МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КОМПЛЕКСНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система автоматичного складання розкладу. Блок автоматичного формування»

Керівник роботи **Шелехов І.В.**

Студента групи ІН.м. 92 **Руденко М.В.**

СУМИ 2020

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Комп'ютерні Науки»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Руденку Максиму Вячеславовичу

(прізвище, ім'я, по батькові)

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

РЕФЕРАТ

Записка: 47 стор., 20 рис. , 13 джерел.

Об’єкт дослідження — створення автоматичної системи складання розкладу.

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

Методи дослідження — інформаційно-екстремальна інтелектуальна технологія.

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

АНАЛІТИЧНО-ІНФОРМАЦІЙНА СИСТЕМА,
ІНФОРМАЦІЙНО-ЕКСТРЕМАЛЬНА ІНТЕЛЕКТУАЛЬНА
ТЕХНОЛОГІЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

XML - Extensible Markup Language

DOM - Document Object Model

JSC - JavaScriptCore

JSI - JavaScript Interface

API - Application Programming Interface

BaaS - Backend as a Service

SDK - Software Development Kit

RN – React Native

NoSQL – Non relational Structure Query Language

ВСТУП

На даний момент в Україні існує близько 800 ВНЗ. Тобто для студентів, які прагнуть здобути вищу освіту, існує безліч університетів на вибір. Через конкурентоспроможність сектора вищої освіти комфорт навчання став важливим аспектом для студентів при виборі коледжу чи університету разом із зручностями, навчальним планом та можливостями працевлаштування.

Для того, щоб покращити фактор залучення студентів, університети повинні забезпечити, щоб інформація важлива для менеджменту часу студентів, була доступна в потрібному місці та в потрібний час. Мобільний додаток є одним з найзручніших варіантів перевірки розкладу і при цьому майже кожен студент володіє смартфоном за допомогою якого він може перевірити розклад. Саме тому було створено мобільний додаток за допомгою якого можна перевіряти поточний розклад, чи навпаки редагувати його.

Графічний інтерфейс дозволяє з легкістю знайти потрібний пункт в розкладі, тому не потрібно особливих навичок для користування ним.

Проект був реалізований за допомогою JavaScript та фреймворку з відкритим кодом React Native для створення самої программи, та Expo для швидкого тестування та запуску.

1 ЛІТЕРАТУРНИЙ ОГЛЯД

Постановка задачі

Наявність легкодоступного розкладу є доволі зручною особливістю деяких навчальних закладів. Отже з’явилась ідея створити систему для формування розкладу. Початковими задачами були:

1) Вибір фрейворку для реалізації проекту,

2) Обрати бібліотеки які зможуть полегшити чи взагалі дозволити реалізувати поставлене завдання.

3) Проектування та розробка серверної частини додатку.

4) Проектування та розробка мобільного додатку для користувача.

5) Проектування та розробка додатку для адміністратора.

6) Проектування та розробка системи формування розкладу.

1.1 Загальний огляд кроссплатформної розробки

Крос-платформна розробка - це практика побудови програмного забезпечення, сумісного з декількома типами апаратних платформ. Хорошим прикладом кроссплатформенної програми є веб-браузер або Adobe Flash, які виконують те саме, незалежно від того, на якому комп’ютері чи мобільному пристрої ви працюєте.[10, с, 27]

Крос-платформа вважається святим граалем розробки програмного забезпечення - ви можете створити свою кодову базу один раз, а потім запустити її на будь-якій платформі, на відміну від програмного забезпечення, створеного спеціально для певної платформи. Розробники можуть використовувати інструменти, якими вони володіють, такі як JavaScript або C#, для створення программ, які будуть використовуватись на платформах, які стандартно не призначені для платформ на цих мовах. Розробники програмного забезпечення також зацікавлені в цьому, оскільки розробка

продукту з точки зору часових витрат скорочується вдвічі. Основними характеристиками крос-платформної розробки є:

1) Ширша аудиторія, тобто не потрібно вирішувати, на яку аудиторію орієнтуватися, на користувачів iOS або Android, оскільки крос-платформне програмне забезпечення працює на обох, що дає вам доступ до ширшої бази користувачів.

2) Послідовність платформи, адже існують деякі різниці між навігацією та дизайном між iOS та Android, які - у крос-платформній розробці такі проблеми вирішуються за замовчуванням завдяки спільній кодовій базі. Це допомагає створити ідентичність додатку на обох платформах із меншими зусиллями, ніж при нативній розробці.

3) Повторне використання коду, одна з найбільших переваг крос-платформної розробки – вам потрібно створити лише одну базу коду як для Android, так і для iOS. Для розробки нативних додатків потрібно писати код окремо, і для виконання роботи часто потрібні два різні розробники програмного забезпечення - один для iOS та інший для Android.

4) Швидший розвиток, оскільки для обробки iOS та Android потрібна лише одна кодова база, завдяки цьому, розробка продукту відбувається набагато швидше. Крос-платформні додатки побудовані як єдині проекти, хоча вони підтримують різні пристрої.[8]

Основними недоліками крос платформи в свою чергу є те що:

1) Потрібен більший досвід для забезпечення високої продуктивності адже існує поширений міф, що крос-платформні програми працюють гірше, ніж їхні нативні аналоги. У більшості випадків крос-платформні програми можуть працювати за тими ж стандартами, що і власні програми, за умови, що розробники мають достатню кваліфікацію та досвід.

2) Більш складний дизайн коду, оскільки крос-платформні програми повинні працювати на різних пристроях та платформах, це робить написання коду більш складним. Це призводить до більшої кількості роботи для розробників, яким доводиться включати винятки для різних пристроїв та

платформ, щоб врахувати відмінності - особливо, якщо мова йде про більш складні функції.[8]

Аналогами React Native є Flutter та Xamarin. Flutter - випущений в 2017 році Google, який Flutter ідеально підходить для експериментів з новими функціями та виправлення невеликих помилок завдяки функції швидкого оновлення. Це дозволяє розробникам миттєво перевіряти зміни, внесені останніми оновленнями, без необхідності перезапускати програму після редагування вихідного коду. Xamarin - розроблене корпорацією Майкрософт безкоштовне рішення з відкритим кодом яке дозволяє розподіляти 75-90% коду між різними системами. Він написаний на C#, що вимагає від розробників знання мови.

1.2 Фреймворк React Native та його переваги

React Native має можливість взаємодії з обома сферами - потоками на основі JavaScript та існуючими, власними потоками додатків.

Як це працює? React Native використовує так званий "міст". Незважаючи на те, що JavaScript та власні потоки написані абсолютно різними мовами, саме функція мосту робить двонаправлене спілкування можливим. Ось чудова візуалізація концепції мосту:

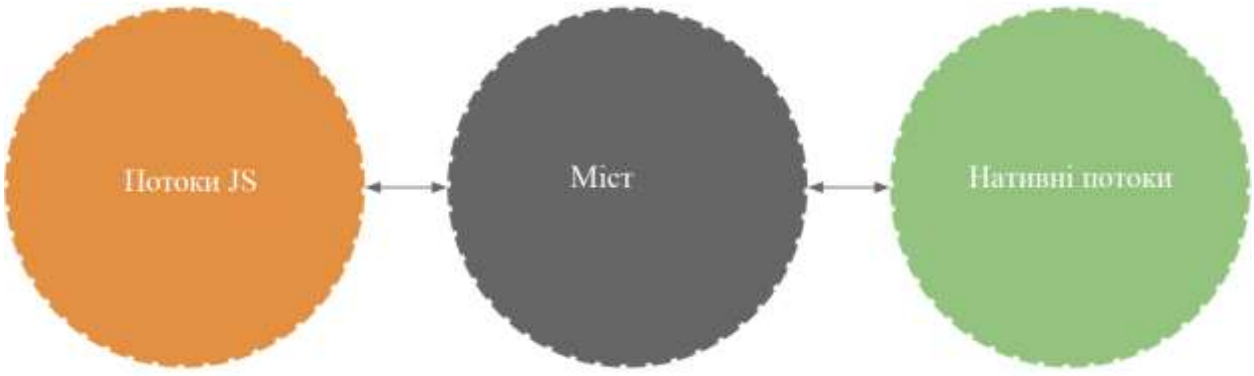

Рисунок 1.1 – *візуалізація концепції мосту*

Це означає, що - якщо у вас вже є власний додаток для iOS або Android - ви все одно можете використовувати його компоненти або перейти до розробки React Native.[5]

Різниця між React Native та іншими крос-платформними рішеннями для розробки (наприклад, Flutter та Xamarin) полягає в тому, що React Native працює на реальних, власних уявленнях та компонентах. Це одна з причин вражаючого успіху React Native.

Можливість повторного використання коду є найбільшою перевагою React Native адже можна інтегрувати 90% власного середовища для повторного використання коду для обох операційних систем.

Ще одна чудова новина - можна використовувати код веб-програми для розробки мобільних додатків, якщо вони обидва використовують React Native. Це також пришвидшує час розробки, оскільки включає попередньо розроблені компоненти, які входять до бібліотеки з відкритим кодом. [8]

React Native - це платформа JavaScript з відкритим кодом, яка дозволяє розробникам внести свої знання у розробку фреймворку, який є у вільному доступі для всіх. Якщо виникає проблема під час розробки програми, то можна звернутися за підтримкою до спільноти, тобто завжди знайдеться хтось, хто зможе допомогти вирішити свої проблеми - це також позитивно впливає на вдосконалення навичок написання коду. React Native розробник використовує React JavaScript для побудови інтерфейсу програми, що дає їй можливість швидше реагувати на дії користувачв із зменшеним часом завантаження, що призводить до загальної кращої взаємодії з користувачем. Завдяки такому інтерфейсу та підходу на основі компонентів, фреймворк ідеально підходить для створення додатків як з простим, так і зі складним дизайном.[10, с.5]

Саме базуючись на вищеописаних перевагах для реалізації проекту було обрано саме React Native.

1.3 Основи взаємодії додатків з сервером

Клієнт-серверна архітектура - це розподілена структура додатків, яка розділяє завдання або робоче навантаження між провайдерами ресурсу, які

називаються серверами, та запитувачами послуг, які називаються клієнтами. В архітектурі клієнт-сервер, коли клієнтський комп'ютер надсилає запит на дані на сервер через Інтернет, сервер приймає запитуваний процес і доставляє клієнтові запитувані пакети даних. Клієнти не діляться жодним із своїх ресурсів.

Клієнт: Коли ми говоримо Клієнт, це означає людину чи організацію, яка користується певною послугою. Так само в цифровому світі Клієнтом є комп'ютер (хост), тобто здатний отримувати інформацію або використовувати певну послугу від постачальників послуг (серверів).

Сервери: Подібним чином, коли ми Сервери, це означає людину чи засіб, який чимось служить. Так само в цьому цифровому світі Сервер - це віддалений комп'ютер, який надає інформацію (дані) або доступ до певних служб.

Отже, в основному Клієнт щось запитує, а Сервер обслуговує його до тих пір, поки він присутній у базі даних.[7]

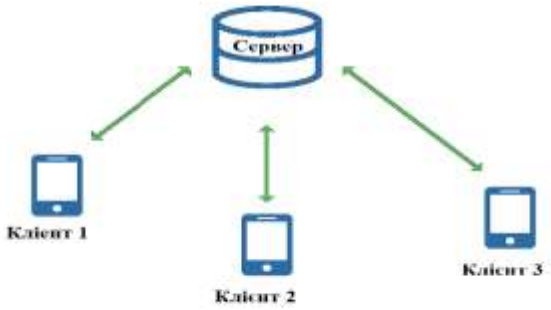

Рисунок. 1.2 - *зображення клієнт-серверної архітектури*

Основними перевагами клієнт-серверної архітектури є:

1) Централізована система з усіма даними в одному місці.

2) Менші витрат на технічне обслуговування, і можливе відновлення даних.

3) Ємність клієнта та серверів можна змінювати окремо.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Загальний огляд існуючих рішень

В інтернеті є досить велика кількість різних календарів з можливостями додавання подій, та спеціальних додатків для окремих навчальних закладів. Деякі з цих додатків можна використовувати без доступу до сервера, тобто лише додаючи різні події та нагадування до календаря, але такі додатки досить примітивні і не несуть такої користі як додадки, які можуть отримати дані розкладу з серверу й автоматичного його сформувати.

Розглянемо деякі готові додакти, для того, що зрозуміти які саме функції потрібно додати.

2.2 Google Календар

Google Календар дозволяє користувачам створювати та редагувати події. Можна активувати нагадування для подій, доступні параметри для типу та часу. Також можна додати місця подій, та можна запросити інших користувачів на події. Користувачі можуть включати дні народження, де додаток отримує дати від контактів Google і щорічно відображає листівки з днями народження та святами. З часом Google додав функціональність, включаючи "Події з Gmail", де інформація про події з повідомлень користувача Gmail автоматично додається до Календаря Google.

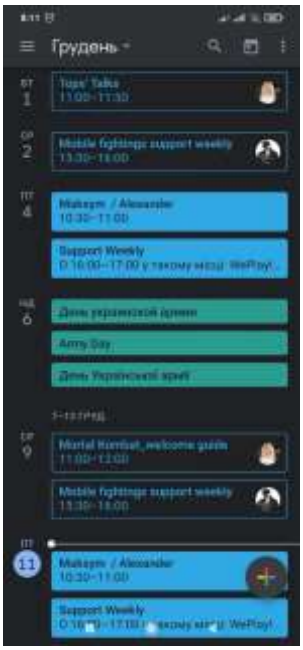

Рисунок 2.1 - *відображення подій в Google Календар*

В більшості випадків гугл календар використовуються або для більш приватного користування де користувач додатку просто додає події, сповіщення, зустрічі, і т.д. до свого календаря. Другий варіант це використання G Suite з власним доменом де зазвичай більшість подій додається однією людиною и яка створює загальні події а інші створюються іншими співробітниками для синхронізації свого графіку

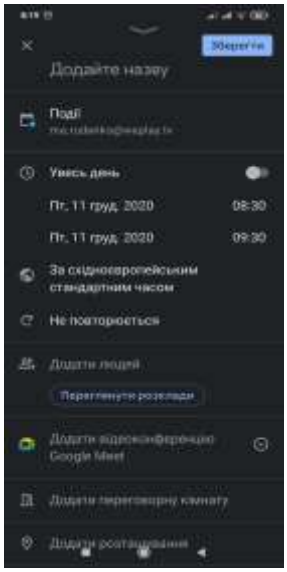

Рисунок 2.2 - *додавання подій в Google Calendar*

2.3 Doodle

Має досить схожий функціонал та дає можливість створювати додаткові календарі паралельно основному, має можливість більш точно налаштовувати свій графік та події але при цьому вимагає дозволу на використання даних календаря мобільного пристрою.

Рисунок 2.3 – *Інтерфейс Doodle*

2.4 Розклад СумДу

Додаток не має функції редагування чи додавання подій в розклад, але такі можливості йому непотрібні так як його завдання полягає в тому щоб швидко та зручно отримати розклад та вивести його у вигляді списку на екран смартфону. Додаток має кілька налаштуваннь: дата, група, викладач, аудиторія, які дозволяють швидко знайти потрібний пункт в розкладі

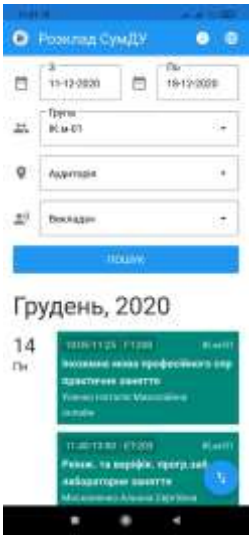

Рисунок 2.4 – *Інтерфейс розкладу СумДу*

2.5 Висновки до розділу

Doodle має найширший функціонал, але при цьому він потребує синхронізації з календарем мобільного пристрою, тому можна сказати що він не зовсім «самостійний»

Google Календар являє собою золоту середину в плані функціоналу але при цьому він не зовсім підходить для того, щоб виконувати функцію простого розкладу.

Розкладу СумДу в свою чергу є оптимальним варіантом так як в ньому немає зайвих функцій та досить просто та зручно реалізовані засоби сортування занять які дозволяють знайти потрібне заняття, саме тому його й було обрано як зразок.

3 Інструменти та засоби реалізації розкладу

3.1 React Native

Так як сам React Native вже було описано у попередніх розділах роботи то в цьому розділі є сенс котороко розповісти про різницю між React Native та React, та принципи роботи першого.

Принципи роботи React Native практично ідентичні React, за винятком того, що React Native не маніпулює DOM через віртуальний DOM. Він працює у фоновому процесі (який інтерпретує написаний розробниками JavaScript) безпосередньо на кінцевому пристрої та взаємодіє з нативною платформою через серіалізаційний, асинхронний та пакетний міст.[11, с. 28]

Компоненти React обгортають наявний нативний код і взаємодіють із власними API через інтерфейс користувача React та JavaScript. Це дозволяє розробляти власні програми цілими новими командами розробників і може дозволити нативним розробникам працювати набагато швидше.

React Native не використовує HTML або CSS. Натомість повідомлення з потоку JavaScript використовуються для маніпулювання власними елментами <view>. React Native також дозволяє розробникам писати власний код такими мовами, як Java для Android та Objective-C або Swift для iOS, що робить його ще більш гнучким.[10, с. 6]

3.2 Середовище розробки

Visual Studio Code - це безкоштовний редактор коду, створений Microsoft для Windows, Linux та macOS. Особливості включають підтримку, рефактиринг, підсвічування синтаксису, інтелектуальне заповнення коду, , та вбудований Git. Можна змінювати тему, комбінації клавіш, налаштування та встановлювати розширення, що додають додаткову функціональність.

Також в редакторі є велика кількість розширень які можуть полегшити роботу з кодом, власне я користувався лише *Visual Studio IntelliCode* та сніпетами для React Native які лише доповнюють код.

Основною перевагою редактору коду є легкий доступ до терміналу та робота з node.js, що дозволяє швидко запускати проект та додавати потрібні бібліотеки, плагіни та інше. На нищенаведеному рисунку зображено інтерфейс редактору.

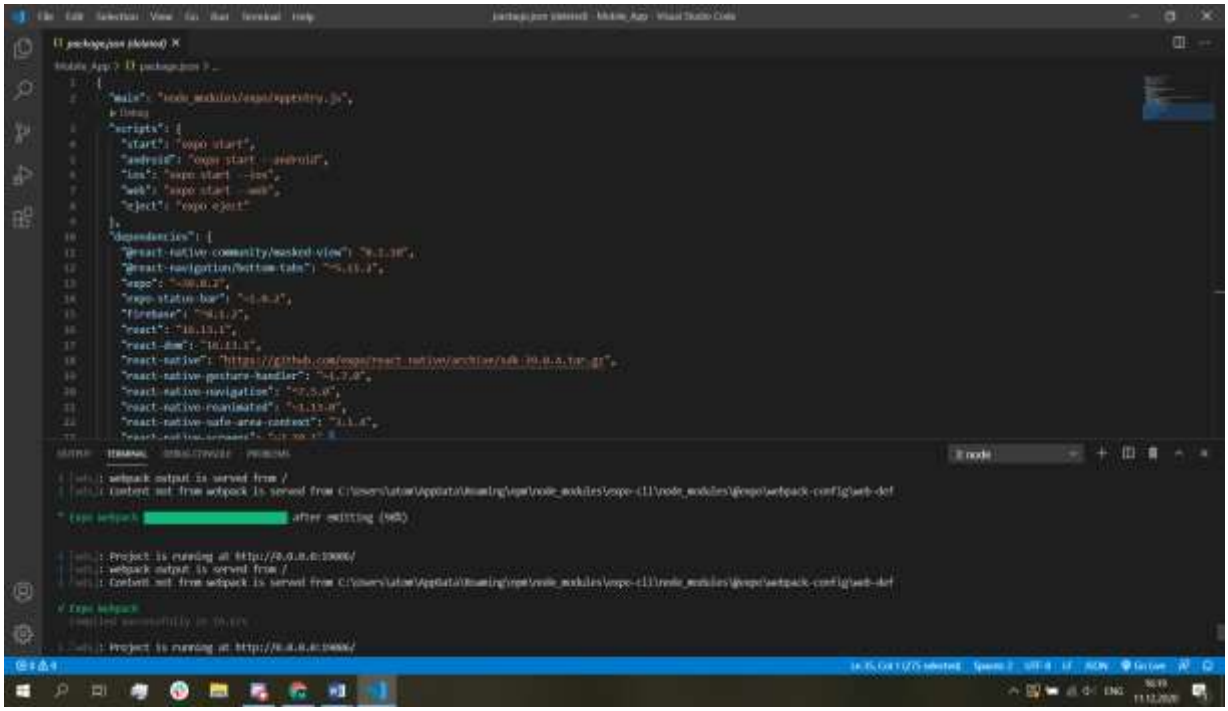

Рисунок 3.1 – *інтерфейс VSCode*

3.3 Структура проекту та архітектура додатків

Для почтатку потрібно було створити сам «чистий» проект в який можна буде додавати потрібні елементи коду для подальшої розробки додатку. Повну структуру проекту можна побачити на нажченавденому скріншоті.

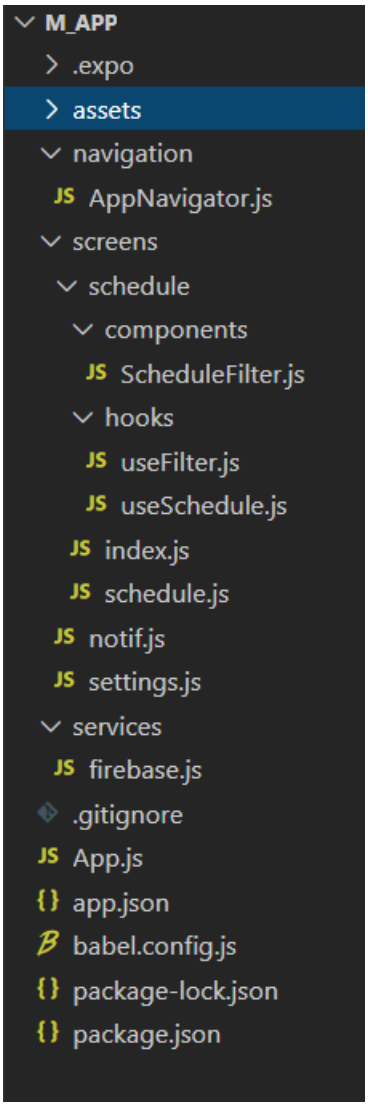

Рисунок 3.2 – *зображення структури проекту*

Також є сенс більш детально розповісти про технічні особливості нового React Narive представлені у 2018 році.

Основною зміною яка може прямо вплинути на написання самого коду це введення хуків, хоча їх використання зовсім не обов’язкове вони спрощують сам процесс розробки. Через природу JavaScript при використанні React Native ми розраховуємо на те що двигун Java Script інтерпретує код так, щоб він працював у нативному додатку. У поточній архітектурі команда React Ntive вирішила використовувати JavaScriptCore (JSC) безпосередньо, згідно з правилами Apple щодо правил iOS "використовувати відповідний фреймворк WebKit та WebKit JavaScript" (JSC використовується WebKit).[6]

Щоб покращити цей елемент (другий блок поточної архітектури React Native), вони вирішили належним чином відокремити вбудований JavaScript, створений від написаного коду, та механізм, який його інтерпретує. Це стало можливим завдяки введенню третього елемента між ними, який називається JavaScript Interface (JSI).[10, с. 28]

JSI не є частиною React Native як такого - це уніфікований, легкий, загальний шар для (теоретично) будь-якого механізму JavaScript. Але, вбудований у нову архітектуру, він дозволяє зробити кілька справді важливих удосконалень. [2]

Перший досить інтуїтивно зрозумілий - потенційно JSC тепер можна було б легше замінити на інші двигуни (або новіші версії JSC, як нещодавно сталося в RN 0.59). Інші варіанти, які ви можете знати, - це ChakraCore від Microsoft та V8 від Google.

Друге вдосконалення - можливо, найважливіша частина всієї реорганізації - полягає в тому, що використовуючи JSI, JavaScript може містити посилання на хост-об'єкти C ++ і викликати на них методи". Це означає, що, це, вирішило стару проблему, React Native, тобто, не буде потреби серіалізувати JSON повідомлення, які потрібно передавати , усуваючи всі затори на мосту.[5]

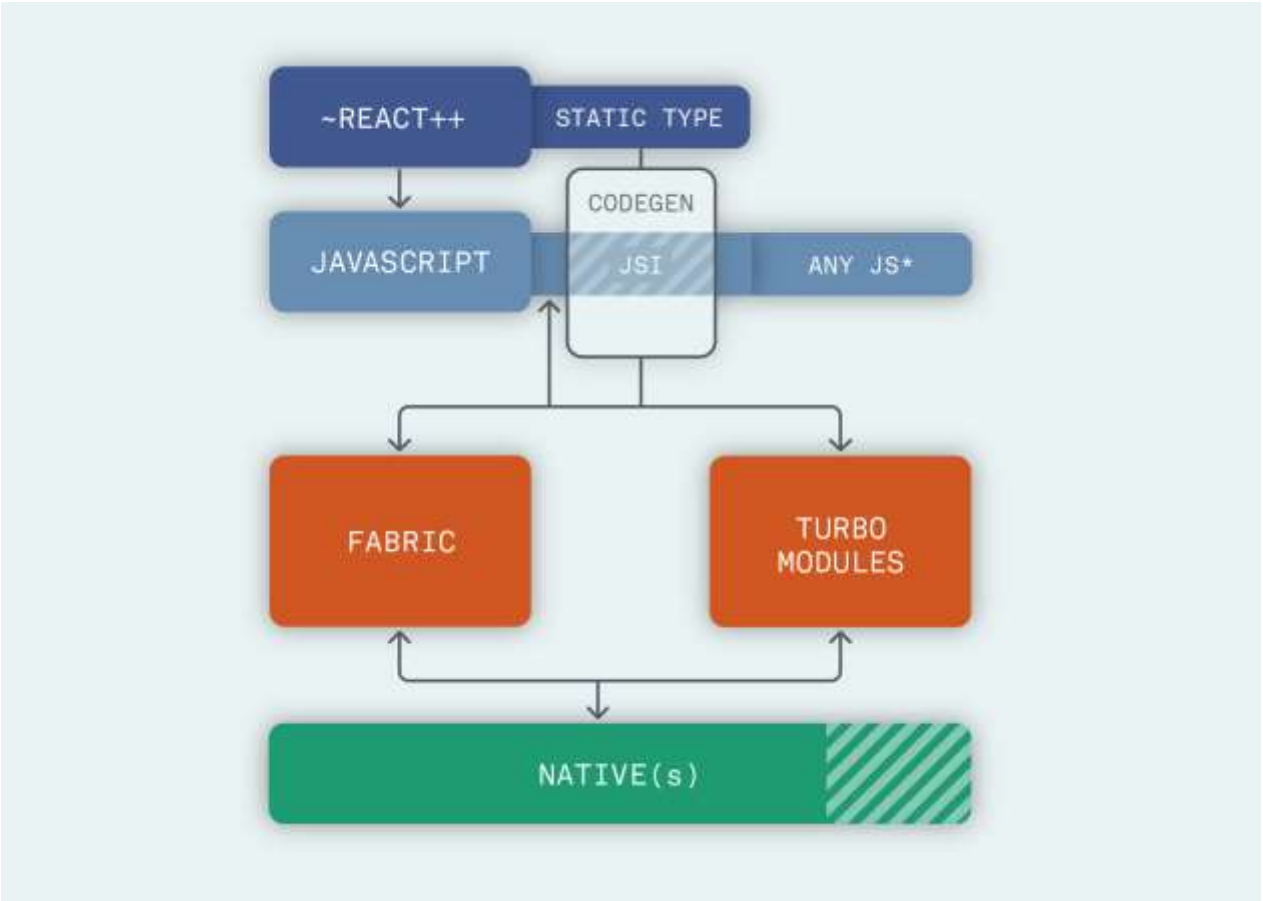

Рисунок 3.3 - *нова архітектура React Native*

3.4 Інструменти розробки

Так як взаємодія двух додатків побудована на FireBase є сенс швидко розповісти про цей сервіс. Вся платформа - це рішення Backend-as-a-Service як для мобільних, так і для веб-додатків, яке включає послуги зі створення, тестування та управління додатками.

Рішення BaaS дозволяють усунути необхідність в управлінні базами даних та утриманні відповідного обладнання. Натомість можна підключити їх до свого додатку через спеціальні API для кожної окремої служби. У випадку з Firebase їх існує 7, які охоплюють весь спектр внутрішніх технологій для програми. Список платформ, з якими інтегрується Firebase, включає Android, iOS, Web та Unity. [13, с. 65]

Cloud Firestore - це ще одна база даних NoSQL яка працює в режимі реального часу, розміщена в хмарному сховищі. На відміну від бази даних Firebase Realtime, Cloud Firestore призначений для корпоративного використання, що передбачає масштабованість, складні моделі даних та розширені параметри запитів. Консоль Firebase може використовуватися для перегляду даних в обох базах даних. Інший спільний момент полягає в тому, що існують SDK для роботи з кодом на стороні сервера обох баз даних. Вони доступні для Python, Node.js, Golang, Ruby, PHP, Java, .NET та C #. В нашому випадку FireBase є досить зручним рішенням, яке також дозволяє в майбутньому розширити та покрищити функціонал додатку, якщо цу буде потрібно.[13. с. 69] .

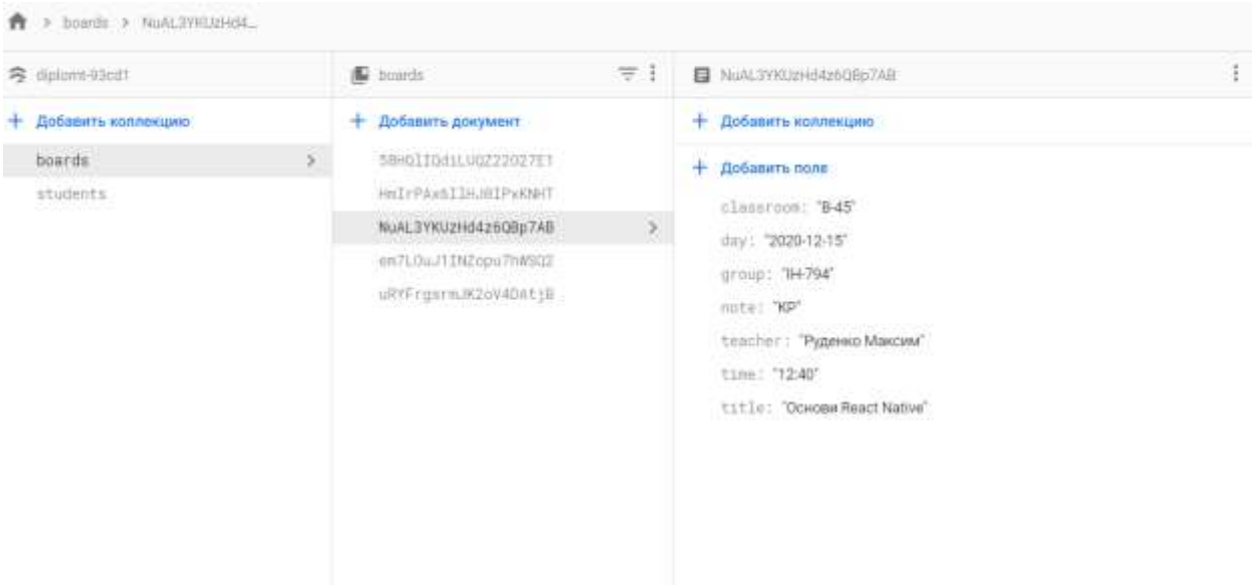

Рисунок 3.4 – *приклад БД*

Інший інструмент, який хотілось би виділити це Expo. Expo - набір інструментів, побудований навколо React Native, щоб допомогти швидко запустити програму. Він спрощує розробку та тестування програми на React Native, надаючи компоненти інтерфейсу користувача та служби, які зазвичай доступні в сторонніх компонентах React Native. Основними перевагами Expo

є швидка та проста ініціалізація та запуск проекту. Також присутня зручна утиліта Expo CLI, яка відкривається у браузері і допомагає перевірити статус програми, пристрої, на яких вона працює, відсканувати QR-код або надіслати посилання електронною поштою, щоб відкрити програму в клієнті Expo.[4]

Expo client - це додаток, який встановлюється з Google Play та Apple Store на телефон. Це дозволяє відкривати проекти під час розробки без збірки через XCode або Android Studio. За допомогою клієнта Expo ви можете надсилати свій додаток іншим на перевірку, що дуже корисно при тестуванні, оскільки ви можете бачити всі зміни в коді в клієнті Expo без створення файлів apk або ipa. [4]

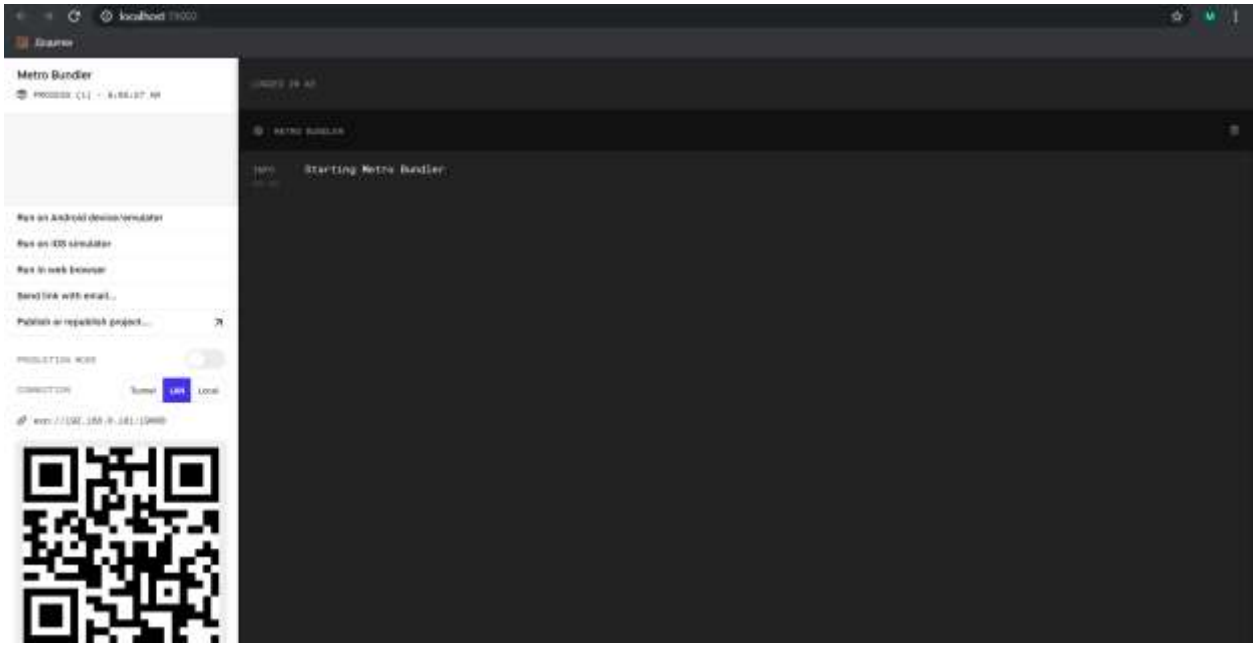

Рисунок 3.5 – *інструменти розробника Expo*

Також можна розробляти програми для iOS без macOS за допомогою пристрою iOS і тестувати їх за допомогою клієнта Expo.

4 Опис програмної реалізації системи управління розкладом

4.1 Опис функцій та принципів роботи адмін панелі

Так як частини проєкту, які я використав для реалізації своєї частини проєкту були реалізовані не мною та не були моїм завданням в проєкті то є сенс лише дуже коротко їх описати. Адмін панель була реалізована на React.js її функціонал включає в себе:

– 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.

Також слід додати що в середовищі Firebase наша головна таблиця має назву “ boards” .В головній колекції знаходяться всі записи, які формуються з адмінпанелі. Всі сформовані записи отримують згенероване імя. 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.

```
var firebaseConfig = {
  apiKey: "AIzaSyAf7AvbzlH4oejMafG1bAuDKAUw8HJPLbc",
  authDomain: "diplomt-93cd1.firebaseapp.com",
  databaseURL: "https://diplomt-93cd1.firebaseio.com",
  projectId: "diplomt-93cd1",
  storageBucket: "diplomt-93cd1.appspot.com",
  messagingSenderId: "764400433559",
  appId: "1:764400433559:web:2904d10bf0c06e53432cd2",
};
```

Рисунок 4.1 – *API FireBase*

4.2 Опис реалізації системи формування розкладу.

На даний момент додаток може лише вивести на екран всю інформацію яка є в базі даних в тому порядку, в якому вони були в неї додані, що взагалі незручно адже користувач буде бачити купу непотрібних йому предметів. Це можна побачити на рисунку 4.1.

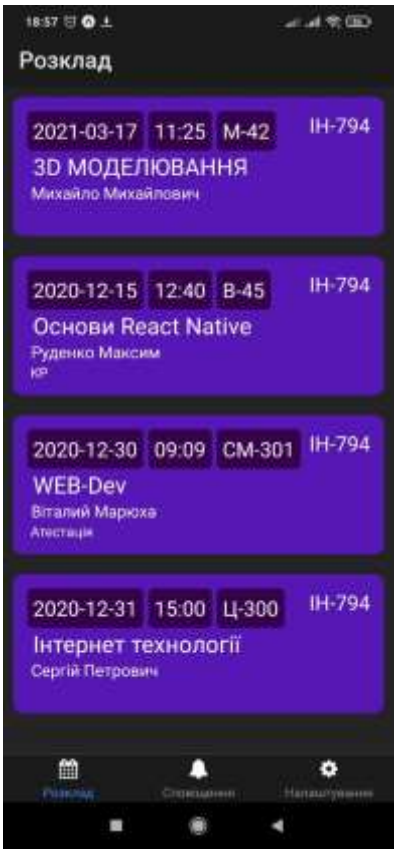

Рисунок 4.1 – *невідсортований розклад*

По-перше потрібно додати на головну сторінку елементи які можуть допомогти користувачу відсортувати потрібні йому предети та прибрати зайві.

Як виявилось в React Native не було нативного елементу Picker який можна було б взяти прямо з докуметації. Тому було прийняте рішення імпортувати одне з готових рішень – компонент React Native Picker. Процесс встановлення компоненту можна побачити на рисунку 4.2.[1],[2]

Рисунок 4.2 – *встановлення React Native Picker.*

Після того, як компонент було встановлено потрібно було його імпортувати для успішного використання в моєму проєкті. Що зображено на рисунку 4.3.

```
import { Picker } from '@react-native-picker/picker';

export const emptyValue = "None";

export function ScheduleFilter({
    list,
    setValue,
    value
}) {
    return <Picker
        selectedValue={value}
        style={{ height: 45, width: '100%', marginBottom: 4, padding: 10 }}
        onValueChange={setValue}>
        <Picker.Item key={emptyValue} label={emptyValue} value={emptyValue} />
        {list.map(el => <Picker.Item key={el} label={el} value={el} />)}
    </Picker>
}
```

Рисунок 4.3 – *імпорт компоненту*

На даному етапі можна сказати, що підготовка та налагодження всіх необхідних інструментів повністю завершена і все, що залишилось це власне написання коду який забезпечить додаток потрібним функціоналом.

Так як до цього додаток виводив на екран пристороюю всі заняття які були в базі даних то потрібно було додати код, який би дозволяв додатку відображати предмети лише тоді, коли було обрано хоча б одну з опцій в фільтрі. Досить невелека проте важлива для правильного функціонування

додатку частина коду, реалізована за допомогою методу массиву some() який з'ясовує, чи містить масив хоч один елемент, для якого зазначена функція callback повертає true. Різницю можна побачити на рис. 4.4.[3]

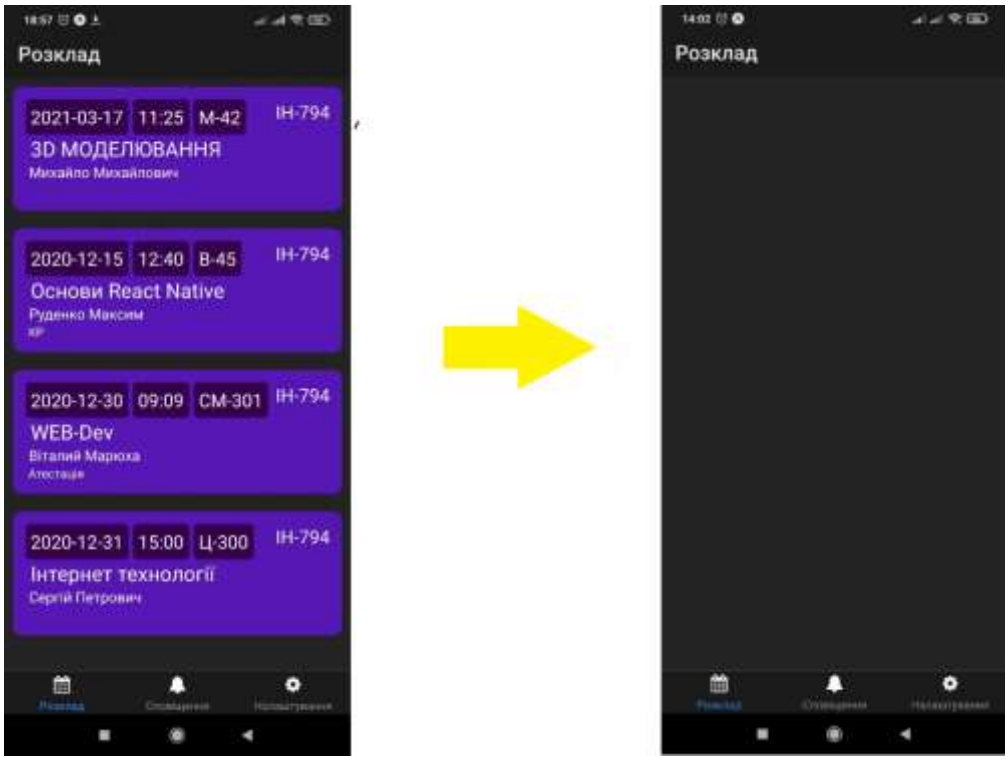

Рисунок 4.4 – *зміна умов відображення розкладу*

Наступним кроком будет просте створення кнопок зі списками параметрів на основі яких будуть обиратися предмети. Після цього потрібно написати функції які будуть додавати до них потрібний функціонал, а саме давати можливість обирати один із параметрів, після чого на екрані будуть відображені всі заняття, які підходять по вибраним параметрам. Результат можна побачити на рисунку 4.5.

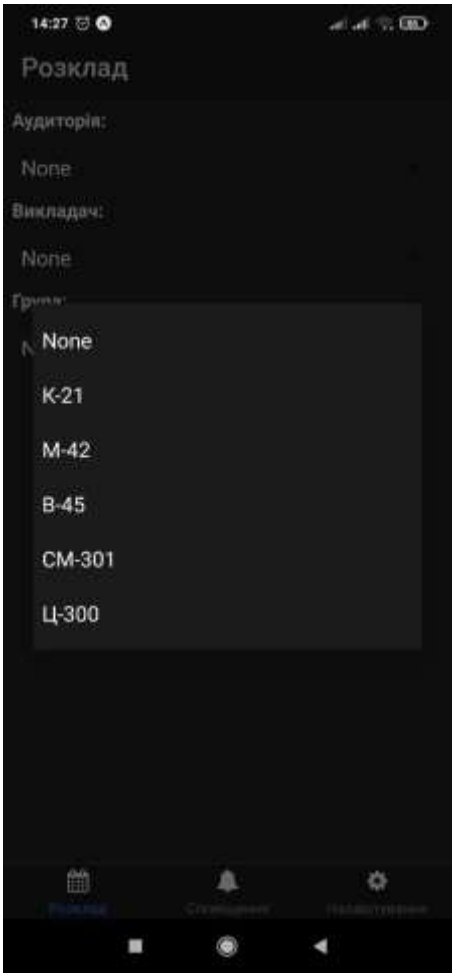

Рисунок 4.5. – *робочий варіант додатку який вже може знаходити потрібні заняття*

Так як майже все поставлені перед проектом завдання вже виконані залишилось лише додати 2 сповщення, які будуть давати користувачу знати чи правильно зараз працює додаток, та що саме йому потрібно робити в деяких випадках. Так як на даний функціонал не досить великий, а відповідно й з принципами роботи розібратись досить легко то я додам лише 2 сповіщення на головному екрані пристрою:

- Сповіщення про те, що потрібно обрати мінімум 1 параметр для того, щоб розклад було відображено.
- Сповіщення про те, що за вибраними параметрами нічого не знайдено.

Приклади роботи цих функцій додатку можна побачити на рисунку 4.6 наведеному нижче.

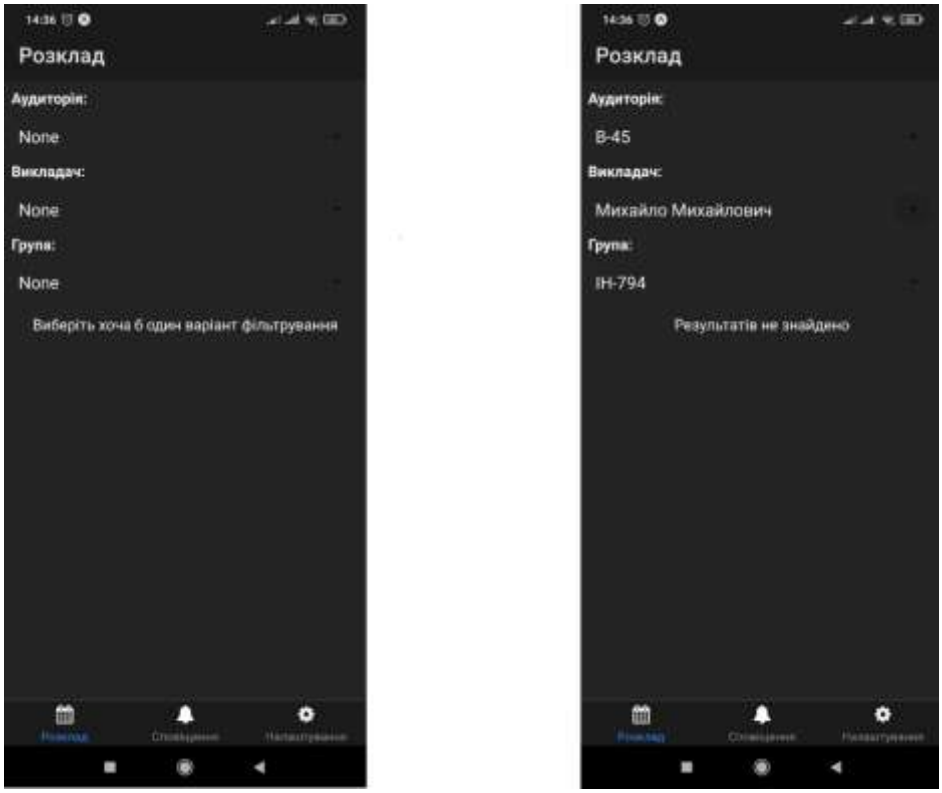

Рисунок 4.6 – *сповіщення про необхідність вибору параметрів, та сповіщення про відсутність результатів*

5 Опис можливостей та інтерфейсу додатку

5.1 Загальний вигляд адмінпанелі

Через те, що панель редагування розкладу, я невід'ємною частиною цього проєкту то є сенс розповісти про неї в цьому розділі, але при цьому, я не буду приділяти їй багато уваги так як вона не була моїм завданням, та вже описана в іншій частні роботи. Загалом нас цікавить лише панель додавання заходів так, як вона має найбільший вплив на роботи моєї частини роботи. Загальний вигляд панелі додавання заходів можна побачина на рисунку 5.1.

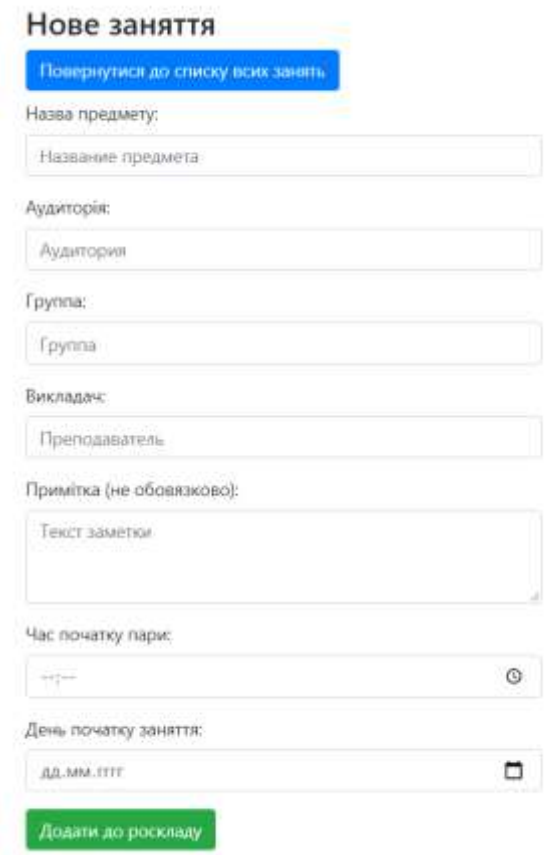

Рисунок 5.1 – *додавання заходів*

Дана панель являє собою сторінку в яку протрібно ввести потрібні параметри, після чого завдання буде додано до розкладу і його можна буде побачити та обрати в частині додатку який формує розклад.

5.2 Загальний вигляд додатку

Так як створення додатку вже повністю завершене то можна продемнострувати те, як він працює та більш детально розповісти про інтерфейс в окремому розділі. Використання додатку повністю зводиться до однієї сторінки де відображається 3 елементи:

1. Аудиторія
2. Викладач
3. Група

Поки жодний з елементів не обраний то не екрані нічого не буде відображатись, якщо обрати один з доступних параметрів то відповідно будуть показані всі події які мають за властивість один з цих параметрів, додавання більшої кількості параметрів відповідно буде зменшувати кількість подій, які будуть відображатись на екрані, у випадку, якщо за обраними параметрами немає підходящих подій то відповідно з’явиться повідомлення яке вкаже не неправильний вибір чи відсутність такої події.

Вищеописані принципи роботи можна побачити на ничженаведеному Рис 5.2

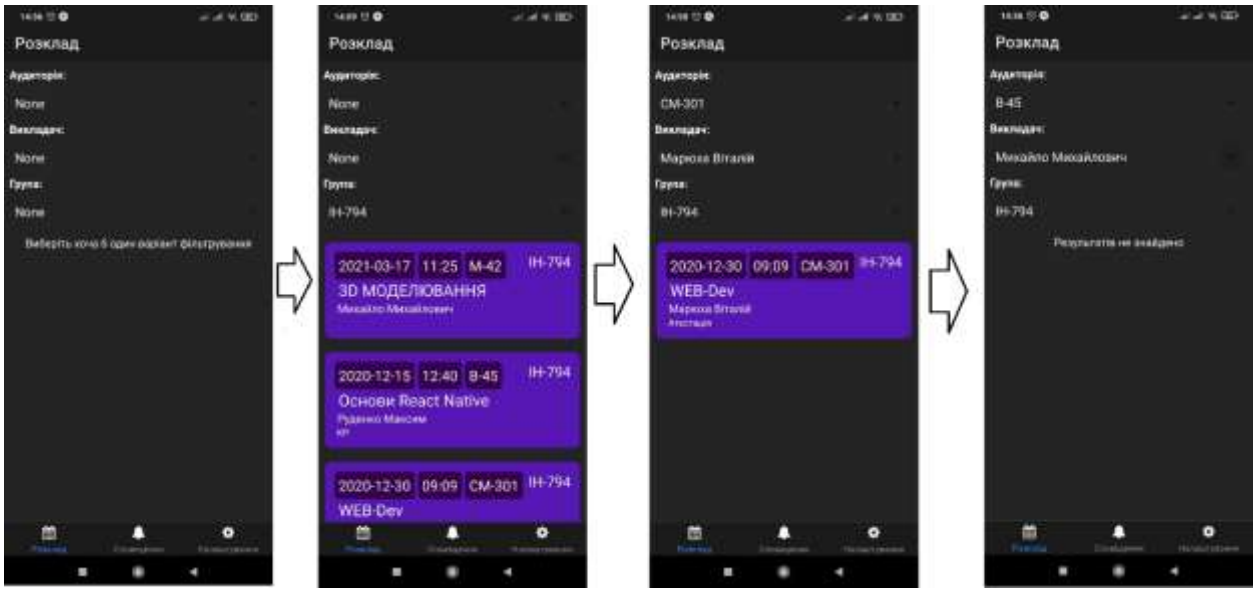

Рисунок 5.2 – *візуальний приклад принципів роботи додатку*

ВИСНОВКИ

В процесі виконання цієї роботи було обгрунтовано важливість реалізації даного проекту та його потенційна користь також був успішно досліджений та освоєний фреймворк для створення мобільних додатків React Native, та деякі інші інструменти та методи розробки мобільних додатків такі як Expo Client, та FireBase які змогли суттєво поліпшити та пришвидишити процесс розробки та тестування додатку.

Також були досліджені існуючі популярні рішення, після чого на основі результатів дослідження було обрано найкращий варіант який було взяти за приклад.

Результатом роботи стало створення мобільного додатку завдяки якому обираючи різні параметри можна швидко перевіряти наявний розклад та знаходити потрібні подіїї. Також було реалізовано панель редагування, та додавання розкладу завдяки якій можна відповідно редагувати та додавати події.

Через вдалий вибір інструментів та архітектури є досить реальна можливість розширити функціонал додатку, якщо буде така потреба. А набутий досвід та розуміння принципів роботи програми може допомогти в виправленні помилок, які не були виявлені при попередніх тестуваннях.

В підсумку можна сказати що всі поставлені початкові завдання були виконані, а всі проблеми з якими ми зіткнулися на шляху до завершення проєкту були вирішені.

СПИСОК ЛІТЕРАТУРИ

1. React Native Picker [Electronic resource]. 2020 URL:https://github.com/react-native-picker/picker

2. React Native Database [Electronic resource]. 2020. URL:https://reactnative.directory/

3. MDN Web Docs. Array.prototype.some() [Electronic resource]. 2020. URL:https://developer.mozilla.org/uk/docs/Web/JavaScript/Reference/Global_Objects/Array/some

4. Expo Docs. Errors and Debugging. Redbox Errors and Strack Traces[Electronic resource]. 2020. URL:https://docs.expo.io/

5. The New React Native Architecture Explained. Part One: React and Codegen [Electronic resource]. 2020. URL:https://formidable.com/blog/2019/react-codegen-part-1/

6. Learn React Js. Представляем Хуки [Electronic resource]. 2020. URL:https://learn-reactjs.ru/core/hooks/introduction

7. Client-Server Model by Haroon Shakirat Oluwatosin School of Computing Universiti Utara Malaysia Kedah, Malaysia [Electronic resource]. Feb 2014. URL: https://www.researchgate.net/profile/Shakirat_Sulyman/publication/271295146_Client-Server_Model/links/5864e11308ae8fce490c1b01/Client-Server-Model.pdf

8. Native Vs Cross-Platform Development: Pros & Cons Revealed by Anastasiya Marchuk [Electronic resource]. 2020.URL:https://uptech.team/blog/native-vs-cross-platform-app-development

10. React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications 2nd ed. Edition by Akshat Paul, Abhishek Nalwaya p. 25 - 36.

11. JavaScript Everywhere: Building Cross-Platform Applications with GraphQL, React, React Native, and Electron 1st Edition, Kindle Edition by Adam D. Scott Format: Kindle Edition p.46 - 52

12. Learning React Native by Bonny Eisenmann. p. 73 – 79.

13. The Definitive Guide to Firebase. The Firebase Realtime Database by Laurence Moroney p. 51-71

ДОДАТОК

Додаток А

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 is{" "}
                <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>Аудитория</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 />
  )
}
```