Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено

Завідувач кафедри Стіренко С. Г. _____

__ _____ 2023 р.

Дипломний проєкт

на здобуття ступеню бакалавра за освітньо-професійною програмою «Комп'ютерні системи та мережі» спеціальності 123 «Комп'ютерна інженерія»

на тему «Двоплатформний мобільний додаток для організації навчального процесу студентів»

Виконав: студент 4-го курсу, групи ІВ-92, Бугайов Віктор Сергійович

Керівник: асистент Шульга Максим Володимирович

Консультант: доцент, к. т. н. Волокита Артем Миколайович

Рецензент: доцент, к. т. н. Галушко Дмитро Олександрович

Засвідчую, що цей дипломний проєкт не має запозичень зі праць інших авторів без відповідних посилань.

Студент _____

Київ, 2023

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти: перший (бакалавр)

Освітньо-професійна програма: «Комп'ютерні мережі та системи»

спеціальності: 123 «Комп'ютерна інженерія»

Затверджую

Завідувач кафедри Стіренко С. Г. _____

__ _____ 2023 р.

Завдання

на бакалаврський дипломний проєкт студента

Бугайова Віктора Сергійовича

Тема роботи: Двоплатформний мобільний додаток для організації навчального процесу студентів

Дата видання завдання: 26 грудня 2022 р.

Термін здання роботи: 20 червня 2023 р.

Вихідні дані до проєкту: теоретичні дані, технічна документація

Зміст пояснювальної записки:

1. Альтернативи
2. Багатоплатформна розробка
3. Технології
4. Додаток Подійни́к

Перелік графічного матеріалу: функціональна схема (дерево станів додатка), структурна схема (структура бази даних)

Консультанти розділів роботи:

Розділ	Консультант	Завдання видав	Завдання прийняв
		підпис, дата	
Альтернативи	Шульга М. В.		
Багатоплатформна розробка			
Технології			
Додаток Подійни́к			

Календарний план

#	Етап	Термін	Примітки
1	Вивчення літератури.	17.04.23 – 21.04.23	
2	Вивчення і аналіз існуючих аналогів.	24.04.23 – 29.04.23	
3	Вибір технологій для розробки.	01.05.23 – 05.05.23	
4	Розробка алгоритмів для проєкту.	08.05.23 – 12.05.23	
5	Програмування та загальне налагодження системи.	15.05.23 – 19.05.23	
6	Оформлення пояснювальної записки	01.06.23 – 20.06.23	
7	Захист програмного продукту	08.06.23	
8	Передзахист	21.06.23	
9	Захист	22.06.23	

Студент Бугайов В. С. _____

Керівник Шульга М. В. _____

Анотація

У бакалаврському дипломному проєкті створено двоплатформний мобільний додаток «Подійни́к». Він призначений для спрощення українським студентам організації навчального процесу.

Пандемія COVID-19 у 2020 році раптово наклала обмеження на можливість студентів перебувати в університетах. Викладачі були змушені швидко винайти новий спосіб викладання, а студенти – навчання. Цей процес відбувався у чатах комунікаційних додатків, але породжував надто багато інформації, щоб користуватися нею у них. Наслідок – усередині групи студенти не оволодівають важливою інформацією одночасно, і це не гарантованим узагалі, оскільки у чаті легко загубити щось.

Власне, цей проєкт є розв’язанням цієї проблеми, і просто спрощенням слідкування за поточним у спокійний час. Він дозволяє студентам групи створити простір для зберігання подій та інформації, зручного її поширення один між одним.

Abstract

This paper for a Bachelor’s Degree describes the creation of the “Podiinyk” cross-platform mobile app. It is supposed to make it easier for Ukrainian students to manage their learning process.

The COVID-19 pandemic abruptly put harsh limitations on the ability of students to be present in universities. Teaching personnel were faced with a challenge to quickly come up with a new approach to teach, and students with an approach to learn. This process was happening in communication app chats, while creating too much information for it to be digested from them. The consequence is that within a group the students do not obtain information at the same time, and it is not even guaranteed to start with, since it is easy to lose a message in chat.

This project is the solution for the emerging problem, besides just being a helper in tracking everything important when nothing crazy is happening.

Формат	Позначення	Назва	Кількість сторінок
А4		Завдання на дипломний проєкт	3
А4	ІАЛЦ.467100.001 ОА	Опис альбому	1
А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	4
А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	57
А4	ІАЛЦ.467100.004 Д1	Дерево станів додатка	1
А4	ІАЛЦ.467100.005 Д2	Структура бази даних	1
А4	ІАЛЦ.467100.006 Д3	Код додатка	68

Змн.	Арк.	# документа	Підпис	Дата	ІАЛЦ.467100.001 ОА		
Розробив		Бугайов В. С.			Двоплатформний мобільний додаток для організації навчального процесу студентів	Літ.	Арк.	Аркушів
Перевірив		Шульга М. В.					1	1
						НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІВ-92		
Нормоконтроль		Волокита А. М.						
Затвердив		Стіренко С. Г.			Опис альбому			

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Двоплатформний мобільний додаток
для організації навчального процесу студентів»

Київ, 2023

ЗМІСТ

					ІАЛЦ.467100.002 ТЗ			
Змн.	Арк.	# документа	Підпис	Дата				
Розробив		Бугайов В. С.			Двоплатформний мобільний додаток для організації навчального процесу студентів Технічне завдання	Літ.	Арк.	Аркушів
Перевірив		Шульга М. В.					1	4
						НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІВ-92		
Нормоконтроль		Волокита А. М.						
Затвердив		Стіренко С. Г.						

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання розроблено для написання дипломної роботи на тему «Двоплатформний мобільний додаток для організації навчального процесу студентів». Передбачено, що він буде використовуватися студентами будь-якого вищого навчального закладу України. Це можливо завдяки правильно обраному рівню абстракції, що не розглядає деталей будь-якого одного ВНЗ.

2. ОСНОВА ДЛЯ РОЗРОБКИ

Основою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр програмної інженерії», що затверджено кафедрою обчислювальної техніки Національного Технічного Університету України «Київського Політехнічного Інституту ім. Ігоря Сікорського».

3. МЕТА РОЗРОБКИ

Метою розробки є спрощення студентам особистої організаційної складової навчання. Створений для цього додаток дозволяє зручніше слідкувати за подіями групи, ділитися та зберігати важливу інформацію.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є досвід буття студентом у спокійний і складний через пандемію та війну час, документація інструментів для створення мобільних додатків, зокрема Flutter, документація мови програмування Dart, документація сервісів платформи Firebase, зокрема Authentication та Cloud Firestore.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до додатка:

- Двоплатформність (Android, iOS)
- Мінімалістичність
- Надійність
- Захищеність від некоректної поведінки користувача
- Надійне зберігання даних у хмарному середовищі

5.2 Вимоги до програмного забезпечення:

- ОС Android версії щонайменше 4.4 або ОС iOS

5.3 Вимоги до апаратного забезпечення:

- Здатність доступу до інтернету
- Сенсорний екран
- Об’єм оперативної пам’яті від 4 ГБ
- Вільний об’єм постійної пам’яті від 100 МБ

6. ЕТАПИ РОЗРОБКИ

Етап	Термін
Вивчення літератури.	17.04.23 – 21.04.23
Вивчення і аналіз існуючих аналогів.	24.04.23 – 29.04.23
Вибір технологій для розробки.	01.05.23 – 05.05.23
Розробка алгоритмів для проєкту.	08.05.23 – 12.05.23
Програмування та загальне налагодження системи.	15.05.23 – 19.05.23
Оформлення пояснювальної записки	01.06.23 – 07.06.23
Захист програмного продукту	08.06.23
Передзахист	09.06.23
Захист	22.06.23

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Двоплатформний мобільний додаток
для організації навчального процесу студентів»

Київ, 2023

ЗМІСТ

СКОРОЧЕННЯ

ОС – операційна система

ПЗ – програмне забезпечення

AI (Artificial Intelligence) – штучний інтелект

PWA (Progressive Web Application) – прогресивний веб-додаток

UX (User Experience) – взаємодія користувача з додатком

CPU (Central Processing Unit) – головний обчислювальний процесор

GPU (Central Graphics Unit) – головний графічний процесор

MVP (Minimum Viable Product) – мінімальний життєздатний продукт

SDK (Software Development Kit) – набір інструментів для розробки ПЗ

BaaS (Backend as a Service) – *backend* як сервіс

1. АЛЬТЕРНАТИВИ

1.1. Google Keep

Рисунок 1.1 – логотип Google Keep

Google Keep – система для створювання простих нотаток. Вона представлена веб-додатком, а також мобільними додатками для Android та iOS.

Google Keep пропонує різні інструменти для створювання нотаток:

- текст
- списки
- зображення
- аудіо

Можна автоматично отримати текст із голосових файлів, а також із зображень за допомогою оптичного розпізнавання символів.

Інтерфейс дозволяє переглядати нотатки одним або кількома стовпцями. Якщо якась із них часто використовується, буде зручно скористатися можливістю закріплювати їх нагорі. Нотатки можна позначати кольором і додавати до них позначки, щоб їх можна було зручно розділяти. Також над ними можна працювати одночасно разом з іншими.

1.2. Notion

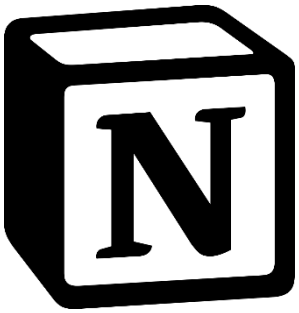


Рисунок 1.2 – логотип Notion

Notion – веб-додаток для підвищення продуктивності та створення нотаток. Він надає організовувальні інструменти, зокрема керування завданнями, відстеження проєктів, списки справ, додавання закладок тощо. Комп'ютерні та мобільні додатки, доступні для Windows, macOS, Android та iOS, надають додаткові *offline*-функції. Користувачі можуть створювати власні шаблони, вставляти відео, а також співпрацювати з іншими наживо.

1.2.1. Блоки

Усе в Notion складається з блоків. Це включає текст, зображення, таблиці та навіть самі сторінки. Це дозволяє користувачам легко налаштовувати сторінки, додаючи та переміщуючи блоки. Знедавна у програмі з'явився «синхронізований блок», копії якого мають єдиний стан на всіх сторінках.

1.2.2. Бази даних

Однією з найцікавіших особливостей Notion є бази даних. Вони використовуються для зберігання інформації і можуть містити будь-яку кількість рядків і стовпців. Бази даних мають щонайменше два стовпці: «Назва» та «Теги». Користувачі можуть додавати більше стовпців, як дата, прапорець, вибір кількох елементів, URL-адреса тощо. Бази даних можуть знаходитися на сторінках разом з іншим вмістом або мати окремі сторінки.

1.2.3. Шаблони

Notion надає можливість створювати шаблони та ділитися ними з іншими. Існує галерея, де користувачі можуть переглядати шаблони, створені іншими творцями. Однак не всі ці шаблони можна використовувати безкоштовно. Деякі автори також мають прибуток від продажу шаблонів Notion.

1.2.4. Notion AI

Notion AI базується на штучному інтелекті, який дозволяє прогнозувати результати, генерувати та розв’язувати математичні задачі. Цей штучний інтелект може автоматизовувати складні завдання, писати краще, а також допомагати користувачам отримувати стислий зміст тексту або нарис з одного рядка.

1.3. Google Classroom

Рисунок 1.3 – логотип Google Classroom

Google Classroom – безкоштовна платформа *змішаного навчання* [6], розроблена компанією Google для навчальних закладів, що спрощує створювання, розповсюджування та оцінювання завдань. Мета платформи – упорядкувати та пришвидшити процес обміну файлами між викладачами та студентами. Станом на 2021 рік приблизно 150 мільйонів користувачів використовують Google Classroom.

Студентів можна запросити приєднатися до класу за допомогою приватного коду або автоматично імпортувати з домену школи. Кожний клас створює окрему папку на Google Drive користувачів, куди учень може надсилати роботу для оцінювання вчителем. Вчителі можуть відстежувати прогрес кожного учня, переглядаючи історію оглядів документа, і після оцінення вчителі можуть повернути роботу разом із оцінкою та коментарями.

Google Classroom доступна як веб-додаток, так і мобільні додатки для систем Android та iOS.

1.3.1. Завдання

Завдання зберігаються та оцінюються у програмах Google для роботи з документами. Надсилання документів викладачеві працює не так, що студент завантажує їх на свій Google Drive і надає йому доступ до них. Радше, надсилаючи їх викладачеві через Google Classroom, вони зберігаються на Google Drive студента зі зручним інтерфейсом для перегляду для викладача.

Викладачі мають можливість створювати завдання в різних шаблонах і форматах із різними параметрами доступності, такими як дозволи на перегляд, редагування та коментарі. Виконане завдання може бути здане для оцінки; викладач має можливість надати відгук. Студент може додавати інші документи зі свого Google Drive до свого завдання. Або іншого джерела – документи буде завантажено на Google Drive автоматично.

1.3.2. Оцінювання

Google Classroom підтримує різні схеми оцінювання. Здане завдання може бути оцінене вчителем та повернене з відгуком, що дозволяє студентові змінювати свою роботу перед остаточним зданням. Здане завдання може редагувати лише вчитель.

1.3.3. Комунікування

Вчителі можуть публікувати оголошення у потоці класу (*class stream*), до яких учасники класу (учні) можуть залишати коментарі. Учні також можуть публікувати дописи у потоці класу, але роль модераторів залишається за викладачами. До оголошень і дописів можна додавати різні види файлів із продуктів Google, такі як відео з YouTube і файли з Google Drive. Gmail також надає вчителям можливість надіслати листа одному або кільком студентам класу.

1.3.4. Звіт про схожість (Originality Report)

Originality Report є вбудованим інструментом для виявляння плагіату, доступ до якого мають як студенти, так і викладачі. Викладачі можуть переглядати звіт про схожість, що дозволяє їм здійснювати перевірку на академічну доброчесність.

1.3.5. Архівування класів

Google Classroom дозволяє викладачам архівувати курси в кінці семестру або року. Коли клас архівується, він видаляється з домашньої сторінки та переходить до розділу заархівованих класів. Викладачі та студенти можуть переглядати такі класи, але не можуть вносити змін, якщо курс не відновлено.

ВИСНОВОК ДО РОЗДІЛУ 1

Розглянуті альтернативи є системами для створювання нотаток, зберігання інформації, підвищення продуктивності, спрощення обміну файлами між викладачами та студентами. Кожна з них чудово допомагає користувачам на своєму рівні.

Проте я намагаюся вирішити проблему не лише зберігання інформації. Я прагну створити систему спеціально для навчання у групах. Головна ідея – створити спільний простір для всіх студентів групи, де будь-хто з них може додавати нове. Із розглянутих альтернатив лише Google Classroom має цей аспект спільного простору, проте ця система відрізняється від проєкту, який я прагну створити. По-перше, цей спільний простір є спільним і з викладачами. По-друге, він передбачений радше як простір для одного предмета, а не для групи. Коли я хочу зібрати усе, за чим потрібно слідкувати, і все, що потрібно пам’ятати, в одному місці, і для всієї групи, без викладачів.

Тому я вважаю створення нового додатка доречним. Не кажучи про знання та досвід, які я отримаю в цій подорожі процесу розробки.

2. БАГАТОПЛАТФОРМНА РОЗРОБКА

2.1. Мотивація

Сьогодні багато компаній мають необхідність створювати мобільні додатки для кількох платформ, зокрема для Android та iOS. Це стало причиною появи інструментів для багатоплатформної мобільної розробки. І сьогодні цей напрямок є одним із найпопулярніших у розробці програмного забезпечення.

За даними платформи Statista [2], у першому кварталі 2021 року було доступно 3.48 мільйона мобільних додатків у Google Play Store і 2.22 мільйона додатків у App Store, причому на Android та iOS тепер припадає 99% світового ринку мобільних ОС.

Як створити мобільний додаток, який може охопити обидві аудиторії: Android та iOS?

2.2. Підходи до мобільної розробки

2.2.1. Окремі нативні додатки для кожної ОС

Створюючи нативний додаток, розробники створюють програму для певної ОС та покладаються на інструменти та мови програмування, розроблені спеціально для цієї платформи: Kotlin або Java для Android, Swift або Objective-C для iOS.

Ці інструменти та мови надають доступ до функцій і можливостей конкретної ОС і дозволяють створювати адаптивні програми з інтуїтивно зрозумілими інтерфейсами. Але якщо потрібно охопити як аудиторію Android, так і iOS, доведеться створювати окремі програми, а для цього потрібно багато часу та зусиль.

2.2.2. Прогресивні веб-додатки (PWAs)

Прогресивні веб-додатки поєднують у собі функції мобільних програм із рішеннями, що використовуються у веб-розробці. Грубо кажучи, вони пропонують поєднання веб-сайту та мобільного додатка. *PWAs* створюють за допомогою веб-технологій, таких як JavaScript, HTML, CSS і WebAssembly.

Веб-додатки не вимагають додаткових збірок чи розповсюдження, а можуть бути просто опубліковані онлайн. Вони доступні через браузер і їх не потрібно встановлювати через Google Play або App Store.

Недолік тут полягає в тому, що користувач не може використовувати всіх функцій свого пристрою. Наприклад, календар, контакти, телефон та інші ресурси. Це призводить до обмеження UX. З точки зору продуктивності, нативні програми мають перевагу.

2.2.3. Багатоплатформні додатки

Багатоплатформні додатки створюються працювати ідентично на різних мобільних платформах. Інструменти для багатоплатформної розробки дозволяють писати код, що його частина або навіть він увесь може бути спільним. Це означає, що розробники можуть створювати та використовувати ресурси, які працюють як на Android, так і на iOS, без необхідності створювати їх окремо для кожної платформи.

Перевагою цього підходу є ефективність в об’ємі роботи, часі та вартості.

2.2.4. Гібридні додатки

Переглядаючи різні джерела, можна помітити, що деякі люди використовують терміни «багатоплатформна мобільна розробка» та «гібридна мобільна розробка» як синоніми. Однак, це не є так.

У багатоплатформних додатках розробники можуть написати код один раз і використовувати його на різних платформах. З іншого боку, розробка гібридних програм – підхід, який поєднує нативні та веб-технології. Він передбачає використання коду, написаного мовою веб-розробки, як HTML, CSS або JavaScript, у нативній програмі. Це можна зробити за допомогою фреймворків, таких як Ionic Capacitor і Apache Cordova, використовуючи додаткові інструменти для отримання доступу до нативних функцій платформ.

Єдина подібність між багатоплатформною та гібридною розробками полягає у здатності коду бути спільним. З точки зору продуктивності, гібридні програми не зрівняються з нативними. Оскільки гібридні додатки працюють на єдиній кодовій базі, деякі їх елементи можуть бути напрямлені на певну ОС і погано функціонувати на інших.

2.3. Нативна розробка проти багатоплатформної

Дискусія навколо нативної та багатоплатформної розробки залишаються невирішеними в технічній спільноті. Обидві технології постійно розвиваються та мають свої переваги й обмеження.

Деякі експерти досі обирають нативну мобільну розробку перед багатоплатформними підходами, визначаючи вищу продуктивність і кращий UX як одні з найважливіших переваг.

Однак, багатьом сучасним компаніям потрібно зменшити час виходу додатків на ринок і витрати на розробку, при цьому маючи на меті осягати користувачів як Android, так і iOS.

Вибір підходу до мобільної розробки залежить від багатьох факторів: бізнес-вимог, мети тощо. Як і будь-яке інше рішення, багатоплатформна мобільна розробка має переваги та недоліки.

2.4. Переваги багатоплатформної розробки

2.4.1. Можливість коду бути використаним повторно

Завдяки багатоплатформному програмуванню мобільним розробникам не потрібно писати новий код для кожної ОС. Використання єдиної кодової бази дозволяє їм зменшити час на виконання повторюваних завдань, таких як виклики API, зберігання даних, серіалізація даних і впровадження аналітики.

2.4.2. Економія часу

Завдяки можливості багаторазового використання коду, багатоплатформні програми мають менше коду, а коли мова про програмування, як то кажуть, «less code is more». Час економиться завдяки тому, що не потрібно писати так багато коду, як коли створюючи додатки окремо для кожної платформи. Крім того, з меншою кількістю коду менша й імовірність появи помилок, що зменшує і час на тестування та підтримку коду.

2.4.3. Ефективне використовування ресурсів

Створення окремих програм коштує. Наявність єдиної кодової бази допомагає компаніям ефективно керувати своїми фінансовими ресурсами. Їхні команди розробників для платформ Android і iOS можуть навчитися писати та використовувати спільний код.

2.4.4. Цікавість для розробників

Багато мобільних розробників залюбки мали б сучасні багатоплатформні технології у наборі технологій (*tech-stack*) продуктів. Розробники може стати нудно працювати над повторюваними та рутинними завданнями, наприклад механізм розбирання JSON. Однак нові технології та завдання можуть повернути їхню зацікавленість, мотивацію та радість від виконання завдань. Це означає, що сучасний набір технологій може фактично спростити процес пошуку людей для команди мобільної розробки.

2.4.5. Швидші час виходу та зміни

Оскільки не потрібно створювати кілька програм для різних платформ, продукти виходять швидше. Крім того, якщо додаток потрібно змінити, розробникам буде простіше і швидше це зробити, оскільки це потібно зробити лише в одній програмі. Це також дозволяє краще реагувати на відгуки користувачів.

2.5. Проблеми багатоплатформної розробки

Усі підходи мають свої обмеження. Деякі представники технічної спільноти стверджують, що багатоплатформна розробка досі не перемогла проблем, пов’язаних із продуктивністю. Крім того, керівники проєктів можуть мати страх, що їх мета оптимізувати процес розробки матиме негативний вплив на UX. Проте з удосконаленням технологій багатоплатформні рішення стають все більш стабільними, адаптивними та гнучкими.

Іншим занепокоєнням є неможливість надійного використання додатками нативних функцій платформ і те, що використовувані додатками технології можуть підтримуватися не усіма платформами. Однак, часто є можливість визначити своє рішення для кожної платформи, на якій планується використовувати додаток.

Ці проблеми ставлять запитання: чи помітна користувачам різниця між нативним та багатоплатформними додатками?

Оскільки сучасні багатоплатформні фреймворки продовжують розвиватися, вони дедалі більше дозволяють мобільним розробникам створювати багатоплатформні додатки, що не відрізнити за якістю від нативних. Якщо програма написана добре, користувач не помітить різниці. Однак якість сильно залежить від використовуваних багатоплатформних інструментів розробки.

2.6. Інструменти для багатоплатформної розробки

2.6.1. Flutter

Рисунок 2.1 – логотип Flutter

Flutter – фреймворк для багатоплатформної розробки, створений Google. Він використовує мову програмування Dart. Flutter підтримує нативні функції, такі як визначення місцезнаходження, функції камери, доступ до жорсткого диска тощо. Якщо необхідно використати певну особливу функцію ОС і це є неможливим зі Flutter, є можливість написати нативний для платформи код із технологією Platform Channel.

Додатки, створені за допомогою Flutter, використовують єдиний інтерфейс та UX на всіх платформах, тому вони не завжди мають цілком нативний вигляд. Однією з найкорисніших переваг цього фреймворку є його функція швидкого оновлення («hot reload»), яка дозволяє розробникам миттєво переглядати внесені до коду зміни.

Flutter може бути найкращим варіантом у таких ситуаціях:

- додатку необхідно мати однаковий вигляд на всіх платформах, близький до нативного
- додаток може створювати велике навантаження на CPU/GPU

- MVP-додаток

Популярні додатки, створені за допомогою Flutter:

- Google Pay, Google Classroom, Google Ads
- Crédit Agricole
- BMW, Toyota
- CrowdSource
- Abbey Road Studios
- Hamilton
- eBay Motors

2.6.2. React Native

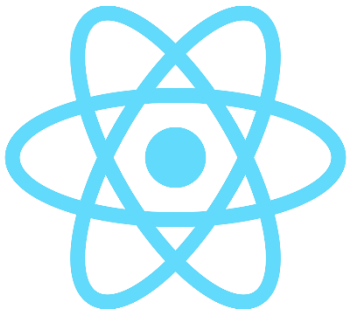


Рисунок 2.2 – логотип React Native

Facebook представив React Native у 2015 році як фреймворк із відкритим кодом, призначенням якого є допомагати мобільним розробникам створювати гібридні нативні та багатоплатформні програми. Він заснований на ReactJS – бібліотеці JavaScript для створення інтерфейсів. Іншими словами, він використовує JavaScript для створювання мобільних додатків для систем Android та iOS.

React Native надає доступ до сторонніх інтерфейсових бібліотек з готовими компонентами, допомагаючи мобільним розробникам економити час під час розробки. Подібно до Flutter, завдяки функції швидкого оновлення він дозволяє одразу бачити всі зміни, внесені до коду.

Слід розглянути використання React Native у таких випадках:

- додаток є відносно простим і очкується бути легким
- команда розробників вільно володіє JavaScript або React

Популярні програми, створені за допомогою React Native:

- Facebook
- Instagram
- Skype
- Uber Eats

2.6.3. Kotlin Multiplatform Mobile

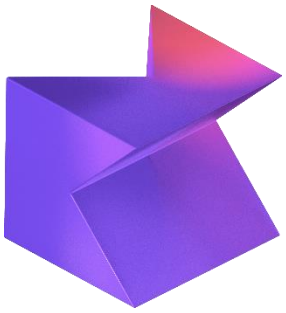


Рисунок 2.3 – логотип Kotlin Multiplatform Mobile

Kotlin Multiplatform Mobile – SDK для багатоплатформної мобільної розробки, підтримуваний компанією JetBrains. Це дозволяє створювати програми для Android та iOS із спільною логікою. Переваги:

- легка й непомітна інтеграція до проєктів

- повний контроль над інтерфейсом з можливістю використовувати найновіші інтерфейсові фреймворки, такі як SwiftUI та Jetpack Compose
- легкий доступ до SDK систем Android та iOS без обмежень

Великі компанії та стартапи вже використовують Kotlin Multiplatform Mobile для оптимізації та прискорення мобільної розробки. Переваги цього підходу можна зрозуміти з досвіду компаній, що вже його застосували:

- Команда розробників відзначеного нагородами додатка Todoist почала використовувати Kotlin Multiplatform Mobile для синхронізації логіки сортування свого додатка на кількох платформах, і таким чином вони поєднали переваги нативних і багатоплатформних додатків.
- Поява Kotlin Multiplatform дозволила Philips швидше впроваджувати нові функції та покращило взаємодію між розробниками Android та iOS.
- Shopify вдалося використати Kotlin Multiplatform для зроблення 95% свого коду спільним між платформами, що також забезпечило значне підвищення продуктивності.

ВИСНОВОК ДО РОЗДІЛУ 2

Оскільки інструменти для багатоплатформної розробки продовжують розвиватися, їхні обмеження починають тьмяніти у порівнянні з перевагами, які вони надають. На сьогодні доступно багато різноманітних технологій, які відповідають різним способам організації робочого процесу і наборам вимог.

Зрештою, обрати правильний шлях дозволить лише ретельний аналіз вимог і своєї мети. Оскільки я є єдиним автором цього проєкту і прагнув охопити системи Android та iOS, я обрав багатоплатформний шлях.

Існують і інші інструменти для багатоплатформної розробки, яких я не описав у цьому розділі. Але розглянуті 3 мають достатньо унікальних особливостей, щоб показати, наскільки різними вони бувають.

Вони мають і спільне. Наприклад, багато з них підтримують можливість швидкого оновлення додатка після внесення змін до коду. Це не є можливим із будь-якими змінами, на жаль, але це все одно економить розробникам багато часу.

Розповсюдженою серед фреймворків є також передбаченість їх використання не як головна платформа для розробки, а як побічний інструмент для створення частини програми, що може бути спільною для кількох платформ.

Для додатка Подійни́к я обрав створити двоплатформний додаток за допомогою Flutter.

3. *BACKEND* ЯК СЕРВІС (BaaS)

3.1. Що є BaaS?

3.1.1. Опис

BaaS – модель хмарного сервісу, яка передбачає передання розробниками сторонній платформі частини прихованих від користувача аспектів веб- або мобільної програми, щоб вони могли зосередитися на відкритій користувачу складовій та лише на частині прихованої. BaaS-платформи надають програмне забезпечення для дій, що здійснюються на серверах, таких як автентифікація користувачів, керування базою даних, сповіщення (для мобільних додатків), хмарне сховище та хостинг.

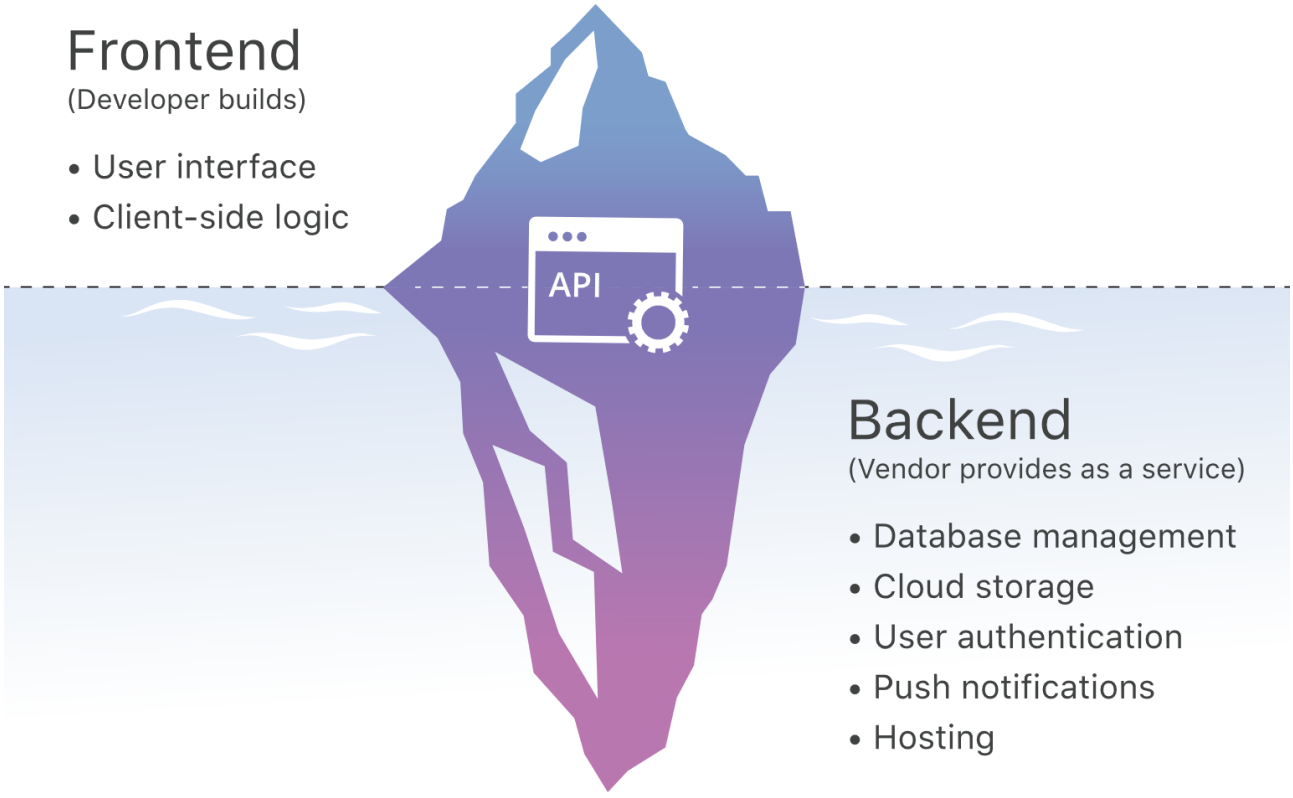


Рисунок 3.1 – поділ аспектів додатка у моделі BaaS

Можна уявити, що розробка програми без використання BaaS-платформи це постановка фільму. Режисер відповідає за керування знімальними групами, освітленням, будівництвом декорацій, гардеробом, підбором акторів і розкладом, а також за власне знімання та постановку сцен, які з'являться у фільмі. Що якби існувала служба, яка займалася б усіма вторинними процесами, і все, що режисер мав би робити, це керувати сценою та знімати її? Власне, це ідея BaaS: платформа бере на себе «світло» та «камеру» (серверні функції), щоб режисер (розробник) міг просто зосередитися на «сцені» – що користувач бачить і з чим взаємодіє.

BaaS дозволяє розробникам зосередитися на створенні інтерфейсу та логіки під ним. Через API та SDK, які надають BaaS-платформи, розробники можуть інтегрувати всі необхідні їм функції серверної частини, без необхідності створювати її власноруч. Їм також не потрібно займатися серверами, віртуальними машинами або контейнерами, аби забезпечити роботу програми. Таким чином, вони не лише можуть створювати та запускати мобільні та веб-додатки швидше, а й набагато легше підтримувати їх роботу.

3.1.2. Які сервіси надають BaaS-платформи?

Більшість платформ надають:

- керування базою даних
- хмарне сховище
- автентифікація користувачів
- сповіщення
- хостинг

Деякі платформи мають додаткові унікальні сервіси. Наприклад, Firebase, яку я розглядаю пізніше.

3.2. BaaS і безсерверні обчислювання

BaaS і безсерверні обчислювання (serverless computing) мають дещо спільне, оскільки в обох випадках розробникам не потрібно думати про серверну частину. Крім того, багато BaaS-платформ власне пропонують безсерверні обчислювання як один із сервісів. Однак існують значні операційні відмінності між програмами, створеними з використанням BaaS, і справжньою безсерверною архітектурою.

3.2.1. Будова програми

Backend безсерверної програми розбитий на функції, кожна з яких реагує на події та виконує лише одну дію. У той час як серверна частина BaaS-програми визначається самою платформою; розробникам не потрібно витрачати час на кодування будь-чого, окрім інтерфейсу програми та логіки під ним.

3.2.2. За яких умов код виконується

Безсерверні архітектури керуються подіями, себто запускаються у відповідь на них. Кожна функція виконується лише тоді, коли її викликає певна подія, і за жодних інших умов. Програми, що використовують BaaS, зазвичай не керуються подіями, а це означає, що вони вимагають більше ресурсів сервера.

3.2.3. Де код виконується

Безсерверні функції можуть бути викликані з будь-якого місця на будь-якому пристрої за умови, що вони мають зв’язок з рештою програми. Це дає змогу мати периферійні обчислення (edge computing) в архітектурі програми. *Backend* як сервіс не обов’язково налаштований на викликання виконання коду з будь-якого місця та в будь-який час, але це залежить від платформи.

3.2.4. Як додаток масштабується

Здатність до масштабування є однією з найбільших відмінностей безсерверної архітектури від інших. Система постачальника просто створює тимчасовий об’єкти кожної функції за потреби. Програма, що використовує BaaS-платформу, не налаштована на масштабування таким чином, якщо лише платформа не пропонує безсерверні обчислювання як сервіс і вбудував це до програми.

3.3. Платформи

3.3.1. Firebase

Рисунок 3.2 – логотип Firebase

Платформа для розроблювання додатків, підтримувана компанією Google. Вона має багато сервісів, які пришвидшують розробку автоматизованою *backend* інфраструктурою, дозволяють надійно запускати проєкти та слідкувати за їх стабільністю і продуктивністю, збагачують залученість користувачів зручними системами сповіщень та тестування змін без повного їх упровадження.

3.3.1.1. Authentication

Рисунок 3.3 – ілюстрація Firebase Authentication

Багатьом додаткам потрібно знати особистість користувача. Це дозволяє забезпечувати єдину форму додатка на всіх пристроях. Firebase Authentication надає *backend* сервіси та бібліотеки для користування ними та побудови інтерфейсу. Із цим легко автентифікувати користувача.

На створення власної системи автентифікації можуть знадобитися місяці, і для її подальшого обслуговування потрібна окрема команда. З Firebase Authentication процес автентифікації можна вмістити у лічені рядки коду. Не складно здійснити й незвичайні випадки, такі як злиття акаунтів.

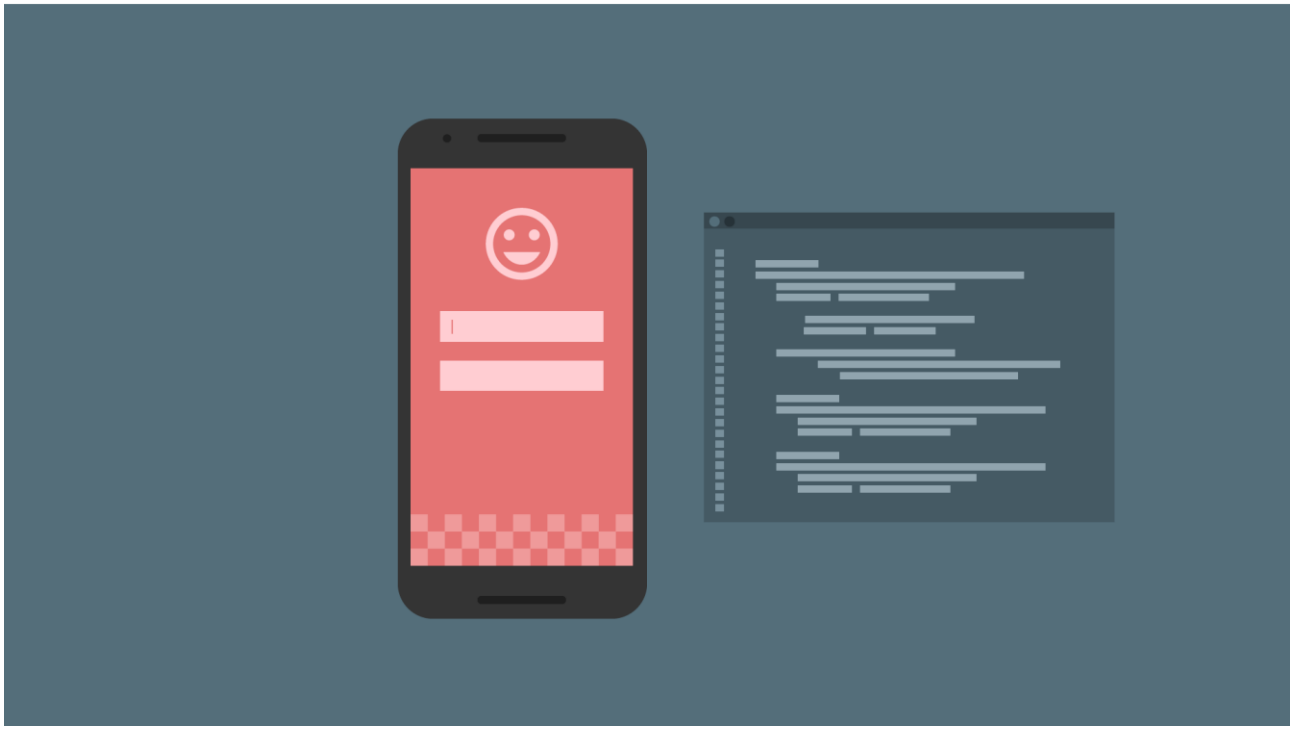


Рисунок 3.4 – ілюстрація простоти впровадження

Система безпеки Firebase створена тією ж командою, що розробила Google Sign-in, Smart Lock і Chrome Password Manager. Вона застосовує внутрішній досвід Google керування однією з найбільших акаунтових баз даних у світі.

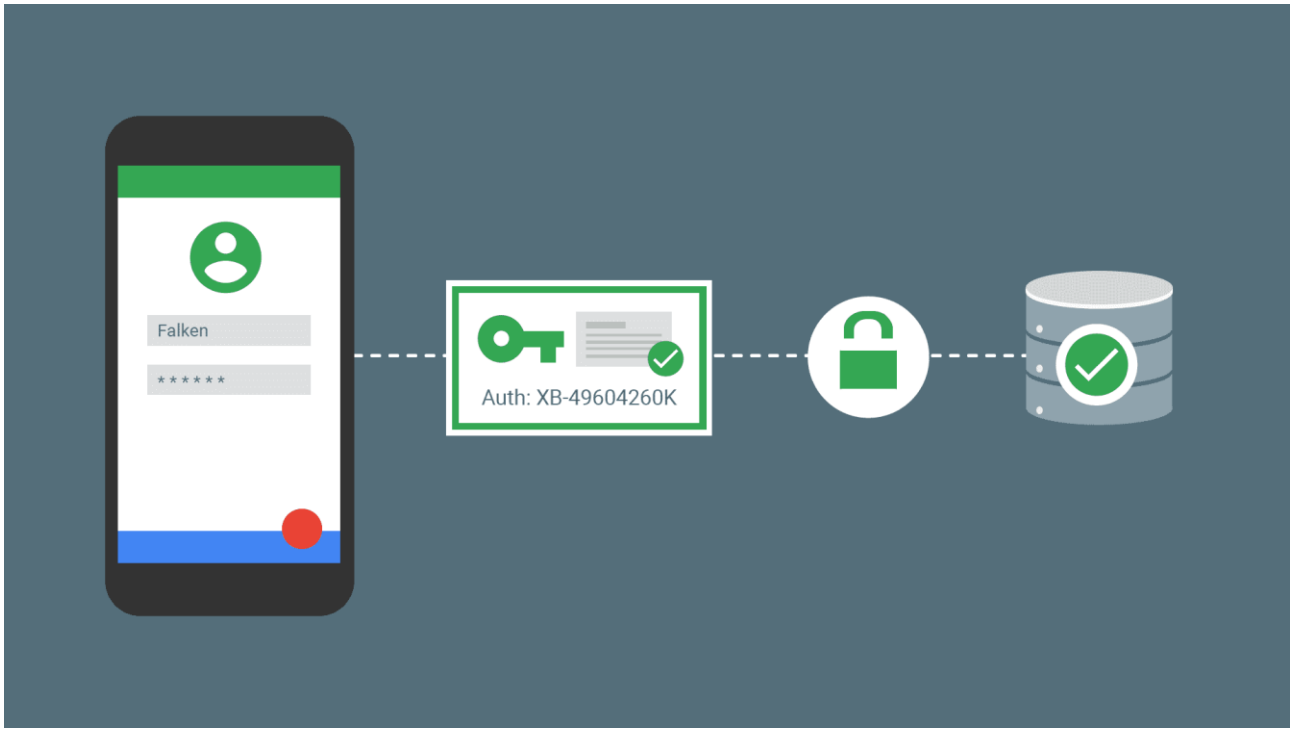


Рисунок 3.5 – ілюстрація системи безпеки

Цей сервіс дозволяє автентифікацію паролем, номером телефону, акаунтами багатьох відомих платформ: Google, Twitter, GitHub тощо.

Рисунок 3.6 – ілюстрація шляхів автентифікації

3.3.1.2. Cloud Firestore

Рисунок 3.7 – ілюстрація Cloud Firestore

Гнучка база даних для мобільних пристроїв, вебу і розроблювання серверів. Вона тримає дані синхронізованими між усіма клієнтами стеженням наживо і дозволяє *offline* використання, тобто без додаткового коду додатки є чуйними до доступу до інтернету. Cloud Firestore дозволяє легке поєднання з іншими сервісами Firebase та Google Cloud.

Дані структуруються за допомогою колекцій і документів. Можна створювати ієрархії для зберігання пов’язаних даних і легкого отримування необхідних даних за допомогою запитів.

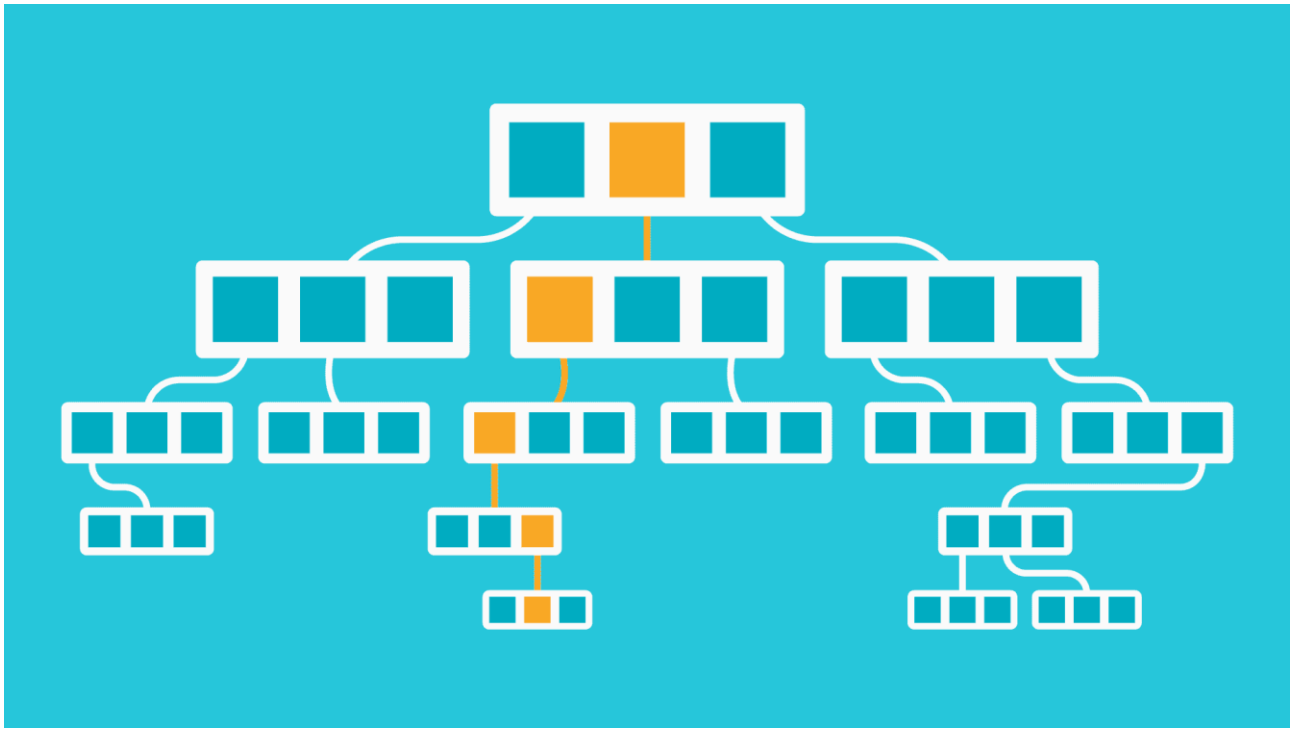


Рисунок 3.8 – ілюстрація структури даних

Дані додатків автоматично синхронізовуються між пристроями. Про зміни даних можна отримувати сповіщення, щойно вони відбуваються, що дає легко створювати додатки для роботи наживо, зокрема для спільної роботи з іншими. Користувачі можуть отримати доступ до своїх даних і внести зміни в них у будь-який час, навіть коли вони *offline*.

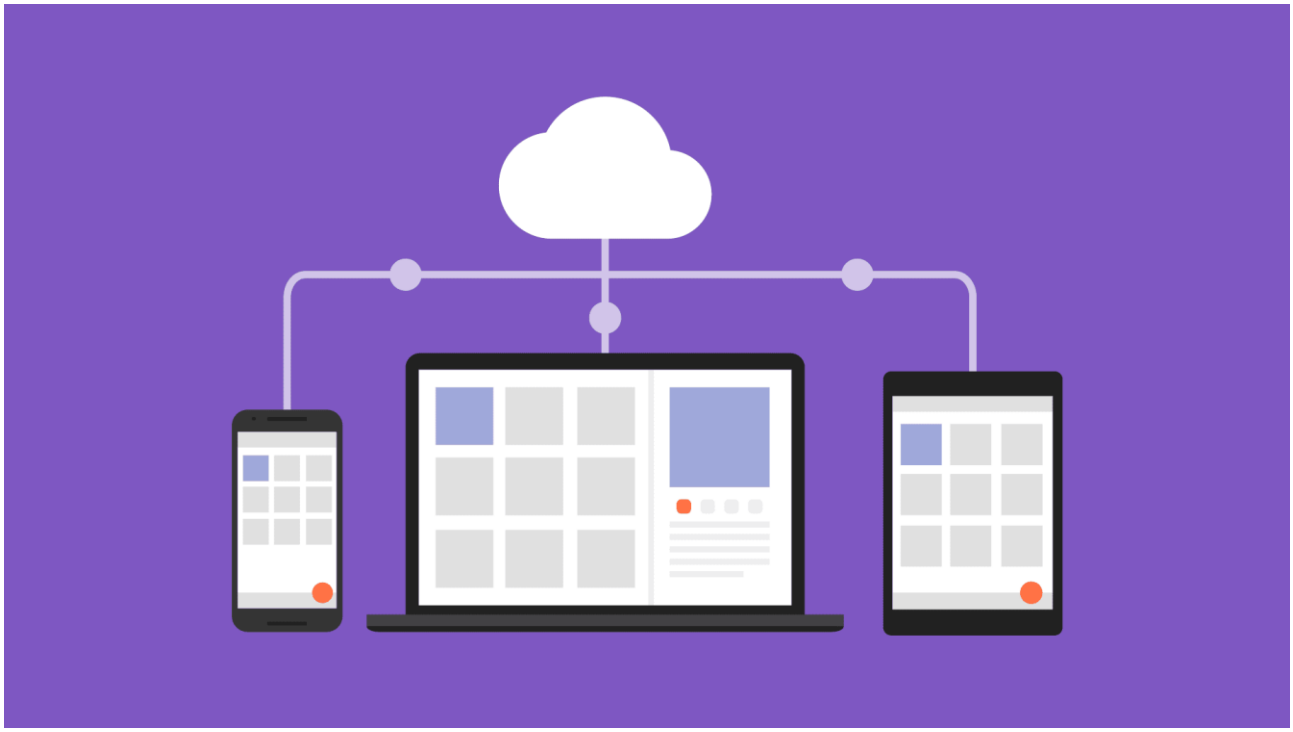


Рисунок 3.9 – ілюстрація синхронізації

Cloud Firestore надає *SDK* для мобільних пристроїв та вебу і всеосяжний набір правил безпеки, щоб отримувати доступ до своєї бази даних було можливо без власного сервера. Використовуючи Cloud Functions, інший безсерверний обчислювальний сервіс Firestore, можна виконувати серверний код, який реагує на зміни даних у базі даних. Звісно, доступ до Cloud Firestore можна отримати і за допомогою бібліотек у Node, Python, Dart, Go тощо.

Рисунок 3.10 – ілюстрація безсерверності

З інфраструктурою систем зберігання Google, Cloud Firestore створена для росту разом з бізнесами. Можна зосередитися на створенні додатків замість того, щоб думати про сервери або турбуватися про несистематичність.

Рисунок 3.11 – ілюстрація масштабування

3.3.2. Supabase

Рисунок 3.12 – логотип Supabase

Supabase – це альтернатива до Firebase із відкритим кодом для створення безпечних і продуктивних *backend*-ів із Postgres із мінімальною конфігурацією.

3.3.2.1. Authentication

Supabase має просту систему автентифікації, що працює без жодних сторонніх сервісів. Вона включає автентифікацію, авторизацію і керування користувачами.

Дані користувачів зберігаються так само у Supabase, тому не потрібно думати про безпекові проблеми, пов’язані зі сторонніми сервісами. Дані хостяться у 8-ми різних місцях.

Система підтримує автентифікацію акаунтами багатьох платформ, таких як Google, Apple, GitHub, Twitter, Discord тощо.

3.3.2.2. Database

Кожний проєкт у Supabase це база даних Postgres – реляційна база даних, якій у світі довіряють чи не найбільше. У проєкт можна вбудувати вже наявну базу.

База даних підтримує автентифікацію, що використовує технологію PostgreSQL Row Level Security, що чітко контролює, до яких даних користувачі можуть мати доступ.

Є можливим отримувати сповіщення про зміни у базі з мілісекундною затримкою, щоб у відповідь на них виконувати дії.

3.3.2.3. Storage

Надійне сховище для об’єктів з необмеженою можливістю до масштабування, для будь-якого типу файлів. Є можливим легко визначати власні правила доступу.

Сховище є дуже швидким через легкий шар API, що користується продуктивністю Postgres.

Воно сумісне з рештою сервісів Supabase, такими як Authentication і Database.

3.3.2.4. Edge Functions

Периферійні функції дозволяють виконувати код фізично якнайближче до користувачів задля якнайменшої затримки. Вони зберігаються у 29 регіонах по всій планеті. Їх виконання можна викликати будь-якою подією у базі даних.

ВИСНОВОК ДО РОЗДІЛУ 3

Backend як сервіс – модель, що дозволяє зосередитися на інтерфейсі та взаємодії користувача з додатком за рахунок передання сторонній платформі серверних функцій. BaaS-платформи надають різні сервіси, серед яких: автентифікація користувачів, база даних, сповіщення, хостинг та інші. Вони дозволяють не лише пришвидшити процес створення додатка, а й зменшити кількість зусиль, необхідних на підтримування.

Було розглянуто 2 платформи: Firebase та Supabase. Перша з них є старшою і має набагато більше сервісів. Основна база даних Firebase, Cloud Firestore, має іншу парадигму, ніж база у Supabase. Парадигма першої – документова, а другої – реляційна. Різні парадигми використовуються для різних випадків. Враховуючи усі відмінності між цими BaaS-платформами і свої потреби для цього проєкту, я обрав Firebase.

4. ДОДАТОК ПОДІЙНИ́К

Рисунок 4.1 – логотип додатка Подійни́к

Подійни́к – мінімалістичний двоплатформний мобільний додаток для українських студентів.

Мета – спростити слідкування за подіями і зберігання інформації, а також витягнути важливу інформацію, пов’язану з навчанням, з додатків для спілкування. Я прагнув створити спільний простір для студентів групи, де кожний може додати подію, інформацію, зробити оголошення або поставити запитання.

4.1. Початок

Студент почув про додаток і вирішив спробувати впровадити його у своїй групі. Він встановлює його і реєструється з акаунтом Google або Apple. Подійни́к створює новий ідентифікатор групи, який цей студент повідомляє своїм одногрупникам. Ним вони приєднуються до тієї ж групи – того ж «простору».

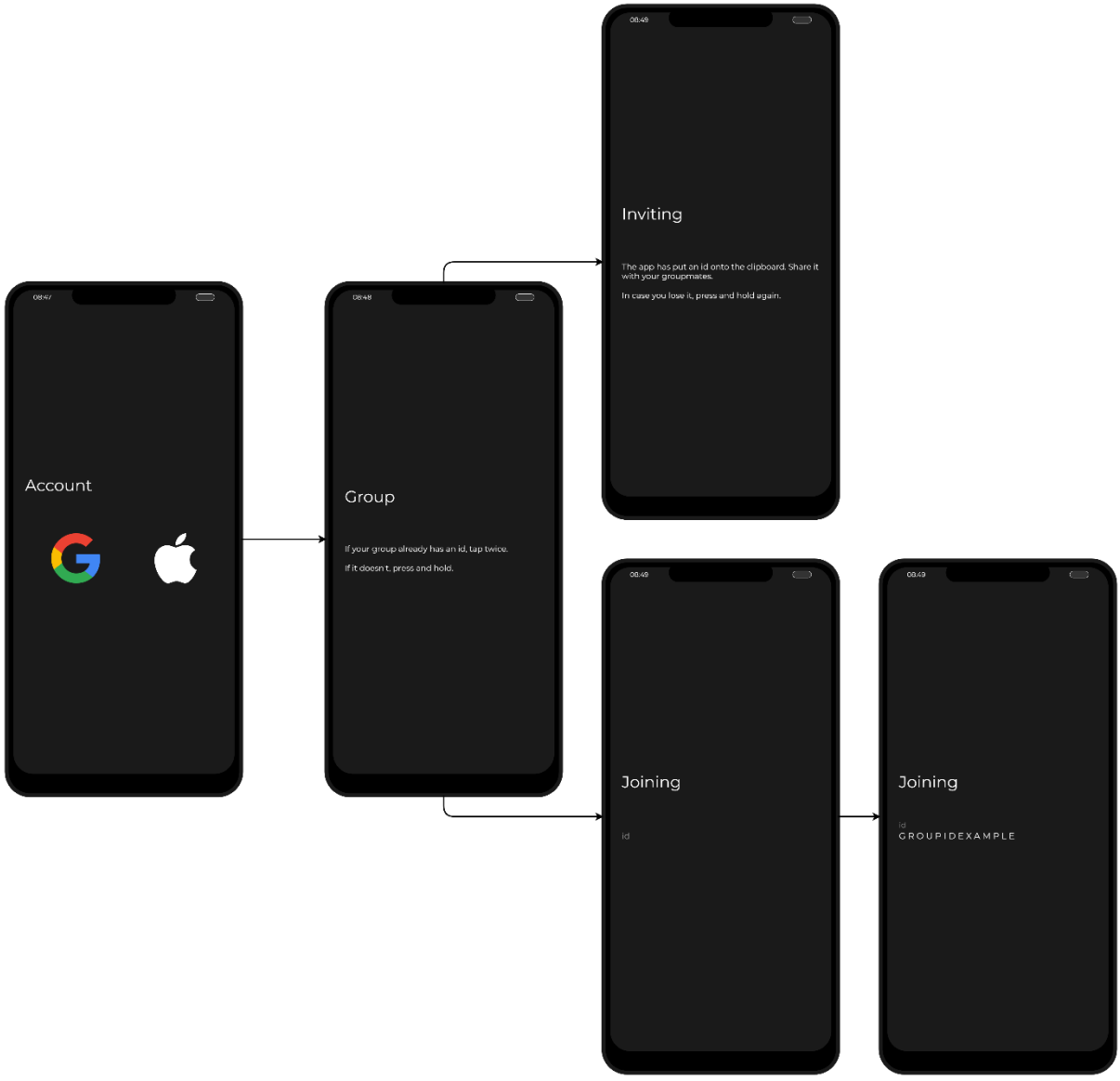


Рисунок 4.2 – дерево станів для користувача, не приєднаного до групи

Тепер ці студенти використовують додаток як група. Додане одним – бачене всіма.

4.2. Розділи

4.2.1. Події

Подія – будь-що, прив’язане до певного моменту в часі. Вона може належати предметові, як реченець лабораторної роботи або контрольна робота, або бути окремою, як обрання предметів на наступний семестр. Подія може мати нотатку, на випадок доречності додання чогось.

Рисунок 4.3 – дерево станів, пов’язаних із подіями

4.2.2. Предмети

Предмет може бути або загальним – таким, що вивчає вся група, або обираним. До нього можна додавати інформацію, як навчальний план або контакти викладача. Сторінка предмету має підрозділи для його інформації та предметів. Якщо він обираний – і для студентів, що його вивчають.

Рисунок 4.4 – дерево станів, пов’язаних із предметами

4.2.3. Окреме

Цей розділ – для окремих подій та інформації. Як і подіям, інформації також можливо не належати предметові, як посилання на файли групи або інформація про неї, запитувана в документах.

Рисунок 4.5 – дерево станів, пов’язаних із окремим

4.2.4. Повідомлення

Повідомлення може бути нагаданням або способом для студента повідомити щось решті групи. Наприклад, коли росія перетворює університет на руїни і він робить оголошення про зміну навчального процесу.

Рисунок 4.6 – дерево станів, пов’язаних із повідомленнями

4.2.5. Вони

Студенти групи, окрім користувача. Сторінка студента має інформацію про нього, якщо він її додав, та обрані ним предмети, якщо вони є.

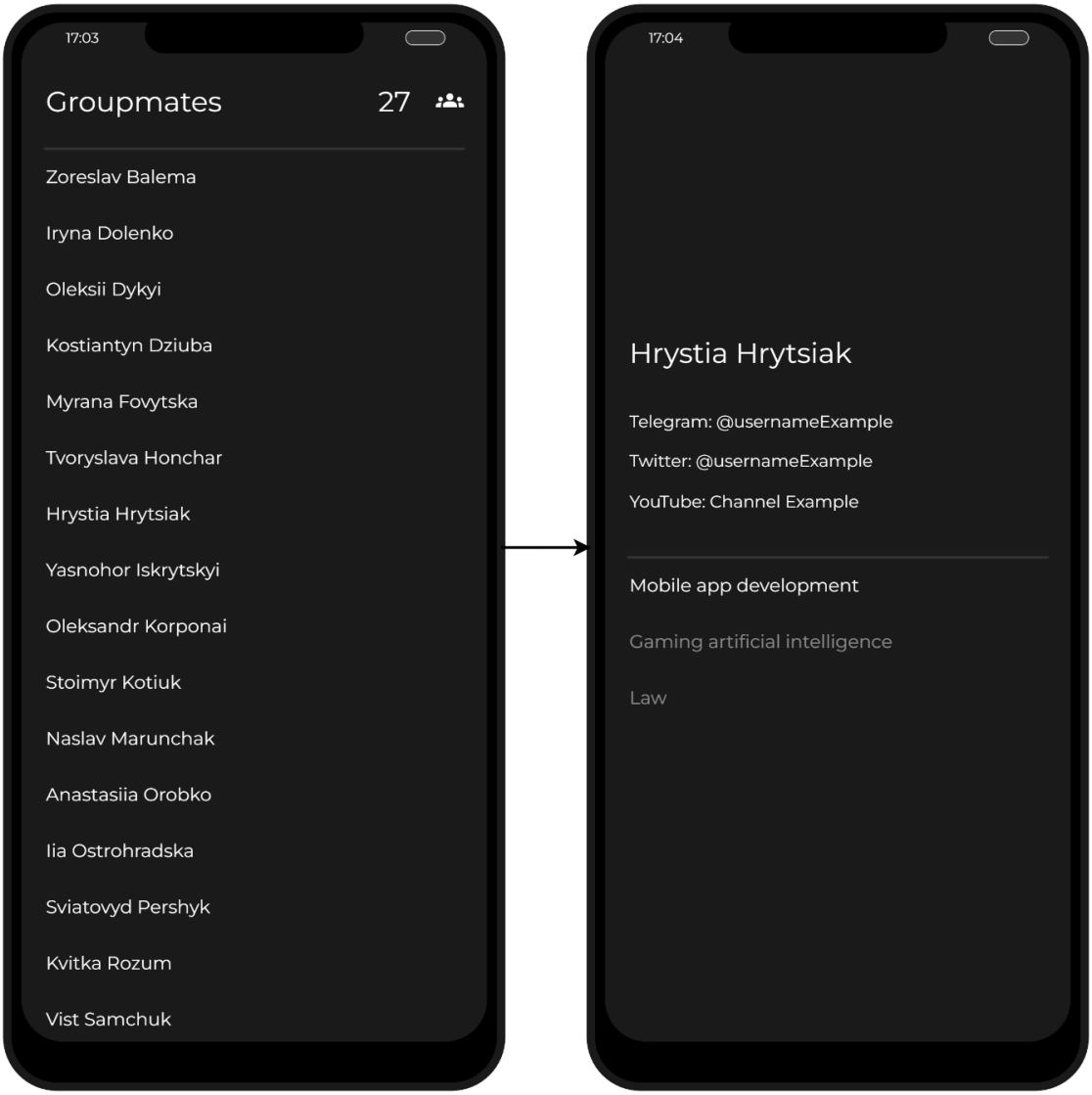


Рисунок 4.7 – дерево станів, пов’язаних із одногрупниками

4.2.6. Я

Розділ, де можна змінити ім’я, інформацію на своїй сторінці, мову. Змінити групу та залишити акаунт також можна тут. І змінити кольори також можна… буде одного дня.

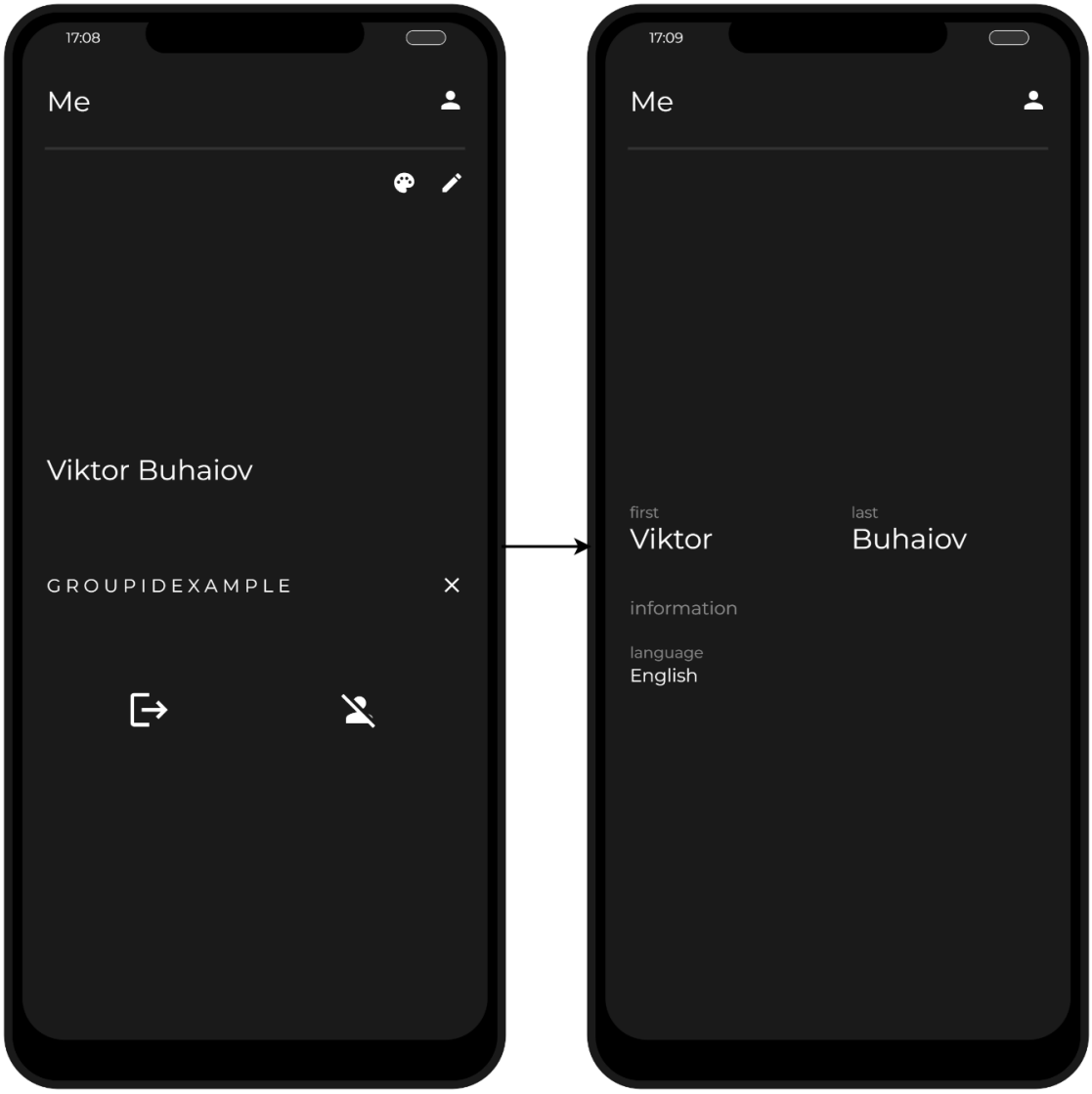


Рисунок 4.8 – дерево станів, пов’язаних із користувачем

4.3. Залежності

4.3.1. Flutter Hooks

Рисунок 4.9 – логотип Flutter Hooks

Пакет, що імплементує для Flutter *гачки* (hooks), подібно до відповідного проєкту для React. *Гачки* – новий запропонований вид об’єктів, що керують життєвим циклом віджетів у Flutter. Мета їх існування – прибрати повторюваний код і надати коду віджетів здатності бути спільним для кількох віджетів.

Змінні віджети у Flutter (stateful widgets) мають особливий життєвий цикл. Для зміни етапів цього циклу його методи можна перевизначати. Однак недоліком цих віджетів є те, що якщо ідентична перевизначена логіка зустрічається ще деінде, не існує способів використати її повторно. Доводиться тримати копії цієї логіки для кожного змінного віджета.

Наприклад, якщо віджет має текстове поле і потребу взаємодіяти з ним будь-як, окрім отримування нових значень, йому потрібно створити об’єкт TextEditingController, оскільки взаємодія з текстовими полями здійснюється через ці об’єкти. Для гарантованого звільнення ресурсів, що використовує цей контролер, після «смерті» віджета, метод життєвого циклу dispose необхідно перевизначити, додавши до нього виклик однойменного метода об’єкта контролера. Ця логіка має високу ймовірність зустрітися у проєкті ще, але Flutter не має можливості об’єднати їх, що призводить до повторюваного коду. *Гачки* вирішують цю проблему, забираючи цю логіку всередину себе.

Приклад зі проєкту: сторінка Joining на рисунку 5.2. Вона має текстове поле. Її віджетові JoiningPage, визначеному у файлі states/identification/joining_page.dart, потрібно взаємодіяти з цим полем. Для створення об’єкта TextEditingController він використовує *гачок* useTextEditingController, що вже визначено у пакеті. Метод build віджета JoiningPage:

```
@override
  Widget build(BuildContext context, WidgetRef ref) {
        final idField = useTextEditingController();

        return GestureDetector(
              onDoubleTap: () => _handleJoin(ref, idField.text),
              child: Scaffold(body: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                          const Text("Joining"),
                          // do: textInputAction
                          TextField(
                                controller: idField,
                                decoration: const InputDecoration(labelText: "id"),
                          )
                    ]
              ))
        );
  }
```

Як можна зрозуміти, окрім можливості створювати власні *гачки*, пакет Flutter Hooks надає низку передвизначених, що найчастіше використовуються розробниками.

4.3.2. Riverpod

Рисунок 4.10 – логотип Riverpod

Riverpod – реактивний фреймворк кешування для Flutter/Dart. Він може автоматично отримувати, кешувати, комбінувати та повторно здійснювати мережеві запити, також беручи помилки під час цих дій на себе.

Сучасні додатки рідко одразу мають усю інформацію, необхідну для відтворення інтерфейсу користувача. Дані часто асинхронно витягуються з сервера.

Проблема в тому, що працювати з асинхронним кодом нелегко. Хоча Flutter має певний спосіб зберігання стану, фактично, це усе, що він робить. Таким чином, низка проблем залишається невирішеною:

- Асинхронні запити потрібно кешувати локально, оскільки було б не обґрунтовано повторно виконувати їх кожного разу, коли оновлюється інтерфейс користувача.
- Оскільки є кеш, він може застаріти, бути необережними
- Також потрібно обробляти помилки та стани під час очікування на результат запиту.

Розв’язування цих проблем у великому додатку може бути важким, і вони залежать від великої кількості можливостей у ньому:

- потягнути, що відволіктися
- нескінченні списки / запити на нові дані під час гортання

- пошук під час введення
- усування відмов в асинхронних запитах
- скасовування асинхронних запитів, коли вони більше не використовуються
- оптимістичний UI
- режим *offline*

Ці функції можуть бути складними для здійснення, але вони мають вирішальне значення для приємної взаємодії користувача з додатком. Проте мало пакетів намагаються вирішити ці проблеми напряму, і багато роботи доводиться виконувати власноруч.

Тут на допомогу приходить Riverpod. Він намагається вирішити ці проблеми, пропонуючи новий унікальний спосіб написання бізнес-логіки, натхненний віджетами Flutter. Багато в чому Riverpod можна порівняти з віджетами, але для стану.

Використовуючи цей новий підхід, ці складні можливості здебільшого вбудовані у використовувані інструменти. Усе, що лишилося – зосередитися на інтерфейсі.

4.3.3. Hive

Рисунок 4.11 – логотип Hive

Hive – проста локальна база даних парадигми ключ-значення, що використовує *області* для структурування всіх даних. Область можна порівняти з таблицею в SQL, але вона не має структури і може містити будь-що.

Для малого додатка однієї області може бути достатньо. Для складніших випадків, кілька – чудовий спосіб організувати дані. Області можуть бути кодовані для зберігання прихованої інформації.

Hive підтримує всі примітивні типи, List, Map, DateTime та Uint8List. Є можливим зберігати об’єкти будь-якого іншого типу. Для цього потрібно зареєструвати відповідний TypeAdapter до бажаного типу, що описує, як конвертувати об’єкти до двійкової форми та навпаки. Можна як написати TypeAdapter власноруч, так і згенерувати його автоматично. Найчастіше, згенерований адаптер є продуктивним. Проте інколи є маленькі речі, що можна покращити, написавши його власноруч.

Hive є дуже продуктивною через свою легкість у порівнянні з реляційними базами даних. API є дуже близьким до того, як дані зберігаються на диску.

У базах даних парадигми ключ-значення можна зберігати майже будь-який вид даних:

- профілі користувачів
- інформація про сеанс використання
- коментарі до допису
- повідомлення
- вміст кошика
- категорії товарів
- двійкові дані

Hive також є одним з найкращих варіантів, коли мова йде про багатоплатформну підтримку. Вона працює і у браузері, використовуючи IndexedDB.

З однієї сторони, будь-які дані можуть зберігатися у Hive, якщо вони правильно модельовані. З іншої, інколи може бути зручніше мати реляційну базу, як SQLite. Зручніше, але не продуктивніше. Надто, коли дані мають складні зв'язки і додаток спирається на використовування індексів у даних та складні запити до бази.

Оскільки база даних Hive така легка і ледве-ледве збільшить розмір додатка, можна поєднувати її в одному проєкті і з іншими базами.

4.3.4. Пакети-інтерфейси до сервісів Firebase

Рештою залежностей є пакети, що надають API для використовування сервісів Firebase: firebase_core як основа для всіх, cloud_firestore для бази даних Cloud Firestore, firebase_auth і google_sign_in для системи Authentication.

У розділі 3.3.1.1 було розглянуто сервіс Firebase Authentication, де зазначалося, що автентифікація з ним є дуже простою. Приклад зі проєкту: сторінка Account на рисунку 4.2. Вона надає можливість автентифікації акаунтом двох платформ: Google та Apple, хоча у поточній версії імплементовано лише перший варіант. Код її віджета Authentication, визначеного у файлі states/authentication/authentication.dart:

```
class Authentication extends ConsumerWidget {
    const Authentication();

    @override
    Widget build(BuildContext context, WidgetRef ref) {
        return Scaffold(body: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
                const Text("Authentication"),
                Row(
```

```
                                mainAxisAlignment: MainAxisAlignment.spaceAround,
                                children: [
                                        TextButton(
                                                onPressed: () => _signInWithGoogle(ref),
                                                child: const Text('Google')
                                        ),
                                        const TextButton(
                                                onPressed: null,
                                                child: Text('Apple')
                                        )
                                ]
                        )
                ]
        ));
}

Future<void> _signInWithGoogle(WidgetRef ref) async {
        final account = await GoogleSignIn().signIn();
        if (account == null) return;

        final accountAuth = await account.authentication;
        final authCred = GoogleAuthProvider.credential(
                idToken: accountAuth.idToken,
                accessToken: accountAuth.accessToken
        );
        final userCred = await FirebaseAuth.instance.signInWithCredential(authCred);

        final userId = userCred.user!.uid;
        final userRepository = ref.read(userRepositoryProvider);
        final User user;
        final AppState appState;

        if (userCred.additionalUserInfo!.isNewUser) {
                user = User(
                        id: userId,
                        firstName: account.displayName ?? '',
                        lastName: ''
                );
                await userRepository.initAccount(user);

                appState = AppState.identification;
                print('Authentication: new $userId');
        }
        else {
                user = await userRepository.user(id: userId);
                appState = user.groupId != null ? AppState.home : AppState.identification;
                print('Authentication: existing $userId(${user.groupId})');
        }
```

```
            ref.read(initialUserProvider.notifier).state = user;
            ref.read(appStateProvider.notifier).state = appState;
        }
    }
```

Логіку автентифікації визначено у методі _signInWithGoogle. Як бачимо, це справді небагато коду. І жодного власного автентифікаційного сервісу.

ВИСНОВОК

Робота описує створення двоплатформного мобільного додатка для організації навчального процесу студентів. Він дозволяє групам студентів мати спільний простір для зберігання подій та інформації, пов’язаних із навчанням, а також робити оголошення у ньому.

У першому розділі розглянуто програми та системи, що могли б бути альтернативою створеному додатку. З метою кращого визначення вимог до нього, було охоплено різні типи альтернатив: від простої загальної нотаткової системи до системи, розробленої спеціально для навчальних закладів із метою зробити зручнішою певну складову навчального процесу.

Другий розділ описує багатоплатформну розробку. У ньому пояснено, чому вона виникла як підхід до мобільної розробки, розглядає інші підходи: окремі програми для кожної платформи (нативна розробка), прогресивні веб-додатки, гібридні додатки. Проведено порівняння нативного та багатоплатформного підходів, виявлено переваги та проблеми останнього, оглянуто інструменти (фреймворки) для нього. На основі виконаного аналізу прийнято рішення про підхід до розробки й обрано інструменти для неї.

У третьому розділі розглянуто модель *Backend* як сервіс (Backend as a Service) і чому вона набуває такої популярності, як саме вона пришвидшує розробку, спрощує підтримку і підвищує надійність. Проведено огляд двох популярних BaaS-платформ і їхніх сервісів, на основі якого зроблено вибір платформи для створеного додатка.

Четвертий розділ присвячено огляду та опису додатка, а також додаткових використаних технологій. Пояснено, як користувач, який ніколи не чув про додаток, починає його використовування. Описано кожний розділ інтерфейсу і показано всі можливі стани додатка разом зі зв’язками між ними. Наведено приклади з проєкту, як додаткові використані технології можуть бути корисні.

ДЖЕРЕЛА

1. What is cross-platform mobile development? [Електронний ресурс] – URL: https://kotlinlang.org/docs/cross-platform-mobile-development.html
2. GlobalStats. Mobile Operating System Market Share Worldwide [Електронний ресурс] – URL: https://gs.statcounter.com/os-market-share/mobile/worldwide
3. Wikipedia. Google Keep. [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/Google_Keep
4. Wikipedia. Notion. [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/Notion_(productivity_software)
5. Google Classroom. [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/Google_Classroom
6. Blended learning. [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/Blended_learning
7. Flutter. Documentation. [Електронний ресурс] – URL: https://docs.flutter.dev
8. Flutter. Showcase. [Електронний ресурс] – URL: https://flutter.dev/showcase
9. Flutter. API reference. [Електронний ресурс] – URL: https://api.flutter.dev
10. React Native. [Електронний ресурс] – URL: https://reactnative.dev
11. Kotlin Multiplatform. [Електронний ресурс] – URL: https://kotlinlang.org/lp/multiplatform
12. Cloudflare. What is BaaS? [Електронний ресурс] – URL: https://www.cloudflare.com/learning/serverless/glossary/backend-as-a-service-baas

13. Supabase. [Електронний ресурс] – URL:
https://supabase.com
14. Firebase. Documentation. [Електронний ресурс] – URL:
https://firebase.google.com/docs
15. Firebase. Authentication. [Електронний ресурс] – URL:
https://firebase.google.com/products/auth
16. Firebase. Cloud Firestore. [Електронний ресурс] – URL:
https://firebase.google.com/products/firestore
17. GitHub. Flutter Hooks. [Електронний ресурс] – URL:
https://github.com/rrousselGit/flutter_hooks
18. Riverpod. [Електронний ресурс] – URL:
https://riverpod.dev
19. Hive. Docs. [Електронний ресурс] – URL:
https://docs.hivedb.dev/#

ДОДАТОК 1

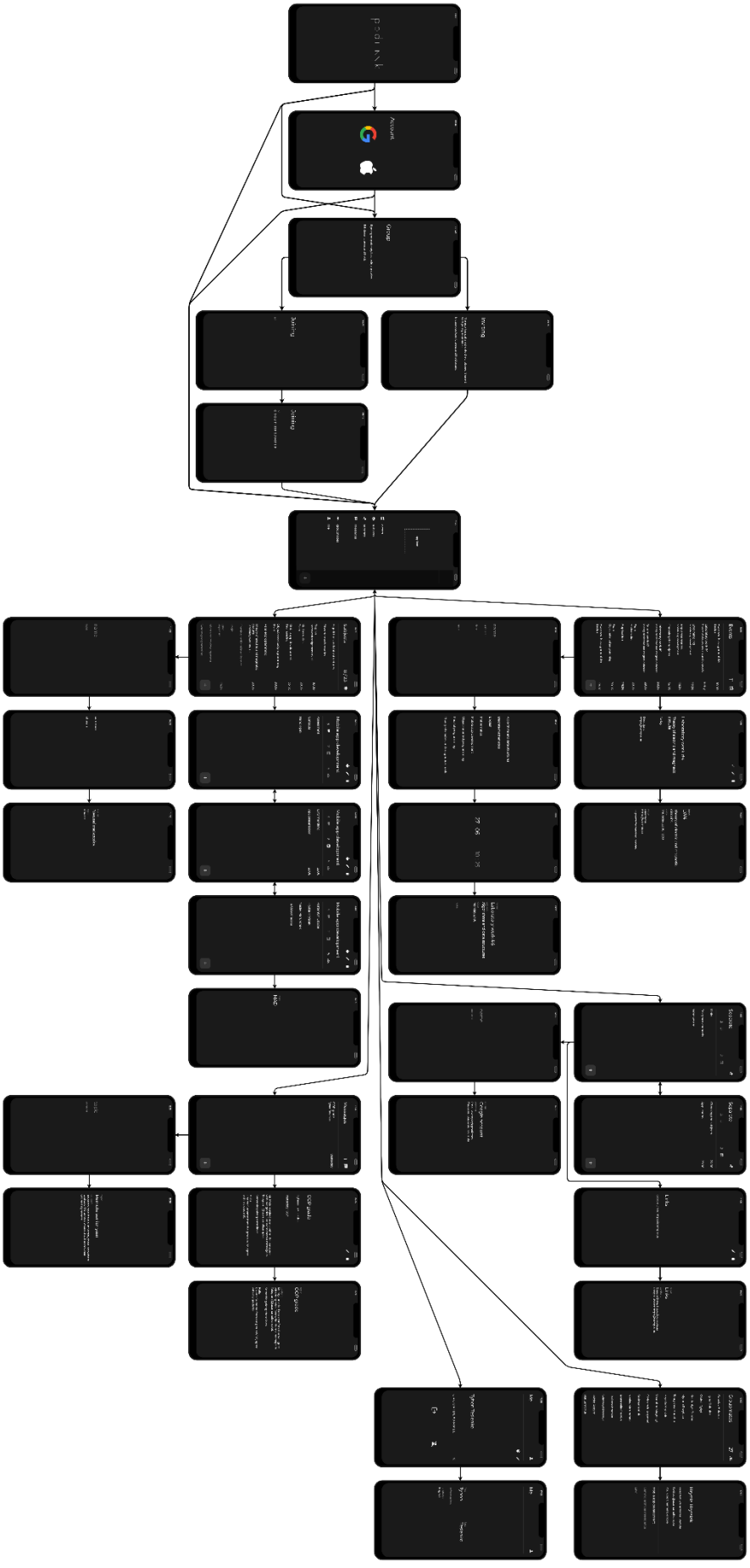
Двоплатформний мобільний додаток для організації навчального процесу студентів

Дерево станів додатка

ІАЛЦ.467100.004 Д1

1 аркуш

Київ, 2023



			ІАЛЦ.467100.004 Д1	Арк.
				1
Змн.	Арк.	# документа		

ДОДАТОК 2

Двоплатформний мобільний додаток для організації навчального процесу студентів

Структура бази даних

ІАЛЦ.467100.005 Д2

1 аркуш

Київ, 2023

users

```
$id

name: [
    String $first,
    String $last
]
groupId?: String
info: String?
irrelevantEvents?: [String $id, ...]
chosenSubjects?: [String $id, ...]
recordedVisit: Timestamp
```

students

```
$groupId

$id: {
    name: [
        String $name,
        String $surname
    ]
    chosenSubjects: [String $id, ...]
}
...
```

events

```
$groupId

$id: {
    name: String
    subject?: String $id
    date: Timestamp
    hasTime: bool
    note?: String
}
...
```

subjects

```
$groupId

$id: {
    name: String
    isCommon: bool
}
...
```

details

```
$subjectId

info: {
    $id: {
        name: String
        content: String
    }
    ...
}
```

info

```
$groupId

$id: {
    name: String
    content: String
}
...
```

messages

```
$groupId

$id: {
    name: String
    content: String
    author: String $id
    date: Timestamp
}
...
```

ДОДАТОК З

Двоплатформний мобільний додаток для організації навчального процесу студентів

Код додатка

ІАЛЦ.467100.006 ДЗ

68 аркушів

Київ, 2023

main.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'states/authentication/authentication.dart';
import 'states/home/presentation/home.dart';
import 'states/identification/identification.dart';
import 'states/loading/loading.dart';


void main() async {
	runApp(const ProviderScope(child: App()));
}


enum AppState {
	loading,
	auth,
	identification,
	home
}

final appStateProvider = StateProvider<AppState>((ref) => AppState.loading);

class App extends StatelessWidget {
	const App();

	@override
	Widget build(BuildContext context) {
		return MaterialApp(
			title: 'Подійник',
			darkTheme: ThemeData(
				useMaterial3: true,
				colorScheme: const ColorScheme.dark(),
				canvasColor: const Color(0xff1a1a1a),
				snackBarTheme: const SnackBarThemeData(backgroundColor:
Color(0xff333333))
			),
			home: Consumer(builder: (context, ref, _) {
				switch (ref.watch(appStateProvider)) {
					case AppState.loading:
						return const Loading();
					case AppState.auth:
						return const Authentication();
					case AppState.identification:
						return const Identification();
					case AppState.home:
						return const Home();
				}
			})
```

```
            );
        }
    }
```

states/loading/loading.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';

import 'package:podiinyk/main.dart';

import 'package:podiinyk/core/data/firebase_options.dart';
import 'package:podiinyk/core/data/user_repository.dart';
import 'package:podiinyk/core/domain/user/state.dart';


class Loading extends ConsumerWidget {
    const Loading();

    @override
    Widget build(BuildContext context, WidgetRef ref) {
        _initApp(ref);
        return const Scaffold(body: Center(child: Icon(Icons.access_time)));
    }

    Future<void> _initApp(WidgetRef ref) async {
        await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);

        final authUser = FirebaseAuth.instance.currentUser;
        final AppState appState;

        if (authUser != null) {
            final userRepository = ref.read(userRepositoryProvider);
            userRepository.recordVisit();

            final user = await userRepository.user(id: authUser.uid);
            ref.read(initialUserProvider.notifier).state = user;
            appState = user.groupId != null ? AppState.home : AppState.identification;
        }
        else {
            appState = AppState.auth;
        }

        print('Loading: ${appState.name}');
        ref.read(appStateProvider.notifier).state = appState;
    }
}
```

states/authentication/authentication.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:firebase_auth/firebase_auth.dart' hide User;
import 'package:google_sign_in/google_sign_in.dart';

import 'package:podiinyk/main.dart';

import 'package:podiinyk/core/data/user_repository.dart';
import 'package:podiinyk/core/domain/user/state.dart';
import 'package:podiinyk/core/domain/user/user.dart';


class Authentication extends ConsumerWidget {
    const Authentication();

    @override
    Widget build(BuildContext context, WidgetRef ref) {
        return Scaffold(body: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
                const Text("Authentication"),
                Row(
                    mainAxisAlignment: MainAxisAlignment.spaceAround,
                    children: [
                        TextButton(
                            onPressed: () => _signInWithGoogle(ref),
                            child: const Text('Google')
                        ),
                        const TextButton(
                            onPressed: null,
                            child: Text('Apple')
                        )
                    ]
                )
            ]
        ));
    }

    Future<void> _signInWithGoogle(WidgetRef ref) async {
        final account = await GoogleSignIn().signIn();
        if (account == null) return;

        final accountAuth = await account.authentication;
        final authCred = GoogleAuthProvider.credential(
            idToken: accountAuth.idToken,
            accessToken: accountAuth.accessToken
);
```

```
                final userCred = await FirebaseAuth.instance.signInWithCredential(authCred);

                final userId = userCred.user!.uid;
                final userRepository = ref.read(userRepositoryProvider);
                final User user;
                final AppState appState;

                if (userCred.additionalUserInfo!.isNewUser) {
                        user = User(
                                id: userId,
                                firstName: account.displayName ?? '',
                                lastName: ''
                        );
                        await userRepository.initAccount(user);

                        appState = AppState.identification;
                        print('Authentication: new $userId');
                }
                else {
                        user = await userRepository.user(id: userId);
                        appState = user.groupId != null ? AppState.home : AppState.identification;
                        print('Authentication: existing $userId(${user.groupId})');
                }

                ref.read(initialUserProvider.notifier).state = user;
                ref.read(appStateProvider.notifier).state = appState;
        }
    }
```

states/identification/identification.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'joining_page.dart';
    import 'question_page.dart';
    import 'sharing_page.dart';


    enum IdentificationPage {
        question,
        sharing,
        joining
    }

    final identificationPageProvider = StateProvider.autoDispose<IdentificationPage>(
        (ref) => IdentificationPage.question
    );
```

```
class Identification extends ConsumerWidget {
	const Identification();

	@override
	Widget build(BuildContext context, WidgetRef ref) {
		switch (ref.watch(identificationPageProvider)) {
			case IdentificationPage.question:
				return const QuestionPage();
			case IdentificationPage.sharing:
				return const SharingPage();
			case IdentificationPage.joining:
				return const JoiningPage();
		}
	}
}
```

states/identification/question_page.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/core/domain/user/state.dart';
import 'identification.dart';


class QuestionPage extends ConsumerWidget {
	const QuestionPage();

	@override
	Widget build(BuildContext context, WidgetRef ref) {
		final page = ref.watch(identificationPageProvider.notifier);

		return GestureDetector(
			onLongPress: () {
				ref.read(userProvider.notifier).createGroup();
				page.state = IdentificationPage.sharing;
			},
			onDoubleTap: () => page.state = IdentificationPage.joining,
			child: Scaffold(body: Column(
				mainAxisAlignment: MainAxisAlignment.center,
				crossAxisAlignment: CrossAxisAlignment.start,
				children: const [
					Text("Group"),
					Text("Tap twice to join.\nLong press to create.")
				]
			))
		);
	}
}
```

states/identification/sharing_page.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/main.dart';
import 'package:podiinyk/core/domain/user/state.dart';


class SharingPage extends ConsumerWidget {
    const SharingPage();

    @override
    Widget build(BuildContext context, WidgetRef ref) {
        final user = ref.watch(userProvider);
        final groupExists = user.groupId != null;

        if (groupExists) _copyId(user.groupId);

        return GestureDetector(
            onDoubleTap: () {
                print('Identification: new ${user.groupId}');
                ref.read(appStateProvider.notifier).state = AppState.home;
            },
            onLongPress: () {
                if (groupExists) _copyId(user.groupId);
            },
            child: Scaffold(
                body: !groupExists ?
                    const Center(child: Text("Creating the group")) :
                    Column(
                        mainAxisAlignment: MainAxisAlignment.center,
                        crossAxisAlignment: CrossAxisAlignment.start,
                        children: const [
                            Text("Group"),
                            Text("Share the code on the clipboard.")
                        ]
                    )
            )
        );
    }

    Future<void> _copyId(String? id) => Clipboard.setData(ClipboardData(text: id));
}
```

states/identification/joining_page.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter_hooks/flutter_hooks.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/main.dart';
import 'package:podiinyk/core/domain/user/state.dart';


class JoiningPage extends HookConsumerWidget {
    const JoiningPage();

    @override
    Widget build(BuildContext context, WidgetRef ref) {
        final idField = useTextEditingController();

        return GestureDetector(
            onDoubleTap: () => _handleJoin(ref, idField.text),
            child: Scaffold(body: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                    const Text("Joining"),
                    TextField(
                        controller: idField,
                        decoration: const InputDecoration(labelText: "id"),
                    )
                ]
            ))
        );
    }

    Future<void> _handleJoin(WidgetRef ref, String id) async {
        if (id.isEmpty) return;

        await ref.read(userProvider.notifier).joinGroup(id);
        print("Identification: existing $id");
        ref.read(appStateProvider.notifier).state = AppState.home;
    }
}
```

states/home/presentation/home.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'widgets/tiles/drawer_tile.dart';
import 'state.dart';
```

```
class Home extends ConsumerWidget {
	const Home();

	@override
	Widget build(BuildContext context, WidgetRef ref) {
		return Scaffold(
			body: ref.watch(homeStateProvider).widget,
			drawer: Drawer(child: Center(child: ListView(
				shrinkWrap: true,
				children: const [
					Icon(Icons.all_inclusive),
					DrawerTile(Section.events),
					DrawerTile(Section.subjects),
					DrawerTile(Section.separate),
					DrawerTile(Section.messages),
					DrawerTile(Section.groupmates),
					DrawerTile(Section.settings),
				]
			))),
			drawerEdgeDragWidth: 80
		);
	}
}
```

states/home/presentation/state.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'sections/events/section.dart';
import 'sections/user/section.dart';
import 'sections/messages/section.dart';
import 'sections/separate/section.dart';
import 'sections/groupmates/section.dart';
import 'sections/subjects/section.dart';


enum Section {
	events(
		name: "events",
		icon: Icons.event,
		widget: EventsSection()
	),
	subjects(
		name: "subjects",
		icon: Icons.school,
		widget: SubjectsSection()
	),
	separate(
		name: "separate",
```

```
            icon: Icons.bubble_chart,
            widget: SeparateSection()
        ),
        messages(
            name: "messages",
            icon: Icons.chat,
            widget: MessagesSection()
        ),
        groupmates(
            name: "groupmates",
            icon: Icons.groups,
            widget: GroupmatesSection()
        ),
        settings(
            name: "me",
            icon: Icons.person,
            widget: UserSection()
        );

        const Section({
            required this.name,
            required this.icon,
            required this.widget
        });

        final String name;
        final IconData icon;
        final Widget widget;
    }

    final homeStateProvider = StateProvider<Section>((ref) => Section.events);
```

states/home/presentation/sections/events/section.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/providers/events.dart';

    import '../../state.dart';
    import '../../widgets/bars/section_bar.dart';

    import 'list.dart';


    class EventsSection extends ConsumerWidget {
        const EventsSection();
```

```
        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final user = ref.watch(userProvider);
                final events = ref.watch(eventsProvider)?.where(
                        (e) => e.subject == null || user.studies(e.subject!)
                );

                return Scaffold(
                        appBar: SectionBar(
                                section: Section.events,
                                count: events?.where(user.eventIsRelevant).length
                        ),
                        body: EventList(events)
                );
        }
    }
```

states/home/presentation/sections/events/list.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/entities/event.dart';

    import '../../widgets/entity_list.dart';
    import '../../widgets/tiles/entity_tile.dart';

    import 'form.dart';
    import 'page.dart';


    class EventList extends ConsumerWidget {
        const EventList(
                this.events, {
                        this.isExtendable = true,
                        this.showSubjects = true
                }
        );

        final Iterable<Event>? events;
        final bool isExtendable;
        final bool showSubjects;

        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final user = ref.watch(userProvider); final user = ref.watch(userProvider);

                final split = events?.where(user.eventIsRelevant).followedBy(
```

```
                events!.where((e) => !user.eventIsRelevant(e))
            );

            return EntityList<Event>(
                split,
                tile: (event) => Opacity(
                    opacity: user.eventIsRelevant(event) ? 1 : .5,
                    child: EntityTile(
                        title: event.name,
                        subtitle: showSubjects ? event.subject?.name : null,
                        trailing: event.date.shortRepr,
                        pageBuilder: (context) => EventPage(event)
                    )
                ),
                formBuilder: isExtendable ? (context) => const EventForm() : null
            );
        }
    }
```

states/home/presentation/sections/events/page.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/entities/event.dart';
    import '../../../domain/providers/events.dart';

    import '../../widgets/bars/action_bar.dart';
    import '../../widgets/bars/action_button.dart';


    class EventPage extends ConsumerWidget {
        const EventPage(this.event);

        final Event event;

        @override
        Widget build(BuildContext context, WidgetRef ref) {
            final user = ref.watch(userProvider);
            final isRelevant = user.eventIsRelevant(event);

            return Scaffold(body: SafeArea(child: Stack(children: [
                Center(child: ListView(
                    shrinkWrap: true,
                    children: [
                        Text(event.name),
                        if (event.subject != null) Text(event.subject!.name),
                        Text(event.date.repr),
```

```
                                        if (event.note != null) Text(event.note!)
                                ]
                        )),
                        ActionBar(children: [
                                ActionButton(
                                        icon: isRelevant ? Icons.check : Icons.undo,
                                        // do: inform about the Future
                                        action: () =>
    ref.read(userProvider.notifier).toggleEventIsRelevant(event)
                                ),
                                Consumer(builder: (context, ref, _) => ActionButton(
                                        icon: Icons.delete,
                                        action: () => _delete(context, ref)
                                ))
                        ])
                ])));
        }

        void _delete(BuildContext context, WidgetRef ref) {
                ref.read(eventsProvider.notifier).delete(event);
                Navigator.of(context).pop();
        }
    }
```

states/home/presentation/sections/events/form.dart:

```
    import 'package:flutter/material.dart';
    import 'package:flutter_hooks/flutter_hooks.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/id.dart';
    import 'package:podiinyk/core/domain/types/date.dart';
    import 'package:podiinyk/core/domain/user/state.dart';
    import 'package:podiinyk/core/domain/user/user.dart';

    import '../../../domain/entities/event.dart';
    import '../../../domain/entities/subject.dart';
    import '../../../domain/providers/events.dart';
    import '../../../domain/providers/subjects.dart';

    import '../../widgets/fields/date_field.dart';
    import '../../widgets/fields/option_field.dart';


    class EventForm extends HookConsumerWidget {
        const EventForm();

        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final user = ref.watch(userProvider);
```

```
        final subjects = ref.watch(subjectsProvider)!.where((s) => user.studies(s));

        final nameField = useTextEditingController();
        final subject = useRef<Subject?>(null);
        final date = useRef<Date?>(null);
        final noteField = useTextEditingController();

        return GestureDetector(
            onDoubleTap: () => _handleAdd(
                context,
                ref,
                user,
                nameField.text,
                subject.value,
                date.value,
                noteField.text
            ),
            child: Scaffold(body: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                    TextField(
                        controller: nameField,
                        decoration: const InputDecoration(labelText: 'name')
                    ),
                    if (subjects.isNotEmpty) OptionField<Subject>(
                        label: 'subject',
                        options: [
                            for (final subject in subjects)
                                MapEntry(subject.name, subject)
                        ],
                        onPick: (s) => subject.value = s,
                        isRequired: false
                    ),
                    DateField(onPick: (d) => date.value = d),
                    TextField(
                        controller: noteField,
                        decoration: const InputDecoration(labelText: 'note')
                    )
                ]
            ))
        );
    }

    void _handleAdd(
        BuildContext context,
        WidgetRef ref,
        User user,
        String name,
        Subject? subject,
        Date? date,
        String note
    ) {
```

```
            if (name.isEmpty || date == null) return;

            ref.read(eventsProvider.notifier).add(Event(
                id: newId(user: user),
                name: name,
                subject: subject,
                date: date,
                note: note.isEmpty ? null : note
            ));
            Navigator.of(context).pop();
        }
    }
```

states/home/presentation/sections/subjects/section.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/providers/events.dart';
    import '../../../domain/providers/subjects.dart';

    import '../../widgets/bars/section_bar.dart';
    import '../../state.dart';

    import 'list.dart';


    class SubjectsSection extends ConsumerWidget {
        const SubjectsSection();

        @override
        Widget build(BuildContext context, WidgetRef ref) {
            final user = ref.watch(userProvider);
            final subjects = ref.watch(subjectsProvider);
            final events = ref.watch(eventsProvider);

            return Scaffold(
                appBar: SectionBar(
                    section: Section.subjects,
                    // think: also display the number of unstudied subjects
                    count: subjects?.where(user.studies).length
                ),
                body: SubjectList(subjects, events: events)
            );
        }
    }
```

states/home/presentation/sections/subjects/list.dart:

```
import 'package:collection/collection.dart';
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/core/domain/user/state.dart';

import '../../../domain/entities/event.dart';
import '../../../domain/entities/subject.dart';

import '../../widgets/entity_list.dart';
import '../../widgets/tiles/entity_tile.dart';

import 'form.dart';
import 'page.dart';


class SubjectList extends ConsumerWidget {
	const SubjectList(
		this.subjects, {
			this.showNextEvent = true,
			this.events
		}
	);

	final Iterable<Subject>? subjects;
	final bool showNextEvent;
	final Iterable<Event>? events;

	@override
	Widget build(BuildContext context, WidgetRef ref) {
		final dataIsPresent = subjects != null && (events != null || showNextEvent == false);

		final user = ref.watch(userProvider);
		final split = subjects?.where(user.studies).followedBy(
			subjects!.where((s) => !user.studies(s))
		);

		return EntityList<Subject>(
			dataIsPresent ? split : null,
			tile: (subject) {
				final nextEvent = events?.firstWhereOrNull((e) => e.subject == subject);
				return Opacity(
					opacity: user.studies(subject) ? 1 : .5,
					child: EntityTile(
						title: subject.name,
						subtitle: nextEvent?.name,
						trailing: nextEvent?.date.shortRepr,
						pageBuilder: (context) => SubjectPage(subject)
					)
```

```
                        );
                    },
                    formBuilder: (context) => const SubjectForm()
                );
            }
        }
```

states/home/presentation/sections/subjects/page.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/entities/subject.dart';
    import '../../../domain/providers/events.dart';
    import '../../../domain/providers/subjects.dart';

    import '../../state.dart';

    import '../../widgets/bars/action_bar.dart';
    import '../../widgets/bars/action_button.dart';
    import '../../widgets/bars/counted_icon.dart';
    import '../../widgets/bars/entity_lists_tab_bar.dart';
    import '../../widgets/info/list.dart';

    import '../events/list.dart';
    import '../groupmates/list.dart';


    class SubjectPage extends ConsumerWidget {
        const SubjectPage(this.subject);

        final Subject subject;

        @override
        Widget build(BuildContext context, WidgetRef ref) {
            final user = ref.watch(userProvider);
            final details = ref.watch(subjectDetailsFamily(subject));
            final events = ref.watch(eventsProvider)?.where((e) => e.subject == subject);

            final isCommon = subject.isCommon;
            final isStudied = user.studies(subject);
            final students = details?.students;

            return ScaffoldMessenger(child: Scaffold(body: DefaultTabController(
                length: isCommon ? 2 : 3,
                child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
```

```
                                SafeArea(child: ActionBar(children: [
                                        if (!isCommon) ActionButton(
                                                icon: isStudied ?
                                                        Icons.local_library :
                                                        Icons.hotel,
                                                // do: inform about the Future
                                                action: () =>
ref.read(userProvider.notifier).toggleSubjectIsStudied(subject)
                                        ),
                                        Builder(builder: (context) => ActionButton(
                                                icon: Icons.delete,
                                                action: () => _showDeletingOptions(context, ref)
                                        ))
                                ])),
                                Text(subject.name),
                                EntityListsTabBar(tabIcons: [
                                        CountedIcon(
                                                icon: Icons.notes,
                                                count: details?.info.length
                                        ),
                                        CountedIcon(
                                                icon: Section.events.icon,
                                                count: events?.length
                                        ),
                                        if (!isCommon) CountedIcon(
                                                icon: Icons.people,
                                                count: students?.length
                                        )
                                ]),
                                Expanded(child: TabBarView(children: [
                                        InfoList(
                                                details?.info,
                                                subject: subject,
                                                isExtendable: isStudied
                                        ),
                                        EventList(
                                                events,
                                                isExtendable: isStudied,
                                                showSubjects: false
                                        ),
                                        if (!isCommon) GroupmateList(students)
                                ]))
                        ]
                )
        )));
    }

    void _showDeletingOptions(BuildContext context, WidgetRef ref) {
        ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Row(
                        mainAxisAlignment: MainAxisAlignment.spaceAround,
                        children: [
```

```
                                        IconButton(
                                                icon: const Icon(Icons.delete),
                                                onPressed: () => _delete(context, ref)
                                        ),
                                        IconButton(
                                                icon: const Icon(Icons.delete_sweep),
                                                onPressed: () => _clear(context, ref)
                                        )
                                ]
                        ))
                );
        }

        void _delete(BuildContext context, WidgetRef ref) {
                ref.read(subjectsProvider.notifier).delete(subject);
                Navigator.of(context).pop();
        }

        void _clear(BuildContext context, WidgetRef ref) {
                ref.read(subjectsProvider.notifier).clear();
                Navigator.of(context).pop();
        }
    }
```

states/home/presentation/sections/subjects/form.dart:

```
    import 'package:flutter/material.dart';
    import 'package:flutter_hooks/flutter_hooks.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/id.dart';
    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/entities/subject.dart';
    import '../../../domain/providers/subjects.dart';

    import '../../widgets/fields/option_field.dart';


    class SubjectForm extends HookConsumerWidget {
        const SubjectForm();

        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final nameField = useTextEditingController();
                final isCommon = useRef<bool?>(null);

                return GestureDetector(
                        onDoubleTap: () => _handleAdd(context, ref, nameField.text, isCommon.value),
                        child: Scaffold(body: Column(
```

```
                        mainAxisAlignment: MainAxisAlignment.center,
                        children: [
                                TextField(
                                        controller: nameField,
                                        decoration: const InputDecoration(labelText: 'name')
                                ),
                                OptionField<bool>(
                                        label: 'type',
                                        options: const [
                                                MapEntry('common', true),
                                                MapEntry('chosen', false)
                                        ],
                                        onPick: (value) => isCommon.value = value
                                )
                        ]
                ))
        );
    }

    void _handleAdd(BuildContext context, WidgetRef ref, String name, bool? isCommon) {
        if (name.isEmpty || isCommon == null) return;

        ref.read(subjectsProvider.notifier).add(Subject(
                id: newId(user: ref.read(userProvider)),
                name: name,
                isCommon: isCommon
        ));
        Navigator.of(context).pop();
    }
}
```

states/home/presentation/sections/separate/section.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import '../../../domain/providers/events.dart';
import '../../../domain/providers/info.dart';

import '../../widgets/bars/counted_icon.dart';
import '../../widgets/bars/entity_lists_tab_bar.dart';
import '../../widgets/bars/section_bar.dart';
import '../../widgets/info/list.dart';

import '../../state.dart';
import '../events/list.dart';


class SeparateSection extends ConsumerWidget {
    const SeparateSection();
```

```
        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final info = ref.watch(infoProvider);
                final events = ref.watch(eventsProvider)?.where((e) => e.subject == null);

                return DefaultTabController(
                        length: 2,
                        child: Scaffold(
                                appBar: SectionBar(
                                        section: Section.separate,
                                        bottom: EntityListsTabBar(tabIcons: [
                                                CountedIcon(
                                                        icon: Icons.notes,
                                                        count: info?.length
                                                ),
                                                CountedIcon(
                                                        icon: Section.events.icon,
                                                        count: events?.length
                                                )
                                        ])
                                ),
                                body: TabBarView(children: [
                                        InfoList(info),
                                        EventList(events),
                                ])
                        )
                );
        }
    }
```

states/home/presentation/sections/messages/section.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import '../../../domain/entities/message.dart';
    import '../../../domain/providers/messages.dart';

    import '../../state.dart';

    import '../../widgets/entity_list.dart';
    import '../../widgets/bars/section_bar.dart';
    import '../../widgets/tiles/entity_tile.dart';

    import 'form.dart';
    import 'page.dart';


    class MessagesSection extends ConsumerWidget {
        const MessagesSection();
```

```
        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final messages = ref.watch(messagesProvider);

                return Scaffold(
                        appBar: SectionBar(
                                section: Section.messages,
                                count: messages?.length
                        ),
                        body: EntityList<Message>(
                                messages,
                                tile: (message) => EntityTile(
                                        title: message.name,
                                        subtitle: message.author.fullName,
                                        trailing: message.date.shortRepr,
                                        pageBuilder: (context) => MessagePage(message)
                                ),
                                formBuilder: (context) => const MessageForm()
                        )
                );
        }
    }
```

states/home/presentation/sections/messages/page.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/entities/message.dart';
    import '../../../domain/providers/messages.dart';

    import '../../widgets/bars/action_bar.dart';
    import '../../widgets/bars/action_button.dart';


    class MessagePage extends ConsumerWidget {
        const MessagePage(this.message);

        final Message message;

        @override
        Widget build(BuildContext context, WidgetRef ref) {
                return Scaffold(body: SafeArea(child: Stack(children: [
                        Center(child: ListView(
                                shrinkWrap: true,
                                children: [
                                        Text(message.name),
                                        Text(message.author.fullName),
```

```
                                Text(message.date.repr),
                                Text(message.content),
                            ]
                        )),
                        if (ref.watch(userProvider).isAuthor(message)) ActionBar(children: [
                            Consumer(builder: (context, ref, _) => ActionButton(
                                icon: Icons.delete,
                                action: () => _delete(context, ref)
                            ))
                        ])
                ])));
        }

        void _delete(BuildContext context, WidgetRef ref) {
                ref.read(messagesProvider.notifier).delete(message);
                Navigator.of(context).pop();
        }
    }
```

states/home/presentation/sections/messages/form.dart:

```
    import 'package:flutter/material.dart';
    import 'package:flutter_hooks/flutter_hooks.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/id.dart';
    import 'package:podiinyk/core/domain/types/date.dart';
    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/entities/message.dart';
    import '../../../domain/providers/messages.dart';


    class MessageForm extends HookConsumerWidget {
        const MessageForm();

        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final nameField = useTextEditingController();
                final contentField = useTextEditingController();

                return GestureDetector(
                        onDoubleTap: () => _handleAdd(context, ref, nameField.text, contentField.text),
                        child: Scaffold(body: Column(
                                mainAxisAlignment: MainAxisAlignment.center,
                                children: [
                                        TextField(
                                                controller: nameField,
                                                decoration: const InputDecoration(labelText: 'topic')
                                        ),
```

```
                                        TextField(
                                                controller: contentField,
                                                decoration: const InputDecoration(labelText: 'content')
                                        )
                                ]
                        ))
                );
        }

        void _handleAdd(BuildContext context, WidgetRef ref, String name, String content) {
                if (name.isEmpty || content.isEmpty) return;

                final user = ref.read(userProvider);
                ref.read(messagesProvider.notifier).add(Message(
                        id: newId(user: user),
                        name: name,
                        content: content,
                        author: user.student,
                        date: Date.now()
                ));
                Navigator.of(context).pop();
        }
    }
```

states/home/presentation/sections/groupmates/section.dart:

```
    import 'package:flutter/material.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import '../../../domain/providers/students.dart';

    import '../../state.dart';
    import '../../widgets/bars/section_bar.dart';

    import 'list.dart';


    class GroupmatesSection extends ConsumerWidget {
        const GroupmatesSection();

        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final students = ref.watch(studentsProvider);

                return Scaffold(
                        appBar: SectionBar(
                                section: Section.groupmates,
                                count: students?.length
                        ),
                        body: GroupmateList(students)
```

```
            );
        }
    }
```

states/home/presentation/sections/groupmates/list.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/core/domain/user/state.dart';
import '../../../domain/entities/student.dart';

import '../../widgets/entity_list.dart';
import '../../widgets/tiles/entity_tile.dart';

import 'page.dart';


class GroupmateList extends ConsumerWidget {
    const GroupmateList(this.students);

    final Iterable<Student>? students;

    @override
    Widget build(BuildContext context, WidgetRef ref) {
        final user = ref.watch(userProvider);

        return EntityList<Student>(
            students?.where((s) => s.id != user.id),
            tile: (student) => EntityTile(
                title: student.fullName,
                pageBuilder: (context) => StudentPage(student)
            )
        );
    }
}
```

states/home/presentation/sections/groupmates/page.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/states/home/domain/providers/students.dart';

import '../../../domain/entities/student.dart';


class StudentPage extends ConsumerWidget {
```

```
        const StudentPage(this.student);

        final Student student;

        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final details = ref.watch(studentDetailsFamily(student));
                return Scaffold(body: Center(child: ListView(
                        shrinkWrap: true,
                        children: [
                                Text(student.fullName),
                                if (details != null) ...[
                                        if (details.info != null) Text(details.info!),
                                        // fix: SubjectList is a ListView and a child of another ListView
                                        // SubjectList(details.subjects, showNextEvent: false)
                                ]
                        ]
                )));
        }
    }
```

states/home/presentation/sections/user/section.dart:

```
    import 'package:flutter/material.dart';
    import 'package:flutter_hooks/flutter_hooks.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import '../../state.dart';
    import '../../widgets/bars/section_bar.dart';
    import 'editing_page.dart';
    import 'page.dart';


    final userIsEditingProvider = StateProvider.autoDispose<bool>((ref) => false);

    class UserSection extends HookConsumerWidget {
        const UserSection();

        @override
        Widget build(BuildContext context, WidgetRef ref) {
                final pages = usePageController();
                ref.listen(userIsEditingProvider, (_, bool isEditing) {
                        pages.animateToPage(
                                isEditing ? 1 : 0,
                                duration: const Duration(milliseconds: 200),
                                curve: Curves.easeInOutCubic
                        );
                });

                return Scaffold(
```

```
                appBar: const SectionBar(section: Section.settings),
                body: PageView(
                        controller: pages,
                        physics: const NeverScrollableScrollPhysics(),
                        children: const [
                                UserPage(),
                                UserEditingPage()
                        ]
                )
        );
    }
}
```

states/home/presentation/sections/user/page.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/core/domain/user/state.dart';

import '../../widgets/bars/action_bar.dart';
import '../../widgets/bars/action_button.dart';
import 'section.dart';


class UserPage extends ConsumerWidget {
    const UserPage();

    @override
    Widget build(BuildContext context, WidgetRef ref) {
        final user = ref.watch(userProvider);
        final userNotifier = ref.watch(userProvider.notifier);

        return Center(child: ListView(
                shrinkWrap: true,
                children: [
                        ActionBar(children: [ActionButton(
                                icon: Icons.edit,
                                action: () => ref.read(userIsEditingProvider.notifier).state = true
                        )]),
                        Text(user.student.fullName),
                        if (user.info != null) Text(user.info!),
                        ListTile(
                                title: Text(user.groupId!),
                                trailing: IconButton(
                                        icon: const Icon(Icons.clear),
                                        onPressed: userNotifier.leaveGroup
                                ),
```

```
                                onTap: () => Clipboard.setData(ClipboardData(text:
user.groupId))
                        ),
                        Row(
                                mainAxisAlignment: MainAxisAlignment.spaceAround,
                                children: [
                                        IconButton(
                                                icon: const Icon(Icons.logout),
                                                onPressed: userNotifier.signOut
                                        ),
                                        IconButton(
                                                icon: const Icon(Icons.person_off),
                                                onPressed: userNotifier.deleteAccount
                                        )
                                ]
                        )
                ]
        ));
    }
}
```

states/home/presentation/sections/user/editing_page.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter_hooks/flutter_hooks.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/core/domain/user/state.dart';
import 'package:podiinyk/core/domain/user/user.dart';

import 'section.dart';


class UserEditingPage extends HookConsumerWidget {
    const UserEditingPage();

    @override
    Widget build(BuildContext context, WidgetRef ref) {
        final user = ref.watch(userProvider);

        final firstNameField = useTextEditingController(text: user.firstName);
        final lastNameField = useTextEditingController(text: user.lastName);
        final infoField = useTextEditingController(text: user.info);

        return GestureDetector(
                onDoubleTap: () => _handleUpdate(ref, user, firstNameField.text,
lastNameField.text, infoField.text),
                child: Center(child: ListView(
                        shrinkWrap: true,
                        children: [
```

```
                                        Row(children: [
                                                Expanded(child: TextField(
                                                        controller: firstNameField,
                                                        decoration: const InputDecoration(labelText: 'first')
                                                )),
                                                Expanded(child: TextField(
                                                        controller: lastNameField,
                                                        decoration: const InputDecoration(labelText: 'last')
                                                ))
                                        ]),
                                        TextField(
                                                controller: infoField,
                                                decoration: const InputDecoration(labelText: 'info')
                                        )
                                ]
                        ))
                );
        }

        Future<void> _handleUpdate(
                WidgetRef ref,
                User user,
                String firstName,
                String lastName,
                String rawInfo
        ) async {
                final info = rawInfo.isNotEmpty ? rawInfo : null;
                final changed = firstName != user.firstName || lastName != user.lastName || info !=
    user.info;

                if (changed) {
                        await ref.read(userProvider.notifier).update(
                                firstName: firstName,
                                lastName: lastName,
                                info: info
                        );
                }

                ref.read(userIsEditingProvider.notifier).state = false;
        }
    }
```

states/home/presentation/widgets/entity_list.dart:

```
    import 'package:flutter/material.dart';

    import 'package:podiinyk/core/presentation/open_page.dart';

    import '../../domain/entities/entity.dart';
```

```
class EntityList<E extends Entity> extends StatelessWidget {
	const EntityList(
		this.entities, {
			required this.tile,
			this.formBuilder
		});

	final Iterable<E>? entities;
	final Widget Function(E) tile;
	final Widget Function(BuildContext)? formBuilder;

	@override
	Widget build(BuildContext context) {
		if (entities == null) return const Center(child: Icon(Icons.access_time));

		return Scaffold(
			body: ListView(
				padding: EdgeInsets.zero,
				children: [
					for (final entity in entities!) tile(entity)
				]
			),
			floatingActionButton: formBuilder != null ?
				FloatingActionButton(
					onPressed: () => openPage(
						context: context,
						builder: (context) => formBuilder!(context)
					),
					child: const Icon(Icons.add)
				) :
				null
		);
	}
}
```

states/home/presentation/widgets/info/list.dart:

```
import 'package:flutter/material.dart';

import '../../../domain/entities/info.dart';
import '../../../domain/entities/subject.dart';

import '../entity_list.dart';
import '../tiles/entity_tile.dart';

import 'form.dart';
import 'page.dart';
```

```
class InfoList extends StatelessWidget {
	const InfoList(
		this.info, {
			this.subject,
			this.isExtendable = true
		}
	);

	final Iterable<Info>? info;
	final Subject? subject;
	final bool isExtendable;

	@override
	Widget build(BuildContext context) {
		return EntityList<Info>(
			info,
			tile: (item) => EntityTile(
				title: item.name,
				pageBuilder: (context) => InfoPage(item)
			),
			formBuilder: isExtendable ?
				(context) => InfoForm(subject: subject) : null,
		);
	}
}
```

states/home/presentation/widgets/info/page.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import '../../../domain/entities/info.dart';
import '../../../domain/providers/info.dart';
import '../../../domain/providers/subjects.dart';

import '../bars/action_bar.dart';
import '../bars/action_button.dart';


class InfoPage extends StatelessWidget {
	const InfoPage(this.item);

	final Info item;

	@override
	Widget build(BuildContext context) {
		return Scaffold(body: SafeArea(child: Stack(children: [
			Center(child: ListView(
				shrinkWrap: true,
				children: [
```

```
                                Text(item.name),
                                Text(item.content),
                            ]
                        )),
                        ActionBar(children: [
                            Consumer(builder: (context, ref, _) => ActionButton(
                                icon: Icons.delete,
                                action: () => _delete(context, ref)
                            ))
                        ])
                ])));
        }

        void _delete(BuildContext context, WidgetRef ref) {
            if (item.subject != null) {
                ref.read(subjectDetailsFamily(item.subject!).notifier).deleteInfo(item);
            }
            else {
                ref.read(infoProvider.notifier).delete(item);
            }
            Navigator.of(context).pop();
        }
    }
```

states/home/presentation/widgets/info/form.dart:

```
    import 'package:flutter/material.dart';
    import 'package:flutter_hooks/flutter_hooks.dart';
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import 'package:podiinyk/core/domain/id.dart';
    import 'package:podiinyk/core/domain/user/state.dart';

    import '../../../domain/entities/info.dart';
    import '../../../domain/entities/subject.dart';
    import '../../../domain/providers/info.dart';
    import '../../../domain/providers/subjects.dart';


    class InfoForm extends HookConsumerWidget {
        const InfoForm({this.subject});

        final Subject? subject;

        @override
        Widget build(BuildContext context, WidgetRef ref) {
            final nameField = useTextEditingController();
            final contentField = useTextEditingController();

            return GestureDetector(
```

```
                onDoubleTap: () => _handleAdd(context, ref, nameField.text, contentField.text),
                child: Scaffold(body: Column(
                        mainAxisAlignment: MainAxisAlignment.center,
                        children: [
                                TextField(
                                        controller: nameField,
                                        decoration: const InputDecoration(labelText: 'name')
                                ),
                                TextField(
                                        controller: contentField,
                                        decoration: const InputDecoration(labelText: 'content')
                                )
                        ]
                ))
        );
    }

    void _handleAdd(BuildContext context, WidgetRef ref, String name, String content) {
        if (name.isEmpty || content.isEmpty) return;

        final item = Info(
                id: newId(user: ref.read(userProvider)),
                name: name,
                subject: subject,
                content: content
        );
        if (subject != null) {
                ref.read(subjectDetailsFamily(subject!).notifier).addInfo(item);
        }
        else {
                ref.read(infoProvider.notifier).add(item);
        }

        Navigator.of(context).pop();
    }
}
```

states/home/presentation/widgets/bars/section_bar.dart:

```
import 'package:flutter/material.dart';

import '../../state.dart';
import 'counted_icon.dart';


class SectionBar extends StatelessWidget implements PreferredSizeWidget {
    const SectionBar({
        required this.section,
        this.count,
        this.bottom
```

```
        });

        final Section section;
        final int? count;
        final PreferredSizeWidget? bottom;

        @override
        Size get preferredSize {
                double height = 56;
                if (bottom != null) height += bottom!.preferredSize.height;
                return Size.fromHeight(height);
        }

        @override
        Widget build(BuildContext context) {
                return AppBar(
                        title: Row(
                                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                                children: [
                                        Text(section.name),
                                        CountedIcon(icon: section.icon, count: count)
                                ]
                        ),
                        bottom: bottom
                );
        }
    }
```

states/home/presentation/widgets/bars/counted_icon.dart:

```
    import 'package:flutter/material.dart';


    class CountedIcon extends StatelessWidget {
        const CountedIcon({
                required this.icon,
                this.count
        });

        final IconData icon;
        final int? count;

        @override
        Widget build(BuildContext context) {
                return Row(
                        mainAxisAlignment: MainAxisAlignment.center,
                        children: [
                                if (count != null) Text(count.toString()),
                                Icon(icon)
                        ]
```

```
            );
        }
    }
```

states/home/presentation/widgets/bars/entity_list_tab_bar.dart:

```
import 'package:flutter/material.dart';


class EntityListsTabBar extends StatelessWidget implements PreferredSizeWidget {
        const EntityListsTabBar({required this.tabIcons});

        final List<Widget> tabIcons;

        @override
        Size get preferredSize => const Size.fromHeight(48);

        @override
        Widget build(BuildContext context) {
                return TabBar(tabs: [
                        for (final icon in tabIcons) Tab(child: icon),
                ]);
        }
}
```

states/home/presentation/widgets/bars/action_bar.dart:

```
    import 'package:flutter/material.dart';


    class ActionBar extends StatelessWidget {
        const ActionBar({required this.children});

        final List<Widget> children;

        @override
        Widget build(BuildContext context) {
                return SizedBox(
                        height: 56,
                        child: Row(
                                mainAxisAlignment: MainAxisAlignment.end,
                                children: children
                        )
                );
        }
    }
```

states/home/presentation/widgets/bars/action_button.dart:

```
import 'package:flutter/material.dart';


class ActionButton extends StatelessWidget {
	const ActionButton({
		required this.icon,
		required this.action
	});

	final IconData icon;
	final void Function() action;

	@override
	Widget build(BuildContext context) {
		return IconButton(
			icon: Icon(icon),
			onPressed: action
		);
	}
}
```

states/home/presentation/widgets/tiles/drawer_tile.dart:

```
import 'package:flutter/material.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import '../../state.dart';


class DrawerTile extends ConsumerWidget {
	const DrawerTile(this.section);

	final Section section;

	@override
	Widget build(BuildContext context, WidgetRef ref) {
		return ListTile(
			onTap: () {
				ref.read(homeStateProvider.notifier).state = section;
				Navigator.of(context).pop();
			},
			selected: ref.watch(homeStateProvider) == section,
			title: Text(section.name),
			leading: Icon(section.icon),
		);
	}
}
```

states/home/presentation/widgets/tiles/entity_tile.dart:

```
import 'package:flutter/material.dart';

import 'package:podiinyk/core/presentation/open_page.dart';


class EntityTile extends StatelessWidget {
	const EntityTile({
		required this.title,
		this.subtitle,
		this.trailing,
		required this.pageBuilder
	});

	final String title;
	final String? subtitle;
	final String? trailing;
	final Widget Function(BuildContext) pageBuilder;

	@override
	Widget build(BuildContext context) {
		return ListTile(
			title: Text(title),
			subtitle: subtitle != null ? Text(subtitle!) : null,
			trailing: trailing != null ? Text(trailing!) : null,
			onTap: () => openPage(
				context: context,
				builder: pageBuilder
			)
		);
	}
}
```

states/home/presentation/widgets/fields/option_field.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter_hooks/flutter_hooks.dart';

import 'package:podiinyk/core/presentation/open_page.dart';


class OptionField<O> extends HookWidget {
	const OptionField({
		required this.label,
		required this.options,
		required this.onPick,
		this.isRequired = true
	});
```

```
    final String label;
    final Iterable<MapEntry<String, O>> options;
    final void Function(O?) onPick;
    final bool isRequired;

    @override
    Widget build(BuildContext context) {
        final field = useTextEditingController();
        final current = useRef<O?>(null);
        if (isRequired) _onPick(field, options.first.key, current, options.first.value);

        return GestureDetector(
            onTap: () => _handleTap(context, field, current),
            child: TextField(
                controller: field,
                enabled: false,
                decoration: InputDecoration(labelText: label)
            )
        );
    }

    void _handleTap(
        BuildContext context,
        TextEditingController field,
        ObjectRef<O?> current
    ) {
        final shown = options.where((o) => o.value != current.value).toList();
        final nullIsShown = current.value != null && !isRequired;

        if (shown.length == 1 && !nullIsShown) {
            _onPick(field, shown.first.key, current, shown.first.value);
            return;
        }

        openPage(context: context, builder: (context) => _OptionsPage<O>(
            options: shown,
            nullIsShown: nullIsShown,
            onPick: (repr, option) => _onPick(field, repr, current, option)
        ));
    }

    void _onPick(TextEditingController field, String repr, ObjectRef<O?> current, O? option) {
        field.text = repr;
        current.value = option;
        onPick(option);
    }
}


class _OptionsPage<O> extends StatelessWidget {
    const _OptionsPage({
        required this.options,
```

```
            required this.nullIsShown,
            required this.onPick
        });

        final Iterable<MapEntry<String, O>> options;
        final bool nullIsShown;
        final void Function(String, O?) onPick;

        @override
        Widget build(BuildContext context) {
            final close = Navigator.of(context).pop;

            return Scaffold(body: Center(child: ListView(
                shrinkWrap: true,
                children: [
                    for (final option in options) ListTile(
                        onTap: () {
                            onPick(option.key, option.value);
                            close();
                        },
                        title: Text(option.key)
                    ),
                    if (nullIsShown) ListTile(
                        onTap: () {
                            onPick('', null);
                            close();
                        },
                        title: const Text('none')
                    )
                ]
            )));
        }
    }
```

states/home/presentation/widgets/fields/date_field.dart:

```
    import 'dart:math';

    import 'package:flutter/material.dart';
    import 'package:flutter_hooks/flutter_hooks.dart';

    import 'package:podiinyk/core/domain/types/date.dart';
    import 'package:podiinyk/core/domain/types/int.dart';
    import 'package:podiinyk/core/domain/types/date_time.dart';
    import 'package:podiinyk/core/presentation/open_page.dart';


    class DateField extends HookWidget {
        const DateField({
            this.initial,
```

```
            required this.onPick
        });

        final Date? initial;
        final void Function(Date) onPick;

        @override
        Widget build(BuildContext context) {
            final field = useTextEditingController(text: initial?.repr);
            final date = useRef(initial ?? Date.now(hasTime: false));

            return GestureDetector(
                onTap: () => openPage(
                    context: context,
                    builder: (context) => _DatePage(
                        initial: date.value,
                        onPick: (d) {
                            field.text = d.repr;
                            date.value = d;
                            onPick(d);
                        }
                    )
                ),
                child: TextField(
                    controller: field,
                    enabled: false,
                    decoration: const InputDecoration(labelText: 'date')
                )
            );
        }
    }


    class _DatePage extends HookWidget {
        const _DatePage({required this.initial, required this.onPick});

        final Date initial;
        final void Function(Date) onPick;
        static const minuteStep = 5;

        @override
        Widget build(BuildContext context) {
            final months = _months();
            final days = useValueNotifier(const <int>[]);
            final hours = useValueNotifier(const <int>[]);
            final minutes = useValueNotifier(const <int>[]);
            final timeIsIncluded = useValueNotifier(initial.hasTime);

            var monthObject = DateTime(initial.value.year, initial.value.month).latest(months.first);
            final date = useRef(_dateWithMonth(initial.value, monthObject, days, hours, minutes));

            return GestureDetector(
```

```
                onDoubleTap: () {
                        Navigator.of(context).pop();
                        onPick(Date(date.value, hasTime: timeIsIncluded.value));
                },
                child: Scaffold(body: Row(children: [
                        const Spacer(),
                        Flexible(child: HookBuilder(builder: (context) {
                                useListenable(days);
                                return _NumberWheel<int>(
                                        options: days.value,
                                        initial: date.value.day,
                                        optionRepr: (day) => day.twoDigitRepr,
                                        onPick: (day) => date.value = _dateWithDay(date.value,
day, hours, minutes)
                                );
                        })),
                        Flexible(child: _NumberWheel<DateTime>(
                                options: months,
                                optionRepr: (object) => object.month.twoDigitRepr,
                                onPick: (month) => date.value = _dateWithMonth(date.value,
month, days, hours, minutes)
                        )),
                        const Spacer(),
                        Flexible(child: GestureDetector(
                                onVerticalDragDown: (_) => timeIsIncluded.value = true,
                                child: HookBuilder(builder: (context) {
                                        useListenable(hours);
                                        useListenable(timeIsIncluded);

                                        return AnimatedOpacity(
                                                opacity: timeIsIncluded.value ? 1 : .5,
                                                duration: const Duration(milliseconds: 200),
                                                child: _NumberWheel<int>(
                                                        options: hours.value,
                                                        initial: date.value.hour,
                                                        optionRepr: (hour) => hour.twoDigitRepr,
                                                        onPick: (hour) => date.value =
_dateWithHour(date.value, hour, minutes)
                                                )
                                        );
                                })
                        )),
                        Flexible(child: GestureDetector(
                                onVerticalDragDown: (_) => timeIsIncluded.value = true,
                                child: HookBuilder(builder: (context) {
                                        useListenable(minutes);
                                        useListenable(timeIsIncluded);

                                        return AnimatedOpacity(
                                                opacity: timeIsIncluded.value ? 1 : .5,
                                                duration: const Duration(milliseconds: 200),
                                                child: _NumberWheel<int>(
```

```
                                                    options: minutes.value,
                                                    initial: date.value.minute,
                                                    optionRepr: (minute) =>
minute.twoDigitRepr,
                                                    onPick: (minute) => date.value =
date.value.copyWith(minute: minute)
                                                )
                                            );
                                        })
                                )),
                                const Spacer()
                        ]))
                );
        }

        DateTime _dateWithMonth(
                DateTime date,
                DateTime monthObject,
                ValueNotifier<List<int>> days,
                ValueNotifier<List<int>> hours,
                ValueNotifier<List<int>> minutes
        ) {
                days.value = _days(monthObject);
                final day = date.day.clamp(days.value.first, days.value.last);
                final dateWithMonth = date.copyWith(year: monthObject.year, month:
monthObject.month);
                return _dateWithDay(dateWithMonth, day, hours, minutes);
        }

        DateTime _dateWithDay(
                DateTime date,
                int day,
                ValueNotifier<List<int>> hours,
                ValueNotifier<List<int>> minutes
        ) {
                hours.value = _hours(date.copyWith(day: day, hour: 0, minute: 0));
                final hour = max(date.hour, hours.value.first);
                return _dateWithHour(date.copyWith(day: day), hour, minutes);
        }

        DateTime _dateWithHour(DateTime date, int hour, ValueNotifier<List<int>> minutes) {
                final hourObject = date.copyWith(hour: hour, minute: 0);
                minutes.value = _minutes(hourObject);
                final minute = max(date.minute, minutes.value.first);
                return hourObject.copyWith(minute: minute);
        }

        List<DateTime> _months() {
                final now =  DateTime.now();

                final currentMonthIsEmpty = now.day == now.monthDayCount &&
_dayIsEmpty(now.hour, now.minute);
```

```
            final start = !currentMonthIsEmpty ? now.month : now.month + 1;
            return [
                for (int month = start; month <= 12; month++)
                    DateTime(now.year, month),
                for (int month = 1; month < now.month; month++)
                    DateTime(now.year + 1, month),
                if (now.day != 1)
                    DateTime(now.year + 1, now.month)
            ];
    }

    List<int> _days(DateTime monthObject) {
            final now = DateTime.now();
            final currentMonthObject = DateTime(now.year, now.month);
            final nextCurrentMonthObject = DateTime(now.year + 1, now.month);

            final start = monthObject != currentMonthObject ? 1 :
                (!_dayIsEmpty(now.hour, now.minute) ? now.day : now.day + 1);
            final end = monthObject != nextCurrentMonthObject ? monthObject.monthDayCount :
now.day - 1;
            return [
                for (int day = start; day <= end; day++) day
            ];
    }

    bool _dayIsEmpty(int hour, int minute) => hour == 23 && _hourIsEmpty(minute);

    List<int> _hours(DateTime dayObject) {
            final now = DateTime.now();
            final today = DateTime(now.year, now.month, now.day);

            final start = dayObject != today ? 0 : (!_hourIsEmpty(now.minute) ? now.hour :
now.hour + 1);
            return [
                for (int hour = start; hour < 24; hour++) hour
            ];
    }

    bool _hourIsEmpty(int minute) => minute >= 60 - minuteStep;

    List<int> _minutes(DateTime hourObject) {
            final now = DateTime.now();
            final currentHour = DateTime(now.year, now.month, now.day, now.hour);

            final start = hourObject != currentHour ? 0 : now.minute + minuteStep - now.minute %
minuteStep;
            return [
                for (int minute = start; minute < 60; minute += minuteStep) minute
            ];
    }
}
```

```
class _NumberWheel<O> extends StatelessWidget {
	_NumberWheel({
		required this.options,
		O? initial,
		required this.optionRepr,
		required this.onPick
	}) :
		_initialIndex = initial != null ? options.indexOf(initial) : 0;

	final List<O> options;
	final String Function(O) optionRepr;
	final void Function(O) onPick;
	final int _initialIndex;

	@override
	Widget build(BuildContext context) {
		final wheel = FixedExtentScrollController(initialItem: _initialIndex);

		return NotificationListener<ScrollEndNotification>(
			onNotification: (notification) {
				onPick(options[wheel.selectedItem]);
				return true;
			},
			child: ListWheelScrollView(
				key: ValueKey(options),
				controller: wheel,
				physics: const FixedExtentScrollPhysics(),
				itemExtent: 56,
				diameterRatio: 1024,
				children: [
					for (final option in options) Text(
						optionRepr(option),
						style: const TextStyle(fontSize: 28)
					)
				]
			)
		);
	}
}
```

states/home/domain/providers/events.dart:

```
import 'package:hooks_riverpod/hooks_riverpod.dart';

import '../../data/repository.dart';
import '../entities/event.dart';
import '../entities/subject.dart';


class EventNotifier extends StateNotifier<List<Event>?> {
```

```
    EventNotifier({required this.repository}) : super(null) {
        if (repository != null) _init();
    }

    final HomeRepository? repository;

    Future<void> _init() async {
        final events = await repository!.events();
        state = events.toList()..sort();
    }

    Future<void> add(Event event) async {
        await repository!.addEvent(event);
        state = state!.toList()..add(event)..sort();
    }

    Future<void> delete(Event event) async {
        await repository!.deleteEvent(event);
        state = state!.toList()..remove(event);
    }

    void removeSubjectEvents(Subject subject) {
        state = state!.toList()..removeWhere((e) => e.subject == subject);
    }

    void removeAllSubjectEvents() {
        state = state!.toList()..removeWhere((e) => e.subject != null);
    }
}

final eventsProvider = StateNotifierProvider<EventNotifier, List<Event>?>(
    (ref) => EventNotifier(repository: ref.watch(homeRepositoryProvider))
);
```

states/home/domain/providers/subjects.dart:

```
import 'package:hooks_riverpod/hooks_riverpod.dart';

import '../../data/repository.dart';

import '../entities/info.dart';
import '../entities/subject.dart';

import 'events.dart';


class SubjectNotifier extends StateNotifier<List<Subject>?> {
    SubjectNotifier({
        required this.eventNotifier,
        required this.repository
```

```
    }) : super(null) {
        if (repository != null) _init();
    }

    final EventNotifier eventNotifier;
    final HomeRepository? repository;

    Future<void> _init() async {
        final subjects = await repository!.subjects();
        state = subjects.toList()..sort();
    }

    Future<void> add(Subject subject) async {
        await repository!.addSubject(subject);
        state = state!.toList()..add(subject)..sort();
    }

    Future<void> delete(Subject subject) async {
        await repository!.deleteSubject(subject);
        state = state!.toList()..remove(subject);
        eventNotifier.removeSubjectEvents(subject);
    }

    Future<void> clear() async {
        await repository!.clearSubjects();
        state = const <Subject>[];
        eventNotifier.removeAllSubjectEvents();
    }
}

final subjectsProvider = StateNotifierProvider<SubjectNotifier, List<Subject>?>(
    (ref) => SubjectNotifier(
        eventNotifier: ref.watch(eventsProvider.notifier),
        repository: ref.watch(homeRepositoryProvider),
    )
);


class SubjectDetailsNotifier extends StateNotifier<SubjectDetails?> {
    SubjectDetailsNotifier(this.subject, {required this.repository}) : super(null) {
        if (repository != null) _init();
    }

    final Subject subject;
    final HomeRepository? repository;

    Future<void> _init() async {
        final details = await repository!.subjectDetails(subject);
        state = SubjectDetails(
            info: details.info.toList()..sort(),
            students: details.students?.toList()?..sort()
        );
```

```
        }

        Future<void> addInfo(Info item) async {
                await repository!.addSubjectInfo(subject, item);
                state = state!.withInfo(state!.info.toList()..add(item)..sort());
        }

        Future<void> deleteInfo(Info item) async {
                await repository!.deleteSubjectInfo(subject, item);
                state = state!.withInfo(state!.info.toList()..remove(item));
        }
    }

    final subjectDetailsFamily = StateNotifierProvider.family<SubjectDetailsNotifier, SubjectDetails?,
    Subject>(
        (ref, subject) => SubjectDetailsNotifier(subject, repository:
    ref.watch(homeRepositoryProvider))
    );
```

states/home/domain/providers/info.dart:

```
    import 'package:hooks_riverpod/hooks_riverpod.dart';

    import '../../data/repository.dart';
    import '../entities/info.dart';


    class InfoNotifier extends StateNotifier<List<Info>?> {
        InfoNotifier({required this.repository}) : super(null) {
                if (repository != null) _init();
        }

        final HomeRepository? repository;

        Future<void> _init() async {
                final info = await repository!.info();
                state = info.toList()..sort();
        }

        Future<void> add(Info item) async {
                await repository!.addInfo(item);
                state = state!.toList()..add(item)..sort();
        }

        Future<void> delete(Info item) async {
                await repository!.deleteInfo(item);
                state = state!.toList()..remove(item);
        }
    }
```

```
final infoProvider = StateNotifierProvider<InfoNotifier, List<Info>?>(
    (ref) => InfoNotifier(repository: ref.watch(homeRepositoryProvider))
);
```

states/home/domain/providers/messages.dart:

```
import 'package:hooks_riverpod/hooks_riverpod.dart';

import '../../data/repository.dart';
import '../entities/message.dart';


class MessageNotifier extends StateNotifier<List<Message>?> {
    MessageNotifier({required this.repository}) : super(null) {
        if (repository != null) _init();
    }

    final HomeRepository? repository;

    Future<void> _init() async {
        final messages = await repository!.messages();
        state = messages.toList()..sort();
    }

    Future<void> add(Message message) async {
        await repository!.addMessage(message);
        state = state!.toList()..add(message)..sort();
    }

    Future<void> delete(Message message) async {
        await repository!.deleteMessage(message);
        state = state!.toList()..remove(message);
    }
}

final messagesProvider = StateNotifierProvider<MessageNotifier, List<Message>?>(
    (ref) => MessageNotifier(repository: ref.watch(homeRepositoryProvider))
);
```

states/home/domain/providers/students.dart:

```
import 'package:hooks_riverpod/hooks_riverpod.dart';

import '../../data/repository.dart';
import '../entities/student.dart';


class StudentNotifier extends StateNotifier<List<Student>?> {
```

```
        StudentNotifier({required this.repository}) : super(null) {
                if (repository != null) _init();
        }

        final HomeRepository? repository;

        Future<void> _init() async {
                final students = await repository!.students();
                state = students.toList()..sort();
        }
    }

    final studentsProvider = StateNotifierProvider<StudentNotifier, List<Student>?>(
        (ref) => StudentNotifier(repository: ref.watch(homeRepositoryProvider))
    );


    class StudentDetailsNotifier extends StateNotifier<StudentDetails?> {
        StudentDetailsNotifier(this.student, {required this.repository}) : super(null) {
                if (repository != null) _init();
        }

        final Student student;
        final HomeRepository? repository;

        Future<void> _init() async {
                final details = await repository!.studentDetails(student);
                state = details.withSubjects(details.subjects.toList()..sort());
        }
    }

    final studentDetailsFamily = StateNotifierProvider.family<StudentDetailsNotifier, StudentDetails?,
    Student>(
        (ref, student) => StudentDetailsNotifier(student, repository:
    ref.watch(homeRepositoryProvider))
    );
```

states/home/domain/entities/entity.dart:

```
    import 'package:flutter/material.dart';


    @immutable
    abstract class Entity implements Comparable {
        const Entity({required this.id});

        final String id;

        @override
        bool operator ==(Object other) => other is Entity && id == other.id;
```

```
    @override
    int get hashCode => id.hashCode;
}
```

states/home/domain/entities/event.dart:

```
import 'package:podiinyk/core/domain/types/date.dart';

import 'entity.dart';
import 'subject.dart';


class Event extends Entity {
    const Event({
        required String id,
        required this.name,
        this.subject,
        required this.date,
        this.note
    }) :
        super(id: id);

    final String name;
    final Subject? subject;
    final Date date;
    final String? note;

    @override
    int compareTo(covariant Event other) => date.compareTo(other.date);
}
```

states/home/domain/entities/subject.dart:

```
import 'entity.dart';
import 'info.dart';
import 'student.dart';


class Subject extends Entity {
    const Subject({
        required String id,
        required this.name,
        required this.isCommon
    }) :
        super(id: id);

    final String name;
    final bool isCommon;
```

```
    @override
    int compareTo(covariant Subject other) => name.compareTo(other.name);
}


class SubjectDetails {
    const SubjectDetails({
        required this.info,
        this.students
    });

    final Iterable<Info> info;
    final Iterable<Student>? students;

    SubjectDetails withInfo(Iterable<Info> info) => SubjectDetails(
        info: info,
        students: students
    );
}
```

states/home/domain/entities/info.dart:

```
import 'entity.dart';
import 'subject.dart';


class Info extends Entity {
    const Info({
        required String id,
        required this.name,
        this.subject,
        required this.content
    }) :
        super(id: id);

    final String name;
    final Subject? subject;
    final String content;

    @override
    int compareTo(covariant Info other) => name.compareTo(other.name);
}
```

states/home/domain/entities/message.dart:

```
import 'package:podiinyk/core/domain/types/date.dart';

import 'entity.dart';
```

```
import 'student.dart';


class Message extends Entity {
	const Message({
		required String id,
		required this.name,
		required this.content,
		required this.author,
		required this.date,
	}) :
		super(id: id);

	final String name;
	final String content;
	final Student author;
	final Date date;

	@override
	int compareTo(covariant Message other) => -date.compareTo(other.date);
}
```

states/home/domain/entities/student.dart:

```
import 'entity.dart';
import 'subject.dart';


class Student extends Entity {
	const Student({
		required String id,
		required this.firstName,
		required this.lastName,
		required this.chosenSubjectIds
	}) :
		super(id: id);

	final String firstName;
	final String lastName;
	final Set<String> chosenSubjectIds;

	String get fullName => '$firstName $lastName';

	bool chose(Subject subject) => chosenSubjectIds.contains(subject.id);

	@override
	int compareTo(covariant Student other) => lastName.compareTo(other.lastName);
}
```

```
class StudentDetails {
	const StudentDetails({
		this.info,
		required this.subjects
	});

	final String? info;
	final Iterable<Subject> subjects;

	StudentDetails withSubjects(Iterable<Subject> subjects) => StudentDetails(
		info: info,
		subjects: subjects
	);
}
```

states/home/data/repository.dart:

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/core/data/types/document.dart';
import 'package:podiinyk/core/data/types/field.dart';
import 'package:podiinyk/core/data/types/object_map.dart';
import 'package:podiinyk/core/data/user_doc_ref.dart';
import 'package:podiinyk/core/domain/types/date.dart';
import 'package:podiinyk/core/domain/user/state.dart';

import '../domain/entities/event.dart';
import '../domain/entities/info.dart';
import '../domain/entities/message.dart';
import '../domain/entities/student.dart';
import '../domain/entities/subject.dart';


class HomeRepository {
	const HomeRepository({required this.groupId});

	final String groupId;

	Future<void> addEvent(Event event) async {
		await _ref(Document.events).update({
			event.id: {
				Field.name.name: event.name,
				if (event.subject != null) Field.subject.name: event.subject?.id,
				Field.date.name: event.date.value,
				Field.hasTime.name: event.date.hasTime,
				if (event.note != null) Field.note.name: event.note
			}
		});
	}
```

```
Future<void> addSubject(Subject subject) async {
	await Future.wait([
		_ref(Document.subjects).update({
			subject.id: {
				Field.name.name: subject.name,
				Field.isCommon.name: subject.isCommon
			}
		}),
		_subjectDetailsRef(subject).set({
			Field.info.name: const <String, ObjectMap>{}
		})
	]);
}

Future<void> addSubjectInfo(Subject subject, Info item) async {
	await _subjectDetailsRef(subject).update({
		'${Field.info.name}.${item.id}': {
			Field.name.name: item.name,
			Field.content.name: item.content
		}
	});
}

Future<void> addInfo(Info item) async {
	await _ref(Document.info).update({
		item.id: {
			Field.name.name: item.name,
			Field.content.name: item.content
		}
	});
}

Future<void> addMessage(Message message) async {
	await _ref(Document.messages).update({
		message.id: {
			Field.name.name: message.name,
			Field.content.name: message.content,
			Field.author.name: message.author.id,
			Field.date.name: message.date.value
		}
	});
}

Future<Iterable<Event>> events() async {
	late final DocumentSnapshot<ObjectMap> snapshot;
	late final Iterable<Subject> subjects;
	await Future.wait([
		_ref(Document.events).get().then((s) => snapshot = s),
		this.subjects().then((s) => subjects = s)
	]);

	return snapshot.data()!.entries.map((entry) => Event(
```

```
                id: entry.key,
                name: entry.value[Field.name.name],
                subject: entry.value.containsKey(Field.subject.name) ?
                        subjects.firstWhere((s) => s.id == entry.value[Field.subject.name]) :
                        null,
                date: Date(
                        (entry.value[Field.date.name] as Timestamp).toDate(),
                        hasTime: entry.value[Field.hasTime.name]
                ),
                note: entry.value[Field.note.name]
        ));
}

Future<Iterable<Subject>> subjects() async {
        final snapshot = await _ref(Document.subjects).get();
        return snapshot.data()!.entries.map((entry) => Subject(
                id: entry.key,
                name: entry.value[Field.name.name],
                isCommon: entry.value[Field.isCommon.name]
        ));
}

Future<SubjectDetails> subjectDetails(Subject subject) async {
        late final DocumentSnapshot<ObjectMap> snapshot;
        Iterable<Student>? students;
        await Future.wait([
                _subjectDetailsRef(subject).get().then((s) => snapshot = s),
                if (!subject.isCommon) this.students().then((s) => students = s)
        ]);

        final infoMap = Map<String, ObjectMap>.from(snapshot.data()![Field.info.name]);
        final info = infoMap.entries.map((entry) => Info(
                id: entry.key,
                name: entry.value[Field.name.name],
                subject: subject,
                content: entry.value[Field.content.name]
        ));

        return SubjectDetails(
                info: info,
                students: students?.where((s) => s.chose(subject))
        );
}

Future<Iterable<Info>> info() async {
        final snapshot = await _ref(Document.info).get();
        return snapshot.data()!.entries.map((entry) => Info(
                id: entry.key,
                name: entry.value[Field.name.name],
                content: entry.value[Field.content.name]
        ));
}
```

```
    Future<Iterable<Message>> messages() async {
        late final DocumentSnapshot<ObjectMap> snapshot;
        late final Iterable<Student> students;
        await Future.wait([
            _ref(Document.messages).get().then((s) => snapshot = s),
            this.students().then((s) => students = s)
        ]);

        return snapshot.data()!.entries.map((entry) => Message(
            id: entry.key,
            name: entry.value[Field.name.name],
            content: entry.value[Field.content.name],
            author: students.firstWhere((s) => s.id == entry.value[Field.author.name]),
            date: Date((entry.value[Field.date.name] as Timestamp).toDate())
        ));
    }

    Future<Iterable<Student>> students() async {
        final snapshot = await _ref(Document.students).get();
        return snapshot.data()!.entries.map((entry) {
            final name = entry.value[Field.name.name] as List<dynamic>;
            return Student(
                id: entry.key,
                firstName: name.first,
                lastName: name.last,
                chosenSubjectIds:
Set<String>.from(entry.value[Field.chosenSubjects.name])
            );
        });
    }

    Future<StudentDetails> studentDetails(Student student) async {
        late final DocumentSnapshot<ObjectMap> snapshot;
        late final Iterable<Subject> subjects;
        await Future.wait([
            userDocRef(student.id).get().then((s) => snapshot = s),
            this.subjects().then((s) => subjects = s)
        ]);

        final map = snapshot.data()!;
        final subjectIds = map[Field.chosenSubjects.name] as List<dynamic>;

        return StudentDetails(
            info: map[Field.info.name],
            subjects: subjects.where((s) => subjectIds.contains(s.id))
        );
    }

    Future<void> deleteEvent(Event event) async {
        await _ref(Document.events).update({
            event.id: FieldValue.delete()
        });
```

```
    }

    Future<void> deleteSubject(Subject subject) async {
        final eventsRef = _ref(Document.events);
        final eventsSnapshot = await eventsRef.get();
        final eventEntries = eventsSnapshot.data()!.entries.where(
            (e) => e.value[Field.subject.name] == subject.id
        );

        await Future.wait([
            _ref(Document.subjects).update({
                subject.id: FieldValue.delete()
            }),
            _subjectDetailsRef(subject).delete(),
            eventsRef.update({
                for (final entry in eventEntries) entry.key: FieldValue.delete()
            })
        ]);
    }

    Future<void> clearSubjects() async {
        final eventsRef = _ref(Document.events);
        final eventsSnapshot = await eventsRef.get();
        final eventEntries = eventsSnapshot.data()!.entries.where(
            (e) => e.value[Field.subject.name] != null
        );

        await Future.wait([
            _ref(Document.subjects).set({}),
            eventsRef.update({
                for (final entry in eventEntries) entry.key: FieldValue.delete()
            })
        ]);
    }

    Future<void> deleteSubjectInfo(Subject subject, Info item) async {
        await _subjectDetailsRef(subject).update({
            '${Field.info.name}.${item.id}': FieldValue.delete()
        });
    }

    Future<void> deleteInfo(Info item) async {
        await _ref(Document.info).update({
            item.id: FieldValue.delete()
        });
    }

    Future<void> deleteMessage(Message message) async {
        await _ref(Document.messages).update({
            message.id: FieldValue.delete()
        });
    }
```

```
    DocumentReference<ObjectMap> _ref(Document document) => document.ref(groupId);

        DocumentReference<ObjectMap> _subjectDetailsRef(Subject subject) {
                return _ref(Document.subjects).collection('details').doc(subject.id);
        }
    }

    final homeRepositoryProvider = Provider<HomeRepository?>((ref) {
        final groupId = ref.watch(userProvider).groupId;
        return groupId != null ? HomeRepository(groupId: groupId) : null;
    });
```

core/presentation/open_page.dart:

```
    import 'package:flutter/material.dart';


    void openPage({
        required BuildContext context,
        required Widget Function(BuildContext) builder
    }) {
        final navigator = Navigator.of(context);
        navigator.push(MaterialPageRoute(
                builder: (context) => builder(context)
        ));
    }
```

core/domain/id.dart:

```
    import 'user/user.dart';


    String newId({required User user}) {
        final timeComponent = DateTime.now().microsecondsSinceEpoch.toRadixString(36);
        final userComponent = user.id.substring(0, 2);
        return ('$timeComponent$userComponent'.split('')..shuffle()).join().toUpperCase();
    }
```

core/domain/user/user.dart:

```
    import 'package:podiinyk/states/home/domain/entities/event.dart';
    import 'package:podiinyk/states/home/domain/entities/message.dart';
    import 'package:podiinyk/states/home/domain/entities/student.dart';
    import 'package:podiinyk/states/home/domain/entities/subject.dart';


    class User {
```

```
    const User({
        required this.id,
        required this.firstName,
        required this.lastName,
        this.info,
        this.groupId,
        this.irrelevantEventIds,
        this.chosenSubjectIds
    });

    final String id;
    final String firstName;
    final String lastName;
    final String? info;
    final String? groupId;
    final Set<String>? irrelevantEventIds;
    final Set<String>? chosenSubjectIds;

    Student get student => Student(
        id: id,
        firstName: firstName,
        lastName: lastName,
        chosenSubjectIds: chosenSubjectIds!
    );

    bool eventIsRelevant(Event event) => !irrelevantEventIds!.contains(event.id);

    bool studies(Subject subject) {
        return subject.isCommon || chosenSubjectIds!.contains(subject.id);
    }

    bool isAuthor(Message message) => id == message.author.id;

    User copyWith({
        String? firstName,
        String? lastName,
        required String? info,
        required String? groupId,
        required Set<String>? irrelevantEventIds,
        required Set<String>? chosenSubjectIds
    }) => User(
        id: id,
        firstName: firstName ?? this.firstName,
        lastName: lastName ?? this.lastName,
        info: info,
        groupId: groupId,
        irrelevantEventIds: irrelevantEventIds,
        chosenSubjectIds: chosenSubjectIds
    );
}
```

core/domain/user/state.dart:

```
import 'package:firebase_auth/firebase_auth.dart' hide User;
import 'package:hooks_riverpod/hooks_riverpod.dart';

import 'package:podiinyk/main.dart';

import 'package:podiinyk/states/home/domain/entities/event.dart';
import 'package:podiinyk/states/home/domain/entities/subject.dart';

import '../../data/user_repository.dart';
import '../id.dart';
import 'user.dart';


final initialUserProvider = StateProvider<User?>((ref) => null);


class UserNotifier extends StateNotifier<User> {
	UserNotifier({
		required User initial,
		required this.repository,
		required this.appStateController
	}) :
		super(initial);

	final UserRepository repository;
	final StateController<AppState> appStateController;

	Future<void> createGroup() async {
		final id = newId(user: state);
		final user = state.copyWith(
			info: state.info,
			groupId: id,
			irrelevantEventIds: <String>{},
			chosenSubjectIds: <String>{}
		);
		await repository.createGroup(user: user);
		state = user;
	}

	Future<void> joinGroup(String id) async {
		final user = state.copyWith(
			info: state.info,
			groupId: id,
			irrelevantEventIds: <String>{},
			chosenSubjectIds: <String>{}
		);
		await repository.joinGroup(user: user);
		state = user;
	}
```

```
    Future<void> toggleEventIsRelevant(Event event) async {
        final ids = state.irrelevantEventIds!;

        if (state.eventIsRelevant(event)) {
            ids.add(event.id);
        }
        else {
            ids.remove(event.id);
        }

        await _update(
            info: state.info,
            irrelevantEventIds: ids,
            chosenSubjectIds: state.chosenSubjectIds
        );
    }

    Future<void> toggleSubjectIsStudied(Subject subject) async {
        final ids = state.chosenSubjectIds!;

        if (!state.studies(subject)) {
            ids.add(subject.id);
        }
        else {
            ids.remove(subject.id);
        }

        await _update(
            info: state.info,
            irrelevantEventIds: state.irrelevantEventIds,
            chosenSubjectIds: ids
        );
    }

    Future<void> update({String? firstName, String? lastName, required String? info}) =>
_update(
        firstName: firstName,
        lastName: lastName,
        info: info,
        irrelevantEventIds: state.irrelevantEventIds,
        chosenSubjectIds: state.chosenSubjectIds
    );

    Future<void> _update({
        String? firstName,
        String? lastName,
        required String? info,
        required Set<String>? irrelevantEventIds,
        required Set<String>? chosenSubjectIds
    }) async {
        final user = state.copyWith(
            firstName: firstName,
```

```
                lastName: lastName,
                info: info,
                groupId: state.groupId,
                irrelevantEventIds: irrelevantEventIds,
                chosenSubjectIds: chosenSubjectIds
            );
            await repository.update(user);
            state = user;
        }

        Future<void> leaveGroup() async {
            await repository.leaveGroup(user: state);
            state = state.copyWith(
                info: state.info,
                groupId: null,
                irrelevantEventIds: null,
                chosenSubjectIds: null
            );
            appStateController.state = AppState.identification;
        }

        Future<void> signOut() async {
            await FirebaseAuth.instance.signOut();
            appStateController.state = AppState.auth;
        }

        Future<void> deleteAccount() async {
            await repository.deleteAccount(state);
            appStateController.state = AppState.auth;
        }
    }

    final userProvider = StateNotifierProvider<UserNotifier, User>(
        (ref) => UserNotifier(
            initial: ref.watch(initialUserProvider)!,
            repository: ref.watch(userRepositoryProvider),
            appStateController: ref.watch(appStateProvider.notifier)
        )
    );
```

core/domain/types/int.dart:

```
    extension IntExtension on int {
        String get twoDigitRepr => toString().padLeft(2, '0');
    }
```

core/domain/types/date.dart:

```
import 'int.dart';


class Date implements Comparable {
	const Date(this.value, {this.hasTime = true});

	Date.now({this.hasTime = true}) :
		value = DateTime.now();

	final DateTime value;
	final bool hasTime;

	String get shortRepr => _dayRepr(implyToday: true);

	String get repr {
		final dayRepr = _dayRepr(includeWeekday: true);
		return hasTime ? '$dayRepr, $timeRepr' : dayRepr;
	}

	String _dayRepr({bool includeWeekday = false, bool implyToday = false}) {
		final day = DateTime(value.year, value.month, value.day);

		final now = DateTime.now();
		final today = DateTime(now.year, now.month, now.day);
		final tomorrow = DateTime(now.year, now.month, now.day + 1);
		final yesterday = DateTime(now.year, now.month, now.day - 1);

		if (day.isAfter(tomorrow) || day.isBefore(yesterday)) {
			return !includeWeekday ? dateRepr : '$weekdayRepr $dateRepr';
		}

		if (day == tomorrow) return 'tomorrow';

		if (day == today) {
			return !implyToday || !hasTime ? 'today' : timeRepr;
		}

		return 'yesterday';
	}

	String get dateRepr => '${value.day.twoDigitRepr}.${value.month.twoDigitRepr}';

	String get timeRepr => '${value.hour.twoDigitRepr}:${value.minute.twoDigitRepr}';

	String get weekdayRepr {
		switch (value.weekday) {
			case 1: return 'Monday';
			case 2: return 'Tuesday';
			case 3: return 'Wednesday';
```

```
                case 4: return 'Thursday';
                case 5: return 'Friday';
                case 6: return 'Saturday';
                default: return 'Sunday';
            }
        }

        @override
        int compareTo(covariant Date other) => value.compareTo(other.value);
    }
```

core/domain/types/date_time.dart:

```
    extension DateTimeExtension on DateTime {
        int get monthDayCount {
            switch (month) {
                case DateTime.september: return 30;
                case DateTime.october: return 31;
                case DateTime.november: return 30;

                case DateTime.december: return 31;
                case DateTime.january: return 31;
                case DateTime.february:
                    final yearIsLeap = year % 4 == 0 && (year % 100 != 0 || year % 400 ==
    0);
                    return !yearIsLeap ? 28 : 29;

                case DateTime.march: return 31;
                case DateTime.april: return 30;
                case DateTime.may: return 31;

                case DateTime.june: return 30;
                case DateTime.july: return 31;
                default: return 31;
            }
        }

        DateTime latest(DateTime other) => other.isAfter(this) ? other : this;

        DateTime copyWith({
            int? year,
            int? month,
            int? day,
            int? hour,
            int? minute
        }) => DateTime(
            year ?? this.year,
            month ?? this.month,
            day ?? this.day,
            hour ?? this.hour,
```

```
            minute ?? this.minute
        );
    }
```

core/data/user_repository.dart:

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart' hide User;
import 'package:hooks_riverpod/hooks_riverpod.dart';

import '../domain/user/user.dart';

import 'types/document.dart';
import 'types/field.dart';
import 'types/object_map.dart';

import 'user_doc_ref.dart';


class UserRepository {
    const UserRepository();

    Future<void> initAccount(User user) async {
        await userDocRef(user.id).set({
            Field.name.name: [user.firstName, user.lastName]
        });
    }

    Future<User> user({required String id}) async {
        final snapshot = await userDocRef(id).get();
        final map = snapshot.data()!;

        final name = map[Field.name.name] as List<dynamic>;
        final groupId = map[Field.groupId.name];
        final hasGroup = groupId != null;

        return User(
            id: id,
            firstName: name.first,
            lastName: name.last,
            info: map[Field.info.name],
            groupId: groupId,
            irrelevantEventIds: hasGroup ?
Set<String>.from(map[Field.irrelevantEvents.name]) : null,
            chosenSubjectIds: hasGroup ?
Set<String>.from(map[Field.chosenSubjects.name]) : null
        );
    }
```

```
Future<void> recordVisit() async {

}

Future<void> createGroup({required User user}) async {
	final groupId = user.groupId!;
	final chosenSubjectIds = user.chosenSubjectIds!.toList();
	const emptyMap = <String, ObjectMap>{};

	await Future.wait([
		userDocRef(user.id).update({
			Field.groupId.name: groupId,
			Field.irrelevantEvents.name: user.irrelevantEventIds!.toList(),
			Field.chosenSubjects.name: chosenSubjectIds
		}),
		Document.events.ref(groupId).set(emptyMap),
		Document.subjects.ref(groupId).set(emptyMap),
		Document.info.ref(groupId).set(emptyMap),
		Document.messages.ref(groupId).set(emptyMap),
		Document.students.ref(groupId).set({
			user.id: {
				Field.name.name: [user.firstName, user.lastName],
				Field.chosenSubjects.name: chosenSubjectIds
			}
		})
	]);
}

Future<void> joinGroup({required User user}) async {
	final chosenSubjectIds = user.chosenSubjectIds!.toList();

	await Future.wait([
		userDocRef(user.id).update({
			Field.groupId.name: user.groupId,
			Field.irrelevantEvents.name: user.irrelevantEventIds!.toList(),
			Field.chosenSubjects.name: chosenSubjectIds
		}),
		Document.students.ref(user.groupId!).update({
			user.id: {
				Field.name.name: [user.firstName, user.lastName],
				Field.chosenSubjects.name: chosenSubjectIds
			}
		})
	]);
}

Future<void> update(User user) async {
	final name = [user.firstName, user.lastName];
	final chosenSubjectIds = user.chosenSubjectIds?.toList();
	final hasGroup = user.groupId != null;

	await Future.wait([
```

```
                userDocRef(user.id).update({
                        Field.name.name: name,
                        Field.info.name: user.info,
                        if (hasGroup) Field.irrelevantEvents.name:
user.irrelevantEventIds!.toList(),
                        if (hasGroup) Field.chosenSubjects.name: chosenSubjectIds
                }),
                Document.students.ref(user.groupId!).update({
                        '${user.id}.${Field.name.name}': name,
                        if (hasGroup) '${user.id}.${Field.chosenSubjects.name}':
chosenSubjectIds
                })
        ]);
    }

    Future<void> leaveGroup({required User user}) async {
        await Future.wait([
                userDocRef(user.id).update({
                        Field.groupId.name: FieldValue.delete(),
                        Field.irrelevantEvents.name: FieldValue.delete(),
                        Field.chosenSubjects.name: FieldValue.delete()
                }),
                Document.students.ref(user.groupId!).update({
                        user.id: FieldValue.delete()
                }),
        ]);
    }

    Future<void> deleteAccount(User user) async {
        final groupId = user.groupId!;
        final messagesSnapshot = await Document.messages.ref(groupId).get();
        final messageEntries = messagesSnapshot.data()!.entries.where(
                (entry) => entry.value[Field.author.name] == user.id
        );

        await Future.wait([
                FirebaseAuth.instance.currentUser!.delete(),
                userDocRef(user.id).delete(),
                Document.students.ref(groupId).update({
                        user.id: FieldValue.delete()
                }),
                Document.messages.ref(groupId).update({
                        for (final entry in messageEntries) entry.key: FieldValue.delete()
                })
        ]);
    }
}

final userRepositoryProvider = Provider<UserRepository>(
    (ref) => const UserRepository()
);
```

core/data/user_doc_ref.dart:

```
import 'package:cloud_firestore/cloud_firestore.dart';

import 'types/object_map.dart';


DocumentReference<ObjectMap> userDocRef(String id) {
    return FirebaseFirestore.instance.collection('users').doc(id);
}
```

core/data/types/document.dart:

```
import 'package:cloud_firestore/cloud_firestore.dart';

import 'object_map.dart';


enum Document {
    events,
    info,
    messages,
    students,
    subjects;

    DocumentReference<ObjectMap> ref(String groupId) {
        return FirebaseFirestore.instance.collection(name).doc(groupId);
    }
}
```

core/data/types/field.dart:

```
enum Field {
    author,
    chosenSubjects,
    content,
    date,
    groupId,
    hasTime,
    id,
    info,
    irrelevantEvents,
    isCommon,
    name,
    note,
    subject
}
```

core/data/types/object_map.dart:

```
typedef ObjectMap = Map<String, dynamic>;
```