МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

“До захисту допущена”
Зав. кафедри комп'ютерних наук
та прикладної математики
Турбал Ю.В.
“____” _____ 2024р.

КВАЛІФІКАЦІЙНА РОБОТА

Програмна реалізація автоматизованої системи пропуску

Виконав: Гадземан Максим Сергійович

група ІПЗ-41 _____
підпис

Керівник: к.ф.-м.н., доцент кафедри комп’ютерних наук та прикладної математики, Прищепа О.В.

_____
підпис

Рівне-2024

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 51 с., 7 рисунків, 13 джерел.

Мета роботи: розробка застосунку для навчального закладу на платформах Android та IOS з автоматизованою системою пропуску.

Об’єкт дослідження: Мобільні застосунки для освітніх закладів.

Предмет дослідження: Розробка мобільного застосунку для освітнього закладу з вбудованою системою пропуску.

Методи вивчення: React, React Native, Expo, TypeScript

Проведено аналіз ринку та визначено основні функції застосунку. Створено технічне завдання з переліком вимог. Проаналізовано можливі системи для розробки мобільних застосунків на платформи IOS та Android. Здійснено розробку та подальше тестування застосунку на основі вибраної системи.

Ключові слова: цифровізація, React Native, Expo, мобільний застосунок, система пропуску, навчальний заклад.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ТЗ – технічне завдання

API – прикладний програмний інтерфейс (Application Programming Interface)

CLI – інтерфейс командного рядка (Command-line interface)

DOM – об'єктна модель документа (Document Object Model)

HTML – мова розмітки гіпертексту (HyperText Markup Language)

SDK – набір для розробки програмного забезпечення (Software Development Kit)

ВСТУП

У сучасному світі студенти стикаються з численними викликами, одним з яких є необхідність витрачати значну кількість часу на фізичне переміщення до університету для отримання актуальної інформації. Зокрема, розклад занять, новини та оголошення часто розміщуються на стендах, що вимагає особистої присутності студента. Це не лише знижує ефективність використання часу, але й обмежує можливості для навчання та саморозвитку. В умовах, коли кожна хвилина на вагу золота, такі перешкоди стають суттєвим бар’єром на шляху до оптимізації навчального процесу та особистісного зростання. Тому постає нагальна потреба у впровадженні більш ефективних та зручних способів комунікації, які дозволяють студентам оперативно отримувати необхідну інформацію без додаткових витрат часу та зусиль.

Через наявність такої проблеми було взято за мету створити мобільний застосунок для студентів, який забезпечує комплексний підхід до отримання необхідної інформації. Він дозволить студентам в режимі реального часу переглядати актуальний розклад занять, включаючи зміни та оновлення, що дозволить завжди бути в курсі майбутніх лекцій, семінарів та інших навчальних заходів. Крім того, він матиме всі актуальні новини, щоб студент завжди був обізнаний про життя коледжу. Також однією з цікавих функції буде наявність унікального бар-коду, який буде виступати в ролі пропускного ключа до будівлі коледжу, що підвищить безпеку студентів. За допомогою цього пропуску студент також зможе відзначати свою присутність на заняттях

Сам застосунок розроблено за допомогою фреймворку react-native та бібліотеки expo, які сфокусовані на створенні функціональних та інтуїтивно зрозумілих застосунків з максимальним зворотним зв'язком з користувачем. Зокрема фреймворк має велику кількість додаткових бібліотек, які полегшать реалізацію всього функціоналу застосунку.

Отже, даний застосунок має за мету знищити будь-які перешкоди для студента в отриманні актуальної інформації, переносячи її в новітній та зручний інтерфейс з естетично приємним дизайном, який завжди знаходиться в них під рукою. Це дозволить студентам без зайвих зусиль, затрат часу швидко та зручно звертатися до необхідної інформації, підвищуючи ефективність навчання та сприяючи їхньому успіху.

РОЗДІЛ 1
ОГЛЯД ЦИФРОВІЗАЦІЇ ОСВІТНЬОГО ПРОЦЕСУ

1.1. Проблеми та перспективи

Навчальні заклади, що не використовують мобільні застосунку, стикаються з рядом суттєвих проблем, які можуть негативно впливати на ефективність освітнього процесу та комунікацію між усіма учасниками навчання. Однією з головних проблем є обмежений доступ до інформації. Без мобільних застосунків студенти та викладачі не мають зручного способу оперативно отримувати розклади, навчальні матеріали, результати тестів та інші важливі дані. Це може призводити до ситуацій, коли студенти запізнюються або пропускають заняття, оскільки не мають актуальної інформації під рукою.

Крім того, відсутність інтерактивних інструментів для навчання може знижувати зацікавленість студентів у навчальному процесі. Мобільні застосунки часто пропонують інтерактивні завдання, вікторини, симуляції та інші елементи гейміфікації, що роблять навчання більш захопливим та стимулюючим. Без таких інструментів студенти можуть втрачати мотивацію, що негативно впливає на їхні результати.

Іншою значною проблемою є інформаційна перевантаженість. Викладачі та адміністративний персонал змушені використовувати безліч різних платформ та інструментів для управління навчальним процесом, що може призводити до плутанини та втрати важливої інформації. Наприклад, студенти можуть отримувати розклади на електронну пошту, завдання через іншу платформу, а результати тестів – через ще одну. Така фрагментація ускладнює процеси та створює додаткове навантаження на всіх учасників освітнього процесу.

До цього додається складність комунікації. Оперативна та ефективна комунікація між студентами, викладачами та адміністрацією є важливим елементом успішного навчального процесу. Мобільні застосунки дозволяють

швидко обмінюватися повідомленнями, отримувати сповіщення про зміни у розкладі чи важливі події. Без них учасники освітнього процесу можуть відчувати нестачу комунікації, що призводить до зниження ефективності взаємодії.

У випадку необхідності переходу на дистанційне навчання відсутність мобільних застосунків ще більше ускладнює організацію навчального процесу та взаємодію зі студентами. Мобільні застосунки можуть значно полегшити цей процес, надаючи можливість проводити онлайн-лекції, обмінюватися матеріалами та виконувати завдання дистанційно. Без цих технологій навчальні заклади змушені шукати інші, часто менш ефективні способи забезпечення навчального процесу.

До всього цього додаються труднощі з оцінюванням та відстеженням прогресу. Відсутність автоматизованих систем оцінювання ускладнює процес відстеження прогресу студентів та надання зворотного зв'язку. Мобільні застосунки можуть надавати детальну інформацію про результати тестів, домашніх завдань та інших видів оцінювання, що дозволяє студентам та викладачам краще розуміти, на яких аспектах слід зосередитися.

Впровадження мобільних застосунки у навчальні заклади, які раніше їх не використовували, відкриває перед установами широкий спектр нових можливостей та перспектив для покращення навчального процесу та управління ним. Цей крок може значно підвищити ефективність, зручність та доступність освіти для всіх учасників освітнього процесу, а також централізувати потік інформації в навчальному закладі.

1.2. Мобільні застосунки для навчальних закладів

Ми живемо в цифрову епоху, коли технології революціонізували кожен аспект нашого життя, в тому числі й систему освіти. У сучасному швидкоплинному світі традиційне університетське середовище зазнає значних

змін, поступаючись місцем концепції цифрового кампусу. Однією з частиною цієї концепції є застосунок для смартфона, за допомогою якого студенти зможуть слідкувати за своїми успіхами, а також мати певний інтерактив з самим навчальним закладом. Якщо подивитися на статистику, то на цей час 90% всіх студентів світу володію смартфоном тим паче 86% свого часу вони проводять за використанням мобільних застосунків, з чого можна зробити висновок, що студент і його смартфон нерозлучні. Через це наявність застосунку з брендом навчального закладу, в якому вони навчаються, є найкращим способом привернути їхню увагу до життя закладу, а також зробити їхній досвід більш приємним та запам'ятовувальним. Але сама наявність в студента смартфона не є єдиним плюсом застосунку, таких плюсів багато і ми б хотіли більш детально розібрати кожен з них.

Нижче наведено перелік певних переваги, які отримують навчальні заклади, створюючи мобільні застосунки:

1. Залучення учнів через мобільні пристрої:

Мобільні застосунки полегшують учням доступ до контенту, виконання завдань та співпрацю з однолітками, а також допомагають налагодити міцніші стосунки з викладачами та спільнотою кампусу.

Мобільні застосунки – це чудовий спосіб залучити учнів до навчального процесу. Якщо учень пропустив заняття, він може легко отримати інформацію, яку пропустив, декількома дотиками пальців.

2. Будьте доступними будь-де і будь-коли

Хоча студентський портал слугує хорошим засобом комунікації, саме фактор доступності робить мобільний застосунок набагато ефективнішим. Студенти матимуть доступ до всіх ресурсів навчального закладу в будь-який час. Що ще важливіше, що під час надзвичайних ситуацій та ситуацій, які потребують

негайної уваги, мобільний застосунок набагато зручніший у порівнянні з підтримкою через браузер.

3. Оптимізація каналів комунікації

Навіщо витрачати час на розробку та друк брошур, памфлетів або надсилання електронного листа, сподіваючись, що студенти його побачать, якщо є можливість легко завантажити контент прямо в мобільний застосунок? Заощаджуйте багато часу та ресурсів, одночасно легко відстежуючи рівень залученості студентів. Ще більшою перевагою мобільних застосунків є можливість надсилати персоналізований контент обраним демографічним групам студентів і таким чином стимулювати їхню залученість.

4. Автоматизуйте рутинні адміністративні дії

Призначення зустрічей, планування заходів, а потім відстеження процесу реєстрації може бути досить неспокійним. Навіть після кількох спроб розсилки оголошень та флаєрів, ви все одно можете не отримати необхідної реєстрації чи відвідуваності. Мобільне рішення може не лише забезпечити швидку відповідь, але й заощадити час та зусилля, автоматизувавши процес.

5. Миттєва комунікація зі студентами

При наявності власного мобільного застосунку під брендом навчального закладу, відпадає потреба турбуватися про оголошення в останню хвилину, сповіщення або будь-яку інформацію, яку потрібно негайно донести до всіх студентів або цільових груп і все завдяки можливості надсилати пуш-сповіщення для обраної цільової групи в їхні мобільних застосунки за лічені секунди.

6. Будьте завжди на зв'язку з онлайн-службою підтримки та базою знань

За допомогою мобільного застосунку можна автоматизувати весь процес пошуку студентів, заощадивши багато часу та зусиль. Потенційні та зареєстровані студенти можуть легко створювати сервісні квитки щодо питань,

які потребують підтримки, через мобільний застосунок, в будь-який час і в будь-якому місці. Вони також матимуть доступ до бази знань навчального закладу, поширених запитань та посилань на всю корисну інформацію, яка їм може знадобитися.

7. Підвищити рівень залученості студентських груп та спільнот

Завдяки власному мобільному застосунку під брендом коледжу можна створювати студентські групи та спільноти, як для навчання, так і для дозвілля, які залучатимуть студентів, заохочуючи їх до участі в режимі реального часу.

Це покращує досвід роботи в класі. Студенти зможуть легко ділитися контентом, ставити лайки та коментувати, що сприяє більшій соціальній взаємодії.

8. Швидкий доступ до релевантної інформації про студентів

Хоча система SIS/LMS допомагає зберігати всю інформацію, все одно не завжди можливо знайти потрібну інформацію в потрібному місці в потрібний час. Саме тому потрібна система, яка інтегрується з системами SIS/LMS, яка зіставляє ці дані та надає змістовну інформацію. Ці дані допоможуть швидко отримати будь-яку інформацію про окремого студента одним клацанням миші та бути завжди готовими простягнути руку допомоги.

9. Централізовану систему для всіх потреб студентів

Однією з головних переваг є те, що вся комунікація, контент, співробітники та студентський каталог доступні в одній централізованій системі. Замість того, щоб використовувати кілька платформ для різних видів комунікації та інформації, є можливість використовувати одне рішення для всіх потреб у залученні студентів. Від створення запиту до відповіді на опитування та анкетування і все це контролюється з однієї адмін панелі.

В підсумку, до того чому кожен навчальний заклад повинен мати свій мобільний застосунок, можна сказати що студенти не тільки можуть бути завжди в дорозі та отримувати миттєвий доступ до всіх найважливіших шкільних предметів, але й користуватися більшою гнучкістю, ніж раніше. Оскільки вся інформація зосереджена в одному місці, студенти можуть використовувати лише одну платформу для розв'язання багатьох питань. Наприклад, вони можуть отримати доступ до ресурсів і можливостей, які зазвичай доводиться шукати в різних місцях на території школи або за її межами.

Що більше, застосунки для навчального закладу можуть допомогти студентам налагоджувати та підтримувати соціальні зв'язки зі своїми однолітками, навіть через онлайн-платформи. Багато застосунків використовують групові дії, щоб об'єднати студентів зі схожими інтересами, знайти спільні захоплення тощо[1-3].

РОЗДІЛ 2
РОЗРОБКА ЗАСТОСУНКУ

2.1. Постановка задачі

Перед початком розробки застосунку було вирішено описати його можливості та вигляд у форматі “Технічного завдання”. Технічне завдання (ТЗ) – це документ, який визначає основне призначення виробу, показники його якості, техніко-економічні характеристики, а також спеціальні вимоги до виробу, обсягів та етапів розробки. Дане ТЗ було створено разом з замовниками від навчального закладу та включає наступні пункти

1) Загальний огляд
    - Назва проєкту: MyPass
    - Тип застосунку: Мобільний застосунок для управління завданнями
    - Платформа: Android та iOS
2) Опис проєкту. MyPass – це застосунок, що спрямований на допомогу користувачам (студентам та викладачам) дізнаватися свій розклад, переглядати власну інформацію та отримувати доступ до приміщень в коледжі відповідно до розкладу.
3) Вимоги до функціональності

- Авторизація
    - Створити скелет для авторизації користувача для подальшої імплементації з “Google Oauth2”
    - Отримання даних користувача з бази даних при авторизації.
    - Зберігання ключа сесії користувача в зашифроване сховище
- Розклад
    - Користувач може переглядати свій розклад залежно від його групи(якщо це студент) або пов’язані з ним події(якщо це викладач)
    - Кожна пара/подія повинна мати опис

- Профіль. Сторінка “Мій профіль” – виводить ПІП користувача, групу та фотографію профілю
- Карта доступу. Інтеграція квитка доступу в застосунок у форматі баркоду
- Новини. Користувач має мати можливість переглядати актуальні новини про навчальний заклад.
- Технічні вимоги
    - Мова програмування: TypeScript.
    - Дизайн інтерфейсу: Мінімалістичний, зручний для користувача, використання артбуку коледжу.
    - Сумісність: Android 6.0 і вище, iOS 12 і вище.

4) Терміни виконання
    - Розробка розраховано 4 місяці.
    - Тестування буде проведено впродовж літа протягом 2-3 місяців.

2.2. React

React – це бібліотека JavaScript для створення користувацьких інтерфейсів, розроблена Facebook і вперше випущена у 2013 році. Вона є потужним інструментом для розробки односторінкових додатків, забезпечуючи високу швидкість відображення та інтерфейсну інтерактивність. Головна концепція React полягає у використанні компонентів, що робить код написаний з його використанням організованим і легким у підтримці.

React також відомий своїм одностороннім потоком даних, який полегшує відстеження стану та управління даними в додатку. Односторонній потік даних означає, що дані передаються в одному напрямку, від батьківського компонента до дочірнього. Це робить логіку застосунок передбачуваною і простішою для розуміння. Крім того, React підтримується великою спільнотою розробників, що забезпечує доступ до численних готових рішень, бібліотек і інструментів для розробки [4].

2.2.1. Компоненти React

Основними будівельними блоками будь-якого React проєкту є компоненти. Вони дозволяють розробникам розбивати інтерфейс користувача на незалежні, багаторазові частини, які можна легко підтримувати та оновлювати. Кожен компонент інкапсулює власну логіку та відображення, що робить код більш організованим і зрозумілим.

В React існують два основних типи компонентів: функціональні компоненти та класові компоненти. Функціональні компоненти – це прості JavaScript функції, які приймають аргументи (props) і повертають React елементи, що описують, що має бути відображено на екрані. З появою хуків функціональні компоненти стали більш популярними, оскільки вони можуть тепер мати стан і інші можливості, раніше доступні тільки класовим компонентам. Класові компоненти – це компоненти, визначені за допомогою ES6 класів. Вони є більш повнофункціональними та можуть використовувати методи життєвого циклу React. Класові компоненти містять метод render, який повертає React елементи. Крім того, вони можуть мати власний стан (state) і доступ до різних методів життєвого циклу.

Props (властивості) – це механізм передачі даних від одного компонента до іншого. Вони є незмінними та передаються як аргументи до компонента. Props дозволяють налаштовувати компоненти та робити їх більш гнучкими. Наприклад, можна створити кнопку і передати їй текст через props, що дозволить використовувати один і той самий компонент кнопки з різним текстом у різних частинах застосунку.

State (стан) – це динамічні дані, що зберігаються в компоненті та можуть змінюватися під час його життєвого циклу. Стан зазвичай використовується для зберігання інформації, яка може змінюватися у відповідь на дії користувача або інші фактори. Коли стан змінюється, компонент автоматично перерендрується,

щоб відобразити ці зміни. В класових компонентах стан задається в конструкторі, а в функціональних компонентах використовується хук useState.

В незалежності від типу компоненти, вони мають повертати певний код який буде показувати які HTML елементи мають попасти в DOM. Для цього React має в собі вбудований JSX, розширення синтаксису JavaScript, яке дозволяє писати HTML-подібний код в JavaScript. Він має відмінності від оригінального HTML(наприклад для того, щоб присвоїти клас елементу потрібно використовувати конструкцію className замість class) а також певні додаткові функції які спрощують його використання. Однією з таких функцій є можливість вставити JS код використовуючи фігурні дужки, щоб передати дані для відображення.

Також компоненти в React мають кілька етапів життєвого циклу, які можна умовно розділити на три фази: монтування, оновлення і розмонтування.

- **Монтування**: Фаза, коли компонент створюється і додається до DOM. У класових компонентах ця фаза включає виклики методів constructor, componentDidMount та render.
- **Оновлення**: Фаза, коли компонент оновлюється у відповідь на зміни в props або стані. Включає методи shouldComponentUpdate, componentDidUpdate та render.
- **Розмонтування**: Фаза, коли компонент видаляється з DOM. Включає метод componentWillUnmount.

Розробники часто розділяють компоненти на контейнерні та презентаційні. Презентаційні компоненти відповідають за відображення інтерфейсу користувача і зазвичай не містять логіки, пов’язаної з бізнес-процесами або управлінням станом. Контейнерні компоненти, навпаки, відповідають за взаємодію з логікою застосунку та управлінням станом, передаючи дані презентаційним компонентам через props[5].

2.2.2. Віртуальний DOM

Однією з головних переваг React є його швидкість та ефективність, що досягається завдяки використанню віртуального DOM. Якщо не вдаватись до деталі, то віртуальний DOM – це легка копія реального DOM, яка існує тільки в пам'яті комп'ютера. Коли ви вносите зміни до вашого застосунку, React спочатку оновлює віртуальний DOM, а не реальний DOM. Це дозволяє React швидко визначити, які саме частини реального DOM потрібно оновити.

Процес виглядає так: коли стан або пропси компонентів змінюються, React оновлює віртуальний DOM, порівнюючи його з попередньою версією. Цей процес називається "діффінг" (diffing). React виявляє зміни, які відбулися, і створює список оновлень, необхідних для синхронізації віртуального DOM з реальним DOM. Після цього React ефективно застосовує ці оновлення до реального DOM.

Цей підхід дозволяє мінімізувати кількість змін, які потрібно внести до реального DOM. Оскільки взаємодія з реальним DOM є відносно повільною операцією, мінімізація таких змін значно покращує продуктивність застосунку. Віртуальний DOM також дозволяє React виконувати батчинг (batching), який замість того, щоб перемальовувати компоненти при кожній зміні станів бере і запам’ятовує всі оновлення які відбулись після останнього відображення і викликає їх за один цикл рендеру, що знову ж таки зменшує кількість операцій з реальним DOM.

2.2.3. React Hooks

Одним з основних елементів функціональних компонентів які зробили їх зручнішими та ефективнішими за класові стали хуки які з’явилися в React версії 1.68. Вони дозволяють використовувати стан і інші можливості React у функціональних компонентах, які раніше були доступні тільки в класових компонентах. Хуки роблять код чистішим, зручнішим і більш модульним,

дозволяючи розробникам легко додавати складну логіку в компоненти без використання класів.

Основними хуками є useState і useEffect. Хук useState дозволяє додати стан до функціональних компонентів. Виклик useState повертає масив з двох елементів: поточне значення стану і функцію для його оновлення. Це дозволяє компонентам динамічно змінюватися у відповідь на дії користувача. Хук useEffect виконує побічні ефекти у функціональних компонентах, такі як запити до API, налаштування підписок або маніпуляції з DOM. Він об'єднує функціональність методів життєвого циклу класових компонентів (componentDidMount, componentDidUpdate, componentWillUnmount) в один API. useEffect приймає два аргументи: функцію ефекту та масив залежностей, які визначають, коли цей ефект повинен виконуватися. Також функія ефекту може повертати cleanup функцію які імітує поведінку функцію componentWillUnmount.

Існують також інші корисні хуки, такі як useContext, який дозволяє компонентам отримувати доступ до контексту без необхідності обгортати їх у компоненти-контейнери, та useRef, який створює об'єкт, що зберігає значення через рендери, і часто використовується для доступу до DOM елементів або збереження змінних, які не викликають повторний рендер при зміні. Хук useMemo мемоізує обчислення значень і повертає запам'ятоване значення, якщо залежності не змінилися, що допомагає оптимізувати продуктивність, зменшуючи кількість непотрібних обчислень. Хук useCallback повертає запам'ятовану версію колбек-функції, яка не змінюється між рендерами, якщо її залежності не змінилися, що корисно для оптимізації продуктивності в компонентах, які передають колбеки дочірнім компонентам.

Хуки дозволяють розбивати логіку компонентів на менші, багаторазові частини. Наприклад, можна створити користувацький хук для роботи з формами, який об'єднує всю необхідну логіку і може бути повторно використаний в різних

компонентах. Це спрощує управління побічними ефектами, оскільки весь код для одного ефекту можна тримати разом, що робить код більш зрозумілим і легшим у підтримці.

Розробники можуть створювати свої власні хуки для повторного використання логіки стану та ефектів. Користувацькі хуки – це звичайні JavaScript функції, назви яких повинні починатися з "use". Вони можуть використовувати інші хуки всередині себе і допомагають структурувати та організувати код. Наприклад, можна створити користувацький хук для управління формами або для роботи з API, що дозволяє винести складну логіку з компонентів, роблячи їх більш читабельними та менш завантаженими.

Хуки мають кілька важливих переваг. Вони зменшують кількість коду, оскільки логіку можна винести в користувацькі хуки. Весь код, пов'язаний з певним аспектом логіки, зосереджений в одному місці, що полегшує читання і підтримку. Функціональні компоненти з хуками є простішими для розуміння та використання в порівнянні з класовими компонентами. Хуки, такі як useMemo та useCallback, допомагають оптимізувати продуктивність, уникаючи непотрібних обчислень та повторних рендерів[6].

2.3. React Native

React Native – це популярний фреймворк на основі JavaScript, який дозволяє створювати мобільні застосунки з нативною візуалізацією для iOS та Android. Фреймворк дозволяє створювати застосунки для різних платформ, використовуючи ту саму кодову базу.

React Native був вперше випущений Facebook як проєкт з відкритим вихідним кодом у 2015 році. Всього за пару років він став одним з топових рішень для мобільної розробки. Розробка на React Native лежить в основі деяких провідних світових мобільних застосунків, включаючи Instagram, Facebook та Skype [7].

В основі React Native лежить концепція компонентів, яка також використовується в React для веб-розробки. Компоненти є основними будівельними блоками інтерфейсу користувача, які можна повторно використовувати. React Native надає ряд вбудованих основних компонентів, готових до використання.

Основні компоненти:

- **View**: найфундаментальніший компонент для побудови інтерфейсу користувача. Є контейнером, який підтримує компонування за допомогою flexbox, стилі та певну обробку дотиків користувача. View напряму перетворюється на нативні “view” еквівалентні до платформи на які React Native працює, незалежно від того чи це UIView, , android.view і тому подібні;
- **Text**: компонент для відображення тексту. Він підтримує наслідування, стилі, а також управління дотиками;
- **Image**: компонент для відображення різних типів зображень, зокрема мережевих зображень, статичних ресурсів, тимчасових локальних зображень та зображень з локального диска;
- **TextInput**: компонент для введення тексту в застосунок з клавіатури.

Компоненти інтерфейсу користувача:

- **Button**: простий і базовий компонент який дозволяє швидко додавати кнопки з основною функціональністю натискання, проте має обмежені можливості для стилізації та кастомізації;
- **Switch**: відображає стандартну кнопку з логічним входом(on/off).

Компоненти для відображення списків:

- **FlatList**: ефективний компонент для візуалізації списків з можливістю прокрутки;
- **SectionList**: те саме що й FlatList, але має поділ на секції.

Крім того, розробники можуть створювати свої власні компоненти, поєднуючи базові компоненти та додаючи їм специфічні функціональні можливості.

Стилі в React Native пишуться за допомогою об’єкта StyleSheet, який є схожим на CSS, але має свої особливості та обмеження. Наприклад, стильові властивості у React Native використовують camelCase, а не kebab-case, як у CSS. Стилі можна застосовувати безпосередньо до компонентів, що дозволяє створювати адаптивні та гнучкі інтерфейси.

React Native також надає доступ до багатьох нативних API, таких як доступ до камери, геолокації, сенсорів тощо. Якщо потрібно використовувати специфічний нативний функціонал, який не підтримується стандартними засобами, можна створювати власні нативні модулі на Java (для Android) або Swift/Objective-C (для iOS). Це дозволяє інтегрувати специфічні нативні функціональні можливості у React Native застосунки.

Для управління станом застосунку часто використовується бібліотека Redux, яка допомагає організувати та централізувати стан застосунку. Redux надає просту і передбачувану структуру для управління станом застосунку, що робить код більш організованим і легшим для підтримки.

Однією з важливих особливостей React Native є підтримка технологій, таких як Hot Reloading та Fast Refresh, які дозволяють розробникам швидко бачити зміни в коді без необхідності перезавантажувати весь застосунок. Це прискорює процес розробки та полегшує налагодження.

Основний механізм взаємодії між JavaScript і нативними компонентами називається мостом (bridge). Це дозволяє виконувати JavaScript код і нативний код одночасно, але в різних потоках, забезпечуючи зв'язок між ними. Міст дозволяє JavaScript викликати нативні методи і навпаки, що робить можливим реалізацію складних функціональних можливостей і доступ до нативних API[8].

2.3.1. React Navigation

React Navigation - це бібліотека для організації навігації в React Native застосунках, яка дозволяє створювати складні навігаційні структури та забезпечує плавний користувацький досвід. Основним компонентом є NavigationContainer, який створює контекст для навігації та обгортає весь застосунок або його основну частину, забезпечуючи коректну роботу навігаційної системи. Всі екрани застосунку визначаються як компоненти, що підключаються до навігаторів і мають унікальні ключі для навігації. Навігаційні дії (navigate, push, pop, goBack) дозволяють переміщуватися між екранами або змінювати навігаційний стан. Параметри можна передавати між екранами через ці дії, і вони доступні у компоненті екрана через об'єкт route. Кожен навігатор підтримує налаштування зовнішнього вигляду і поведінки, такі як заголовки, стилі переходів, та кнопки на панелі. Для опису навігації використовується два основних компоненти:

- **Screen** – це компонент який відповідає за окремий екран в застосунку. Він використовується для визначення маршруту(або сцени) за яким користувач може перейти. Кожен “Screen” пов’язаний з компонентом, який відображається при переході на цей маршрут. Назву маршруту та пов’язаний компонент можна вказати в атрибутах “name” та “component” відповідно. Для додаткової кастомізації екрана використовується атрибут “options”
- **Navigator** – це контейнер, який організовує управління переходами між вказаним в середині нього екранами. Він визначає тип навігаційної структури (стек, вкладки, шухляда тощо) і керує логікою переміщення між екранами.

React Navigation має три види навігації: “Stack Navigation”, “Tab Navigation” та “Drawer Navigation”.

“**Stack Navigation”** використовується для побудови ланцюжків екранних переходів, де кожен новий екран накладається поверх попереднього, утворюючи стек. Це схоже на те, як працює навігація у веб-браузері, де кожна нова сторінка додається до історії. Користувачі можуть повернутися до попередніх екранів, видаляючи верхній екран зі стека, що дозволяє реалізувати послідовні переходи з можливістю повернення.

“Tab Navigation” дозволяє створювати застосунки з нижньою або верхньою панеллю вкладок. Кожна вкладка відповідає за окремий екран або групу екранів. Це корисно для застосунків, де необхідно забезпечити швидкий доступ до основних розділів, таких як домашня сторінка, пошук, повідомлення або профіль користувача. “Tab Navigation” забезпечує зручну та інтуїтивно зрозумілу організацію застосунку.

“Drawer Navigation” створює бічне меню, яке висувається з краю екрана. Це меню може містити посилання на різні екрани або розділи застосунку. “Drawer Navigation” часто використовується для доступу до менш часто використовуваних функцій або налаштувань, дозволяючи зберігати основний екран чистим та зрозумілим. Користувачі можуть відкрити меню жестом або натисканням на спеціальну іконку[9].

2.4. Expo

Expo є потужним інструментом для розробки мобільних застосунків, що базуються на React Native, забезпечуючи значне спрощення процесу розробки, тестування та розгортання застосунків для платформ iOS та Android. Він надає все необхідне для швидкого початку роботи, включаючи командний інтерфейс, мобільний застосунок для миттєвого перегляду змін та багатий набір API для доступу до нативних функцій пристроїв.

Основою платформи є Expo CLI, командний інтерфейс, що дозволяє створювати нові проєкти, запускати їх у режимі розробки та керувати білдами.

CLI інтегрується з Expo DevTools, зручним веб-інтерфейсом для управління проєктами.

Застосунок Expo Go для iOS та Android дозволяє розробникам переглядати свої проєкти в режимі реального часу. Це усуває потребу в складних налаштуваннях для тестування застосунків на різних пристроях та використання емулятора, надаючи можливість миттєвого перегляду змін шляхом сканування QR-коду з DevTools.

Expo SDK надає доступ до широкого спектра нативних функцій пристроїв через готові модулі та API. Розробники можуть використовувати ці модулі для інтеграції таких функцій, як камера, геолокація, сенсори та сповіщення, без необхідності писати нативний код. Це дозволяє зосередитися на функціональності застосунку, використовуючи вже готові рішення для доступу до апаратних ресурсів та платформних можливостей.

Сервіси Expo Application Services (EAS) автоматизують багато аспектів розробки, включаючи створення білдів та оновлення застосунків. EAS Build дозволяє автоматизувати процес створення білдів для iOS та Android, знімаючи необхідність налаштування складного середовища для білдів на локальному комп'ютері. Натомість розробники можуть відправляти свої проєкти на сервери Expo, де створюються білди, готові до розгортання. EAS Update забезпечує безшовне оновлення додатків, дозволяючи впроваджувати зміни у коді без проходження через процеси верифікації в App Store або Google Play, що значно скорочує час впровадження нових функцій та виправлення помилок.

Для реалізації навігації в застосунках використовується Expo Router який має за основу React Navigation. Дана навігація базується на прописанні маршрутів за допомогою файлового дерева. Кожен шлях прописується в теці “app” яка є коренем всієї навігації. Кожна тека відповідає за свій маршрут, а файл з ім’ям “index” описує компонент до якого він веде. Цікавою можливістю є створення групи маршрутів, назва якої буде ігноруватись в стрічці запиту. Для

створення групи її назву огортають в дужки(наприклад “(tabs)”). Також можна описувати загальний шаблон маршруту який буде огортати всі дочірні компоненти. Для опису шаблону створюється файл “layout” в середині теки маршруту. Всі основні види навігації були також перенесені з “React Navigation” в “Expo Router” [10].

2.5. Скелет застосунку

Перед тим як створити основу застосунку потрібно завантажити усі необхідні інструменти для розробки, а саме node.js та expo cli. Node.js можна завантажити з офіційного сайту. Для цього на головній сторінці натиснути на кнопку “Dowload Node.js(LTS)” яка завантажить msi файли, відкривши який потрібно пройти процес встановлення. Після встановлення можна перевірити чи все пройшло успішно ввівши наступну команду:

```
node -v
```

Якщо все коректно, вона виведе поточну версію node.js. Наступним кроком буде встановлення expo cli глобально для системи. Для цього потрібно виконати наступну команду:

```
npm install -g expo-cli
```

Після цього ми готові створити початковий макет. Ми будемо використовувати вже наявний шаблон, який створить проєкт з використанням Tab Navigation. Для цього потрібно прописати наступну команду:

```
npx create-expo-app ClientDev --template tabs
```

По завершенню створення проєкту в нас створиться застосунок, який має дві вкладки, а також одну додаткову сторінку, яка випливає при натисканні

кнопки в заголовку застосунку. Також в нижній частині екрана знаходиться блок навігації, за допомогою якого ми можемо переміщуватись між вкладками (рис. 2.1).

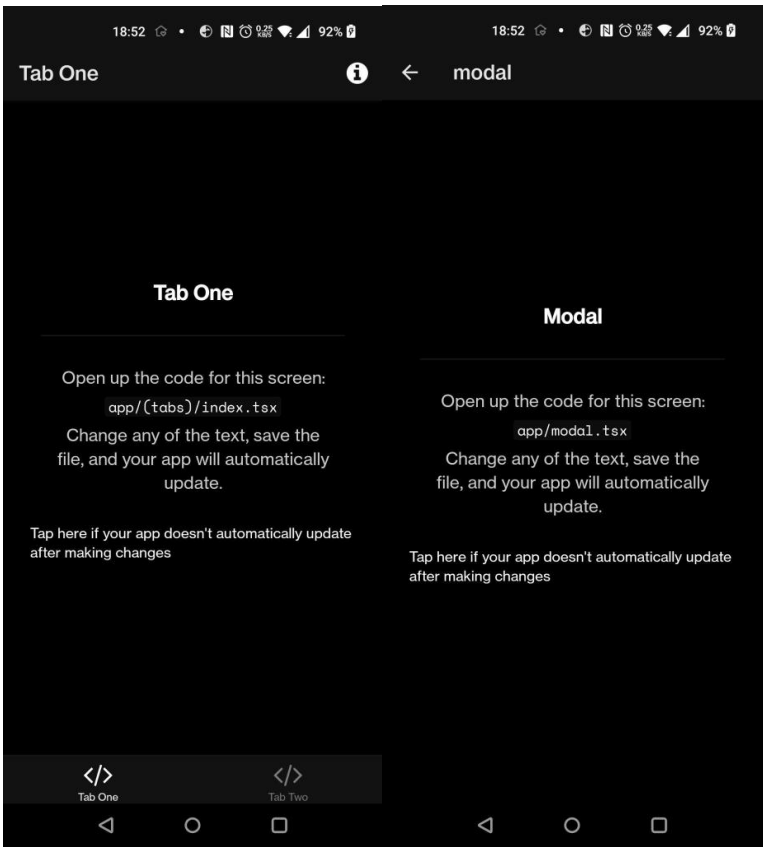

Рис. 2.1. Вигляд застосунку за створеним шаблоном

Даний скелет потребує певних модифікацій, однією з яких є додавання авторизації. Але перед тим як створювати логіку авторизації потрібно також створити власний React Hook під назвою useStorageState для роботи з SecureStorage. Цей хук буде зберігати передані йому дані за вказаним ключем локально в засекреченому сховищі пристрою. Він має один аргумент під назвою key, що записує в себе значення ключа, за яким ми потім звертаємось до сховища, щоб отримати дані записані під ним. В середині хука є поле “state” – масив з двох елементи (isLoading та value), який ми отримали за допомогою вбудованого React хука useState, для зберігання даних записаних в сховище, функцію setValue, яка

встановлює передане її значення для поля в сховищі за вказаним ключем, а також useEffect який дістає дані з SecureStorage за вказаним ключем і встановлю їх для поля “state” при запуску застосунку або при зміні ключа.

Далі на основі хука useStorageState створюємо новий AuthContext, в якому буде реалізована логіка авторизації. Для цього в теці “components” ми створюємо нову теку “context” з файлом “AuthContext.tsx”. В середині файлу ми ініціалізуємо контекст за допомогою хука React.createContext, який має два стани(session, isLoading) та дві функції(SignIn, SingOut). Обидва стани ми отримуємо в середині SessionProvider викликавши хук useStorageState передавши йому ключ з назвою “session”. Сам SessionProvider повертає JSX який складається з елемента AuthContext.Provider. Даний елемент має обов’язковий параметр value, в який ми передаємо об’єкт зі створеними станами та описуємо обидві функції для входу в акаунт та виходу з нього. Функція SingOut є доволі простою, оскільки все що ми в ній робимо це викликаємо функції setSession з параметром “null”. SingIn має трохи важчий функціонал, оскільки тут ми маємо провести авторизацію користувача, і тільки, якщо вона є успішною, викликати функцію setSession з поверненим токеном користувача.

Ще однією функцією нашого застосунку є можливість змінювати тему з темної на світлу. Для цього потрібно створити ще один контекст, який матиме лише один стан ("state") та функції для його контролю. Спочатку у теці "context" створюємо новий файл під назвою "ThemeContext" і відповідний контекст. Далі, на основі цього контексту, створюємо нового провайдера, де прописуємо весь функціонал для керування темою застосунку. У ньому створюємо стан "theme", початкове значення якого буде відповідати поточній темі пристрою. Для отримання теми пристрою можна використати вбудований у React Native хук useColorScheme. Для зміни теми ініціалізуємо функцію toggleTheme, яка змінюватиме значення стану "theme", а також записуватиме нову тему в асинхронне сховище пристрою. Для отримання поточної теми зі сховища

потрібно створити useEffect, який буде завантажувати її при запуску застосунку. Код useEffect:

```
useEffect(() => {
 const getTheme = async () => {
   try {
     const savedTheme = await AsyncStorage.getItem('theme');
     if (savedTheme) {
       setTheme(savedTheme as 'light' | 'dark');
     }
   } catch (error) {
     console.log('Error loading theme:', error);
   }
 };
 getTheme();
}, []);
```

Ще один useEffect потрібний для встановлення теми пристрою при її зміні:

```
useEffect(() => {
 if (colorScheme) {
   setTheme(colorScheme);
 }
}, [colorScheme]);
```

Наступним кроком є змінення дерева навігації в теці "app" для врахування авторизації користувача. Для цього всі елементи, що знаходяться в кореневій теці навігації, переносяться в групу “(app)”. У самій теці створюються два нові шляхи: "sign-in" та "not-found", а також файл "layout.tsx". Зміст кожного файлу описано далі:

“not-found” – це невеликий файл, в якому описується вигляд сторінки, на яку переходить користувач у разі некоректного маршруту. Він містить елемент Text зі значенням “Такого шляху не існує!!!”, а також Link, при натисканні якого користувача повертає на початковий екран.

“sign-in” – це маршрут, на який автоматично перенаправляється користувач, якщо він не авторизований. На цій сторінці є одна кнопка “Sign-in”, при натисканні якої викликається функція signIn, отримана за допомогою хука useSession. Після виклику цієї функції перевіряється, чи була авторизація успішною, і якщо так, то поточний маршрут замінюється на кореневий за допомогою функції route.replace('/'). Кнопка не є елементом Button, а реалізована за допомогою вбудованого в Expo елемента Pressable, всередині якого знаходиться елемент Text.

“layout.tsx” – це файл, що описує структуру пов’язаних з ним маршрутів. JSX, який повертається цим файлом, містить елемент “Slot”, що буде замінюватися відповідними елементами згідно з поточним маршрутом. Цей елемент огортається в “SessionProvider”, щоб забезпечити доступ до контексту авторизації всередині застосунку, а також в “ThemeProvide” для керування темою застосунку. У цьому файлі ми також підключаємо шрифти, які будуть використовуватися для тексту нашого застосунку. Наразі весь текст буде використовувати лише один шрифт під назвою “Gillroy”, який ми підключаємо за допомогою вбудованого в Expo хука “useFonts”. Цей хук приймає об’єкт, де ключі – це назви шрифтів, а значення – шляхи до файлів шрифтів. Він повертає масив, де перший елемент (“fontsLoaded”) є булевим значенням, що вказує на те, чи завантажені шрифти, а другий елемент(“error”) вказує, чи виникла помилка під час завантаження шрифтів.

Після налаштування логіки авторизації переходимо до теки “(app)”, де змінюємо файл макета маршрутів для роботи з авторизацією. Викликаємо хук useSession для отримання змінних session та isLoading. Якщо isLoading дорівнює істині, ми повертаємо блок з текстом “Loading…”. Якщо ні, переходимо до наступної перевірки – чи дорівнює змінна session значенню “null”. Якщо це твердження вірне, ми перенаправляємо користувача на сторінку “sign-in”. В іншому випадку повертаємо елемент “Stack”, який має два можливі маршрути: “(tabs)” та “expanded_info”. “expanded_info” – це маршрут, перейшовши за яким,

користувач побачить розширену інформацію про свій профіль. Група “(tabs)” також потребує певних змін, оскільки на відміну від початкового шаблону, ми маємо три конкретні сторінки, які відображають інформацію для користувача. Наразі ми не будемо реалізовувати логіку для цих сторінок, а просто відображатимемо їхні назви в центрі екрана. Цими сторінками є:

- **schedule**: сторінка для перегляду розкладу;
- **profile**: сторінка для перегляду інформації про користувача. У проєктній ієрархії вона називається “index.tsx”, оскільки це початкова сторінка, яку бачить користувач при запуску застосунку;
- **announcements**: сторінка для відображення новин.

Оскільки в нас змінилися основні маршрути, потрібно оновити код шаблону групи “(tabs)”, замінивши поточні маршрути на новостворені. Також для кожного маршруту у властивості “options” в полі “tabBarIcon” ми встановлюємо відповідну іконку.

Підсумовуючи, ми створили основу нашої програми з усіма основними маршрутами(рис. 2.2) та логікою для авторизації. Наступним етапом є додавання необхідного функціоналу до цієї основи та оформлення кожної сторінки.

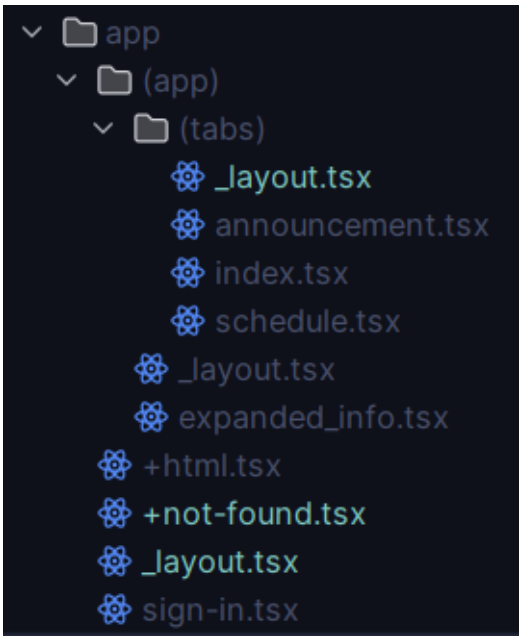

Рис. 2.2. Поточне дерево маршрутів

2.6. Елементи UI

Дизайн нашого застосунку в багатьох випадках відрізняться від шаблону, який ми використовували при створенні скелета програми, особливо це видно по заголовку, а також нижнього блоку навігації. Вбудовані методи налаштування цих елементів не мають того функціоналу, який потрібний, для того, щоб відтворити ці елементи відповідно до дизайну. Через що нами потрібно створити свої власні “Tabbar” та “Header”. Tab Navigation дозволяє змінити дані елементи на власні за допомогою аргументів “tabBar” та “header” в елементі Tab. Дані аргументи приймаю функції, які мають повертати наші “Tabbar” та “Header”.

Наступник кроком є створення файлів “CustomHeader” та “CustomTabBar” для наших елементів. Оскільки заголовок є доволі простим і не має в собі великої кількості логіки можна почати з нього.

Код CustomHeader:

```
export default function CustomHeader({options}:
BottomTabHeaderProps) {
 const {theme, toggleTheme} = useContext(ThemeContext)

 const switchThemeHandler = () => {
   toggleTheme(theme === 'light' ? 'dark' : 'light')
 }

 return <LinearGradient
   style={styles.headerContainer}
   colors={theme === "light" ? ['#F4F4F4', '#600CEB'] :
['#600CEB', '#fff']}
   start={[0, 0]}
   end={[1, 0]}
 >
   <Text style={[styles.title]}
```

```
lightColor={"#fff"}>{options.title}</Text>

   <Pressable onPress={switchThemeHandler}
style={[styles.icon, {backgroundColor: theme === "light" ?
"#600CEB" : "#fff"}]}>
     {theme === "light" ? <Moon /> : <Sun />}
   </Pressable>
 </LinearGradient>
}
```

Даний елемент має єдиний аргумент - “option”, який надає інформацію про поточну сторінку. З нього ми можемо дістати назву сторінки для її подальшого відображення. Всередині функції елемента для початку ми отримуємо метод для зміни теми застосунку – “toggleTheme” використовуючи “ThemeContext”. Даний метод присвоюється до події “onPress” елемента “Pressable”, який містить в собі іконки “місяця” або “сонця” залежно від того, на яку тему потрібно перейти при натисканні. Фон заголовка містить градієнт, створений за допомогою елемента "LinearGradient" з бібліотеки "expo-linear-gradient", який відображає лінійний градієнт із заданими кольорами. Назву сторінку ми відображаємо за допомогою поля “Text”(рис. 2.3).

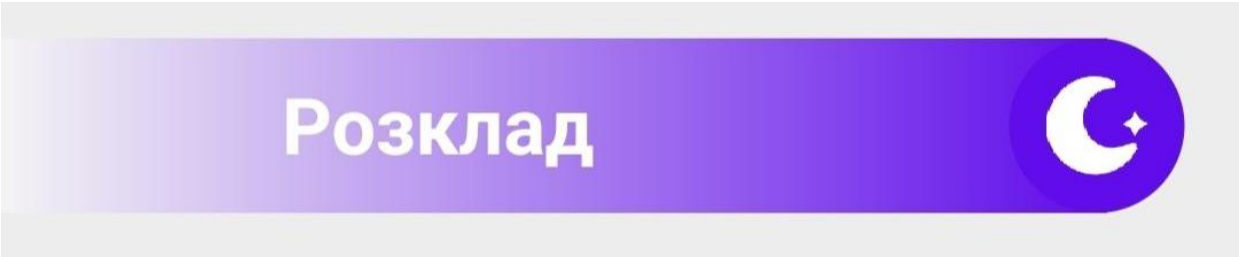

Рис. 2.3 Вигляд заголовка сторінки

Наступним елементом є “CustomTabBar”, який складається з контейнера, що містить кнопки, кожна з яких відповідає за свою сторінку. При відтворенні цього елемента React Native передає в нього декілька аргументів (state, descriptors, navigation), які ми можемо використати для кастомізації його

поведінки, а також вигляду. Для відображення кнопок в середині контейнера ми ітеруємо через масив “routes” аргументу “state”, який надає повну інформацію про кожну вкладку. Кожна кнопка має зміну isFocused, яка вказує на те чи вкладка, за яку вона відповідає, є вибрана. Кожна кнопка повертає елементи “Pressable”, до якого ми додаємо дві події “onPress” та “onLongPress”. Подія “onPress” буде переправляти користувача на вибрану сторінку використовуючи метод “navigate” аргументу “navigation” за умови, що зміна isFocused не є істиною.

Для покращення досвіду користувача ми додамо задній фон для вибраного елемента. Оскільки переміщення фону має бути анімоване ми його створюємо використовуючи елемент “Animated.View” бібліотеки “react-reanimated”. Створений фон буде переміщуватись між координатами центру кнопок, які ми можемо отримати використовуючи метод “onLayout”. Для подальшого використання ми записуємо отримані координати в масив “elementPositions”. Для збереження позиції вибраної кнопки створюємо зміну “backgrounbPos” використовуючи хук “useSharedValue”. На основі даної змінної ми створюємо новий анімований стиль, який присвоюємо нашому фону. Для відстежування зміни поточної вкладки прописуємо “useEffect”, який буде викликати функцію “moveBackground” при зміні “state.index”. Всередині функції “moveBackground” ми отримуємо значення з масиву “elementPositions” за вказаним індексом і присвоюємо його змінній “backgrounbPos”. Оскільки перехід до даного значення має бути анімованим, ми огортаємо його в функцію “withTiming”(рис. 2.4).

Рис. 2.4 Вигляд блоку навігації

Контент кожної сторінки огорнутий в однаковий контейнер, тому створимо для нього свій елемент під назвою “MainContainer”. Даний елемент повертає аргумент “children” огорнутий в “View”. Використовуючи Stylesheet, заокруглюємо границі елемента та присвоюємо кольори для обох тем, використовуючи параметри “lightColor” та “darkColor” елемента “View”.

Певні елементи в нашому застосунку можуть розгортатись при натисканні на них, для їх реалізації було створено компонент “CollapsableComponent”. Він складається з елемента “Animated.View”, зміну параметрів якого ми можемо анімувати. Для показу розгортання елементу потрібно змінювати висоту елемента, яку ми можемо отримати використовуючи функцію onLayout. Отриману висоту записуємо в новий “state” під назвою “height”. Для визначення поточного стану елемента був створений аргумент “expanded”, при зміні якого компонент буде відкриватись або закриватись. Під кінець створюємо новий “useEffect”, який буде запускати анімацію використовуючи функцію “withDelay” при зміні значення “expanded”.

Основні компоненти UI було створено, наступним етапом є реалізація сторінок застосунку та їх наповнення.

2.7. Profile

При запуску застосунку перше що буде бачити користувач це сторінку його профілю, яка містить інформацію про нього. Дана сторінка поділиться на два блоки, перший відображає: ПІП користувача, назву групи та унікальний номер, другий – бар-код, за допомогою якого можна відмічатись в середині навчального закладу. Якщо висота екрана користувача є нижча за 640 пікселів блок з баркодом ховається, оскільки в іншому випадку він заходив би за межі блоку навігації. Для отримання висоти екрана пристрою використовується об’єкт “Dimension”, а саме його метод “get(“window”).height”. Для відображення баркода використовується елемент “Barcode” бібліотеки “expo-barcode-generator”. Даний елемент має наступні параметри:

```
<Barcode options={{
 format: 'CODE128B',
 width: 1.4,
 height: 97,
 displayValue: false,
 background: theme === 'light' ? '#F4F4F4' : '#FFFFFF',
 marginLeft: 20,
 marginRight: 20,
}} value='Example1234'
/>
```

Переходимо до блоку відображення інформації, який ми переносимо в окремий елемент під назвою “ProfileInfo”. Для відображення фотографії користувача створюємо новий елемент “ProfileCircle”, скелет якого будуємо на основі “svg” фотографії з “Figma”. Після зображення профілю користувача іде три поля “Text”: ПІП, група та унікальний номер. Останніми елементами є кнопка “Вийти” та елемент “Link”, який перенаправляє користувача на сторінку з розгорнутою інформацією. Використовуючи хук “useSession” отримуємо функцію “signOut”, яку прив’язуємо до події “onPress” кнопки “Вийти”. Створюємо нову сторінку для відображення розширеної інформації про користувача та прив'язуємо її до елемента “Link”.

Сторінка “expanded_info” поділяється на два блоки: заголовок та баркод. В заголовку ми відображаємо: ПІП користувача, його роль, унікальний номер та елемент “Link”, за допомогою якого користувач може повернутись на головну сторінку. В нижній частині головного контейнера відображаємо зображення користувача використовуючи елемент “ProfileCircle”, а також баркод зі збільшеним відступом між паралельними лініями.

Останнім елементом, який залишилось додати до сторінки профілю користувача, є лого та назва навчального закладу, які ми переміщуємо в елемент “MainTitle”. Даний елемент складається з фотографії лого в форматі “svg” та поля “Text” який відображає назву.

2.8. Announcements

Сторінка “announcements” має в собі завдання відображати основні новини про навчальний заклад. Сама її структура доволі проста і складається з єдиного елемента “Flatlist”. Даний елемент потребує дані для відображення, які ми отримуємо з сервера використовуючи функцію “fetch”. Для перевірки чи запит на сервер був вдалий огортаємо виклик функції, а також обробку отриманих даних в структуру “try…catch”. Отримані дані зберігаємо в новому “state”, який називаємо “announcementsData” та передаємо його в атрибут “data” елементу “FlatList”.

Наступним етапом є створення елементу “AnnouncmentItem”? який буде відображати отримані дані. Він має чотири аргументи:

- title - заголовок новини
- text - текст новини
- images - масив зображення про новину
- date - дата публікації

По замовчуванню елемент буде відображати тільки основне зображення, заголовок, дату та кнопку “Більше”. Формати дати буде змінюватись залежно від того коли новина була опублікована, якщо публікація була сьогодні буде відображатись точний час, якщо цього року - число та назва місяця, у всіх інших випадках дата в форматі - “dd.mm.yyyy”.

Код для визначення формату дати:

```
let dateText = date.toLocaleDateString("uk-UA");

if(isToday(date)) {
 dateText = formatDate(date, 'kk:mm', );
} else if(date.getFullYear() === (new Date()).getFullYear())
{
 dateText = date.toLocaleDateString("uk-UA", {
```

```
    month: 'long',
    day: 'numeric'
  })
}
```

Для показу зображення використовується елемент “Image” бібліотеки “expo-image”. Прихований блок з текстом новини, а також всіма зображеннями огортаємо в елемент “CollapsableComponent” стан якого буде керуватись за допомогою зміни “expanded”. До події “onPress” кнопки “Більше” було приєднано функцію “expandHandler” яка змінює стан змінної “expanded” на протилежний та запускає анімацію зміщення головного зображення за межі елементу за допомогою функції “withDelay”.

2.9. Schedule

Сторінка “schedule” є останньою в циклі розробки та відповідає за показ розкладу користувача. Вона поділяється на дві частини – календар та вікно з подіями. Вікно з подіями знаходиться безпосередньо в середині головного контейнера сторінки та складається з елементів “FlatList” та “Text”. Елемент “Text” повинен відображати вибрану користувачем дату. Для роботи з цією датою ми створюємо новий стан під назвою “selectedDate” та присвоюємо йому значення “new Date()”, яке поверне об’єкт “Date” з поточним часом. При передачі змінної до елементу “Text” ми викликаємо функцію “toLocaleDateString()”, яка поверне дату в форматі число та назва місяця (наприклад “5 червня”) при передачі в неї наступних параметрів:

```
locales: "uk-UA",
options: {
     month: "long",
     day: "numeric"
}
```

Елемент “FlatList” вимагає дані, на основі яких будуть будуватись елементи списку. Для отримання створюємо новий “useEffect” в середині якого використовуючи функцію “fetch” отримуємо дані про події з сервера. Оскільки сервер повертає тільки ті дні, які мали хоча б одну подію нам потрібно заповнити прогалини об’єктами, які мають пусті маси для полів “events”. За реалізацію цієї логіки відповідає наступний код:

```
let tempDates = []

tempDates.push(EVENTS[0])

for(let i = 0; i < EVENTS.length - 1; i++) {
 const difference = differenceInDays(EVENTS[i+1].date,
EVENTS[i].date)
 if(difference > 1) {
   for(let j = 1; j < difference; j++) {
     tempDates.push({date: addDays(EVENTS[i].date, j),
events: []})
   }
 }
 tempDates.push(EVENTS[i+1])
}
setDates(tempDates)
```

“EVENTS” – це масив отриманих від сервера подій, на основі якого ми заповнюємо новий тимчасовий масив “tempDates” паралельно добавляючи пропущені дні. Процес додавання нових елементів відбувається наступним чином:

1. В “tempDate” додається перший елемент масиву “EVENTS” після чого ми переходимо в цикл, який обходить всі елементи окрім останнього;
2. Між датами поточного та наступного елементу обраховується різниця в днях і записується в зміну “diffence”;

3. Якщо різниця більша за одиницю в масив “tempDates” додається “diffecnce-1” пустих елементів;
4. В кінці циклу додаємо наступний після поточного елемент;
5. Отриманий масив “tempDates” записуємо в стан dates використовуючи функцію “setDates”.

Переходимо до налаштування списку “FlatList”. На основі таб. 2.1 виставляємо значення атрибутів

Таблиця 2.1

Значення атрибутів елемента “Flatlist”

Атрибут	Значення
ref	carouselRef
data	dates
horizontal	true
snapToAlignment	start

Продовження таблиці 2.1

snapToInterval	width
decelarationRate	fast
keyExtracror	(_, index) => index.toString()
viewabilityConfig	{itemVisiblePercentThreshold: 50}

Крім того, нам потрібно заповнити атрибути “renderItem” та “onMomentumScrollEnd”. Перший відповідає за вигляд елементу списку, а другий буде викликати передану в нього функцію при закінченні прокручування списку. Функція яку ми передаємо в “onMomentumScrollEnd” буде обраховувати індекс поточного елементу та встановлювати нову вибрану дату на його основі.

Атрибут “renderItem” приймає функцію з аргументом “day”, в який передається інформація про поточний елемент списку, взята з атрибута “data”. На основі даних отриманих з даного аргументу буде створюватись елемент списку. Перши чином ми дістаємо масив подій і перевіряємо чи він пустий, якщо так, то ми відображаємо текстове поле зі значенням “Сьогодні ви вільні”, якщо ні, то ми повертаємо компонент “ScrollView” який буде відображати події пов’язані з датою поточного елемента. Для оформлення кожного блоку події створюємо новий компонент “EventItem”.

Компонент “EventItem” є інтерактивним і при натисканні на нього випливають поля з додатковою інформацією. Для реалізації даної поведінки було створено стан “expanded” для трекінгу поточного стану об’єкта. Для його керування використовується функція “expanHandler”. Дану функцію ми прив’язуємо до події “onPress” елемента “Pressable”, який буде огортати весь компонент. Сам компонент в згорнутому стані показує час початку події, а також її назву за допомогою елементів “Text”. Поля додаткової інформації огортаються в створений раніше елемент “CollapsableContainer”, який може змінювати свій стан (розгорнутий чи згорнутий) залежно від атрибута “expanded”. В середині цього елемента буде знаходитись наступних три поля: “Організатор”, “Аудиторія” та “Тип заняття”.

Переходимо до компонента “RowCalendar”, який відповідає за відображення стрічкового календаря. Даний елемент складається з “карусель” яка містить в собі розписані тижні, а також двох аргументів один із яких це вибраний день(“selectedDae”), а другий це функція для його зміни(“setSelectedDate”). Для створення карусель був використаний компонент “Carousel” бібліотеки “react-native-reanimated-carouse”. Даний елемент є схожим за структурою з “FlatList”, але має інший функціонал який дозволяє створювати інтерактивні “каруселі”. За замовчування він створює нескінчену карусель, яка при досягненні останнього елемента переходить до першого. Даний функціонал допоможе нам реалізувати нескінчений список тижні. Для початку нам потрібно

створити масив даних, який буде використовуватись. Для цього створюємо функцію “getWeeks”, яка отримує інтервал, між яким ми хочемо отримати масив днів тижнів. Спочатку на основі інтервалу ми отримуємо перші дні кожного тижня використовуючи функцію “eachWeekOfInterval” бібліотеки “date-fns”. Після чого ми проходимось по кожному дню і на його основі створюємо масив днів тижня з відповідними датами, який потім записуємо в загальний масив. Для збереження цього двовимірного масиву створюємо новий стан “dateData”. Даний масив завжди буде складатись з трьох елементів, другий елемент якого буде відповідати за поточний тиждень, а перший та третій за попередній та наступний тижні відповідно.

Додаємо до JSX коду компонент “Carousel” з атрибутами наведеним в таб. 2.2

Таблиця 2.2

Значення атрибутів елемента “Carousel”

Атрибут	Значення
ref	carouselRef
disableIntervalMomentum	true
defaultIndex	1
width	sliderWidth
date	dateData
scrollAmimationDuration	500

Референс під назвою “carouselRef” створюємо на початку компонента використовуючи хук “useRef”. Даний референс допоможе керувати станом карусель за допомогою команд. Зміна “sliderWidth” обраховується за межами функції компонента на основі ширини екрана користувача яку ми можемо взяти за допомогою об’єкта “Dimension.window” та його властивості “width”. Також елемент “Carousel” має атрибути “renderItem” та “onScrollEnd” які потребують детальнішого опису.

Атрибут “renderItem” відповідає за вигляд елементів “каруселі”. Дані елемент будуть складатись з семи блоків? кожен з яких відповідає за свій день тижня. Кожен блок має відображати скорочену назву дня тижня, а також число. Для отримання числа місця використовувалась функція “getDate”, а для назви тижня – “toLocalDateSting”. Якщо поточний день є вибраним ми його виділяємо надаючи йому окремий стиль. Для перевірки чи поточний день є вибраним ми порівнюємо його з “selectedDate” використовуючи функцію “isEqual”. Сам “JSX” компонента складається з елемента “Pressable”, всередині якого знаходяться два текстових поля для відображення числа, місяця та назви тижня. При натисканні

на даний компонент його дата буде встановлюватись як вибрана за допомогою функції “setSelectedDate”.

Атрибут “onScrollEnd” отримує функцію “handleSnap”, яка відповідає за зміну тижнів. Дана функція отримує індекс поточного елемента списку і на основі його дати переписує наступний та попередній тижні. Для цього на її початку ми визначаємо індекси наступного та минулого елементів використовуючи наступний код:

```
let prevIndex = index === 0 ? 2 : index - 1
let nextIndex = index === 2 ? 0 : index + 1
```

Далі створюємо два нових масиви днів тижня використовуючи функцію eachDayOfInterval. Дана функція потребує інтервал, між яким ми хочемо отримати дані. Для попереднього тижня, ми за початкову дату беремо понеділок поточного тижня мінус сім днів, для кінцевої дати, беремо день який був перед понеділком. Для наступного тижня інтервал буде між першим днем після неділі та сьомим днем після неділі. З поточного тижня, а також двох новостворених робимо новий двовимірний масив та присвоюємо його стану “dateData”[13].

Переходимо назад до сторінки “schedule” та створюємо новий “useEffect” який буде переходити до вибраної дати при її зміні. Для цього в масив залежностей, додаємо “selectedDate”, а в середині ефекту використовуючи функцію “findIndex” знаходимо індекс елементу, дата якого дорівнює вибраній. Якщо такий елемент існує, ми переходимо до нього викликаючи функцію “scrollToIndex” на референсі “carouseRef”.

РОЗДІЛ 3
ТЕСТУВАННЯ РЕАЛІЗОВАНОЇ СИСТЕМИ

Кінцевий застосунок має три основні сторінки, між якими можна переміщуватись за допомогою блоку нижньої навігації, а також один додатковий екран для зручнішого показу баркоду.

При першому запуску застосунку користувач буде бачити сторінку його профілю. На даній сторінці користувач може переглянути основну інформацію про себе, а саме: ПІП, назву його групи, унікальний номер та фото профілю. У верхній лівій частині блоку знаходиться кнопка “Вийти”, при натиску якої користувач вийде з поточного акаунту і перейде до сторінки авторизації. У нижньому лівому кутку знаходиться кнопка для переміщення до сторінки з додатковою інформацією. Унікальний баркод користувача, який надає можливість відмічатись в навчальному закладі, знаходиться під блоком з основною інформацією(рис. 3.1).

Сторінка з розширеною інформацією містить: ПІП користувача, його роль, групу та унікальний номер. З лівої частини від інформації про профіль, розташована кнопка для переходу до основної сторінки. В нижній частині знаходяться фотографія користувача, а також баркод зі збільшеною шириною що полегшує його сканування (рис. 3.2).

а) б)

Рис. 3.1(а, б). Вигляд сторінок профілю користувача та розширеної інформації

Для переміщення між вкладками, в нижній частині знаходиться “tabbar” з відповідними до кожної вкладки іконками. Також можна використовувати кнопку “Повернутись назад”, щоб перейти до головної сторінки. Кожна вкладка має заголовок, який відображає назву вибраної сторінки, а також кнопку для перемикання теми застосунку. При першому запуску тема застосунку буде відповідати темі, вибраній на пристрої, при подальшій зміні її налаштування будуть зберігатись в локальному сховищі пристрою.

З правої сторони від інформації про профіль, знаходиться сторінка для перегляду новин. Кожна новина знаходиться в окремому блоці який відображає заголовок новини, час її створення, основне зображення та кнопку “Більше”. При

натисканні кнопки, основний блок розширюється і розкриває опис новини, а також додаткові фотографії. З правої сторони над фотографіями знаходиться кнопка “Сховати”, яка згортає блок до початкового стану. Кожен елемент з новинами знаходиться в хронологічному порядку в блоці, який можна вертикально перегортати для переходу до інших новин (рис. 3.3).

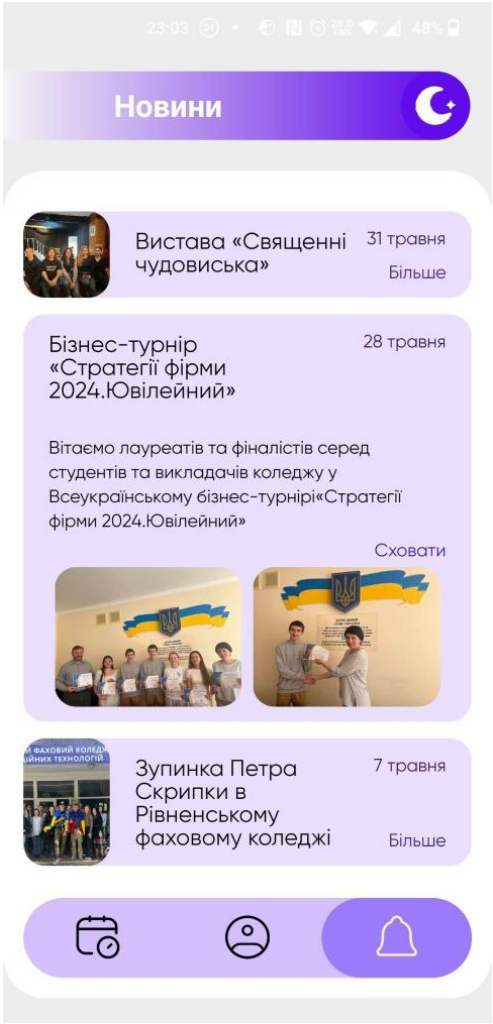

Рис. 3.2. Вигляд сторінки з новинами

Якщо користувач натисне на іконку з календарем в “tabbar”, він перейде на сторінку з його розкладом. Дана сторінка поділяється на блок для відображення подія та календарний рядок для вибору конкретної дати. Для переміщення між датами, можна свайпати вліво або вправо по блоку з подіями. У верхній частині блоку знаходиться рядок, який відображає вибраний день у форматі: число, назва

місяця. З лівої нижньої сторони знаходиться кнопка для переміщення до поточної дати.

Для вибору конкретної дати, існує календарний рядок, який відображає кожен день вибраного тижня. Для переміщення між тижнями, можна свайпати вліво або вправо. Для вибору конкретної дати, потрібно натиснути на блок з відповідним числом. Після натискання блок події автоматично переміститься до вибраної дати (рис. 3.3).

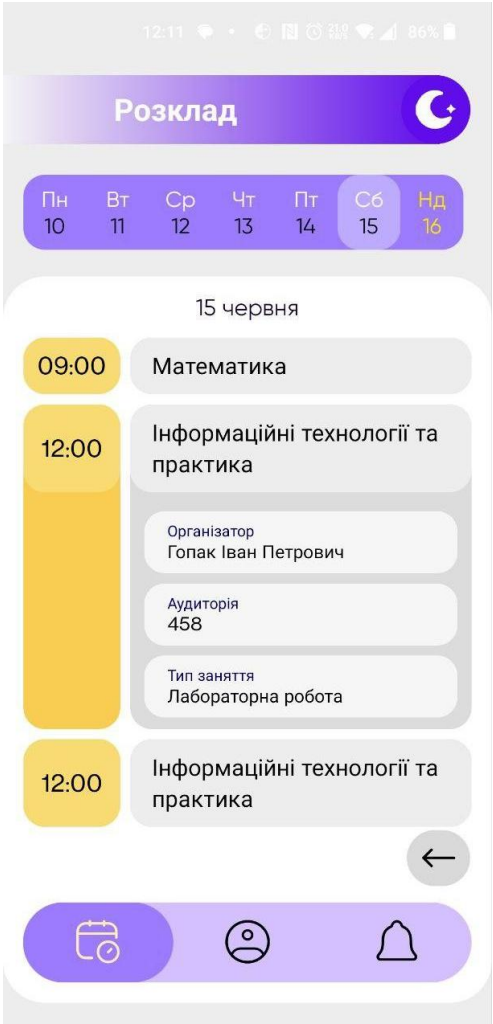

Рис. 3.3. Вигляд сторінки з розкладом

ВИСНОВКИ

У даному проєкті було розроблено мобільний застосунок під IOS та Andorid для навчального закладу з вбудованою системою пропуску. Головною метою проєкту було заміна застарілих методів отримання інформації студентами та викладачами про навчальний заклад на новітній застосунок, який дозволить отримувати важливу інформацію про навчальний процес будь-де і будь-коли.

Застосунок був розроблений за допомогою новітнього фреймворку React Native та бібліотеки Expo, які дозволяють створювати сучасні проєкти відразу під обидві основні платформи, Andorid та IOS. Він дозволяє користувачеві швидко переглянути розклад занять для ефективного планування дня, актуальні новини, а також основну інформацію про профіль. Ще однією важливою частиною даного застосунку є унікальний ключ у форматі баркоду, який дозволяє відмічати свою присутність на парах.

Дизайн застосунку є новітнім і відповідає всім стандартам брендбуку навчального закладу. Для кращої навігації користувача майже кожна дія супроводжується плавною анімацією. Сам застосунок має дві основні теми – темну і світлу.

Даний проєкт успішно переносить всю необхідну інформацію в кишеню користувача, що дозволяє без зайвих зусиль і витрат часу швидко й зручно звертатися до потрібної інформації підвищуючи ефективність навчання.

Код проєкту знаходиться за даним посиланням: https://gitlab.yspace.pp.ua/mypass/mypass-app-public

Для подальшого розвитку теми кваліфікаційної роботи можна зосередитися на вдосконаленні застосунку, шляхом таких заходів як:

- створення сторінки для перегляду журналу;
- вбудовування чату між студентом та викладачем для покращення комунікації;

- розширити налаштування застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lauren Robinson. - “The Digital Campus: Benefits of University Apps for Students”. URL: https://eventee.co/blog/digital-campus-benefits-of-university-apps-for-students (дата звернення: 27.02.2024).
2. Saify Technologies. - “Why all schools should develop mobile applications”. URL: https://www.linkedin.com/pulse/why-all-schools-should-develop-mobile-applications-/ (дата звернення: 27.02.2024).
3. Engage2Serve. - “10 reasons why college mobile apps are a must?”.URL: https://medium.com/@engageserve/10-reasons-why-college-mobile-apps-are-a-must-392a46de8fc2 (дата звернення: 27.02.2024).
4. React Documentation. URL: https://react.dev/learn (дата звернення: 30.03.2024).
5. React Component. URL: https://react.dev/reference/react/Component (дата звернення: 31.03.2024).
6. React Hooks. URL: https://react.dev/reference/react/hooks (дата звернення: 31.03.2024).
7. Maciej Budziński. - “What Is React Native? Complex Guide for 2024”. URL: https://www.netguru.com/glossary/react-native (дата звернення: 28.05.2024).
8. React Native Documentation. URL: https://reactnative.dev/docs/getting-started (дата звернення: 30.03.2024).
9. React Navigation Documentation. URL: https://reactnavigation.org/docs/getting-started/ (дата звернення: 31.03.2024).
10. Expo Documentation. URL: https://docs.expo.dev/ (дата звернення: 04.04.2024).
11. Expo Router. URL: https://docs.expo.dev/router/advanced/tabs/ (дата звернення: 05.04.2024).

12. React SVGR. URL: https://react-svgr.com/playground/?native=true&typescript=true (дата звернення: 15.04.2024).
13. Bouteiller < A2N > Alan. - “How to make a horizontal calendar slider in React Native with Flatlist (long story)”. URL: https://a2nb.medium.com/how-to-make-a-horizontal-calendar-slider-in-react-native-with-flatlist-f1797ffa4dee (дата звернення: 18.04.2024).