Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА ANDROID – ЗАСТОСУНКУ ДЛЯ КОНТРОЛЮ ЗА ЗДОРОВ'ЯМ

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення" 121

Керівник курсової роботи с.в.
_Борозенний С.О.__
(прізвище та ініціали)
_____
(підпис)
"____" _____ 2024 р.

Виконала студентка
___Петрова О.В._____
(прізвище та ініціали)
"____" _____ 2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри
мультимедійних систем,
доцент, к.ф-м.н.
_____О. П. Жежерун
(підпис)
„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Петровій Олесі Владиславівні факультету інформатики 3-го курсу

ТЕМА Розробка Android-застосунку для контролю за здоров'ям

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Календарний план

Вступ

1 Аналіз предметної області

2 Архітектура застосунку, бібліотеки, технології

3 UI/UX

Висновок

Джерела

Дата видачі „___” _____ 2023 р.

Борозенний О.С. _____ (підпис)

Завдання отримала _____ (підпис)

Календарний план виконання роботи

Тема: Розробка Android-застосунку для контролю за здоров'ям

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Визначення теми кваліфікаційної роботи	Жовтень 2023	
2.	Отримання завдання на кваліфікаційну роботу	Жовтень 2023	
3.	Огляд літератури	Листопад - Грудень 2023	
4.	Розробка дизайну застосунку	Січень 2024	
5.	Розробка застосунку	Лютий – Травень 2024	
6.	Написання текстової частини	Травень 2024	

Петрова О. В. _____

Борозенний С. О. _____

“_____” _____

Зміст

Анотація

Метою курсової роботи є створення Android додатку під назвою UniFit, який містить в собі функції фітнес тренувань, трекеру прийому таблеток та здачі аналізів, та трекеру сну, відправки сповіщень для нагадувань та відображення статистики користувача.

Застосунок розроблявся в IDE Android Studio та написаний на мові програмування Kotlin, використовуючи XML Views.

Ключові слова: Android, Kotlin, здоров’я, View model, Firebase, Firestore Database, Storage, Room, Hilt, Fragment, Coroutines, Flow, LiveData.

Вступ

Здоров’я – це найважливіше, що має кожна людина. Підтримування здорового способу життя має бути пріоритетом і регулярністю. Оскільки ми живемо у вік цифрових технологвй, то для того щоб допомогти користувачу було створено багато мобільних застосунків. Фітнес-додатки дозволяють користувачам виконувати тренування вдома та встановлювати нагадування про них. Трекери прийому таблеток і аналізів є необхідними для людей, що регулярно приймають ліки або проходять курс лікування. Трекери сну дозволяють користувачам моніторити їхній режим, вимірюючи тривалість сну та показуючи статистику. Дуже важливо та зручно мати все в одному місці та в швидкому доступі, так як враховуючи сидячий спосіб життя і часто віддалену роботу або навчання можна забутися в прийомі таблеток, або виконанні вправ, що є недопустимим для здорового способу життя. Саме тому, за мету курсової роботи була поставлена розробка мобільного додатку для догляду за здоров’ям зі сповіщеннями.

Новизна складається в тому, що застосунок включає в собі трекер прийому таблеток та здачі аналізів, разом з фітнес та трекером режиму сну, зберігаючи та відображаючи всю статистику разом, так як мені зустрічалися додатки або тільки для трекінгу прийому таблеток або тільки для фітнесу. Мені хотілося б мати комплексний підхід до здоров’я і щоб для всіх моїх потреб існував один застосунок.

Основна частина роботи складається з 3 розділів:

- Аналіз предметної області та цільової аудиторії: проблеми з якими стикається користувач і вирішення їх застосунком.
- Архітектура застосунку, використані бібліотеки, технології.
- UI/UX додатку.

1. Аналіз предметної області

1.1 Проблеми, з якими стикається користувач

У світі цифрових технологій, коли працювати і вчитися можна з дому, доволі важко підтримувати здоровий спосіб життя : лягати вчасно спати, робити регулярні фізичні вправи, не забувати про профілактику захворювань. Багато людей проводять більшу частину дня сидячи, що може призвести до проблем зі здоров'ям, такими як проблеми зі спиною, ожиріння, серцево-судинні захворювання та інші. Через відсутність дисципліни можна легко забути про щорічні необхідні аналізи, прийом таблетки чи заклопотатися і пізно лягти спати.

Отже, розберемо з якими проблеми стикається користувач, якому буде цікавий даний застосунок:

- Обмежений час: у повсякденній рутині буває настільки багато задач, що піти в зал навіть не має часу, особливо якщо ти – працюючий студент, або за кимось дивишся, наприклад за дитиною.
- Фінансові обмеження: Вартість абонементу до фітнес-залу зазвичай висока, а якщо це преміальний зал, то вона може бути космічною. Якщо ти маєш бажання ще й наняти персонального тренера, або ходити на групові тренування, то треба заплатити додаткову ціну.
- Потреба у налагодженні режиму сну: Зручно мати під рукою додаток, де можна записати дані про свій сон і який потім покаже статистику.
- Дисципліна у прийомі таблеток: Деякі таблетки треба приймати курсом у один і той же час, тому для таких випадків це особливо актуально. Особливо регулярний прийом ліків важливий для людей з хронічними захворюваннями та захворюваннями в гострій стадії.
- Дисципліна у здачі аналізів: Обов’язково для корегування лікування необхідно здати відповідні аналізи, або варто просто не забувати про профілактичну перевірку здоров’я, особливо якщо є схильність до

певних захворювань або вони перебувають в хронічній стадії і треба попередити загострення.

- Потреба в нагадуванні щоденних дій: легко можна забути про прийом таблетки, тощо.

Отже, для користувача важливо мати під рукою програму, яка йому допоможе в дисципліні, контролі та покращенні здоров’я.

1.2 Вирішення проблем користувача мобільним застосунком

Додаток для фітнесу може надати стимул для більш активного способу життя, надаючи особливі тренувальні програми, нагадування про них. Додаток для слідкування за прийомом таблеток та здачею відповідних аналізів допоможе користувачу стати більш дисциплінованим та краще організувати свій медичний простір. Додаток для трекінгу сну допоможе налагодити режим та побачити динаміку покращення або погіщення сну. Все це даний додаток зберігає в одному місці, показуючи користувачу статистику даних, і працює як онлайн, так і офлайн.

1.3 Перспективи розвитку

Було придумано ще багато можливостей, які чекають своєї реалізації.

- Додавати друзів і бачити їхній прогрес;
- Зробити інтеграцію з смарт годинниками і брати з них дані;
- Додати розділ харчування;
- Додати можливість користувачам самостійно створювати тренування і викладати їх;
- Додати статті від користувачів на відповідну тему здоров’я;
- Додати AI для того, щоб бачити як користувач виконує вправи і давати поради по техніці.
- Додати персонального AI помічника, з яким можна проконсультуватися.

Вважаю, дану тему завжди актуальною і перспективною, оскільки не зустрічала аналогів, де було б зібрано настільки широкий функціонал в одному застосунку, особливо якщо підхопити тему ШІ.

2. Архітектура застосунку, бібліотеки, технології

2.1 Архітектура

Додаток використовує Single-Activity [1] архітектуру, що є рекомендованою і зручною. Вся логіка розташована в Fragment. Ось основні переваги такого підходу:

- Покращений запуск: Завдяки тому, що лише одна активність потребує запуску, Single-Activity архітектура покращує швидкість додатка, так як обробка Fragment набагато легша і швидша.
- Менеджмент даних: легше менеджити дані, так як можна взяти viewModel із Activity, NavGraph, Fragment, що сприяє гнучкому управлінню даними. Якщо вся програма потребує спільних даних, було визначено ViewModel на рівні активності — activityViewModels(..). Якщо екран вимагає лише даних для себе, визначалась ViewModel на рівні фрагмента —viewModels(..).
- Navigation graph: він надає кращу структуризацію навігацій між Fragment.

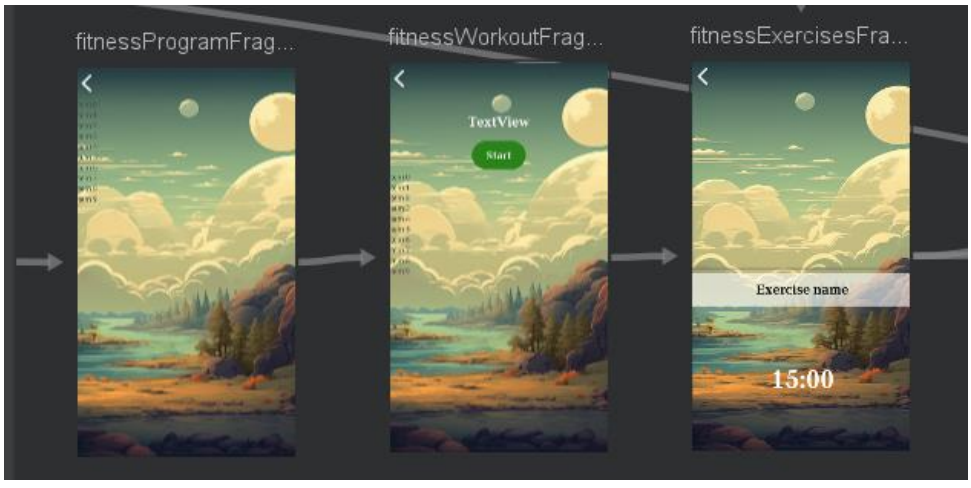

Рис. 2.1 Фрагмент navigation graph

Наприклад, на рис. 2.1 можна побачити як виглядає navigation graph окремих Fragments додатку. Видно, що є стрілка повернення, але не має стрілки назад в самому графі, це завдяки тому, що BackButtonView – це

кастомна кнопка , що при кліку визиває метод findNavController().navigateUp(). findNavController() – повертає NavController, асоційований з View.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/androi
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".presentation.MainActivity">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/nav_host_fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph" />

</FrameLayout>
```

Рис. 2.2 actimivty_main.xml

Додаток дізнається про navigation graph через xml атрибут app:navGraph (див. Рис. 2.2). Варто зазначити, що для оптимізації роботи використовується layout FrameLayout, що є легкишим порівняно, наприклад, з ConstraintsLayout, який нам пропонує IDE.

Даний додаток притримується Separation of concern принципу, тому архітектура поділена на три package (layers) – data, domain, presentation (див. Рис. 2.3) [2]. Було прийняте рішення поділити не на модулі, а на пакети.

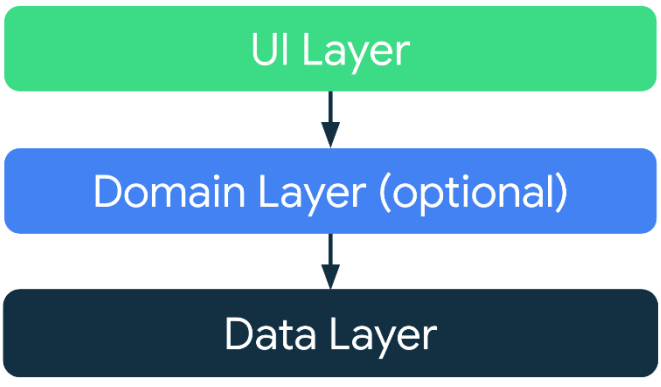

Рис 2.3 Архітектура застосунку.

Розберемо ці рівні детальніше і роль кожного з них.

Роль рівня інтерфейсу користувача (або рівня презентації) полягає у відображенні даних програми на екрані. Щоразу, коли дані змінюються через взаємодію користувача (наприклад, натискання кнопки) або зовнішній запит (наприклад, відповідь мережі), інтерфейс користувача має оновлюватися, щоб відобразити зміни.

Рівень інтерфейсу користувача складається з двох компонентів і шаблон за яким спроєктовано дану архітектуру називається MVVM:

1. Елементи інтерфейсу користувача, які відображають дані на екрані. Ці елементи можуть бути створені за допомогою Views або Jetpack Compose. В даному додатку прийнято було рішення використовувати Views, так як Jetpack Compose відносно нова технологія та ще не так широко використовується в розробці. Зв’язок між xml і view відбувається через binding.
2. ViewModel, які зберігають дані, надають їх інтерфейсу користувача та обробляють логіку. Для кожного Fragment з окремою логікою була створена своя ViewModel. Вона виступає мостом для зв’язку з domain шаром. Якщо треба зробити обрахунки, то створюється функція в ViewModel і ця функція викликається з Fragment. В Fragment створюється об’єкт ViewModel не як звичайна змінна, а через by viewModels() або by viewModels(Factory.provideFactory()) (якщо ми передаємо змінні в ViewModel). Це нам потрібно для того, щоб ViewModel знала про життєвий цикл компонента, в якому створюється, була створена і видалена після відповідного стану компонента (onDestroyView), так як в неї є свої методи життєвого цикла, а також не перестворювалася при зміні конфігурації (тобто зміні орієнтації, нічного режиму на денний і навпаки, мови) . Ключове слово «by» використовується для делегування в Kotlin, фактично в описаній ініціалізації ViewModel створюється інстанція

ViewModelProvider, яка відповідає за створення та управління ViewModel.

Так як Activity не знає коли оновлюватися, на це їй вказує LiveData або Flow. Activity може observe LiveData, інакше кажучи підписатися на її оновлення, тобто, наприклад, коли з API приходить відповідь, вона поміщається в LiveData, а вона в свою чергу сповіщає про підписаний на неї компонент про зміни і відбувається перемалювання екрану. Варто зазначити, що LiveData знає про життєвий цикл компонента, на відміну від Flow. Flow - це нова концепція у Kotlin, яка прийшла на заміну RxJava, це асинхронний потік даних, який не почне збиратися поки не викличеться collect. Flow можна перетворити на LiveData. Flow не інтегрована з життєвим циклом додатка, тому щоб не спричинити виток пам’яті треба управляти її циклом самостійно. В даному додатку було прийняте рішення використовувати як LiveData, так і Flow для більшої гнучкості, виклик потоку Flow відбувається з контексту view’s lifecycle owner. [6]

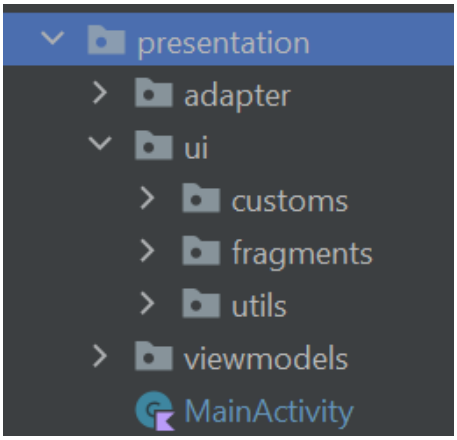

Рис. 2.4 Презентаційний шар в додатку UniFit

На рис. 2.4 представлена структура мого презентаційного архітектурного шару , про призначення viewmodels і fragments вже було описано вище, в uitils – знаходяться утиліти для ui, наприклад об’єкт який допомагає підтримувати edge to edge [3], тобто status bar в додатку є прозорим, але margin в контент (кнопки) автоматично не додаються, так як

Activity вважає це своєю робочою областю. Даний клас вирішує цю проблему, додаючи відступи до system bars і status bar. У пакеті customs розташовані реалізації кастомних view: наприклад, попереджувальних діалогів, в моєму випадку вони використовувалися, як наприклад вікно для редагування або вводу контенту. Пакет adapters містить в собі recycler view адаптери – невід'ємна частина для реалізації recycler view логіки. У мене є 2 типи адаптерів в проєкті – PagingDataAdapter і ListAdapter. PagingDataAdapter застосовується для відображення списків взятих з серверу, а ListAdapter для локальних невеликих списків. MainActivity також знаходиться в цій папці без пакету, бо вона у нас одна, тому підпапки їй не потрібні.

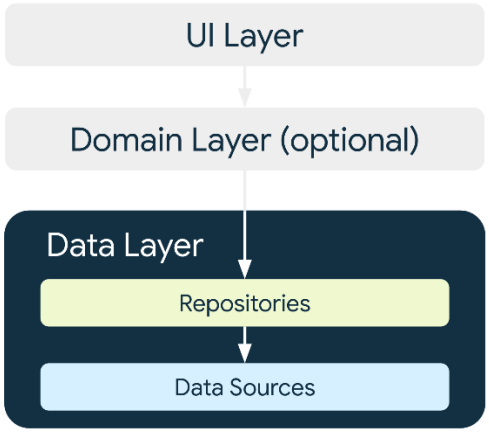

Рис.2.4 Дата шар

Перейдемо до дата шару (див. рис 2.4) [4], він складається з двох основних рівнів – репозиторії, та джерела даних та містить в собі бізнес логіку.

Репозиторії займаються: менеджментом перемиканнями між ресурсами (наприклад, якщо ми закешували дані, то ми беремо їх з кеша, а не з API), є мостом між доменом шаром і дата шаром, централізують зміни даних.

Для кожного типу отримання даних є свій дата ресурс, вони по суті тільки і займаються отримуванням, створенням, зміною, видалення даних. У моємо застосунку є 2 дата ресурси – локальний і серверний. На рис. 2.5 чітко можна побачити поділення на local і remote пакети. За remote базу даних була

взята Firestore Database і Firebase Storage. Для того щоб мати можливість робити мережеві запити треба вказати в манфіесті дозвіл на інтернет. Про них я більше розкажу в розділі про технології. Для local бази даних була використана бібліотека Room. . Про неї я більше розкажу в розділі про бібліотеки.

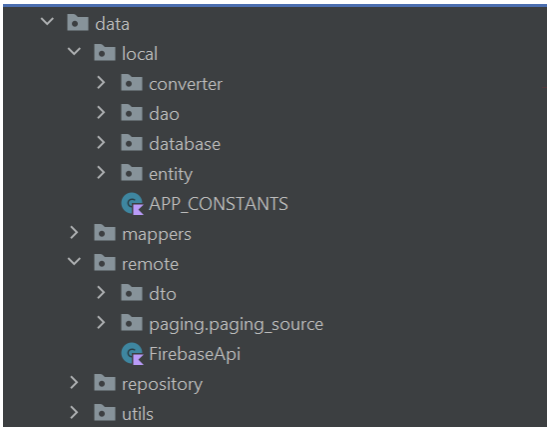

Рис. 2.5 Дата шар в додатку UniFit

В пакеті utils знаходиться клас, який допомагає працювати з мережевими запитами, тим що відповідь з сервера можна обгорнути в типом відповіді, яка від нього прийшла (див. рис. 2.6). Інші дата ресурси потребують детального огляду з описом внутрішної архітектури даних і принципу їхньої роботи, що буде зазначено в наступних розділах.

```
sealed class Resource<T>(
    val data: T? = null,
    val error: Throwable? = null
) {
    caleyass
    class Success<T>(data: T) : Resource<T>(data)


    caleyass
    class Error<T>(throwable: Throwable, data: T? = null) : Resource<T>(data, throwable)
}
```

Рис. 2.6 Допоміжний клас для роботи з мережевими запитами

Останній, третій архітектурний шар, який ми розглянемо – це доменний шар, в офіційні документації [5] написано, що він необов’язковий, але я вважаю, що він обов’язковий для кращого розділення обов’язків (separation of

concerns), масштабування, чистої архітектури загалом. Він інкапсулює складну бізнес логіку, слугує зв'язком між дата шаром та презентаційним шаром, включаючи в собі так звані use case, кожний з яких включає в себе конкретну дію з базою даних. Потім ці use case використовуються в view model (а точніше «inject» (вставляються), про це поговоримо в розділі бібліотеки, бо це частина DI).

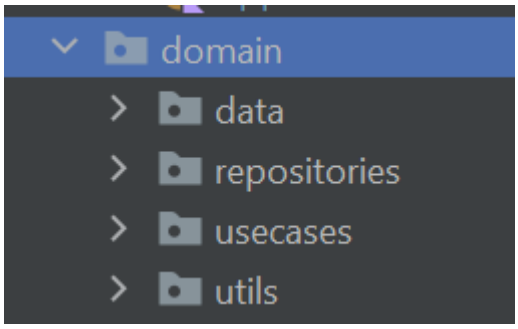

Рис. 2.7 Доменний шар в архітектурі додатку UniFit

На рис. 2.7 зображено доменний шар в архітектурі додатку UniFit, в пакеті data знаходяться data класи, які використовуються в презентаційному шарі, наприклад дата клас User, який містить в собі дані про залогованого користувача і зберігається в shared preferences. Пакет repositories містить в собі інтерфейси репозиторіїв, які імплементуються в дата шарі. Це зроблено для того , щоб при передачі в use case передавати не репозиторій з дата шару, а інтерфейс цього репозиторію з того ж шару – доменного. Також це дуже допоможе при тестуванні.

Пакет utils містить в собі Broadcast Reciever. Broadcast Reciever – це компонент, який дозволяє додатку реагувати на різноманітні системні події або повідомлення, які можуть бути отримані від самого себе, системи, інших додатків в системі, подібно до шаблону публікація-підписник. [7] Система оптимізує доставку повідомлень, щоб підтримувати оптимальний стан системи. Тому терміни доставки повідомлень не гарантуються. Для того, щоб використовувати Broadcast Reciever треба задекларувати його в маніфесті Також в цьому пакеті розташований об'єкт, який і тригерить Broadcast

Reciever – RemindersManager. Він використовує AlarmManager, так як WorkManager не гарантує час виконання, AlarmManager є найкращим варіантом для запуску певної події в певний час, а також є безпечнішим варіантом у режимі дрімання.

Схема за якою працює система сповіщень:

1. ReminderManager відповідає за запуск і зупинку нагадувань.

2. ReminderManager планує нагадування за допомогою AlarmManager.

3. AlarmManager запустить AlarmReceiver, коли настане запланований час.

4. AlarmReceiver надішле сповіщення та за бажанням перенесе нагадування.

Для того, щоб сповіщення працювали треба додати дозволи до маніфесту такі як:

- POST_NOTIFICATIONS дозволяє додатку створювати та відправляти сповіщення користувачеві.
- SCHEDULE_EXACT_ALARM дозволяє додатку розкладати точні спрацьовування будильників.
- USE_EXACT_ALARM вказує на те, що додаток може використовувати точні будильники.
- SET_ALARM дозволяє створювати будильники.

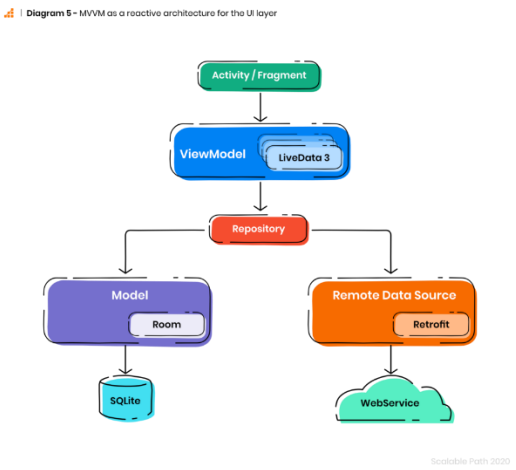

Рис.2.8 Архітектура застосунку UniFit

Варто зазначити, що код додатку відповідає всім п’яти принципам SOLID [21]:

- Принцип єдиної відповідальності: клас повинен виконувати лише одну конкретну функцію або мати лише одну область відповідальності. Наприклад, для роботи з локальною і для роботи з віддаленою базою даних використовуються різні класи.
- Принцип відкритості/закритості: програмні сутності повинні бути відкритими для розширення, але закритими для модифікації. Наприклад, кожний репозиторій в дата шарі має свій інтерфейс в доменному шарі.
- Принцип підстановки Лісков: об'єкти підтипу повинні бути замінимі за своїм базовим типом без впливу на коректність програми. Наприклад, ін’єкція репозиторіїв відбувається тільки в use case.
- Принцип інтерфейсу: кожен клієнт повинен мати доступ лише до тих методів, які необхідні для його роботи. Наприклад, view model містить тільки ті use case, які потрібні для конкретного Fragment.

- Інверсії залежностей: модулі високого рівня не повинні залежати від модулів нижчого рівня, а обидва типи модулів повинні залежати від абстракцій; абстракції не повинні залежати від деталей, деталі повинні залежати від абстракцій.

Також використовується Clean code – функції розбиті на малі частини, максимально зменшена кількість повторного коду, додані generic. Наприклад, PagingSource клас для пагінації викроистовує generic для однотипних даних різного призначення. [22]

Додаток використовує всі можливості Kotlin, а саме null-safety. Наприклад binding і view ініціалізуються не через lateinit var, а через Binding? = null і в методі onDestroyView він прирівнюється до null. Ініціалізація через lateinit var може призвести до memory leak, при повторному приєднанні фрагменту можуть в binding зберігатися посилання на старі view.

Отже, підсумуємо інформацію про схему архітектури додатку UniFit (див. рис.2.8). Вона складається з трьох архітектурних шарів : презентаційного, доменного та дата шару. Презентаційний шар складається з Fragment, xml-layout, вони пов’язані між собою через binding, а також view model, що включає в себе live data, flow, UI дізнається про зміни в даних за допомогою того, що «підписується» на view model. Доменний шар складається з інтерфейсів репозиторіїв та use case, які беруть з репозиторіїв методи, розподіляючи їх по одному, щоб один use case відповідав одній певній дії з дата ресурсами. Дата шар складається з реалізації репозиторіїв та різних типів дата ресурсів . В даному застосунку локального та серверного.

2.2 Бібліотеки

Бібліотеки, застосовані в додатку UniFit і які будуть роглянути в цьому підрозділі: Hilt, Room, Gson, Glide, Paging 3, MPAndroidChart.

Hilt - це бібліотека для ін’єкцій залежностей (DI) для Android, яка скорочує код тим, що дає можливість автоматично вставляти залежності в

певні фрагменти коду. [9] Це офіційний інструмент від команди розробників Android у Google і він базується на Dagger. у вашому проекті. Hilt пропонує стандартний спосіб використання DI у програмі, надаючи контейнери для кожного класу Android у проєкті та автоматично керуючи їхніми життєвими циклами. Контейнери - це об'єкти, які утримують і керують залежностями всередині додатка. Hilt надає кілька вбудованих контейнерів для керування різними типами залежностей : компонент додатка, активності, фрагменту, ViewModel, сервісу. Не можливо зробити ін’єкцію в клас, який не є компонентом. У даному додатку використані всі контейнери, крім сервісів. Для того щоб клас став контейнером треба : додати анотацію, залежно від типу контейнера, та використати анотації Hilt в цьому контейнері. Контейнер додатку є обов’язковим і він задається анотацією @HiltAndroidApp і зазначається в маніфесті, так як наслідує Application.

У архітектурі додатку об’єкт – модуль, який надає залежності знаходиться в окремому пакеті, не належачи до жодного архітектурного шару. На рис. 2.9 показана частина коду з модуля Hilt в додатку UniFit. @Module – анотація, яка позначає об'єкт як модуль; анотація @InstallIn(SingletonComponent::class) означає, що модуль буде доступний на всіх рівнях в додатку та буде існувати протягом усього життєвого циклу додатка; @Provides позначає метод, який надає залежність; @Singleton вказує, що повернутий об'єкт буде одним і тим же для всіх запитів у межах свого життєвого циклу. У даному випадку надається залежність моєї бази даних Room, в параметри передається контекст застосунку, що автоматично наповнюється завдяки ін’єкції через анотацію ApplicationContext.

```
@Module
@InstallIn(SingletonComponent::class)
object AppModule {

    caleyass *
    @Provides
    @Singleton
    fun provideFitnessDatabase(@ApplicationContext context : Context): MyDatabase {
        return MyDatabase.getDatabase(context)
    }
```

Рис. 2.9 Частина коду з модуля Hilt в додатку UniFit

Hilt дуже спрощує написання коду наприклад в use case, як показано на рис.2.10, можемо легко отримати об’єкт PillRepository, не думаючи звідки його взяти, а концентруючись на бізнес-логіці.

```
new *
class GetPillsUseCase @Inject constructor(private val pillRepository: PillRepository) {
    new *
    suspend fun execute() = pillRepository.getAllPills()

}
```

Рис.2.10 Use case з Hilt ін’єкцією

Room - це бібліотека архітектурних компонентів для Android, що надає абстракцію бази даних SQLite та спрощує взаємодію з базою даних у додатках. [10] На рис. 2.11 зображена архітектура бібліотеки Room.

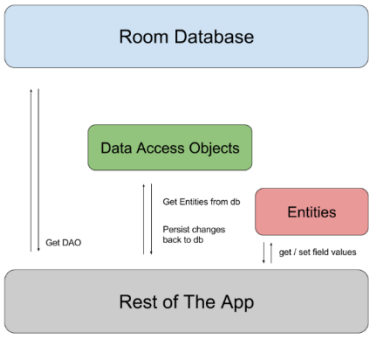

Рис. 2.11 Діаграма архітектури бібліотеки Room

Вона складається з трьох основних компонентів: Entity, DAO, Database. Нижче розгянуто ширше кожний.

Entity - клас, який представляє структуру таблиці в базі даних. Кожен об'єкт Entity відповідає рядку в таблиці бази даних. Їх в мене п’ять і всі вони включають в себе унікальні id типу Int: PillEntity (рис. 2.12), що відповідає за таблетки і включає в себе назву, дозу, дату закінчення, AnalysisEntity, що відповідає за аналізи і включає в себе назву, дату, доктора, AlarmEntity відповідає за сповіщення і включає в себе необов’язкове id об’єкта, тобто таблетки, до прийому якої воно може бути прив’язане, час, чи активоване

сповіщення, текст сповіщення та назву сповіщення, SleepEntity, що відповідає за сон і включає в себе час початку, час кінця, дату та якість сну, TrainingEntity, що відповідає за тренування і включає в себе дату та спалені калорії.

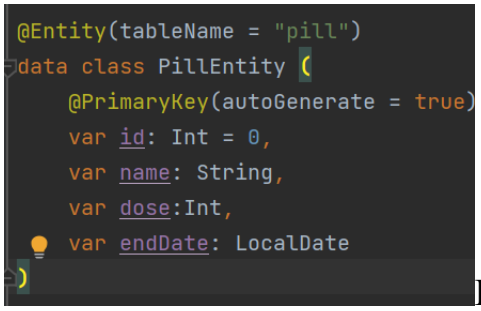

Рис. 2.12 Клас Entity

На рис. 2.12 зображено приклад класу Entity в моєму додатку, він позначає таблицю, яка містить в собі таблетки, стовпці в цій таблиці – це змінні в дата класі. Анотація PrimaryKey визначає ключ і в дужках зазначаємо, що вона автоматично генерує значення. Цікавий момент в тому, як створюються зв’язки між Entity, є декілька способів, але продемонструю той, що застосовую в проєкті.

Рис. 2.13 Зв’язок Entity один-до-багатьох

На рис. 2.13 представлено спосіб зв’язку двох Entity – в моєму випадку це один до багатьох. @Embedded – анотація, яка вказує, що об'єкт PillEntity повинен бути вбудований у цей об'єкт PillWithAlarms. Тобто, поля об'єкту PillEntity будуть збережені разом з полями об'єкту PillWithAlarms у тій же таблиці бази даних. @Relation – анотація, що вказує на колонки, які використовуються для з'єднання об'єктів.

DAO (Data Access Object): Це інтерфейс, який містить методи для виконання операцій з базою даних, таких як вставка, вибірка, оновлення та

видалення даних. Room автоматично генерує ці методи, варто лише вказати анотацію , назву, параметри, тип, що повертається. Анотації, що використовуються Query, Insert, Delete, Update. Якщо з останніми все зрозуміло, так як це базові CRUD операції, то Query анотація приймає як параметр SQL запит.

Database - це абстрактний клас, який відображає базу даних SQLite. На рис. 2.15 показано, як виглядає Database в додатку UniFit, він містить в собі статичну змінну, яка повертає базу данних, всі DAO, @TypeConverters містить в собі клас, який дає інструкції як зберігати LocalDate змінні. Для цього її треба перетворити в Long тип, визвавши toEpochDay() метод, і аналогічно для перетворення з Long в LocalDate використовуємо ofEpochDay(value).

```
@Database(entities = [
    AlarmEntity::class,
    AnalysisEntity::class,
    PillEntity::class,
    SleepEntity::class,
    TrainingEntity::class], version = 4, exportSchema = false)
@TypeConverters(LocalDateConverter::class, DateConverter::class)
abstract class MyDatabase : RoomDatabase() {
    abstract val alarmDao : AlarmDao
    abstract val analysisDao : AnalysisDao
    abstract val pillDao : PillDao
    abstract val sleepDao : SleepDao
    abstract val trainingDao : TrainingDao
    caleyass
    companion object{
        @Volatile
        private var INSTANCE: MyDatabase? = null

        caleyass
        fun getDatabase(context: Context): MyDatabase {
            return INSTANCE ?: synchronized( lock: this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
```

Рис. 2.14 Database в проєкті UniFit

Gson - це бібліотека Java, яку можна використовувати для перетворення об’єктів Java у їх представлення JSON. Його також можна використовувати для перетворення рядка JSON на еквівалентний об’єкт Java. [11] Ця бібліотека використовується в випадку, коли треба зберегти дані про залогованого користувача в Shared Preferences.

Glide – швидка та ефективна бібліотека для медіа менеджменту та завантаження гіфок, картинок, відео.[12]

Paging 3 – бібліотека, що є частиною Android Jetpack і допомагає завантажувати та відображати сторінки даних із більшого набору даних. Цей підхід дозволяє програмі ефективніше використовувати як пропускну здатність мережі, так і системні ресурси. [13]

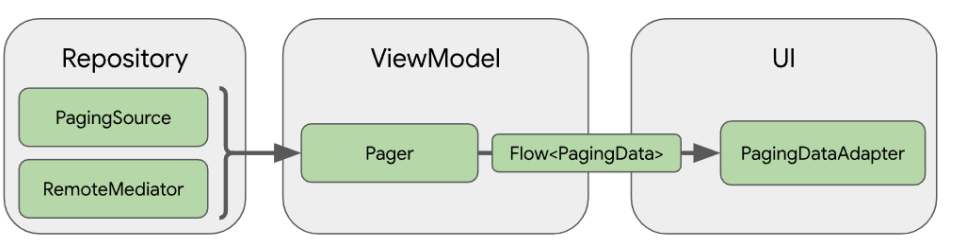

Рис. 2.15 Як Paging бібліотека має архітектурно розташовуватися в проєкті

Основним компонентом бібліотеки Paging в архітектурному дата шарі є PagingSource і RemoteMediator. PagingSource – обгортка над джерелом, з якого беруться дані, RemoteMediator – клас, який займається кешуванням завантажених даних в локальне сховище. У даному додатку RemoteMediator не використовується, бо Firestore Database автоматично завантажує дані у свій кеш, це допомагає оптимізувати відображення даних, так як доступитися до API більш часозатратно, ніж до локальної бази даних і дозволяє працювати додатку в офлайн режимі. Планую вдосконалювати свій додаток і додати кешування не автоматичне, а своє в Room для кращого управління даними. PagingSource у даному додатку при завантаженні однієї сторінки підвантажує і наступну.

Pager – клас, який займається організацією того, щоб правильно підгружати дані і перетворювати їх в Flow<PagingData>. Цей Flow вже і відображається в PagingDataAdapter.

Основним компонентом бібліотеки Paging в презентаційному архітектурному шарі є PagingDataAdapter, адаптер RecyclerView, який обробляє розбиті на сторінки дані.

MPAndroidChart – це бібліотека, яка будує графіки. У даному застосунку використано лише BarChart графік. [20]

Отже, для DI використано Hilt, для локального зберігання даних - Room, для серіалізації - Gson, для підвантаження медіа-файлів - Glide, для пагінації - Paging 3, для відображення графіків – MPAndroidChart.

2.3 Технології

Технологія, що застосована в цьому додатку і яка буде розглянута в цьому розділі – Firebase. [15] Вона є набором інструментів та сервісів, які застосовуються для розробки додатків або рішень. Firebase надає зручний та ефективний спосіб вирішувати відповідні задачі, такі як зберігання та синхронізація даних, аутентифікація користувачів, надсилання повідомлень тощо.

У даному додатку використовується аутентифікація двох видів: через Google і через пошту з паролем. За все це відповідає клас FirebaseAuth, що є вхідною точкою для Firebase Authentication SDK. Для зв’язку з Firebase на модульному рівні проєкту знаходиться конфігураційний файл google-services.json, який містить в собі ідентифікатор додатку, і ключі до доступу до API.

Для того, щоб додаток підтримував автентифікацію через Google sign in в google-services.json є секція oauth_client, яка містить в собі єдиного клієнта – мій застосунок (його ідентифікатори). [18] Для того, щоб з’явилась ця секція в конфігураційному файлі google-services.json треба згенерувати SHA fingerprint і додати в Firebase console. Для цього треба запустити команду gradle signingReport, вона аналізує проєкт та виводить звіт, який містить інформацію про всі використані ключі та сертифікати, такі як SHA-1 хеші

тощо. SHA-1 є унікальним ключем для проєкту і надасть доступ до Google sign in [19]. Після проходження вступного тесту, де збираються базові дані про користувача, вони записуються на сервер в Firestore Database.

Firestore Database - це хмарна NoSQL база даних, яка пропонує розробникам зручний та масштабований спосіб зберігання та синхронізації даних. [16] Ключові можливості:

- Гнучкість : підтримує гнучкі ієрархічні структури даних. Зберігайте свої дані в документах, організованих у колекції. Документи можуть містити складні вкладені об’єкти.
- Параметризовані запити: Cloud Firestore SDK містить можливості параметризованого отримання даних за полями в документі, наприклад за певними параметрами, відсортовані дані, тощо.
- Офлайн підтримка: в коді можна вказати, що отримання даних відбувається з кеша, але Cloud Firestore SDK автоматично перемикається на кеш за відсутності мережі.

На рис. 2.16 представлена модель даних в Firestore Database для проєкту UniFit. Колекція складається з розділів фітнесу, кожний з яких містить документи, що відповідають конкретним тренуванням зі своєю програмою. Програма тренування містить в собі назву, за якою можна відсортувати список, і основні елементи тренування – головну частину, розігрів і після-тренувальні вправи для розтяжки і відпочинку. Вони зберігаються у вигляді списків стрічок. Для того, щоб отримати інформацію про конкретну вправу з гіфкою, де демонструється як вона виконується. В коді при отриманні стрічок з назвами вправ робиться запит по шляху вказаному на рис. 2.17. Починаємо з документу з назвою частини тренування, який знаходиться на тому ж рівні, що і документ з категоріями тренувань, що показані на рис. 2.16. Далі в колекції вправ містяться назви всіх прав, які використовуються в усіх категоріях. На рис. 2.18 зображено як виглядають дані про окрему вправу –

назва і посилання на гіфку, яке береться з Storage. Розглянемо детальніше цю технологію.

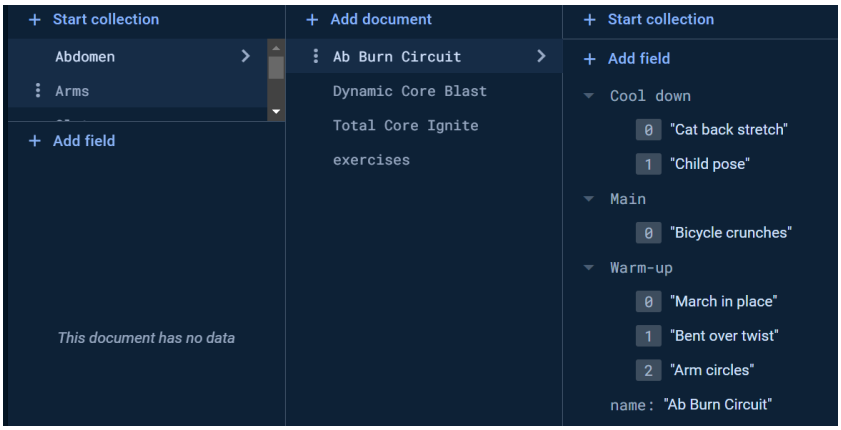

Рис. 2.16 Модель даних для категорій тренувань в Firestore Database для проєкту UniFit

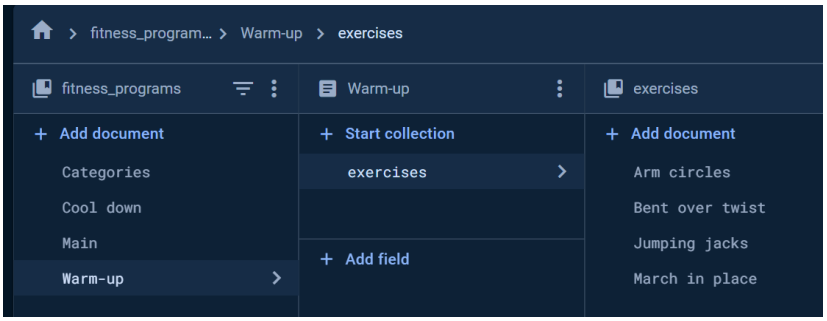

Рис. 2.17 Модель даних для вправ за частинами тренування в Firestore Database для проєкту UniFit

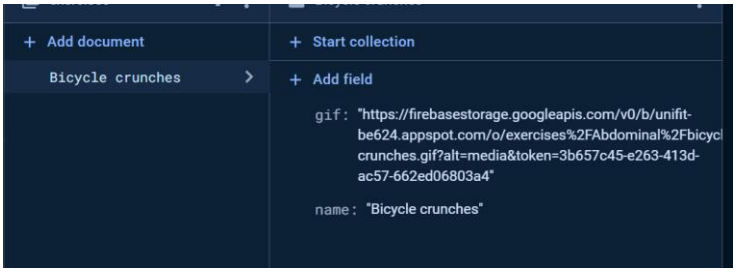

Рис. 2.18 Модель даних для окремої вправи Firestore Database для проєкту UniFit

Cloud Storage для Firebase – це потужне, просте хмарне сховище, яке побудоване на швидкій та безпечній Google Cloud інфраструктурі. [17] Його ключові можливості:

- Надійні операції: Firebase SDKs для Cloud Storage може виконувати завантаження та вивантаження не зважаючи на якість з’єднання.
- Висока безпека: Firebase SDKs для Cloud Storage інтегроване Firebase Authentication, тому можна використовувати декларативну модель безпеки для надання доступу базуючись на ім’я файлу, розміру тощо.
- Здатність до швидкого масштабування.

На рис. 2.19 показані папки, в яких організований візуальний контент. що демонструють спосіб виконання вправ. У fitness_program знаходяться jpg картинки, що відповідають категоріям тренувань і названі відповідно. У exercises знаходяться папки, що розбиті за категоріями, а також Warm-up і Cool down зазначені як окремі категорії, у яких розташовані гіфки для вправ, посилання на які і містить документ з вправою в Cloud Firestore.

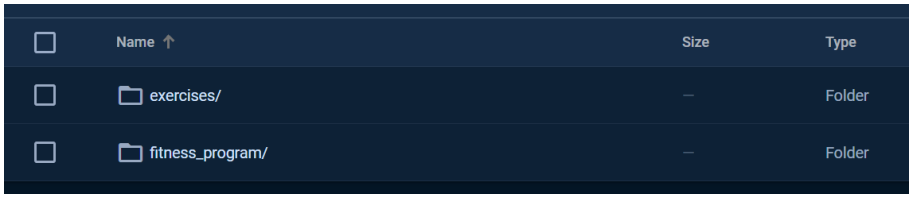

Рис. 2.19 Модель даних для візуального контенту Cloud Storage для проєкту UniFit

Отже, дані про користувачів, програми, тренування зберігаються структуровано в документах і колекціях в Cloud Firestore, автентифікація відбувається через Firebase Authentication, а медіа ресурси зберігаються по папках в Cloud Storage. Всі ці функції надає Firebase.

3. UI/UX

3.1 Ідея

UniFit – назва додатку, що є абревіатурою від University of Fitness. На рис. 3.1 можна побачити іконку даного застосунку. Його задумка – це гантель, яка складається з різнокольорових елементів, що демонструє те, з яких малих факторів складається наше здоров'я.

Рис. 3.1 Іконка UniFit

Кольори, які були обрані для застунку: зелений, та різні його відтінки. Текст або чорний або білий. Задумка була така, що кожний компонент здоров'я – це свій окремий всесвіт, тому фон для кожного – це своя природня місцевість (див. рис. 3.2), а їх в мене 4. Картинки є унікальними, бо були згенеровані за авторськими запитами через Midjourney.[22]

Рис. 3.2 Фони для додатку UniFit

3.2 Інтерфейс та можливості застосунку

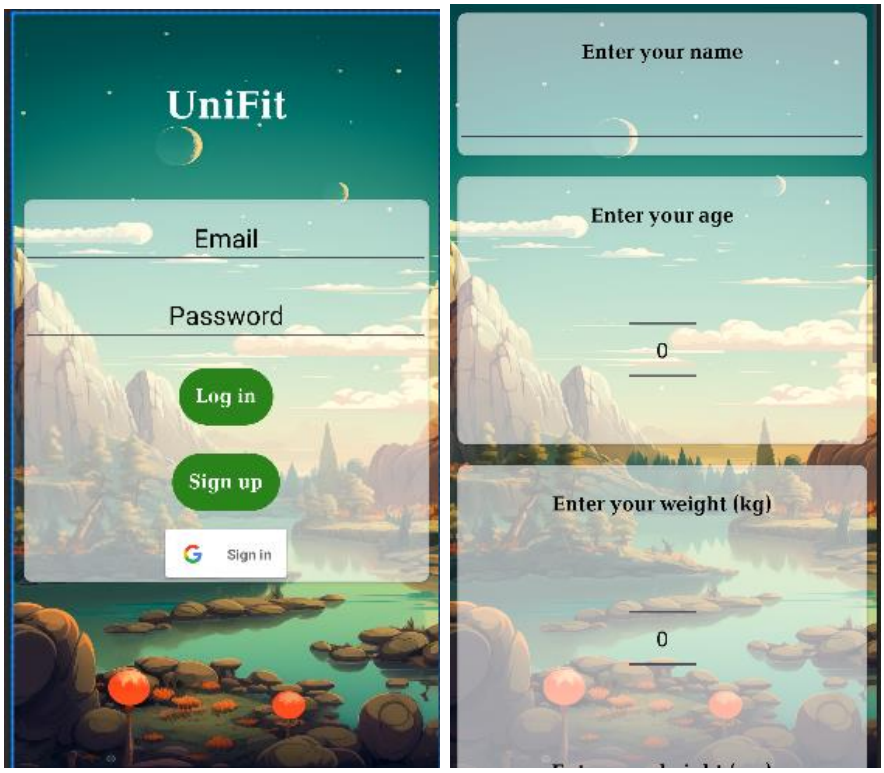

Рис. 3.4

На рис. 3.4 зображено початковий екран, де користувач може увійти або зареєструватися за допомогою пошти і пароля або за допомогою гугл акаунту. Після цього його треба пройти тест з початковими відомостями, який включає в себе ім’я, вік, стать, вагу і зріст, бажану вагу, рівень фізичної підготовки. Всі ці дані потім відображаються в профілі. Emai береться з поля де користувач реєструвався і його змінити буде не можна. При помилці викликається edit text error.

Далі користувач потрапляє на головний екран. (рис. 3.5), де бачить кнопки переходу по всім категоріям, статистику по датам сну, тренувань, кількість таблеток, які треба сьогодні випити.

Дані про користувача можна подивитися в його профілі і відредагувати. До налаштувань можна перейти через головне меню за кнопкою в правому верхньому куту (див. рис. 3.5). В налаштуваннях можна ввімкнути або вимкнути музику, відредагувати сповіщення, яке сповіщає про фітнес

тренування, відредагувати сповіщення, яке сповіщає про час йти до сну. Біля світчу сповіщень показано час, в який воно з’являється. За кнопкою Profile відкривається профіль користувача з усією інформацією введеною раніше та можливістю розлогінитися (див рис. 3.6). Також при зміні ваги кожного разу обраховується індекс маси тіла.

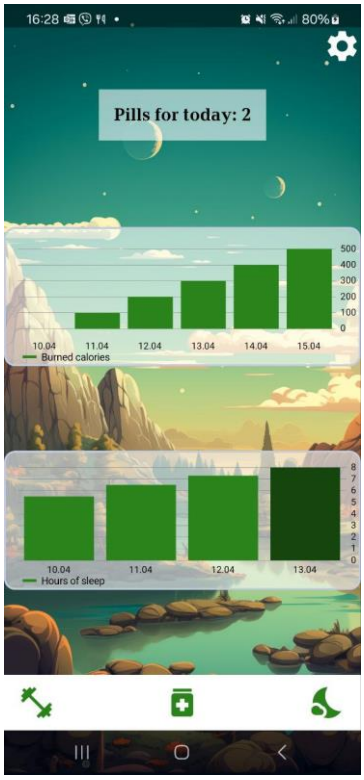

Рис. 3.5 Головний екран UniFit

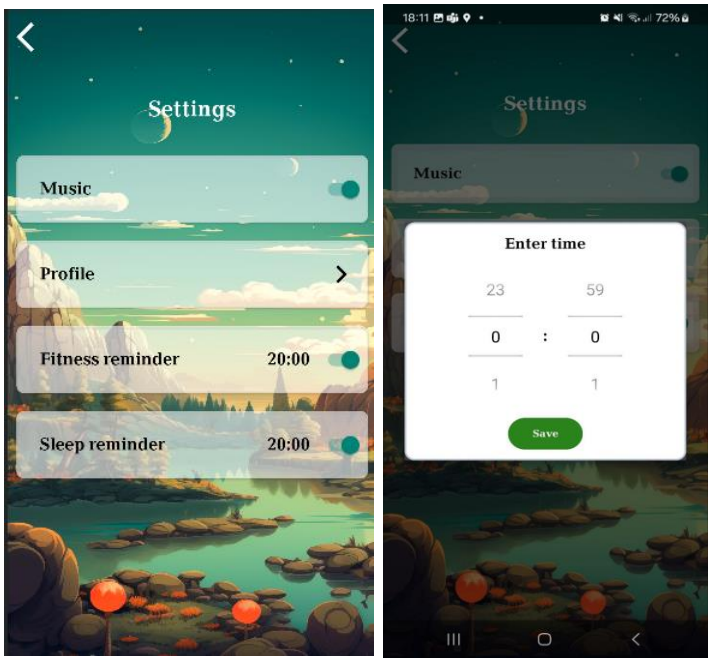

Рис. 3.5 Екран налаштувань

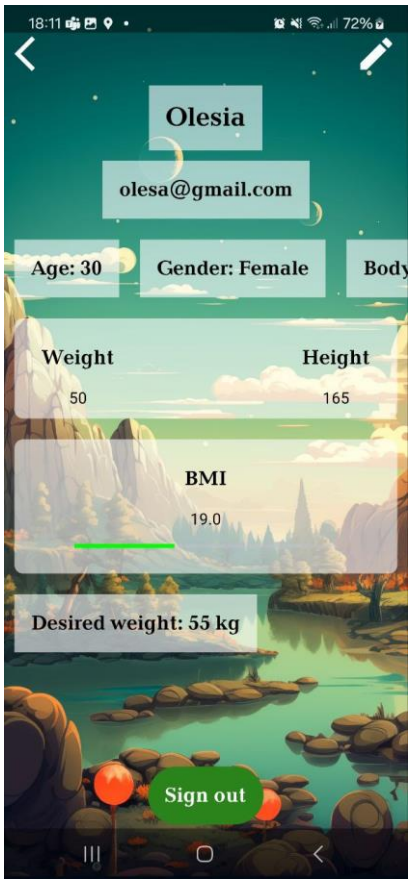

Рис. 3.6 Екран профілю користувача

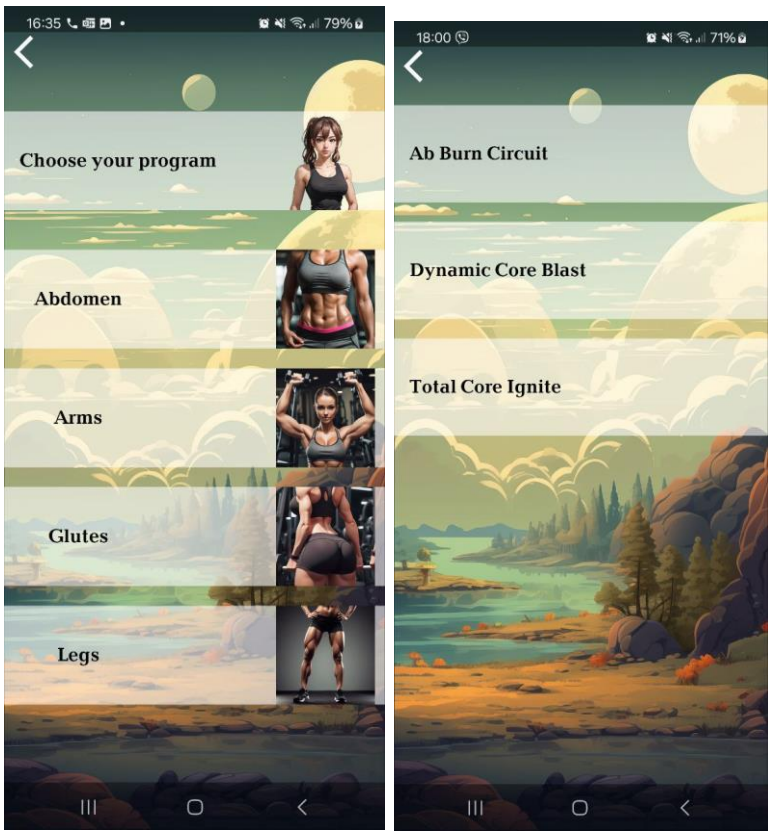

Рис. 3.7 Фітнес розділ UniFit

На рис. 3.7 зображено, як виглядає сторінка з фітнес категоріями і програмами тренувань.

Програми тренувань містять в собі перелік вправ та таймер. Час виконання вправ залежить від рівня фізичної підготовки користувача, який вказаний в його профілі (початківець, середній або професіонал). Після кожної вправи йде відпочинок, час якого можна збільшити або пропустити. Під час відпочинку завантажується і показується наступна вправа. Вправи розбиті за категоріями – розігрів, основна частина, розтяжка. В кінці тренування показана його статистика – тривалість, спалені кілокалорії та кількість виконаних вправ. Також користувач може тут відразу змінити свою вагу і зріст і буде обрахований автоматично новий індекс маси тіла.

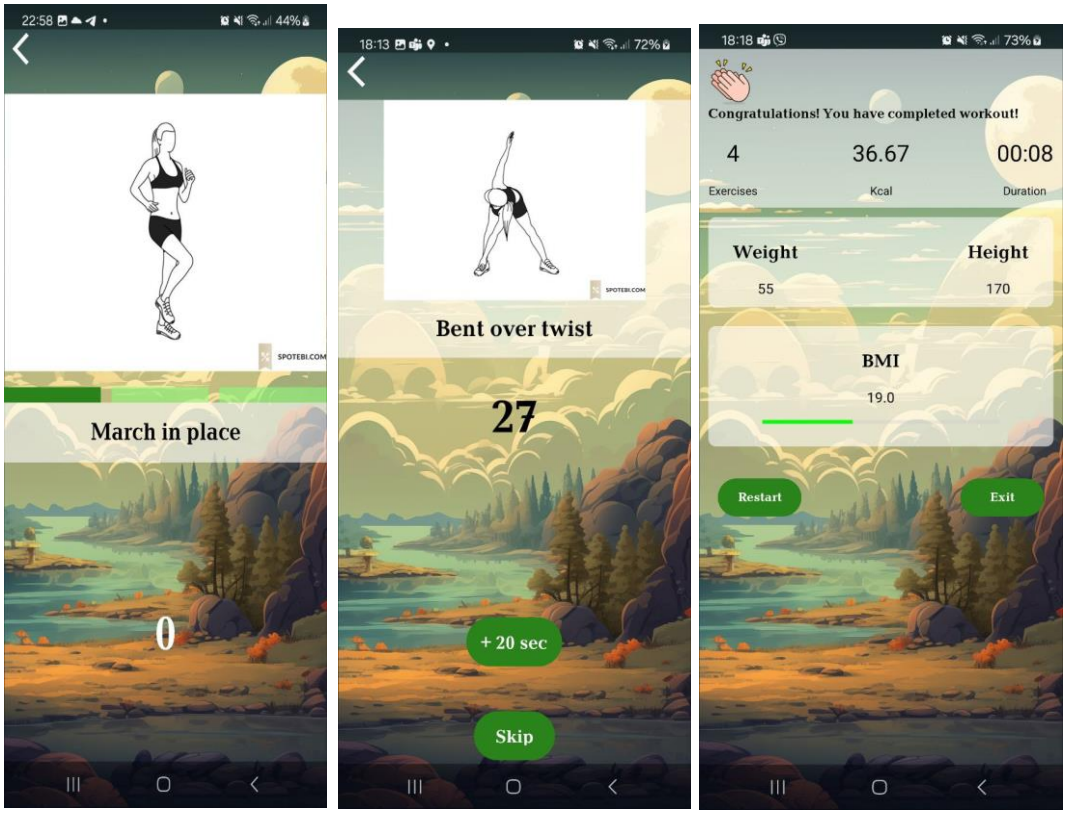

Рис. 3.8 Тренування UniFit

В категорії таблеток на першій сторінці відображаються таблетки і аналізи (див. рис. 3.9).

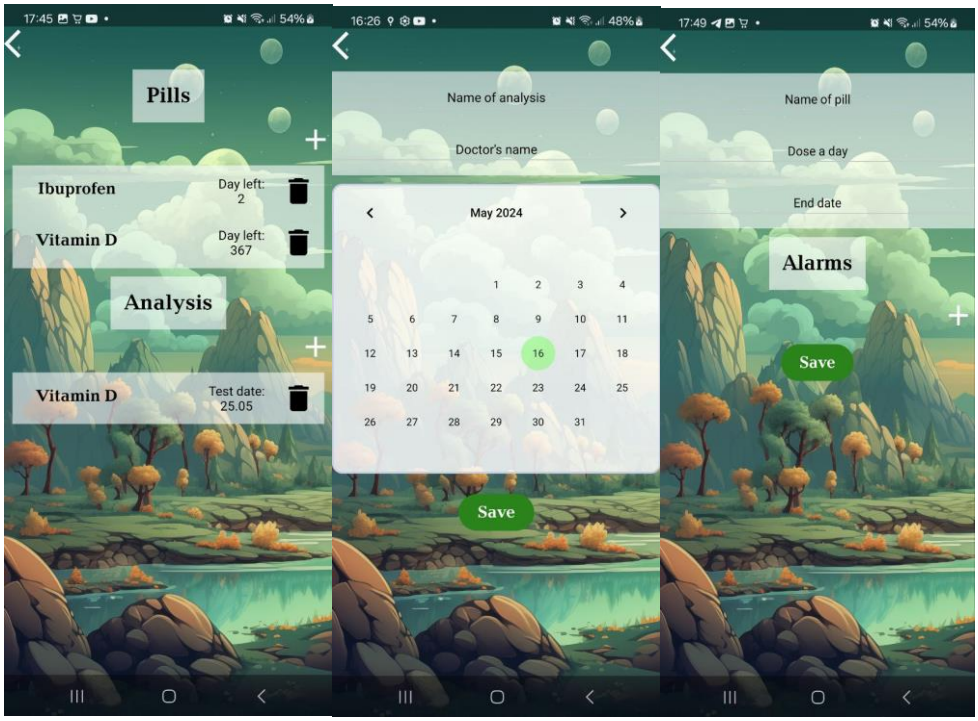

Рис. 3.9 Таблетки і аналізи. Головний екран і екрани, де вони додаються та редагуються

При переході на кнопку додати в правому верхньому кутку кожної категорії відкривається сторінка, де можна додати або таблетки, або аналізи.

Для кожної таблетки можна вказати довільну кількість сповіщень, але краще вказувати стільки, скільки прийомів її має бути за день. Сповіщення будуть приходити кожний день у вказаний час, поки в таблетки не закінчиться термін приймання, тоді вона автоматично видалиться зі списку.

При додаванні таблетки треба вказати її назву, щоденну дозу (скільки разів в день приймати), та кінцеву дату її приймання. При додаванні аналізу треба вказати його назву, дату та не обов’язково доктора, який його назначив.

У списку завжди можна перейти на таблетку або аналіз і відредагувати її. Біля таблетки в списку пишеться скільки днів залишилось до кінця курсу, біля сповіщення його час, біля аналізу його дату.

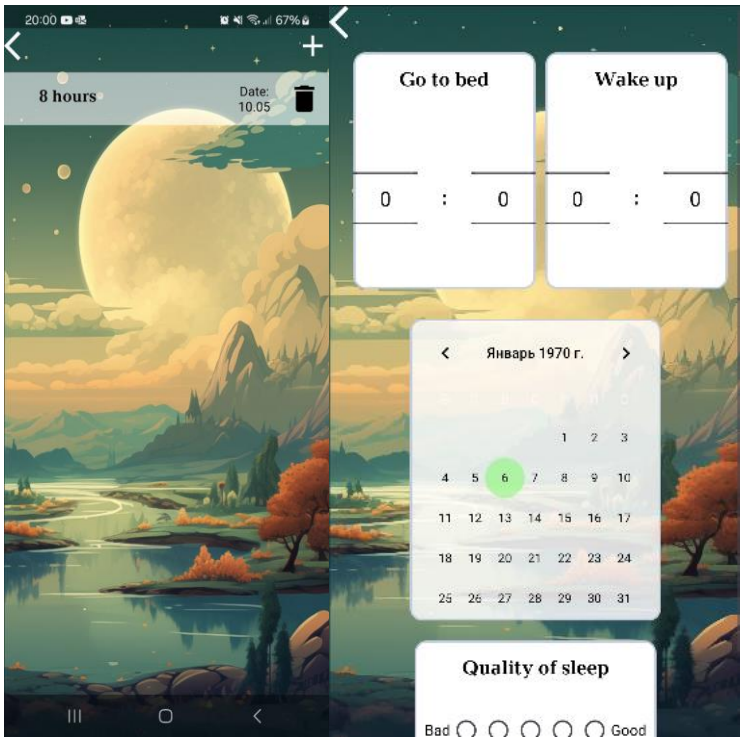

Рис. 3.10 Сон. Головний екран і екран створення

В категорії сну (див. рис. 3.10) знаходиться список записаних даних про сон. У верхньому правому кутку знаходиться кнопка додати. При її натиснені відбувається перехід на сторінку з полями, що необхідні для заповнення для додавання сну. А саме час, в який користувач ліг спати і час, в який він встав, дата, коли він прокинувся, якість сну від 0 до 5. Останні 5 даних беруться до уваги в статистиці сну.

Отже, в даному розділі розглянуто інтерфейс додатку UniFit. Він складається з трьох категорій – сну, таблеток з аналізами, фітнесу і головного екрану, де показано статистику показників користувача. Також є можливість залогінитися і змінити профіль. Є налаштування сповіщень і музики. Навігаційний граф додатку виглядає наступним чином (див. рис. 3.11).

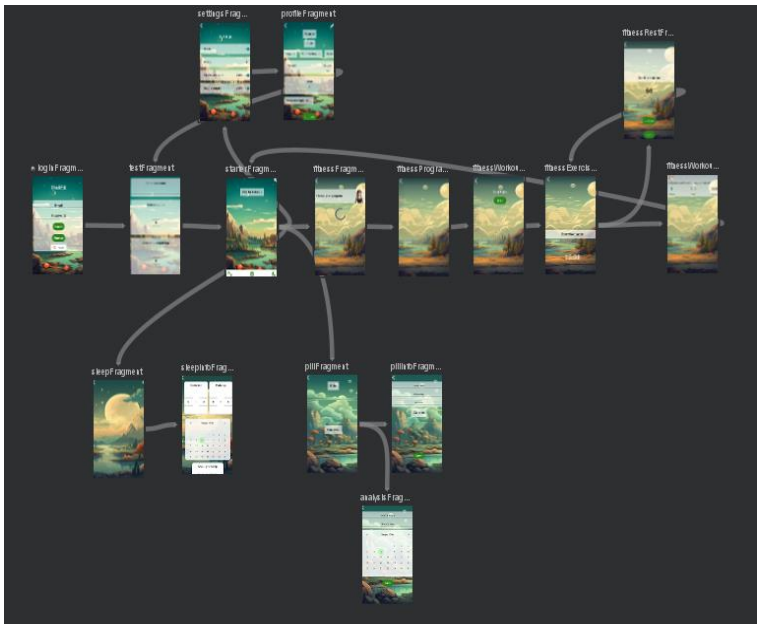

Рис. 3.11 Навігаційний граф всіх екранів UniFit

Висновок

Здоровий спосіб життя є дуже важливою частиною життя людини, але його важко підтримувати в сучасному світі, бо дуже поширеними стають дистанційна робота та навчання. Ходити в спортивні зали, пам’ятати про всі аналізи, таблетки, сон далеко не у всіх є можливість і бажання. На допомогу приходять такі мобільні додатки як UniFit. Додаток закриває багато потреб користувача, таких як відстеження фітнес-тренувань, моніторинг режиму сну, трекінг прийому ліків та аналізів. Зосередження всіх цих функцій у одному додатку робить його унікальним та корисним.

У ході розробки курсової роботи було проведено аналіз предметної області та цільової аудиторії. Також була описана архітектура застосунку – додаток використовує чисту архітектуру і 3 архітектурні шари, бібліотеки та технології – додаток використовує останні версії бібліотек і актуальні технології, які допомагають забезпечити ефективну та надійну роботу додатку. У розділі UI/UX було надано інформацію щодо інтерфейсу користувача та можливостей застосунку.

Новизна полягає в тому, що додаток включає в себе функції і фітнес додатку, і трекери прийому таблеток, і здачі аналізів, і трекер режиму сну, показуючи користувачу статистику та зберігаючи дані про його профіль.

Отже, розробка мобільного додатку для догляду за здоров'ям є актуальною та важливою задачею в сучасному суспільстві. Додаток UniFit може значно полегшити життя людей, допомагаючи їм вести більш здоровий спосіб життя.

Джерела

1. Single-Activity pattern – [Електронний ресурс]. – Доступно з: https://medium.com/nerd-for-tech/single-activity-2659f6ac09e8
2. Guide to app architecture– [Електронний ресурс]. – Доступно з: https://developer.android.com/topic/architecture
3. Display content edge-to-edge in your app – [Електронний ресурс]. – Доступно з: https://developer.android.com/develop/ui/views/layout/edge-to-edge
4. Data layer app – [Електронний ресурс]. – Доступно з: https://developer.android.com/topic/architecture/data-layer
5. Domain layer app – [Електронний ресурс]. – Доступно з: https://developer.android.com/topic/architecture/domain-layer
6. Flow Vs LiveData in Android Architecture Components – [Електронний ресурс]. – Доступно з: https://medium.com/@khunsoemoeaung/flow-vs-livedata-in-android-architecture-components-51745200e42c
7. Broadcasts overview – [Електронний ресурс]. – Доступно з: https://developer.android.com/develop/background-work/background-tasks/broadcasts
8. Android — Repeat notification daily on specific time – [Електронний ресурс]. – Доступно з: https://blog.protein.tech/android-repeat-notification-daily-on-specific-time-c2b0f7788f93
9. Dependency injection with Hilt – [Електронний ресурс]. – Доступно з: https://developer.android.com/training/dependency-injection/hilt-android
10. Save data in a local database using Room – [Електронний ресурс]. – Доступно з: https://developer.android.com/training/data-storage/room
11. Gson – [Електронний ресурс]. – Доступно з: https://github.com/google/gson
12. Glide – [Електронний ресурс]. – Доступно з: https://github.com/bumptech/glide

13. Paging library overview – [Електронний ресурс]. – Доступно з: https://developer.android.com/topic/libraries/architecture/paging/v3-overview
14. Authenticate with Firebase using Password-Based Accounts on Android – [Електронний ресурс]. – Доступно з: https://firebase.google.com/docs/auth/android/password-auth?hl=en&authuser=0
15. Firebase – [Електронний ресурс]. – Доступно з: https://firebase.google.com/docs/projects/learn-more
16. Cloud Firestore – [Електронний ресурс]. – Доступно з: https://firebase.google.com/docs/firestore
17. Cloud Storage for Firebase – [Електронний ресурс]. – Доступно з: https://firebase.google.com/docs/storage
18. Authenticate with Google on Android. – [Електронний ресурс]. – Доступно з: https://firebase.google.com/docs/auth/android/google-signin?hl=en&authuser=0
19. SHA 1 & SHA 256 — Why? How to generate? – [Електронний ресурс]. – Доступно з: https://medium.com/@harshanacz/sha-1-sha-256-why-how-to-generate-b78233db2555
20. MPAndroidChart – [Електронний ресурс]. – Доступно з: https://github.com/PhilJay/MPAndroidChart
21. SOLID – [Електронний ресурс]. – Доступно з: https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html
22. Robert C. Martin Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2009. 464 с.
23. Midjourney – [Електронний ресурс]. – Доступно з: https://www.midjourney.com/home