Start new chat

Projects

Chats

Recents

Reviewing Shared Content

Integrating React Native App with Firebase

Reviewing Previous Chat History

Untitled

Continuing Project Development

Conversation in Ukrainian

How Simparica Flea and Tick Tablets Work

Interactive Document Creation with Wrap

Professional plan

V

gosovych@gmail.com

V

Conversation in Ukrainian

9+

Shared

V

Давай українською

Edit

Звичайно, я з радістю спілкуватимусь українською мовою! Чим я можу вам допомогти сьогодні?

Retry

V

Розкажи хто ти і які в тебе переваги над іншими популярними ai

Edit

Я - Claude, асистент зі штучного інтелекту, створений компанією Anthropic. Я є частиною родини моделей Claude 3, а конкретно я - Claude 3.7 Sonnet, випущений у лютому 2025 року.

Щодо моїх переваг над іншими популярними AI:

Я маю глибоке розуміння контексту та можу вести тривалі осмислені розмови, підтримуючи цілісність діалогу.

Я вмію мислити поетапно при вирішенні складних проблем, що дозволяє мені надавати виважені та обґрунтовані відповіді.

Я можу спілкуватися багатьма мовами, включаючи українську, як ви бачите зараз.

Я добре працюю з різними типами контенту: від написання текстів і аналізу даних до створення коду та відповідей на складні концептуальні запитання.

Я розроблений з урахуванням етичних принципів, щоб бути корисним, правдивим і безпечним у використанні.

Як модель Claude 3.7 Sonnet, я маю покращені можливості розуміння та аналізу порівняно з попередніми версіями.

Чи є щось конкретне, про що ви хотіли б дізнатися більше або з чим я міг би вам допомогти?

Retry

V

ти добре пишеш код?

Edit

Так, я досить добре пишу код. Я можу працювати з багатьма мовами програмування, включаючи Python, JavaScript, Java, C++, HTML/CSS, SQL та інші.

Я здатний:

Створювати функції та класи для конкретних завдань

Аналізувати та виправляти помилки в існуючому коді

Оптимізувати код для кращої продуктивності

Пояснювати, як працює певний алгоритм чи рішення

Допомагати з проектуванням архітектури програмного забезпечення

Хоча я маю певні обмеження (наприклад, я не можу виконувати код або взаємодіяти безпосередньо з файловою системою), я можу надати вам готові рішення, які ви зможете використовувати у своїх проектах.

Чи є конкретне завдання з програмування, з яким я міг би вам допомогти?

Retry

V

я хочу розгорнути проект, це додаток під android та iOS

Edit

Зрозуміло! Розгортання мобільного додатку для Android та iOS — це важливий процес. Я можу допомогти вам з основними кроками та порадами щодо цього.

Для успішного розгортання вам потрібно врахувати такі аспекти:

1. Підготовка додатку до публікації:

Завершіть всі функції та переконайтеся, що додаток стабільно працює

Виконайте тестування на різних пристроях

Переконайтеся, що всі ресурси (зображення, тексти) готові та оптимізовані

2. Для Android (Google Play Store):

Створіть або увійдіть в обліковий запис розробника Google ($25 одноразово)

Підготуйте APK або Android App Bundle файл

Створіть сторінку додатку з описом, скріншотами, іконкою

Налаштуйте ціноутворення, категорії, вікові обмеження

Пройдіть процес перевірки Google

3. Для iOS (App Store):

Зареєструйте обліковий запис Apple Developer ($99 на рік)

Підготуйте додаток в Xcode

Створіть профілі дистрибуції та сертифікати

Налаштуйте App Store Connect з метаданими, скріншотами

Пройдіть процес перевірки Apple (який зазвичай суворіший ніж у Google)

4. Додаткові міркування:

Налаштуйте аналітику для відстеження використання

Підготуйте стратегію оновлень

Розгляньте варіанти монетизації (реклама, платні функції, підписки)

Яку технологію ви використовуєте для розробки вашого додатку? Це нативна розробка окремо для кожної платформи, чи крос-платформене рішення (React Native, Flutter, тощо)?

Retry

V

Спочатку потрібно його розробити

Edit

Звичайно, ви праві. Перш ніж розгортати додаток, його потрібно спочатку розробити. Давайте розглянемо основні етапи розробки мобільного додатку для Android та iOS:

1. Планування та дизайн:

Визначте цілі та функціональність вашого додатку

Створіть user stories та визначте вимоги

Розробіть дизайн інтерфейсу (UI/UX)

Створіть прототип для тестування концепції

2. Вибір технології розробки:

Нативна розробка:

Android: Java або Kotlin + Android Studio

iOS: Swift або Objective-C + Xcode

Крос-платформена розробка:

Flutter (використовує Dart)

React Native (використовує JavaScript)

Xamarin (використовує C#)

3. Розробка:

Налаштування середовища розробки

Розробка основних функцій додатку

Інтеграція серверної частини (якщо потрібно)

Підключення API (за необхідності)

Налаштування бази даних

4. Тестування:

Модульне тестування

Інтеграційне тестування

UI тестування

Бета-тестування з реальними користувачами

5. Підготовка до запуску:

Оптимізація продуктивності

Забезпечення безпеки даних

Налаштування аналітики

Який тип додатку ви плануєте розробляти? Чи маєте вже певну ідею щодо функціональності та технологій, які хочете використовувати?

Retry

V

Чи можеш ти проаналізувати вже існуючий додаток?

Edit

Так, я можу допомогти проаналізувати існуючий додаток. Це важливий крок як для розуміння конкурентів, так і для вдосконалення власного продукту.

Для проведення аналізу мені потрібна інформація про додаток:

Назва додатку та платформи, на яких він доступний

Категорія (ігри, продуктивність, соціальні мережі тощо)

Основні функції та особливості

Цільова аудиторія

Бізнес-модель (безкоштовний, платний, freemium, з рекламою)

Під час аналізу я можу розглянути:

Користувацький інтерфейс та досвід використання

Основні функції та їх реалізацію

Технічні аспекти (швидкість, стабільність, використання ресурсів)

Бізнес-модель та монетизацію

Відгуки користувачів та проблемні місця

Конкурентні переваги та недоліки

Можливості для вдосконалення

Якщо ви можете надати більше інформації про додаток, який хочете проаналізувати, я зможу надати більш конкретний та детальний аналіз.

Retry

V

Додаток називається wishlist на iOS

Edit

Зрозуміло, я проаналізую додаток "Wishlist" для iOS. На платформі iOS є кілька додатків з назвою "Wishlist", тому я проаналізую загальні характеристики типових додатків цієї категорії.

Аналіз додатку Wishlist для iOS:

Загальна концепція і призначення:

Додаток створений для відстеження бажаних покупок і створення персоналізованих списків бажань

Допомагає користувачам організувати свої бажання для себе або для поширення серед друзів і родини

Типові функції:

Створення списків бажань з різними категоріями або тематиками

Додавання предметів з фото, описом, ціною та посиланнями на магазини

Можливість сканування штрих-кодів для додавання товарів

Відстеження цін та повідомлення про знижки

Поширення списків з друзями та родиною

Інтеграція з популярними магазинами або маркетплейсами

Синхронізація між пристроями через iCloud

Інтерфейс та UX:

Ймовірно має чистий, мінімалістичний дизайн, характерний для iOS-додатків

Інтуїтивна навігація між різними списками

Функція швидкого додавання нових предметів

Можливість сортування і фільтрування елементів

Бізнес-модель:

Можливо безкоштовна базова версія з платними преміум-функціями

Альтернативно, може мати партнерські програми з магазинами (affiliate-посилання)

Може містити платні додаткові теми або інструменти організації

Потенційні обмеження:

Можливо обмежена інтеграція з певними магазинами

Залежність від iOS екосистеми

Ймовірно, відсутність крос-платформеної підтримки (лише для iOS)

Можливості для вдосконалення:

Додавання соціальних функцій (групові списки бажань для подій)

Покращена інтеграція з більшою кількістю магазинів

Розширені функції відстеження цін та повідомлень

Додавання ШІ-рекомендацій на основі існуючих вподобань

Інтеграція з календарем для нагадувань про особливі події

Якщо ви маєте на увазі конкретний додаток Wishlist або хочете зосередитись на певних аспектах аналізу, я можу надати більш детальну інформацію за цими напрямками.

Retry

V

Я хочу створити схожий додаток, але краще і із зручнішим функціоналом

Edit

Це чудова ідея! Створення покращеної версії додатку Wishlist може бути перспективним проектом. Ось як ми можемо підійти до розробки кращого додатку:

Концепція покращеного Wishlist додатку:

1. Розширений функціонал:

Автоматичне відстеження цін у різних магазинах для одного й того ж товару

Розумні сповіщення про знижки з прогнозуванням оптимального часу для покупки

Інтеграція з календарем для нагадувань про дні народження та свята

Можливість створення групових списків бажань для спільних подій

Функція резервування елементів у списку бажань для запобігання дублювання подарунків

2. Покращений UX/UI:

Інтуїтивний інтерфейс з мінімальною кількістю кроків для додавання товарів

Персоналізовані теми та налаштування інтерфейсу

Режим перегляду товарів у AR для кращого уявлення про розмір і вигляд

Швидке додавання товарів через розпізнавання зображень

Голосове управління для додавання елементів без набору тексту

3. Соціальні функції та інтеграції:

Безпечне поширення списків серед друзів та родини через різні канали

Інтеграція з соціальними мережами для легкого доступу до вподобань друзів

Можливість додавання коментарів та пропозицій до елементів списку

Інтеграція з популярними маркетплейсами України та міжнародними платформами

4. Розумні рекомендації:

Система рекомендацій на основі ваших вподобань та попередніх елементів

Категоризація елементів за типами, ціновими діапазонами тощо

Сезонні пропозиції та рекомендації на основі тенденцій

5. Розширені можливості організації:

Створення різних списків за категоріями та цілями

Встановлення пріоритетів для елементів

Фінансове планування та бюджетування для досягнення цілей

Технічні аспекти розробки:

Нативна розробка для iOS з використанням Swift та SwiftUI для сучасного інтерфейсу

Використання Core ML для функцій машинного навчання та рекомендацій

Інтеграція з ARKit для функцій доповненої реальності

Розробка серверної частини для синхронізації даних та відстеження цін

Інтеграція з API різних магазинів для автоматичного оновлення цін

Шляхи монетизації:

Freemium модель: базові функції безкоштовно, розширені функції за підпискою

Партнерські програми з магазинами для отримання комісії від покупок

Преміум-теми та розширені інструменти організації як одноразові покупки

З чого б ви хотіли почати розробку? Можемо розробити більш детальний план, прототип інтерфейсу або специфікацію функцій.

Retry

V

Мені потрібно розробити додаток під андроїд та ios

Edit

Зрозуміло. Для розробки вашого покращеного додатку Wishlist для обох платформ - Android та iOS - є кілька підходів. Розгляньмо найбільш ефективні варіанти:

Варіанти розробки для обох платформ:

1. Крос-платформна розробка:

Flutter + Dart:

Найкраще підходить для створення додатків з майже ідентичним інтерфейсом на обох платформах

Висока продуктивність та нативний вигляд

Великий набір готових компонентів для швидкої розробки

Одна кодова база для обох платформ

React Native + JavaScript/TypeScript:

Добрий вибір, якщо у вас є досвід роботи з JavaScript/React

Використовує нативні компоненти для кожної платформи

Велика спільнота та багато готових бібліотек

2. Нативна розробка:

Android: Kotlin + Android Studio

iOS: Swift + Xcode

Переваги кожного підходу:

Крос-платформна розробка:

Швидший час виходу на ринок

Нижча вартість розробки

Менша команда розробників

Легша підтримка та оновлення

Нативна розробка:

Максимальна продуктивність

Повний доступ до всіх API пристрою

Найкраща інтеграція з екосистемою платформи

Більш відточений користувацький досвід

Ключові компоненти для вашого додатку:

Інтерфейс користувача:

Головний екран зі списками бажань

Сторінка деталей товару

Інтерфейс пошуку та фільтрації

Система сповіщень

Серверна частина:

База даних для зберігання списків та елементів

API для синхронізації між пристроями

Система для відстеження цін

Інтеграції:

API популярних магазинів

Платіжні системи (якщо плануєте покупки через додаток)

Соціальні мережі для поширення

Для початку, який у вас досвід розробки і з якими технологіями ви найбільш знайомі? Це допоможе визначити оптимальний підхід і наступні кроки для вашого проекту.

Retry

V

давай будемо робити це на React Native + JavaScript

Edit

Чудовий вибір! React Native + JavaScript - це дійсно ефективне рішення для крос-платформної розробки. Ось план розробки вашого Wishlist додатку з використанням цієї технології:

План розробки Wishlist додатку на React Native:

1. Налаштування середовища розробки:

Встановлення Node.js та npm

Встановлення React Native CLI

Налаштування Android Studio для Android-емулятора

Налаштування Xcode для iOS-симулятора (потрібен Mac)

Налаштування VSCode або іншого IDE з відповідними плагінами

2. Створення проекту та базова структура:

Ініціалізація проекту React Native

Налаштування навігації (React Navigation)

Налаштування управління станом (Redux або Context API)

Створення базової структури папок і файлів

3. Розробка ключових компонентів UI:

Екран авторизації/реєстрації

Головний екран зі списками бажань

Екран деталей товару

Екран пошуку та додавання товарів

Профіль користувача та налаштування

4. Функціональність додатку:

Система авторизації та управління профілем

CRUD-операції для списків бажань та їхніх елементів

Пошук товарів та додавання їх до списків

Система відстеження цін

Поширення списків та соціальні функції

5. Інтеграції з API:

Створення власного бекенду або використання готових рішень (Firebase)

Інтеграція з API магазинів для отримання даних про товари

Інтеграція з платіжними системами (за необхідності)

6. Тестування та оптимізація:

Тестування на різних пристроях Android та iOS

Оптимізація продуктивності

Перевірка UI/UX на відповідність до гайдлайнів обох платформ

7. Підготовка до релізу:

Підготовка зображень, іконок, описів

Налаштування конфігурацій для обох платформ

Підготовка APK для Android та IPA для iOS

Рекомендовані бібліотеки для проекту:

Copy

// Основні залежності

react-native

react-navigation/native (для навігації)

redux (для управління станом)

redux-thunk або redux-saga (для асинхронних операцій)

react-native-vector-icons (для іконок)

react-native-firebase (якщо використовуватимете Firebase)

// UI-компоненти

react-native-elements або native-base (набори UI-компонентів)

react-native-gesture-handler (для жестів)

react-native-reanimated (для анімацій)

// Інтеграції

react-native-camera (для сканування штрих-кодів)

react-native-image-picker (для роботи з фото)

react-native-share (для поширення контенту)

Початкова структура проекту:

Copy

/src

/assets # Зображення, шрифти тощо

/components # Перевикористовувані компоненти

/screens # Екрани додатку

/navigation # Конфігурація навігації

/redux # Redux store, actions, reducers

/services # API та інші сервіси

/utils # Допоміжні функції

/hooks # Користувацькі хуки

/config # Конфігураційні файли

App.js # Кореневий компонент

З чого би ви хотіли почати? Можу допомогти з базовою структурою проекту, створенням прототипу інтерфейсу або детальним планом розробки конкретних компонентів.

Retry

V

ти маєш запамятати що ти будеш моїм розробником так як я не маю досвіду в розробці я хочу щоб ти мені зробив це від початку до саого готового додатку

Edit

Зрозуміло! Я буду вашим розробником і допоможу створити додаток Wishlist від початку до готового продукту. Оскільки ви не маєте досвіду в розробці, я проведу вас через весь процес і поясню кожен етап.

Давайте почнемо з найперших кроків розробки на React Native:

Крок 1: Встановлення необхідного програмного забезпечення

Для початку роботи над React Native проектом вам потрібно встановити:

Node.js і npm (Node Package Manager)

React Native CLI

Android Studio (для Android-розробки)

Xcode (для iOS-розробки, потрібен Mac)

Редактор коду (я рекомендую Visual Studio Code)

Чи маєте ви вже встановлене якесь із цього програмного забезпечення? Також, на якій операційній системі ви працюєте зараз (Windows, macOS, Linux)? Це допоможе мені надати точні інструкції для встановлення.

Далі, нам потрібно буде:

Створити базовий проект React Native

Налаштувати структуру проекту

Запустити додаток на емуляторі/симуляторі

Починати розробку компонентів та екранів

Після того, як ми розберемося з вашим поточним станом, я підготую детальний план і почнемо поступову розробку вашого додатку Wishlist.

Retry

V

давай почнемо із першого кроку

Edit

Чудово! Почнемо з першого кроку — встановлення необхідного програмного забезпечення для React Native розробки.

Крок 1: Встановлення необхідного програмного забезпечення

1. Встановлення Node.js і npm

Node.js — це середовище виконання JavaScript поза браузером, а npm — це менеджер пакетів для JavaScript.

Перейдіть на офіційний сайт Node.js

Завантажте та встановіть LTS (Long Term Support) версію

Після встановлення відкрийте термінал або командний рядок і перевірте встановлення:

Copy

node -v

npm -v

Ви повинні побачити версії Node.js та npm, якщо встановлення пройшло успішно.

2. Встановлення React Native CLI

React Native CLI — це інструмент командного рядка для створення та управління React Native проектами.

Відкрийте термінал і виконайте команду:

Copy

npm install -g react-native-cli

Перевірте встановлення:

Copy

react-native --version

3. Встановлення JDK (Java Development Kit)

Для Android-розробки потрібен JDK:

Завантажте та встановіть JDK 11 з Oracle або використовуйте OpenJDK

Встановіть змінну середовища JAVA\_HOME, що вказує на каталог JDK

4. Встановлення Android Studio

Для розробки Android-частини додатку:

Завантажте та встановіть Android Studio

Під час встановлення виберіть "Custom" і переконайтеся, що відмічені:

Android SDK

Android SDK Platform

Android Virtual Device (AVD)

Після встановлення відкрийте Android Studio, перейдіть до SDK Manager і встановіть:

Android SDK Platform 29 (або новішу)

Android SDK Build-Tools

Intel HAXM (для прискорення емуляції)

5. Налаштування змінних середовища для Android

Вам потрібно додати наступні змінні середовища:

ANDROID\_HOME - шлях до вашого Android SDK (зазвичай C:\Users\YOUR\_USERNAME\AppData\Local\Android\Sdk на Windows або /Users/YOUR\_USERNAME/Library/Android/sdk на macOS)

Додайте до PATH:

%ANDROID\_HOME%\platform-tools (Windows)

%ANDROID\_HOME%\tools (Windows)

$ANDROID\_HOME/platform-tools (macOS/Linux)

$ANDROID\_HOME/tools (macOS/Linux)

6. Встановлення Visual Studio Code (VS Code)

Завантажте та встановіть VS Code

Встановіть корисні розширення для React Native:

React Native Tools

ESLint

Prettier

JavaScript and TypeScript Intellisense

7. Для розробки під iOS (лише для macOS)

Якщо ви працюєте на Mac і хочете розробляти для iOS:

Встановіть Xcode з App Store

Встановіть інструменти командного рядка Xcode:

Copy

xcode-select --install

Встановіть CocoaPods:

Copy

sudo gem install cocoapods

Коли всі ці компоненти будуть встановлені, ви будете готові до створення вашого першого React Native проекту. Дайте мені знати, коли завершите цей етап або якщо маєте труднощі з будь-яким із кроків, і ми перейдемо до наступного кроку — створення базового проекту React Native.

Retry

V

я буду використовувати додаток Warp врахуй це

Edit

Дякую за цю інформацію! Використання Warp — це чудовий вибір, оскільки він є сучасним альтернативним терміналом з численними покращеннями порівняно з традиційними терміналами.

Враховуючи використання Warp, ось оновлені інструкції для Кроку 1:

Крок 1: Встановлення необхідного програмного забезпечення (з використанням Warp)

1. Встановлення Node.js і npm

Оскільки ви використовуєте Warp, ви можете скористатися його покращеним інтерфейсом для виконання команд:

Відкрийте Warp

Для встановлення Node.js ви можете використати nvm (Node Version Manager) для кращого управління версіями. В терміналі Warp виконайте:

bash

Copy

# Встановлення nvm (для macOS/Linux)

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash

# Для Windows можна використати nvm-windows, інсталятор можна завантажити з GitHub

# Після встановлення nvm, встановіть актуальну LTS версію Node.js

nvm install --lts

# Перевірка встановлення

node -v

npm -v

2. Встановлення React Native CLI

У Warp виконайте:

bash

Copy

npm install -g react-native-cli

Перевірте встановлення:

bash

Copy

react-native --version

3-6. Інсталяція JDK, Android Studio, VS Code та налаштування змінних

Ці кроки залишаються в основному такими ж, як описано раніше. Warp буде корисним для перевірки та налаштування змінних середовища.

Після інсталяції Android Studio ви можете перевірити налаштування змінних середовища у Warp:

bash

Copy

# Перевірка змінної ANDROID\_HOME

echo $ANDROID\_HOME

# Додавання змінних середовища в ~/.zshrc або ~/.bash\_profile (для macOS/Linux)

echo 'export ANDROID\_HOME=$HOME/Library/Android/sdk' >> ~/.zshrc

echo 'export PATH=$PATH:$ANDROID\_HOME/platform-tools' >> ~/.zshrc

echo 'export PATH=$PATH:$ANDROID\_HOME/tools' >> ~/.zshrc

source ~/.zshrc

# Перевірка налаштувань

echo $PATH

Переваги використання Warp для React Native розробки:

Автодоповнення: Warp має покращене автодоповнення команд

Блоки команд: Легше відрізнити різні команди та їх вивід

Інтеграція з AI: Допомагає знаходити потрібні команди

Історія команд: Краще управління історією виконаних команд

Після встановлення всіх компонентів, давайте перевіримо, що все налаштовано правильно. У Warp виконайте:

bash

Copy

# Перевірка Node.js та npm

node -v

npm -v

# Перевірка React Native CLI

react-native --version

# Перевірка Java (для Android розробки)

java -version

# Перевірка Android SDK

$ANDROID\_HOME/tools/bin/sdkmanager --list

Коли підтвердите, що всі компоненти встановлені правильно, ми перейдемо до створення нового проекту React Native.

Retry

V

react-native --version

react-native-cli: 2.0.1

react-native: n/a - not inside a React Native project directory

echo $ANDROID\_HOME

echo 'export ANDROID\_HOME=$HOME/Library/Android/sdk' >> ~/.zshrc

echo 'export PATH=$PATH:$ANDROID\_HOME/platform-tools' >> ~/.zshrc

echo 'export PATH=$PATH:$ANDROID\_HOME/tools' >> ~/.zshrc

source ~/.zshrc

echo $PATH

/usr/local/opt/mongodb-community@6.0/bin:/usr/local/opt/mongodb-community@6.0/bin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin:/Library/Apple/usr/bin:/Users/test/Library/Android/sdk/platform-tools:/Users/test/Library/Android/sdk/tools

Edit

Чудово! Я бачу, що ви успішно встановили React Native CLI (версія 2.0.1) та правильно налаштували змінні середовища для Android розробки. Шлях $PATH тепер включає необхідні директорії Android SDK.

Тепер можемо перейти до наступного кроку — створення базового проекту React Native для нашого додатку Wishlist.

Крок 2: Створення проекту React Native

Давайте створимо новий проект React Native за допомогою CLI. У вашому терміналі Warp виконайте:

bash

Copy

# Створення нового проекту

npx react-native init WishlistApp

# Перейти в директорію проекту

cd WishlistApp

# Перевірити структуру проекту

ls -la

Цей процес може зайняти кілька хвилин, оскільки він завантажує та встановлює всі необхідні залежності.

Після створення проекту, давайте переконаємося, що він працює, запустивши його на емуляторі Android або iOS:

Для Android:

bash

Copy

# Запустити Android емулятор (попередньо створений в Android Studio)

# або підключити реальний пристрій

# Запустити додаток на Android

npx react-native run-android

Для iOS (якщо у вас Mac):

bash

Copy

# Встановити залежності iOS

cd ios && pod install && cd ..

# Запустити додаток на iOS симуляторі

npx react-native run-ios

Якщо все працює правильно, ви повинні побачити екран привітання React Native на емуляторі/симуляторі.

Дайте мені знати, коли успішно створите та запустите проект, і ми перейдемо до налаштування структури проекту та встановлення необхідних залежностей для нашого Wishlist додатку.

Retry

V

Я хочу втановити проект на шлях /Users/test/Documents

Edit

Звичайно, давайте встановимо проект у вказану директорію.

bash

Copy

# Перейдіть до потрібного шляху

cd /Users/test/Documents

# Створення нового проекту в цій директорії

npx react-native init WishlistApp

# Перейти в директорію проекту

cd WishlistApp

# Перевірити структуру проекту

ls -la

Після виконання цих команд, ваш новий проект React Native буде створено в директорії /Users/test/Documents/WishlistApp.

Як тільки процес створення проекту завершиться, ви можете запустити додаток для перевірки, що все працює коректно:

Для Android:

bash

Copy

# Запустити додаток на Android

npx react-native run-android

Для iOS (якщо у вас Mac):

bash

Copy

# Встановити залежності iOS

cd ios && pod install && cd ..

# Запустити додаток на iOS симуляторі

npx react-native run-ios

Дайте мені знати результат виконання цих команд, і ми продовжимо розробку вашого Wishlist додатку.

Retry

npx react-native init WishlistApp This will walk you through creating a new React Native project in /Users/test/Documents/WishlistApp Using yarn v1.22.22 Installing react-native... yarn add v1.22.22 i

pasted

V

Edit

Бачу, що у вас виникла помилка при створенні проекту з використанням react-native-cli. Схоже, у вас встановлена застаріла версія CLI, яка несумісна з новішими версіями React Native. Давайте виправимо це і створимо проект правильним способом.

Спочатку я рекомендую видалити глобально встановлений react-native-cli і використовувати npx для запуску актуальної версії:

bash

Copy

# Видалення застарілої версії CLI

npm uninstall -g react-native-cli

# Створення проекту з використанням npx та @react-native-community/cli

cd /Users/test/Documents

npx react-native@latest init WishlistApp

Якщо ви отримаєте запитання під час ініціалізації, краще вибрати рекомендовані значення.

Після створення проекту перевірте, що все працює:

bash

Copy

cd WishlistApp

# Для iOS (якщо у вас Mac)

cd ios && pod install && cd ..

npx react-native run-ios

# Для Android

npx react-native run-android

Якщо ви все ще отримуєте помилки, можна спробувати альтернативний підхід з використанням Expo, який часто простіший у налаштуванні:

bash

Copy

cd /Users/test/Documents

npx create-expo-app WishlistApp

cd WishlistApp

npx expo start

Дайте мені знати, який з цих підходів спрацював для вас, і ми продовжимо розробку додатку.

Retry

V

npx react-native@latest init WishlistApp

Need to install the following packages:

react-native@0.78.0

Ok to proceed? (y) y

npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.

npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported

npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

🚨️ The init command is deprecated.

- Switch to npx @react-native-community/cli init for the identical behavior.

- Refer to the documentation for information about alternative tools: https://reactnative.dev/docs/getting-started

Exiting...

Edit

Схоже, що команда react-native init застаріла. Давайте використаємо рекомендовану альтернативу:

bash

Copy

cd /Users/test/Documents

npx @react-native-community/cli init WishlistApp

Якщо і ця команда викликає проблеми, можна спробувати створити проект за допомогою Expo, який пропонує більш просте налаштування та інструменти для розробки:

bash

Copy

cd /Users/test/Documents

npx create-expo-app WishlistApp

cd WishlistApp

Після створення проекту з Expo, можна додати бібліотеки React Native і необхідні залежності. Це часто найпростіший шлях для початківців у розробці React Native.

Дайте мені знати, який варіант працює, і ми продовжимо налаштування проекту Wishlist.

Retry

V

мені потрібно почати розробку без конфліктів і все із самими новими способами і ресурсами

Edit

Розумію, вам потрібно почати розробку з найсучаснішими інструментами і без конфліктів. Давайте використаємо Expo, який є найбільш рекомендованим і безпроблемним способом почати розробку React Native у 2025 році:

bash

Copy

# Перейдіть до бажаної директорії

cd /Users/test/Documents

# Створіть новий проект з Expo

npx create-expo-app@latest WishlistApp

# Перейдіть до директорії проекту

cd WishlistApp

Після створення проекту ви можете запустити його за допомогою:

bash

Copy

npx expo start

Цей підхід має кілька переваг:

Простота налаштування - мінімум конфігурації

Вбудовані інструменти - не потрібно налаштовувати окремо симулятори/емулятори

Швидкий старт - можете одразу почати писати код

Доступ до екосистеми Expo - багато готових компонентів і API

Легкий перехід до "чистого" React Native - коли потрібно, можна "виконати eject"

Коли проект буде створено, ми можемо встановити основні залежності для нашого Wishlist додатку:

bash

Copy

npx expo install react-navigation @react-navigation/native @react-navigation/stack

npx expo install react-native-gesture-handler react-native-reanimated

npx expo install @react-native-async-storage/async-storage

Дайте знати, коли проект буде успішно створено, і ми почнемо налаштовувати структуру та компоненти для вашого Wishlist додатку.

Retry

V

перд тим як встановлювати залеженості давай спроектуємо архітектуру проекту, потрібно зробити так, щоб в подальшому ми могли безболісно додвати функції

Edit

Звичайно, це дуже правильний підхід! Перед встановленням залежностей спроектуємо архітектуру проекту так, щоб вона була масштабованою і добре організованою.

Архітектура проекту Wishlist:

1. Структура папок:

Copy

/WishlistApp

/src

/api # API клієнти та сервіси для взаємодії з бекендом

/assets # Статичні ресурси (зображення, шрифти, тощо)

/components # Перевикористовувані компоненти

/common # Загальні компоненти (кнопки, поля введення, тощо)

/layout # Компоненти для розміщення контенту

/wishlist # Компоненти, специфічні для списків бажань

/constants # Константи додатку (кольори, розміри, тощо)

/context # React Context для управління станом додатку

/hooks # Користувацькі React hooks

/navigation # Конфігурація навігації

/screens # Екрани додатку

/auth # Екрани авторизації

/wishlist # Екрани для роботи зі списками бажань

/profile # Екрани профілю користувача

/settings # Екрани налаштувань

/services # Бізнес-логіка та сервіси

/store # Управління станом (якщо використовуємо Redux)

/types # TypeScript типи (якщо використовуємо TypeScript)

/utils # Допоміжні функції

App.js # Головний компонент додатку

/ios # iOS специфічні файли

/android # Android специфічні файли

2. Архітектурні патерни:

Компонентний підхід:

Розділяємо UI на маленькі, перевикористовувані компоненти

Застосовуємо принцип "single responsibility" (одна відповідальність)

Управління станом:

Для локального стану використовуємо React hooks (useState, useReducer)

Для глобального стану використовуємо Context API (для простіших додатків) або Redux (для складніших)

Навігація:

Використовуємо React Navigation для управління переходами між екранами

Організовуємо навігацію за допомогою стеків та вкладених навігаторів

Взаємодія з даними:

Відділяємо логіку API запитів від компонентів

Використовуємо паттерн Repository для роботи з даними

3. Основні модулі додатку:

Авторизація та профіль користувача:

Реєстрація/Вхід

Управління профілем

Налаштування користувача

Управління списками бажань:

Створення/Редагування/Видалення списків

Категоризація списків

Приватні та публічні списки

Управління елементами списку:

Додавання/Редагування/Видалення елементів

Пріоритезація елементів

Статуси елементів (бажаний, придбаний, тощо)

Відстеження цін:

Інтеграція з API магазинів

Сповіщення про зміну цін

Історія цін

Соціальні функції:

Поширення списків

Колаборація на списках

Інтеграція з соціальними мережами

Пошук та рекомендації:

Пошук по товарах

Рекомендації на основі існуючих списків

4. Технічні аспекти:

Локальне зберігання даних:

AsyncStorage для локального кешування

Можливість роботи офлайн

Мережеві запити:

Axios або fetch для API запитів

Обробка помилок та повторні спроби

Безпека:

Безпечне зберігання токенів авторизації

Валідація вхідних даних

Продуктивність:

Використання React.memo для оптимізації

Віртуалізація списків для великих наборів даних

Тестування:

Підготовка архітектури для юніт та інтеграційних тестів

Цей підхід до архітектури дозволить нам гнучко додавати нові функції в майбутньому, забезпечить достатню модульність та дасть можливість легко підтримувати і розширювати додаток.

Чи є у вас якісь конкретні побажання щодо архітектури або ви хочете внести корективи до запропонованого підходу?

Retry

V

Гаразд зараз я організую все так як тиописав

Edit

Чудово! Коли ви створите базову структуру проекту відповідно до запропонованої архітектури, ми зможемо перейти до наступних кроків:

Встановлення необхідних залежностей для кожного модуля

Налаштування навігації між екранами

Розробка основних компонентів інтерфейсу

Імплементація логіки збереження та управління вашими списками бажань

Є кілька підходів до створення такої структури:

Ви можете створити всі папки вручну

Або використати скрипт для автоматичного створення структури

Дайте знати, коли будете готові продовжити, або якщо вам потрібна допомога зі створенням структури.

Retry

Знімок екрана 20250313 о 23.20.06.png

V

ось як виглядає архітектура зараз що мені потрібно зробити щоб була така архітектура як ти описав

Edit

Бачу, що ви вже створили базову структуру проекту з Expo. Тепер нам потрібно реорганізувати її відповідно до архітектури, яку я описав раніше.

Ось що потрібно зробити:

Створіть папку src в корені проекту, яка буде містити всю вашу кодову базу

В папці src створіть підпапки згідно з архітектурою:

bash

Copy

# Перейдіть в корінь проекту

cd /Users/test/Documents/WishlistApp

# Створіть основну структуру папок

mkdir -p src/api

mkdir -p src/assets

mkdir -p src/components/common

mkdir -p src/components/layout

mkdir -p src/components/wishlist

mkdir -p src/constants

mkdir -p src/context

mkdir -p src/hooks

mkdir -p src/navigation

mkdir -p src/screens/auth

mkdir -p src/screens/wishlist

mkdir -p src/screens/profile

mkdir -p src/screens/settings

mkdir -p src/services

mkdir -p src/store

mkdir -p src/types

mkdir -p src/utils

Створіть початкові файли:

bash

Copy

# Створіть головний компонент App в папці src

touch src/App.js

# Створіть основні константи

touch src/constants/colors.js

touch src/constants/theme.js

# Створіть файл навігації

touch src/navigation/index.js

# Створіть базові екрани

touch src/screens/auth/LoginScreen.js

touch src/screens/auth/RegisterScreen.js

touch src/screens/wishlist/WishlistsScreen.js

touch src/screens/wishlist/WishlistDetailScreen.js

touch src/screens/profile/ProfileScreen.js

Перемістіть існуючі ресурси з папки assets у src/assets:

bash

Copy

# Якщо у вас є файли в папці assets, перемістіть їх

mkdir -p src/assets/images

mkdir -p src/assets/fonts

# Копіюйте файли відповідно до їх типу

Змініть головний файл App.js в корені проекту, щоб він імпортував компонент з нової структури:

javascript

Copy

// App.js (в корені проекту)

import App from './src/App';

export default App;

Створіть новий файл src/App.js з початковим кодом:

javascript

Copy

// src/App.js

import React from 'react';

import { StatusBar } from 'expo-status-bar';

import { SafeAreaProvider } from 'react-native-safe-area-context';

import { Text, View } from 'react-native';

export default function App() {

return (

<SafeAreaProvider>

<View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>

<Text>Welcome to Wishlist App!</Text>

<StatusBar style="auto" />

</View>

</SafeAreaProvider>

);

}

Після цих кроків у вас буде правильна структура проекту, і ви зможете почати розробку компонентів та функціоналу. Дайте мені знати, коли виконаєте ці кроки, і ми продовжимо з налаштуванням навігації та основних компонентів.

Retry

V

Змініть головний файл App.js в корені проекту, щоб він імпортував компонент з нової структури, де його знайти?

Edit

В проектах створених з Expo, файл App.js або App.tsx знаходиться в корені проекту. Це той файл, який автоматично був створений системою Expo при ініціалізації проекту.

Вам потрібно відкрити цей файл в корені проекту (не в src, а в корені WishlistApp) та змінити його вміст так, щоб він імпортував App компонент з нової структури, яку ми створили.

Шлях до файлу буде приблизно такий:

Copy

/Users/test/Documents/WishlistApp/App.js

або

Copy

/Users/test/Documents/WishlistApp/App.tsx

Якщо ви відкриєте цей файл в текстовому редакторі, ви побачите початковий код, який був згенерований Expo. Замініть його вміст на:

javascript

Copy

import App from './src/App';

export default App;

Це дозволить використовувати нашу нову структуру проекту, зберігаючи при цьому точку входу, яку очікує Expo.

Retry

Знімок екрана 20250313 о 23.25.31.png

V

ось що є у мене

Edit

Я бачу вашу структуру проекту. У вас вже є директорія app, що може служити як альтернатива папці src. В сучасних Expo проектах часто використовують папку app для file-based маршрутизації.

У вашому випадку є два шляхи:

Використовувати папку src (як я рекомендував): Ви створили папку src, тепер нам треба перенести код з app в нашу нову структуру в src.

Використовувати існуючу папку app: Ми можемо адаптувати нашу архітектуру до структури app, створивши потрібні підпапки там.

Оскільки ви вже створили папку src, я рекомендую переходити на неї для кращої організації коду.

Для цього вам потрібно знайти або створити файл App.js (або App.tsx) у корені проекту. Якщо він уже є, відредагуйте його для імпорту з нової структури. Якщо його немає, створіть новий.

javascript

Copy

// App.js або App.tsx в корені проекту

import App from './src/App';

export default App;

Дайте мені знати, чи є у вас файл App.js в корені проекту, або чи хочете ви замість цього використати існуючу структуру app?

Retry

V

а папку app видалити?

Edit

Так, якщо ви переходите на архітектуру з папкою src, то папку app можна видалити, але спочатку переконайтеся, що ви збережете всі важливі компоненти та файли, які можуть бути вам потрібні.

Ось як можна діяти:

Переконайтеся, що всі важливі компоненти з папки app перенесені у відповідні директорії в src

Перевірте, що основний файл App.js у корені проекту імпортує додаток з нової структури src

Спробуйте запустити додаток щоб переконатися, що все працює з новою структурою

Лише після успішного запуску з новою структурою можна видалити папку app

Якщо у вас виникнуть проблеми з запуском після переходу на нову структуру, завжди можна повернутися до попередньої конфігурації.

Retry

V

Logs for your project will appear below. Press Ctrl+C to exit.

Error: ENOENT: no such file or directory, scandir '/Users/test/Documents/WishlistApp/assets/images'

at Object.readdir (node:internal/fs/promises:948:18)

at getAbsoluteAssetRecord (/Users/test/Documents/WishlistApp/node\_modules/metro/src/Assets.js:77:17)

at getAsset (/Users/test/Documents/WishlistApp/node\_modules/metro/src/Assets.js:191:18)

at Server.\_processSingleAssetRequest (/Users/test/Documents/WishlistApp/node\_modules/metro/src/Server.js:335:20)

at Server.\_processRequest (/Users/test/Documents/WishlistApp/node\_modules/metro/src/Server.js:417:7)

Unable to resolve asset "./assets/images/icon.png" from "icon" in your app.json or app.config.js

iOS Bundling failed 977ms node\_modules/expo-router/entry.js (1019 modules)

Unable to resolve "@/assets/images/react-logo.png" from "app/(tabs)/explore.tsx"

Edit

Бачу, що у вас виникає помилка через те, що файли зображень неможливо знайти за вказаними шляхами. Це пов'язано з переміщенням ресурсів у нову структуру.

Давайте виправимо ці проблеми:

Помилка з app.json: Вам потрібно оновити шляхи до ресурсів у файлі app.json. Відкрийте цей файл і змініть шляхи до зображень:

json

Copy

// app.json

{

"expo": {

// Інші поля...

"icon": "./src/assets/images/icon.png",

"splash": {

"image": "./src/assets/images/splash.png",

// Інші поля splash...

}

// Інші поля...

}

}

Переконайтеся, що ці файли існують:

Перевірте, чи створили ви папку src/assets/images

Скопіюйте необхідні зображення (icon.png, splash.png, react-logo.png та інші) з папки assets в нову папку src/assets/images

Оновлення імпортів у компонентах: В файлі app/(tabs)/explore.tsx замініть імпорт

javascript

Copy

import ... from "@/assets/images/react-logo.png"

на

javascript

Copy

import ... from "../../src/assets/images/react-logo.png"

або налаштуйте аліаси в tsconfig.json (якщо ви використовуєте TypeScript).

Створіть необхідні директорії та файли:

bash

Copy

mkdir -p src/assets/images

# Скопіюйте файли з поточної директорії assets/images (якщо вона існує)

cp -R assets/images/\* src/assets/images/ 2>/dev/null || :

Якщо файли із зображеннями відсутні, вам потрібно створити їх або отримати з іншого джерела. Для тестування можна тимчасово використати заглушки:

bash

Copy

# Створення пустого зображення icon.png

touch src/assets/images/icon.png

# Так само для інших необхідних зображень

touch src/assets/images/splash.png

touch src/assets/images/react-logo.png

Після цього спробуйте запустити додаток знову:

bash

Copy

npx expo start

Retry

V

це виникає через демо контент чи можемо ми очистити наш проект від нього і як це зробити\*

Edit

Так, ви праві. Ці помилки виникають через демо-контент, який включено в базовий шаблон проекту Expo. Давайте очистимо проект від демо-контенту та підготуємо його для нашого додатку Wishlist.

Ось кроки для очищення проекту:

Видаліть папку app з демо-контентом:

bash

Copy

rm -rf app

Оновіть файл app.json, щоб прибрати посилання на файли, які не існують: Відкрийте файл app.json і спростіть його. Ось приклад оновленої версії:

json

Copy

{

"expo": {

"name": "WishlistApp",

"slug": "wishlistapp",

"version": "1.0.0",

"orientation": "portrait",

"icon": "./src/assets/icon.png",

"userInterfaceStyle": "light",

"splash": {

"image": "./src/assets/splash.png",

"resizeMode": "contain",

"backgroundColor": "#ffffff"

},

"assetBundlePatterns": [

"\*\*/\*"

],

"ios": {

"supportsTablet": true

},

"android": {

"adaptiveIcon": {

"foregroundImage": "./src/assets/adaptive-icon.png",

"backgroundColor": "#ffffff"

}

}

}

}

Створіть базові файли зображень для уникнення помилок:

bash

Copy

mkdir -p src/assets

# Скопіюйте необхідні зображення з default assets Expo

cp -r node\_modules/expo/assets/\* src/assets/ 2>/dev/null || :

touch src/assets/icon.png

touch src/assets/splash.png

touch src/assets/adaptive-icon.png

Створіть простий App.js в корені проекту:

javascript

Copy

import App from './src/App';

export default App;

Створіть базовий src/App.js:

javascript

Copy

import React from 'react';

import { StyleSheet, Text, View } from 'react-native';

export default function App() {

return (

<View style={styles.container}>

<Text style={styles.text}>Welcome to Wishlist App!</Text>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#fff',

alignItems: 'center',

justifyContent: 'center',

},

text: {

fontSize: 20,

fontWeight: 'bold',

},

});

Коли ви виконаєте ці кроки, ваш проект буде очищений від демо-контенту і підготовлений для розробки нашого Wishlist додатку з чистого аркуша. Тепер ми можемо почати розробку за нашою архітектурою без перешкод.

Після очищення проекту запустіть його для перевірки:

bash

Copy

npx expo start

Retry

V

Супер

Edit

Чудово! Тепер, коли ми очистили проект від демо-контенту, ми готові почати розробку нашого Wishlist додатку відповідно до спроектованої архітектури.

Давайте продовжимо з наступних кроків:

Встановимо необхідні пакети для нашого проекту:

bash

Copy

# Навігація

npx expo install @react-navigation/native @react-navigation/stack @react-navigation/bottom-tabs

npx expo install react-native-screens react-native-safe-area-context

# Керування станом

npx expo install @react-native-async-storage/async-storage

# UI компоненти

npx expo install react-native-paper

npx expo install react-native-vector-icons

# Форми та валідація

npx expo install react-hook-form

Налаштуємо базову навігацію

Давайте створимо файл /src/navigation/index.js:

javascript

Copy

import React from 'react';

import { NavigationContainer } from '@react-navigation/native';

import { createStackNavigator } from '@react-navigation/stack';

import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

import HomeScreen from '../screens/wishlist/WishlistsScreen';

import WishlistDetailScreen from '../screens/wishlist/WishlistDetailScreen';

import ProfileScreen from '../screens/profile/ProfileScreen';

import { View, Text } from 'react-native';

// Тимчасові заглушки для екранів, які ще не створені

const SettingsScreen = () => (

<View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>

<Text>Settings Screen</Text>

</View>

);

const Stack = createStackNavigator();

const Tab = createBottomTabNavigator();

// Навігація зі списками бажань

const WishlistStack = () => (

<Stack.Navigator>

<Stack.Screen name="Wishlists" component={HomeScreen} />

<Stack.Screen name="WishlistDetail" component={WishlistDetailScreen} />

</Stack.Navigator>

);

// Головна навігація з вкладками

const MainNavigator = () => (

<Tab.Navigator>

<Tab.Screen name="Home" component={WishlistStack} />

<Tab.Screen name="Profile" component={ProfileScreen} />

<Tab.Screen name="Settings" component={SettingsScreen} />

</Tab.Navigator>

);

export default function Navigation() {

return (

<NavigationContainer>

<MainNavigator />

</NavigationContainer>

);

}

Оновимо наш App.js для використання навігації:

javascript

Copy

import React from 'react';

import { SafeAreaProvider } from 'react-native-safe-area-context';

import Navigation from './navigation';

import { StatusBar } from 'expo-status-bar';

export default function App() {

return (

<SafeAreaProvider>

<Navigation />

<StatusBar style="auto" />

</SafeAreaProvider>

);

}

Створимо заглушки для основних екранів:

Файл /src/screens/wishlist/WishlistsScreen.js:

javascript

Copy

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity } from 'react-native';

export default function WishlistsScreen({ navigation }) {

return (

<View style={styles.container}>

<Text style={styles.title}>My Wishlists</Text>

<TouchableOpacity

style={styles.item}

onPress={() => navigation.navigate('WishlistDetail', { id: 1, name: 'Birthday Wishlist' })}

>

<Text style={styles.itemText}>Birthday Wishlist</Text>

</TouchableOpacity>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemText: {

fontSize: 18,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

Файл /src/screens/wishlist/WishlistDetailScreen.js:

javascript

Copy

import React from 'react';

import { View, Text, StyleSheet, FlatList } from 'react-native';

export default function WishlistDetailScreen({ route }) {

const { name } = route.params;

const dummyItems = [

{ id: '1', name: 'iPhone 14', price: 999, priority: 'High' },

{ id: '2', name: 'MacBook Pro', price: 1999, priority: 'Medium' },

{ id: '3', name: 'AirPods Pro', price: 249, priority: 'Low' },

];

return (

<View style={styles.container}>

<Text style={styles.title}>{name}</Text>

<FlatList

data={dummyItems}

keyExtractor={(item) => item.id}

renderItem={({ item }) => (

<View style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemPrice}>${item.price}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

)}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemPrice: {

fontSize: 16,

color: '#666',

marginTop: 4,

},

itemPriority: {

fontSize: 14,

marginTop: 8,

fontWeight: '500',

},

});

Файл /src/screens/profile/ProfileScreen.js:

javascript

Copy

import React from 'react';

import { View, Text, StyleSheet, Image } from 'react-native';

export default function ProfileScreen() {

return (

<View style={styles.container}>

<View style={styles.header}>

<View style={styles.avatarContainer}>

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>JD</Text>

</View>

</View>

<Text style={styles.name}>John Doe</Text>

<Text style={styles.email}>john.doe@example.com</Text>

</View>

<View style={styles.statsContainer}>

<View style={styles.statItem}>

<Text style={styles.statNumber}>5</Text>

<Text style={styles.statLabel}>Wishlists</Text>

</View>

<View style={styles.statItem}>

<Text style={styles.statNumber}>42</Text>

<Text style={styles.statLabel}>Items</Text>

</View>

<View style={styles.statItem}>

<Text style={styles.statNumber}>3</Text>

<Text style={styles.statLabel}>Shared</Text>

</View>

</View>

<View style={styles.section}>

<Text style={styles.sectionTitle}>Account Settings</Text>

<View style={styles.sectionItem}>

<Text style={styles.sectionItemText}>Edit Profile</Text>

</View>

<View style={styles.sectionItem}>

<Text style={styles.sectionItemText}>Notifications</Text>

</View>

<View style={styles.sectionItem}>

<Text style={styles.sectionItemText}>Privacy</Text>

</View>

<View style={styles.sectionItem}>

<Text style={styles.sectionItemText}>Log Out</Text>

</View>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

},

avatarContainer: {

marginBottom: 10,

},

avatarPlaceholder: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

fontSize: 40,

fontWeight: 'bold',

color: '#4a90e2',

},

name: {

color: '#fff',

fontSize: 24,

fontWeight: 'bold',

marginVertical: 5,

},

email: {

color: '#fff',

fontSize: 16,

},

statsContainer: {

flexDirection: 'row',

backgroundColor: '#fff',

padding: 15,

justifyContent: 'space-around',

borderBottomWidth: 1,

borderBottomColor: '#eee',

},

statItem: {

alignItems: 'center',

},

statNumber: {

fontSize: 20,

fontWeight: 'bold',

color: '#4a90e2',

},

statLabel: {

color: '#666',

marginTop: 5,

},

section: {

margin: 15,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 10,

},

sectionItem: {

backgroundColor: '#fff',

padding: 15,

borderRadius: 5,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 1,

elevation: 1,

},

sectionItemText: {

fontSize: 16,

},

});

Після виконання цих кроків у вас буде базова структура додатку з навігацією та заглушками для основних екранів. Ви зможете запустити додаток і побачити функціонуючий прототип з мінімальним функціоналом.

Давайте знати, коли це буде готово, і ми перейдемо до наступного етапу - створення компонентів для додавання та редагування списків бажань та їх елементів.

Retry

IMG\_0E4628CD6C151.jpeg

V

ось що я бачу

Edit

Я бачу, що ви успішно запустили проект, але він все ще відображає стандартний текст "Hello World". Нам потрібно оновити вашу конфігурацію, щоб використовувати нашу структуру проекту та навігацію.

Здається, що код, який ми підготували, не було правильно інтегровано. Давайте це виправимо:

Перевірте, чи правильно оновлено App.js в корені проекту: Відкрийте файл App.js в корені проекту і переконайтеся, що він має такий вміст:

javascript

Copy

import App from './src/App';

export default App;

Переконайтеся, що створений файл src/App.js: Якщо він ще не створений, створіть файл src/App.js із кодом, який я надав раніше:

javascript

Copy

import React from 'react';

import { SafeAreaProvider } from 'react-native-safe-area-context';

import Navigation from './navigation';

import { StatusBar } from 'expo-status-bar';

export default function App() {

return (

<SafeAreaProvider>

<Navigation />

<StatusBar style="auto" />

</SafeAreaProvider>

);

}

Встановіть необхідні залежності для навігації:

bash

Copy

npx expo install @react-navigation/native @react-navigation/stack @react-navigation/bottom-tabs

npx expo install react-native-screens react-native-safe-area-context

Створіть файл src/navigation/index.js та інші необхідні файли для екранів, як описано в моєму попередньому повідомленні.

Перезапустіть ваш проект:

bash

Copy

npx expo start --clear

Будь ласка, переконайтеся, що ви виконали всі ці кроки, особливо встановлення необхідних пакетів навігації, оскільки без них додаток не зможе використовувати нашу навігаційну структуру.

Якщо ви ще не встановили пакети або бачите помилки при запуску, будь ласка, дайте мені знати, і я допоможу вам усунути проблеми.

Retry

V

не допомогло

Edit

Зрозуміло. Давайте спробуємо інший підхід, щоб виправити ситуацію. Здається, у нас проблема з інтеграцією нової структури.

Я пропоную почати спочатку, використовуючи таку послідовність дій:

Відкрийте файл App.js в корені проекту і змініть його вміст на простий компонент:

javascript

Copy

import React from 'react';

import { StyleSheet, Text, View } from 'react-native';

export default function App() {

return (

<View style={styles.container}>

<Text style={styles.title}>Welcome to Wishlist App!</Text>

<Text style={styles.subtitle}>Your personal wishlist manager</Text>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#fff',

alignItems: 'center',

justifyContent: 'center',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

},

subtitle: {

fontSize: 18,

color: '#666',

},

});

Перезапустіть додаток з очищенням кешу:

bash

Copy

npx expo start --clear

Якщо цей базовий приклад запрацює, ми зможемо поступово додавати нову структуру та навігацію.

Дайте мені знати, чи запрацював цей спрощений підхід, і ми продовжимо звідти.

Retry

V

ні нічого не змінилось

Edit

Зрозуміло, схоже, що ми маємо більш глибоку проблему з конфігурацією проекту. Давайте спробуємо повністю перезавантажити проект, щоб точно усунути проблему.

Ось що я пропоную:

Закрийте поточний сервер розробки (якщо він запущений) натиснувши Ctrl+C у терміналі.

Перевірте структуру проекту з терміналу:

bash

Copy

ls -la /Users/test/Documents/WishlistApp

Виконайте повне очищення кешу Expo та Node.js:

bash

Copy

# Очистіть кеш Expo

npx expo-cli clean

# або

rm -rf node\_modules/.cache/expo

# Очистіть кеш npm

npm cache clean --force

# Видаліть папку node\_modules та файли блокування

rm -rf node\_modules

rm -f package-lock.json

rm -f yarn.lock

# Перевстановіть залежності

npm install

Перевірте файл package.json щоб переконатися, що проект правильно налаштований:

bash

Copy

cat package.json

Головне, щоб в package.json було:

"main": "node\_modules/expo/AppEntry.js" або щось подібне, що вказує на точку входу

залежності для expo та react-native

Перезапустіть проект:

bash

Copy

npx expo start --clear

Якщо проблема залишається, можливо, найшвидшим рішенням буде створити новий проект з нуля:

bash

Copy

# Створити новий проект

cd /Users/test/Documents

npx create-expo-app WishlistAppNew

# Перейти в нову директорію

cd WishlistAppNew

# Запустити проект

npx expo start

Потім ми можемо систематично будувати нашу нову архітектуру в цьому чистому проекті.

Дайте мені знати результати цих кроків.

Retry

Знімок екрана 20250313 о 23.40.13.png

V

cat package.json

{

"name": "wishlistapp",

"main": "expo-router/entry",

"version": "1.0.0",

"scripts": {

"start": "expo start",

"reset-project": "node ./scripts/reset-project.js",

"android": "expo start --android",

"ios": "expo start --ios",

"web": "expo start --web",

"test": "jest --watchAll",

"lint": "expo lint"

},

"jest": {

"preset": "jest-expo"

},

"dependencies": {

"@expo/vector-icons": "^14.0.2",

"@react-native-async-storage/async-storage": "1.23.1",

"@react-navigation/bottom-tabs": "^7.2.1",

"@react-navigation/native": "^7.0.15",

"@react-navigation/stack": "^7.1.2",

"expo": "~52.0.38",

"expo-blur": "~14.0.3",

"expo-constants": "~17.0.8",

"expo-font": "~13.0.4",

"expo-haptics": "~14.0.1",

"expo-linking": "~7.0.5",

"expo-router": "~4.0.18",

"expo-splash-screen": "~0.29.22",

"expo-status-bar": "~2.0.1",

"expo-symbols": "~0.2.2",

"expo-system-ui": "~4.0.8",

"expo-web-browser": "~14.0.2",

"react": "18.3.1",

"react-dom": "18.3.1",

"react-hook-form": "^7.54.2",

"react-native": "0.76.7",

"react-native-gesture-handler": "~2.20.2",

"react-native-paper": "^5.13.1",

"react-native-reanimated": "~3.16.1",

"react-native-safe-area-context": "4.12.0",

"react-native-screens": "~4.4.0",

"react-native-vector-icons": "^10.2.0",

"react-native-web": "~0.19.13",

"react-native-webview": "13.12.5"

},

"devDependencies": {

"@babel/core": "^7.25.2",

"@types/jest": "^29.5.12",

"@types/react": "~18.3.12",

"@types/react-test-renderer": "^18.3.0",

"jest": "^29.2.1",

"jest-expo": "~52.0.6",

"react-test-renderer": "18.3.1",

"typescript": "^5.3.3"

},

"private": true

}

може ці дані допоможуть вирішити проблему

Learn more: https://docs.expo.dev/guides/new-architecture/

Web Bundled 4256ms node\_modules/expo-router/entry.js (678 modules)

LOG [web] Logs will appear in the browser console

Edit

Тепер ситуація стала зрозумілою! Я бачу, що ваш проект використовує expo-router як головну точку входу (в package.json "main": "expo-router/entry"). Це означає, що ваш проект створений з використанням Expo Router, що є більш сучасним підходом ніж React Navigation, який я пропонував раніше.

Expo Router використовує файлову систему для навігації, тому всі наші файли App.js та структура src не будуть працювати. Нам потрібно використовувати архітектуру, що базується на Expo Router.

Ось як можна реорганізувати проект:

Залиште папку app (а не видаляйте її), оскільки вона є основою для маршрутизації Expo Router.

Організуйте файлову структуру для Expo Router:

Copy

app/

(tabs)/

index.js # Головний екран (список вішлістів)

wishlist/

[id].js # Деталі вішліста (динамічний маршрут)

profile.js # Профіль користувача

settings.js # Налаштування

\_layout.js # Головний макет додатку

components/

common/

wishlist/

hooks/

services/

constants/

Створіть базові файли:

app/\_layout.js:

javascript

Copy

import { Stack } from 'expo-router';

import { StatusBar } from 'expo-status-bar';

export default function RootLayout() {

return (

<>

<StatusBar style="auto" />

<Stack />

</>

);

}

app/(tabs)/\_layout.js:

javascript

Copy

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function TabsLayout() {

return (

<Tabs screenOptions={{ headerShown: false }}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

app/(tabs)/index.js:

javascript

Copy

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link } from 'expo-router';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>My Wishlists</Text>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

app/(tabs)/profile.js:

javascript

Copy

import { View, Text, StyleSheet } from 'react-native';

export default function ProfileScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Profile</Text>

<View style={styles.profileContainer}>

<View style={styles.avatar}>

<Text style={styles.avatarText}>JD</Text>

</View>

<Text style={styles.name}>John Doe</Text>

<Text style={styles.email}>john.doe@example.com</Text>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

profileContainer: {

alignItems: 'center',

padding: 20,

},

avatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

marginBottom: 16,

},

avatarText: {

color: 'white',

fontSize: 36,

fontWeight: 'bold',

},

name: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 8,

},

email: {

fontSize: 16,

color: '#666',

},

});

app/(tabs)/settings.js:

javascript

Copy

import { View, Text, StyleSheet } from 'react-native';

export default function SettingsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Settings</Text>

<View style={styles.section}>

<Text style={styles.sectionTitle}>Account</Text>

<View style={styles.item}>

<Text>Edit Profile</Text>

</View>

<View style={styles.item}>

<Text>Notifications</Text>

</View>

<View style={styles.item}>

<Text>Privacy</Text>

</View>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

section: {

marginBottom: 20,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 10,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 8,

},

});

app/wishlist/[id].js:

javascript

Copy

import { View, Text, StyleSheet, FlatList } from 'react-native';

import { useLocalSearchParams } from 'expo-router';

// Тестові дані

const dummyWishlists = {

'1': {

name: 'Birthday Wishlist',

items: [

{ id: '1', name: 'iPhone 14', price: 999, priority: 'High' },

{ id: '2', name: 'MacBook Pro', price: 1999, priority: 'Medium' },

{ id: '3', name: 'AirPods Pro', price: 249, priority: 'Low' },

]

},

'2': {

name: 'Christmas Ideas',

items: [

{ id: '1', name: 'Nintendo Switch', price: 299, priority: 'Medium' },

{ id: '2', name: 'Kindle Paperwhite', price: 139, priority: 'Low' },

]

},

'3': {

name: 'Tech Gadgets',

items: [

{ id: '1', name: 'Drone', price: 799, priority: 'Medium' },

{ id: '2', name: 'GoPro', price: 349, priority: 'Low' },

{ id: '3', name: 'Smart Watch', price: 399, priority: 'High' },

]

}

};

export default function WishlistDetailScreen() {

const { id } = useLocalSearchParams();

const wishlist = dummyWishlists[id] || { name: 'Unknown Wishlist', items: [] };

return (

<View style={styles.container}>

<Text style={styles.title}>{wishlist.name}</Text>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<View style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemPrice}>${item.price}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

)}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemPrice: {

fontSize: 16,

color: '#666',

marginTop: 4,

},

itemPriority: {

fontSize: 14,

marginTop: 8,

fontWeight: '500',

},

});

Перезапустіть додаток:

bash

Copy

npx expo start --clear

Цей підхід використовує сучасний Expo Router, який був вже налаштований у вашому проекті. Він надає файлову систему маршрутизації, яка дуже інтуїтивна і проста у використанні.

Retry

V

давай переорганізуємо все що у нс є до цієї методики, надай чіткі інструкціх

Edit

Зрозуміло. Давайте переорганізуємо проект з використанням Expo Router. Ось чіткі інструкції:

Крок 1: Видалення непотрібних папок і файлів

bash

Copy

# Видаліть src папку якщо вона була створена

rm -rf src

# Зберігаємо папку app, яка потрібна для Expo Router

Крок 2: Створення структури проекту для Expo Router

bash

Copy

# Створіть необхідні папки

mkdir -p app/(tabs)

mkdir -p app/wishlist

mkdir -p components/common

mkdir -p components/wishlist

mkdir -p hooks

mkdir -p services

mkdir -p constants

Крок 3: Створення основних файлів

Спочатку створимо головний layout:

bash

Copy

# Створення файлу головного макету

cat > app/\_layout.js << 'EOF'

import { Stack } from 'expo-router';

import { StatusBar } from 'expo-status-bar';

export default function RootLayout() {

return (

<>

<StatusBar style="auto" />

<Stack />

</>

);

}

EOF

Тепер створимо layout для вкладок:

bash

Copy

# Створення файлу макету вкладок

cat > app/\(tabs\)/\_layout.js << 'EOF'

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function TabsLayout() {

return (

<Tabs screenOptions={{ headerShown: false }}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

Створимо головний екран зі списком бажань:

bash

Copy

cat > app/\(tabs\)/index.js << 'EOF'

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link } from 'expo-router';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>My Wishlists</Text>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

Створимо екран профілю:

bash

Copy

cat > app/\(tabs\)/profile.js << 'EOF'

import { View, Text, StyleSheet } from 'react-native';

export default function ProfileScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Profile</Text>

<View style={styles.profileContainer}>

<View style={styles.avatar}>

<Text style={styles.avatarText}>JD</Text>

</View>

<Text style={styles.name}>John Doe</Text>

<Text style={styles.email}>john.doe@example.com</Text>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

profileContainer: {

alignItems: 'center',

padding: 20,

},

avatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

marginBottom: 16,

},

avatarText: {

color: 'white',

fontSize: 36,

fontWeight: 'bold',

},

name: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 8,

},

email: {

fontSize: 16,

color: '#666',

},

});

EOF

Створимо екран налаштувань:

bash

Copy

cat > app/\(tabs\)/settings.js << 'EOF'

import { View, Text, StyleSheet } from 'react-native';

export default function SettingsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Settings</Text>

<View style={styles.section}>

<Text style={styles.sectionTitle}>Account</Text>

<View style={styles.item}>

<Text>Edit Profile</Text>

</View>

<View style={styles.item}>

<Text>Notifications</Text>

</View>

<View style={styles.item}>

<Text>Privacy</Text>

</View>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

section: {

marginBottom: 20,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 10,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 8,

},

});

EOF

Створимо екран деталей списку бажань:

bash

Copy

cat > app/wishlist/\[id\].js << 'EOF'

import { View, Text, StyleSheet, FlatList } from 'react-native';

import { useLocalSearchParams } from 'expo-router';

// Тестові дані

const dummyWishlists = {

'1': {

name: 'Birthday Wishlist',

items: [

{ id: '1', name: 'iPhone 14', price: 999, priority: 'High' },

{ id: '2', name: 'MacBook Pro', price: 1999, priority: 'Medium' },

{ id: '3', name: 'AirPods Pro', price: 249, priority: 'Low' },

]

},

'2': {

name: 'Christmas Ideas',

items: [

{ id: '1', name: 'Nintendo Switch', price: 299, priority: 'Medium' },

{ id: '2', name: 'Kindle Paperwhite', price: 139, priority: 'Low' },

]

},

'3': {

name: 'Tech Gadgets',

items: [

{ id: '1', name: 'Drone', price: 799, priority: 'Medium' },

{ id: '2', name: 'GoPro', price: 349, priority: 'Low' },

{ id: '3', name: 'Smart Watch', price: 399, priority: 'High' },

]

}

};

export default function WishlistDetailScreen() {

const { id } = useLocalSearchParams();

const wishlist = dummyWishlists[id] || { name: 'Unknown Wishlist', items: [] };

return (

<View style={styles.container}>

<Text style={styles.title}>{wishlist.name}</Text>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<View style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemPrice}>${item.price}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

)}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemPrice: {

fontSize: 16,

color: '#666',

marginTop: 4,

},

itemPriority: {

fontSize: 14,

marginTop: 8,

fontWeight: '500',

},

});

EOF

Додамо файл з константами:

bash

Copy

cat > constants/theme.js << 'EOF'

export default {

colors: {

primary: '#4a90e2',

background: '#f8f8f8',

card: '#ffffff',

text: '#333333',

border: '#dddddd',

notification: '#ff3b30',

error: '#e74c3c',

warning: '#f39c12',

success: '#27ae60',

},

spacing: {

xs: 4,

sm: 8,

md: 16,

lg: 24,

xl: 32,

},

fontSizes: {

xs: 12,

sm: 14,

md: 16,

lg: 18,

xl: 20,

xxl: 24,

},

};

EOF

Крок 4: Запустіть додаток

bash

Copy

# Очистіть кеш і запустіть додаток

npx expo start --clear

Після виконання цих кроків ви повинні мати функціонуючий додаток з наступними функціями:

Головний екран зі списком бажань

Екран деталей списку бажань (при натисканні на елемент списку)

Вкладки для перемикання між списками бажань, профілем та налаштуваннями

Це базова структура, яку ми можемо далі розширювати, додаючи нові функції, такі як:

Додавання нових списків бажань

Додавання/редагування/видалення елементів у списках

Інтеграція з бекендом для збереження даних

Авторизація користувача

Відстеження цін

Поширення списків тощо

Retry

V

# Створення файлу макету вкладок

cat > app/\(tabs\)/\_layout.js << 'EOF'

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function TabsLayout() {

return (

<Tabs screenOptions={{ headerShown: false }}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

zsh: no such file or directory: app/(tabs)/\_layout.js

Edit

У вас виникає помилка через спеціальні символи в шляху. В терміналі дужки (tabs) мають спеціальне значення, тому вам потрібно екранувати їх або використати інший підхід.

Давайте спробуємо створити ці файли інакше:

bash

Copy

# Створимо директорію (tabs)

mkdir -p "app/(tabs)"

# Тепер створимо файли безпосередньо

touch "app/(tabs)/\_layout.js"

touch "app/(tabs)/index.js"

touch "app/(tabs)/profile.js"

touch "app/(tabs)/settings.js"

Тепер ви можете відкрити ці файли у текстовому редакторі та скопіювати-вставити відповідний код з мого попереднього повідомлення.

Альтернативно, ви можете використовувати редактор, наприклад VS Code, щоб створити ці файли та папки візуально, а потім вставити код.

Retry

V

Що дальше, чим їх і як через термінал наповнити?

Edit

Для наповнення файлів через термінал можна використати команду nano, яка відкриє простий текстовий редактор. Ось як це зробити:

bash

Copy

# Для файлу \_layout.js в папці (tabs)

nano "app/(tabs)/\_layout.js"

Коли відкриється редактор, вставте код, який я надав раніше. Після завершення редагування натисніть Ctrl+O, щоб зберегти файл, і Ctrl+X, щоб вийти з редактора.

Повторіть цей процес для кожного файлу:

bash

Copy

# Інші файли

nano "app/(tabs)/index.js"

nano "app/(tabs)/profile.js"

nano "app/(tabs)/settings.js"

nano "app/wishlist/[id].js"

nano "app/\_layout.js"

Для [id].js спочатку створіть папку:

bash

Copy

mkdir -p "app/wishlist"

touch "app/wishlist/[id].js"

nano "app/wishlist/[id].js"

Для папки constants:

bash

Copy

mkdir -p constants

nano constants/theme.js

Після наповнення всіх файлів запустіть додаток:

bash

Copy

npx expo start --clear

Якщо вам незручно використовувати nano, ви можете використати інший текстовий редактор чи IDE для редагування файлів.

Retry

V

Для [id].js спочатку створіть папку давай повернемось до цього

Edit

Так, давайте створимо папку для динамічного маршруту з [id].js:

bash

Copy

# Створюємо папку wishlist

mkdir -p "app/wishlist"

# Створюємо файл [id].js в цій папці

touch "app/wishlist/[id].js"

# Тепер редагуємо файл

nano "app/wishlist/[id].js"

Коли відкриється редактор nano, вставте наступний код:

javascript

Copy

import { View, Text, StyleSheet, FlatList } from 'react-native';

import { useLocalSearchParams } from 'expo-router';

// Тестові дані

const dummyWishlists = {

'1': {

name: 'Birthday Wishlist',

items: [

{ id: '1', name: 'iPhone 14', price: 999, priority: 'High' },

{ id: '2', name: 'MacBook Pro', price: 1999, priority: 'Medium' },

{ id: '3', name: 'AirPods Pro', price: 249, priority: 'Low' },

]

},

'2': {

name: 'Christmas Ideas',

items: [

{ id: '1', name: 'Nintendo Switch', price: 299, priority: 'Medium' },

{ id: '2', name: 'Kindle Paperwhite', price: 139, priority: 'Low' },

]

},

'3': {

name: 'Tech Gadgets',

items: [

{ id: '1', name: 'Drone', price: 799, priority: 'Medium' },

{ id: '2', name: 'GoPro', price: 349, priority: 'Low' },

{ id: '3', name: 'Smart Watch', price: 399, priority: 'High' },

]

}

};

export default function WishlistDetailScreen() {

const { id } = useLocalSearchParams();

const wishlist = dummyWishlists[id] || { name: 'Unknown Wishlist', items: [] };

return (

<View style={styles.container}>

<Text style={styles.title}>{wishlist.name}</Text>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<View style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemPrice}>${item.price}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

)}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemPrice: {

fontSize: 16,

color: '#666',

marginTop: 4,

},

itemPriority: {

fontSize: 14,

marginTop: 8,

fontWeight: '500',

},

});

Після того, як наповните файл, натисніть Ctrl+O для збереження та Ctrl+X для виходу з редактора.

Цей файл буде відповідати за відображення деталей конкретного списку бажань, коли користувач натисне на нього у головному списку. Параметр id в URL буде використовуватися для вибору відповідних даних.

Retry

V

а можемо створити оразу із вмістом цей файл через термінал?

Edit

Так, звичайно! Ви можете створити файл з вмістом одразу через термінал. Ось як це зробити:

bash

Copy

# Створюємо папку wishlist

mkdir -p "app/wishlist"

# Створюємо файл [id].js з вмістом

cat > "app/wishlist/[id].js" << 'EOF'

import { View, Text, StyleSheet, FlatList } from 'react-native';

import { useLocalSearchParams } from 'expo-router';

// Тестові дані

const dummyWishlists = {

'1': {

name: 'Birthday Wishlist',

items: [

{ id: '1', name: 'iPhone 14', price: 999, priority: 'High' },

{ id: '2', name: 'MacBook Pro', price: 1999, priority: 'Medium' },

{ id: '3', name: 'AirPods Pro', price: 249, priority: 'Low' },

]

},

'2': {

name: 'Christmas Ideas',

items: [

{ id: '1', name: 'Nintendo Switch', price: 299, priority: 'Medium' },

{ id: '2', name: 'Kindle Paperwhite', price: 139, priority: 'Low' },

]

},

'3': {

name: 'Tech Gadgets',

items: [

{ id: '1', name: 'Drone', price: 799, priority: 'Medium' },

{ id: '2', name: 'GoPro', price: 349, priority: 'Low' },

{ id: '3', name: 'Smart Watch', price: 399, priority: 'High' },

]

}

};

export default function WishlistDetailScreen() {

const { id } = useLocalSearchParams();

const wishlist = dummyWishlists[id] || { name: 'Unknown Wishlist', items: [] };

return (

<View style={styles.container}>

<Text style={styles.title}>{wishlist.name}</Text>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<View style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemPrice}>${item.price}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

)}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemPrice: {

fontSize: 16,

color: '#666',

marginTop: 4,

},

itemPriority: {

fontSize: 14,

marginTop: 8,

fontWeight: '500',

},

});

EOF

Ця команда створить файл [id].js у папці app/wishlist і відразу заповнить його вказаним кодом. Частина << 'EOF' та EOF в кінці — це "here document" синтаксис, який дозволяє вводити багаторядковий текст безпосередньо у команду.

Таким же способом ви можете створити й інші файли з їхнім вмістом:

bash

Copy

# Для \_layout.js

cat > "app/\_layout.js" << 'EOF'

import { Stack } from 'expo-router';

import { StatusBar } from 'expo-status-bar';

export default function RootLayout() {

return (

<>

<StatusBar style="auto" />

<Stack />

</>

);

}

EOF

# Для (tabs)/\_layout.js

cat > "app/(tabs)/\_layout.js" << 'EOF'

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function TabsLayout() {

return (

<Tabs screenOptions={{ headerShown: false }}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

Це дуже зручний спосіб для створення файлів з необхідним вмістом через термінал.

Retry

V

Що дальше?

Edit

Тепер, коли ми створили базову структуру для файлу [id].js, давайте продовжимо створення інших файлів таким же способом:

bash

Copy

# Створюємо файл (tabs)/index.js (головний екран зі списком бажань)

cat > "app/(tabs)/index.js" << 'EOF'

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link } from 'expo-router';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>My Wishlists</Text>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

# Створюємо файл (tabs)/profile.js

cat > "app/(tabs)/profile.js" << 'EOF'

import { View, Text, StyleSheet } from 'react-native';

export default function ProfileScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Profile</Text>

<View style={styles.profileContainer}>

<View style={styles.avatar}>

<Text style={styles.avatarText}>JD</Text>

</View>

<Text style={styles.name}>John Doe</Text>

<Text style={styles.email}>john.doe@example.com</Text>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

profileContainer: {

alignItems: 'center',

padding: 20,

},

avatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

marginBottom: 16,

},

avatarText: {

color: 'white',

fontSize: 36,

fontWeight: 'bold',

},

name: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 8,

},

email: {

fontSize: 16,

color: '#666',

},

});

EOF

# Створюємо файл (tabs)/settings.js

cat > "app/(tabs)/settings.js" << 'EOF'

import { View, Text, StyleSheet } from 'react-native';

export default function SettingsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Settings</Text>

<View style={styles.section}>

<Text style={styles.sectionTitle}>Account</Text>

<View style={styles.item}>

<Text>Edit Profile</Text>

</View>

<View style={styles.item}>

<Text>Notifications</Text>

</View>

<View style={styles.item}>

<Text>Privacy</Text>

</View>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

section: {

marginBottom: 20,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 10,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 8,

},

});

EOF

Після створення всіх необхідних файлів, давайте запустимо додаток:

bash

Copy

# Запуск додатку з очищенням кешу

npx expo start --clear

Тепер ви повинні побачити ваш додаток з навігацією по вкладках (Wishlists, Profile, Settings) та можливістю переходити до деталей списку бажань.

Після успішного запуску ми можемо продовжити розширювати функціональність додатку:

Додати можливість створення нових списків бажань

Реалізувати додавання елементів до списків

Додати функцію редагування та видалення

Впровадити зберігання даних локально за допомогою AsyncStorage

Підключити справжній бекенд для синхронізації даних

З чого б ви хотіли продовжити розробку?

Retry

V

Logs for your project will appear below. Press Ctrl+C to exit.

Error: ENOENT: no such file or directory, scandir '/Users/test/Documents/WishlistApp/src/assets'

at Object.readdir (node:internal/fs/promises:948:18)

at getAbsoluteAssetRecord (/Users/test/Documents/WishlistApp/node\_modules/metro/src/Assets.js:77:17)

at getAsset (/Users/test/Documents/WishlistApp/node\_modules/metro/src/Assets.js:191:18)

at Server.\_processSingleAssetRequest (/Users/test/Documents/WishlistApp/node\_modules/metro/src/Server.js:335:20)

at Server.\_processRequest (/Users/test/Documents/WishlistApp/node\_modules/metro/src/Server.js:417:7)

Unable to resolve asset "./src/assets/icon.png" from "icon" in your app.json or app.config.js

iOS Bundled 11178ms node\_modules/expo-router/entry.js (1112 modules)

(NOBRIDGE) LOG Bridgeless mode is enabled

INFO

💡 JavaScript logs will be removed from Metro in React Native 0.77! Please use React Native DevTools as your default tool. Tip: Type j in the terminal to open (requires Google Chrome or Microsoft Edge).

(NOBRIDGE) WARN Linking requires a build-time setting scheme in the project's Expo config (app.config.js or app.json) for production apps, if it's left blank, your app may crash. The scheme does not apply to development in the Expo client but you should add it as soon as you start working with Linking to avoid creating a broken build. Learn more: https://docs.expo.dev/guides/linking/

(NOBRIDGE) WARN 🚨 React Native's New Architecture is always enabled in Expo Go, but it is not explicitly enabled your project app config. This may lead to unexpected behavior when you create a production or development build. Set "newArchEnabled": true in your app.json.

Learn more: https://docs.expo.dev/guides/new-architecture/

Edit

Бачу помилку, пов'язану з асетами. Додаток шукає файл icon.png у папці src/assets, але не може його знайти. Потрібно виправити шляхи до зображень у файлі app.json.

Давайте вирішимо цю проблему:

Створимо необхідну папку assets:

bash

Copy

mkdir -p assets

Відредагуймо app.json, щоб виправити шляхи до зображень:

bash

Copy

cat > app.json << 'EOF'

{

"expo": {

"name": "WishlistApp",

"slug": "wishlistapp",

"version": "1.0.0",

"orientation": "portrait",

"icon": "./assets/icon.png",

"userInterfaceStyle": "light",

"splash": {

"image": "./assets/splash.png",

"resizeMode": "contain",

"backgroundColor": "#ffffff"

},

"assetBundlePatterns": [

"\*\*/\*"

],

"ios": {

"supportsTablet": true

},

"android": {

"adaptiveIcon": {

"foregroundImage": "./assets/adaptive-icon.png",

"backgroundColor": "#ffffff"

}

},

"web": {

"favicon": "./assets/favicon.png"

},

"scheme": "wishlistapp",

"newArchEnabled": true

}

}

EOF

Створимо заглушки для зображень:

bash

Copy

# Створюємо порожні файли зображень, щоб уникнути помилок

touch assets/icon.png

touch assets/splash.png

touch assets/adaptive-icon.png

touch assets/favicon.png

Після цих змін, спробуйте перезапустити додаток:

bash

Copy

npx expo start --clear

Це має виправити помилки, пов'язані з відсутніми асетами, і дозволити додатку запуститися. Також я додав схему "wishlistapp" та увімкнув нову архітектуру, як рекомендує Expo у своїх попередженнях.

Retry

Знімок екрана 20250313 о 23.56.52.png

V

ось що бачу

Edit

Здається, ми маємо проблему з завантаженням наших компонентів. Виглядає так, що додаток все ще показує стандартний контент замість нашого кастомного інтерфейсу.

Давайте перевіримо, чи існують файли та чи правильні вони. Проблема може бути в тому, що якийсь із наших файлів не коректно імпортується або не має правильного експорту.

Спробуємо кілька кроків для виправлення:

Давайте перевіримо структуру проекту:

bash

Copy

ls -la app/

ls -la "app/(tabs)"

Спробуймо оновити файл app/(tabs)/index.js, щоб переконатися, що він правильно експортується:

bash

Copy

cat > "app/(tabs)/index.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link } from 'expo-router';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function HomeScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>My Wishlists</Text>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

Перевірмо файл app/\_layout.js, щоб переконатися що він правильно налаштований:

bash

Copy

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Stack } from 'expo-router';

import { StatusBar } from 'expo-status-bar';

export default function RootLayout() {

return (

<>

<StatusBar style="auto" />

<Stack screenOptions={{ headerShown: false }} />

</>

);

}

EOF

Перезапустимо додаток повністю з очищенням кешу:

bash

Copy

# Зупинити поточний процес (Ctrl+C)

# Очистити кеш

rm -rf node\_modules/.cache

# Перезапустити додаток

npx expo start --clear

Якщо після цих кроків інтерфейс все ще не відображається правильно, спробуймо альтернативний підхід:

Перевірте, чи існує файл app/index.js (він може конфліктувати з нашими маршрутами):

bash

Copy

ls -la app/index.js

# Якщо він існує, спробуйте перейменувати або видалити його

Повідомте результати цих кроків, і ми вирішимо, як рухатись далі.

Retry

V

може це допоможе Logs for your project will appear below. Press Ctrl+C to exit.

Web Bundled 6606ms node\_modules/expo-router/entry.js (787 modules)

LOG [web] Logs will appear in the browser console

Error: Could not find MIME for Buffer <null>

at Jimp.parseBitmap (/Users/test/Documents/WishlistApp/node\_modules/jimp-compact/dist/jimp.js:1:125518)

at Jimp.parseBitmap (/Users/test/Documents/WishlistApp/node\_modules/jimp-compact/dist/jimp.js:1:8514)

at /Users/test/Documents/WishlistApp/node\_modules/jimp-compact/dist/jimp.js:1:7613

at FSReqCallback.readFileAfterClose [as oncomplete] (node:internal/fs/read/context:68:3)

Edit

Помилка вказує на проблему при обробці зображень. Схоже, Expo намагається обробити порожні файли зображень, які ми створили, але не може визначити їх MIME-тип.

Давайте виправимо цю проблему, замінивши порожні файли зображень на дійсні:

Завантажмо базові зображення для нашого додатку:

bash

Copy

# Створюємо копію стандартних зображень з шаблону Expo

cp node\_modules/expo/template-files/assets/\* assets/

Якщо ця команда не працює, можна завантажити файли з репозиторію Expo або створити прості зображення.

Альтернативно, ми можемо модифікувати app.json, щоб тимчасово вимкнути використання зображень:

bash

Copy

cat > app.json << 'EOF'

{

"expo": {

"name": "WishlistApp",

"slug": "wishlistapp",

"version": "1.0.0",

"orientation": "portrait",

"userInterfaceStyle": "light",

"assetBundlePatterns": [

"\*\*/\*"

],

"ios": {

"supportsTablet": true

},

"android": {

"adaptiveIcon": {

"backgroundColor": "#ffffff"

}

},

"web": {},

"scheme": "wishlistapp",

"newArchEnabled": true

}

}

EOF

Якщо наша проблема пов'язана з маршрутизацією, спробуймо спростити структуру. Створимо базовий index.js у папці app:

bash

Copy

cat > "app/index.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet } from 'react-native';

export default function Home() {

return (

<View style={styles.container}>

<Text style={styles.title}>Wishlist App</Text>

<Text style={styles.subtitle}>Your personal wishlist manager</Text>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#fff',

alignItems: 'center',

justifyContent: 'center',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

},

subtitle: {

fontSize: 18,

color: '#666',

},

});

EOF

Перезапустімо додаток ще раз:

bash

Copy

npx expo start --clear

Якщо проблема залишається, можливо, нам доведеться переглянути наш підхід. Для цього є кілька варіантів:

Спробувати оновити Expo до останньої версії

Створити новий проект і поступово перенести код

Перевірити, чи всі необхідні залежності встановлено правильно

Давайте спробуємо ці виправлення і подивимось, чи зможемо вирішити проблему.

Retry

V

Logs for your project will appear below. Press Ctrl+C to exit.

iOS Bundled 10296ms node\_modules/expo-router/entry.js (1113 modules)

(NOBRIDGE) LOG Bridgeless mode is enabled

INFO

💡 JavaScript logs will be removed from Metro in React Native 0.77! Please use React Native DevTools as your default tool. Tip: Type j in the terminal to open (requires Google Chrome or Microsoft Edge).

(NOBRIDGE) ERROR Warning: Error: The route files "./index.tsx" and "./index.js" conflict on the route "/index". Please remove or rename one of these files.

This error is located at:

in ContextNavigator (created by ExpoRoot)

in ExpoRoot (created by App)

in App (created by ErrorOverlay)

in ErrorToastContainer (created by ErrorOverlay)

in ErrorOverlay (created by withDevTools(ErrorOverlay))

in withDevTools(ErrorOverlay)

in RCTView (created by View)

in View (created by AppContainer)

in RCTView (created by View)

in View (created by AppContainer)

in AppContainer

in main(RootComponent)

(NOBRIDGE) ERROR Warning: Error: The route files "./index.tsx" and "./index.js" conflict on the route "/index". Please remove or rename one of these files.

This error is located at:

in ContextNavigator (created by ExpoRoot)

in ExpoRoot (created by App)

in App (created by ErrorOverlay)

in ErrorToastContainer (created by ErrorOverlay)

in ErrorOverlay (created by withDevTools(ErrorOverlay))

in withDevTools(ErrorOverlay)

in RCTView (created by View)

in View (created by AppContainer)

in RCTView (created by View)

in View (created by AppContainer)

in AppContainer

in main(RootComponent)

Edit

Тепер ми бачимо конкретну помилку! У вас є конфлікт між двома файлами: ./index.tsx та ./index.js, які конфліктують на маршруті "/index".

Це просто виправити:

bash

Copy

# Перевіримо, які файли index у нас є

ls -la app/index.\*

# Видалимо файл index.tsx (якщо він існує)

rm -f app/index.tsx

Якщо у вас є обидва файли (index.tsx та index.js), вам потрібно залишити тільки один з них. Оскільки ми створили index.js, краще видалити index.tsx.

Після видалення конфліктуючого файлу перезапустіть додаток:

bash

Copy

npx expo start --clear

Якщо проблема залишається, можливо також є конфлікт між app/(tabs)/index.js та app/index.js. В такому випадку, нам потрібно залишити тільки один із них:

bash

Copy

# Якщо у вас є обидва файли, видаліть один

rm -f app/index.js

# або

rm -f "app/(tabs)/index.js"

Спробуйте просто залишити структуру з app/(tabs)/index.js та видалити app/index.js, якщо він є, і подивіться, чи запрацює навігація.

Retry

Знімок екрана 20250314 о 00.00.57.png

V

ось що тепер бачу

Edit

Чудово! Тепер ми маємо прогрес - ми бачимо стартовий екран з назвою "Wishlist App". Це означає, що початковий файл index.js завантажується правильно.

Тепер нам потрібно налаштувати навігацію вкладок. Здається, наш файл \_layout.js для вкладок не застосовується. Давайте перевіримо структуру та вміст файлів:

Оновимо наш app/\_layout.js для забезпечення коректної навігації:

bash

Copy

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Slot } from 'expo-router';

import { StatusBar } from 'expo-status-bar';

export default function RootLayout() {

return (

<>

<StatusBar style="auto" />

<Slot />

</>

);

}

EOF

Перевірмо, що файл (tabs)/\_layout.js існує і має правильний вміст:

bash

Copy

ls -la "app/(tabs)/\_layout.js"

cat "app/(tabs)/\_layout.js"

Додамо нашу структуру вкладок. Створімо файл безпосередньо за допомогою текстового редактора або через термінал:

bash

Copy

cat > "app/(tabs)/\_layout.js" << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function TabsLayout() {

return (

<Tabs screenOptions={{ headerShown: false }}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

Оновимо наш файл index.js у папці (tabs):

bash

Copy

cat > "app/(tabs)/index.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link } from 'expo-router';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>My Wishlists</Text>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

Переконаємося, що інші файли вкладок також існують:

bash

Copy

cat > "app/(tabs)/profile.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet } from 'react-native';

export default function ProfileScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Profile</Text>

<View style={styles.profileContainer}>

<View style={styles.avatar}>

<Text style={styles.avatarText}>JD</Text>

</View>

<Text style={styles.name}>John Doe</Text>

<Text style={styles.email}>john.doe@example.com</Text>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

profileContainer: {

alignItems: 'center',

padding: 20,

},

avatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

marginBottom: 16,

},

avatarText: {

color: 'white',

fontSize: 36,

fontWeight: 'bold',

},

name: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 8,

},

email: {

fontSize: 16,

color: '#666',

},

});

EOF

bash

Copy

cat > "app/(tabs)/settings.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet } from 'react-native';

export default function SettingsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Settings</Text>

<View style={styles.section}>

<Text style={styles.sectionTitle}>Account</Text>

<View style={styles.item}>

<Text>Edit Profile</Text>

</View>

<View style={styles.item}>

<Text>Notifications</Text>

</View>

<View style={styles.item}>

<Text>Privacy</Text>

</View>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

section: {

marginBottom: 20,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 10,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 8,

},

});

EOF

Перезапустіть додаток:

bash

Copy

npx expo start --clear

Після цих змін ви повинні побачити навігацію по вкладках і можливість переключатися між різними екранами. Повідомте, що ви бачите після цих змін.

Retry

IMG\_DFA77B0A53481.jpeg

V

Edit

Бачу, що ми все ще маємо проблему - додаток відображає лише заголовок і підзаголовок, але без вкладок і нашого основного вмісту.

Здається, є конфлікт між файлами маршрутизації. Ймовірно, файл app/index.js все ще має пріоритет над нашою структурою вкладок.

Давайте спробуємо інший підхід з більш послідовним перебудуванням структури:

Давайте видалимо всі файли в директорії app і почнемо з чистого аркуша:

bash

Copy

# Зберігаємо резервну копію наших файлів

mkdir -p backup/app

cp -r app/\* backup/app/

# Видаляємо всі файли в app

rm -rf app/\*

# Створюємо новий файл index.js в app

cat > "app/index.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link } from 'expo-router';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>My Wishlists</Text>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

# Створюємо файл \_layout.js в app

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function AppLayout() {

return (

<Tabs screenOptions={{ headerShown: false }}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

# Створюємо файл profile.js в app

cat > "app/profile.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet } from 'react-native';

export default function ProfileScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Profile</Text>

<View style={styles.profileContainer}>

<View style={styles.avatar}>

<Text style={styles.avatarText}>JD</Text>

</View>

<Text style={styles.name}>John Doe</Text>

<Text style={styles.email}>john.doe@example.com</Text>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

profileContainer: {

alignItems: 'center',

padding: 20,

},

avatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

marginBottom: 16,

},

avatarText: {

color: 'white',

fontSize: 36,

fontWeight: 'bold',

},

name: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 8,

},

email: {

fontSize: 16,

color: '#666',

},

});

EOF

# Створюємо файл settings.js в app

cat > "app/settings.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet } from 'react-native';

export default function SettingsScreen() {

return (

<View style={styles.container}>

<Text style={styles.title}>Settings</Text>

<View style={styles.section}>

<Text style={styles.sectionTitle}>Account</Text>

<View style={styles.item}>

<Text>Edit Profile</Text>

</View>

<View style={styles.item}>

<Text>Notifications</Text>

</View>

<View style={styles.item}>

<Text>Privacy</Text>

</View>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

section: {

marginBottom: 20,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 10,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 8,

},

});

EOF

# Створюємо директорію wishlist та файл [id].js

mkdir -p app/wishlist

cat > "app/wishlist/[id].js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList } from 'react-native';

import { useLocalSearchParams } from 'expo-router';

// Тестові дані

const dummyWishlists = {

'1': {

name: 'Birthday Wishlist',

items: [

{ id: '1', name: 'iPhone 14', price: 999, priority: 'High' },

{ id: '2', name: 'MacBook Pro', price: 1999, priority: 'Medium' },

{ id: '3', name: 'AirPods Pro', price: 249, priority: 'Low' },

]

},

'2': {

name: 'Christmas Ideas',

items: [

{ id: '1', name: 'Nintendo Switch', price: 299, priority: 'Medium' },

{ id: '2', name: 'Kindle Paperwhite', price: 139, priority: 'Low' },

]

},

'3': {

name: 'Tech Gadgets',

items: [

{ id: '1', name: 'Drone', price: 799, priority: 'Medium' },

{ id: '2', name: 'GoPro', price: 349, priority: 'Low' },

{ id: '3', name: 'Smart Watch', price: 399, priority: 'High' },

]

}

};

export default function WishlistDetailScreen() {

const { id } = useLocalSearchParams();

const wishlist = dummyWishlists[id] || { name: 'Unknown Wishlist', items: [] };

return (

<View style={styles.container}>

<Text style={styles.title}>{wishlist.name}</Text>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<View style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemPrice}>${item.price}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

)}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemPrice: {

fontSize: 16,

color: '#666',

marginTop: 4,

},

itemPriority: {

fontSize: 14,

marginTop: 8,

fontWeight: '500',

},

});

EOF

Після перебудови структури, перезапустіть додаток:

bash

Copy

npx expo start --clear

Ця нова структура повинна бути простішою і більш зрозумілою для Expo Router. Тепер ми маємо:

Основні екрани безпосередньо в папці app

Головний layout з вкладками в \_layout.js

Екран деталей вішліста в підпапці wishlist

Це більш відповідає рекомендованій структурі для Expo Router v2.

Retry

IMG\_EF7F79E25BB11.jpeg

V

Edit

Чудово! Ми досягли успіху - додаток тепер показує інтерфейс зі списком бажань та вкладками внизу екрану. Це відображає структуру, яку ми запланували:

Головний екран зі списком вішлістів

Навігація по вкладках (Wishlists, Profile, Settings)

Можливість переходити до деталей кожного вішліста

Тепер ми можемо продовжити розвивати додаток, додаючи функціональність:

Наступним кроком можна реалізувати додавання нових списків бажань:

Створити новий екран для форми додавання

Реалізувати збереження нових списків

Потім можна зосередитись на функціоналі списків:

Додавання нових елементів до списку

Редагування існуючих елементів

Видалення елементів або списків

Пізніше можна реалізувати:

Зберігання даних локально (AsyncStorage)

Інтеграція з бекендом (якщо потрібно)

Додаткові функції, такі як пошук, фільтрація, сортування

Чи бажаєте ви продовжити розробку додатку з будь-якого конкретного аспекту?

Retry

V

дивись, нам потрібно, щоб користувач між знахрдити своїх друзів за імям, і міг додати їх до списку друзів, і відображати цей список друзів у профілі користувача, і ці друзі можуть бачити вішлисти один одного

Edit

Чудово! Це важлива соціальна функція для нашого додатку Wishlist. Давайте реалізуємо функціонал пошуку друзів, додавання їх до списку друзів та відображення списків друзів у профілі.

Ось план реалізації:

Спочатку створимо екран пошуку друзів:

bash

Copy

mkdir -p app/friends

cat > "app/friends/search.js" << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TextInput, FlatList, TouchableOpacity, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані користувачів

const dummyUsers = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'John Doe', username: 'john\_doe', avatar: null },

{ id: '3', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '4', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

{ id: '5', name: 'Robert Brown', username: 'rob\_brown', avatar: null },

];

export default function SearchFriendsScreen() {

const [searchQuery, setSearchQuery] = useState('');

const [results, setResults] = useState([]);

const handleSearch = (text) => {

setSearchQuery(text);

// Фільтруємо користувачів на основі пошукового запиту

if (text.trim() === '') {

setResults([]);

} else {

const filteredUsers = dummyUsers.filter(

user =>

user.name.toLowerCase().includes(text.toLowerCase()) ||

user.username.toLowerCase().includes(text.toLowerCase())

);

setResults(filteredUsers);

}

};

const renderUserItem = ({ item }) => (

<TouchableOpacity style={styles.userItem}>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<View style={styles.userInfo}>

<Text style={styles.userName}>{item.name}</Text>

<Text style={styles.userUsername}>@{item.username}</Text>

</View>

<TouchableOpacity style={styles.addButton}>

<FontAwesome name="user-plus" size={20} color="#4a90e2" />

</TouchableOpacity>

</TouchableOpacity>

);

return (

<View style={styles.container}>

<Text style={styles.title}>Find Friends</Text>

<View style={styles.searchContainer}>

<FontAwesome name="search" size={20} color="#999" style={styles.searchIcon} />

<TextInput

style={styles.searchInput}

placeholder="Search by name or username"

value={searchQuery}

onChangeText={handleSearch}

autoCapitalize="none"

/>

</View>

{searchQuery.trim() !== '' && (

<FlatList

data={results}

keyExtractor={item => item.id}

renderItem={renderUserItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>No users found</Text>

}

/>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

marginTop: 60,

},

searchContainer: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

borderRadius: 8,

padding: 10,

marginBottom: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

searchIcon: {

marginRight: 10,

},

searchInput: {

flex: 1,

fontSize: 16,

},

userItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 12,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

avatarContainer: {

marginRight: 15,

},

avatar: {

width: 50,

height: 50,

borderRadius: 25,

},

avatarPlaceholder: {

width: 50,

height: 50,

borderRadius: 25,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

color: 'white',

fontSize: 24,

fontWeight: 'bold',

},

userInfo: {

flex: 1,

},

userName: {

fontSize: 16,

fontWeight: 'bold',

},

userUsername: {

fontSize: 14,

color: '#777',

},

addButton: {

padding: 10,

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Оновимо екран профілю, щоб включити список друзів:

bash

Copy

cat > "app/profile.js" << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { Link } from 'expo-router';

// Тестові дані для друзів

const dummyFriends = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '3', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

];

export default function ProfileScreen() {

const [activeTab, setActiveTab] = useState('info');

const renderFriendItem = ({ item }) => (

<TouchableOpacity style={styles.friendItem}>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<Text style={styles.friendName}>{item.name}</Text>

</TouchableOpacity>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<View style={styles.profileAvatarContainer}>

<View style={styles.profileAvatar}>

<Text style={styles.profileAvatarText}>JD</Text>

</View>

</View>

<Text style={styles.profileName}>John Doe</Text>

<Text style={styles.profileUsername}>@john\_doe</Text>

<View style={styles.statsContainer}>

<View style={styles.statItem}>

<Text style={styles.statNumber}>5</Text>

<Text style={styles.statLabel}>Wishlists</Text>

</View>

<View style={styles.statItem}>

<Text style={styles.statNumber}>{dummyFriends.length}</Text>

<Text style={styles.statLabel}>Friends</Text>

</View>

</View>

</View>

<View style={styles.tabContainer}>

<TouchableOpacity

style={[styles.tab, activeTab === 'info' && styles.activeTab]}

onPress={() => setActiveTab('info')}

>

<Text style={[styles.tabText, activeTab === 'info' && styles.activeTabText]}>Info</Text>

</TouchableOpacity>

<TouchableOpacity

style={[styles.tab, activeTab === 'friends' && styles.activeTab]}

onPress={() => setActiveTab('friends')}

>

<Text style={[styles.tabText, activeTab === 'friends' && styles.activeTabText]}>Friends</Text>

</TouchableOpacity>

</View>

{activeTab === 'info' ? (

<View style={styles.infoContainer}>

<View style={styles.infoItem}>

<FontAwesome name="envelope" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>john.doe@example.com</Text>

</View>

<View style={styles.infoItem}>

<FontAwesome name="calendar" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>Member since January 2023</Text>

</View>

</View>

) : (

<View style={styles.friendsContainer}>

<Link href="/friends/search" asChild>

<TouchableOpacity style={styles.addFriendButton}>

<FontAwesome name="user-plus" size={18} color="#fff" style={styles.addFriendIcon} />

<Text style={styles.addFriendText}>Find Friends</Text>

</TouchableOpacity>

</Link>

<FlatList

data={dummyFriends}

keyExtractor={item => item.id}

renderItem={renderFriendItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>You have no friends yet</Text>

}

/>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

paddingTop: 60,

},

profileAvatarContainer: {

marginBottom: 10,

},

profileAvatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

profileAvatarText: {

fontSize: 40,

fontWeight: 'bold',

color: '#4a90e2',

},

profileName: {

color: '#fff',

fontSize: 24,

fontWeight: 'bold',

},

profileUsername: {

color: '#e6e6e6',

fontSize: 16,

marginTop: 4,

},

statsContainer: {

flexDirection: 'row',

marginTop: 20,

width: '80%',

justifyContent: 'space-around',

},

statItem: {

alignItems: 'center',

},

statNumber: {

color: '#fff',

fontSize: 20,

fontWeight: 'bold',

},

statLabel: {

color: '#e6e6e6',

fontSize: 14,

marginTop: 4,

},

tabContainer: {

flexDirection: 'row',

backgroundColor: 'white',

marginBottom: 10,

},

tab: {

flex: 1,

alignItems: 'center',

paddingVertical: 12,

borderBottomWidth: 2,

borderBottomColor: 'transparent',

},

activeTab: {

borderBottomColor: '#4a90e2',

},

tabText: {

fontSize: 16,

color: '#999',

},

activeTabText: {

color: '#4a90e2',

fontWeight: 'bold',

},

infoContainer: {

padding: 20,

},

infoItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

infoIcon: {

marginRight: 15,

},

infoText: {

fontSize: 16,

color: '#333',

},

friendsContainer: {

padding: 20,

},

addFriendButton: {

backgroundColor: '#4a90e2',

flexDirection: 'row',

alignItems: 'center',

justifyContent: 'center',

padding: 12,

borderRadius: 8,

marginBottom: 20,

},

addFriendIcon: {

marginRight: 8,

},

addFriendText: {

color: 'white',

fontSize: 16,

fontWeight: 'bold',

},

friendItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

avatarContainer: {

marginRight: 15,

},

avatar: {

width: 40,

height: 40,

borderRadius: 20,

},

avatarPlaceholder: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

color: 'white',

fontSize: 18,

fontWeight: 'bold',

},

friendName: {

fontSize: 16,

fontWeight: 'bold',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Створимо екран деталей друга, щоб переглядати його вішлісти:

bash

Copy

cat > "app/friends/[id].js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList, TouchableOpacity } from 'react-native';

import { useLocalSearchParams, Link } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані для друзів

const dummyFriends = {

'1': {

id: '1',

name: 'Anna Smith',

username: 'anna\_smith',

avatar: null,

wishlists: [

{ id: '1', name: 'Birthday Ideas', itemCount: 7 },

{ id: '2', name: 'Books to Read', itemCount: 12 },

]

},

'2': {

id: '2',

name: 'Mike Johnson',

username: 'mike\_j',

avatar: null,

wishlists: [

{ id: '1', name: 'Gadgets', itemCount: 5 },

{ id: '2', name: 'Outdoor Gear', itemCount: 3 },

]

},

'3': {

id: '3',

name: 'Sarah Williams',

username: 'sarah\_w',

avatar: null,

wishlists: [

{ id: '1', name: 'Kitchen Stuff', itemCount: 9 },

{ id: '2', name: 'Travel Wishlist', itemCount: 15 },

]

},

};

export default function FriendDetailsScreen() {

const { id } = useLocalSearchParams();

const friend = dummyFriends[id] || { name: 'Unknown', wishlists: [] };

const renderWishlistItem = ({ item }) => (

<Link href={`/friends/wishlist/${friend.id}/${item.id}`} asChild>

<TouchableOpacity style={styles.wishlistItem}>

<Text style={styles.wishlistName}>{item.name}</Text>

<Text style={styles.wishlistCount}>{item.itemCount} items</Text>

<FontAwesome name="chevron-right" size={16} color="#999" />

</TouchableOpacity>

</Link>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<View style={styles.avatarContainer}>

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{friend.name.charAt(0)}</Text>

</View>

</View>

<Text style={styles.friendName}>{friend.name}</Text>

<Text style={styles.friendUsername}>@{friend.username}</Text>

</View>

<View style={styles.wishlistsContainer}>

<Text style={styles.sectionTitle}>{friend.name}'s Wishlists</Text>

<FlatList

data={friend.wishlists}

keyExtractor={item => item.id}

renderItem={renderWishlistItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>This friend has no wishlists</Text>

}

/>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

paddingTop: 60,

},

avatarContainer: {

marginBottom: 10,

},

avatarPlaceholder: {

width: 80,

height: 80,

borderRadius: 40,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

fontSize: 32,

fontWeight: 'bold',

color: '#4a90e2',

},

friendName: {

color: '#fff',

fontSize: 22,

fontWeight: 'bold',

},

friendUsername: {

color: '#e6e6e6',

fontSize: 16,

marginTop: 4,

},

wishlistsContainer: {

padding: 20,

flex: 1,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 15,

},

wishlistItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

wishlistName: {

fontSize: 16,

fontWeight: 'bold',

flex: 1,

},

wishlistCount: {

color: '#666',

marginRight: 10,

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Додамо екран для перегляду вішліста друга:

bash

Copy

mkdir -p app/friends/wishlist

cat > "app/friends/wishlist/[friendId]/[wishlistId].js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList } from 'react-native';

import { useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані для вішлістів друзів

const dummyFriendsWishlists = {

'1': {

'1': {

name: 'Birthday Ideas',

items: [

{ id: '1', name: 'Bluetooth Speaker', price: 79, priority: 'High' },

{ id: '2', name: 'Art Supplies', price: 45, priority: 'Medium' },

{ id: '3', name: 'Cookbook', price: 25, priority: 'Low' },

]

},

'2': {

name: 'Books to Read',

items: [

{ id: '1', name: 'The Alchemist', price: 15, priority: 'Medium' },

{ id: '2', name: 'Atomic Habits', price: 18, priority: 'High' },

]

}

},

'2': {

'1': {

name: 'Gadgets',

items: [

{ id: '1', name: 'Wireless Earbuds', price: 129, priority: 'High' },

{ id: '2', name: 'Smart Watch', price: 249, priority: 'Medium' },

]

},

'2': {

name: 'Outdoor Gear',

items: [

{ id: '1', name: 'Hiking Boots', price: 120, priority: 'Medium' },

{ id: '2', name: 'Tent', price: 200, priority: 'Low' },

]

}

},

'3': {

'1': {

name: 'Kitchen Stuff',

items: [

{ id: '1', name: 'Stand Mixer', price: 299, priority: 'High' },

{ id: '2', name: 'Chef Knife Set', price: 150, priority: 'Medium' },

]

},

'2': {

name: 'Travel Wishlist',

items: [

{ id: '1', name: 'Carry-on Luggage', price: 120, priority: 'High' },

{ id: '2', name: 'Travel Pillow', price: 30, priority: 'Low' },

]

}

}

};

// Дані про друзів

const dummyFriends = {

'1': { name: 'Anna', username: 'anna\_smith' },

'2': { name: 'Mike', username: 'mike\_j' },

'3': { name: 'Sarah', username: 'sarah\_w' },

};

export default function FriendWishlistScreen() {

const { friendId, wishlistId } = useLocalSearchParams();

const friend = dummyFriends[friendId] || { name: 'Unknown' };

const wishlist = dummyFriendsWishlists[friendId]?.[wishlistId] || { name: 'Unknown List', items: [] };

const renderItem = ({ item }) => (

<View style={styles.item}>

<View style={styles.itemHeader}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

<View style={styles.itemPriceContainer}>

<Text style={styles.itemPrice}>${item.price}</Text>

<TouchableButtonGroup />

</View>

</View>

);

const TouchableButtonGroup = () => (

<View style={styles.buttonGroup}>

<View style={[styles.button, styles.giftButton]}>

<FontAwesome name="gift" size={16} color="#fff" />

<Text style={styles.buttonText}>Gift</Text>

</View>

<View style={[styles.button, styles.wishlistButton]}>

<FontAwesome name="plus" size={16} color="#fff" />

<Text style={styles.buttonText}>Add to My List</Text>

</View>

</View>

);

return (

<View style={styles.container}>

<Text style={styles.title}>{friend.name}'s {wishlist.name}</Text>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={renderItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>This wishlist is empty</Text>

}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

paddingTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemHeader: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 8,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

flex: 1,

},

itemPriority: {

fontSize: 14,

fontWeight: '500',

marginLeft: 8,

},

itemPriceContainer: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

},

itemPrice: {

fontSize: 16,

color: '#666',

},

buttonGroup: {

flexDirection: 'row',

},

button: {

flexDirection: 'row',

alignItems: 'center',

paddingVertical: 6,

paddingHorizontal: 12,

borderRadius: 4,

marginLeft: 8,

},

giftButton: {

backgroundColor: '#e74c3c',

},

wishlistButton: {

backgroundColor: '#4a90e2',

},

buttonText: {

color: 'white',

fontSize: 12,

fontWeight: 'bold',

marginLeft: 4,

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Оновимо роутер для нових екранів (видаліть існуючий і створіть новий \_layout.js):

bash

Copy

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function AppLayout() {

return (

<Tabs screenOptions={{ headerShown: false }}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="wishlist"

options={{

href: null, // Приховуємо цю вкладку з панелі вкладок

}}

/>

<Tabs.Screen

name="friends"

options={{

href: null, // Приховуємо цю вкладку з панелі вкладок

}}

/>

</Tabs>

);

}

EOF

Після створення цих файлів перезапустіть додаток:

bash

Copy

npx expo start --clear

Тепер ваш додаток матиме наступні функції для роботи з друзями:

Пошук та додавання друзів на екрані "Friends / Search"

Перегляд списку друзів на вкладці "Friends" в профілі

Перегляд профілю друга та його вішлістів

Перегляд конкретного вішліста друга з можливістю додавати елементи у свій вішліст або відмічати як подарунок

Це базова реалізація з тестовими даними. В реальному додатку ви б зберігали ці дані в базі даних та синхронізували їх між користувачами.

Чи бажаєте зосередитись на якомусь конкретному аспекті цієї функціональності?

Retry

V

....

fontSize: 14,

fontWeight: '500',

marginLeft: 8,

},

itemPriceContainer: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

},

itemPrice: {

fontSize: 16,

color: '#666',

},

buttonGroup: {

flexDirection: 'row',

},

button: {

flexDirection: 'row',

alignItems: 'center',

paddingVertical: 6,

paddingHorizontal: 12,

borderRadius: 4,

marginLeft: 8,

},

giftButton: {

backgroundColor: '#e74c3c',

},

wishlistButton: {

backgroundColor: '#4a90e2',

},

buttonText: {

color: 'white',

fontSize: 12,

fontWeight: 'bold',

marginLeft: 4,

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

zsh: no such file or directory: app/friends/wishlist/[friendId]/[wishlistId].js

Edit

Ви маєте рацію, маршрут з вкладеними динамічними параметрами не може бути правильно створений в такому форматі. Давайте змінимо підхід і спростимо структуру маршрутів:

bash

Copy

# Створюємо більш просту структуру для маршруту вішліста друга

mkdir -p app/friends/wishlist

cat > "app/friends/wishlist.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList } from 'react-native';

import { useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані для вішлістів друзів

const dummyFriendsWishlists = {

'1': {

'1': {

name: 'Birthday Ideas',

items: [

{ id: '1', name: 'Bluetooth Speaker', price: 79, priority: 'High' },

{ id: '2', name: 'Art Supplies', price: 45, priority: 'Medium' },

{ id: '3', name: 'Cookbook', price: 25, priority: 'Low' },

]

},

'2': {

name: 'Books to Read',

items: [

{ id: '1', name: 'The Alchemist', price: 15, priority: 'Medium' },

{ id: '2', name: 'Atomic Habits', price: 18, priority: 'High' },

]

}

},

'2': {

'1': {

name: 'Gadgets',

items: [

{ id: '1', name: 'Wireless Earbuds', price: 129, priority: 'High' },

{ id: '2', name: 'Smart Watch', price: 249, priority: 'Medium' },

]

}

},

'3': {

'1': {

name: 'Kitchen Stuff',

items: [

{ id: '1', name: 'Stand Mixer', price: 299, priority: 'High' },

{ id: '2', name: 'Chef Knife Set', price: 150, priority: 'Medium' },

]

}

}

};

// Дані про друзів

const dummyFriends = {

'1': { name: 'Anna', username: 'anna\_smith' },

'2': { name: 'Mike', username: 'mike\_j' },

'3': { name: 'Sarah', username: 'sarah\_w' },

};

export default function FriendWishlistScreen() {

const { friendId, wishlistId } = useLocalSearchParams();

const friend = dummyFriends[friendId] || { name: 'Unknown' };

const wishlist = dummyFriendsWishlists[friendId]?.[wishlistId] || { name: 'Unknown List', items: [] };

const renderItem = ({ item }) => (

<View style={styles.item}>

<View style={styles.itemHeader}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

<View style={styles.itemPriceContainer}>

<Text style={styles.itemPrice}>${item.price}</Text>

<TouchableButtonGroup />

</View>

</View>

);

const TouchableButtonGroup = () => (

<View style={styles.buttonGroup}>

<View style={[styles.button, styles.giftButton]}>

<FontAwesome name="gift" size={16} color="#fff" />

<Text style={styles.buttonText}>Gift</Text>

</View>

<View style={[styles.button, styles.wishlistButton]}>

<FontAwesome name="plus" size={16} color="#fff" />

<Text style={styles.buttonText}>Add to My List</Text>

</View>

</View>

);

return (

<View style={styles.container}>

<Text style={styles.title}>{friend.name}'s {wishlist.name}</Text>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={renderItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>This wishlist is empty</Text>

}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

paddingTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemHeader: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 8,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

flex: 1,

},

itemPriority: {

fontSize: 14,

fontWeight: '500',

marginLeft: 8,

},

itemPriceContainer: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

},

itemPrice: {

fontSize: 16,

color: '#666',

},

buttonGroup: {

flexDirection: 'row',

},

button: {

flexDirection: 'row',

alignItems: 'center',

paddingVertical: 6,

paddingHorizontal: 12,

borderRadius: 4,

marginLeft: 8,

},

giftButton: {

backgroundColor: '#e74c3c',

},

wishlistButton: {

backgroundColor: '#4a90e2',

},

buttonText: {

color: 'white',

fontSize: 12,

fontWeight: 'bold',

marginLeft: 4,

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Тепер нам потрібно змінити посилання у файлі app/friends/[id].js, щоб вони вказували на правильну структуру:

bash

Copy

# Оновлюємо посилання в екрані деталей друга

cat > "app/friends/[id].js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList, TouchableOpacity } from 'react-native';

import { useLocalSearchParams, Link } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані для друзів

const dummyFriends = {

'1': {

id: '1',

name: 'Anna Smith',

username: 'anna\_smith',

avatar: null,

wishlists: [

{ id: '1', name: 'Birthday Ideas', itemCount: 7 },

{ id: '2', name: 'Books to Read', itemCount: 12 },

]

},

'2': {

id: '2',

name: 'Mike Johnson',

username: 'mike\_j',

avatar: null,

wishlists: [

{ id: '1', name: 'Gadgets', itemCount: 5 },

{ id: '2', name: 'Outdoor Gear', itemCount: 3 },

]

},

'3': {

id: '3',

name: 'Sarah Williams',

username: 'sarah\_w',

avatar: null,

wishlists: [

{ id: '1', name: 'Kitchen Stuff', itemCount: 9 },

{ id: '2', name: 'Travel Wishlist', itemCount: 15 },

]

},

};

export default function FriendDetailsScreen() {

const { id } = useLocalSearchParams();

const friend = dummyFriends[id] || { name: 'Unknown', wishlists: [] };

const renderWishlistItem = ({ item }) => (

<Link href={{

pathname: "/friends/wishlist",

params: { friendId: friend.id, wishlistId: item.id }

}} asChild>

<TouchableOpacity style={styles.wishlistItem}>

<Text style={styles.wishlistName}>{item.name}</Text>

<Text style={styles.wishlistCount}>{item.itemCount} items</Text>

<FontAwesome name="chevron-right" size={16} color="#999" />

</TouchableOpacity>

</Link>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<View style={styles.avatarContainer}>

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{friend.name.charAt(0)}</Text>

</View>

</View>

<Text style={styles.friendName}>{friend.name}</Text>

<Text style={styles.friendUsername}>@{friend.username}</Text>

</View>

<View style={styles.wishlistsContainer}>

<Text style={styles.sectionTitle}>{friend.name}'s Wishlists</Text>

<FlatList

data={friend.wishlists}

keyExtractor={item => item.id}

renderItem={renderWishlistItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>This friend has no wishlists</Text>

}

/>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

paddingTop: 60,

},

avatarContainer: {

marginBottom: 10,

},

avatarPlaceholder: {

width: 80,

height: 80,

borderRadius: 40,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

fontSize: 32,

fontWeight: 'bold',

color: '#4a90e2',

},

friendName: {

color: '#fff',

fontSize: 22,

fontWeight: 'bold',

},

friendUsername: {

color: '#e6e6e6',

fontSize: 16,

marginTop: 4,

},

wishlistsContainer: {

padding: 20,

flex: 1,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 15,

},

wishlistItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

wishlistName: {

fontSize: 16,

fontWeight: 'bold',

flex: 1,

},

wishlistCount: {

color: '#666',

marginRight: 10,

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Перезапустіть додаток після цих змін:

bash

Copy

npx expo start --clear

Тепер структура маршрутів буде працювати коректно. Замість вкладених динамічних параметрів, ми передаємо обидва параметри до одного маршруту, що простіше і менш схильно до помилок у Expo Router.

Retry

V

проаналізуй що ти зробив

Edit

Я створив функціональність для соціальної взаємодії між користувачами у додатку Wishlist, що дозволяє знаходити друзів, переглядати їхні списки бажань та взаємодіяти з ними. Ось що було зроблено:

Пошук і додавання друзів:

Створив екран пошуку друзів (app/friends/search.js), де користувачі можуть шукати інших користувачів за іменем або ім'ям користувача

Реалізував інтерфейс з полем пошуку та списком результатів пошуку

Додав функцію для додавання знайдених користувачів до списку друзів

Управління списком друзів:

Оновив профіль користувача (app/profile.js), додавши вкладку "Friends"

Створив інтерфейс для відображення списку друзів

Додав кнопку для пошуку нових друзів

Перегляд профілів друзів:

Створив екран деталей друга (app/friends/[id].js), де відображається базова інформація про друга

Додав список вішлістів друга, які можна переглянути

Реалізував перехід до конкретного списку бажань друга

Перегляд вішлістів друзів:

Створив екран для перегляду конкретного вішліста друга (app/friends/wishlist.js)

Додав можливість переглядати елементи у вішлісті друга

Реалізував кнопки для взаємодії з елементами (позначити як подарунок, додати до власного списку)

Навігація та маршрутизація:

Оновив структуру маршрутів, додавши нові екрани

Налаштував правильну передачу параметрів між екранами

Вирішив проблему з вкладеними динамічними параметрами, спростивши структуру маршрутів

Архітектурні рішення:

Використав модульний підхід, створюючи окремі файли для кожного екрану

Застосував параметризовані маршрути для передачі ID друзів та вішлістів

Реалізував тестові дані для демонстрації функціоналу без підключення до бекенду

Створив інтуїтивний користувацький інтерфейс з правильною візуальною ієрархією

Важливо відзначити, що зараз це працює з тестовими даними, але в реальному додатку потрібно буде підключити базу даних та реалізувати синхронізацію даних між користувачами.

Retry

IMG\_31AACAB7A7A31.jpeg

V

ось як це все виглядає, трішки не правильно, у мене як в користувача має бути тільки поле для пошуку на головному екрані, нажавши на нього як маю можливість шукати своїх друзів серед усіх зареєстрованих користувачів, коли я знаходжу свого друга я маю можливість додати його до друзів нажавши на + на відповідного юзера який пявився в результаті пошуку, після того як мій друг прийме заявку на дружбу я зможу переглядати інформацію про нього і його вішлісти, точно так само він може робити все те ж саме і так кожен юзер який зареєстований

Edit

Зрозуміло, вам потрібен більш структурований та інтуїтивний підхід до функції пошуку друзів. Давайте переосмислимо цей процес і створимо правильний потік дій для користувача:

Додамо кнопку пошуку на головному екрані з вішлістами

Створимо повноцінний екран пошуку з відображенням результатів

Додамо функції запиту дружби та їх підтвердження/відхилення

Оновимо профіль для відображення друзів та запитів на дружбу

Давайте почнемо з оновлення головного екрану:

bash

Copy

# Оновлюємо головний екран, додаючи кнопку пошуку

cat > "app/index.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

const handleSearch = () => {

router.push('/search');

};

return (

<View style={styles.container}>

<View style={styles.header}>

<Text style={styles.title}>My Wishlists</Text>

<TouchableOpacity

style={styles.searchButton}

onPress={handleSearch}

>

<FontAwesome name="search" size={20} color="#4a90e2" />

</TouchableOpacity>

</View>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

Тепер створимо новий екран пошуку:

bash

Copy

# Створюємо екран пошуку користувачів

cat > "app/search.js" << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TextInput, FlatList, TouchableOpacity, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { router } from 'expo-router';

// Тестові дані користувачів

const dummyUsers = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'John Doe', username: 'john\_doe', avatar: null },

{ id: '3', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '4', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

{ id: '5', name: 'Robert Brown', username: 'rob\_brown', avatar: null },

];

export default function SearchScreen() {

const [searchQuery, setSearchQuery] = useState('');

const [results, setResults] = useState([]);

const [sentRequests, setSentRequests] = useState({}); // Відстежуємо надіслані запити

const handleSearch = (text) => {

setSearchQuery(text);

// Фільтруємо користувачів на основі пошукового запиту

if (text.trim() === '') {

setResults([]);

} else {

const filteredUsers = dummyUsers.filter(

user =>

user.name.toLowerCase().includes(text.toLowerCase()) ||

user.username.toLowerCase().includes(text.toLowerCase())

);

setResults(filteredUsers);

}

};

const handleSendFriendRequest = (userId) => {

// В реальному додатку тут було б API для надсилання запиту дружби

setSentRequests(prev => ({ ...prev, [userId]: true }));

};

const renderUserItem = ({ item }) => (

<TouchableOpacity

style={styles.userItem}

onPress={() => router.push(`/friends/${item.id}`)}

>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<View style={styles.userInfo}>

<Text style={styles.userName}>{item.name}</Text>

<Text style={styles.userUsername}>@{item.username}</Text>

</View>

{sentRequests[item.id] ? (

<View style={[styles.addButton, styles.requestSent]}>

<FontAwesome name="check" size={18} color="#27ae60" />

</View>

) : (

<TouchableOpacity

style={styles.addButton}

onPress={() => handleSendFriendRequest(item.id)}

>

<FontAwesome name="user-plus" size={18} color="#4a90e2" />

</TouchableOpacity>

)}

</TouchableOpacity>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>Find Friends</Text>

</View>

<View style={styles.searchContainer}>

<FontAwesome name="search" size={20} color="#999" style={styles.searchIcon} />

<TextInput

style={styles.searchInput}

placeholder="Search by name or username"

value={searchQuery}

onChangeText={handleSearch}

autoCapitalize="none"

autoFocus={true}

/>

{searchQuery.length > 0 && (

<TouchableOpacity

style={styles.clearButton}

onPress={() => setSearchQuery('')}

>

<FontAwesome name="times-circle" size={18} color="#999" />

</TouchableOpacity>

)}

</View>

{searchQuery.trim() !== '' ? (

<FlatList

data={results}

keyExtractor={item => item.id}

renderItem={renderUserItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>No users found</Text>

}

/>

) : (

<View style={styles.startSearchContainer}>

<FontAwesome name="users" size={60} color="#ddd" />

<Text style={styles.startSearchText}>

Search for users to add as friends

</Text>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchContainer: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

borderRadius: 8,

padding: 10,

marginBottom: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

searchIcon: {

marginRight: 10,

},

searchInput: {

flex: 1,

fontSize: 16,

},

clearButton: {

padding: 5,

},

userItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 12,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

avatarContainer: {

marginRight: 15,

},

avatar: {

width: 50,

height: 50,

borderRadius: 25,

},

avatarPlaceholder: {

width: 50,

height: 50,

borderRadius: 25,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

color: 'white',

fontSize: 24,

fontWeight: 'bold',

},

userInfo: {

flex: 1,

},

userName: {

fontSize: 16,

fontWeight: 'bold',

},

userUsername: {

fontSize: 14,

color: '#777',

},

addButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

borderWidth: 1,

borderColor: '#4a90e2',

},

requestSent: {

borderColor: '#27ae60',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

startSearchContainer: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

paddingBottom: 100,

},

startSearchText: {

marginTop: 20,

fontSize: 16,

color: '#999',

textAlign: 'center',

},

});

EOF

Тепер оновимо профіль, щоб додати секцію запитів на дружбу:

bash

Copy

# Оновлюємо екран профілю, додаючи запити на дружбу

cat > "app/profile.js" << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { Link } from 'expo-router';

// Тестові дані для друзів

const dummyFriends = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '3', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

];

// Тестові дані для запитів на дружбу

const dummyFriendRequests = [

{ id: '4', name: 'Robert Brown', username: 'rob\_brown', avatar: null },

];

export default function ProfileScreen() {

const [activeTab, setActiveTab] = useState('info');

const renderFriendItem = ({ item }) => (

<Link href={`/friends/${item.id}`} asChild>

<TouchableOpacity style={styles.friendItem}>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<Text style={styles.friendName}>{item.name}</Text>

</TouchableOpacity>

</Link>

);

const renderFriendRequestItem = ({ item }) => (

<View style={styles.requestItem}>

<View style={styles.requestUserInfo}>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<View style={styles.userTextInfo}>

<Text style={styles.friendName}>{item.name}</Text>

<Text style={styles.usernameText}>@{item.username}</Text>

</View>

</View>

<View style={styles.requestButtons}>

<TouchableOpacity style={[styles.requestButton, styles.acceptButton]}>

<FontAwesome name="check" size={16} color="#fff" />

</TouchableOpacity>

<TouchableOpacity style={[styles.requestButton, styles.rejectButton]}>

<FontAwesome name="times" size={16} color="#fff" />

</TouchableOpacity>

</View>

</View>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<View style={styles.profileAvatarContainer}>

<View style={styles.profileAvatar}>

<Text style={styles.profileAvatarText}>JD</Text>

</View>

</View>

<Text style={styles.profileName}>John Doe</Text>

<Text style={styles.profileUsername}>@john\_doe</Text>

<View style={styles.statsContainer}>

<View style={styles.statItem}>

<Text style={styles.statNumber}>5</Text>

<Text style={styles.statLabel}>Wishlists</Text>

</View>

<View style={styles.statItem}>

<Text style={styles.statNumber}>{dummyFriends.length}</Text>

<Text style={styles.statLabel}>Friends</Text>

</View>

</View>

</View>

<View style={styles.tabContainer}>

<TouchableOpacity

style={[styles.tab, activeTab === 'info' && styles.activeTab]}

onPress={() => setActiveTab('info')}

>

<Text style={[styles.tabText, activeTab === 'info' && styles.activeTabText]}>Info</Text>

</TouchableOpacity>

<TouchableOpacity

style={[styles.tab, activeTab === 'friends' && styles.activeTab]}

onPress={() => setActiveTab('friends')}

>

<Text style={[styles.tabText, activeTab === 'friends' && styles.activeTabText]}>Friends</Text>

</TouchableOpacity>

{dummyFriendRequests.length > 0 && (

<TouchableOpacity

style={[styles.tab, activeTab === 'requests' && styles.activeTab]}

onPress={() => setActiveTab('requests')}

>

<View style={styles.tabWithBadge}>

<Text style={[styles.tabText, activeTab === 'requests' && styles.activeTabText]}>Requests</Text>

<View style={styles.badge}>

<Text style={styles.badgeText}>{dummyFriendRequests.length}</Text>

</View>

</View>

</TouchableOpacity>

)}

</View>

{activeTab === 'info' && (

<View style={styles.infoContainer}>

<View style={styles.infoItem}>

<FontAwesome name="envelope" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>john.doe@example.com</Text>

</View>

<View style={styles.infoItem}>

<FontAwesome name="calendar" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>Member since January 2023</Text>

</View>

</View>

)}

{activeTab === 'friends' && (

<View style={styles.friendsContainer}>

<Link href="/search" asChild>

<TouchableOpacity style={styles.addFriendButton}>

<FontAwesome name="search" size={18} color="#fff" style={styles.addFriendIcon} />

<Text style={styles.addFriendText}>Find Friends</Text>

</TouchableOpacity>

</Link>

<FlatList

data={dummyFriends}

keyExtractor={item => item.id}

renderItem={renderFriendItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>You have no friends yet</Text>

}

/>

</View>

)}

{activeTab === 'requests' && (

<View style={styles.friendsContainer}>

<Text style={styles.sectionTitle}>Friend Requests</Text>

<FlatList

data={dummyFriendRequests}

keyExtractor={item => item.id}

renderItem={renderFriendRequestItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>No pending friend requests</Text>

}

/>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

paddingTop: 60,

},

profileAvatarContainer: {

marginBottom: 10,

},

profileAvatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

profileAvatarText: {

fontSize: 40,

fontWeight: 'bold',

color: '#4a90e2',

},

profileName: {

color: '#fff',

fontSize: 24,

fontWeight: 'bold',

},

profileUsername: {

color: '#e6e6e6',

fontSize: 16,

marginTop: 4,

},

statsContainer: {

flexDirection: 'row',

marginTop: 20,

width: '80%',

justifyContent: 'space-around',

},

statItem: {

alignItems: 'center',

},

statNumber: {

color: '#fff',

fontSize: 20,

fontWeight: 'bold',

},

statLabel: {

color: '#e6e6e6',

fontSize: 14,

marginTop: 4,

},

tabContainer: {

flexDirection: 'row',

backgroundColor: 'white',

marginBottom: 10,

},

tab: {

flex: 1,

alignItems: 'center',

paddingVertical: 12,

borderBottomWidth: 2,

borderBottomColor: 'transparent',

},

activeTab: {

borderBottomColor: '#4a90e2',

},

tabText: {

fontSize: 16,

color: '#999',

},

activeTabText: {

color: '#4a90e2',

fontWeight: 'bold',

},

tabWithBadge: {

flexDirection: 'row',

alignItems: 'center',

},

badge: {

backgroundColor: '#e74c3c',

borderRadius: 10,

width: 20,

height: 20,

justifyContent: 'center',

alignItems: 'center',

marginLeft: 6,

},

badgeText: {

color: 'white',

fontSize: 12,

fontWeight: 'bold',

},

infoContainer: {

padding: 20,

},

infoItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

infoIcon: {

marginRight: 15,

},

infoText: {

fontSize: 16,

color: '#333',

},

friendsContainer: {

padding: 20,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 15,

},

addFriendButton: {

backgroundColor: '#4a90e2',

flexDirection: 'row',

alignItems: 'center',

justifyContent: 'center',

padding: 12,

borderRadius: 8,

marginBottom: 20,

},

addFriendIcon: {

marginRight: 8,

},

addFriendText: {

color: 'white',

fontSize: 16,

fontWeight: 'bold',

},

friendItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

requestItem: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

requestUserInfo: {

flexDirection: 'row',

alignItems: 'center',

flex: 1,

},

userTextInfo: {

marginLeft: 15,

},

usernameText: {

fontSize: 14,

color: '#777',

},

requestButtons: {

flexDirection: 'row',

},

requestButton: {

width: 36,

height: 36,

borderRadius: 18,

justifyContent: 'center',

alignItems: 'center',

marginLeft: 10,

},

acceptButton: {

backgroundColor: '#27ae60',

},

rejectButton: {

backgroundColor: '#e74c3c',

},

avatarContainer: {

marginRight: 15,

},

avatar: {

width: 40,

height: 40,

borderRadius: 20,

},

avatarPlaceholder: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

color: 'white',

fontSize: 18,

fontWeight: 'bold',

},

friendName: {

fontSize: 16,

fontWeight: 'bold',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Тепер давайте переконаємося, що налаштування маршрутів правильне. Оновимо файл \_layout.js:

bash

Copy

# Оновлюємо основний макет маршрутів

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function AppLayout() {

return (

<Tabs screenOptions={{ headerShown: false }}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="wishlist"

options={{

href: null, // Приховуємо з панелі вкладок

}}

/>

<Tabs.Screen

name="friends"

options={{

href: null, // Приховуємо з панелі вкладок

}}

/>

<Tabs.Screen

name="search"

options={{

href: null, // Приховуємо з панелі вкладок

}}

/>

</Tabs>

);

}

EOF

Після всіх цих змін у вас буде повністю функціональний потік роботи з друзями:

На головному екрані є кнопка пошуку

При натисканні на кнопку пошуку відкривається екран пошуку, де можна шукати користувачів

Знайдених користувачів можна додати до друзів, надіславши їм запит

У профілі відображаються друзі та запити на дружбу, які можна прийняти або відхилити

Можна переглядати профілі друзів та їхні списки бажань

Ця реалізація відповідає описаному вами потоку взаємодії, де користувач може:

Шукати друзів серед усіх зареєстрованих користувачів

Відправляти запити на дружбу

Приймати запити від інших користувачів

Переглядати профілі та вішлісти друзів після підтвердження дружби

Перезапустіть додаток після цих змін:

bash

Copy

npx expo start --clear

2 / 2

Retry

V

а можемо видалити попередньо створені сторінки в нижній навігації вони ж вже не потрібні чи не так?

Edit

Ви абсолютно праві! Деякі з нижніх вкладок стали неактуальними після наших змін в архітектурі додатку. Давайте очистимо нижню навігацію, залишивши тільки необхідні вкладки:

bash

Copy

# Оновлюємо основний макет маршрутів, прибираючи зайві вкладки

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function AppLayout() {

return (

<Tabs screenOptions={{

headerShown: false,

tabBarActiveTintColor: '#4a90e2',

}}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

{/\* Ці екрани будуть доступні через навігацію, але не показуватимуться в нижній панелі вкладок \*/}

<Tabs.Screen

name="wishlist"

options={{

href: null,

}}

/>

<Tabs.Screen

name="friends"

options={{

href: null,

}}

/>

<Tabs.Screen

name="search"

options={{

href: null,

}}

/>

</Tabs>

);

}

EOF

Тепер нижня панель вкладок буде містити лише три основні вкладки:

Wishlists (списки бажань)

Profile (профіль користувача)

Settings (налаштування)

Інші екрани, такі як перегляд вішлістів (wishlist), профілі друзів (friends) та пошук (search) будуть доступні через навігацію в додатку, але не будуть з'являтися в нижній панелі вкладок.

Перезапустіть додаток, щоб побачити зміни:

bash

Copy

npx expo start --clear

Retry

V

все ще містить 7 табів

Edit

Вибачте за непорозуміння. Здається, що Expo Router може показувати всі екрани як таби навіть з опцією href: null. Давайте використаємо інший підхід для організації нашої навігації, щоб чітко розділити таби від інших екранів.

Нам потрібно переструктурувати файли, використовуючи інший підхід з Expo Router. Зробимо так:

bash

Copy

# Створимо структуру для табів

mkdir -p app/(tabs)

# Перемістимо основні файли табів

mv app/index.js app/(tabs)/

mv app/profile.js app/(tabs)/

mv app/settings.js app/(tabs)/

# Створимо новий \_layout.js для табів

cat > "app/(tabs)/\_layout.js" << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function TabsLayout() {

return (

<Tabs screenOptions={{

headerShown: false,

tabBarActiveTintColor: '#4a90e2',

}}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

# Оновимо головний \_layout.js

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Stack } from 'expo-router';

export default function RootLayout() {

return (

<Stack screenOptions={{ headerShown: false }}>

<Stack.Screen name="(tabs)" options={{ headerShown: false }} />

<Stack.Screen name="wishlist/[id]" options={{ headerShown: false }} />

<Stack.Screen name="search" options={{ headerShown: false }} />

<Stack.Screen name="friends/[id]" options={{ headerShown: false }} />

<Stack.Screen name="friends/wishlist" options={{ headerShown: false }} />

</Stack>

);

}

EOF

# Створимо новий індексний файл, що перенаправляє на таби

cat > "app/index.js" << 'EOF'

import { Redirect } from 'expo-router';

export default function IndexPage() {

return <Redirect href="/(tabs)" />;

}

EOF

Ця структура чітко розділяє основні таби від інших екранів. Таким чином:

У нижній навігації будуть лише 3 таби: Wishlists, Profile, Settings

Інші екрани (деталі вішліста, пошук, профілі друзів) будуть відображатися як окремі сторінки, а не таби

Перезапустіть додаток, щоб побачити зміни:

bash

Copy

npx expo start --clear

Це повинно вирішити проблему із зайвими табами в нижній панелі навігації.

Retry

IMG\_4F61CD3D1B2D1.jpeg

V

Edit

Виникла помилка з маршрутизацією. Здається, шлях "(tabs)" не може бути коректно визначений. Давайте виправимо це, використовуючи більш простий підхід до структури проекту.

Давайте повернемо все до попереднього стану, але очистимо нижню навігацію:

bash

Copy

# Видалимо створені файли

rm -f app/index.js

rm -rf app/(tabs)

# Відновимо основні файли в корені app

cat > "app/index.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

const handleSearch = () => {

router.push('/search');

};

return (

<View style={styles.container}>

<View style={styles.header}>

<Text style={styles.title}>My Wishlists</Text>

<TouchableOpacity

style={styles.searchButton}

onPress={handleSearch}

>

<FontAwesome name="search" size={20} color="#4a90e2" />

</TouchableOpacity>

</View>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

# Створимо новий, спрощений \_layout.js

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function Layout() {

return (

<Tabs screenOptions={{

headerShown: false,

tabBarActiveTintColor: '#4a90e2',

}}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

Після цього потрібно зробити ще один важливий крок - створити або перемістити вкладені екрани в нову структуру, щоб вони не відображалися як таби:

bash

Copy

# Створення директорії для вкладених екранів

mkdir -p app/modals

# Переміщення екранів пошуку та деталей в цю директорію

cat > "app/modals/search.js" << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TextInput, FlatList, TouchableOpacity, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { router } from 'expo-router';

// Тестові дані користувачів

const dummyUsers = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'John Doe', username: 'john\_doe', avatar: null },

{ id: '3', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '4', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

{ id: '5', name: 'Robert Brown', username: 'rob\_brown', avatar: null },

];

export default function SearchScreen() {

const [searchQuery, setSearchQuery] = useState('');

const [results, setResults] = useState([]);

const [sentRequests, setSentRequests] = useState({}); // Відстежуємо надіслані запити

const handleSearch = (text) => {

setSearchQuery(text);

// Фільтруємо користувачів на основі пошукового запиту

if (text.trim() === '') {

setResults([]);

} else {

const filteredUsers = dummyUsers.filter(

user =>

user.name.toLowerCase().includes(text.toLowerCase()) ||

user.username.toLowerCase().includes(text.toLowerCase())

);

setResults(filteredUsers);

}

};

const handleSendFriendRequest = (userId) => {

// В реальному додатку тут було б API для надсилання запиту дружби

setSentRequests(prev => ({ ...prev, [userId]: true }));

};

const renderUserItem = ({ item }) => (

<TouchableOpacity

style={styles.userItem}

onPress={() => router.push(`/modals/friend/${item.id}`)}

>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<View style={styles.userInfo}>

<Text style={styles.userName}>{item.name}</Text>

<Text style={styles.userUsername}>@{item.username}</Text>

</View>

{sentRequests[item.id] ? (

<View style={[styles.addButton, styles.requestSent]}>

<FontAwesome name="check" size={18} color="#27ae60" />

</View>

) : (

<TouchableOpacity

style={styles.addButton}

onPress={() => handleSendFriendRequest(item.id)}

>

<FontAwesome name="user-plus" size={18} color="#4a90e2" />

</TouchableOpacity>

)}

</TouchableOpacity>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>Find Friends</Text>

</View>

<View style={styles.searchContainer}>

<FontAwesome name="search" size={20} color="#999" style={styles.searchIcon} />

<TextInput

style={styles.searchInput}

placeholder="Search by name or username"

value={searchQuery}

onChangeText={handleSearch}

autoCapitalize="none"

autoFocus={true}

/>

{searchQuery.length > 0 && (

<TouchableOpacity

style={styles.clearButton}

onPress={() => setSearchQuery('')}

>

<FontAwesome name="times-circle" size={18} color="#999" />

</TouchableOpacity>

)}

</View>

{searchQuery.trim() !== '' ? (

<FlatList

data={results}

keyExtractor={item => item.id}

renderItem={renderUserItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>No users found</Text>

}

/>

) : (

<View style={styles.startSearchContainer}>

<FontAwesome name="users" size={60} color="#ddd" />

<Text style={styles.startSearchText}>

Search for users to add as friends

</Text>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchContainer: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

borderRadius: 8,

padding: 10,

marginBottom: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

searchIcon: {

marginRight: 10,

},

searchInput: {

flex: 1,

fontSize: 16,

},

clearButton: {

padding: 5,

},

userItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 12,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

avatarContainer: {

marginRight: 15,

},

avatar: {

width: 50,

height: 50,

borderRadius: 25,

},

avatarPlaceholder: {

width: 50,

height: 50,

borderRadius: 25,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

color: 'white',

fontSize: 24,

fontWeight: 'bold',

},

userInfo: {

flex: 1,

},

userName: {

fontSize: 16,

fontWeight: 'bold',

},

userUsername: {

fontSize: 14,

color: '#777',

},

addButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

borderWidth: 1,

borderColor: '#4a90e2',

},

requestSent: {

borderColor: '#27ae60',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

startSearchContainer: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

paddingBottom: 100,

},

startSearchText: {

marginTop: 20,

fontSize: 16,

color: '#999',

textAlign: 'center',

},

});

EOF

Також оновимо функцію пошуку на головному екрані, щоб вона вказувала на новий шлях:

bash

Copy

# Оновлюємо функцію пошуку в index.js

cat > "app/index.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

const handleSearch = () => {

router.push('/modals/search');

};

return (

<View style={styles.container}>

<View style={styles.header}>

<Text style={styles.title}>My Wishlists</Text>

<TouchableOpacity

style={styles.searchButton}

onPress={handleSearch}

>

<FontAwesome name="search" size={20} color="#4a90e2" />

</TouchableOpacity>

</View>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

Перезапустіть додаток після цих змін:

bash

Copy

npx expo start --clear

Цей підхід повинен вирішити проблему з маршрутизацією. Тепер ми маємо:

Три основні таби в нижній навігації (Wishlists, Profile, Settings)

Екран пошуку в модальній структурі, який не буде відображатися як таб

Retry

IMG\_33BB6FE3AD6E1.jpeg

V

Edit

Чудово! Тепер у нас є головний екран із списками бажань, кнопка пошуку у правому верхньому куті та нижня навігація з основними вкладками.

Однак я бачу, що у нижній навігації все ще є зайві вкладки (search, modals, friends, тощо). Давайте спробуємо інший підхід, щоб вирішити цю проблему.

В Expo Router є концепція "групування" маршрутів, які можна виключити з навігації. Давайте використаємо цей підхід:

Давайте модифікуємо \_layout.js:

javascript

Copy

import React from 'react';

import { Stack } from 'expo-router';

export default function Layout() {

return (

<Stack

screenOptions={{

headerShown: false

}}

/>

);

}

Тепер створимо групи для табів та інших екранів:

Copy

app/

\_layout.js // Головний лейаут (Stack)

(tabs)/ // Група для таббарів

\_layout.js // Таб лейаут

index.js // Головний екран вішлістів

profile.js // Профіль

settings.js // Налаштування

(modals)/ // Група для модальних екранів

search.js // Пошук друзів

friends/

[id].js // Профіль друга

wishlist.js // Вішліст друга

(others)/ // Інші екрани

wishlist/

[id].js // Деталі вішліста

Ця структура дозволить нам чітко розділити екрани, які повинні бути у вкладках, та ті, які повинні відображатися як окремі екрани.

Давайте впровадимо цю структуру:

Очистимо все та створимо нову структуру:

bash

Copy

# Створюємо резервну копію всіх файлів

mkdir -p backup

cp -r app/\* backup/

# Видаляємо всі файли в app

rm -rf app/\*

# Створюємо нову структуру

mkdir -p app/(tabs)

mkdir -p app/(modals)/friends

mkdir -p app/(others)/wishlist

Створюємо файли лейаута:

bash

Copy

# Головний лейаут

cat > "app/\_layout.js" << 'EOF'

import React from 'react';

import { Stack } from 'expo-router';

export default function RootLayout() {

return (

<Stack screenOptions={{ headerShown: false }} />

);

}

EOF

# Таб лейаут

cat > "app/(tabs)/\_layout.js" << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function TabsLayout() {

return (

<Tabs screenOptions={{

headerShown: false,

tabBarActiveTintColor: '#4a90e2',

}}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

Створюємо основні файли для кожної вкладки:

bash

Copy

# Файл для головного екрану вішлістів

cat > "app/(tabs)/index.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

const handleSearch = () => {

router.push('/(modals)/search');

};

return (

<View style={styles.container}>

<View style={styles.header}>

<Text style={styles.title}>My Wishlists</Text>

<TouchableOpacity

style={styles.searchButton}

onPress={handleSearch}

>

<FontAwesome name="search" size={20} color="#4a90e2" />

</TouchableOpacity>

</View>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/(others)/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

# Файл для профілю

cat > "app/(tabs)/profile.js" << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { Link } from 'expo-router';

// Тестові дані для друзів

const dummyFriends = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '3', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

];

// Тестові дані для запитів на дружбу

const dummyFriendRequests = [

{ id: '4', name: 'Robert Brown', username: 'rob\_brown', avatar: null },

];

export default function ProfileScreen() {

const [activeTab, setActiveTab] = useState('info');

const renderFriendItem = ({ item }) => (

<Link href={`/(modals)/friends/${item.id}`} asChild>

<TouchableOpacity style={styles.friendItem}>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<Text style={styles.friendName}>{item.name}</Text>

</TouchableOpacity>

</Link>

);

const renderFriendRequestItem = ({ item }) => (

<View style={styles.requestItem}>

<View style={styles.requestUserInfo}>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<View style={styles.userTextInfo}>

<Text style={styles.friendName}>{item.name}</Text>

<Text style={styles.usernameText}>@{item.username}</Text>

</View>

</View>

<View style={styles.requestButtons}>

<TouchableOpacity style={[styles.requestButton, styles.acceptButton]}>

<FontAwesome name="check" size={16} color="#fff" />

</TouchableOpacity>

<TouchableOpacity style={[styles.requestButton, styles.rejectButton]}>

<FontAwesome name="times" size={16} color="#fff" />

</TouchableOpacity>

</View>

</View>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<View style={styles.profileAvatarContainer}>

<View style={styles.profileAvatar}>

<Text style={styles.profileAvatarText}>JD</Text>

</View>

</View>

<Text style={styles.profileName}>John Doe</Text>

<Text style={styles.profileUsername}>@john\_doe</Text>

<View style={styles.statsContainer}>

<View style={styles.statItem}>

<Text style={styles.statNumber}>5</Text>

<Text style={styles.statLabel}>Wishlists</Text>

</View>

<View style={styles.statItem}>

<Text style={styles.statNumber}>{dummyFriends.length}</Text>

<Text style={styles.statLabel}>Friends</Text>

</View>

</View>

</View>

<View style={styles.tabContainer}>

<TouchableOpacity

style={[styles.tab, activeTab === 'info' && styles.activeTab]}

onPress={() => setActiveTab('info')}

>

<Text style={[styles.tabText, activeTab === 'info' && styles.activeTabText]}>Info</Text>

</TouchableOpacity>

<TouchableOpacity

style={[styles.tab, activeTab === 'friends' && styles.activeTab]}

onPress={() => setActiveTab('friends')}

>

<Text style={[styles.tabText, activeTab === 'friends' && styles.activeTabText]}>Friends</Text>

</TouchableOpacity>

{dummyFriendRequests.length > 0 && (

<TouchableOpacity

style={[styles.tab, activeTab === 'requests' && styles.activeTab]}

onPress={() => setActiveTab('requests')}

>

<View style={styles.tabWithBadge}>

<Text style={[styles.tabText, activeTab === 'requests' && styles.activeTabText]}>Requests</Text>

<View style={styles.badge}>

<Text style={styles.badgeText}>{dummyFriendRequests.length}</Text>

</View>

</View>

</TouchableOpacity>

)}

</View>

{activeTab === 'info' && (

<View style={styles.infoContainer}>

<View style={styles.infoItem}>

<FontAwesome name="envelope" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>john.doe@example.com</Text>

</View>

<View style={styles.infoItem}>

<FontAwesome name="calendar" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>Member since January 2023</Text>

</View>

</View>

)}

{activeTab === 'friends' && (

<View style={styles.friendsContainer}>

<Link href="/(modals)/search" asChild>

<TouchableOpacity style={styles.addFriendButton}>

<FontAwesome name="search" size={18} color="#fff" style={styles.addFriendIcon} />

<Text style={styles.addFriendText}>Find Friends</Text>

</TouchableOpacity>

</Link>

<FlatList

data={dummyFriends}

keyExtractor={item => item.id}

renderItem={renderFriendItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>You have no friends yet</Text>

}

/>

</View>

)}

{activeTab === 'requests' && (

<View style={styles.friendsContainer}>

<Text style={styles.sectionTitle}>Friend Requests</Text>

<FlatList

data={dummyFriendRequests}

keyExtractor={item => item.id}

renderItem={renderFriendRequestItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>No pending friend requests</Text>

}

/>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

// Всі стилі залишаються такими ж

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

paddingTop: 60,

},

profileAvatarContainer: {

marginBottom: 10,

},

profileAvatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

profileAvatarText: {

fontSize: 40,

fontWeight: 'bold',

color: '#4a90e2',

},

profileName: {

color: '#fff',

fontSize: 24,

fontWeight: 'bold',

},

profileUsername: {

color: '#e6e6e6',

fontSize: 16,

marginTop: 4,

},

statsContainer: {

flexDirection: 'row',

marginTop: 20,

width: '80%',

justifyContent: 'space-around',

},

statItem: {

alignItems: 'center',

},

statNumber: {

color: '#fff',

fontSize: 20,

fontWeight: 'bold',

},

statLabel: {

color: '#e6e6e6',

fontSize: 14,

marginTop: 4,

},

tabContainer: {

flexDirection: 'row',

backgroundColor: 'white',

marginBottom: 10,

},

tab: {

flex: 1,

alignItems: 'center',

paddingVertical: 12,

borderBottomWidth: 2,

borderBottomColor: 'transparent',

},

activeTab: {

borderBottomColor: '#4a90e2',

},

tabText: {

fontSize: 16,

color: '#999',

},

activeTabText: {

color: '#4a90e2',

fontWeight: 'bold',

},

tabWithBadge: {

flexDirection: 'row',

alignItems: 'center',

},

badge: {

backgroundColor: '#e74c3c',

borderRadius: 10,

width: 20,

height: 20,

justifyContent: 'center',

alignItems: 'center',

marginLeft: 6,

},

badgeText: {

color: 'white',

fontSize: 12,

fontWeight: 'bold',

},

infoContainer: {

padding: 20,

},

infoItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

infoIcon: {

marginRight: 15,

},

infoText: {

fontSize: 16,

color: '#333',

},

friendsContainer: {

padding: 20,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 15,

},

addFriendButton: {

backgroundColor: '#4a90e2',

flexDirection: 'row',

alignItems: 'center',

justifyContent: 'center',

padding: 12,

borderRadius: 8,

marginBottom: 20,

},

addFriendIcon: {

marginRight: 8,

},

addFriendText: {

color: 'white',

fontSize: 16,

fontWeight: 'bold',

},

friendItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

requestItem: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

requestUserInfo: {

flexDirection: 'row',

alignItems: 'center',

flex: 1,

},

userTextInfo: {

marginLeft: 15,

},

usernameText: {

fontSize: 14,

color: '#777',

},

requestButtons: {

flexDirection: 'row',

},

requestButton: {

width: 36,

height: 36,

borderRadius: 18,

justifyContent: 'center',

alignItems: 'center',

marginLeft: 10,

},

acceptButton: {

backgroundColor: '#27ae60',

},

rejectButton: {

backgroundColor: '#e74c3c',

},

avatarContainer: {

marginRight: 15,

},

avatar: {

width: 40,

height: 40,

borderRadius: 20,

},

avatarPlaceholder: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

color: 'white',

fontSize: 18,

fontWeight: 'bold',

},

friendName: {

fontSize: 16,

fontWeight: 'bold',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Не забудьте створити екран налаштувань:

bash

Copy

cat > "app/(tabs)/settings.js" << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

const settingsSections = [

{

title: 'Account',

items: [

{ id: '1', title: 'Edit Profile', icon: 'user' },

{ id: '2', title: 'Change Password', icon: 'lock' },

{ id: '3', title: 'Privacy', icon: 'shield' },

]

},

{

title: 'Notifications',

items: [

{ id: '4', title: 'Push Notifications', icon: 'bell' },

{ id: '5', title: 'Email Notifications', icon: 'envelope' },

]

},

{

title: 'About',

items: [

{ id: '6', title: 'Terms of Service', icon: 'file-text-o' },

{ id: '7', title: 'Privacy Policy', icon: 'shield' },

{ id: '8', title: 'Help & Support', icon: 'question-circle' },

]

},

];

export default function SettingsScreen() {

const renderSettingItem = ({ item }) => (

<TouchableOpacity style={styles.settingItem}>

<FontAwesome name={item.icon} size={20} color="#4a90e2" style={styles.settingIcon} />

<Text style={styles.settingTitle}>{item.title}</Text>

<FontAwesome name="chevron-right" size={16} color="#999" />

</TouchableOpacity>

);

const renderSectionHeader = ({ section }) => (

<Text style={styles.sectionTitle}>{section.title}</Text>

);

return (

<View style={styles.container}>

<Text style={styles.screenTitle}>Settings</Text>

<FlatList

data={settingsSections}

keyExtractor={(item, index) => `section-${index}`}

renderItem={({ item }) => (

<View style={styles.section}>

<Text style={styles.sectionTitle}>{item.title}</Text>

{item.items.map(settingItem => (

<TouchableOpacity key={settingItem.id} style={styles.settingItem}>

<FontAwesome name={settingItem.icon} size={20} color="#4a90e2" style={styles.settingIcon} />

<Text style={styles.settingTitle}>{settingItem.title}</Text>

<FontAwesome name="chevron-right" size={16} color="#999" />

</TouchableOpacity>

))}

</View>

)}

/>

<TouchableOpacity style={styles.logoutButton}>

<FontAwesome name="sign-out" size={18} color="#e74c3c" style={styles.logoutIcon} />

<Text style={styles.logoutText}>Logout</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

screenTitle: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 20,

marginTop: 60,

},

section: {

marginBottom: 24,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 12,

color: '#333',

},

settingItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 8,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

settingIcon: {

marginRight: 16,

width: 24,

textAlign: 'center',

},

settingTitle: {

fontSize: 16,

color: '#333',

flex: 1,

},

logoutButton: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginTop: 24,

justifyContent: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

logoutIcon: {

marginRight: 8,

},

logoutText: {

fontSize: 16,

color: '#e74c3c',

fontWeight: 'bold',

},

});

EOF

Створіть основний файл перенаправлення:

bash

Copy

cat > "app/index.js" << 'EOF'

import { Redirect } from 'expo-router';

export default function Index() {

return <Redirect href="/(tabs)" />;

}

EOF

Перезапустіть додаток із очищенням кешу:

bash

Copy

npx expo start --clear

Retry

V

# Створюємо резервну копію всіх файлів

mkdir -p backup

cp -r app/\* backup/

# Видаляємо всі файли в app

rm -rf app/\*

# Створюємо нову структуру

mkdir -p app/(tabs)

mkdir -p app/(modals)/friends

mkdir -p app/(others)/wishlist

zsh: no matches found: app/\*

zsh: sure you want to delete all the files in /Users/test/Documents/WishlistApp/app [yn]? y

zsh: no matches found: app/\*

zsh: number expected

zsh: no matches found: app/(modals)/friends

zsh: no matches found: app/(others)/wishlist

Edit

Здається, у вас виникла проблема зі створенням папок через термінал. Давайте спробуємо інший підхід.

Зробимо наступне:

Спочатку створимо базову структуру папок:

bash

Copy

# Створюємо базові папки

mkdir -p app

mkdir -p app/\(tabs\)

mkdir -p app/\(modals\)/friends

mkdir -p app/\(others\)/wishlist

Зверніть увагу на екранування дужок \( та \) в назвах папок. Це потрібно, щоб bash не інтерпретував їх як спеціальні символи.

Перевірте, чи створилися папки:

bash

Copy

ls -la app

Якщо папки не створилися через термінал, можна використати графічний інтерфейс файлового менеджера для створення такої структури:

Copy

app/

(tabs)/

(modals)/

friends/

(others)/

wishlist/

Після того, як структура папок буде готова, ми можемо продовжити створення файлів з кодом.

Retry

V

продовжимо

Edit

Чудово, давайте продовжимо і створимо всі необхідні файли для нашої нової структури:

Спочатку створимо головний файл \_layout.js для нашого додатка:

bash

Copy

cat > app/\_layout.js << 'EOF'

import React from 'react';

import { Stack } from 'expo-router';

export default function RootLayout() {

return (

<Stack screenOptions={{ headerShown: false }} />

);

}

EOF

Створимо файл \_layout.js для вкладок:

bash

Copy

cat > app/\(tabs\)/\_layout.js << 'EOF'

import React from 'react';

import { Tabs } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function TabsLayout() {

return (

<Tabs screenOptions={{

headerShown: false,

tabBarActiveTintColor: '#4a90e2',

}}>

<Tabs.Screen

name="index"

options={{

title: 'Wishlists',

tabBarIcon: ({ color }) => <FontAwesome name="list" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="profile"

options={{

title: 'Profile',

tabBarIcon: ({ color }) => <FontAwesome name="user" size={24} color={color} />,

}}

/>

<Tabs.Screen

name="settings"

options={{

title: 'Settings',

tabBarIcon: ({ color }) => <FontAwesome name="gear" size={24} color={color} />,

}}

/>

</Tabs>

);

}

EOF

Створимо файли для основних вкладок:

bash

Copy

# Файл для головного екрану вішлістів

cat > app/\(tabs\)/index.js << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

const handleSearch = () => {

router.push('/(modals)/search');

};

return (

<View style={styles.container}>

<View style={styles.header}>

<Text style={styles.title}>My Wishlists</Text>

<TouchableOpacity

style={styles.searchButton}

onPress={handleSearch}

>

<FontAwesome name="search" size={20} color="#4a90e2" />

</TouchableOpacity>

</View>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/(others)/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

# Файл для профілю

cat > app/\(tabs\)/profile.js << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { Link } from 'expo-router';

// Тестові дані для друзів

const dummyFriends = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '3', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

];

// Тестові дані для запитів на дружбу

const dummyFriendRequests = [

{ id: '4', name: 'Robert Brown', username: 'rob\_brown', avatar: null },

];

export default function ProfileScreen() {

const [activeTab, setActiveTab] = useState('info');

const renderFriendItem = ({ item }) => (

<Link href={`/(modals)/friends/${item.id}`} asChild>

<TouchableOpacity style={styles.friendItem}>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<Text style={styles.friendName}>{item.name}</Text>

</TouchableOpacity>

</Link>

);

const renderFriendRequestItem = ({ item }) => (

<View style={styles.requestItem}>

<View style={styles.requestUserInfo}>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<View style={styles.userTextInfo}>

<Text style={styles.friendName}>{item.name}</Text>

<Text style={styles.usernameText}>@{item.username}</Text>

</View>

</View>

<View style={styles.requestButtons}>

<TouchableOpacity style={[styles.requestButton, styles.acceptButton]}>

<FontAwesome name="check" size={16} color="#fff" />

</TouchableOpacity>

<TouchableOpacity style={[styles.requestButton, styles.rejectButton]}>

<FontAwesome name="times" size={16} color="#fff" />

</TouchableOpacity>

</View>

</View>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<View style={styles.profileAvatarContainer}>

<View style={styles.profileAvatar}>

<Text style={styles.profileAvatarText}>JD</Text>

</View>

</View>

<Text style={styles.profileName}>John Doe</Text>

<Text style={styles.profileUsername}>@john\_doe</Text>

<View style={styles.statsContainer}>

<View style={styles.statItem}>

<Text style={styles.statNumber}>5</Text>

<Text style={styles.statLabel}>Wishlists</Text>

</View>

<View style={styles.statItem}>

<Text style={styles.statNumber}>{dummyFriends.length}</Text>

<Text style={styles.statLabel}>Friends</Text>

</View>

</View>

</View>

<View style={styles.tabContainer}>

<TouchableOpacity

style={[styles.tab, activeTab === 'info' && styles.activeTab]}

onPress={() => setActiveTab('info')}

>

<Text style={[styles.tabText, activeTab === 'info' && styles.activeTabText]}>Info</Text>

</TouchableOpacity>

<TouchableOpacity

style={[styles.tab, activeTab === 'friends' && styles.activeTab]}

onPress={() => setActiveTab('friends')}

>

<Text style={[styles.tabText, activeTab === 'friends' && styles.activeTabText]}>Friends</Text>

</TouchableOpacity>

{dummyFriendRequests.length > 0 && (

<TouchableOpacity

style={[styles.tab, activeTab === 'requests' && styles.activeTab]}

onPress={() => setActiveTab('requests')}

>

<View style={styles.tabWithBadge}>

<Text style={[styles.tabText, activeTab === 'requests' && styles.activeTabText]}>Requests</Text>

<View style={styles.badge}>

<Text style={styles.badgeText}>{dummyFriendRequests.length}</Text>

</View>

</View>

</TouchableOpacity>

)}

</View>

{activeTab === 'info' && (

<View style={styles.infoContainer}>

<View style={styles.infoItem}>

<FontAwesome name="envelope" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>john.doe@example.com</Text>

</View>

<View style={styles.infoItem}>

<FontAwesome name="calendar" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>Member since January 2023</Text>

</View>

</View>

)}

{activeTab === 'friends' && (

<View style={styles.friendsContainer}>

<Link href="/(modals)/search" asChild>

<TouchableOpacity style={styles.addFriendButton}>

<FontAwesome name="search" size={18} color="#fff" style={styles.addFriendIcon} />

<Text style={styles.addFriendText}>Find Friends</Text>

</TouchableOpacity>

</Link>

<FlatList

data={dummyFriends}

keyExtractor={item => item.id}

renderItem={renderFriendItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>You have no friends yet</Text>

}

/>

</View>

)}

{activeTab === 'requests' && (

<View style={styles.friendsContainer}>

<Text style={styles.sectionTitle}>Friend Requests</Text>

<FlatList

data={dummyFriendRequests}

keyExtractor={item => item.id}

renderItem={renderFriendRequestItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>No pending friend requests</Text>

}

/>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

paddingTop: 60,

},

profileAvatarContainer: {

marginBottom: 10,

},

profileAvatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

profileAvatarText: {

fontSize: 40,

fontWeight: 'bold',

color: '#4a90e2',

},

profileName: {

color: '#fff',

fontSize: 24,

fontWeight: 'bold',

},

profileUsername: {

color: '#e6e6e6',

fontSize: 16,

marginTop: 4,

},

statsContainer: {

flexDirection: 'row',

marginTop: 20,

width: '80%',

justifyContent: 'space-around',

},

statItem: {

alignItems: 'center',

},

statNumber: {

color: '#fff',

fontSize: 20,

fontWeight: 'bold',

},

statLabel: {

color: '#e6e6e6',

fontSize: 14,

marginTop: 4,

},

tabContainer: {

flexDirection: 'row',

backgroundColor: 'white',

marginBottom: 10,

},

tab: {

flex: 1,

alignItems: 'center',

paddingVertical: 12,

borderBottomWidth: 2,

borderBottomColor: 'transparent',

},

activeTab: {

borderBottomColor: '#4a90e2',

},

tabText: {

fontSize: 16,

color: '#999',

},

activeTabText: {

color: '#4a90e2',

fontWeight: 'bold',

},

tabWithBadge: {

flexDirection: 'row',

alignItems: 'center',

},

badge: {

backgroundColor: '#e74c3c',

borderRadius: 10,

width: 20,

height: 20,

justifyContent: 'center',

alignItems: 'center',

marginLeft: 6,

},

badgeText: {

color: 'white',

fontSize: 12,

fontWeight: 'bold',

},

infoContainer: {

padding: 20,

},

infoItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

infoIcon: {

marginRight: 15,

},

infoText: {

fontSize: 16,

color: '#333',

},

friendsContainer: {

padding: 20,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 15,

},

addFriendButton: {

backgroundColor: '#4a90e2',

flexDirection: 'row',

alignItems: 'center',

justifyContent: 'center',

padding: 12,

borderRadius: 8,

marginBottom: 20,

},

addFriendIcon: {

marginRight: 8,

},

addFriendText: {

color: 'white',

fontSize: 16,

fontWeight: 'bold',

},

friendItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

requestItem: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

requestUserInfo: {

flexDirection: 'row',

alignItems: 'center',

flex: 1,

},

userTextInfo: {

marginLeft: 15,

},

usernameText: {

fontSize: 14,

color: '#777',

},

requestButtons: {

flexDirection: 'row',

},

requestButton: {

width: 36,

height: 36,

borderRadius: 18,

justifyContent: 'center',

alignItems: 'center',

marginLeft: 10,

},

acceptButton: {

backgroundColor: '#27ae60',

},

rejectButton: {

backgroundColor: '#e74c3c',

},

avatarContainer: {

marginRight: 15,

},

avatar: {

width: 40,

height: 40,

borderRadius: 20,

},

avatarPlaceholder: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

color: 'white',

fontSize: 18,

fontWeight: 'bold',

},

friendName: {

fontSize: 16,

fontWeight: 'bold',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

# Файл для налаштувань

cat > app/\(tabs\)/settings.js << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

const settingsSections = [

{

title: 'Account',

items: [

{ id: '1', title: 'Edit Profile', icon: 'user' },

{ id: '2', title: 'Change Password', icon: 'lock' },

{ id: '3', title: 'Privacy', icon: 'shield' },

]

},

{

title: 'Notifications',

items: [

{ id: '4', title: 'Push Notifications', icon: 'bell' },

{ id: '5', title: 'Email Notifications', icon: 'envelope' },

]

},

{

title: 'About',

items: [

{ id: '6', title: 'Terms of Service', icon: 'file-text-o' },

{ id: '7', title: 'Privacy Policy', icon: 'shield' },

{ id: '8', title: 'Help & Support', icon: 'question-circle' },

]

},

];

export default function SettingsScreen() {

const renderSettingItem = ({ item }) => (

<TouchableOpacity style={styles.settingItem}>

<FontAwesome name={item.icon} size={20} color="#4a90e2" style={styles.settingIcon} />

<Text style={styles.settingTitle}>{item.title}</Text>

<FontAwesome name="chevron-right" size={16} color="#999" />

</TouchableOpacity>

);

const renderSectionHeader = ({ section }) => (

<Text style={styles.sectionTitle}>{section.title}</Text>

);

return (

<View style={styles.container}>

<Text style={styles.screenTitle}>Settings</Text>

<FlatList

data={settingsSections}

keyExtractor={(item, index) => `section-${index}`}

renderItem={({ item }) => (

<View style={styles.section}>

<Text style={styles.sectionTitle}>{item.title}</Text>

{item.items.map(settingItem => (

<TouchableOpacity key={settingItem.id} style={styles.settingItem}>

<FontAwesome name={settingItem.icon} size={20} color="#4a90e2" style={styles.settingIcon} />

<Text style={styles.settingTitle}>{settingItem.title}</Text>

<FontAwesome name="chevron-right" size={16} color="#999" />

</TouchableOpacity>

))}

</View>

)}

/>

<TouchableOpacity style={styles.logoutButton}>

<FontAwesome name="sign-out" size={18} color="#e74c3c" style={styles.logoutIcon} />

<Text style={styles.logoutText}>Logout</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

screenTitle: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 20,

marginTop: 60,

},

section: {

marginBottom: 24,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 12,

color: '#333',

},

settingItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 8,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

settingIcon: {

marginRight: 16,

width: 24,

textAlign: 'center',

},

settingTitle: {

fontSize: 16,

color: '#333',

flex: 1,

},

logoutButton: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginTop: 24,

justifyContent: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

logoutIcon: {

marginRight: 8,

},

logoutText: {

fontSize: 16,

color: '#e74c3c',

fontWeight: 'bold',

},

});

EOF

Додамо файл для екрану пошуку:

bash

Copy

# Створюємо файл пошуку

mkdir -p app/\(modals\)

cat > app/\(modals\)/search.js << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TextInput, FlatList, TouchableOpacity, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { router } from 'expo-router';

// Тестові дані користувачів

const dummyUsers = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'John Doe', username: 'john\_doe', avatar: null },

{ id: '3', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '4', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

{ id: '5', name: 'Robert Brown', username: 'rob\_brown', avatar: null },

];

export default function SearchScreen() {

const [searchQuery, setSearchQuery] = useState('');

const [results, setResults] = useState([]);

const [sentRequests, setSentRequests] = useState({}); // Відстежуємо надіслані запити

const handleSearch = (text) => {

setSearchQuery(text);

// Фільтруємо користувачів на основі пошукового запиту

if (text.trim() === '') {

setResults([]);

} else {

const filteredUsers = dummyUsers.filter(

user =>

user.name.toLowerCase().includes(text.toLowerCase()) ||

user.username.toLowerCase().includes(text.toLowerCase())

);

setResults(filteredUsers);

}

};

const handleSendFriendRequest = (userId) => {

// В реальному додатку тут було б API для надсилання запиту дружби

setSentRequests(prev => ({ ...prev, [userId]: true }));

};

const renderUserItem = ({ item }) => (

<TouchableOpacity

style={styles.userItem}

onPress={() => router.push(`/(modals)/friends/${item.id}`)}

>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<View style={styles.userInfo}>

<Text style={styles.userName}>{item.name}</Text>

<Text style={styles.userUsername}>@{item.username}</Text>

</View>

{sentRequests[item.id] ? (

<View style={[styles.addButton, styles.requestSent]}>

<FontAwesome name="check" size={18} color="#27ae60" />

</View>

) : (

<TouchableOpacity

style={styles.addButton}

onPress={() => handleSendFriendRequest(item.id)}

>

<FontAwesome name="user-plus" size={18} color="#4a90e2" />

</TouchableOpacity>

)}

</TouchableOpacity>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>Find Friends</Text>

</View>

<View style={styles.searchContainer}>

<FontAwesome name="search" size={20} color="#999" style={styles.searchIcon} />

<TextInput

style={styles.searchInput}

placeholder="Search by name or username"

value={searchQuery}

onChangeText={handleSearch}

autoCapitalize="none"

autoFocus={true}

/>

{searchQuery.length > 0 && (

<TouchableOpacity

style={styles.clearButton}

onPress={() => setSearchQuery('')}

>

<FontAwesome name="times-circle" size={18} color="#999" />

</TouchableOpacity>

)}

</View>

{searchQuery.trim() !== '' ? (

<FlatList

data={results}

keyExtractor={item => item.id}

renderItem={renderUserItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>No users found</Text>

}

/>

) : (

<View style={styles.startSearchContainer}>

<FontAwesome name="users" size={60} color="#ddd" />

<Text style={styles.startSearchText}>

Search for users to add as friends

</Text>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchContainer: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

borderRadius: 8,

padding: 10,

marginBottom: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

searchIcon: {

marginRight: 10,

},

searchInput: {

flex: 1,

fontSize: 16,

},

clearButton: {

padding: 5,

},

userItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 12,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

avat

2 / 2

Retry

V

ти зупинився на Додамо файл для екрану пошуку:

Edit

Давайте продовжимо і закінчимо файл для екрану пошуку:

bash

Copy

cat > app/\(modals\)/search.js << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TextInput, FlatList, TouchableOpacity, Image } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { router } from 'expo-router';

// Тестові дані користувачів

const dummyUsers = [

{ id: '1', name: 'Anna Smith', username: 'anna\_smith', avatar: null },

{ id: '2', name: 'John Doe', username: 'john\_doe', avatar: null },

{ id: '3', name: 'Mike Johnson', username: 'mike\_j', avatar: null },

{ id: '4', name: 'Sarah Williams', username: 'sarah\_w', avatar: null },

{ id: '5', name: 'Robert Brown', username: 'rob\_brown', avatar: null },

];

export default function SearchScreen() {

const [searchQuery, setSearchQuery] = useState('');

const [results, setResults] = useState([]);

const [sentRequests, setSentRequests] = useState({}); // Відстежуємо надіслані запити

const handleSearch = (text) => {

setSearchQuery(text);

// Фільтруємо користувачів на основі пошукового запиту

if (text.trim() === '') {

setResults([]);

} else {

const filteredUsers = dummyUsers.filter(

user =>

user.name.toLowerCase().includes(text.toLowerCase()) ||

user.username.toLowerCase().includes(text.toLowerCase())

);

setResults(filteredUsers);

}

};

const handleSendFriendRequest = (userId) => {

// В реальному додатку тут було б API для надсилання запиту дружби

setSentRequests(prev => ({ ...prev, [userId]: true }));

};

const renderUserItem = ({ item }) => (

<TouchableOpacity

style={styles.userItem}

onPress={() => router.push(`/(modals)/friends/${item.id}`)}

>

<View style={styles.avatarContainer}>

{item.avatar ? (

<Image source={{ uri: item.avatar }} style={styles.avatar} />

) : (

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{item.name.charAt(0)}</Text>

</View>

)}

</View>

<View style={styles.userInfo}>

<Text style={styles.userName}>{item.name}</Text>

<Text style={styles.userUsername}>@{item.username}</Text>

</View>

{sentRequests[item.id] ? (

<View style={[styles.addButton, styles.requestSent]}>

<FontAwesome name="check" size={18} color="#27ae60" />

</View>

) : (

<TouchableOpacity

style={styles.addButton}

onPress={() => handleSendFriendRequest(item.id)}

>

<FontAwesome name="user-plus" size={18} color="#4a90e2" />

</TouchableOpacity>

)}

</TouchableOpacity>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>Find Friends</Text>

</View>

<View style={styles.searchContainer}>

<FontAwesome name="search" size={20} color="#999" style={styles.searchIcon} />

<TextInput

style={styles.searchInput}

placeholder="Search by name or username"

value={searchQuery}

onChangeText={handleSearch}

autoCapitalize="none"

autoFocus={true}

/>

{searchQuery.length > 0 && (

<TouchableOpacity

style={styles.clearButton}

onPress={() => setSearchQuery('')}

>

<FontAwesome name="times-circle" size={18} color="#999" />

</TouchableOpacity>

)}

</View>

{searchQuery.trim() !== '' ? (

<FlatList

data={results}

keyExtractor={item => item.id}

renderItem={renderUserItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>No users found</Text>

}

/>

) : (

<View style={styles.startSearchContainer}>

<FontAwesome name="users" size={60} color="#ddd" />

<Text style={styles.startSearchText}>

Search for users to add as friends

</Text>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchContainer: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

borderRadius: 8,

padding: 10,

marginBottom: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

searchIcon: {

marginRight: 10,

},

searchInput: {

flex: 1,

fontSize: 16,

},

clearButton: {

padding: 5,

},

userItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 12,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

avatarContainer: {

marginRight: 15,

},

avatar: {

width: 50,

height: 50,

borderRadius: 25,

},

avatarPlaceholder: {

width: 50,

height: 50,

borderRadius: 25,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

color: 'white',

fontSize: 24,

fontWeight: 'bold',

},

userInfo: {

flex: 1,

},

userName: {

fontSize: 16,

fontWeight: 'bold',

},

userUsername: {

fontSize: 14,

color: '#777',

},

addButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

borderWidth: 1,

borderColor: '#4a90e2',

},

requestSent: {

borderColor: '#27ae60',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

startSearchContainer: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

paddingBottom: 100,

},

startSearchText: {

marginTop: 20,

fontSize: 16,

color: '#999',

textAlign: 'center',

},

});

EOF

Тепер додамо файл для відображення деталей друга:

bash

Copy

mkdir -p app/\(modals\)/friends

cat > app/\(modals\)/friends/\[id\].js << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList, TouchableOpacity } from 'react-native';

import { useLocalSearchParams, Link, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані для друзів

const dummyFriends = {

'1': {

id: '1',

name: 'Anna Smith',

username: 'anna\_smith',

avatar: null,

wishlists: [

{ id: '1', name: 'Birthday Ideas', itemCount: 7 },

{ id: '2', name: 'Books to Read', itemCount: 12 },

]

},

'2': {

id: '2',

name: 'Mike Johnson',

username: 'mike\_j',

avatar: null,

wishlists: [

{ id: '1', name: 'Gadgets', itemCount: 5 },

{ id: '2', name: 'Outdoor Gear', itemCount: 3 },

]

},

'3': {

id: '3',

name: 'Sarah Williams',

username: 'sarah\_w',

avatar: null,

wishlists: [

{ id: '1', name: 'Kitchen Stuff', itemCount: 9 },

{ id: '2', name: 'Travel Wishlist', itemCount: 15 },

]

},

};

export default function FriendDetailsScreen() {

const { id } = useLocalSearchParams();

const friend = dummyFriends[id] || { name: 'Unknown', wishlists: [] };

const renderWishlistItem = ({ item }) => (

<Link href={{

pathname: "/(modals)/friends/wishlist",

params: { friendId: friend.id, wishlistId: item.id }

}} asChild>

<TouchableOpacity style={styles.wishlistItem}>

<Text style={styles.wishlistName}>{item.name}</Text>

<Text style={styles.wishlistCount}>{item.itemCount} items</Text>

<FontAwesome name="chevron-right" size={16} color="#999" />

</TouchableOpacity>

</Link>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#fff" />

</TouchableOpacity>

<View style={styles.avatarContainer}>

<View style={styles.avatarPlaceholder}>

<Text style={styles.avatarText}>{friend.name.charAt(0)}</Text>

</View>

</View>

<Text style={styles.friendName}>{friend.name}</Text>

<Text style={styles.friendUsername}>@{friend.username}</Text>

</View>

<View style={styles.wishlistsContainer}>

<Text style={styles.sectionTitle}>{friend.name}'s Wishlists</Text>

<FlatList

data={friend.wishlists}

keyExtractor={item => item.id}

renderItem={renderWishlistItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>This friend has no wishlists</Text>

}

/>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

paddingTop: 60,

position: 'relative',

},

backButton: {

position: 'absolute',

left: 20,

top: 60,

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'rgba(255,255,255,0.3)',

justifyContent: 'center',

alignItems: 'center',

},

avatarContainer: {

marginBottom: 10,

},

avatarPlaceholder: {

width: 80,

height: 80,

borderRadius: 40,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

avatarText: {

fontSize: 32,

fontWeight: 'bold',

color: '#4a90e2',

},

friendName: {

color: '#fff',

fontSize: 22,

fontWeight: 'bold',

},

friendUsername: {

color: '#e6e6e6',

fontSize: 16,

marginTop: 4,

},

wishlistsContainer: {

padding: 20,

flex: 1,

},

sectionTitle: {

fontSize: 18,

fontWeight: 'bold',

marginBottom: 15,

},

wishlistItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

wishlistName: {

fontSize: 16,

fontWeight: 'bold',

flex: 1,

},

wishlistCount: {

color: '#666',

marginRight: 10,

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Додамо файл для перегляду вішліста друга:

bash

Copy

cat > app/\(modals\)/friends/wishlist.js << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList, TouchableOpacity } from 'react-native';

import { useLocalSearchParams, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані для вішлістів друзів

const dummyFriendsWishlists = {

'1': {

'1': {

name: 'Birthday Ideas',

items: [

{ id: '1', name: 'Bluetooth Speaker', price: 79, priority: 'High' },

{ id: '2', name: 'Art Supplies', price: 45, priority: 'Medium' },

{ id: '3', name: 'Cookbook', price: 25, priority: 'Low' },

]

},

'2': {

name: 'Books to Read',

items: [

{ id: '1', name: 'The Alchemist', price: 15, priority: 'Medium' },

{ id: '2', name: 'Atomic Habits', price: 18, priority: 'High' },

]

}

},

'2': {

'1': {

name: 'Gadgets',

items: [

{ id: '1', name: 'Wireless Earbuds', price: 129, priority: 'High' },

{ id: '2', name: 'Smart Watch', price: 249, priority: 'Medium' },

]

}

},

'3': {

'1': {

name: 'Kitchen Stuff',

items: [

{ id: '1', name: 'Stand Mixer', price: 299, priority: 'High' },

{ id: '2', name: 'Chef Knife Set', price: 150, priority: 'Medium' },

]

}

}

};

// Дані про друзів

const dummyFriends = {

'1': { name: 'Anna', username: 'anna\_smith' },

'2': { name: 'Mike', username: 'mike\_j' },

'3': { name: 'Sarah', username: 'sarah\_w' },

};

export default function FriendWishlistScreen() {

const { friendId, wishlistId } = useLocalSearchParams();

const friend = dummyFriends[friendId] || { name: 'Unknown' };

const wishlist = dummyFriendsWishlists[friendId]?.[wishlistId] || { name: 'Unknown List', items: [] };

const renderItem = ({ item }) => (

<View style={styles.item}>

<View style={styles.itemHeader}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

<View style={styles.itemPriceContainer}>

<Text style={styles.itemPrice}>${item.price}</Text>

<TouchableButtonGroup />

</View>

</View>

);

const TouchableButtonGroup = () => (

<View style={styles.buttonGroup}>

<TouchableOpacity style={[styles.button, styles.giftButton]}>

<FontAwesome name="gift" size={16} color="#fff" />

<Text style={styles.buttonText}>Gift</Text>

</TouchableOpacity>

<TouchableOpacity style={[styles.button, styles.wishlistButton]}>

<FontAwesome name="plus" size={16} color="#fff" />

<Text style={styles.buttonText}>Add to My List</Text>

</TouchableOpacity>

</View>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>{friend.name}'s {wishlist.name}</Text>

</View>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={renderItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>This wishlist is empty</Text>

}

/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemHeader: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 8,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

flex: 1,

},

itemPriority: {

fontSize: 14,

fontWeight: '500',

marginLeft: 8,

},

itemPriceContainer: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

},

itemPrice: {

fontSize: 16,

color: '#666',

},

buttonGroup: {

flexDirection: 'row',

},

button: {

flexDirection: 'row',

alignItems: 'center',

paddingVertical: 6,

paddingHorizontal: 12,

borderRadius: 4,

marginLeft: 8,

},

giftButton: {

backgroundColor: '#e74c3c',

},

wishlistButton: {

backgroundColor: '#4a90e2',

},

buttonText: {

color: 'white',

fontSize: 12,

fontWeight: 'bold',

marginLeft: 4,

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

});

EOF

Тепер додамо файл для перегляду вішліста користувача:

bash

Copy

mkdir -p app/\(others\)/wishlist

cat > app/\(others\)/wishlist/\[id\].js << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList, TouchableOpacity } from 'react-native';

import { useLocalSearchParams, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані для вішлістів

const dummyWishlists = {

'1': {

name: 'Birthday Wishlist',

items: [

{ id: '1', name: 'iPhone 14', price: 999, priority: 'High' },

{ id: '2', name: 'MacBook Pro', price: 1999, priority: 'Medium' },

{ id: '3', name: 'AirPods Pro', price: 249, priority: 'Low' },

]

},

'2': {

name: 'Christmas Ideas',

items: [

{ id: '1', name: 'Nintendo Switch', price: 299, priority: 'Medium' },

{ id: '2', name: 'Kindle Paperwhite', price: 139, priority: 'Low' },

]

},

'3': {

name: 'Tech Gadgets',

items: [

{ id: '1', name: 'Drone', price: 799, priority: 'Medium' },

{ id: '2', name: 'GoPro', price: 349, priority: 'Low' },

{ id: '3', name: 'Smart Watch', price: 399, priority: 'High' },

]

}

};

export default function WishlistDetailScreen() {

const { id } = useLocalSearchParams();

const wishlist = dummyWishlists[id] || { name: 'Unknown Wishlist', items: [] };

const renderItem = ({ item }) => (

<View style={styles.item}>

<View style={styles.itemHeader}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

<Text style={styles.itemPrice}>${item.price}</Text>

</View>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>{wishlist.name}</Text>

<TouchableOpacity style={styles.editButton}>

<FontAwesome name="pencil" size={20} color="#4a90e2" />

</TouchableOpacity>

</View>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={renderItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>This wishlist is empty</Text>

}

/>

<TouchableOpacity style={styles.addButton}>

<Text style={styles.addButtonText}>+ Add New Item</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

flex: 1,

},

editButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemHeader: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 8,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

flex: 1,

},

itemPriority: {

fontSize: 14,

fontWeight: '500',

marginLeft: 8,

},

itemPrice: {

fontSize: 16,

color: '#666',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

Нарешті, створимо кореневий файл index.js для перенаправлення на табовий інтерфейс:

bash

Copy

cat > app/index.js << 'EOF'

import { Redirect } from 'expo-router';

export default function Index() {

return <Redirect href="/(tabs)" />;

}

EOF

Тепер перезапустіть додаток з очищенням кешу:

bash

Copy

npx expo start --clear

Тепер ваш додаток повинен відображати:

Нижню панель з 3 вкладками (Wishlists, Profile, Settings)

Пошук друзів, доступний через кнопку в правому верхньому куті головного екрану

Екрани друзів та їхніх вішлістів, доступні через профіль

Деталі вішліста користувача

Надіюсь, цей підхід вирішить проблеми з маршрутизацією, які ми мали раніше!

Retry

V

ідеально

Edit

Чудово, радий, що ми успішно вирішили проблему! Тепер у вас є повноцінний прототип додатку Wishlist з такими ключовими функціями:

Управління списками бажань:

Перегляд власних списків бажань

Перегляд деталей списків з елементами

Можливість додавати нові списки та елементи

Соціальні можливості:

Пошук друзів серед користувачів

Відправка запитів дружби

Перегляд профілів друзів

Перегляд списків бажань друзів

Можливість додавати елементи зі списків друзів у свої списки або відмітити їх як подарунки

Управління профілем:

Інформація про користувача

Перегляд друзів

Управління запитами дружби

Налаштування:

Основні налаштування профілю, повідомлень тощо

Ця структура є масштабованою і дозволяє легко додавати нові функції в майбутньому. Для подальшого розвитку додатку ви можете:

Підключити базу даних або API для зберігання даних

Реалізувати авторизацію користувачів

Додати відстеження цін товарів

Розширити соціальні функції (коментарі, спільні списки тощо)

Додати сповіщення про знижки, події тощо

Чи є щось ще, що ви хотіли б додати або змінити в поточній версії додатку?

Retry

V

давай продовжимо розробку відповідно до плану

Edit

Звичайно, давайте продовжимо розробку додатку відповідно до плану. Наступним логічним кроком буде реалізація функціоналу додавання нових списків бажань та елементів до них.

Ось план для подальшої розробки:

1. Реалізація додавання нових списків бажань

Створимо екран для додавання нового списку бажань:

bash

Copy

mkdir -p app/\(modals\)/wishlist

cat > app/\(modals\)/wishlist/create.js << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TextInput, TouchableOpacity, Alert } from 'react-native';

import { router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function CreateWishlistScreen() {

const [name, setName] = useState('');

const [description, setDescription] = useState('');

const [isPrivate, setIsPrivate] = useState(false);

const handleCreate = () => {

if (!name.trim()) {

Alert.alert('Error', 'Please enter a name for your wishlist');

return;

}

// В реальному додатку тут було б збереження через API або в локальне сховище

Alert.alert('Success', 'Wishlist created successfully', [

{

text: 'OK',

onPress: () => router.back()

}

]);

};

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>Create New Wishlist</Text>

</View>

<View style={styles.form}>

<Text style={styles.label}>Name</Text>

<TextInput

style={styles.input}

value={name}

onChangeText={setName}

placeholder="Enter wishlist name"

/>

<Text style={styles.label}>Description (optional)</Text>

<TextInput

style={[styles.input, styles.textarea]}

value={description}

onChangeText={setDescription}

placeholder="Enter description"

multiline

numberOfLines={4}

textAlignVertical="top"

/>

<TouchableOpacity

style={styles.privacyOption}

onPress={() => setIsPrivate(!isPrivate)}

>

<View style={styles.checkboxContainer}>

<View style={[styles.checkbox, isPrivate && styles.checkboxActive]}>

{isPrivate && <FontAwesome name="check" size={14} color="#fff" />}

</View>

</View>

<Text style={styles.privacyText}>Make this wishlist private</Text>

</TouchableOpacity>

<TouchableOpacity

style={styles.createButton}

onPress={handleCreate}

>

<Text style={styles.createButtonText}>Create Wishlist</Text>

</TouchableOpacity>

</View>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 24,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

form: {

flex: 1,

},

label: {

fontSize: 16,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

input: {

backgroundColor: 'white',

borderRadius: 8,

padding: 12,

marginBottom: 16,

fontSize: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

textarea: {

height: 100,

paddingTop: 12,

},

privacyOption: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 24,

},

checkboxContainer: {

marginRight: 12,

},

checkbox: {

width: 24,

height: 24,

borderRadius: 4,

borderWidth: 2,

borderColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

},

checkboxActive: {

backgroundColor: '#4a90e2',

},

privacyText: {

fontSize: 16,

color: '#333',

},

createButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

marginTop: 'auto',

marginBottom: 24,

},

createButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

2. Реалізація додавання нового елемента в список бажань

Створимо екран для додавання нового елемента:

bash

Copy

cat > app/\(modals\)/wishlist/add-item.js << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TextInput, TouchableOpacity, Alert, ScrollView } from 'react-native';

import { router, useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

const PRIORITIES = ['Low', 'Medium', 'High'];

export default function AddItemScreen() {

const { wishlistId } = useLocalSearchParams();

const [name, setName] = useState('');

const [price, setPrice] = useState('');

const [link, setLink] = useState('');

const [notes, setNotes] = useState('');

const [priority, setPriority] = useState('Medium');

const handleAddItem = () => {

if (!name.trim()) {

Alert.alert('Error', 'Please enter a name for the item');

return;

}

if (price && isNaN(parseFloat(price))) {

Alert.alert('Error', 'Please enter a valid price');

return;

}

// В реальному додатку тут було б збереження через API або в локальне сховище

Alert.alert('Success', 'Item added successfully', [

{

text: 'OK',

onPress: () => router.back()

}

]);

};

return (

<ScrollView style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>Add New Item</Text>

</View>

<View style={styles.form}>

<Text style={styles.label}>Item Name \*</Text>

<TextInput

style={styles.input}

value={name}

onChangeText={setName}

placeholder="Enter item name"

/>

<Text style={styles.label}>Price (optional)</Text>

<TextInput

style={styles.input}

value={price}

onChangeText={setPrice}

placeholder="Enter price"

keyboardType="numeric"

/>

<Text style={styles.label}>Link (optional)</Text>

<TextInput

style={styles.input}

value={link}

onChangeText={setLink}

placeholder="Enter product link"

keyboardType="url"

autoCapitalize="none"

/>

<Text style={styles.label}>Priority</Text>

<View style={styles.priorityContainer}>

{PRIORITIES.map((p) => (

<TouchableOpacity

key={p}

style={[

styles.priorityButton,

priority === p && styles.priorityButtonActive,

priority === p && { backgroundColor: p === 'High' ? '#e74c3c' : p === 'Medium' ? '#f39c12' : '#27ae60' }

]}

onPress={() => setPriority(p)}

>

<Text

style={[

styles.priorityButtonText,

priority === p && styles.priorityButtonTextActive

]}

>

{p}

</Text>

</TouchableOpacity>

))}

</View>

<Text style={styles.label}>Notes (optional)</Text>

<TextInput

style={[styles.input, styles.textarea]}

value={notes}

onChangeText={setNotes}

placeholder="Enter notes"

multiline

numberOfLines={4}

textAlignVertical="top"

/>

<TouchableOpacity

style={styles.addButton}

onPress={handleAddItem}

>

<Text style={styles.addButtonText}>Add Item</Text>

</TouchableOpacity>

</View>

</ScrollView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 24,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

form: {

flex: 1,

paddingBottom: 40,

},

label: {

fontSize: 16,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

input: {

backgroundColor: 'white',

borderRadius: 8,

padding: 12,

marginBottom: 16,

fontSize: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

textarea: {

height: 100,

paddingTop: 12,

},

priorityContainer: {

flexDirection: 'row',

marginBottom: 16,

},

priorityButton: {

flex: 1,

paddingVertical: 10,

alignItems: 'center',

borderWidth: 1,

borderColor: '#ddd',

marginRight: 8,

borderRadius: 4,

},

priorityButtonActive: {

borderColor: 'transparent',

},

priorityButtonText: {

color: '#666',

},

priorityButtonTextActive: {

color: 'white',

fontWeight: 'bold',

},

addButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

marginTop: 24,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

3. Оновимо головний екран вішлістів, щоб використовувати нові функції

bash

Copy

cat > app/\(tabs\)/index.js << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList } from 'react-native';

import { Link, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані

const dummyLists = [

{ id: '1', name: 'Birthday Wishlist', itemCount: 5 },

{ id: '2', name: 'Christmas Ideas', itemCount: 8 },

{ id: '3', name: 'Tech Gadgets', itemCount: 12 },

];

export default function WishlistsScreen() {

const handleSearch = () => {

router.push('/(modals)/search');

};

const handleAddWishlist = () => {

router.push('/(modals)/wishlist/create');

};

return (

<View style={styles.container}>

<View style={styles.header}>

<Text style={styles.title}>My Wishlists</Text>

<TouchableOpacity

style={styles.searchButton}

onPress={handleSearch}

>

<FontAwesome name="search" size={20} color="#4a90e2" />

</TouchableOpacity>

</View>

<FlatList

data={dummyLists}

keyExtractor={item => item.id}

renderItem={({ item }) => (

<Link href={`/(others)/wishlist/${item.id}`} asChild>

<TouchableOpacity style={styles.item}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={styles.itemCount}>{item.itemCount} items</Text>

</TouchableOpacity>

</Link>

)}

/>

<TouchableOpacity

style={styles.addButton}

onPress={handleAddWishlist}

>

<Text style={styles.addButtonText}>+ Add New Wishlist</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

},

searchButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

},

itemCount: {

color: '#666',

marginTop: 4,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

4. Оновимо екран деталей вішліста, щоб використовувати нову функцію додавання елементів

bash

Copy

cat > app/\(others\)/wishlist/\[id\].js << 'EOF'

import React from 'react';

import { View, Text, StyleSheet, FlatList, TouchableOpacity } from 'react-native';

import { useLocalSearchParams, router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

// Тестові дані для вішлістів

const dummyWishlists = {

'1': {

name: 'Birthday Wishlist',

items: [

{ id: '1', name: 'iPhone 14', price: 999, priority: 'High' },

{ id: '2', name: 'MacBook Pro', price: 1999, priority: 'Medium' },

{ id: '3', name: 'AirPods Pro', price: 249, priority: 'Low' },

]

},

'2': {

name: 'Christmas Ideas',

items: [

{ id: '1', name: 'Nintendo Switch', price: 299, priority: 'Medium' },

{ id: '2', name: 'Kindle Paperwhite', price: 139, priority: 'Low' },

]

},

'3': {

name: 'Tech Gadgets',

items: [

{ id: '1', name: 'Drone', price: 799, priority: 'Medium' },

{ id: '2', name: 'GoPro', price: 349, priority: 'Low' },

{ id: '3', name: 'Smart Watch', price: 399, priority: 'High' },

]

}

};

export default function WishlistDetailScreen() {

const { id } = useLocalSearchParams();

const wishlist = dummyWishlists[id] || { name: 'Unknown Wishlist', items: [] };

const handleAddItem = () => {

router.push({

pathname: '/(modals)/wishlist/add-item',

params: { wishlistId: id }

});

};

const renderItem = ({ item }) => (

<View style={styles.item}>

<View style={styles.itemHeader}>

<Text style={styles.itemName}>{item.name}</Text>

<Text style={[

styles.itemPriority,

{ color: item.priority === 'High' ? '#e74c3c' : item.priority === 'Medium' ? '#f39c12' : '#27ae60' }

]}>

{item.priority}

</Text>

</View>

<Text style={styles.itemPrice}>${item.price}</Text>

</View>

);

return (

<View style={styles.container}>

<View style={styles.header}>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<Text style={styles.title}>{wishlist.name}</Text>

<TouchableOpacity style={styles.editButton}>

<FontAwesome name="pencil" size={20} color="#4a90e2" />

</TouchableOpacity>

</View>

<FlatList

data={wishlist.items}

keyExtractor={item => item.id}

renderItem={renderItem}

ListEmptyComponent={

<Text style={styles.emptyMessage}>This wishlist is empty</Text>

}

/>

<TouchableOpacity

style={styles.addButton}

onPress={handleAddItem}

>

<Text style={styles.addButtonText}>+ Add New Item</Text>

</TouchableOpacity>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

padding: 16,

backgroundColor: '#f8f8f8',

},

header: {

flexDirection: 'row',

alignItems: 'center',

marginBottom: 16,

marginTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginRight: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

title: {

fontSize: 24,

fontWeight: 'bold',

flex: 1,

},

editButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

item: {

backgroundColor: 'white',

padding: 16,

borderRadius: 8,

marginBottom: 12,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

itemHeader: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginBottom: 8,

},

itemName: {

fontSize: 18,

fontWeight: 'bold',

flex: 1,

},

itemPriority: {

fontSize: 14,

fontWeight: '500',

marginLeft: 8,

},

itemPrice: {

fontSize: 16,

color: '#666',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

addButton: {

backgroundColor: '#4a90e2',

padding: 16,

borderRadius: 8,

alignItems: 'center',

marginTop: 16,

},

addButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

Ці файли додадуть важливий функціонал до нашого додатку:

Можливість створювати нові списки бажань

Можливість додавати нові елементи в існуючі списки

Більше інформації про елементи (посилання, нотатки, пріоритети)

Після додавання цих файлів перезапустіть додаток:

bash

Copy

npx expo start --clear

Це дозволить користувачам створювати та заповнювати свої списки бажань, що є одним з основних функціоналів нашого додатку.

Наступними кроками розробки могли б бути:

Реалізація локального зберігання даних (AsyncStorage)

Додавання можливості редагування та видалення вішлістів та елементів

Реалізація авторизації користувачів

Інтеграція з API для відстеження цін товарів

Що з цього ви хотіли б реалізувати наступним?

Retry

V

Я хочу зробити так, щоб користувачі могли реєструватись, та входити в свій акаунт якщо вже зареєстровані, реєстрація має проходити за допомогою відправки коду підтвердження на емейл, тобто флоу таке, користувач відкриває додаток вперше, ми показуємо йому вікно реєстрації, де йому потрібно ввести свій емейл, він вводить емейл, і нажимає на кнопку "продовжити" після цього ми показуємо йому екран із полем де потрібно ввести код підтвердження який складається із 4 х цифр, ми відправляємо йому цей код на пошту, він бере цей код з пошти і вводить у нашому додатку, натискає кнопку продовжити і проходить до свого створеного профілю

Edit

Давайте реалізуємо систему автентифікації з підтвердженням електронної пошти. Потрібно створити кілька екранів: для входу, реєстрації та підтвердження коду.

Почнемо зі створення структури:

bash

Copy

# Створюємо папку для екранів автентифікації

mkdir -p app/\(auth\)

1. Створимо екран привітання/входу:

bash

Copy

cat > app/\(auth\)/welcome.js << 'EOF'

import React, { useState } from 'react';

import { View, Text, StyleSheet, TextInput, TouchableOpacity, Image, KeyboardAvoidingView, Platform } from 'react-native';

import { router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function WelcomeScreen() {

const [email, setEmail] = useState('');

const [isExistingUser, setIsExistingUser] = useState(false);

const handleContinue = () => {

if (!email || !email.includes('@')) {

alert('Please enter a valid email address');

return;

}

// В реальному додатку тут буде перевірка, чи існує користувач

// Для демонстрації ми використаємо змінну стану

if (isExistingUser) {

// Користувач вже зареєстрований - відправляємо код підтвердження для входу

router.push({

pathname: '/(auth)/verify',

params: { email, isLogin: 'true' }

});

} else {

// Новий користувач - відправляємо код підтвердження для реєстрації

router.push({

pathname: '/(auth)/verify',

params: { email, isLogin: 'false' }

});

}

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<View style={styles.logoContainer}>

<View style={styles.logo}>

<FontAwesome name="gift" size={60} color="#4a90e2" />

</View>

<Text style={styles.appName}>Wishlist</Text>

<Text style={styles.tagline}>Your personal wishlist manager</Text>

</View>

<View style={styles.formContainer}>

<Text style={styles.welcomeText}>

{isExistingUser ? 'Welcome back!' : 'Create an account'}

</Text>

<Text style={styles.label}>Email</Text>

<TextInput

style={styles.input}

value={email}

onChangeText={setEmail}

placeholder="Enter your email"

keyboardType="email-address"

autoCapitalize="none"

autoCorrect={false}

/>

<TouchableOpacity

style={styles.continueButton}

onPress={handleContinue}

>

<Text style={styles.continueButtonText}>Continue</Text>

</TouchableOpacity>

<TouchableOpacity

style={styles.switchButton}

onPress={() => setIsExistingUser(!isExistingUser)}

>

<Text style={styles.switchButtonText}>

{isExistingUser

? "Don't have an account? Sign up"

: "Already have an account? Log in"}

</Text>

</TouchableOpacity>

</View>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

logoContainer: {

alignItems: 'center',

marginTop: 80,

marginBottom: 40,

},

logo: {

width: 100,

height: 100,

borderRadius: 20,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 2 },

shadowOpacity: 0.2,

shadowRadius: 4,

elevation: 4,

marginBottom: 16,

},

appName: {

fontSize: 32,

fontWeight: 'bold',

color: '#333',

},

tagline: {

fontSize: 16,

color: '#666',

marginTop: 8,

},

formContainer: {

flex: 1,

paddingHorizontal: 24,

},

welcomeText: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 24,

color: '#333',

},

label: {

fontSize: 16,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

input: {

backgroundColor: 'white',

borderRadius: 8,

padding: 16,

fontSize: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

marginBottom: 24,

},

continueButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

},

continueButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

switchButton: {

marginTop: 24,

alignItems: 'center',

},

switchButtonText: {

color: '#4a90e2',

fontSize: 16,

},

});

EOF

2. Створимо екран верифікації коду:

bash

Copy

cat > app/\(auth\)/verify.js << 'EOF'

import React, { useState, useRef, useEffect } from 'react';

import {

View,

Text,

StyleSheet,

TextInput,

TouchableOpacity,

KeyboardAvoidingView,

Platform,

Alert

} from 'react-native';

import { router, useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function VerifyScreen() {

const { email, isLogin } = useLocalSearchParams();

const [code, setCode] = useState(['', '', '', '']);

const inputRefs = useRef([]);

const [timer, setTimer] = useState(60);

const [canResend, setCanResend] = useState(false);

useEffect(() => {

// В реальному додатку тут буде відправка коду на пошту

Alert.alert('Code Sent', `A verification code has been sent to ${email}`);

// Запускаємо таймер для можливості повторної відправки

const interval = setInterval(() => {

setTimer(prevTimer => {

if (prevTimer <= 1) {

clearInterval(interval);

setCanResend(true);

return 0;

}

return prevTimer - 1;

});

}, 1000);

return () => clearInterval(interval);

}, []);

const handleCodeChange = (text, index) => {

const newCode = [...code];

newCode[index] = text;

setCode(newCode);

// Автоматичний перехід до наступного поля

if (text.length === 1 && index < 3) {

inputRefs.current[index + 1].focus();

}

};

const handleVerify = () => {

const fullCode = code.join('');

if (fullCode.length !== 4) {

Alert.alert('Error', 'Please enter the 4-digit code');

return;

}

// В реальному додатку тут буде перевірка коду через API

// Для демонстрації використаємо тестовий код "1234"

if (fullCode === '1234') {

if (isLogin === 'true') {

// Користувач увійшов у систему

router.replace('/(tabs)');

} else {

// Користувач зареєструвався, перейдемо до налаштування профілю

router.push({

pathname: '/(auth)/setup-profile',

params: { email }

});

}

} else {

Alert.alert('Error', 'Invalid verification code');

}

};

const handleResendCode = () => {

if (!canResend) return;

// Скидаємо таймер і відправляємо код знову

setTimer(60);

setCanResend(false);

// В реальному додатку тут буде повторна відправка коду

Alert.alert('Code Resent', `A new verification code has been sent to ${email}`);

// Запускаємо таймер знову

const interval = setInterval(() => {

setTimer(prevTimer => {

if (prevTimer <= 1) {

clearInterval(interval);

setCanResend(true);

return 0;

}

return prevTimer - 1;

});

}, 1000);

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<View style={styles.content}>

<Text style={styles.title}>Verification Code</Text>

<Text style={styles.subtitle}>

Enter the 4-digit code sent to {email}

</Text>

<View style={styles.codeContainer}>

{code.map((digit, index) => (

<TextInput

key={index}

ref={el => inputRefs.current[index] = el}

style={styles.codeInput}

value={digit}

onChangeText={text => handleCodeChange(text, index)}

keyboardType="number-pad"

maxLength={1}

/>

))}

</View>

<TouchableOpacity

style={styles.verifyButton}

onPress={handleVerify}

>

<Text style={styles.verifyButtonText}>Verify</Text>

</TouchableOpacity>

<View style={styles.resendContainer}>

<Text style={styles.resendText}>Didn't receive the code? </Text>

<TouchableOpacity

onPress={handleResendCode}

disabled={!canResend}

>

<Text style={[

styles.resendButton,

!canResend && styles.resendButtonDisabled

]}>

{canResend ? 'Resend' : `Resend (${timer}s)`}

</Text>

</TouchableOpacity>

</View>

</View>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

paddingTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginLeft: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

content: {

flex: 1,

paddingHorizontal: 24,

paddingTop: 40,

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

color: '#333',

},

subtitle: {

fontSize: 16,

color: '#666',

marginBottom: 40,

},

codeContainer: {

flexDirection: 'row',

justifyContent: 'space-between',

marginBottom: 40,

},

codeInput: {

width: 70,

height: 70,

borderRadius: 8,

backgroundColor: 'white',

fontSize: 24,

fontWeight: 'bold',

textAlign: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

verifyButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

},

verifyButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

resendContainer: {

flexDirection: 'row',

justifyContent: 'center',

marginTop: 24,

},

resendText: {

color: '#666',

fontSize: 16,

},

resendButton: {

color: '#4a90e2',

fontSize: 16,

fontWeight: 'bold',

},

resendButtonDisabled: {

color: '#999',

},

});

EOF

3. Створимо екран налаштування профілю для нових користувачів:

bash

Copy

cat > app/\(auth\)/setup-profile.js << 'EOF'

import React, { useState } from 'react';

import {

View,

Text,

StyleSheet,

TextInput,

TouchableOpacity,

KeyboardAvoidingView,

Platform,

Image,

ScrollView

} from 'react-native';

import { router, useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

export default function SetupProfileScreen() {

const { email } = useLocalSearchParams();

const [name, setName] = useState('');

const [username, setUsername] = useState('');

const handleFinish = () => {

if (!name.trim()) {

alert('Please enter your name');

return;

}

if (!username.trim()) {

alert('Please choose a username');

return;

}

// В реальному додатку тут буде збереження даних користувача

// Перенаправлення до основного додатку

router.replace('/(tabs)');

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<ScrollView contentContainerStyle={styles.scrollContent}>

<Text style={styles.title}>Set Up Your Profile</Text>

<Text style={styles.subtitle}>Let's create your profile</Text>

<View style={styles.avatarContainer}>

<View style={styles.avatar}>

<FontAwesome name="user" size={60} color="#ddd" />

<View style={styles.addPhotoButton}>

<FontAwesome name="camera" size={16} color="#fff" />

</View>

</View>

<Text style={styles.addPhotoText}>Add Photo</Text>

</View>

<View style={styles.formContainer}>

<Text style={styles.label}>Full Name</Text>

<TextInput

style={styles.input}

value={name}

onChangeText={setName}

placeholder="Enter your name"

/>

<Text style={styles.label}>Username</Text>

<TextInput

style={styles.input}

value={username}

onChangeText={setUsername}

placeholder="Choose a username"

autoCapitalize="none"

/>

<Text style={styles.label}>Email</Text>

<TextInput

style={[styles.input, styles.disabledInput]}

value={email}

editable={false}

/>

<TouchableOpacity

style={styles.finishButton}

onPress={handleFinish}

>

<Text style={styles.finishButtonText}>Finish Setup</Text>

</TouchableOpacity>

</View>

</ScrollView>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

scrollContent: {

padding: 24,

paddingTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

subtitle: {

fontSize: 16,

color: '#666',

marginBottom: 24,

},

avatarContainer: {

alignItems: 'center',

marginBottom: 32,

},

avatar: {

width: 120,

height: 120,

borderRadius: 60,

backgroundColor: '#f0f0f0',

justifyContent: 'center',

alignItems: 'center',

position: 'relative',

marginBottom: 8,

},

addPhotoButton: {

position: 'absolute',

bottom: 0,

right: 0,

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

borderWidth: 3,

borderColor: '#f8f8f8',

},

addPhotoText: {

fontSize: 16,

color: '#4a90e2',

},

formContainer: {

marginBottom: 30,

},

label: {

fontSize: 16,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

input: {

backgroundColor: 'white',

borderRadius: 8,

padding: 16,

fontSize: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

marginBottom: 24,

},

disabledInput: {

backgroundColor: '#f5f5f5',

color: '#666',

},

finishButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

marginTop: 16,

},

finishButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

4. Модифікуємо основний файл для перенаправлення нових користувачів:

bash

Copy

cat > app/index.js << 'EOF'

import { Redirect } from 'expo-router';

// В реальному додатку тут була б перевірка, чи авторизований користувач

const isLoggedIn = false; // Змінити на false для тестування автентифікації

export default function Index() {

// Якщо користувач авторизований, перенаправляємо на головний екран

if (isLoggedIn) {

return <Redirect href="/(tabs)" />;

}

// Інакше перенаправляємо на екран входу/реєстрації

return <Redirect href="/(auth)/welcome" />;

}

EOF

5. Оновимо \_layout.js, щоб забезпечити правильну роботу автентифікації:

bash

Copy

cat > app/\_layout.js << 'EOF'

import React from 'react';

import { Stack } from 'expo-router';

export default function RootLayout() {

return (

<Stack screenOptions={{ headerShown: false }}>

<Stack.Screen name="index" />

<Stack.Screen name="(tabs)" />

<Stack.Screen name="(modals)" />

<Stack.Screen name="(others)" />

<Stack.Screen name="(auth)" />

</Stack>

);

}

EOF

Перезапустіть додаток, щоб побачити нові зміни:

bash

Copy

npx expo start --clear

Тепер у вашому додатку є система автентифікації з:

Екраном входу та реєстрації

Верифікацією через код, надісланий на електронну пошту (імітація)

Налаштуванням профілю для нових користувачів

Для перевірки коду використовуйте "1234".

Після завершення реєстрації або входу користувач перенаправляється на головний екран додатку. У реальному додатку дані автентифікації потрібно було б зберігати (наприклад, через AsyncStorage або іншу систему зберігання) і підключатися до бекенд-серверу для перевірки користувача та надсилання кодів підтвердження.

Retry

V

тепер потрібно зробити так, щоб користувачі які реєструються зберігались в додатку, і після розлогіну, вони могли увійти за своїм емейлом, і ми ідентифікували користувача і показали йому його профіль, також видалити весь демо контент і щоб я зареєстрував акаунт і всі дані які я вводив в реєстрації показувались у мене в профілі

Edit

Зрозуміло. Тепер нам потрібно реалізувати локальне збереження даних користувача, щоб забезпечити збереження профілю після перезапуску додатку і можливість входу існуючого користувача. Давайте використаємо AsyncStorage для цього.

Спочатку встановимо необхідний пакет:

bash

Copy

npx expo install @react-native-async-storage/async-storage

Тепер створимо сервіс для управління аутентифікацією та даними користувачів:

bash

Copy

# Створюємо папку для сервісів

mkdir -p app/services

# Створюємо сервіс аутентифікації

cat > app/services/authService.js << 'EOF'

import AsyncStorage from '@react-native-async-storage/async-storage';

// Ключі для збереження даних

const USERS\_KEY = 'wishlist\_app\_users';

const CURRENT\_USER\_KEY = 'wishlist\_app\_current\_user';

// Функція для генерації коду підтвердження

export const generateVerificationCode = () => {

return Math.floor(1000 + Math.random() \* 9000).toString();

};

// Функція для збереження коду для конкретного email (в реальному додатку це було б на сервері)

export const saveVerificationCode = async (email, code) => {

try {

const codesJson = await AsyncStorage.getItem('verification\_codes') || '{}';

const codes = JSON.parse(codesJson);

codes[email] = {

code,

timestamp: Date.now(),

};

await AsyncStorage.setItem('verification\_codes', JSON.stringify(codes));

} catch (error) {

console.error('Error saving verification code:', error);

}

};

// Функція для перевірки коду підтвердження

export const verifyCode = async (email, code) => {

try {

const codesJson = await AsyncStorage.getItem('verification\_codes') || '{}';

const codes = JSON.parse(codesJson);

const savedCode = codes[email];

// Перевіряємо наявність коду та його актуальність (30 хвилин)

if (savedCode && savedCode.code === code &&

Date.now() - savedCode.timestamp < 30 \* 60 \* 1000) {

return true;

}

return false;

} catch (error) {

console.error('Error verifying code:', error);

return false;

}

};

// Функція для перевірки, чи існує користувач

export const userExists = async (email) => {

try {

const usersJson = await AsyncStorage.getItem(USERS\_KEY) || '{}';

const users = JSON.parse(usersJson);

return !!users[email];

} catch (error) {

console.error('Error checking user existence:', error);

return false;

}

};

// Функція для створення нового користувача

export const createUser = async (userData) => {

try {

const { email } = userData;

// Отримуємо існуючих користувачів

const usersJson = await AsyncStorage.getItem(USERS\_KEY) || '{}';

const users = JSON.parse(usersJson);

// Додаємо нового користувача

users[email] = {

...userData,

wishlists: [],

friends: [],

friendRequests: [],

registeredAt: new Date().toISOString(),

};

// Зберігаємо оновлений список користувачів

await AsyncStorage.setItem(USERS\_KEY, JSON.stringify(users));

// Встановлюємо поточного користувача

await AsyncStorage.setItem(CURRENT\_USER\_KEY, email);

return true;

} catch (error) {

console.error('Error creating user:', error);

return false;

}

};

// Функція для входу користувача

export const loginUser = async (email) => {

try {

// Перевіряємо, чи існує користувач

const exists = await userExists(email);

if (!exists) {

return false;

}

// Встановлюємо поточного користувача

await AsyncStorage.setItem(CURRENT\_USER\_KEY, email);

return true;

} catch (error) {

console.error('Error logging in:', error);

return false;

}

};

// Функція для виходу користувача

export const logoutUser = async () => {

try {

await AsyncStorage.removeItem(CURRENT\_USER\_KEY);

return true;

} catch (error) {

console.error('Error logging out:', error);

return false;

}

};

// Функція для отримання поточного користувача

export const getCurrentUser = async () => {

try {

const email = await AsyncStorage.getItem(CURRENT\_USER\_KEY);

if (!email) {

return null;

}

const usersJson = await AsyncStorage.getItem(USERS\_KEY) || '{}';

const users = JSON.parse(usersJson);

return users[email] || null;

} catch (error) {

console.error('Error getting current user:', error);

return null;

}

};

// Функція для перевірки, чи автентифікований користувач

export const isAuthenticated = async () => {

try {

const email = await AsyncStorage.getItem(CURRENT\_USER\_KEY);

return !!email;

} catch (error) {

console.error('Error checking authentication:', error);

return false;

}

};

// Функція для оновлення даних користувача

export const updateUserData = async (userData) => {

try {

const { email } = userData;

// Отримуємо існуючих користувачів

const usersJson = await AsyncStorage.getItem(USERS\_KEY) || '{}';

const users = JSON.parse(usersJson);

// Перевіряємо, чи існує користувач

if (!users[email]) {

return false;

}

// Оновлюємо дані користувача

users[email] = {

...users[email],

...userData,

};

// Зберігаємо оновлений список користувачів

await AsyncStorage.setItem(USERS\_KEY, JSON.stringify(users));

return true;

} catch (error) {

console.error('Error updating user data:', error);

return false;

}

};

EOF

Тепер створимо сервіс для управління списками бажань:

bash

Copy

cat > app/services/wishlistService.js << 'EOF'

import AsyncStorage from '@react-native-async-storage/async-storage';

import { getCurrentUser, updateUserData } from './authService';

// Функція для створення нового списку бажань

export const createWishlist = async (wishlistData) => {

try {

const user = await getCurrentUser();

if (!user) {

return false;

}

// Створюємо новий список бажань з унікальним ідентифікатором

const newWishlist = {

id: Date.now().toString(),

...wishlistData,

items: [],

createdAt: new Date().toISOString(),

};

// Додаємо новий список до списків користувача

const updatedWishlists = [...(user.wishlists || []), newWishlist];

// Оновлюємо дані користувача

await updateUserData({

...user,

wishlists: updatedWishlists,

});

return newWishlist;

} catch (error) {

console.error('Error creating wishlist:', error);

return false;

}

};

// Функція для отримання всіх списків бажань користувача

export const getUserWishlists = async () => {

try {

const user = await getCurrentUser();

if (!user) {

return [];

}

return user.wishlists || [];

} catch (error) {

console.error('Error getting wishlists:', error);

return [];

}

};

// Функція для отримання конкретного списку бажань

export const getWishlistById = async (wishlistId) => {

try {

const wishlists = await getUserWishlists();

return wishlists.find(wishlist => wishlist.id === wishlistId) || null;

} catch (error) {

console.error('Error getting wishlist by id:', error);

return null;

}

};

// Функція для додавання елемента в список бажань

export const addItemToWishlist = async (wishlistId, itemData) => {

try {

const user = await getCurrentUser();

if (!user) {

return false;

}

// Створюємо новий елемент з унікальним ідентифікатором

const newItem = {

id: Date.now().toString(),

...itemData,

addedAt: new Date().toISOString(),

};

// Знаходимо потрібний список бажань та додаємо елемент

const updatedWishlists = (user.wishlists || []).map(wishlist => {

if (wishlist.id === wishlistId) {

return {

...wishlist,

items: [...(wishlist.items || []), newItem],

};

}

return wishlist;

});

// Оновлюємо дані користувача

await updateUserData({

...user,

wishlists: updatedWishlists,

});

return newItem;

} catch (error) {

console.error('Error adding item to wishlist:', error);

return false;

}

};

// Функція для видалення елемента зі списку бажань

export const removeItemFromWishlist = async (wishlistId, itemId) => {

try {

const user = await getCurrentUser();

if (!user) {

return false;

}

// Знаходимо потрібний список бажань та видаляємо елемент

const updatedWishlists = (user.wishlists || []).map(wishlist => {

if (wishlist.id === wishlistId) {

return {

...wishlist,

items: (wishlist.items || []).filter(item => item.id !== itemId),

};

}

return wishlist;

});

// Оновлюємо дані користувача

await updateUserData({

...user,

wishlists: updatedWishlists,

});

return true;

} catch (error) {

console.error('Error removing item from wishlist:', error);

return false;

}

};

// Функція для видалення списку бажань

export const deleteWishlist = async (wishlistId) => {

try {

const user = await getCurrentUser();

if (!user) {

return false;

}

// Фільтруємо списки бажань, видаляючи потрібний

const updatedWishlists = (user.wishlists || []).filter(

wishlist => wishlist.id !== wishlistId

);

// Оновлюємо дані користувача

await updateUserData({

...user,

wishlists: updatedWishlists,

});

return true;

} catch (error) {

console.error('Error deleting wishlist:', error);

return false;

}

};

EOF

Тепер оновимо екрани автентифікації, щоб вони використовували наші сервіси:

Оновлюємо екран привітання/входу:

bash

Copy

cat > app/\(auth\)/welcome.js << 'EOF'

import React, { useState, useEffect } from 'react';

import { View, Text, StyleSheet, TextInput, TouchableOpacity, KeyboardAvoidingView, Platform, ActivityIndicator } from 'react-native';

import { router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

import { userExists, generateVerificationCode, saveVerificationCode } from '../services/authService';

export default function WelcomeScreen() {

const [email, setEmail] = useState('');

const [isExistingUser, setIsExistingUser] = useState(false);

const [isCheckingUser, setIsCheckingUser] = useState(false);

// Функція для перевірки, чи існує користувач з таким email

const checkUserExists = async (inputEmail) => {

if (!inputEmail || !inputEmail.includes('@')) return;

setIsCheckingUser(true);

const exists = await userExists(inputEmail);

setIsExistingUser(exists);

setIsCheckingUser(false);

};

// Перевіряємо, чи існує користувач, коли змінюється email

useEffect(() => {

const timeoutId = setTimeout(() => {

checkUserExists(email);

}, 500);

return () => clearTimeout(timeoutId);

}, [email]);

const handleContinue = async () => {

if (!email || !email.includes('@')) {

alert('Please enter a valid email address');

return;

}

// Генеруємо код верифікації та зберігаємо його

const code = generateVerificationCode();

await saveVerificationCode(email, code);

// В реальному додатку тут був би запит на сервер для відправки коду на пошту

alert(`Verification code: ${code}`); // Для тестування показуємо код

if (isExistingUser) {

// Користувач вже зареєстрований - відправляємо код підтвердження для входу

router.push({

pathname: '/(auth)/verify',

params: { email, isLogin: 'true' }

});

} else {

// Новий користувач - відправляємо код підтвердження для реєстрації

router.push({

pathname: '/(auth)/verify',

params: { email, isLogin: 'false' }

});

}

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<View style={styles.logoContainer}>

<View style={styles.logo}>

<FontAwesome name="gift" size={60} color="#4a90e2" />

</View>

<Text style={styles.appName}>Wishlist</Text>

<Text style={styles.tagline}>Your personal wishlist manager</Text>

</View>

<View style={styles.formContainer}>

<Text style={styles.welcomeText}>

{isExistingUser ? 'Welcome back!' : 'Create an account'}

</Text>

<Text style={styles.label}>Email</Text>

<View style={styles.inputContainer}>

<TextInput

style={styles.input}

value={email}

onChangeText={setEmail}

placeholder="Enter your email"

keyboardType="email-address"

autoCapitalize="none"

autoCorrect={false}

/>

{isCheckingUser && (

<ActivityIndicator size="small" color="#4a90e2" style={styles.indicator} />

)}

</View>

<TouchableOpacity

style={styles.continueButton}

onPress={handleContinue}

>

<Text style={styles.continueButtonText}>Continue</Text>

</TouchableOpacity>

<TouchableOpacity

style={styles.switchButton}

onPress={() => setIsExistingUser(!isExistingUser)}

>

<Text style={styles.switchButtonText}>

{isExistingUser

? "Don't have an account? Sign up"

: "Already have an account? Log in"}

</Text>

</TouchableOpacity>

</View>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

logoContainer: {

alignItems: 'center',

marginTop: 80,

marginBottom: 40,

},

logo: {

width: 100,

height: 100,

borderRadius: 20,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 2 },

shadowOpacity: 0.2,

shadowRadius: 4,

elevation: 4,

marginBottom: 16,

},

appName: {

fontSize: 32,

fontWeight: 'bold',

color: '#333',

},

tagline: {

fontSize: 16,

color: '#666',

marginTop: 8,

},

formContainer: {

flex: 1,

paddingHorizontal: 24,

},

welcomeText: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 24,

color: '#333',

},

label: {

fontSize: 16,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

inputContainer: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

borderRadius: 8,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

marginBottom: 24,

},

input: {

flex: 1,

padding: 16,

fontSize: 16,

},

indicator: {

marginRight: 16,

},

continueButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

},

continueButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

switchButton: {

marginTop: 24,

alignItems: 'center',

},

switchButtonText: {

color: '#4a90e2',

fontSize: 16,

},

});

EOF

Оновлюємо екран верифікації:

bash

Copy

cat > app/\(auth\)/verify.js << 'EOF'

import React, { useState, useRef, useEffect } from 'react';

import {

View,

Text,

StyleSheet,

TextInput,

TouchableOpacity,

KeyboardAvoidingView,

Platform,

Alert,

ActivityIndicator

} from 'react-native';

import { router, useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

import { verifyCode, generateVerificationCode, saveVerificationCode, loginUser } from '../services/authService';

export default function VerifyScreen() {

const { email, isLogin } = useLocalSearchParams();

const [code, setCode] = useState(['', '', '', '']);

const inputRefs = useRef([]);

const [timer, setTimer] = useState(60);

const [canResend, setCanResend] = useState(false);

const [isVerifying, setIsVerifying] = useState(false);

useEffect(() => {

// В реальному додатку тут буде відправка коду на пошту

Alert.alert('Code Sent', `A verification code has been sent to ${email}`);

// Запускаємо таймер для можливості повторної відправки

const interval = setInterval(() => {

setTimer(prevTimer => {

if (prevTimer <= 1) {

clearInterval(interval);

setCanResend(true);

return 0;

}

return prevTimer - 1;

});

}, 1000);

return () => clearInterval(interval);

}, []);

const handleCodeChange = (text, index) => {

// Перевіряємо, щоб був лише числовий ввід

if (text !== '' && !/^\d+$/.test(text)) {

return;

}

const newCode = [...code];

newCode[index] = text;

setCode(newCode);

// Автоматичний перехід до наступного поля

if (text.length === 1 && index < 3) {

inputRefs.current[index + 1].focus();

}

};

const handleVerify = async () => {

const fullCode = code.join('');

if (fullCode.length !== 4) {

Alert.alert('Error', 'Please enter the 4-digit code');

return;

}

setIsVerifying(true);

// Перевіряємо код

const isValid = await verifyCode(email, fullCode);

setIsVerifying(false);

if (isValid) {

if (isLogin === 'true') {

// Користувач увійшов у систему

await loginUser(email);

router.replace('/(tabs)');

} else {

// Користувач зареєструвався, перейдемо до налаштування профілю

router.push({

pathname: '/(auth)/setup-profile',

params: { email }

});

}

} else {

Alert.alert('Error', 'Invalid verification code');

}

};

const handleResendCode = async () => {

if (!canResend) return;

// Скидаємо таймер і відправляємо код знову

setTimer(60);

setCanResend(false);

// Генеруємо новий код і зберігаємо його

const newCode = generateVerificationCode();

await saveVerificationCode(email, newCode);

// В реальному додатку тут був би запит на сервер для відправки коду на пошту

Alert.alert('Code Resent', `New verification code: ${newCode}`); // Для тестування показуємо код

// Запускаємо таймер знову

const interval = setInterval(() => {

setTimer(prevTimer => {

if (prevTimer <= 1) {

clearInterval(interval);

setCanResend(true);

return 0;

}

return prevTimer - 1;

});

}, 1000);

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<View style={styles.content}>

<Text style={styles.title}>Verification Code</Text>

<Text style={styles.subtitle}>

Enter the 4-digit code sent to {email}

</Text>

<View style={styles.codeContainer}>

{code.map((digit, index) => (

<TextInput

key={index}

ref={el => inputRefs.current[index] = el}

style={styles.codeInput}

value={digit}

onChangeText={text => handleCodeChange(text, index)}

keyboardType="number-pad"

maxLength={1}

/>

))}

</View>

<TouchableOpacity

style={[styles.verifyButton, isVerifying && styles.verifyButtonDisabled]}

onPress={handleVerify}

disabled={isVerifying}

>

{isVerifying ? (

<ActivityIndicator color="#fff" size="small" />

) : (

<Text style={styles.verifyButtonText}>Verify</Text>

)}

</TouchableOpacity>

<View style={styles.resendContainer}>

<Text style={styles.resendText}>Didn't receive the code? </Text>

<TouchableOpacity

onPress={handleResendCode}

disabled={!canResend}

>

<Text style={[

styles.resendButton,

!canResend && styles.resendButtonDisabled

]}>

{canResend ? 'Resend' : `Resend (${timer}s)`}

</Text>

</TouchableOpacity>

</View>

</View>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

paddingTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginLeft: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

content: {

flex: 1,

paddingHorizontal: 24,

paddingTop: 40,

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

color: '#333',

},

subtitle: {

fontSize: 16,

color: '#666',

marginBottom: 40,

},

codeContainer: {

flexDirection: 'row',

justifyContent: 'space-between',

marginBottom: 40,

},

codeInput: {

width: 70,

height: 70,

borderRadius: 8,

backgroundColor: 'white',

fontSize: 24,

fontWeight: 'bold',

textAlign: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

verifyButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

justifyContent: 'center',

height: 56,

},

verifyButtonDisabled: {

backgroundColor: '#a0c5e8',

},

verifyButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

resendContainer: {

flexDirection: 'row',

justifyContent: 'center',

marginTop: 24,

},

resendText: {

color: '#666',

fontSize: 16,

},

resendButton: {

color: '#4a90e2',

fontSize: 16,

fontWeight: 'bold',

},

resendButtonDisabled: {

color: '#999',

},

});

EOF

Оновлюємо екран налаштування профілю:

bash

Copy

cat > app/\(auth\)/setup-profile.js << 'EOF'

import React, { useState } from 'react';

import {

View,

Text,

StyleSheet,

TextInput,

TouchableOpacity,

KeyboardAvoidingView,

Platform,

ScrollView,

Alert,

ActivityIndicator

} from 'react-native';

import { router, useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

import { createUser } from '../services/authService';

export default function SetupProfileScreen() {

const { email } = useLocalSearchParams();

const [name, setName] = useState('');

const [username, setUsername] = useState('');

const [isCreating, setIsCreating] = useState(false);

const handleFinish = async () => {

if (!name.trim()) {

Alert.alert('Error', 'Please enter your name');

return;

}

if (!username.trim()) {

Alert.alert('Error', 'Please choose a username');

return;

}

setIsCreating(true);

// Створення нового користувача

const userData = {

email,

name,

username,

avatar: null, // В майбутньому можна додати завантаження аватара

};

const success = await createUser(userData);

setIsCreating(false);

if (success) {

// Перенаправлення до основного додатку

router.replace('/(tabs)');

} else {

Alert.alert('Error', 'Failed to create account. Please try again.');

}

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<ScrollView contentContainerStyle={styles.scrollContent}>

<Text style={styles.title}>Set Up Your Profile</Text>

<Text style={styles.subtitle}>Let's create your profile</Text>

<View style={styles.avatarContainer}>

<View style={styles.avatar}>

<FontAwesome name="user" size={60} color="#ddd" />

<View style={styles.addPhotoButton}>

<FontAwesome name="camera" size={16} color="#fff" />

</View>

</View>

<Text style={styles.addPhotoText}>Add Photo</Text>

</View>

<View style={styles.formContainer}>

<Text style={styles.label}>Full Name</Text>

<TextInput

style={styles.input}

value={name}

onChangeText={setName}

placeholder="Enter your name"

/>

<Text style={styles.label}>Username</Text>

<TextInput

style={styles.input}

value={username}

onChangeText={setUsername}

placeholder="Choose a username"

autoCapitalize="none"

/>

<Text style={styles.label}>Email</Text>

<TextInput

style={[styles.input, styles.disabledInput]}

value={email}

editable={false

Retry

V

давай спочатку розгяленмо в загальному як ми можемо це зробити які є варіанти

Edit

Для реалізації системи автентифікації користувачів з локальним збереженням даних є кілька підходів:

Використання AsyncStorage:

Простий підхід для збереження даних користувача локально на пристрої

Добре підходить для прототипів та простих додатків

Дані зберігаються у вигляді пар ключ-значення

Обмеження: дані зберігаються тільки на одному пристрої

Використання Redux + AsyncStorage:

Дозволяє організувати глобальний стан додатку

Легко керувати станом автентифікації через всі екрани

Redux Persist можна використовувати для автоматичного збереження стану

Використання Context API + AsyncStorage:

Вбудована в React система управління станом

Менш складна ніж Redux, але достатня для наших потреб

Дозволяє створити AuthContext, який буде доступний з усіх частин додатку

Firebase Authentication:

Готова хмарна платформа для автентифікації

Підтримує електронну пошту/пароль, соціальні мережі, SMS

Можна інтегрувати з Firebase Firestore для збереження даних користувача

Supabase або Appwrite:

Сучасні альтернативи Firebase з відкритим кодом

Надають API для автентифікації та бази даних

Для нашого випадку, враховуючи локальний характер додатку, найкращими варіантами будуть:

Context API + AsyncStorage (найпростіший підхід):

Створимо AuthContext для керування станом автентифікації

Використаємо AsyncStorage для збереження профілю користувача

Реалізуємо функції для реєстрації, входу та виходу

Redux + AsyncStorage (розширюваний підхід):

Створимо redux-слайси для автентифікації та профілю

Використаємо Redux Persist для збереження стану

Додамо middleware для керування автентифікацією

Який з цих підходів ви вважаєте найбільш підходящим для вашого додатку?

Retry

V

що значить локальне?

Edit

"Локальне" в контексті збереження даних означає, що інформація зберігається безпосередньо на пристрої користувача, а не на віддаленому сервері або в хмарі.

Коли я кажу про "локальне збереження даних користувача", це означає:

Дані зберігаються в пам'яті самого пристрою (смартфона)

Інформація доступна навіть коли пристрій не підключений до інтернету

Дані залишаються на пристрої після закриття додатку і його повторного відкриття

Ця інформація не синхронізується автоматично між різними пристроями користувача

Переваги локального збереження:

Працює без підключення до інтернету

Швидкий доступ до даних

Не потребує серверної інфраструктури

Недоліки:

Дані не синхронізуються між пристроями (якщо користувач має кілька пристроїв)

При видаленні додатку або зміні пристрою дані можуть бути втрачені

Обмежений об'єм пам'яті для зберігання

Для повноцінного додатку часто використовують комбінацію локального збереження та хмарного бекенду для синхронізації даних.

Retry

V

чи зможуть ці дані бачити друзі коритувача ?

Edit

Для того, щоб друзі користувача могли бачити його дані (профіль, списки бажань тощо), потрібна синхронізація через сервер або хмарний бекенд. Це важливий момент, який я повинен уточнити:

Якщо ми використовуємо лише локальне збереження даних (AsyncStorage):

Друзі користувача не зможуть бачити його дані

Кожен користувач матиме свою ізольовану базу даних на власному пристрої

Немає можливості обміну даними між різними користувачами/пристроями

Для реалізації соціальних функцій, де друзі можуть бачити списки бажань один одного, потрібен бекенд-сервер, який би:

Зберігав дані всіх користувачів

Забезпечував автентифікацію та авторизацію

Керував доступом до даних інших користувачів

Відправляв повідомлення та сповіщення

Варіанти реалізації з можливістю обміну даними:

Firebase/Firestore:

Готове рішення для автентифікації та бази даних

Просте налаштування правил доступу до даних

Підтримка реального часу

Суверенний бекенд:

Власний сервер (Node.js, Django, тощо)

API для обміну даними

База даних для зберігання інформації користувачів

Гібридний підхід:

Локальне збереження для швидкого доступу

Синхронізація з бекендом для обміну даними

Для повноцінної реалізації функціоналу "друзі можуть бачити вішлісти один одного", нам потрібно використовувати один із цих підходів із серверною складовою, а не лише локальне збереження даних.

Retry

V

давай розглянемо гібридний підхід, як ми можемо це реалізувати?

Edit

Гібридний підхід дозволить нам поєднати переваги локального збереження даних (швидкий доступ, робота офлайн) з можливостями серверної синхронізації (обмін даними між користувачами). Розглянемо, як це можна реалізувати:

Архітектура гібридного підходу

1. Локальне збереження (клієнт)

AsyncStorage для збереження даних користувача та кешування

Context API або Redux для управління станом додатку

Локальна база даних для роботи офлайн

2. Серверна частина

Firebase, Supabase або власний бекенд на Node.js

База даних для зберігання профілів, відносин між користувачами та вішлістів

API для взаємодії з клієнтом

3. Синхронізація

Механізм синхронізації для оновлення даних між пристроєм та сервером

Стратегія вирішення конфліктів (коли дані змінюються в різних місцях)

Реалізація гібридного підходу

1. Клієнтська частина

javascript

Copy

// services/authService.js - Управління автентифікацією

import AsyncStorage from '@react-native-async-storage/async-storage';

import api from './api'; // Сервіс для взаємодії з сервером

// Ключі для локального збереження

const AUTH\_TOKEN\_KEY = 'wishlist\_auth\_token';

const USER\_DATA\_KEY = 'wishlist\_user\_data';

const LAST\_SYNC\_KEY = 'wishlist\_last\_sync';

// Реєстрація нового користувача

export const register = async (userData) => {

try {

// 1. Відправляємо дані на сервер для реєстрації

const response = await api.post('/auth/register', userData);

const { token, user } = response.data;

// 2. Зберігаємо токен та дані користувача локально

await AsyncStorage.setItem(AUTH\_TOKEN\_KEY, token);

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify(user));

return user;

} catch (error) {

console.error('Registration error:', error);

throw error;

}

};

// Вхід користувача

export const login = async (email, code) => {

try {

// 1. Відправляємо запит на сервер для автентифікації

const response = await api.post('/auth/verify', { email, code });

const { token, user } = response.data;

// 2. Зберігаємо токен та дані користувача локально

await AsyncStorage.setItem(AUTH\_TOKEN\_KEY, token);

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify(user));

return user;

} catch (error) {

console.error('Login error:', error);

throw error;

}

};

// Отримання даних поточного користувача

export const getCurrentUser = async () => {

try {

const userData = await AsyncStorage.getItem(USER\_DATA\_KEY);

return userData ? JSON.parse(userData) : null;

} catch (error) {

console.error('Error getting current user:', error);

return null;

}

};

// Вихід користувача

export const logout = async () => {

try {

await AsyncStorage.removeItem(AUTH\_TOKEN\_KEY);

await AsyncStorage.removeItem(USER\_DATA\_KEY);

return true;

} catch (error) {

console.error('Logout error:', error);

return false;

}

};

javascript

Copy

// services/wishlistService.js - Управління списками бажань

import AsyncStorage from '@react-native-async-storage/async-storage';

import api from './api';

// Ключі для локального збереження

const WISHLISTS\_KEY = 'wishlist\_lists';

// Синхронізація з сервером

export const syncWishlists = async () => {

try {

// 1. Отримуємо локальні дані

const localWishlists = await getLocalWishlists();

// 2. Відправляємо на сервер та отримуємо актуальні дані

const response = await api.post('/wishlists/sync', { wishlists: localWishlists });

const serverWishlists = response.data.wishlists;

// 3. Оновлюємо локальні дані

await AsyncStorage.setItem(WISHLISTS\_KEY, JSON.stringify(serverWishlists));

return serverWishlists;

} catch (error) {

console.error('Sync error:', error);

// Повертаємо локальні дані, якщо синхронізація не вдалася

return await getLocalWishlists();

}

};

// Отримання вішлістів (спочатку з локального сховища, потім синхронізація)

export const getWishlists = async (forceSync = false) => {

try {

// 1. Спочатку отримуємо локальні дані для швидкого відображення

const localWishlists = await getLocalWishlists();

// 2. Якщо немає локальних даних або примусова синхронізація, синхронізуємо з сервером

if (!localWishlists.length || forceSync) {

return await syncWishlists();

}

// 3. Запускаємо фонову синхронізацію, але повертаємо локальні дані

syncWishlists().catch(console.error);

return localWishlists;

} catch (error) {

console.error('Error getting wishlists:', error);

return [];

}

};

// Отримання локальних вішлістів

const getLocalWishlists = async () => {

try {

const data = await AsyncStorage.getItem(WISHLISTS\_KEY);

return data ? JSON.parse(data) : [];

} catch (error) {

console.error('Error getting local wishlists:', error);

return [];

}

};

// Створення нового вішліста (локально + на сервері)

export const createWishlist = async (wishlistData) => {

try {

// 1. Генеруємо тимчасовий ID для локального збереження

const tempId = `temp\_${Date.now()}`;

const newWishlist = {

id: tempId,

...wishlistData,

items: [],

createdAt: new Date().toISOString(),

synced: false

};

// 2. Зберігаємо локально

const wishlists = await getLocalWishlists();

wishlists.push(newWishlist);

await AsyncStorage.setItem(WISHLISTS\_KEY, JSON.stringify(wishlists));

// 3. Відправляємо на сервер (асинхронно)

api.post('/wishlists', wishlistData)

.then(response => {

// Оновлюємо ID та позначаємо як синхронізований

const serverWishlist = response.data;

updateLocalWishlist(tempId, {

...serverWishlist,

synced: true

});

})

.catch(console.error);

return newWishlist;

} catch (error) {

console.error('Error creating wishlist:', error);

throw error;

}

};

// Оновлення локального вішліста

const updateLocalWishlist = async (wishlistId, updatedData) => {

try {

const wishlists = await getLocalWishlists();

const index = wishlists.findIndex(w => w.id === wishlistId);

if (index !== -1) {

wishlists[index] = { ...wishlists[index], ...updatedData };

await AsyncStorage.setItem(WISHLISTS\_KEY, JSON.stringify(wishlists));

}

} catch (error) {

console.error('Error updating local wishlist:', error);

}

};

javascript

Copy

// services/friendService.js - Управління друзями

import api from './api';

import AsyncStorage from '@react-native-async-storage/async-storage';

// Ключі для локального збереження

const FRIENDS\_KEY = 'wishlist\_friends';

const FRIEND\_REQUESTS\_KEY = 'wishlist\_friend\_requests';

// Отримання списку друзів (спочатку локально, потім з сервера)

export const getFriends = async (forceSync = false) => {

try {

// 1. Спочатку отримуємо локальні дані

const localFriends = await getLocalFriends();

// 2. Якщо потрібно, синхронізуємо з сервером

if (forceSync) {

const response = await api.get('/friends');

const serverFriends = response.data;

// Оновлюємо локальні дані

await AsyncStorage.setItem(FRIENDS\_KEY, JSON.stringify(serverFriends));

return serverFriends;

}

// 3. Запускаємо фонову синхронізацію

api.get('/friends')

.then(response => {

AsyncStorage.setItem(FRIENDS\_KEY, JSON.stringify(response.data));

})

.catch(console.error);

return localFriends;

} catch (error) {

console.error('Error getting friends:', error);

return [];

}

};

// Отримання локальних друзів

const getLocalFriends = async () => {

try {

const data = await AsyncStorage.getItem(FRIENDS\_KEY);

return data ? JSON.parse(data) : [];

} catch (error) {

console.error('Error getting local friends:', error);

return [];

}

};

// Пошук користувачів для додавання в друзі

export const searchUsers = async (query) => {

try {

const response = await api.get(`/users/search?q=${encodeURIComponent(query)}`);

return response.data;

} catch (error) {

console.error('Error searching users:', error);

return [];

}

};

// Надсилання запиту на дружбу

export const sendFriendRequest = async (userId) => {

try {

const response = await api.post('/friends/requests', { userId });

return response.data;

} catch (error) {

console.error('Error sending friend request:', error);

throw error;

}

};

// Отримання запитів на дружбу

export const getFriendRequests = async () => {

try {

const response = await api.get('/friends/requests');

// Кешуємо запити локально

await AsyncStorage.setItem(FRIEND\_REQUESTS\_KEY, JSON.stringify(response.data));

return response.data;

} catch (error) {

// При помилці повертаємо кешовані дані

console.error('Error getting friend requests:', error);

const cached = await AsyncStorage.getItem(FRIEND\_REQUESTS\_KEY);

return cached ? JSON.parse(cached) : [];

}

};

2. Серверна частина (спрощений приклад на Node.js + Express)

javascript

Copy

// server/routes/auth.js

const express = require('express');

const router = express.Router();

const authController = require('../controllers/authController');

// Маршрути для автентифікації

router.post('/register', authController.register);

router.post('/send-code', authController.sendVerificationCode);

router.post('/verify', authController.verifyCode);

module.exports = router;

// server/routes/wishlists.js

const express = require('express');

const router = express.Router();

const wishlistController = require('../controllers/wishlistController');

const { authenticate } = require('../middleware/auth');

// Всі маршрути захищені middleware автентифікації

router.use(authenticate);

router.get('/', wishlistController.getWishlists);

router.post('/', wishlistController.createWishlist);

router.post('/sync', wishlistController.syncWishlists);

router.get('/:id', wishlistController.getWishlist);

router.put('/:id', wishlistController.updateWishlist);

router.delete('/:id', wishlistController.deleteWishlist);

// Маршрути для елементів списку бажань

router.post('/:id/items', wishlistController.addItem);

router.put('/:id/items/:itemId', wishlistController.updateItem);

router.delete('/:id/items/:itemId', wishlistController.deleteItem);

module.exports = router;

// server/routes/friends.js

const express = require('express');

const router = express.Router();

const friendController = require('../controllers/friendController');

const { authenticate } = require('../middleware/auth');

// Всі маршрути захищені middleware автентифікації

router.use(authenticate);

router.get('/', friendController.getFriends);

router.get('/requests', friendController.getFriendRequests);

router.post('/requests', friendController.sendFriendRequest);

router.put('/requests/:id/accept', friendController.acceptFriendRequest);

router.put('/requests/:id/reject', friendController.rejectFriendRequest);

router.delete('/:id', friendController.removeFriend);

module.exports = router;

3. Інтеграція в додатку (використання Context API)

javascript

Copy

// context/AuthContext.js

import React, { createContext, useState, useEffect, useContext } from 'react';

import \* as authService from '../services/authService';

// Створюємо контекст

const AuthContext = createContext();

// Хук для використання контексту

export const useAuth = () => useContext(AuthContext);

// Провайдер контексту

export const AuthProvider = ({ children }) => {

const [user, setUser] = useState(null);

const [isLoading, setIsLoading] = useState(true);

// Перевіряємо автентифікацію при завантаженні

useEffect(() => {

const checkAuth = async () => {

const userData = await authService.getCurrentUser();

setUser(userData);

setIsLoading(false);

};

checkAuth();

}, []);

// Функція для реєстрації

const register = async (userData) => {

setIsLoading(true);

try {

const newUser = await authService.register(userData);

setUser(newUser);

return newUser;

} finally {

setIsLoading(false);

}

};

// Функція для входу

const login = async (email, code) => {

setIsLoading(true);

try {

const userData = await authService.login(email, code

Retry

V

Давай будемо використовувати локальне збереження і Firebase

Edit

Чудовий вибір! Поєднання локального збереження (для швидкої роботи офлайн) і Firebase (для синхронізації між користувачами та пристроями) дасть нам гнучкий та потужний розв'язок.

Давайте розпочнемо з налаштування Firebase і локального збереження для нашого додатка Wishlist:

1. Встановлення необхідних пакетів

bash

Copy

# Встановлення Firebase

npx expo install firebase

# Встановлення AsyncStorage для локального кешування

npx expo install @react-native-async-storage/async-storage

2. Налаштування Firebase

Створіть файл налаштувань Firebase:

bash

Copy

mkdir -p app/config

cat > app/config/firebase.js << 'EOF'

import { initializeApp } from 'firebase/app';

import { getAuth } from 'firebase/auth';

import { getFirestore } from 'firebase/firestore';

import { getStorage } from 'firebase/storage';

// Firebase конфігурація

// ПРИМІТКА: У реальному додатку це краще зберігати в .env файлі

const firebaseConfig = {

apiKey: "YOUR\_API\_KEY",

authDomain: "your-app.firebaseapp.com",

projectId: "your-app-id",

storageBucket: "your-app.appspot.com",

messagingSenderId: "YOUR\_SENDER\_ID",

appId: "YOUR\_APP\_ID"

};

// Ініціалізуємо Firebase

const app = initializeApp(firebaseConfig);

// Експортуємо необхідні сервіси

export const auth = getAuth(app);

export const db = getFirestore(app);

export const storage = getStorage(app);

export default app;

EOF

3. Створення сервісів для автентифікації та управління даними

bash

Copy

mkdir -p app/services

Сервіс автентифікації з Firebase + локальне кешування:

bash

Copy

cat > app/services/authService.js << 'EOF'

import {

createUserWithEmailAndPassword,

signInWithEmailAndPassword,

sendEmailVerification,

signOut,

updateProfile

} from 'firebase/auth';

import { doc, setDoc, getDoc, serverTimestamp } from 'firebase/firestore';

import AsyncStorage from '@react-native-async-storage/async-storage';

import { auth, db } from '../config/firebase';

// Ключі для локального сховища

const USER\_DATA\_KEY = 'wishlist\_user\_data';

const VERIFICATION\_CODES\_KEY = 'wishlist\_verification\_codes';

// Генерування коду підтвердження

export const generateVerificationCode = () => {

return Math.floor(1000 + Math.random() \* 9000).toString();

};

// Збереження коду підтвердження

export const saveVerificationCode = async (email, code) => {

try {

const codesJson = await AsyncStorage.getItem(VERIFICATION\_CODES\_KEY) || '{}';

const codes = JSON.parse(codesJson);

codes[email] = {

code,

timestamp: Date.now()

};

await AsyncStorage.setItem(VERIFICATION\_CODES\_KEY, JSON.stringify(codes));

return true;

} catch (error) {

console.error('Error saving verification code:', error);

return false;

}

};

// Перевірка коду підтвердження

export const verifyCode = async (email, code) => {

try {

const codesJson = await AsyncStorage.getItem(VERIFICATION\_CODES\_KEY) || '{}';

const codes = JSON.parse(codesJson);

const savedCode = codes[email];

// Перевіряємо наявність коду та його актуальність (30 хвилин)

if (savedCode && savedCode.code === code &&

Date.now() - savedCode.timestamp < 30 \* 60 \* 1000) {

return true;

}

return false;

} catch (error) {

console.error('Error verifying code:', error);

return false;

}

};

// Перевірка, чи існує користувач

export const userExists = async (email) => {

try {

// В Firebase немає прямого методу для перевірки існування користувача

// Але ми можемо перевірити колекцію користувачів у Firestore

const userRef = doc(db, 'users', email);

const userSnap = await getDoc(userRef);

return userSnap.exists();

} catch (error) {

console.error('Error checking user existence:', error);

return false;

}

};

// Створення нового користувача

export const createUser = async (userData) => {

try {

const { email, name, username } = userData;

// Генеруємо тимчасовий пароль для Firebase Authentication

const tempPassword = Math.random().toString(36).slice(-8);

// Створюємо користувача в Firebase Authentication

const userCredential = await createUserWithEmailAndPassword(auth, email, tempPassword);

const user = userCredential.user;

// Оновлюємо профіль користувача

await updateProfile(user, {

displayName: name

});

// Зберігаємо додаткові дані у Firestore

await setDoc(doc(db, 'users', email), {

uid: user.uid,

email,

name,

username,

avatar: null,

wishlists: [],

friends: [],

friendRequests: [],

createdAt: serverTimestamp(),

lastLogin: serverTimestamp()

});

// Кешуємо дані користувача локально

const userData = {

uid: user.uid,

email,

name,

username,

avatar: null

};

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify(userData));

return userData;

} catch (error) {

console.error('Error creating user:', error);

throw error;

}

};

// Вхід користувача

export const loginUser = async (email) => {

try {

// Отримуємо дані користувача з Firestore

const userRef = doc(db, 'users', email);

const userSnap = await getDoc(userRef);

if (!userSnap.exists()) {

throw new Error('User not found');

}

const userData = userSnap.data();

// Оновлюємо час останнього входу

await setDoc(userRef, {

lastLogin: serverTimestamp()

}, { merge: true });

// Кешуємо дані користувача локально

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify({

uid: userData.uid,

email: userData.email,

name: userData.name,

username: userData.username,

avatar: userData.avatar

}));

return userData;

} catch (error) {

console.error('Error logging in:', error);

throw error;

}

};

// Вихід користувача

export const logoutUser = async () => {

try {

await signOut(auth);

await AsyncStorage.removeItem(USER\_DATA\_KEY);

return true;

} catch (error) {

console.error('Error logging out:', error);

return false;

}

};

// Отримання даних поточного користувача

export const getCurrentUser = async () => {

try {

// Спочатку перевіряємо локальне сховище для швидкого доступу

const userData = await AsyncStorage.getItem(USER\_DATA\_KEY);

if (userData) {

return JSON.parse(userData);

}

// Якщо локальних даних немає, але користувач автентифікований у Firebase

const user = auth.currentUser;

if (user) {

// Отримуємо більше даних з Firestore

const userRef = doc(db, 'users', user.email);

const userSnap = await getDoc(userRef);

if (userSnap.exists()) {

const firestoreData = userSnap.data();

// Кешуємо дані локально

const userData = {

uid: user.uid,

email: user.email,

name: firestoreData.name,

username: firestoreData.username,

avatar: firestoreData.avatar

};

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify(userData));

return userData;

}

}

return null;

} catch (error) {

console.error('Error getting current user:', error);

return null;

}

};

// Перевірка автентифікації користувача

export const isAuthenticated = async () => {

try {

// Перевіряємо, чи є локальні дані

const userData = await AsyncStorage.getItem(USER\_DATA\_KEY);

if (userData) {

return true;

}

// Перевіряємо Firebase Authentication

return auth.currentUser !== null;

} catch (error) {

console.error('Error checking authentication:', error);

return false;

}

};

// Оновлення даних користувача

export const updateUserData = async (userData) => {

try {

const { email, name, username, avatar } = userData;

// Оновлюємо дані в Firestore

const userRef = doc(db, 'users', email);

await setDoc(userRef, {

name,

username,

avatar,

updatedAt: serverTimestamp()

}, { merge: true });

// Оновлюємо дані в Firebase Authentication

if (auth.currentUser) {

await updateProfile(auth.currentUser, {

displayName: name,

photoURL: avatar

});

}

// Оновлюємо локальні дані

const localUserData = JSON.parse(await AsyncStorage.getItem(USER\_DATA\_KEY) || '{}');

const updatedUserData = {

...localUserData,

name,

username,

avatar

};

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify(updatedUserData));

return updatedUserData;

} catch (error) {

console.error('Error updating user data:', error);

throw error;

}

};

EOF

Сервіс для управління вішлістами:

bash

Copy

cat > app/services/wishlistService.js << 'EOF'

import {

collection,

doc,

setDoc,

addDoc,

getDoc,

getDocs,

updateDoc,

deleteDoc,

serverTimestamp,

query,

where

} from 'firebase/firestore';

import AsyncStorage from '@react-native-async-storage/async-storage';

import { db } from '../config/firebase';

import { getCurrentUser } from './authService';

// Ключі для локального сховища

const WISHLISTS\_KEY = 'wishlist\_lists';

// Локальне отримання вішлістів

const getLocalWishlists = async () => {

try {

const data = await AsyncStorage.getItem(WISHLISTS\_KEY);

return data ? JSON.parse(data) : [];

} catch (error) {

console.error('Error getting local wishlists:', error);

return [];

}

};

// Локальне збереження вішлістів

const saveLocalWishlists = async (wishlists) => {

try {

await AsyncStorage.setItem(WISHLISTS\_KEY, JSON.stringify(wishlists));

} catch (error) {

console.error('Error saving local wishlists:', error);

}

};

// Отримання вішлістів користувача

export const getUserWishlists = async (forceSync = false) => {

try {

// Спочатку отримуємо з локального сховища для швидкого відображення

const localWishlists = await getLocalWishlists();

if (!forceSync && localWishlists.length > 0) {

return localWishlists;

}

// Отримуємо дані поточного користувача

const user = await getCurrentUser();

if (!user) {

return [];

}

// Отримуємо вішлісти з Firestore

const wishlistsRef = collection(db, 'wishlists');

const q = query(wishlistsRef, where('ownerEmail', '==', user.email));

const querySnapshot = await getDocs(q);

const wishlists = [];

querySnapshot.forEach((doc) => {

wishlists.push({

id: doc.id,

...doc.data()

});

});

// Зберігаємо отримані дані локально

await saveLocalWishlists(wishlists);

return wishlists;

} catch (error) {

console.error('Error getting wishlists:', error);

// При помилці повертаємо локальні дані

return await getLocalWishlists();

}

};

// Отримання вішліста за ID

export const getWishlistById = async (wishlistId) => {

try {

// Спочатку шукаємо локально

const localWishlists = await getLocalWishlists();

const localWishlist = localWishlists.find(w => w.id === wishlistId);

// Отримуємо дані з Firestore

const wishlistRef = doc(db, 'wishlists', wishlistId);

const wishlistSnap = await getDoc(wishlistRef);

if (wishlistSnap.exists()) {

const wishlistData = {

id: wishlistSnap.id,

...wishlistSnap.data()

};

// Оновлюємо локальні дані

if (localWishlist) {

const updatedWishlists = localWishlists.map(w =>

w.id === wishlistId ? wishlistData : w

);

await saveLocalWishlists(updatedWishlists);

}

return wishlistData;

}

// Якщо не знайдено у Firestore, повертаємо локальні дані

return localWishlist || null;

} catch (error) {

console.error('Error getting wishlist by id:', error);

// При помилці повертаємо локальні дані

const localWishlists = await getLocalWishlists();

return localWishlists.find(w => w.id === wishlistId) || null;

}

};

// Створення нового вішліста

export const createWishlist = async (wishlistData) => {

try {

const user = await getCurrentUser();

if (!user) {

throw new Error('User not authenticated');

}

// Підготовка даних для збереження

const newWishlist = {

...wishlistData,

ownerEmail: user.email,

ownerName: user.name,

ownerUsername: user.username,

items: [],

isPrivate: wishlistData.isPrivate || false,

createdAt: serverTimestamp(),

updatedAt: serverTimestamp()

};

// Додаємо до Firestore

const wishlistRef = await addDoc(collection(db, 'wishlists'), newWishlist);

// Отримуємо ID документа

const wishlistId = wishlistRef.id;

// Додаємо ID до об'єкта

const wishlistWithId = {

id: wishlistId,

...newWishlist,

createdAt: new Date().toISOString(), // для локального збереження замінюємо serverTimestamp

updatedAt: new Date().toISOString()

};

// Оновлюємо локальні дані

const localWishlists = await getLocalWishlists();

localWishlists.push(wishlistWithId);

await saveLocalWishlists(localWishlists);

return wishlistWithId;

} catch (error) {

console.error('Error creating wishlist:', error);

throw error;

}

};

// Додавання елемента до вішліста

export const addItemToWishlist = async (wishlistId, itemData) => {

try {

// Отримуємо вішліст

const wishlist = await getWishlistById(wishlistId);

if (!wishlist) {

throw new Error('Wishlist not found');

}

// Створюємо новий елемент

const newItem = {

id: Date.now().toString(), // Унікальний ID для елемента

...itemData,

addedAt: new Date().toISOString()

};

// Додаємо елемент до вішліста

const updatedItems = [...(wishlist.items || []), newItem];

// Оновлюємо у Firestore

const wishlistRef = doc(db, 'wishlists', wishlistId);

await updateDoc(wishlistRef, {

items: updatedItems,

updatedAt: serverTimestamp()

});

// Оновлюємо локальні дані

const localWishlists = await getLocalWishlists();

const updatedWishlists = localWishlists.map(w => {

if (w.id === wishlistId) {

return {

...w,

items: updatedItems,

updatedAt: new Date().toISOString()

};

}

return w;

});

await saveLocalWishlists(updatedWishlists);

return newItem;

} catch (error) {

console.error('Error adding item to wishlist:', error);

throw error;

}

};

// Видалення елемента з вішліста

export const removeItemFromWishlist = async (wishlistId, itemId) => {

try {

// Отримуємо вішліст

const wishlist = await getWishlistById(wishlistId);

if (!wishlist) {

throw new Error('Wishlist not found');

}

// Фільтруємо елементи

const updatedItems = (wishlist.items || []).filter(item => item.id !== itemId);

// Оновлюємо у Firestore

const wishlistRef = doc(db, 'wishlists', wishlistId);

await updateDoc(wishlistRef, {

items: updatedItems,

updatedAt: serverTimestamp()

});

// Оновлюємо локальні дані

const localWishlists = await getLocalWishlists();

const updatedWishlists = localWishlists.map(w => {

if (w.id === wishlistId) {

return {

...w,

items: updatedItems,

updatedAt: new Date().toISOString()

};

}

return w;

});

await saveLocalWishlists(updatedWishlists);

return true;

} catch (error) {

console.error('Error removing item from wishlist:', error);

throw error;

}

};

// Видалення вішліста

export const deleteWishlist = async (wishlistId) => {

try {

// Видаляємо з Firestore

await deleteDoc(doc(db, 'wishlists', wishlistId));

// Оновлюємо локальні дані

const localWishlists = await getLocalWishlists();

const updatedWishlists = localWishlists.filter(w => w.id !== wishlistId);

await saveLocalWishlists(updatedWishlists);

return true;

} catch (error) {

console.error('Error deleting wishlist:', error);

// Навіть якщо видалення з Firestore не вдалося,

// видалимо з локального сховища

const localWishlists = await getLocalWishlists();

const updatedWishlists = localWishlists.filter(w => w.id !== wishlistId);

await saveLocalWishlists(updatedWishlists);

return false;

}

};

// Отримання публічних вішлістів друга

export const getFriendWishlists = async (friendEmail) => {

try {

// Отримуємо вішлісти з Firestore

const wishlistsRef = collection(db, 'wishlists');

const q = query(

wishlistsRef,

where('ownerEmail', '==', friendEmail),

where('isPrivate', '==', false)

);

const querySnapshot = await getDocs(q);

const wishlists = [];

querySnapshot.forEach((doc) => {

wishlists.push({

id: doc.id,

...doc.data()

});

});

return wishlists;

} catch (error) {

console.error('Error getting friend wishlists:', error);

return [];

}

};

EOF

Сервіс для управління друзями:

bash

Copy

cat > app/services/friendService.js << 'EOF'

import {

collection,

doc,

getDoc,

getDocs,

setDoc,

updateDoc,

deleteDoc,

arrayUnion,

arrayRemove,

query,

where,

serverTimestamp,

runTransaction

} from 'firebase/firestore';

import AsyncStorage from '@react-native-async-storage/async-storage';

import { db } from '../config/firebase';

import { getCurrentUser } from './authService';

// Ключі для локального сховища

const FRIENDS\_KEY = 'wishlist\_friends';

const FRIEND\_REQUESTS\_KEY = 'wishlist\_friend\_requests';

// Отримання даних користувача за email

export const getUserByEmail = async (email) => {

try {

const userRef = doc(db, 'users', email);

const userSnap = await getDoc(userRef);

if (userSnap.exists()) {

return {

email,

...userSnap.data()

};

}

return null;

} catch (error) {

console.error('Error getting user by email:', error);

return null;

}

};

// Пошук користувачів

export const searchUsers = async (query) => {

try {

// У Firestore немає вбудованого пошуку тексту

// Тут використовуємо простий метод для прототипу

// В реальному додатку краще використовувати Algolia, Elastic Search або Cloud Functions

// Отримуємо всіх користувачів

const usersRef = collection(db, 'users');

const snapshot = await getDocs(usersRef);

// Фільтруємо локально

const users = [];

const currentUser = await getCurrentUser();

snapshot.forEach((doc) => {

const userData = doc.data();

// Пропускаємо поточного користувача

if (userData.email === currentUser?.email) {

return;

}

// Шукаємо збіги в імені, username або email

const lowerQuery = query.toLowerCase();

if (

userData.name?.toLowerCase().includes(lowerQuery) ||

userData.username?.toLowerCase().includes(lowerQuery) ||

userData.email?.toLowerCase().includes(lowerQuery)

) {

users.push({

email: doc.id,

name: userData.name,

username: userData.username,

avatar: userData.avatar

});

}

});

return users;

} catch (error) {

console.error('Error searching users:', error);

return [];

}

};

// Отримання друзів поточного користувача

export const getFriends = async (forceSync = false) => {

try {

// Отримуємо поточного користувача

const currentUser = await getCurrentUser();

if (!currentUser) {

return [];

}

// Спочатку перевіряємо локальне сховище

if (!forceSync) {

const localFriends = await AsyncStorage.getItem(FRIENDS\_KEY);

if (localFriends) {

return JSON.parse(localFriends);

}

}

// Отримуємо список друзів з Firestore

const userRef = doc(db, 'users', currentUser.email);

const userSnap = await getDoc(userRef);

if (!userSnap.exists()) {

return [];

}

const userData = userSnap.data();

const friendEmails = userData.friends || [];

// Отримуємо деталі про кожного друга

const friends = [];

for (const email of friendEmails) {

const friendData = await getUserByEmail(email);

if (friendData) {

friends.push({

email,

name: friendData.name,

username: friendData.username,

avatar: friendData.avatar

});

}

}

// Зберігаємо в локальному сховищі

await AsyncStorage.setItem(FRIENDS\_KEY, JSON.stringify(friends));

return friends;

} catch (error) {

console.error('Error getting friends:', error);

// При помилці повертаємо локальні дані

const localFriends = await AsyncStorage.getItem(FRIENDS\_KEY);

return localFriends ? JSON.parse(localFriends) : [];

}

};

// Отримання запитів на дружбу

export const getFriendRequests = async () => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

return [];

}

// Отримуємо запити з Firestore

const userRef = doc(db, 'users', currentUser.email);

const userSnap = await getDoc(userRef);

if (!userSnap.exists()) {

return [];

}

const userData = userSnap.data();

const requestEmails = userData.friendRequests || [];

// Отримуємо деталі про кожен запит

const requests = [];

for (const email of requestEmails) {

const userData = await getUserByEmail(email);

if (userData) {

requests.push({

email,

name: userData.name,

username: userData.username,

avatar: userData.avatar

});

}

}

// Зберігаємо в локальному сховищі

await AsyncStorage.setItem(FRIEND\_REQUESTS\_KEY, JSON.stringify(requests));

return requests;

} catch (error) {

console.error('Error getting friend requests:', error);

// При помилці повертаємо локальні дані

const localRequests = await AsyncStorage.getItem(FRIEND\_REQUESTS\_KEY);

return localRequests ? JSON.parse(localRequests) : [];

}

};

// Надсилання запиту на дружбу

export const sendFriendRequest = async (friendEmail) => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

throw new Error('User not authenticated');

}

// Перевіряємо, чи існує користувач

const friendData = await getUserByEmail(friendEmail);

if (!friendData) {

throw new Error('User not found');

}

// Додаємо запит до списку запитів користувача

const friendRef = doc(db, 'users', friendEmail);

await updateDoc(friendRef, {

friendRequests: arrayUnion(currentUser.email)

});

return true;

} catch (error) {

console.error('Error sending friend request:', error);

throw error;

}

};

// Прийняття запиту на дружбу

export const acceptFriendRequest = async (friendEmail) => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

throw new Error('User not authenticated');

}

// Використовуємо транзакцію для атомарного оновлення обох документів

await runTransaction(db, async (transaction) => {

// Документи користувачів

const currentUserRef = doc(db, 'users', currentUser.email);

const friendRef = doc(db, 'users', friendEmail);

// Отримуємо поточні дані

const currentUserDoc = await transaction.get(currentUserRef);

const friendDoc = await transaction.get(friendRef);

if (!currentUserDoc.exists() || !friendDoc.exists()) {

throw new Error('User not found');

}

// Додаємо користувачів до списків друзів один одного

transaction.update(currentUserRef, {

friends: arrayUnion(friendEmail),

friendRequests: arrayRemove(friendEmail)

});

transaction.update(friendRef, {

friends: arrayUnion(currentUser.email)

});

});

// Оновлюємо локальні дані

await getFriends(true);

await getFriendRequests();

return true;

} catch (error) {

console.error('Error accepting friend request:', error);

throw error;

}

};

// Відхилення запиту на дружбу

export const rejectFriendRequest = async (friendEmail) => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

throw new Error('User not authenticated');

}

// Видаляємо запит зі списку

const userRef = doc(db, 'users', currentUser.email);

await updateDoc(userRef, {

friendRequests: arrayRemove(friendEmail)

});

// Оновлюємо локальні дані

await getFriendRequests();

return true;

} catch (error) {

console.error('Error rejecting friend request:', error);

throw error;

}

};

// Видалення друга

export const removeFriend = async (friendEmail) => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

throw new Error

Retry

V

ти зупинився на Сервіс для управління друзями:

Edit

Давайте продовжимо створення сервісу для управління друзями:

bash

Copy

cat > app/services/friendService.js << 'EOF'

import {

collection,

doc,

getDoc,

getDocs,

setDoc,

updateDoc,

deleteDoc,

arrayUnion,

arrayRemove,

query,

where,

serverTimestamp,

runTransaction

} from 'firebase/firestore';

import AsyncStorage from '@react-native-async-storage/async-storage';

import { db } from '../config/firebase';

import { getCurrentUser } from './authService';

// Ключі для локального сховища

const FRIENDS\_KEY = 'wishlist\_friends';

const FRIEND\_REQUESTS\_KEY = 'wishlist\_friend\_requests';

// Отримання даних користувача за email

export const getUserByEmail = async (email) => {

try {

const userRef = doc(db, 'users', email);

const userSnap = await getDoc(userRef);

if (userSnap.exists()) {

return {

email,

...userSnap.data()

};

}

return null;

} catch (error) {

console.error('Error getting user by email:', error);

return null;

}

};

// Пошук користувачів

export const searchUsers = async (query) => {

try {

// У Firestore немає вбудованого пошуку тексту

// Тут використовуємо простий метод для прототипу

// В реальному додатку краще використовувати Algolia, Elastic Search або Cloud Functions

// Отримуємо всіх користувачів

const usersRef = collection(db, 'users');

const snapshot = await getDocs(usersRef);

// Фільтруємо локально

const users = [];

const currentUser = await getCurrentUser();

snapshot.forEach((doc) => {

const userData = doc.data();

// Пропускаємо поточного користувача

if (userData.email === currentUser?.email) {

return;

}

// Шукаємо збіги в імені, username або email

const lowerQuery = query.toLowerCase();

if (

userData.name?.toLowerCase().includes(lowerQuery) ||

userData.username?.toLowerCase().includes(lowerQuery) ||

userData.email?.toLowerCase().includes(lowerQuery)

) {

users.push({

email: doc.id,

name: userData.name,

username: userData.username,

avatar: userData.avatar

});

}

});

return users;

} catch (error) {

console.error('Error searching users:', error);

return [];

}

};

// Отримання друзів поточного користувача

export const getFriends = async (forceSync = false) => {

try {

// Отримуємо поточного користувача

const currentUser = await getCurrentUser();

if (!currentUser) {

return [];

}

// Спочатку перевіряємо локальне сховище

if (!forceSync) {

const localFriends = await AsyncStorage.getItem(FRIENDS\_KEY);

if (localFriends) {

return JSON.parse(localFriends);

}

}

// Отримуємо список друзів з Firestore

const userRef = doc(db, 'users', currentUser.email);

const userSnap = await getDoc(userRef);

if (!userSnap.exists()) {

return [];

}

const userData = userSnap.data();

const friendEmails = userData.friends || [];

// Отримуємо деталі про кожного друга

const friends = [];

for (const email of friendEmails) {

const friendData = await getUserByEmail(email);

if (friendData) {

friends.push({

email,

name: friendData.name,

username: friendData.username,

avatar: friendData.avatar

});

}

}

// Зберігаємо в локальному сховищі

await AsyncStorage.setItem(FRIENDS\_KEY, JSON.stringify(friends));

return friends;

} catch (error) {

console.error('Error getting friends:', error);

// При помилці повертаємо локальні дані

const localFriends = await AsyncStorage.getItem(FRIENDS\_KEY);

return localFriends ? JSON.parse(localFriends) : [];

}

};

// Отримання запитів на дружбу

export const getFriendRequests = async () => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

return [];

}

// Отримуємо запити з Firestore

const userRef = doc(db, 'users', currentUser.email);

const userSnap = await getDoc(userRef);

if (!userSnap.exists()) {

return [];

}

const userData = userSnap.data();

const requestEmails = userData.friendRequests || [];

// Отримуємо деталі про кожен запит

const requests = [];

for (const email of requestEmails) {

const userData = await getUserByEmail(email);

if (userData) {

requests.push({

email,

name: userData.name,

username: userData.username,

avatar: userData.avatar

});

}

}

// Зберігаємо в локальному сховищі

await AsyncStorage.setItem(FRIEND\_REQUESTS\_KEY, JSON.stringify(requests));

return requests;

} catch (error) {

console.error('Error getting friend requests:', error);

// При помилці повертаємо локальні дані

const localRequests = await AsyncStorage.getItem(FRIEND\_REQUESTS\_KEY);

return localRequests ? JSON.parse(localRequests) : [];

}

};

// Надсилання запиту на дружбу

export const sendFriendRequest = async (friendEmail) => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

throw new Error('User not authenticated');

}

// Перевіряємо, чи існує користувач

const friendData = await getUserByEmail(friendEmail);

if (!friendData) {

throw new Error('User not found');

}

// Додаємо запит до списку запитів користувача

const friendRef = doc(db, 'users', friendEmail);

await updateDoc(friendRef, {

friendRequests: arrayUnion(currentUser.email)

});

return true;

} catch (error) {

console.error('Error sending friend request:', error);

throw error;

}

};

// Прийняття запиту на дружбу

export const acceptFriendRequest = async (friendEmail) => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

throw new Error('User not authenticated');

}

// Використовуємо транзакцію для атомарного оновлення обох документів

await runTransaction(db, async (transaction) => {

// Документи користувачів

const currentUserRef = doc(db, 'users', currentUser.email);

const friendRef = doc(db, 'users', friendEmail);

// Отримуємо поточні дані

const currentUserDoc = await transaction.get(currentUserRef);

const friendDoc = await transaction.get(friendRef);

if (!currentUserDoc.exists() || !friendDoc.exists()) {

throw new Error('User not found');

}

// Додаємо користувачів до списків друзів один одного

transaction.update(currentUserRef, {

friends: arrayUnion(friendEmail),

friendRequests: arrayRemove(friendEmail)

});

transaction.update(friendRef, {

friends: arrayUnion(currentUser.email)

});

});

// Оновлюємо локальні дані

await getFriends(true);

await getFriendRequests();

return true;

} catch (error) {

console.error('Error accepting friend request:', error);

throw error;

}

};

// Відхилення запиту на дружбу

export const rejectFriendRequest = async (friendEmail) => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

throw new Error('User not authenticated');

}

// Видаляємо запит зі списку

const userRef = doc(db, 'users', currentUser.email);

await updateDoc(userRef, {

friendRequests: arrayRemove(friendEmail)

});

// Оновлюємо локальні дані

await getFriendRequests();

return true;

} catch (error) {

console.error('Error rejecting friend request:', error);

throw error;

}

};

// Видалення друга

export const removeFriend = async (friendEmail) => {

try {

const currentUser = await getCurrentUser();

if (!currentUser) {

throw new Error('User not authenticated');

}

// Використовуємо транзакцію для атомарного оновлення обох документів

await runTransaction(db, async (transaction) => {

// Документи користувачів

const currentUserRef = doc(db, 'users', currentUser.email);

const friendRef = doc(db, 'users', friendEmail);

// Отримуємо поточні дані

const currentUserDoc = await transaction.get(currentUserRef);

const friendDoc = await transaction.get(friendRef);

if (!currentUserDoc.exists() || !friendDoc.exists()) {

throw new Error('User not found');

}

// Видаляємо користувачів зі списків друзів один одного

transaction.update(currentUserRef, {

friends: arrayRemove(friendEmail)

});

transaction.update(friendRef, {

friends: arrayRemove(currentUser.email)

});

});

// Оновлюємо локальні дані

await getFriends(true);

return true;

} catch (error) {

console.error('Error removing friend:', error);

throw error;

}

};

EOF

4. Створення AuthContext для управління аутентифікацією

Тепер створімо контекст, який буде надавати аутентифікаційні дані всім компонентам:

bash

Copy

mkdir -p app/context

cat > app/context/AuthContext.js << 'EOF'

import React, { createContext, useState, useEffect, useContext } from 'react';

import AsyncStorage from '@react-native-async-storage/async-storage';

import \* as authService from '../services/authService';

// Створюємо контекст

const AuthContext = createContext();

// Провайдер контексту

export const AuthProvider = ({ children }) => {

const [user, setUser] = useState(null);

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

// Отримуємо поточного користувача при завантаженні

useEffect(() => {

const loadUser = async () => {

try {

const currentUser = await authService.getCurrentUser();

setUser(currentUser);

} catch (err) {

setError(err.message);

} finally {

setLoading(false);

}

};

loadUser();

}, []);

// Перевірка існування користувача

const checkUserExists = async (email) => {

try {

return await authService.userExists(email);

} catch (err) {

setError(err.message);

return false;

}

};

// Генерування та відправка коду підтвердження

const sendVerificationCode = async (email) => {

try {

setLoading(true);

const code = authService.generateVerificationCode();

await authService.saveVerificationCode(email, code);

return code; // У реальному додатку ми б не повертали код, а відправляли на емейл

} catch (err) {

setError(err.message);

throw err;

} finally {

setLoading(false);

}

};

// Перевірка коду

const verifyCode = async (email, code) => {

try {

setLoading(true);

return await authService.verifyCode(email, code);

} catch (err) {

setError(err.message);

return false;

} finally {

setLoading(false);

}

};

// Реєстрація нового користувача

const register = async (userData) => {

try {

setLoading(true);

const newUser = await authService.createUser(userData);

setUser(newUser);

return newUser;

} catch (err) {

setError(err.message);

throw err;

} finally {

setLoading(false);

}

};

// Вхід користувача

const login = async (email) => {

try {

setLoading(true);

const userData = await authService.loginUser(email);

setUser(userData);

return userData;

} catch (err) {

setError(err.message);

throw err;

} finally {

setLoading(false);

}

};

// Вихід користувача

const logout = async () => {

try {

setLoading(true);

await authService.logoutUser();

setUser(null);

} catch (err) {

setError(err.message);

} finally {

setLoading(false);

}

};

// Оновлення даних користувача

const updateUser = async (userData) => {

try {

setLoading(true);

const updatedUser = await authService.updateUserData(userData);

setUser(updatedUser);

return updatedUser;

} catch (err) {

setError(err.message);

throw err;

} finally {

setLoading(false);

}

};

// Отримання помилки

const getError = () => {

const currentError = error;

setError(null);

return currentError;

};

const value = {

user,

loading,

error,

checkUserExists,

sendVerificationCode,

verifyCode,

register,

login,

logout,

updateUser,

getError

};

return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;

};

// Хук для використання контексту

export const useAuth = () => {

const context = useContext(AuthContext);

if (context === undefined) {

throw new Error('useAuth must be used within an AuthProvider');

}

return context;

};

EOF

5. Оновлення екранів автентифікації

Тепер оновимо наші екрани автентифікації, щоб використовувати нові сервіси та контекст:

Оновлення екрану привітання/входу:

bash

Copy

cat > app/\(auth\)/welcome.js << 'EOF'

import React, { useState, useEffect } from 'react';

import { View, Text, StyleSheet, TextInput, TouchableOpacity, KeyboardAvoidingView, Platform, ActivityIndicator, Alert } from 'react-native';

import { router } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

import { useAuth } from '../context/AuthContext';

export default function WelcomeScreen() {

const [email, setEmail] = useState('');

const [isExistingUser, setIsExistingUser] = useState(false);

const [isCheckingUser, setIsCheckingUser] = useState(false);

const { checkUserExists, sendVerificationCode, loading, error, getError } = useAuth();

// Перевіряємо помилки

useEffect(() => {

if (error) {

Alert.alert('Error', getError());

}

}, [error]);

// Функція для перевірки, чи існує користувач з таким email

const checkUser = async (inputEmail) => {

if (!inputEmail || !inputEmail.includes('@')) return;

setIsCheckingUser(true);

const exists = await checkUserExists(inputEmail);

setIsExistingUser(exists);

setIsCheckingUser(false);

};

// Перевіряємо, чи існує користувач, коли змінюється email

useEffect(() => {

const timeoutId = setTimeout(() => {

checkUser(email);

}, 500);

return () => clearTimeout(timeoutId);

}, [email]);

const handleContinue = async () => {

if (!email || !email.includes('@')) {

Alert.alert('Error', 'Please enter a valid email address');

return;

}

try {

// Генеруємо код верифікації та зберігаємо його

const code = await sendVerificationCode(email);

// В реальному додатку тут був би запит на сервер для відправки коду на пошту

Alert.alert('Verification Code', `Code: ${code}`); // Для тестування показуємо код

if (isExistingUser) {

// Користувач вже зареєстрований - відправляємо код підтвердження для входу

router.push({

pathname: '/(auth)/verify',

params: { email, isLogin: 'true' }

});

} else {

// Новий користувач - відправляємо код підтвердження для реєстрації

router.push({

pathname: '/(auth)/verify',

params: { email, isLogin: 'false' }

});

}

} catch (err) {

Alert.alert('Error', 'Failed to send verification code. Please try again.');

}

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<View style={styles.logoContainer}>

<View style={styles.logo}>

<FontAwesome name="gift" size={60} color="#4a90e2" />

</View>

<Text style={styles.appName}>Wishlist</Text>

<Text style={styles.tagline}>Your personal wishlist manager</Text>

</View>

<View style={styles.formContainer}>

<Text style={styles.welcomeText}>

{isExistingUser ? 'Welcome back!' : 'Create an account'}

</Text>

<Text style={styles.label}>Email</Text>

<View style={styles.inputContainer}>

<TextInput

style={styles.input}

value={email}

onChangeText={setEmail}

placeholder="Enter your email"

keyboardType="email-address"

autoCapitalize="none"

autoCorrect={false}

/>

{isCheckingUser && (

<ActivityIndicator size="small" color="#4a90e2" style={styles.indicator} />

)}

</View>

<TouchableOpacity

style={[styles.continueButton, loading && styles.disabledButton]}

onPress={handleContinue}

disabled={loading}

>

{loading ? (

<ActivityIndicator size="small" color="#fff" />

) : (

<Text style={styles.continueButtonText}>Continue</Text>

)}

</TouchableOpacity>

<TouchableOpacity

style={styles.switchButton}

onPress={() => setIsExistingUser(!isExistingUser)}

>

<Text style={styles.switchButtonText}>

{isExistingUser

? "Don't have an account? Sign up"

: "Already have an account? Log in"}

</Text>

</TouchableOpacity>

</View>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

logoContainer: {

alignItems: 'center',

marginTop: 80,

marginBottom: 40,

},

logo: {

width: 100,

height: 100,

borderRadius: 20,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 2 },

shadowOpacity: 0.2,

shadowRadius: 4,

elevation: 4,

marginBottom: 16,

},

appName: {

fontSize: 32,

fontWeight: 'bold',

color: '#333',

},

tagline: {

fontSize: 16,

color: '#666',

marginTop: 8,

},

formContainer: {

flex: 1,

paddingHorizontal: 24,

},

welcomeText: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 24,

color: '#333',

},

label: {

fontSize: 16,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

inputContainer: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

borderRadius: 8,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

marginBottom: 24,

},

input: {

flex: 1,

padding: 16,

fontSize: 16,

},

indicator: {

marginRight: 16,

},

continueButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

justifyContent: 'center',

height: 56,

},

disabledButton: {

backgroundColor: '#a0c5e8',

},

continueButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

switchButton: {

marginTop: 24,

alignItems: 'center',

},

switchButtonText: {

color: '#4a90e2',

fontSize: 16,

},

});

EOF

Оновлення екрану верифікації:

bash

Copy

cat > app/\(auth\)/verify.js << 'EOF'

import React, { useState, useRef, useEffect } from 'react';

import {

View,

Text,

StyleSheet,

TextInput,

TouchableOpacity,

KeyboardAvoidingView,

Platform,

Alert,

ActivityIndicator

} from 'react-native';

import { router, useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

import { useAuth } from '../context/AuthContext';

export default function VerifyScreen() {

const { email, isLogin } = useLocalSearchParams();

const [code, setCode] = useState(['', '', '', '']);

const inputRefs = useRef([]);

const [timer, setTimer] = useState(60);

const [canResend, setCanResend] = useState(false);

const { verifyCode, login, sendVerificationCode, loading, error, getError } = useAuth();

useEffect(() => {

if (error) {

Alert.alert('Error', getError());

}

}, [error]);

useEffect(() => {

// Запускаємо таймер для можливості повторної відправки

const interval = setInterval(() => {

setTimer(prevTimer => {

if (prevTimer <= 1) {

clearInterval(interval);

setCanResend(true);

return 0;

}

return prevTimer - 1;

});

}, 1000);

return () => clearInterval(interval);

}, []);

const handleCodeChange = (text, index) => {

// Перевіряємо, щоб був лише числовий ввід

if (text !== '' && !/^\d+$/.test(text)) {

return;

}

const newCode = [...code];

newCode[index] = text;

setCode(newCode);

// Автоматичний перехід до наступного поля

if (text.length === 1 && index < 3) {

inputRefs.current[index + 1].focus();

}

};

const handleVerify = async () => {

const fullCode = code.join('');

if (fullCode.length !== 4) {

Alert.alert('Error', 'Please enter the 4-digit code');

return;

}

try {

// Перевіряємо код

const isValid = await verifyCode(email, fullCode);

if (isValid) {

if (isLogin === 'true') {

// Користувач увійшов у систему

await login(email);

router.replace('/(tabs)');

} else {

// Користувач зареєструвався, перейдемо до налаштування профілю

router.push({

pathname: '/(auth)/setup-profile',

params: { email }

});

}

} else {

Alert.alert('Error', 'Invalid verification code');

}

} catch (err) {

Alert.alert('Error', 'Failed to verify code. Please try again.');

}

};

const handleResendCode = async () => {

if (!canResend) return;

try {

// Скидаємо таймер і відправляємо код знову

setTimer(60);

setCanResend(false);

// Генеруємо новий код і зберігаємо його

const newCode = await sendVerificationCode(email);

// В реальному додатку тут був би запит на сервер для відправки коду на пошту

Alert.alert('Code Resent', `New verification code: ${newCode}`); // Для тестування показуємо код

// Запускаємо таймер знову

const interval = setInterval(() => {

setTimer(prevTimer => {

if (prevTimer <= 1) {

clearInterval(interval);

setCanResend(true);

return 0;

}

return prevTimer - 1;

});

}, 1000);

} catch (err) {

Alert.alert('Error', 'Failed to resend code. Please try again.');

setCanResend(true);

}

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<View style={styles.content}>

<Text style={styles.title}>Verification Code</Text>

<Text style={styles.subtitle}>

Enter the 4-digit code sent to {email}

</Text>

<View style={styles.codeContainer}>

{code.map((digit, index) => (

<TextInput

key={index}

ref={el => inputRefs.current[index] = el}

style={styles.codeInput}

value={digit}

onChangeText={text => handleCodeChange(text, index)}

keyboardType="number-pad"

maxLength={1}

/>

))}

</View>

<TouchableOpacity

style={[styles.verifyButton, loading && styles.verifyButtonDisabled]}

onPress={handleVerify}

disabled={loading}

>

{loading ? (

<ActivityIndicator color="#fff" size="small" />

) : (

<Text style={styles.verifyButtonText}>Verify</Text>

)}

</TouchableOpacity>

<View style={styles.resendContainer}>

<Text style={styles.resendText}>Didn't receive the code? </Text>

<TouchableOpacity

onPress={handleResendCode}

disabled={!canResend || loading}

>

<Text style={[

styles.resendButton,

(!canResend || loading) && styles.resendButtonDisabled

]}>

{canResend ? 'Resend' : `Resend (${timer}s)`}

</Text>

</TouchableOpacity>

</View>

</View>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

paddingTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginLeft: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

content: {

flex: 1,

paddingHorizontal: 24,

paddingTop: 40,

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

color: '#333',

},

subtitle: {

fontSize: 16,

color: '#666',

marginBottom: 40,

},

codeContainer: {

flexDirection: 'row',

justifyContent: 'space-between',

marginBottom: 40,

},

codeInput: {

width: 70,

height: 70,

borderRadius: 8,

backgroundColor: 'white',

fontSize: 24,

fontWeight: 'bold',

textAlign: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height:

Retry

V

ти зупинився на Оновлення екрану верифікації:

Edit

Давайте продовжимо оновлення екрану верифікації:

bash

Copy

cat > app/\(auth\)/verify.js << 'EOF'

import React, { useState, useRef, useEffect } from 'react';

import {

View,

Text,

StyleSheet,

TextInput,

TouchableOpacity,

KeyboardAvoidingView,

Platform,

Alert,

ActivityIndicator

} from 'react-native';

import { router, useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

import { useAuth } from '../context/AuthContext';

export default function VerifyScreen() {

const { email, isLogin } = useLocalSearchParams();

const [code, setCode] = useState(['', '', '', '']);

const inputRefs = useRef([]);

const [timer, setTimer] = useState(60);

const [canResend, setCanResend] = useState(false);

const { verifyCode, login, sendVerificationCode, loading, error, getError } = useAuth();

useEffect(() => {

if (error) {

Alert.alert('Error', getError());

}

}, [error]);

useEffect(() => {

// Запускаємо таймер для можливості повторної відправки

const interval = setInterval(() => {

setTimer(prevTimer => {

if (prevTimer <= 1) {

clearInterval(interval);

setCanResend(true);

return 0;

}

return prevTimer - 1;

});

}, 1000);

return () => clearInterval(interval);

}, []);

const handleCodeChange = (text, index) => {

// Перевіряємо, щоб був лише числовий ввід

if (text !== '' && !/^\d+$/.test(text)) {

return;

}

const newCode = [...code];

newCode[index] = text;

setCode(newCode);

// Автоматичний перехід до наступного поля

if (text.length === 1 && index < 3) {

inputRefs.current[index + 1].focus();

}

};

const handleVerify = async () => {

const fullCode = code.join('');

if (fullCode.length !== 4) {

Alert.alert('Error', 'Please enter the 4-digit code');

return;

}

try {

// Перевіряємо код

const isValid = await verifyCode(email, fullCode);

if (isValid) {

if (isLogin === 'true') {

// Користувач увійшов у систему

await login(email);

router.replace('/(tabs)');

} else {

// Користувач зареєструвався, перейдемо до налаштування профілю

router.push({

pathname: '/(auth)/setup-profile',

params: { email }

});

}

} else {

Alert.alert('Error', 'Invalid verification code');

}

} catch (err) {

Alert.alert('Error', 'Failed to verify code. Please try again.');

}

};

const handleResendCode = async () => {

if (!canResend) return;

try {

// Скидаємо таймер і відправляємо код знову

setTimer(60);

setCanResend(false);

// Генеруємо новий код і зберігаємо його

const newCode = await sendVerificationCode(email);

// В реальному додатку тут був би запит на сервер для відправки коду на пошту

Alert.alert('Code Resent', `New verification code: ${newCode}`); // Для тестування показуємо код

// Запускаємо таймер знову

const interval = setInterval(() => {

setTimer(prevTimer => {

if (prevTimer <= 1) {

clearInterval(interval);

setCanResend(true);

return 0;

}

return prevTimer - 1;

});

}, 1000);

} catch (err) {

Alert.alert('Error', 'Failed to resend code. Please try again.');

setCanResend(true);

}

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<TouchableOpacity

style={styles.backButton}

onPress={() => router.back()}

>

<FontAwesome name="arrow-left" size={20} color="#4a90e2" />

</TouchableOpacity>

<View style={styles.content}>

<Text style={styles.title}>Verification Code</Text>

<Text style={styles.subtitle}>

Enter the 4-digit code sent to {email}

</Text>

<View style={styles.codeContainer}>

{code.map((digit, index) => (

<TextInput

key={index}

ref={el => inputRefs.current[index] = el}

style={styles.codeInput}

value={digit}

onChangeText={text => handleCodeChange(text, index)}

keyboardType="number-pad"

maxLength={1}

/>

))}

</View>

<TouchableOpacity

style={[styles.verifyButton, loading && styles.verifyButtonDisabled]}

onPress={handleVerify}

disabled={loading}

>

{loading ? (

<ActivityIndicator color="#fff" size="small" />

) : (

<Text style={styles.verifyButtonText}>Verify</Text>

)}

</TouchableOpacity>

<View style={styles.resendContainer}>

<Text style={styles.resendText}>Didn't receive the code? </Text>

<TouchableOpacity

onPress={handleResendCode}

disabled={!canResend || loading}

>

<Text style={[

styles.resendButton,

(!canResend || loading) && styles.resendButtonDisabled

]}>

{canResend ? 'Resend' : `Resend (${timer}s)`}

</Text>

</TouchableOpacity>

</View>

</View>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

paddingTop: 60,

},

backButton: {

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: 'white',

justifyContent: 'center',

alignItems: 'center',

marginLeft: 20,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.2,

shadowRadius: 2,

elevation: 2,

},

content: {

flex: 1,

paddingHorizontal: 24,

paddingTop: 40,

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 16,

color: '#333',

},

subtitle: {

fontSize: 16,

color: '#666',

marginBottom: 40,

},

codeContainer: {

flexDirection: 'row',

justifyContent: 'space-between',

marginBottom: 40,

},

codeInput: {

width: 70,

height: 70,

borderRadius: 8,

backgroundColor: 'white',

fontSize: 24,

fontWeight: 'bold',

textAlign: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

verifyButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

justifyContent: 'center',

height: 56,

},

verifyButtonDisabled: {

backgroundColor: '#a0c5e8',

},

verifyButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

resendContainer: {

flexDirection: 'row',

justifyContent: 'center',

marginTop: 24,

},

resendText: {

color: '#666',

fontSize: 16,

},

resendButton: {

color: '#4a90e2',

fontSize: 16,

fontWeight: 'bold',

},

resendButtonDisabled: {

color: '#999',

},

});

EOF

Тепер оновимо екран налаштування профілю:

bash

Copy

cat > app/\(auth\)/setup-profile.js << 'EOF'

import React, { useState } from 'react';

import {

View,

Text,

StyleSheet,

TextInput,

TouchableOpacity,

KeyboardAvoidingView,

Platform,

ScrollView,

Alert,

ActivityIndicator,

Image

} from 'react-native';

import { router, useLocalSearchParams } from 'expo-router';

import { FontAwesome } from '@expo/vector-icons';

import { useAuth } from '../context/AuthContext';

export default function SetupProfileScreen() {

const { email } = useLocalSearchParams();

const [name, setName] = useState('');

const [username, setUsername] = useState('');

const { register, loading, error, getError } = useAuth();

// Перевіряємо помилки

React.useEffect(() => {

if (error) {

Alert.alert('Error', getError());

}

}, [error]);

const handleFinish = async () => {

if (!name.trim()) {

Alert.alert('Error', 'Please enter your name');

return;

}

if (!username.trim()) {

Alert.alert('Error', 'Please choose a username');

return;

}

try {

// Створення нового користувача

const userData = {

email,

name,

username,

avatar: null, // В майбутньому можна додати завантаження аватара

};

await register(userData);

// Перенаправлення до основного додатку

router.replace('/(tabs)');

} catch (err) {

Alert.alert('Error', 'Failed to create account. Please try again.');

}

};

return (

<KeyboardAvoidingView

style={styles.container}

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<ScrollView contentContainerStyle={styles.scrollContent}>

<Text style={styles.title}>Set Up Your Profile</Text>

<Text style={styles.subtitle}>Let's create your profile</Text>

<View style={styles.avatarContainer}>

<View style={styles.avatar}>

<FontAwesome name="user" size={60} color="#ddd" />

<View style={styles.addPhotoButton}>

<FontAwesome name="camera" size={16} color="#fff" />

</View>

</View>

<Text style={styles.addPhotoText}>Add Photo</Text>

</View>

<View style={styles.formContainer}>

<Text style={styles.label}>Full Name</Text>

<TextInput

style={styles.input}

value={name}

onChangeText={setName}

placeholder="Enter your name"

/>

<Text style={styles.label}>Username</Text>

<TextInput

style={styles.input}

value={username}

onChangeText={setUsername}

placeholder="Choose a username"

autoCapitalize="none"

/>

<Text style={styles.label}>Email</Text>

<TextInput

style={[styles.input, styles.disabledInput]}

value={email}

editable={false}

/>

<TouchableOpacity

style={[styles.finishButton, loading && styles.disabledButton]}

onPress={handleFinish}

disabled={loading}

>

{loading ? (

<ActivityIndicator size="small" color="#fff" />

) : (

<Text style={styles.finishButtonText}>Finish Setup</Text>

)}

</TouchableOpacity>

</View>

</ScrollView>

</KeyboardAvoidingView>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

scrollContent: {

padding: 24,

paddingTop: 60,

},

title: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

subtitle: {

fontSize: 16,

color: '#666',

marginBottom: 24,

},

avatarContainer: {

alignItems: 'center',

marginBottom: 32,

},

avatar: {

width: 120,

height: 120,

borderRadius: 60,

backgroundColor: '#f0f0f0',

justifyContent: 'center',

alignItems: 'center',

position: 'relative',

marginBottom: 8,

},

addPhotoButton: {

position: 'absolute',

bottom: 0,

right: 0,

width: 40,

height: 40,

borderRadius: 20,

backgroundColor: '#4a90e2',

justifyContent: 'center',

alignItems: 'center',

borderWidth: 3,

borderColor: '#f8f8f8',

},

addPhotoText: {

fontSize: 16,

color: '#4a90e2',

},

formContainer: {

marginBottom: 30,

},

label: {

fontSize: 16,

fontWeight: 'bold',

marginBottom: 8,

color: '#333',

},

input: {

backgroundColor: 'white',

borderRadius: 8,

padding: 16,

fontSize: 16,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

marginBottom: 24,

},

disabledInput: {

backgroundColor: '#f5f5f5',

color: '#666',

},

finishButton: {

backgroundColor: '#4a90e2',

borderRadius: 8,

padding: 16,

alignItems: 'center',

justifyContent: 'center',

marginTop: 16,

height: 56,

},

disabledButton: {

backgroundColor: '#a0c5e8',

},

finishButtonText: {

color: 'white',

fontWeight: 'bold',

fontSize: 16,

},

});

EOF

Тепер нам потрібно оновити індексний файл для перенаправлення на відповідний екран:

bash

Copy

cat > app/index.js << 'EOF'

import React, { useEffect, useState } from 'react';

import { Redirect } from 'expo-router';

import { ActivityIndicator, View } from 'react-native';

import { isAuthenticated } from './services/authService';

export default function Index() {

const [loading, setLoading] = useState(true);

const [isLoggedIn, setIsLoggedIn] = useState(false);

useEffect(() => {

const checkAuth = async () => {

const authenticated = await isAuthenticated();

setIsLoggedIn(authenticated);

setLoading(false);

};

checkAuth();

}, []);

if (loading) {

return (

<View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>

<ActivityIndicator size="large" color="#4a90e2" />

</View>

);

}

// Якщо користувач автентифікований, перенаправляємо на головний екран

if (isLoggedIn) {

return <Redirect href="/(tabs)" />;

}

// Інакше перенаправляємо на екран входу/реєстрації

return <Redirect href="/(auth)/welcome" />;

}

EOF

Нарешті, оновімо головний файл \_layout.js для підключення нашого AuthProvider:

bash

Copy

cat > app/\_layout.js << 'EOF'

import React from 'react';

import { Stack } from 'expo-router';

import { AuthProvider } from './context/AuthContext';

export default function RootLayout() {

return (

<AuthProvider>

<Stack screenOptions={{ headerShown: false }}>

<Stack.Screen name="index" />

<Stack.Screen name="(tabs)" />

<Stack.Screen name="(modals)" />

<Stack.Screen name="(others)" />

<Stack.Screen name="(auth)" />

</Stack>

</AuthProvider>

);

}

EOF

Тепер у вас є повноцінна система автентифікації з:

Firebase для зберігання даних та синхронізації

Локальне кешування через AsyncStorage для швидкого доступу

Верифікація через код підтвердження (емуляція)

Зберігання профілів користувачів

Вхід існуючих користувачів

Для тестування використовуйте:

Запустіть додаток

Введіть емейл (новий або існуючий)

Отримайте код підтвердження (для тестування показуємо його в Alert)

Введіть код

Для нових користувачів налаштуйте профіль

Увійдіть у додаток

Для повноцінного функціонування потрібно створити проект в Firebase, отримати ключі та підставити їх у файл конфігурації.

Retry

V

(NOBRIDGE) ERROR Error creating user: [TypeError: Cannot read property 'email' of undefined]

(NOBRIDGE) ERROR Error checking user existence: [FirebaseError: Failed to get document because the client is offline.]

(NOBRIDGE) ERROR Error checking user existence: [FirebaseError: Failed to get document because the client is offline.]

(NOBRIDGE) ERROR Error checking user existence: [FirebaseError: Failed to get document because the client is offline.]

(NOBRIDGE) ERROR Error checking user existence: [FirebaseError: Failed to get document because the client is offline.]

Edit

Схоже, у нас виникають проблеми з підключенням до Firebase. Помилка вказує на дві основні проблеми:

Cannot read property 'email' of undefined - це вказує на проблему з передачею даних користувача

Failed to get document because the client is offline - це вказує на те, що клієнт Firebase не може підключитися до сервера

Давайте виправимо ці проблеми:

Перше, що потрібно зробити - це правильно налаштувати конфігурацію Firebase. Вам потрібно створити проект в Firebase Console, отримати конфігураційні ключі та оновити файл конфігурації:

bash

Copy

cat > app/config/firebase.js << 'EOF'

import { initializeApp } from 'firebase/app';

import { getAuth, connectAuthEmulator } from 'firebase/auth';

import { getFirestore, connectFirestoreEmulator } from 'firebase/firestore';

import { getStorage, connectStorageEmulator } from 'firebase/storage';

// Firebase конфігурація

// ПРИМІТКА: Ці ключі мають бути замінені вашими власними ключами з Firebase Console

const firebaseConfig = {

apiKey: "YOUR\_API\_KEY", // Замініть на ваш ключ

authDomain: "your-app.firebaseapp.com",

projectId: "your-app-id",

storageBucket: "your-app.appspot.com",

messagingSenderId: "YOUR\_SENDER\_ID",

appId: "YOUR\_APP\_ID"

};

// Ініціалізуємо Firebase

const app = initializeApp(firebaseConfig);

// Експортуємо необхідні сервіси

export const auth = getAuth(app);

export const db = getFirestore(app);

export const storage = getStorage(app);

// Для розробки можна використовувати локальні емулятори Firebase

if (\_\_DEV\_\_) {

// Розкоментуйте ці рядки, якщо ви використовуєте локальні емулятори Firebase

// connectAuthEmulator(auth, 'http://localhost:9099');

// connectFirestoreEmulator(db, 'localhost', 8080);

// connectStorageEmulator(storage, 'localhost', 9199);

}

export default app;

EOF

Але для швидкого тестування, давайте модифікуємо наш код, щоб він працював з чистим AsyncStorage, без вимоги підключення до Firebase:

bash

Copy

cat > app/services/authService.js << 'EOF'

import AsyncStorage from '@react-native-async-storage/async-storage';

// Ключі для локального сховища

const USER\_DATA\_KEY = 'wishlist\_user\_data';

const USERS\_KEY = 'wishlist\_users';

const VERIFICATION\_CODES\_KEY = 'wishlist\_verification\_codes';

// Генерування коду підтвердження

export const generateVerificationCode = () => {

return Math.floor(1000 + Math.random() \* 9000).toString();

};

// Збереження коду підтвердження

export const saveVerificationCode = async (email, code) => {

try {

const codesJson = await AsyncStorage.getItem(VERIFICATION\_CODES\_KEY) || '{}';

const codes = JSON.parse(codesJson);

codes[email] = {

code,

timestamp: Date.now()

};

await AsyncStorage.setItem(VERIFICATION\_CODES\_KEY, JSON.stringify(codes));

return true;

} catch (error) {

console.error('Error saving verification code:', error);

return false;

}

};

// Перевірка коду підтвердження

export const verifyCode = async (email, code) => {

try {

const codesJson = await AsyncStorage.getItem(VERIFICATION\_CODES\_KEY) || '{}';

const codes = JSON.parse(codesJson);

const savedCode = codes[email];

// Перевіряємо наявність коду та його актуальність (30 хвилин)

if (savedCode && savedCode.code === code &&

Date.now() - savedCode.timestamp < 30 \* 60 \* 1000) {

return true;

}

return false;

} catch (error) {

console.error('Error verifying code:', error);

return false;

}

};

// Перевірка, чи існує користувач

export const userExists = async (email) => {

try {

const usersJson = await AsyncStorage.getItem(USERS\_KEY) || '{}';

const users = JSON.parse(usersJson);

return !!users[email];

} catch (error) {

console.error('Error checking user existence:', error);

return false;

}

};

// Створення нового користувача

export const createUser = async (userData) => {

try {

const { email, name, username } = userData;

// Отримуємо існуючих користувачів

const usersJson = await AsyncStorage.getItem(USERS\_KEY) || '{}';

const users = JSON.parse(usersJson);

// Додаємо нового користувача

users[email] = {

...userData,

wishlists: [],

friends: [],

friendRequests: [],

createdAt: new Date().toISOString(),

};

// Зберігаємо оновлений список користувачів

await AsyncStorage.setItem(USERS\_KEY, JSON.stringify(users));

// Зберігаємо поточного користувача

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify(userData));

return userData;

} catch (error) {

console.error('Error creating user:', error);

throw error;

}

};

// Вхід користувача

export const loginUser = async (email) => {

try {

// Отримуємо дані користувача

const usersJson = await AsyncStorage.getItem(USERS\_KEY) || '{}';

const users = JSON.parse(usersJson);

const userData = users[email];

if (!userData) {

throw new Error('User not found');

}

// Зберігаємо поточного користувача

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify(userData));

return userData;

} catch (error) {

console.error('Error logging in:', error);

throw error;

}

};

// Вихід користувача

export const logoutUser = async () => {

try {

await AsyncStorage.removeItem(USER\_DATA\_KEY);

return true;

} catch (error) {

console.error('Error logging out:', error);

return false;

}

};

// Отримання даних поточного користувача

export const getCurrentUser = async () => {

try {

const userData = await AsyncStorage.getItem(USER\_DATA\_KEY);

return userData ? JSON.parse(userData) : null;

} catch (error) {

console.error('Error getting current user:', error);

return null;

}

};

// Перевірка автентифікації користувача

export const isAuthenticated = async () => {

try {

const userData = await AsyncStorage.getItem(USER\_DATA\_KEY);

return !!userData;

} catch (error) {

console.error('Error checking authentication:', error);

return false;

}

};

// Оновлення даних користувача

export const updateUserData = async (userData) => {

try {

const { email } = userData;

// Отримуємо існуючих користувачів

const usersJson = await AsyncStorage.getItem(USERS\_KEY) || '{}';

const users = JSON.parse(usersJson);

// Перевіряємо, чи існує користувач

if (!users[email]) {

return false;

}

// Оновлюємо дані користувача

users[email] = {

...users[email],

...userData,

updatedAt: new Date().toISOString(),

};

// Зберігаємо оновлений список користувачів

await AsyncStorage.setItem(USERS\_KEY, JSON.stringify(users));

// Оновлюємо дані поточного користувача, якщо це він

const currentUserJson = await AsyncStorage.getItem(USER\_DATA\_KEY);

if (currentUserJson) {

const currentUser = JSON.parse(currentUserJson);

if (currentUser.email === email) {

await AsyncStorage.setItem(USER\_DATA\_KEY, JSON.stringify({

...currentUser,

...userData

}));

}

}

return true;

} catch (error) {

console.error('Error updating user data:', error);

return false;

}

};

EOF

Також оновимо профіль користувача, щоб показувати дані, які були введені при реєстрації:

bash

Copy

cat > app/\(tabs\)/profile.js << 'EOF'

import React, { useState, useEffect } from 'react';

import { View, Text, StyleSheet, TouchableOpacity, FlatList, Image, Alert } from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

import { Link, router } from 'expo-router';

import { useAuth } from '../context/AuthContext';

export default function ProfileScreen() {

const [activeTab, setActiveTab] = useState('info');

const { user, logout } = useAuth();

const handleLogout = async () => {

Alert.alert(

'Logout',

'Are you sure you want to logout?',

[

{

text: 'Cancel',

style: 'cancel'

},

{

text: 'Logout',

onPress: async () => {

await logout();

router.replace('/(auth)/welcome');

}

}

]

);

};

if (!user) {

return (

<View style={[styles.container, { justifyContent: 'center', alignItems: 'center' }]}>

<Text>Loading profile...</Text>

</View>

);

}

return (

<View style={styles.container}>

<View style={styles.header}>

<View style={styles.profileAvatarContainer}>

<View style={styles.profileAvatar}>

<Text style={styles.profileAvatarText}>{user.name ? user.name.charAt(0) : 'U'}</Text>

</View>

</View>

<Text style={styles.profileName}>{user.name || 'User'}</Text>

<Text style={styles.profileUsername}>@{user.username || 'username'}</Text>

<View style={styles.statsContainer}>

<View style={styles.statItem}>

<Text style={styles.statNumber}>0</Text>

<Text style={styles.statLabel}>Wishlists</Text>

</View>

<View style={styles.statItem}>

<Text style={styles.statNumber}>0</Text>

<Text style={styles.statLabel}>Friends</Text>

</View>

</View>

</View>

<View style={styles.tabContainer}>

<TouchableOpacity

style={[styles.tab, activeTab === 'info' && styles.activeTab]}

onPress={() => setActiveTab('info')}

>

<Text style={[styles.tabText, activeTab === 'info' && styles.activeTabText]}>Info</Text>

</TouchableOpacity>

<TouchableOpacity

style={[styles.tab, activeTab === 'friends' && styles.activeTab]}

onPress={() => setActiveTab('friends')}

>

<Text style={[styles.tabText, activeTab === 'friends' && styles.activeTabText]}>Friends</Text>

</TouchableOpacity>

</View>

{activeTab === 'info' ? (

<View style={styles.infoContainer}>

<View style={styles.infoItem}>

<FontAwesome name="envelope" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>{user.email}</Text>

</View>

<View style={styles.infoItem}>

<FontAwesome name="calendar" size={18} color="#666" style={styles.infoIcon} />

<Text style={styles.infoText}>Member since {new Date().toLocaleDateString()}</Text>

</View>

<TouchableOpacity style={styles.logoutButton} onPress={handleLogout}>

<FontAwesome name="sign-out" size={18} color="#e74c3c" style={styles.logoutIcon} />

<Text style={styles.logoutText}>Logout</Text>

</TouchableOpacity>

</View>

) : (

<View style={styles.friendsContainer}>

<Link href="/modals/search" asChild>

<TouchableOpacity style={styles.addFriendButton}>

<FontAwesome name="search" size={18} color="#fff" style={styles.addFriendIcon} />

<Text style={styles.addFriendText}>Find Friends</Text>

</TouchableOpacity>

</Link>

<Text style={styles.emptyMessage}>You have no friends yet</Text>

</View>

)}

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#f8f8f8',

},

header: {

backgroundColor: '#4a90e2',

padding: 20,

alignItems: 'center',

paddingTop: 60,

},

profileAvatarContainer: {

marginBottom: 10,

},

profileAvatar: {

width: 100,

height: 100,

borderRadius: 50,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

},

profileAvatarText: {

fontSize: 40,

fontWeight: 'bold',

color: '#4a90e2',

},

profileName: {

color: '#fff',

fontSize: 24,

fontWeight: 'bold',

},

profileUsername: {

color: '#e6e6e6',

fontSize: 16,

marginTop: 4,

},

statsContainer: {

flexDirection: 'row',

marginTop: 20,

width: '60%',

justifyContent: 'space-around',

},

statItem: {

alignItems: 'center',

},

statNumber: {

color: '#fff',

fontSize: 20,

fontWeight: 'bold',

},

statLabel: {

color: '#e6e6e6',

fontSize: 14,

marginTop: 4,

},

tabContainer: {

flexDirection: 'row',

backgroundColor: 'white',

marginBottom: 10,

},

tab: {

flex: 1,

alignItems: 'center',

paddingVertical: 12,

borderBottomWidth: 2,

borderBottomColor: 'transparent',

},

activeTab: {

borderBottomColor: '#4a90e2',

},

tabText: {

fontSize: 16,

color: '#999',

},

activeTabText: {

color: '#4a90e2',

fontWeight: 'bold',

},

infoContainer: {

padding: 20,

},

infoItem: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginBottom: 10,

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

infoIcon: {

marginRight: 15,

width: 20,

textAlign: 'center',

},

infoText: {

fontSize: 16,

color: '#333',

},

friendsContainer: {

padding: 20,

},

addFriendButton: {

backgroundColor: '#4a90e2',

flexDirection: 'row',

alignItems: 'center',

justifyContent: 'center',

padding: 12,

borderRadius: 8,

marginBottom: 20,

},

addFriendIcon: {

marginRight: 8,

},

addFriendText: {

color: 'white',

fontSize: 16,

fontWeight: 'bold',

},

emptyMessage: {

textAlign: 'center',

color: '#999',

marginTop: 20,

fontSize: 16,

},

logoutButton: {

flexDirection: 'row',

alignItems: 'center',

backgroundColor: 'white',

padding: 15,

borderRadius: 8,

marginTop: 20,

justifyContent: 'center',

shadowColor: '#000',

shadowOffset: { width: 0, height: 1 },

shadowOpacity: 0.1,

shadowRadius: 2,

elevation: 1,

},

logoutIcon: {

marginRight: 8,

},

logoutText: {

fontSize: 16,

color: '#e74c3c',

fontWeight: 'bold',

},

});

EOF

Тепер ваш додаток повинен працювати повністю локально, без необхідності підключення до Firebase. Це дозволить вам протестувати функціональність аутентифікації без налаштування реального Firebase проекту.

Коли ви будете готові перейти на реальний Firebase, вам потрібно буде:

Створити проект у Firebase Console

Отримати ключі конфігурації

Замінити тестові ключі у файлі app/config/firebase.js

Розкоментувати і використовувати реальні сервіси Firebase

Давайте перезапустимо додаток і подивимось, чи працює аутентифікація тепер:

bash

Copy

npx expo start --clear

Retry

Claude can make mistakes. Please double-check responses.

Давай поки цього не робити, я думаю що проблема в тому що я не замінив тестові ключі у файлі `app/config/firebase.js ` Я вже створив проект файрбейс, що потрібно робити далі?

Файл не вибрано

3.7 Sonnet

Choose style

Use shift + return for new line

Chat controls

3.7 Sonnet

Our most intelligent model yet Learn more

Content

Знімок екрана 20250313 о 23.20.06.png

Знімок екрана 20250313 о 23.20.06.png

Знімок екрана 20250313 о 23.25.31.png

Знімок екрана 20250313 о 23.25.31.png

IMG\_0E4628CD6C151.jpeg

IMG\_0E4628CD6C151.jpeg

Знімок екрана 20250313 о 23.40.13.png

Знімок екрана 20250313 о 23.40.13.png

Знімок екрана 20250313 о 23.56.52.png

Знімок екрана 20250313 о 23.56.52.png

Знімок екрана 20250314 о 00.00.57.png

Знімок екрана 20250314 о 00.00.57.png

IMG\_DFA77B0A53481.jpeg

IMG\_DFA77B0A53481.jpeg

IMG\_EF7F79E25BB11.jpeg

IMG\_EF7F79E25BB11.jpeg

IMG\_31AACAB7A7A31.jpeg

IMG\_31AACAB7A7A31.jpeg

IMG\_4F61CD3D1B2D1.jpeg

IMG\_4F61CD3D1B2D1.jpeg

IMG\_33BB6FE3AD6E1.jpeg

IMG\_33BB6FE3AD6E1.jpeg

Pasted content

15.29 KB •354 lines

Conversation in Ukrainian - Claude