити заняття з таблиці і тим самим з бази даних.

На даний момент функцій буде достатньо для управління предметами . Але в майбутньому для покращення зручності можна доповнювати різними цікавими функціями.

На Рисунку 4.1 ми можемо бачити схему системи управління розкладом.

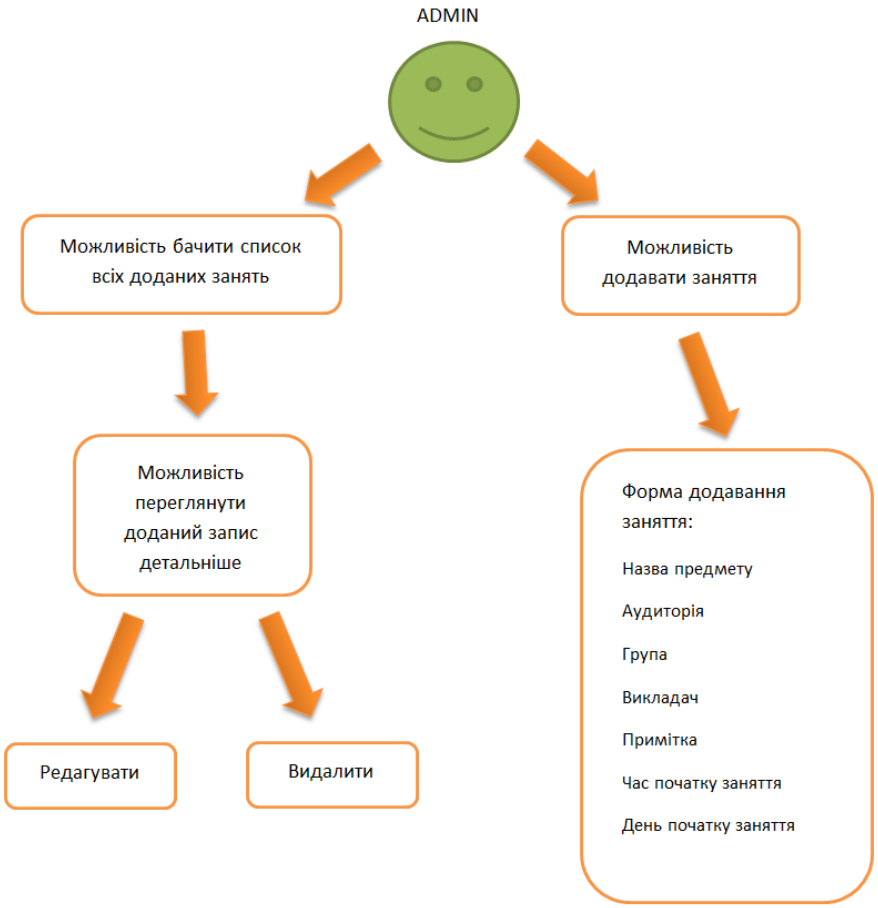

Рисунок 4.1 – Схема системи управління розкладом

4.2 Опис системи для відвідувача заняття

Тепер розглянемо функціональну систему мобільного додатку. Мобільний додаток реалізований на React Native. Написання додатку було

швидким тому, що сам React Native майже копія звичайного React.js хоча із деякими розбіжностями. Схема інформації доступної користувачеві представлена на рисунку

Рисунок 4.2 – Схема інформації доступної користувачеві

Що стосується дизайну, то тут ми не використали Bootstrap, а скористалися звичайними стилями StyleSheet.

4.3 Опис таблиць бази даних

Перш за все ми повинні розуміти, як влаштовані зв'язки в базі даних. А саме, яка структура у нас буде та які дані ми будемо зберігати в таблиці. Для того ,щоб записати якусь інформацію до бази даних для кожного варіанту запису в таблиці створюється клас на мові js , який має публічні властивості , та має місце в таблиці, як об'єкт.

В середовищі Firebase наша головна таблиця має назву " boards" .В головній колекції знаходяться всі записи, які формуються з адмінпанелі.

Таблиця 4.1. Структура головної таблиці “boards”

Ім’я колекції таблиць	Ім’я записів	Опис полів
boards	Всі імена записів зашифровано , приклад (HmIrPAx6IlHJ0IPxKNHT)	classroom"М-42" day"2021-03-17" group"Г-21" note"" teacher"Михайло Михайлович" time"11:25" title"3D МОДЕЛЮВАННЯ"

Всі сформовані записи отримують згенероване імя на приклад (HmIrPAx6IlHJ0IPxKNHT) . Це імя нам і непотрібно тому, що ми користуємося API ,яке за допомогою вказівок з js коду знаходить потрібні нам записи та дозволяє з ними працювати.[11] При створенні бази-даних в FireBase ми отримали API .яке ми можемо бачити на Рисунку 4.2.

```
const config = {
  apiKey: "AIzaSyAf7AvbzlH4oejMafG1bAuDKAUw8HJPLbc",
  authDomain: "diplomt-93cd1.firebaseapp.com",
  databaseURL: "https://diplomt-93cd1.firebaseio.com",
  projectId: "diplomt-93cd1",
  storageBucket: "diplomt-93cd1.appspot.com",
};
```

Рисунок 4.3 – API FireBase для роботи з нашою базою даних

Всі поля в наших записах формуються автоматично в js коді при відправці форми адмінпанелі. Всі об’єкти полів можна побачити в табл. 4.2.

Таблиця 4.2. Типи даних об’єктів наших записів

Імя об’єкту	Тип даних	Опис полів
Classroom	string	Назва аудиторії
Day	date	День початку заняття
group	string	Номер групи
note	string	Примітка
teacher	string	Імя викладача
time	time	Час початку заняття
Title	string	Назва предмету

Ми бачимо що всі дані ,які не потребують особливого формату запису мають тип даних string, всі які потребують -мають залежні від типу даних.

4.4 Архітектурна платформа

Для розробки в нашому проєкті була використана платформа Firebase. Інтерфейс платформи можна побачити Рисунок 4.3

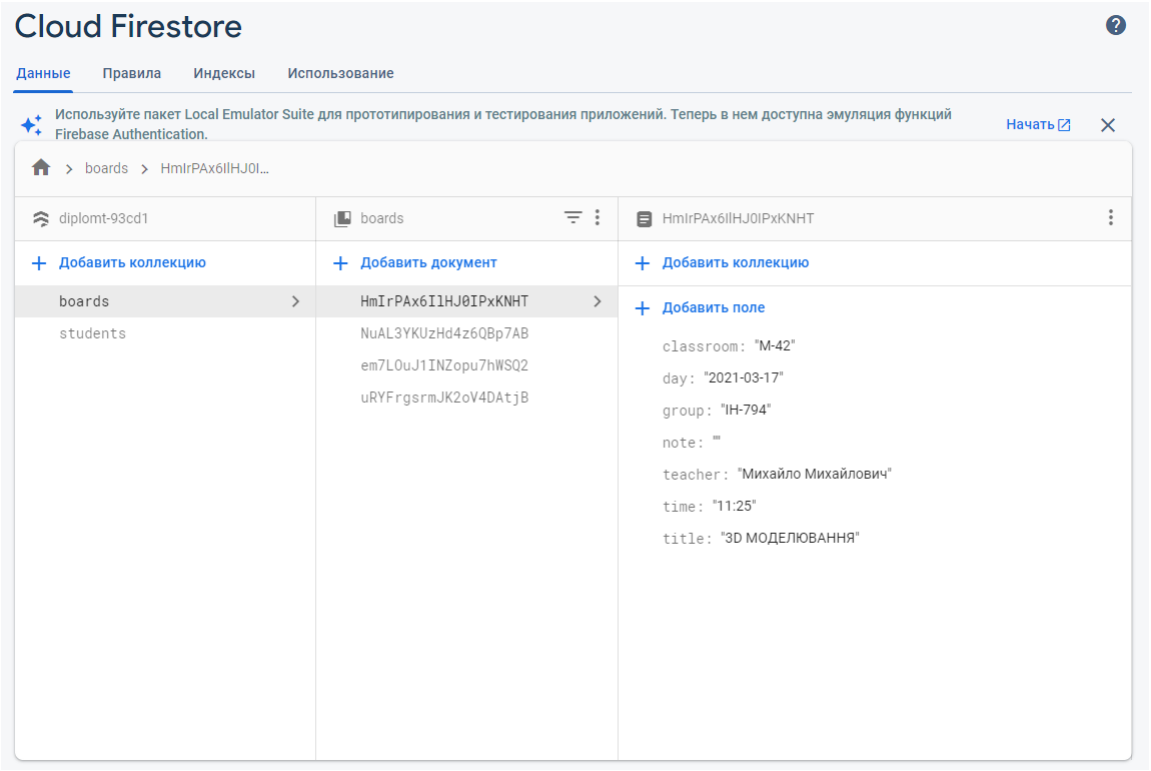

Рисунок 4.4 – Інтерфейс Firebase

В першу чергу платформа Firebase була розроблена для роботи з мобільними додатками ,але працювати з web-додатками ніхто не забороняв. Платформа має зручний спосіб роботи з даними.[11] Також можна налаштувати ролі та права для бази даних (Рисунок 4.3).

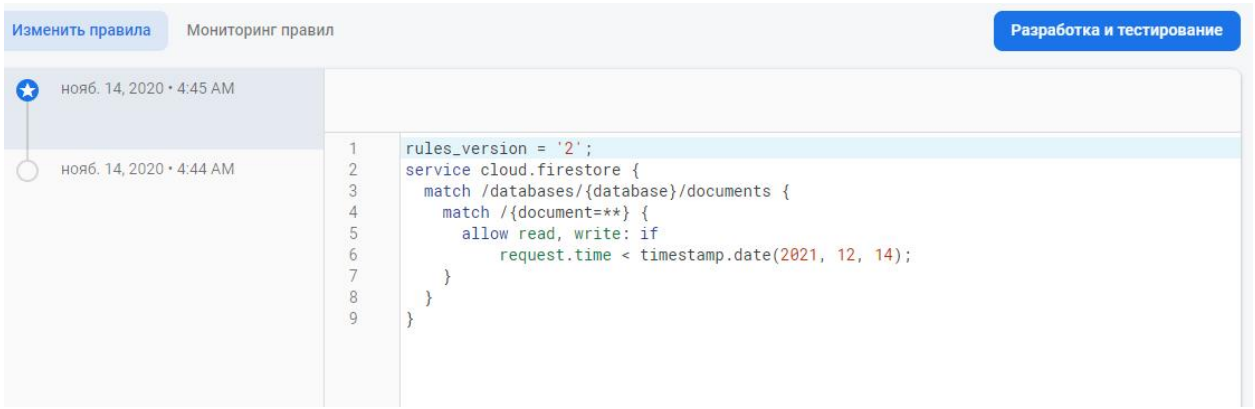

Рисунок 4.5 – Розділ налаштування правил

Ще однією особливістю є можливість слідкувати за графіком використання бази даних , де можна побачити коли і куди були передані дані.

4.5 Висновки до розділу

Як висновок в даному розділі було проаналізовано систему управління розкладом, яка побудована на архітектурній платформі Firebase.

5 ВІЗУАЛЬНИЙ ВИГЛЯД ТА МОЖЛИВОСТІ ДОДАТКІВ

Розглянемо інтерфейс панелі адміністратора та мобільного додатку.

5.1 Інтерфейс адмінпанелі

На головній сторінці розміщена таблиця всіх занять, які були додані до бази даних. Таблиця має поділ на стовпці , це дозволяє зрозуміти, які записи до яких стовпців належать. Також над таблицею розташована кнопка додавання заняття.

Рисунок 5.1 – Інтерфейс головної сторінки адмінпанелі.

При натисканні кнопки «Додати заняття» нас переправляє на сторінку формування нового запису. Іншими словами на сторінку з формою при заповненні та відправки якої данні з форми потрапляють до нашої бази даних. Інтерфейс форми додавання запису можна побачити на рисунку 5.2.

Розглянемо форму додавання заняття трохи детальніше. Для того ,щоб адміністратору не вводити час та дату на клавіатурі були використані типи даних поля Html . Це значить, що при натисканні на спеціальні іконки часів або календаря адміністратор побачить відмет календаря або часів, де зможе обрати час, а також дату в календарі.

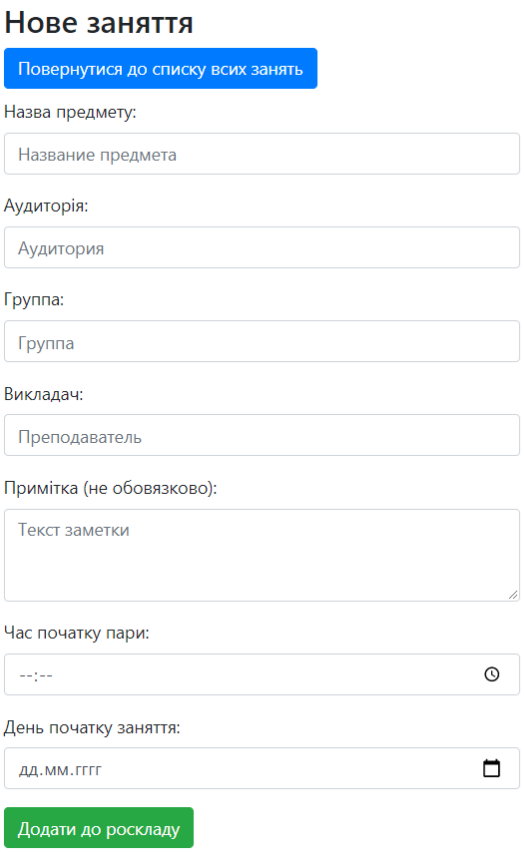

Рисунок 5.2 – Інтерфейс форми додавання заходу

Також форма має валідацію, яка перевіряє чи заповнені всі обов'язкові поля. Наприклад, якщо адміністратор забув заповнити поле «Назва предмету» то отримає помилку з текстом «Заповніть це поле». Фото приклад помилки можна побачити на рисунку 5.3.

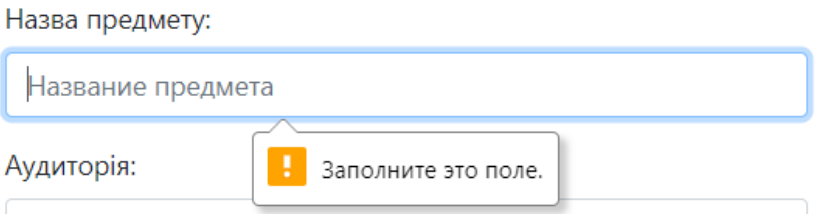

Рисунок 5.3 – Помилка при валідації

Перейдемо до сторінки детального огляду кожного запису. При натисканні на любий із записів потрапляємо на сторінку з даними цього

запису , а також можливістю «Редагувати» або «Видалити» цей запис. Інтерфейс детального огляду запису можна бачити на рисунку 5.4.

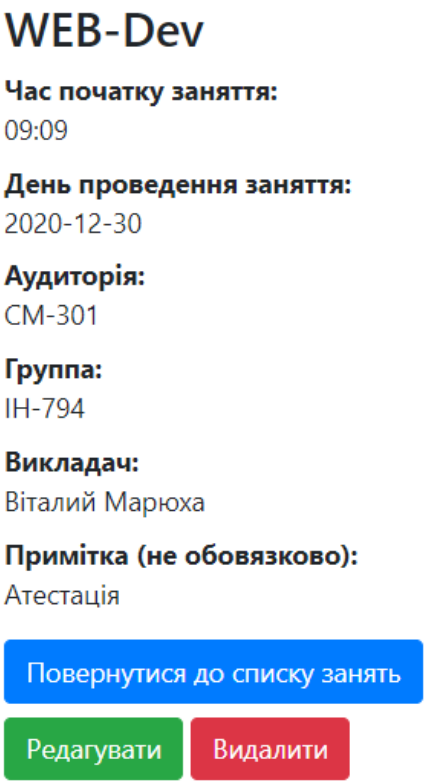

Рисунок 5.4 – Сторінка детального огляду

При натисканні кнопки «Редагувати» потрапляємо на сторінку редагування . Де бачимо форму схожу до форми додавання запису, але вже з інформацією, яка підтяглась з бази даних. Інформацію можна відредагувати та зберегти . Під час збереження вся інформація в базі даних буде перезаписана . Інтерфейс панелі редагування можна бачити на рисунку 5.5.

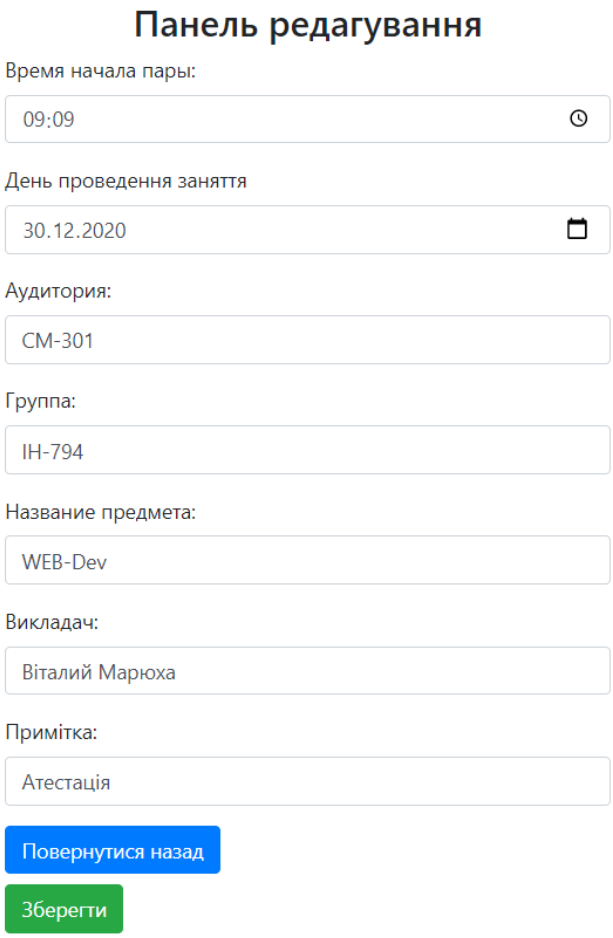

Рисунок 5.5 – Сторінка редагування запису

Після збереження змін , нас переправляє на головну сторінку, де ми можемо додати новий запис або змінювати вже додані . Також інтерфейс повністю адаптивний для роботи на різних розширеннях екрану, а також має підтримку різних браузерів.

Код додатку адмінпанелі буде прикріплено в додатку А.

5.2 Інтерфейс мобільного додатку

Перейдемо до розгляду мобільного додатку. При вході в мобільний додаток бачимо головний екран додатку, який можна бачити на рисунку 5.6. На екрані розташовані записи занять, а також фільтри за якими можна відфільтрувати всі заняття. В нижній частині додатку знаходиться панель з вкладками «Розклад» , «Сповіщення», а також «Налаштування». В верхній частині розташваний заголовок вкладки.

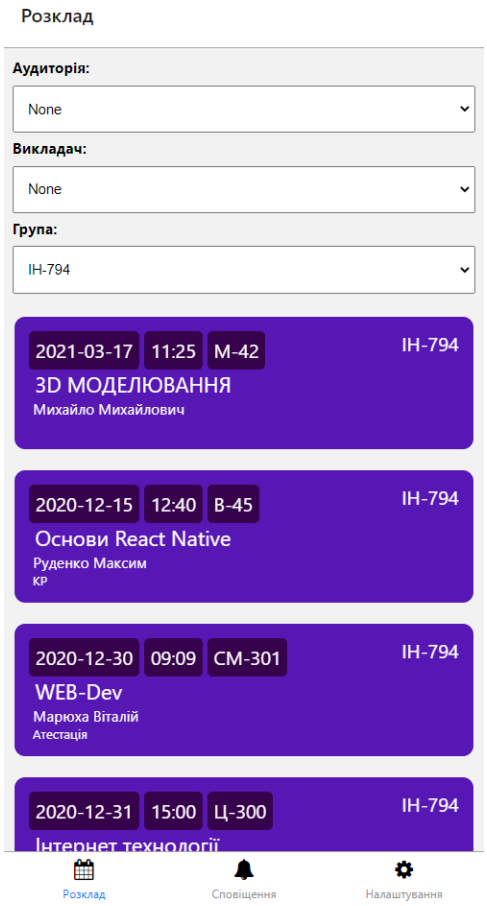

Рисунок 5.6 – Головний екран мобільного додатку

Користувач додатку має можливість фільтрувати всі записи за аудиторією, групою чи викладачем . Це надає зручності в користуванні додатком.

Розглянемо детальніше вигляд запису , який має детальне, але просте на вигляд представлення інформації. В записі вказано день та час початку заняття, групу відвідувача , назву предмету та ім’я викладача . Також вказана примітка, яка є необов’язковою, адже вона не завжди потрібна. Схему запису з поясненнями можна побачити на рисунку 5.7.

Рисунок 5.7 – Схема запису з поясненнями

Код мобільного додатку буде прикріплено в додатку В.

5.3 Інсталяція та системні вимоги

Для роботи мобільного додатку знадобиться пристрій на базі ОС Android версії 4.1 або вище. На даний момент на ринкові 90% усіх телефонів в яких версія Android вище 4.1 . Що до пристрою ,де будуть користуватися додатком, різниці не має так, як наш додаток повністю адаптивний і відображається коректно на усіх пристроях.

ВИСНОВОК

В даному дипломному проєкті було досліджено та реалізовано систему управління розкладом заходів.

Для представлення даних було створено мобільний додаток для Android девайсів, який дозволяє користувачеві продивлятися розклад занять. Також була розроблена панель адміністрування для управління інформацією розкладу. Користувачами даної системи є клієнт заходу та адміністратор, як керівник заходу. В майбутньому розроблювана система буде покращуватися та удосконалюватися.

Було проаналізовано вже готові рішення мобільних додатків. Обґрунтовано рішення створення додатку адміністрування та мобільного додатку ,який є дуже актуальним в наш час. В ході роботи було розглянуто одну з найкращих архітектурних платформ для реалізації своїх проєктів .

Як підсумок в дипломному проєкті було вирішено поставлену задачу, а також вирішено всі проблеми, які виникли під час виконання завдання.

СПИСОК ЛІТЕРАТУРИ

1. Reac - React Native 2020 - https://uk.wikipedia.org/wiki/React#React_Native
2. Android Framework [електронний ресурс]. – 2019. – : https://www.quora.com/What-is-meant-by-Android-framework
3. REST [електронний ресурс]. – 2020 – Режим доступу: https://uk.wikipedia.org/wiki/REST
4. Дубова Н. SOA: //Відкриті системи. — 2004. — № 6. — С. 37–41.
5.Цехнер Марио — Программирование под Android / — М.: Питер, 2012. — 688 с.
6. Сервісно орієнтоване програмування [електронний ресурс] https://www.slideshare.net/v_matush/service-oriented-programming
7. Клієнт-серверна архітектура [електронний ресурс] - https://habr.com/ru/post/495698/
8. Основи React Native [електронний ресурс] - https://tproger.ru/articles/your-first-app-in-react-native/
9. Redux Core [електронний ресурс] - https://redux.js.org/introduction/getting-started
10. Книга - Fullstack React Native - https://coursehunter.net/course/kniga-fullstack-react-native-polnoe-rukovodstvo-po-react-native
11. База документації Firebase [електронний ресурс] - https://riptutorial.com/ru/firebase
12. База Expo.io [електронний ресурс] - https://docs.expo.io/introduction/walkthrough/

ДОДАТОК А

Файл create.js

```
import React, { Component } from "react";
import ReactDOM from "react-dom";
import firebase from "../Firebase";
import { Link } from "react-router-dom";

class Create extends Component {
  constructor() {
    super();
    this.ref = firebase.firestore().collection("boards");
    this.state = {
      time: "",
      classroom: "",
      title: "",
      teacher: "",
      note: "",
      group: "",
      data: [],
    };
  }



  onChange = (e) => {
    const state = this.state;
    state[e.target.name] = e.target.value;
    this.setState(state);
  };

  onSubmit = (e) => {
    e.preventDefault();

    const { title, note, classroom, teacher, group, time } = this.state;

    this.ref
      .add({
        title,
        note,
        classroom,
        teacher,
        time,
        group,
      })
      .then((docRef) => {
        this.setState({
          title: "",
          note: "",
          classroom: "",
          teacher: "",
          time: "",
          group: "",
        });
        this.props.history.push("/");
      })
      .catch((error) => {
        console.error("Error adding document: ", error);
      });
  };

  render() {
```

```
const { title, note, classroom, teacher, group, time, day } = this.state;
return (
  <div class="container container-create">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">Додавання пари</h3>
      </div>
      <div>
      {this.state.data}
    </div>
      <div class="panel-body">
        <h4>
          <Link to="/" class="btn btn-primary">
            Повернутися до списку всих занять
          </Link>
        </h4>
        <form onSubmit={this.onSubmit}>
          <div class="form-group">
            <label for="title">Назва предмету:</label>
            <input
              required
              type="text"
              class="form-control"
              name="title"
              value={title}
              onChange={this.onChange}
              placeholder="Название предмета"
            />
          </div>
          <div class="form-group">
            <label for="classroom">Аудиторія:</label>
            <input
              type="text"
              class="form-control"
              name="classroom"
              value={classroom}
              onChange={this.onChange}
              placeholder="Аудитория"
            />
          </div>
          <div class="form-group">
            <label for="group">Группа:</label>
            <input
              type="text"
              class="form-control"
              name="group"
              value={group}
              onChange={this.onChange}
              placeholder="Группа"
            />
          </div>
          <div class="form-group">
            <label for="teacher">Викладач:</label>
            <input
              type="text"
              class="form-control"
              name="teacher"
              value={teacher}
              onChange={this.onChange}
              placeholder="Преподаватель"
            />
          </div>
          <div class="form-group">
            <label for="note">Примітка (не обовязково):</label>
```

```
                <textArea
                  class="form-control"
                  name="note"
                  onChange={this.onChange}
                  placeholder="Текст заметки"
                  cols="80"
                  rows="3"
                >
                  {note}
                </textArea>
              </div>
              <div class="form-group">
                <label for="time">Час початку пари:</label>
                <input
                  type="time"
                  class="form-control"
                  name="time"
                  value={time}
                  onChange={this.onChange}
                  placeholder="Время начала пары"
                />
              </div>
              <div class="form-group">
                <label for="time">День початку заняття:</label>
                <input
                  type="date"
                  class="form-control"
                  name="time"
                  value={time}
                  onChange={this.onChange}
                  placeholder="Время начала пары"
                />
              </div>
              <button type="submit" class="btn btn-success">
                Додати до розкладу
              </button>
            </form>
          </div>
        </div>
      </div>
    );
  }
}

export default Create;
```

Файл Edit.js

```
import React, { Component } from 'react';
import firebase from '../Firebase';
import { Link } from 'react-router-dom';

class Edit extends Component {

  constructor(props) {
    super(props);
    this.state = {
      key: '',
      title: '',
      note: '',
      time: '',
      day: '',
      group: '',
```

```
    };
  }

  componentDidMount() {
    const ref = firebase.firestore().collection('boards').doc(this.props.matc
h.params.id);
    ref.get().then((doc) => {
      if (doc.exists) {
        const board = doc.data();
        this.setState({
          key: doc.id,
          title: board.title,
          note: board.note,
          classroom: board.classroom,
          teacher: board.teacher,
          time: board.time,
          day: board.day,
          group: board.group,
        });
      } else {
        console.log("No such document!");
      }
    });
  }

  onChange = (e) => {
    const state = this.state
    state[e.target.name] = e.target.value;
    this.setState({board:state});
  }

  onSubmit = (e) => {
    e.preventDefault();

    const { title, note,classroom,teacher,group, time, day } = this.state;

    const updateRef = firebase.firestore().collection('boards').doc(this.stat
e.key);
    updateRef.set({
      title,
      note,
      time,
      day,
      classroom,
      teacher,
      group
    }).then((docRef) => {
      this.setState({
        key: '',
        title: '',
        note: '',
        time: '',
        day: '',
        classroom: '',
        teacher: '',
        group: '',
      });
      this.props.history.push("/show/"+this.props.match.params.id)
    })
    .catch((error) => {
      console.error("Error adding document: ", error);
    });
  }
```

```
  render() {
    return (
      <div class="container container-edit">
        <div class="panel panel-default">
          <div class="panel-heading">
            <h3 class="panel-title text-center">
              Панель редагування
            </h3>
          </div>
          <div class="panel-body">
            <form onSubmit={this.onSubmit}>
            <div class="form-group">
                <label for="time">Время начала пары:</label>
                <input type="time" class="form-
control" name="time" value={this.state.time} onChange={this.onChange} placeho
lder="Время начала пары" />
              </div>
            <div class="form-group">
                <label for="day">День проведення заняття</label>
                <input type="date" class="form-
control" name="day" value={this.state.day} onChange={this.onChange} placehold
er="День проведення заняття" />
              </div>
              <div class="form-group">
                <label for="classroom">Аудитория:</label>
                <input type="text" class="form-
control" name="classroom" value={this.state.classroom} onChange={this.onChang
e} placeholder="Аудитория" />
              </div>
              <div class="form-group">
                <label for="group">Группа:</label>
                <input type="text" class="form-
control" name="group" value={this.state.group} onChange={this.onChange} place
holder="Группа" />
              </div>
              <div class="form-group">
                <label for="title">Название предмета:</label>
                <input required type="text" class="form-
control" name="title" value={this.state.title} onChange={this.onChange} place
holder="Назва предмету" />
              </div>
              <div class="form-group">
                <label for="teacher">Викладач:</label>
                <input type="text" class="form-
control" name="teacher" value={this.state.teacher} onChange={this.onChange} p
laceholder="Викладач" />
              </div>
              <div class="form-group">
                <label for="note">Примітка:</label>
                <input type="text" class="form-
control" name="note" value={this.state.note} onChange={this.onChange} placeho
lder="Примітка" />
              </div>
              <h4><Link to={`/show/${this.state.key}`} class="btn btn-
primary">Повернутися назад</Link></h4>
              <button type="submit" class="btn btn-success">Зберегти</button>
            </form>
          </div>
        </div>
      </div>
    );
  }
}
```

```
export default Edit;
```

Файл Show.js

```
import React, { Component } from "react";
import firebase from "../Firebase";
import { Link } from "react-router-dom";

class Show extends Component {
  constructor(props) {
    super(props);
    this.state = {
      board: {},
      key: "",
    };
  }

  componentDidMount() {
    const ref = firebase
      .firestore()
      .collection("boards")
      .doc(this.props.match.params.id);
    ref.get().then((doc) => {
      if (doc.exists) {
        this.setState({
          board: doc.data(),
          key: doc.id,
          isLoading: false,
        });
      } else {
        console.log("Нет такой записи!");
      }
    });
  }

  delete(id) {
    firebase
      .firestore()
      .collection("boards")
      .doc(id)
      .delete()
      .then(() => {
        console.log("Предмет удалён!");
        this.props.history.push("/");
      })
      .catch((error) => {
        console.error("Ошибка при удалении документа: ", error);
      });
  }

  render() {
    return (
      <div class="container container-show">
        <div class="panel panel-default">
          <div class="panel-heading">
            <h3 class="panel-title">{this.state.board.title}</h3>
          </div>
          <div class="panel-body">
            <dl>
              <dt>Час початку заняття:</dt>
              <dd>{this.state.board.time}</dd>
```

```
              <dt>День проведення заняття:</dt>
              <dd>{this.state.board.day}</dd>
              <dt>Аудиторія:</dt>
              <dd>{this.state.board.classroom}</dd>
              <dt>Группа:</dt>
              <dd>{this.state.board.group}</dd>
              <dt>Викладач:</dt>
              <dd>{this.state.board.teacher}</dd>
              <dt>Примітка (не обовязково):</dt>
              <dd>{this.state.board.note}</dd>
            </dl>
            <h4>
              <Link to="/" class="btn btn-
primary">Повернутися до списку занять</Link>
            </h4>
            <Link to={`/edit/${this.state.key}`} class="btn btn-success">
              Редагувати
            </Link>
             
            <button
              onClick={this.delete.bind(this, this.state.key)}
              class="btn btn-danger"
            >
              Видалити
            </button>
          </div>
        </div>
      </div>
    );
  }
}

export default Show;
```

Файл App.js

```
import React, { Component } from "react";
import { Link } from "react-router-dom";
import "./App.css";
import firebase from "./Firebase";
import axios from "axios";
import Select from "react-select";

class App extends Component {
  constructor(props) {
    super(props);
    this.ref = firebase.firestore().collection("boards");
    this.unsubscribe = null;
    this.state = {
      boards: [],
      selectOptions: [],
      id: "",
      name: "",
    };
  }

  async getOptions() {
    const res = await axios.get(
      "https://mirroxdev.github.io/portfolio/users.json",
      {
        headers: {
```

```
          "Content-Type": "application/json; charset=UTF-8",
        },
      }
    );
    const data = res.data;

    const options = data.map((d) => ({
      value: d.id,
      label: d.name,
    }));
    this.setState({ selectOptions: options });
  }

  handleChange(e) {
    this.setState({ id: e.value, name: e.label });
  }

  onCollectionUpdate = (querySnapshot) => {
    const boards = [];
    querySnapshot.forEach((doc) => {
      const { title, note, classroom, teacher, group, time, day } = doc.data(
);
      boards.push({
        key: doc.id,
        doc,
        title,
        note,
        time,
        day,
        classroom,
        teacher,
        group,
      });
    });
    this.setState({
      boards,
    });
  };

  componentDidMount() {
    this.unsubscribe = this.ref.onSnapshot(this.onCollectionUpdate);
    this.getOptions();
  }

  render() {
    return (
      <div class="container">
        <div class="panel panel-default">
          <div class="panel-heading">
            <h3 class="panel-title text-center">СПИСОК ЗАНЯТЬ</h3>
          </div>

          {/* <div>
            <Select
              options={this.state.selectOptions}
              onChange={this.handleChange.bind(this)}
            />
            <p>
              You have selected <strong>{this.state.name}</strong> whose id i
s{" "}
              <strong>{this.state.id}</strong>
            </p>
          </div> */}
```

```
            <div class="panel-body m-20">
              <h4>
                <Link to="/create" class="btn btn-primary">
                  Додати заняття
                </Link>
              </h4>
              <table class="table table-stripe">
                <thead>
                  <tr>
                    <th>Предмет</th>
                    <th>Час початку заняття</th>
                    <th>День початку заняття</th>
                    <th>Аудиторія</th>
                    <th>Группа</th>
                    <th>Викладач</th>
                    <th>Примітка</th>
                    {/* <th>Видалити</th> */}
                  </tr>
                </thead>
                <tbody>
                  {this.state.boards.map((board) => (
                    <tr>
                      <td>
                        <Link to={`/show/${board.key}`}>{board.title}</Link>
                      </td>
                      <td>{board.time}</td>
                      <td>{board.day}</td>
                      <td>{board.classroom}</td>
                      <td>{board.group}</td>
                      <td>{board.teacher}</td>
                      <td>{board.note}</td>
                    </tr>
                  ))}
                </tbody>
              </table>
            </div>
          </div>
        </div>
      );
    }
}

export default App;
```

ДОДАТОК В

Файл AppNavigator.js

```
import React from "react";
import { createAppContainer } from 'react-navigation'
import { createStackNavigator } from 'react-navigation-stack'
import { createBottomTabNavigator } from 'react-navigation-tabs'

import Icon from 'react-native-vector-icons/FontAwesome';

import schedule from '../screens/schedule'
import notif from '../screens/notif'
import settings from '../screens/settings'

const _SchNavigator = createStackNavigator({
    schedule: {
        screen: schedule,
        navigationOptions: {
            title: 'Розклад'
        }
    }
})
const _NotifNavigator = createStackNavigator({
    schedule: {
        screen: notif,
        navigationOptions: {
            title: 'Сповіщення'
        }
    }
})

const _SetNavigator = createStackNavigator({
    settings: {
        screen: settings,
        navigationOptions: {
            title: 'Налаштування'
        }
    }
})

const AppNavigator = createBottomTabNavigator({
    schedule: {
        screen: _SchNavigator,
        navigationOptions: {
            title: 'Розклад',
            tabBarIcon: ({ tintColor }) => (
                <Icon name="calendar" size={20}/>
              ),
        },
        tabBarOptions: {
            showIcon: true ,
            activeTintColor: '#e91e63',
         },
    },
    notif: {
        screen: _NotifNavigator,
        navigationOptions: {
            title: 'Сповіщення',
            tabBarIcon: ({ tintColor }) => (
                <Icon name="bell" size={20}/>
              ),
        },
```

```
        tabBarOptions: {
            showIcon: true ,
            activeTintColor: '#e91e63',
         },
    },
    settings: {
        screen: _SetNavigator,
        navigationOptions: {
            title: 'Налаштування',
            tabBarIcon: ({ tintColor }) => (
                <Icon name="cog" size={20}/>
              ),
        },
        tabBarOptions: {
            showIcon: true ,
            activeTintColor: '#e91e63',
         },
    },
}, {
    initialRouteName: 'schedule'
})

export default createAppContainer(AppNavigator)
```

Файл schedule.js

```
import React, { useMemo, useState } from "react";
import {
  View,
  StyleSheet,
  SafeAreaView,
  ScrollView,
  Text,
} from "react-native";
import { emptyValue, ScheduleFilter } from "./components/ScheduleFilter";
import { useFilter } from "./hooks/useFilter";

import { useSchedule } from "./hooks/useSchedule";

export function Schedule() {
  const [classroomFilter, setClassroomFilter] = useState(emptyValue);
  const [teacherFilter, seTeacheFilter] = useState(emptyValue);
  const [groupFilter, setGroupFilter] = useState(emptyValue);

  const isFilterSelected = [classroomFilter, teacherFilter, groupFilter].some(el
=> el !== 'None');
  const { classrooms, teachers, groups, scheduleData } = useSchedule();

  const filters = useMemo(() => {
    return {
      ...(classroomFilter !== emptyValue ? { classroom: classroomFilter } : {}),
      ...(teacherFilter !== emptyValue ? { teacher: teacherFilter } : {}),
      ...(groupFilter !== emptyValue ? { group: groupFilter } : {}),
    }
  }, [classroomFilter, teacherFilter, groupFilter]);
  const filtered = useFilter({ scheduleData, filters });
  return (
    <SafeAreaView>
      <View style={page.SFilter}>
```

```
        <Text style={page.FilSelTName}>Аудиторія:</Text>
        <ScheduleFilter style={page.Filter_line} list={classrooms} value={classro
omFilter} setValue={setClassroomFilter} />
        <Text style={page.FilSelTName}>Викладач:</Text>
        <ScheduleFilter style={page.Filter_line} list={teachers} value={teacherFi
lter} setValue={seTeacheFilter} />
        <Text style={page.FilSelTName}>Група:</Text>
        <ScheduleFilter style={page.Filter_line} list={groups} value={groupFilter
} setValue={setGroupFilter} />
      </View>
      {!isFilterSelected ?
        <View><Text style={page.FilSelText}>Виберіть хоча б один варіант фільтрув
ання</Text></View> :
        <ScrollView>
          <View style={{ flex: 1 }}>
            {!filtered.length ? <View><Text style={page.FilSelText}>Результатів н
е знайдено</Text></View> : filtered.map((predmet) => {
              return (
                <View style={page.container}>
                  <View style={page.container_fluid}>
                    <View style={page.container_ta}>
                      <Text style={page.text_time}>{predmet.day}</Text>
                      <Text style={page.text_time_title}>{predmet.time}</Text>
                      <Text style={page.text_time}>{predmet.classroom}</Text>
                    </View>
                    <Text style={page.text}>
                      {/* Группа */}
                      <Text style={page.text}>{predmet.group}</Text>
                    </Text>
                  </View>
                  <Text style={page.text_predmet}>
                    {/* Название предмета */} {predmet.title}
                  </Text>
                  <Text style={page.text_teacher}>
                    {/* Имя преподавателя */} {predmet.teacher}
                  </Text>
                  <Text style={page.text_note}>
                    {/* Имя преподавателя */} {predmet.note}
                  </Text>
                </View>
              );
            })}
          </View>
        </ScrollView>}
    </SafeAreaView>
  );
}

const page = StyleSheet.create({
  container: {
    padding: 14,
    backgroundColor: "#5717B4",
    margin: 10,
    borderRadius: 10,
  },
  container_fluid: {
    flexDirection: "row",
    justifyContent: "space-between",
    flexWrap: "wrap",
```

```
  },
  container_ta: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
  text: {
    fontSize: 18,
    color: "#fff",
  },
  text_time_title: {
    fontSize: 18,
    color: "#fff",
    padding: 6,
    marginRight: 5,
    borderRadius: 5,
    backgroundColor: "#36024B",
  },
  text_time: {
    fontSize: 18,
    color: "#fff",
    marginRight: 5,
    padding: 6,
    borderRadius: 5,
    backgroundColor: "#36024B",
  },
  text_m: {
    fontSize: 20,
    color: "#fff",
  },
  text_predmet: {
    fontSize: 20,
    color: "#fff",
  },
  text_teacher: {
    fontSize: 14,
    color: "#fff",
  },
  text_note: {
    fontSize: 12,
    color: "#fff",
  },
  textH1: {
    fontSize: 30,
    color: "#000",
    textAlign: "center",
    margin: 20,
  },
  SFilter: {
    padding: 8,
  },
  FilSelText: {
    textAlign:"center",
  },
  FilSelTName: {
    paddingBottom: 8,
    fontWeight: 'bold',
  }
});
```

```
//export default App;
```

Файл App.js

```
import React from "react";
import AppNavigator from './navigation/AppNavigator'

export default function App() {
  return (
    <AppNavigator />
  )
}
```