Київський національний торговельно-економічний університет

Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка мобільного додатку фітнес-студії «Body Sculptor» на платформі Android»

Студента 4 курсу, 7групи,
спеціалізація 121 «Інженерія програмна інженерія» _____ Басюк Євгеній Володимирович

підпис студента

Науковий керівник
кандидат технічних наук,
доцент _____ Харченко Олександр Анатолійович

підпис керівника

Гарант освітньої програми,
кандидат технічних наук,
доцент _____ Цензура Микола Олександрович

підпис гаранта

КИЇВ – 2020

Київський національний торговельно-економічний університет

Факультет *інформаційних технологій*

Кафедра *інженерії програмного забезпечення та кібербезпеки*

Освітній ступінь *бакалавр*

Спеціальність *121 «Інженерія програмного забезпечення»*

Спеціалізація / освітня програма *121 «Інженерія програмного забезпечення*

Затверджую

Зав. кафедри інженерії програмного забезпечення та кібербезпеки

Криворучко О.В.

"7" листопада 2019 р.

Завдання

на випускну кваліфікаційну роботу студентові

Басюку Євгенію Володимировичу

(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної роботи Розробка мобільного додатку фітнес-студії «Body Sculptor» на платформі Android

Затверджена наказом ректора від "02" грудня *2019р.* № *4112*

2. Строк здачі студентом закінченої роботи _____

_____

3. Цільова установка та вихідні дані до роботи

Мета роботи: створення мобільного застосування що дасть можливість контролювати прогрес підопічних, вести облік клієнтів та контролювати стан абонементів. Проаналізувати створену програму для фітнес-студії «Body Sculptor» формування певних висновків.

Об'єкт дослідження: зручний облік тренувань для працівників фітнес-центру.

Предмет дослідження: мобільний додаток фітнес-студії «Body Sculptor» на ОС Android.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ІДЕЯ ТА РОЗГЛЯД ПІДХОДІВ ДО РЕАЛІЗАЦІЇ. ОПИС ОСНОВНИХ ЗАДАЧ ПРОГРАМНОГО ПРОДУКТУ

1.1. Створення ідеї та опис плану реалізації.

1.2. Вибір мови програмування.

1.3. Вимоги до програмного продукту та список основних задач.

1.3.1. Вимоги до бібліотек для роботи з базою даних.

1.4. Вимоги до створення візуального інтерфейсу.

1.5. Аналіз існуючих програм для ведення та обліку клієнтів у фінтес-студіях.

1.6. Висновок до розділу 1.

РОЗДІЛ 2. ОПИС ОСНОВНИХ КЛАСІВ

2.1. Опис класу ClientAdapter.

2.2. Опис класу Client.

2.3. Опис ClientsFragment.

2.4. Ознайомлення з Android Studio.

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*		
2.	*Вступ та перелік літературних джерел*		
3.	*Розділ 1. «Ідея та розгляд підходів до реалізації. опис основних задач програмного продукту»*		
4.	*Розділ 2. «Опис основних класів»*		
5.	*Розділ 3. «Розробка мобільної програми фітнес-студії «Body Sculptor»»*		
6.	*Висновки*		
7.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*		
8.	*Підготовка автореферату та презентації доповіді*		
9.	*Попередній захист випускної кваліфікаційної роботи*		
10.	*Зовнішнє рецензування випускної кваліфікаційної роботи*		
11.	*Здача прошитої випускної кваліфікаційної роботи на кафедру*		
12.	*Публічний захист випускної кваліфікаційної роботи*		

7. Дата видачі завдання «___»_____20__ р.

8. Науковий керівник випускної кваліфікаційної роботи Харченко О. А.

9. Гарант освітньої програми Цензура М. О.

10. Завдання прийняв до виконання студент Басюк. Є. В.

11. Відгук керівника випускної кваліфікаційної роботи

Науковий керівник випускної кваліфікаційної роботи _____

(підпис, дата)

Відмітка про попередній захист *Харченко О.А.* _____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну робота

Випускна кваліфікаційна робота студента *Басюк Є.В.* може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____ Цензура М. О.

Завідувач кафедри _____ Криворучко О. В.

«____»_____20___р.

АНОТАЦІЯ

Дипломна робота викладена на 41 сторінках, вона містить 3 розділи, 40 рисунків, та 16 джерел в переліку посилань.

Об'єктом дослідження є зручний облік тренувань для працівників фітнес-центру.

Предмет дослідження - мобільний додаток фітнес-студії «Body Sculptor».

Метою дослідження є створення мобільного застосування що дасть можливість контролювати прогрес підопічних, вести облік клієнтів та контролювати стан абонементів. Проаналізувати створену програму для фітнес-студії «Body Sculptor».

У першому розділі описано технічне завдання у вимогах для створення програмного продукту. Проаналізовано існуючі мобільні додатків фітнес-центрів.

У другому розділі описано два класи в UML діаграмах та розглянута архітектура проекту.

Розділ 3 описано створення програмного додатку з використанням Android studio. Описано створення основних класів, activity та fragment. Поетапно описано інструкцію використання програми фітнес-студії.

Виконано роботу по розробці мобільний додаток для фітнес-студії «Body Sculptor» на операційну систему Android використовуючи мову програмування Kotlin.

ANNOTATION

Thesis is set out on 41 pages, it contains 3 sections, 40 images and 16 sources in the list of references.

The object of study is the convenient accounting of trainings for employees of the fitness center.

The subject of the research is a mobile application of the fitness studio "Body Sculptor".

The purpose of the study is the creation of a mobile application that will allow to monitor the progress of wards, keep records of customers and monitor the status of season tickets. Analyze the created program for the fitness studio "Body Sculptor".

The first section describes the terms of reference in the requirements for creating a software product. Existing mobile applications of fitness centers are analyzed.

The second section describes three classes in UML diagrams and discusses the project architecture.

Chapter 3 describes how to create a software application using Android studio. Describes the creation of basic classes, activity and fragment. The step-by-step instructions for using the fitness studio program are described step by step.

Work has been done to develop a mobile application for the fitness studio "Body Sculptor" for the Android operating system using the Kotlin programming language.

ЗМІСТ

ВСТУП

Мобільні пристрої, невід'ємна частина життя сучасного людини. Це комунікації між людьми, прослуховування улюбленої музики, перегляд аудіовізуальної інформації, блокнот, і ще безліч функцій. Мобільний смартфон це копія комп'ютера яку постійно можна мати при собі. В наш час, найпоширеніша оперативна система ОС Android. Android підтримує велику кількість пристроїв, від різних виробників.

Android містить велику кількість безкоштовних бібліотек для роботи зі сторонніми ресурсами наприклад Google Map API, а в той час для Windows Phone Mobile бібліотека не доступна. ОС Android це не тільки смартфони, але дана ОС використовується в смарт телевізорах, смарт годинниках, в системах розумний дім, планшетних комп’ютерах, т.д. То програма написана наприклад для смартфону технічно може підходити і на інші пристрої зазначених вище.

В даній дипломній роботі буде описано створення програми для фітнес-студії «Body Sculptor» на платформу ОС Android, для користувачів, що працюють в фітнес індрустрії.

Android Studio - офіційне середовище інтегрованого розвитку (IDE) для розробки додатків для Android, засноване на IntelliJ IDEA. Крім потужного редактора коду та інструментів для розробників IntelliJ, Android Studio пропонує ще безліч функцій, що підвищують продуктивність при створенні програм для Android. Окрім цього, це середовище має відкритий код, та є безкоштовним.

Актуальність даної теми має велике значення для тренерів, оскільки такий додаток зекономить час, щоб зосередитися на результатах клієнтів, організувавши всі дані в одному місці. Це дозволяє отримувати швидкий доступ до даних що їх цікавлять.

Метою дослідження випускної кваліфікаційної роботи є створення мобільного застосування що дасть можливість контролювати прогрес підопічних, вести облік клієнтів та контролювати стан абонементів. Проаналізувати створену програму для фітнес-студії «Body Sculptor».

Предмет дослідження – мобільний додаток фітнес-студії «Body Sculptor»

Завдання дослідження:

1. Проаналізувати існуючи програми.
2. Розробка архітектури програми фітнес-студії.
3. Створення мобільного додатку для тренерів фітнес-студії «Body Sculptor».

Об'єкт дослідження – зручний облік тренувань для працівників фітнес-центру.

Методи дослідження – користуючись мобільним додатком з'явитсья досвід аналізувати прогрес клієнтів фітнес-студії.

РОЗДІЛ 1.
ІДЕЯ ТА РОЗГЛЯД ПІДХОДІВ ДО РЕАЛІЗАЦІЇ. ОПИС ОСНОВНИХ ЗАДАЧ ПРОГРАМНОГО ПРОДУКТУ

1.1. Створення ідеї та опис плану реалізації.

Після освоєння операційної системи андроїд було вирішено створити програму для фітнес-студії «Body Sculptor» за допомогою мови програмування Kotlin та розширюваної мови розмітки XML.

Кожній ідеї та розробці потрибний детальний план. План це є другий етап для створення програми. Плани бувають різні за обсягом, методом та видом, але для кожної програми вони є особливі. Для даного програмного продукту:

1. Вибір середовища розробки та мови програмування.
2. Створення списку вимог.
3. Розробка архітектури
    a. опис змінних у функціях,
    b. зв’язки між класами та функціями,
    c. створення таблиць для бази даних,
    d. опис функції.
4. Розробка дизайну.
5. Реалізація.

1.2. Вибір мови програмування.

Оскільки розробка мобільного додатку буде створюватися під платформу Андроїд, то основні мови програмування для цього Kotlin та Java. В даному випадку було обрано мову Kotlin, оскільки у 2019 році компанією на конференції для розробників I / O 2019 Google оголосила, що для програмування Котлин є більш приорітетним, і для Android розробників запропоновано під Android

- нові API та бібліотеки Jetpack будуть публікуватися спочатку на Kotlin. Котлін дозволяє писати значно менше коду, а це означає що обслуговування та тестування займатиме менше часу. Для інтерфейсу буде застосовано мову розмітки XML.

XML мова розмітки яка використовується в системі андроїд для створення інтерфейсу для редагування та створення особистих стилів.

1.3. Вимоги до програмного продукту та список основних задач.

Програма повинна:

- Виконувати реєстрацію користувача за допомогою емейлу та паролю, з можливістю підтвердження поштової скриньки та відновлення паролю в разі його втрати, Google-аккаунту та облікового запису Facebook.
- Створення бази данних клієнтів.
- Структуроване відображення інформації щодо ведення підопічних.

І розпочнемо з впровадження бібліотеки для роботи з базою даних. В нашому випадку ми будемо використовувати базу даних від американської компанії, постачальником хмарних послуг – Firebase. Firebase database Realtime дає можливість що б всі користувачі ділились одним екземпляром бази даних Realtime і автоматично отримували оновлення з оновленими даними.

По друге потрібно створити клас «Клієнт» для роботи з базою даних.

Останнє це зробити інтерфейс, який об'єднує функції програми. Буде написаний на XML та Kotlin.

1.3.1.Вимоги до бібліотек для роботи з базою даних.

В даний етап буде входити написання вимог до взаємодії з базою даних Firebase Database Realtime

Розпочнемо з підключення бібліотеки бази даних, хмарного сховища, яке буде зберігати фото профілів користувачів та їх клієнтів та бібліотеки для реєстрації користувачів.

Вимоги до бази даних.

Перша вимога. Створити структуру бази даних для зберігання інформації про клієнтів та користучів.

Перший стовпець буде зберігати id cid створеного клієнта. Цей id повинен бути унікальним та мати великий розмір для зберігання великих об’ємів даних.

Другий стовпець. Буде зберігати ID uid користувача, що увійшов в систему. ID створюється автоматично при реєструванні користувача.

Третій стовпець. Буде зберігати ID imid в цей стовпець буде записуватися id зображення клієнта, має формат "$cid||||$uid". Даний стовпець буде мати тип string.

Четвертий стовпець. Буде зберігати ім’я firstName клієнта. Даний стовпець буде мати тип string.

П’ятий стовпець. Буде зберігати фамілію secondName клієнта. Даний стовпець буде мати тип string.

Шостий стовпець. Буде зберігати дату Date дату народження клієнта. Даний стовпець буде мати тип datetime і мати вигляд DD/mm/yyyy.

Сьомий стовпець. Буде зберігати поштову скриньку email клієнта. Даний стовпець буде мати тип string.

Восьмий стовпець. Буде зберігати назву тарифу tarif клієнта. Даний стовпець буде мати тип string.

Дев’ятий стовпець. Буде зберігати назву мети goal клієнта. Даний стовпець буде мати тип string.

Останній, десятий стовпець буде зберігати вагу weight клієнта. Даний стовпець буде мати тип float.

База даних клієнтів матиме таку структуру(Рис. 1.1).

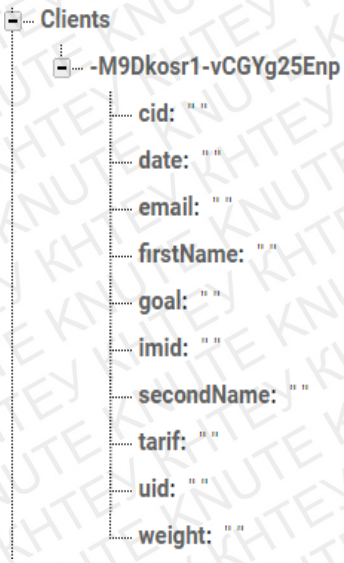

Рис. 1.1. Структура вузла даних клієнтів.

Друга таблиця буде зберігати дані користувачів. Дані до цієї таблиці вносяться під час реєстрації.

Перший стовпець. Буде зберігати ID uid користувача, створюється за допомогою бібліотеки Firebase автоматично. Даний стовпець має тип даних string.

Другий стовпець. Буде зберігати ім’я firstName користувача. Даний стовпець буде мати тип string.

Третій стовпець. Буде зберігати фамілію secondName користувача. Даний стовпець буде мати тип string.

Четвертий стовпець. Буде зберігати ID imid в цей стовпець буде записуватися id зображення користувача, має формат "$cid||||$uid". Даний стовпець буде мати тип string.

П’ятий стовпець. Буде зберігати поштову скриньку email клієнта. Даний стовпець буде мати тип string.

Таблиця має наступну структуру (Рис. 1.2).

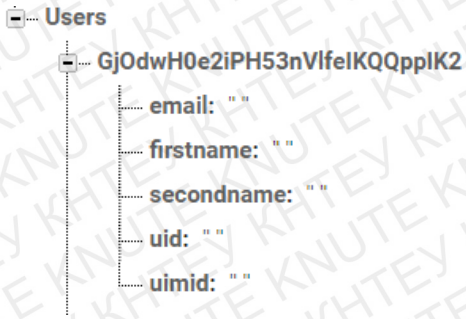

Рис. 1.2. Вузол даних користувачів.

Підключити бібліотеки, що необхідні для роботою з базою даних.

Перша бібліотека Firebase-Realtime-Database, надає методи для додавання та читання вузлів у вигляді JSON до хмарної бази даних в реальному часі. Передбачена можливість безпечного редагування та видалення вузлів з бази даних.

Друга бібліотека Firebase-Cloud-Storage, надає безпечне хмарне сховище, з безпечною фунцією завантаження даних до сховища, наприклад, якщо у користувача відбувається збій інтернет-з'єднання, то після відновлення завантаження продовжить свою роботу без втрати даних та помилок.

Третя бібліотека Firebase-Auth, направлена на те, що б зробити простішою безпечну авторизацію, одночасно покращуючи можливості входу та підключення користувачів. Біблотека має комплексне рішення для ідентифікації та підтримки облікових записів електронної адреси та паролів, аутенифікацію телефону, реєстрацію через Google, Twitter, Facebook, GitHub та багато інших.

1.4. Вимоги до створення візуального інтерфейсу

Інстерфейс створюється інтуїтивно зрозімулим, використовуючи стиль та методичні рекомендації Material Design

Activity авторизації, має в собі назву та логотип мобільного додатку, два поля для вводу емейлу та паролю, клікабельний текст для відображення діалогу

в разі втрати паролю, кнопка «Увійти», дві кнопки для авторизації через Google та Facebook, та кнопка для створення аккаунту.

Activity реєстрація, містить кнопку для завантаження фото користувача, поля вводу імені, фамілії, емейлу, паролю та підтвердження паролю, кнопка «Зареєструватися», дві кнопки для авторизації через Google та Facebook, та кнопка для відображення умов користувацької згоди.

Головне activity що містить FrameLayout, для відображення Fragment's, toolbar із кнопкої бокового меню DrawerLayout в якому знаходяться фрагменти тарифів, програм тренувань, цілей, справки, та налаштувань. Також на головному activity знаходиться BottomNavigationView з кнопками навігації між фрагментами клієнтів, головним фрагментов та фрагментом календар.

На фрагментів з клієнтами знаходиться RecircleView що наповнюється індивідуальними рядками які містять круглі зображення клієнтів в кольоровій рамці що відповідає стану їх абонементів: зелений – більше 50% тренувань, жовтий – лишився тиждень тренувань та червоний, коли абонимент закінчився. По центру рядку відображується ім'я та фамілія клієнта, з правої сторони кнопка, для відображення карточки клієнта,, яка містить основну інформацію, антропометричні дані з можливістю редагування, та відображення змін цих даних у вигляді графіку.

Головний фрагмент містить також RecircleView що наповнюється індивідуальними рядками які містять CheckBox'и для відмітки виконаного тренування, ім'я клієнта, вид тренування та час.

Фрагмент календар даж можливість вибрати необхідних день на календарі та подивитися тренування що були або будуть.

Фрагмент тарифів передбачений для опису, редагування, та створення нових тарифів.

Фрагмент цілей передбачений для опису, редагування, та створення нових цілей.

Фрагмент програм тренувань містить опис, редагування, та створення нових програм.

Фрагмент зі справкою містить інформацію для освоєння програми, опис версій та зміни в оновленнях.

1.5 Аналіз існуючих програм для ведення та обліку клієнтів у фітнес-студіях.

Мобільних додатків для обліку та контролю клієнтів фітнес-студій існує досить мало на платформі Android, всі мають свої переваги та недоліки.

Програма 1. «ЯТренер» – програма будує детальний звіт по оплаті та тренувань клієнтів, відображає суму та кількість сплачених тренувань. Надає можливість вести плануванння тренувань. Звіти надють змогу зробити аналіз пророблені роботи, також має основні функції дял роботи з даними.. Серед недоліків є те, що програма має застарілий, та складний інтерфейс та мало налаштувань. Також не передбачена функція авторизації, що не дає можливості синхронізувати дані між пристроями та відновити дані після повторної інсталяції додатку.

Програма 2. «Журнал-Тренера» – програма розрахована на тренажерний зал, передбачена можливість ведення обліку тренувань, має зрозумілий інтерфейс. Можливо виконати вхід за допомогою облікового запису для синхронізації даних.

1.6 Висновок до розділу 1.

Отже, в даному розділу було описано технічне завдання у вимогах для створення програмного продукту.

Проводячи аналіз зазначених програм та інших. Можна зробити висновок: вибору серед таких програм практично немає, а самі програми мають або недоліки з обмеженою кількістю функцій, та поганою оптимізацією або незрозумілим інтерфейсом та ризиком втратити свії свої дані в разі видалення програми. Ідеальний мобільний додаток повинен бути зрозумілим, мати

достатню кількість основних функцій, давати можливість для синхронізації між пристроями та впевненість в безпеці даних.

Також розписано вимоги до бази даних, бібліотеки керування БД, бібліотеки створення хмарного сховища. Розглянута ідея та вирішення проблеми.

Розглянуто вимоги до створення бази даних, розписані вимоги до типу та розміру даних, зазначені назви вузлів у дереві JSON.

Було розглянуто мови програмування такі як Kotlin та XML. Kotlin використовується для написання класів та функцій для роботи з БД та іншими складовими мобільного додатку. Firebase дає можливість викоористання безкоштовного сервера, дял збереження даних, що є важливим для даної програми. XML допоможе створити більш зручний інтерфейс з дотриманням стилю Material Design.

В даному розділу була виконана теоретична частина диплому де було описано вимоги до створення програми фітнес-студії «Body Sculptor», проаналізовано інші програми для підкреслення проблем та уникання їх.

РОЗДІЛ 2.
ОПИС ОСНОВНИХ КЛАСІВ

2.1 Опис класу ClientAdapter

Даний клас створений для відображення основної інформації про клієнта , що береться з бази даних (Рис. 2.1). Даний клас має в собі 2 функції з роботою бази даних на вивід та ввід інформації.

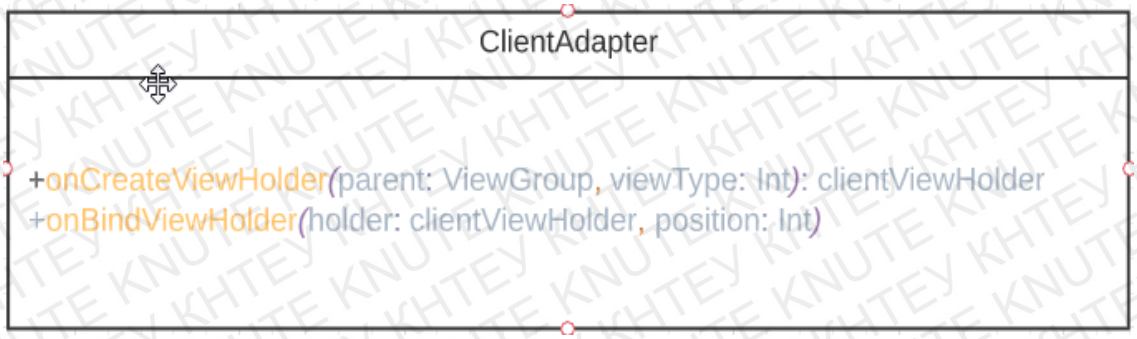

Рис. 2.1. ClientAdapter в UML діаграмі.

Функція onCreateViewHolder(parent: ViewGroup, viewType: Int) створює новий об'єкт ViewHolder щоразу, коли RecyclerView потребує цього. В той момент, коли створюється layout рядки списку, передається об'єкту ViewHolder, і кожен дочірній view-компонент може бути знайдений і збережений.

Функція onBindViewHolder () приймає об'єкт ViewHolder і встановлює необхідні дані для відповідного рядка у view-компоненті.

2.2 Опис класу Client.

Основний клас програми який створений для того що б створівати нові екземляри клієнтів та передавати їх до бази даних, складається з полів десяти полів, які є основною інформацією про клієнта: ID клієнта, ID Користувача, ID зображення профілю, ім’я, фамілія, дата народження, емейл, мета, тариф та вага.

Також створено конструктор simpleClientInfo який буде передавати в RecycrcleView необхідну інформацію дял відображення на фрагменті.

2.3 Опис ClientsFragment.

ClientsFragment створюється для представлення поведінки фрагменту (Рис. 2.2).

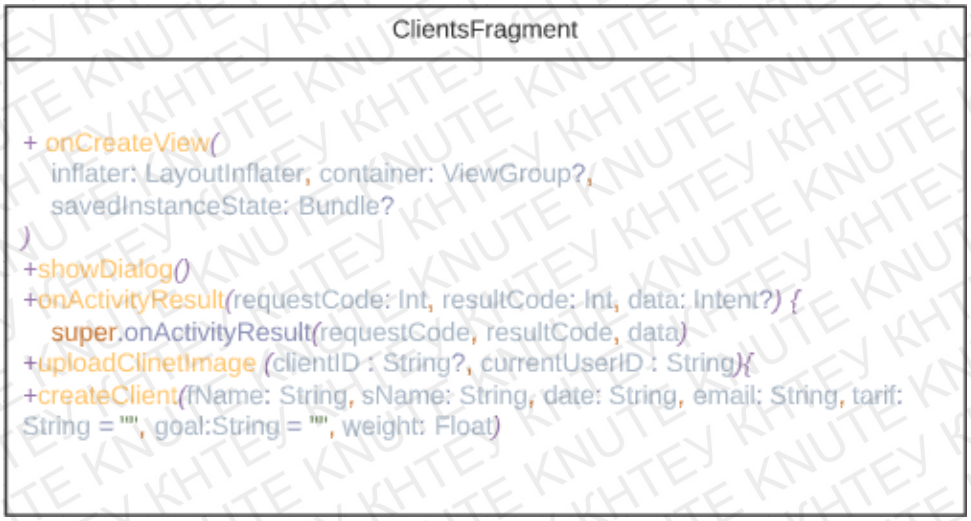

Рис. 2.2. ClientsFragment в UML діаграмі.

Функція onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?) задає початкову установку параметрів при ініціалізації фрагмену.

Функція uploadClinetImage (clientID : String?, currentUserID : String) для завантаження зображення клієнта до Firebase-Storage.

Функція createClient(fName: String, sName: String, date: String, email: String, tarif: String = "", goal:String = "", weight: Float) перевіряє всі поля для створення екземляру об’єкта, в разі якщо все вірно, надсилає запит до БД та створює нового клієнта.

2.4 Ознайомлення з Android Studio.

Android Studio є офіційним інтегрованим середовищем розробки (IDE) для операційної системи Android від Google, побудованого на основі програмного

забезпечення JetBrains IntelliJ IDEA і розробленого спеціально для розробки Android.

Він доступний для завантаження в операційних системах на базі Windows, macOS та Linux.

Android Studio було оголошено 16 травня 2013 року на конференції Google I / O. Це було на етапі попереднього перегляду, починаючи з версії 0.1 в травні 2013 року. Перша стабільна збірка була випущена в грудні 2014 року, починаючи з версії 1.0.

7 травня 2019 року Котлін замінив Java як бажану мову Google для розробки додатків для Android. Java все ще підтримується, як і C ++

Особливості Android Studio:

- У програму вбудований емулятор, що дозволяє легко перевірити роботу додатку на пристроях з різними екранами, співвідношеннями сторін. Ця функція стала особливо актуальна після виходу смартфонів у яких встановлення співвідношення сторін 18:9.
- В програмі реалізовані всі сучасні зручності для упаковки коду.
- Локалізація стає набагато легшою завдяки функції SDK.
- Дозволяє розробляти програми не тільки на мобільні телефони, але і для годинників, телевізорів.
- Рефакторинг вже готового коду.
- Велика бібліотека з готовими шаблонами, що дозволяє суттєво економити час.

2.5 Висновок до розділу 2.

Можна зробити висновки, у другому розділу описано класи в UML діаграмах та розглянута архітектура проекту.

Було розглянуто класи які взаємодіють з базою даних для формування даних для вводу та виводу інформації до компонентів Activity та Fragment.

Проаналізовано платформу Android Studio було описано її особливості та підкреслення мов програмування.

РОЗДІЛ 3.
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ФІТНЕС-СТУДІЇ «BODY SCULPTOR»

3.1. Створення проекту в Android Studio.

Запускаємо Android Studio та вибираємо створити новий android проект (Рис. 3.1).

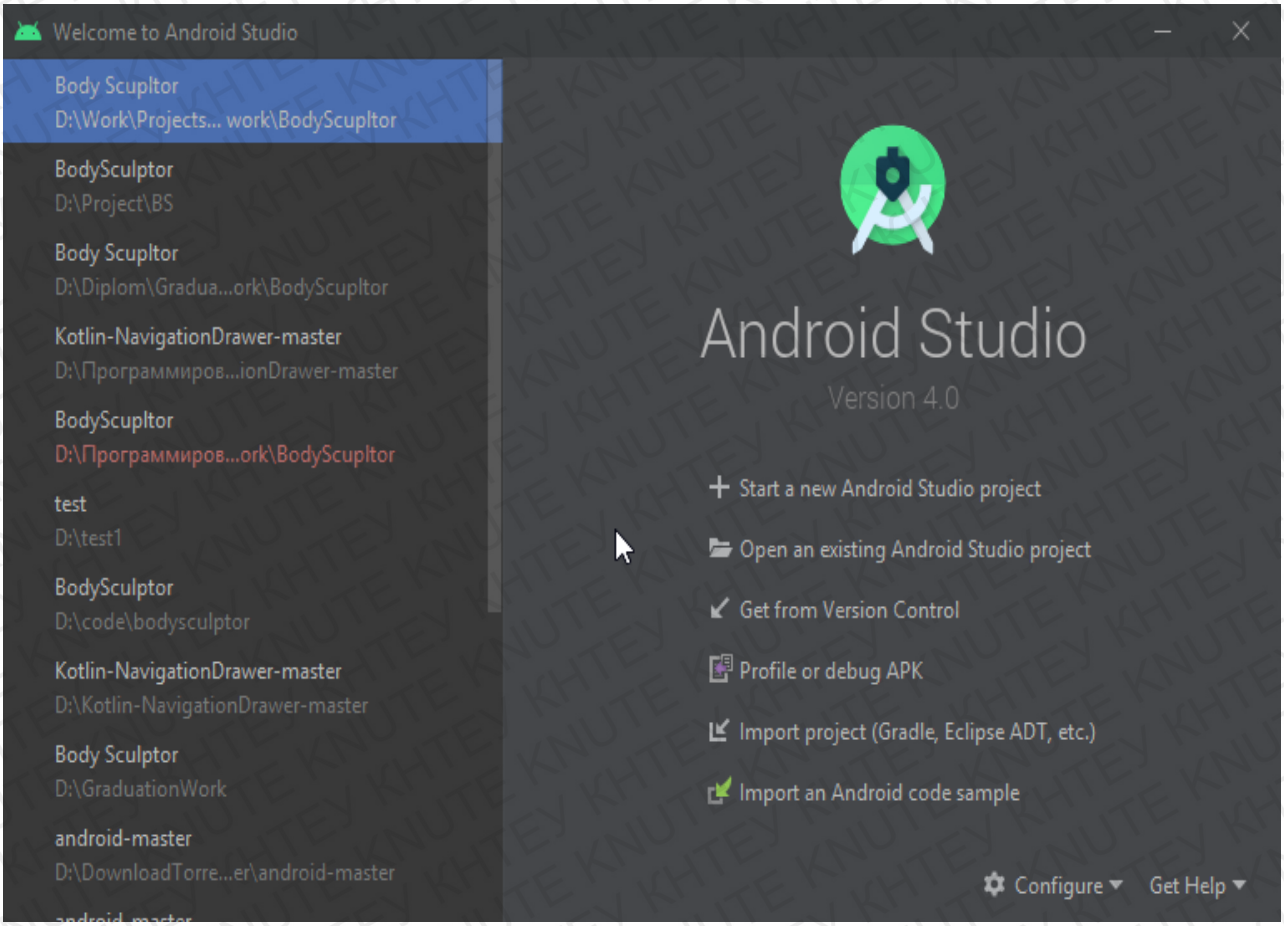

Рис. 3.1. Стартове вікно Android Studio.

Переходимо до наступної форми налаштування проекту тут потрібно обрати Empty Activity (Рис. 3.2).

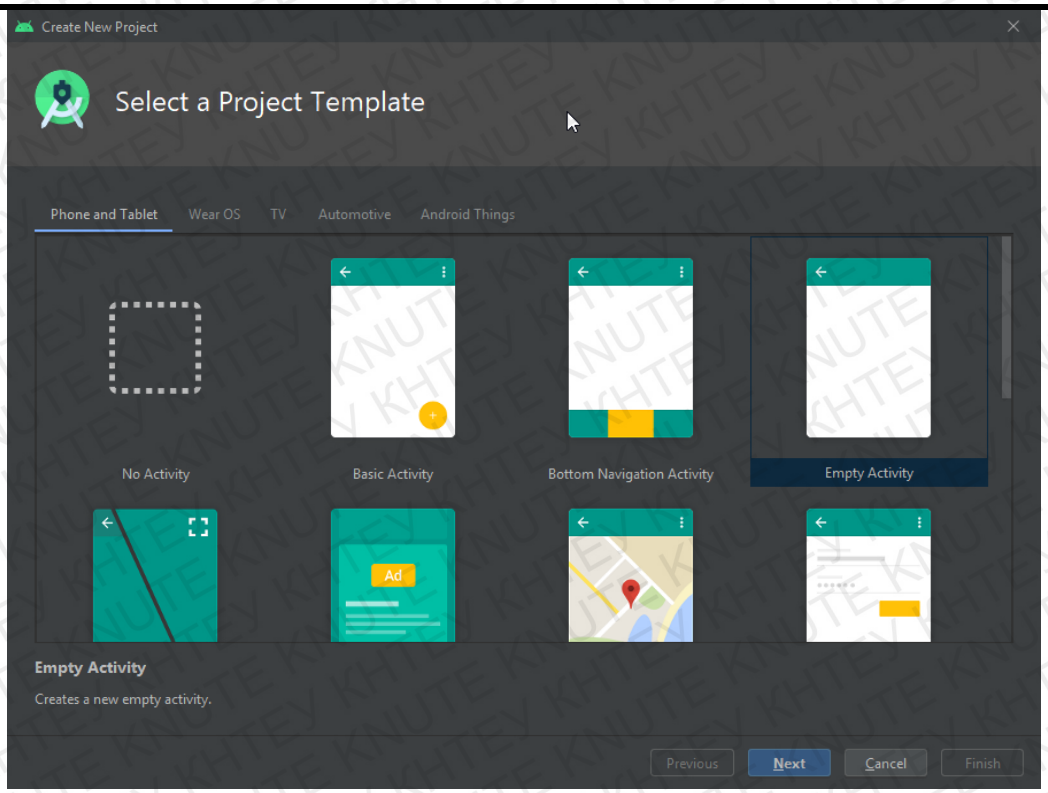

Рис. 3.2. Вікно вибору проекту Android Studio.

Далі вікно налаштування назви та каталогу розташування вибіраємо мову програмування Kotlin та SDK API 23(Рис. 3.3).

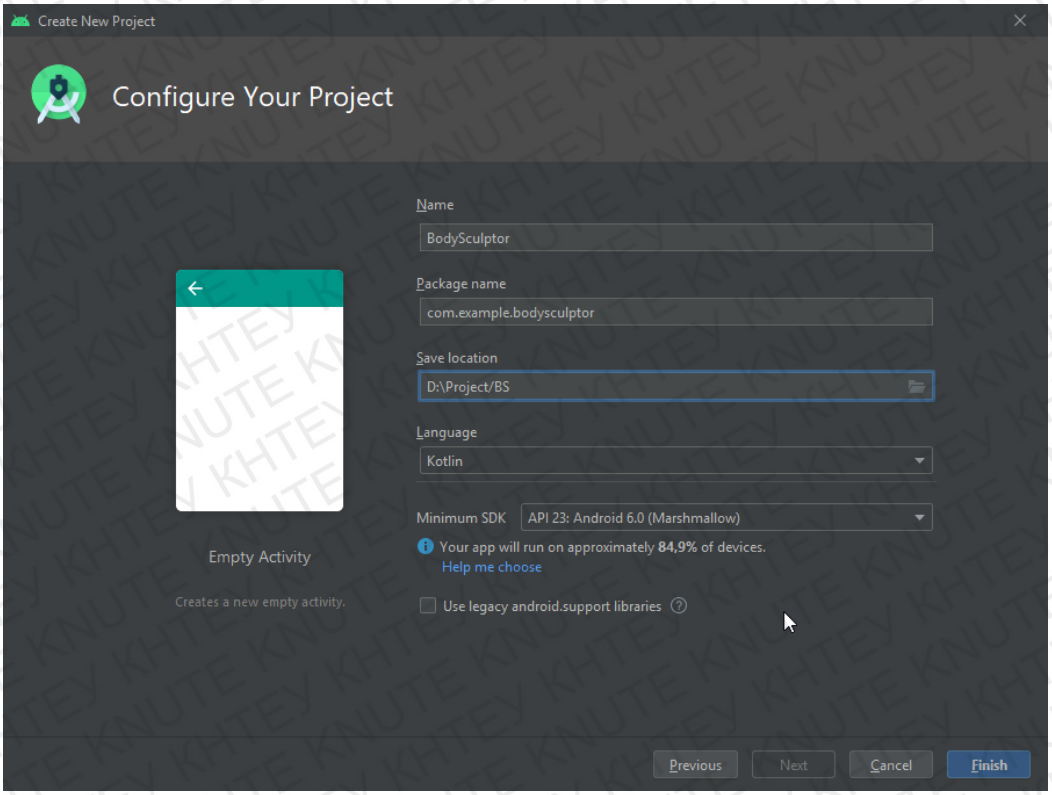

Рис. 3.3. Вікно налаштування назви та каталогу Android Studio.

Проект створений можна розпочинати роботу по створенню програмного додатку для фітнес-студії «Body Sculptor» (Рис. 3.4).

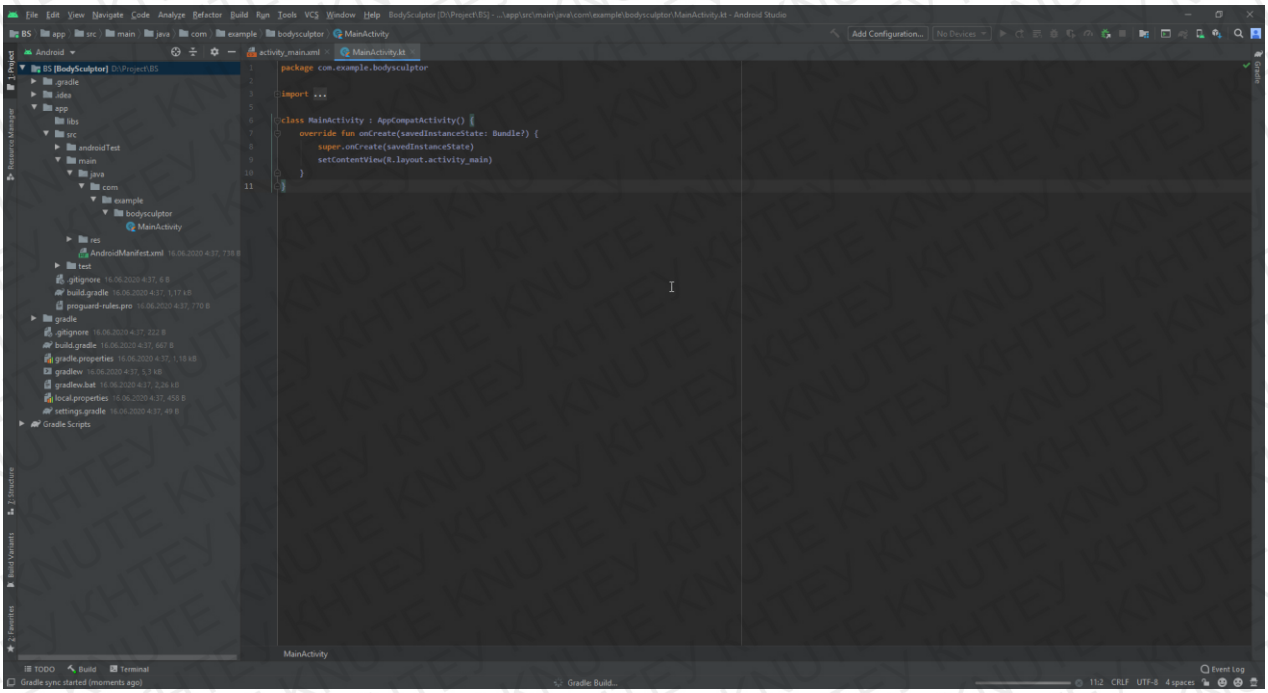

Рис. 3.4. Основне вікно роботи в Android Studio.

3.2. Підключення всіх необхідних бібліотек

Відкриваємо build.gradle (app) та підключаємо необхідні для роботи бібліотеки (Рис. 3.5).

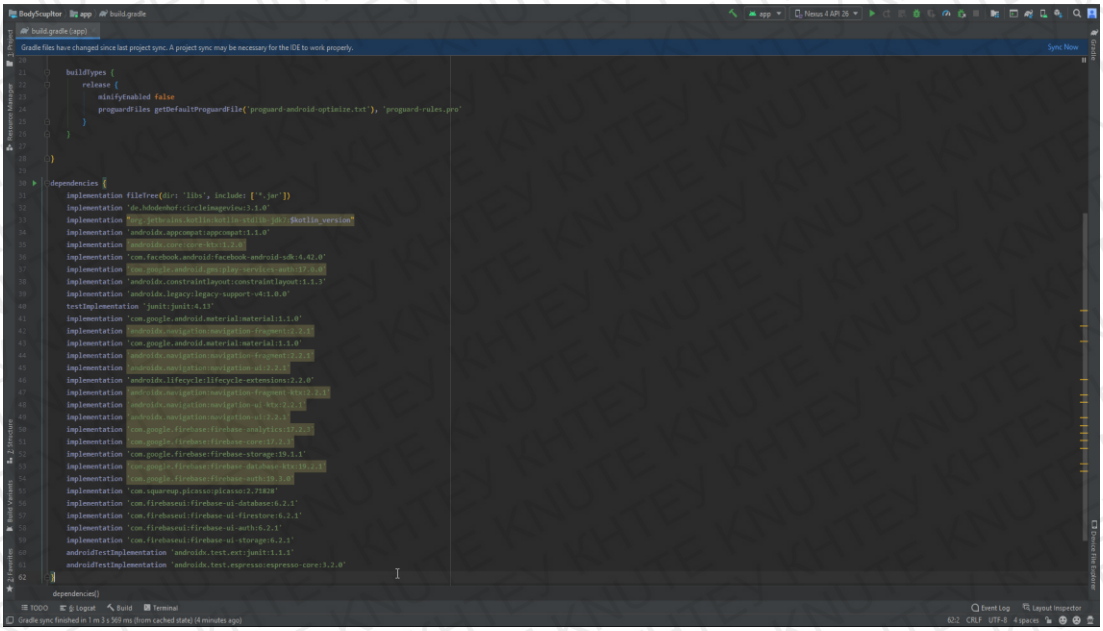

Рис. 3.5. Файл з підключеними бібліотеками build.gradle.

3.2. Підключення Firebase

Після підключення бібліотек необхідно інтегрувати проект із сервісами Firebase а саме Auth, Realtime Database та Firebase Storage. Необхідно натиснути вкладку Tools та вибрати потрібний нам пункт меню (Рис. 3.6).

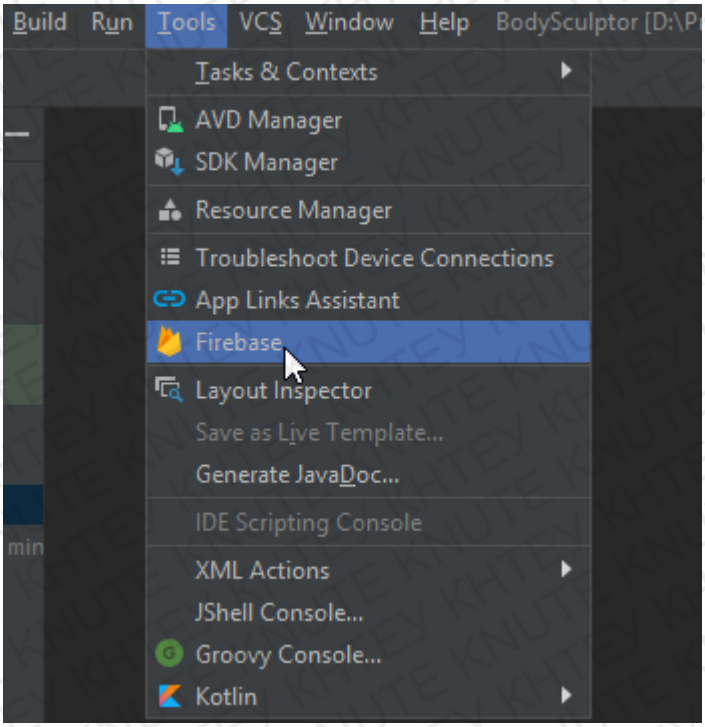

Рис. 3.6 Контекстне меню дял підключення сервісів Firebase.

З’явиться меню Assistant. Ньому вибіраємо сервіси авторизації, бази даних та сховища (Рис. 3.7).

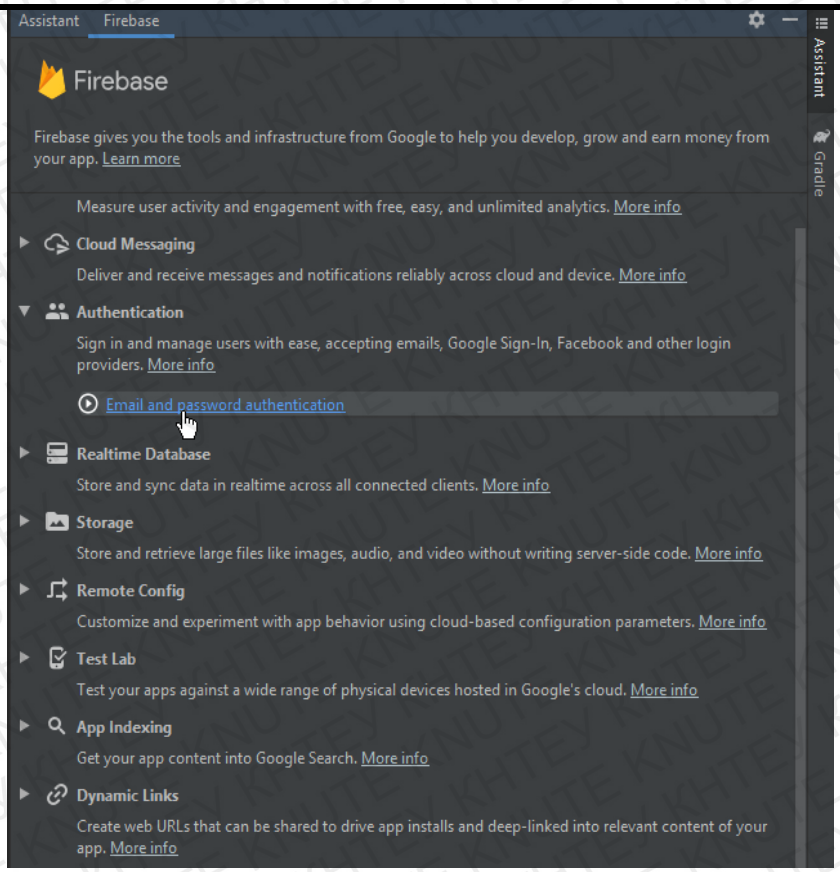

Рис. 3.7 Вкладка Asistant.

Так як бібліотеки вже підключені необхідно додати потрібний код та перевірити на сайті, що все прауює (Рис. 3.8).

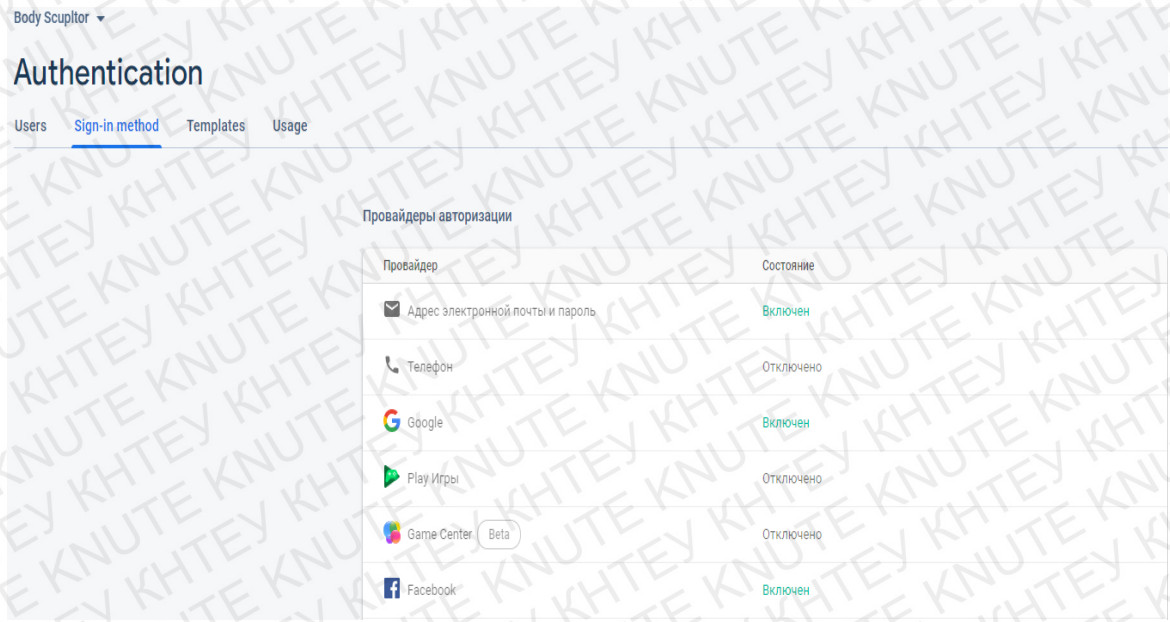

Рис.3.8 Підключені провайдери авторизації на сайті Firebase.

3.3. Створення класу Client.

Для створення класу DatabaseHelper потрібно натиснути ПКМ(правою кнопкою миші) на папку з проектом та вибрати пункт новий та kitlin file / class (Рис. 3.9).

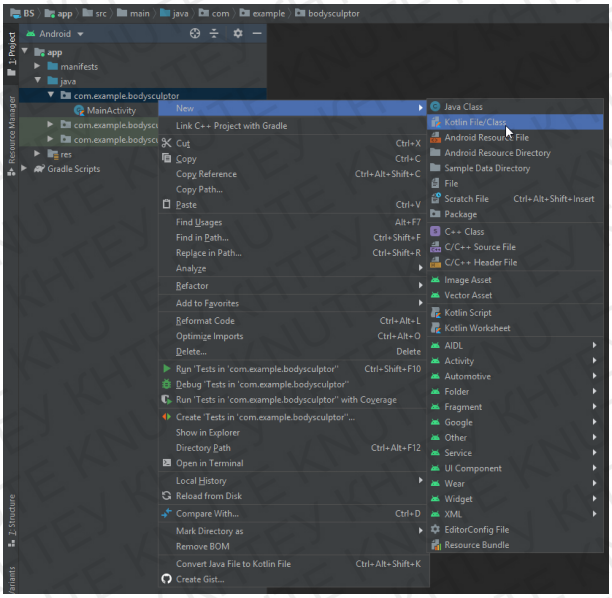

Рис. 3.9. Створення Class в Android Studio.

З’являється вікно з налаштуванням класу, змінюємо лише назву(Рис. 3.10).

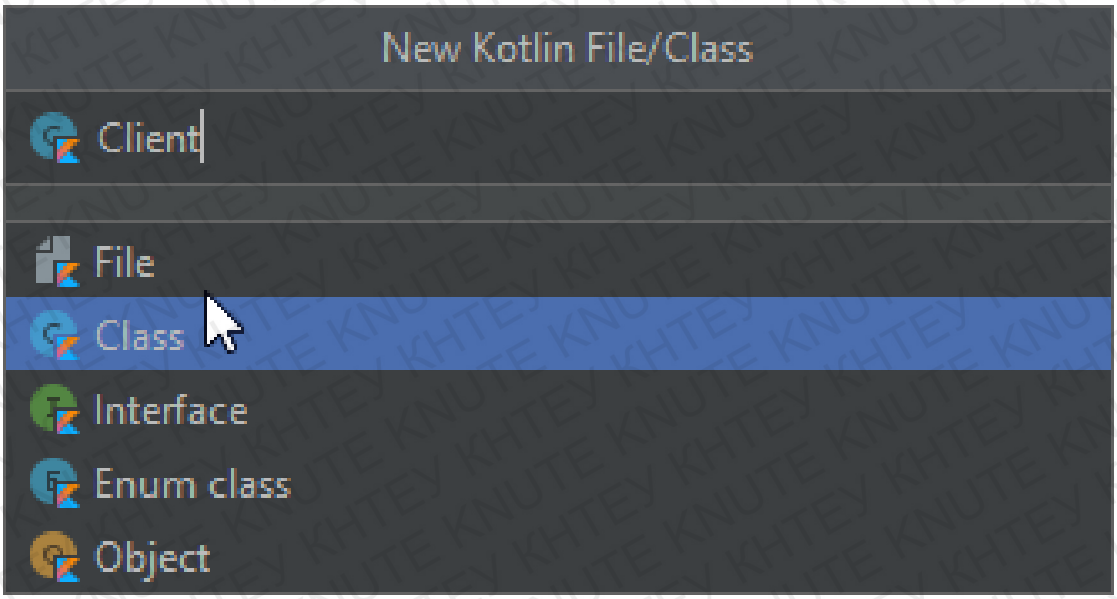

Рис. 3.10. Вікно налаштування Class в Android Studio.

Після натиску кнопки «ОК» з’являється вікно редагування тексту та починаємо писати програмний код для класу Client (Рис. 3.11).

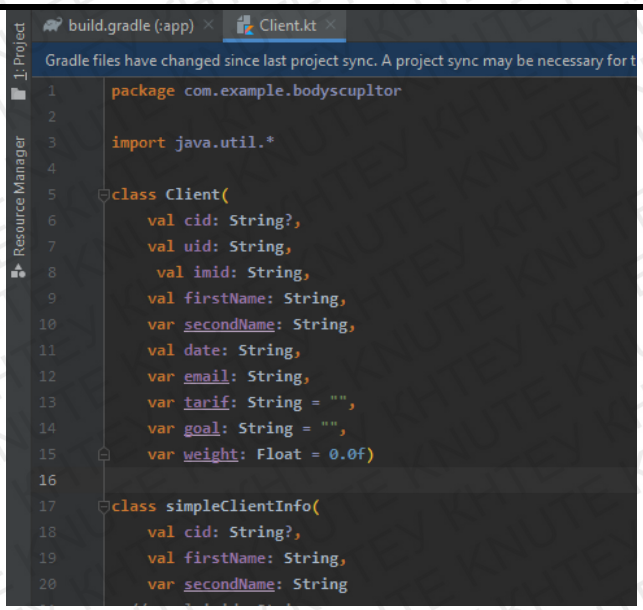

Рис. 3.11. Частина коду класу Client.

3.4. Створення класу clientViewHolder.

Створення класу clientViewHolder починається з кроків натиснути ПКМ(правою кнопкою миші) на папку з проектом та вибрати пункт новий та Kotlin file/class (Рис. 3.12).

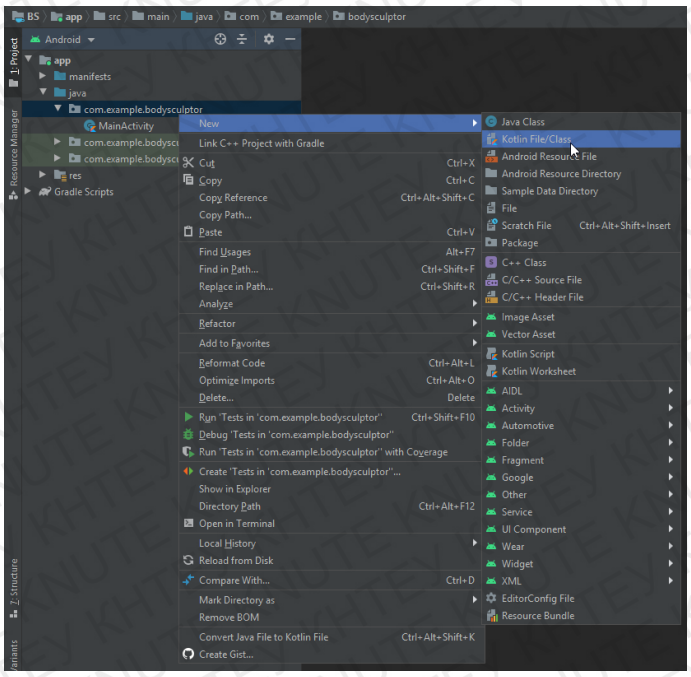

Рис. 3.12. Створення Class в Android Studio.

З’являється вікно з налаштуванням класу, змінюємо лише назву (Рис. 3.13).

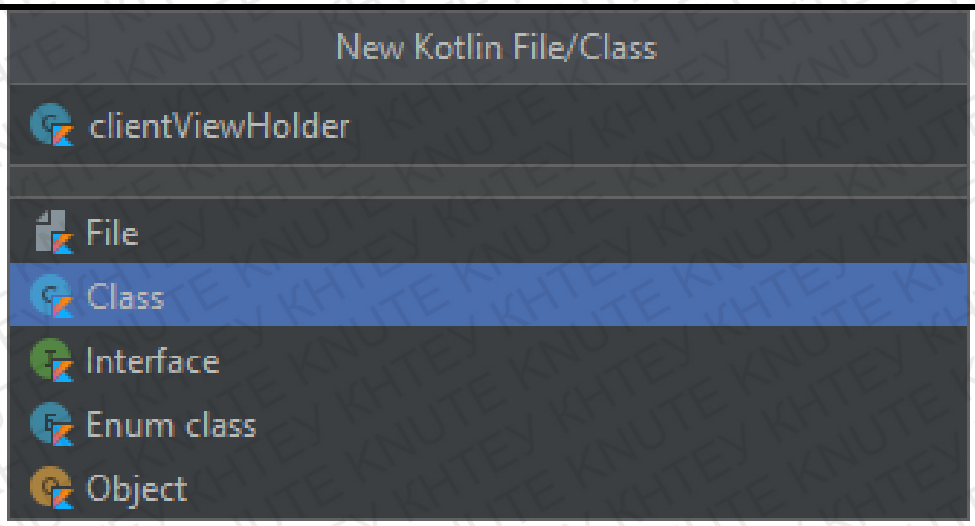

Рис. 3.13. Вікно налаштування clientViewHolder в Android Studio.

Після натиску кнопки «ОК» з'являється вікно редагування тексту та починаємо писати програмний код для класу clientViewHolder (Рис. 3.14).

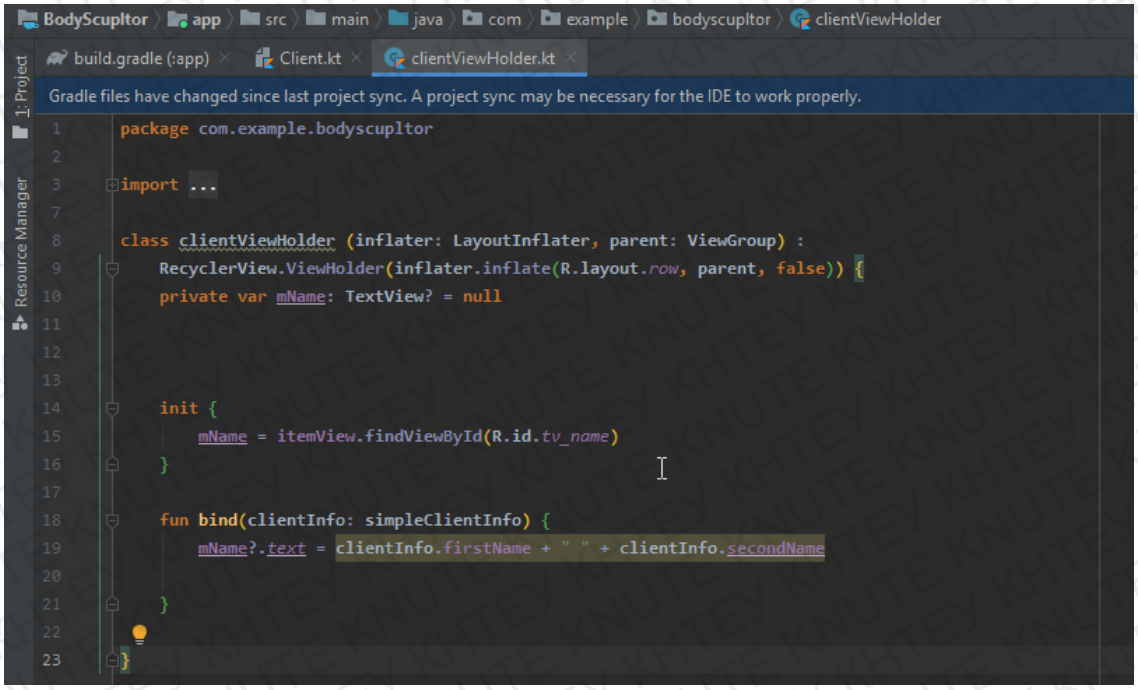

Рис. 3.14. Частина коду класу clientViewHolder.

3.5. Створення Activity.

Основні класи були створенні тепер потрібно створити Activity для зручного керування користувачем (Рис. 3.15).

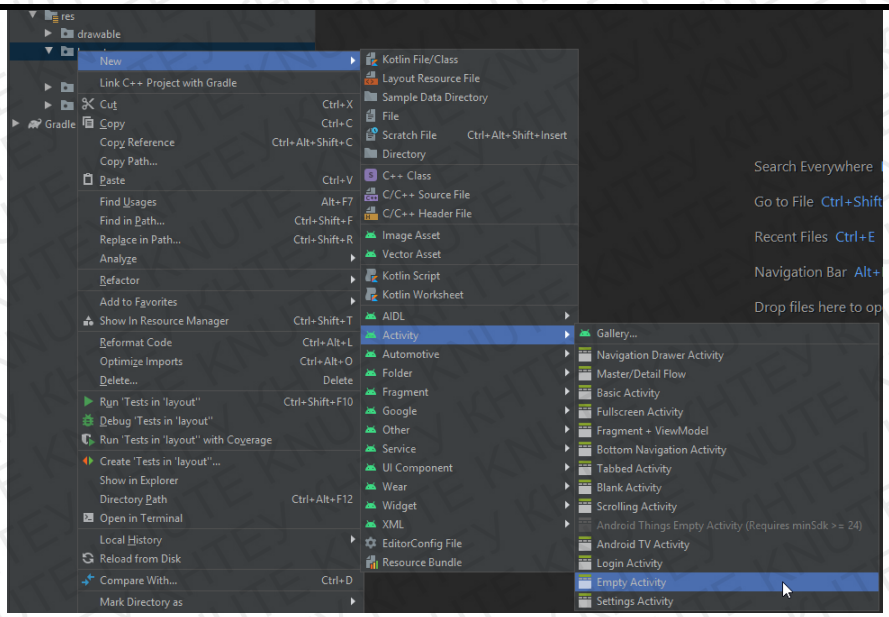

Рис. 3.15. Створення Activity.

Вікно відображення налаштування Activity (Рис. 3.16).

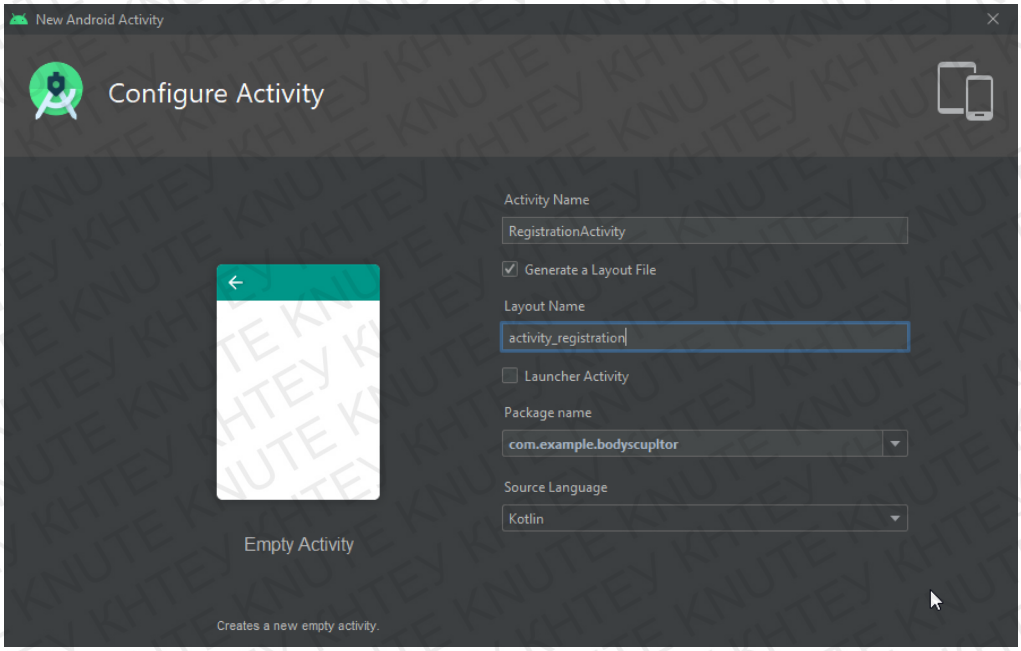

Рис. 3.16.Вікно налаштування Activity.

3.6 Створення Fragment.

Після створення activity необхідно сворити фрагменти, вони дозволяють створити в архітектурі проекту більг гнучку та зручну навігацію(Рис. 3.17).

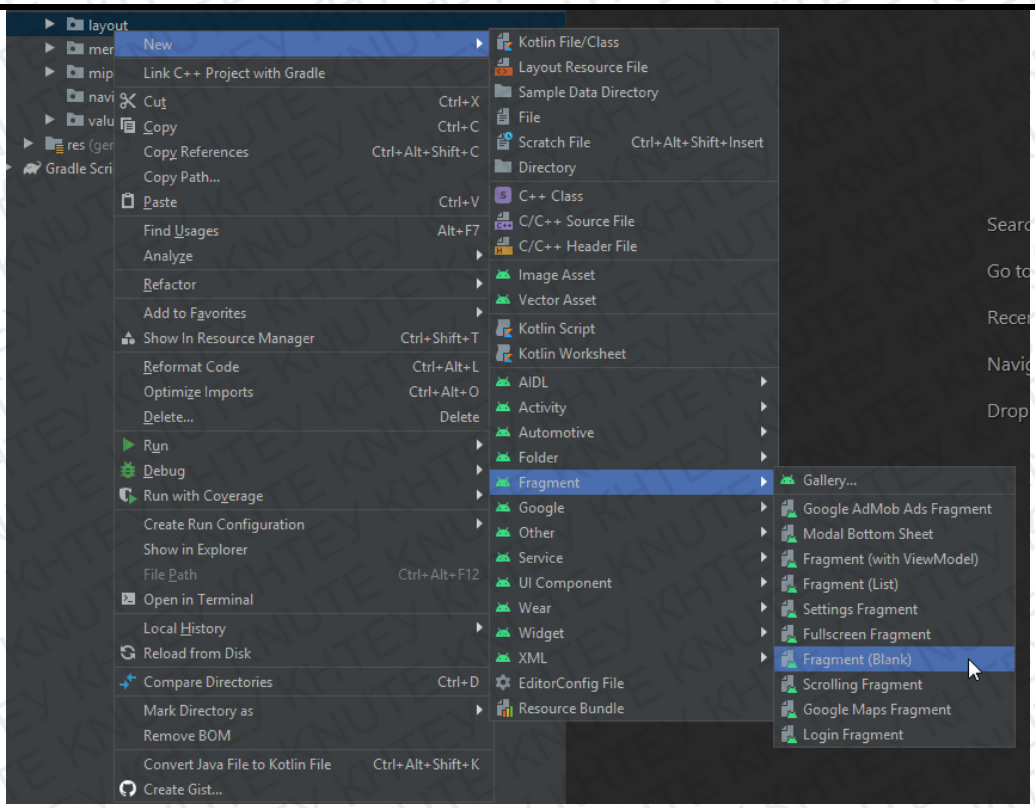

Рис. 3.17 Створення Fragment. Контекстне меню.

Після цього задаємо ім’я для фрагменту та файлу що відповідає за цей фрагменту (Рис. 3.18).

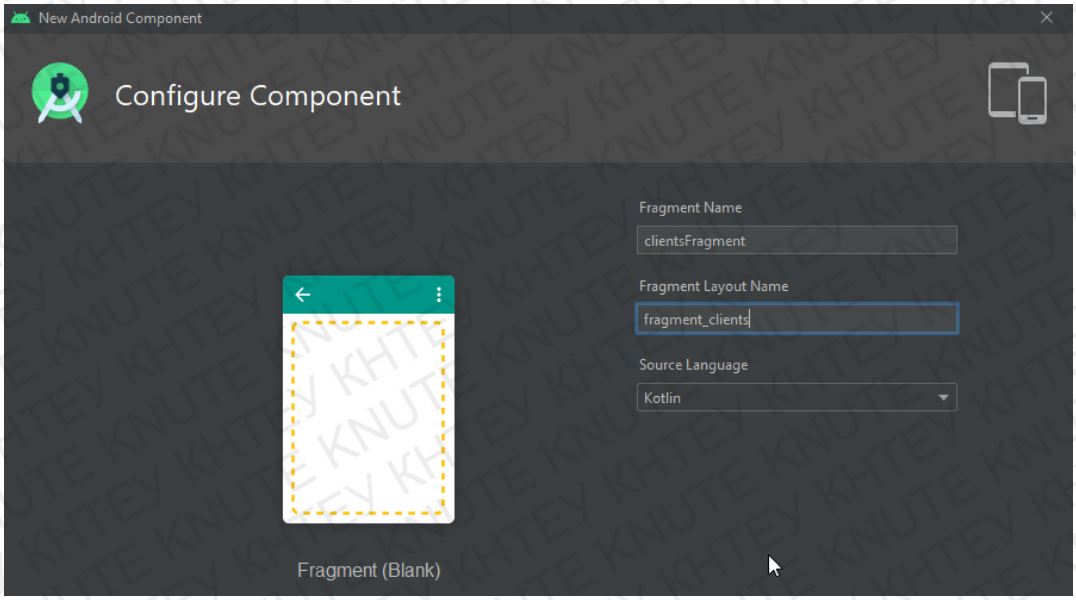

Рис. 3.18 Створення Fragment.

Залишилося створити всі заплановані activity та fragment, які були зазначені в архітектурі програмного забезпечення (Рис. 3.19).

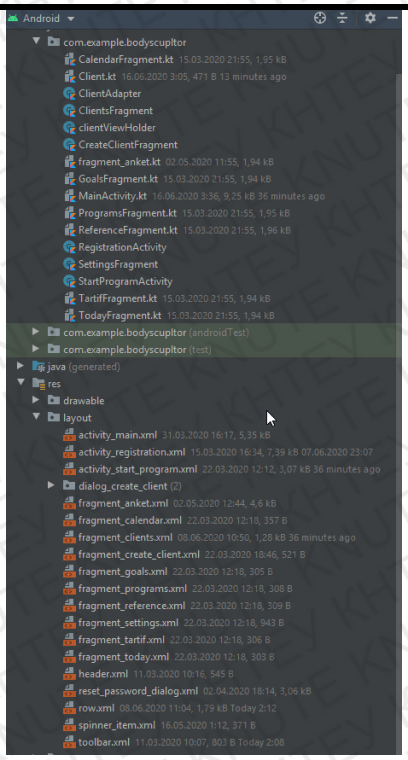

Рис. 3.19. Готові всі Activity та Fragment.

Після створення всіх Activity, Fragment та підключення Firebase можемо починати писати код (Рис. 3.20).

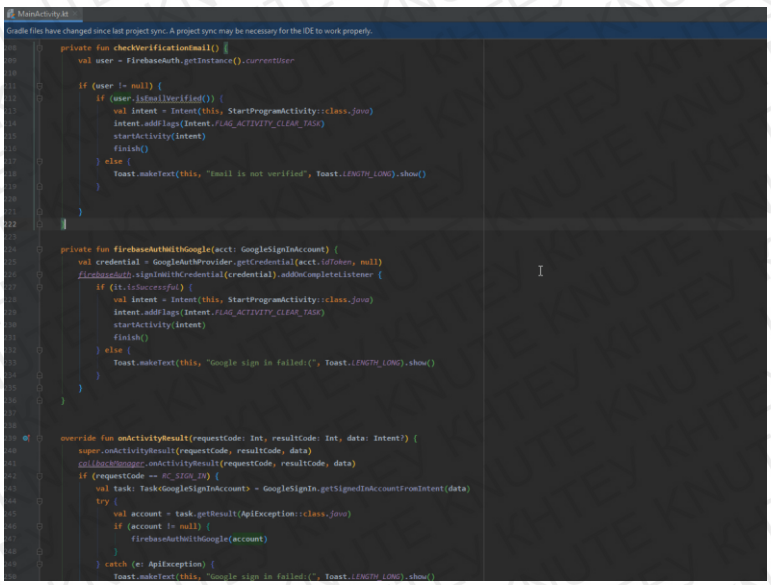

Рис. 3.20. Авторизація користувачів на activity входу.

3.7 Створення дизайну Activity та Fragment.

Налаштування activity виконується у файлі xml. Для дизайну було обрано простий інтерфейс для більш швидкого освоєння користувачем.

Головне вікно редагується файлом Activity_main.xml (Рис. 3.21).

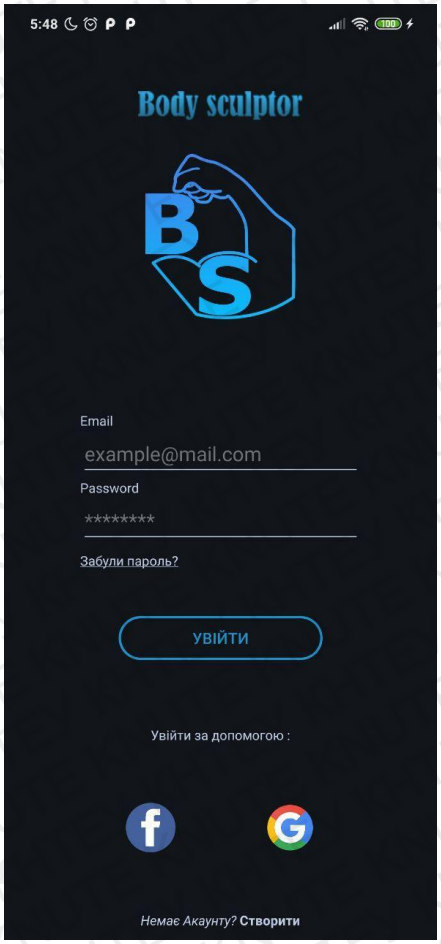

Рис. 3.17. Дизайн Activity_main.

Вікно реєстрації редагується файлом activity_registration.xml (Рис. 3.22).

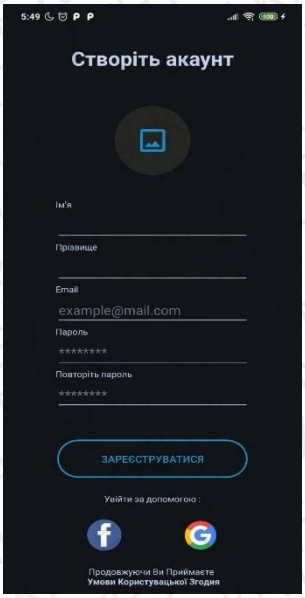

Рис. 3.22. Дизайн Activity_registration.

Діалог скидання паролю редагується файлом reset_password_dialog.xml (Рис. 3.23).

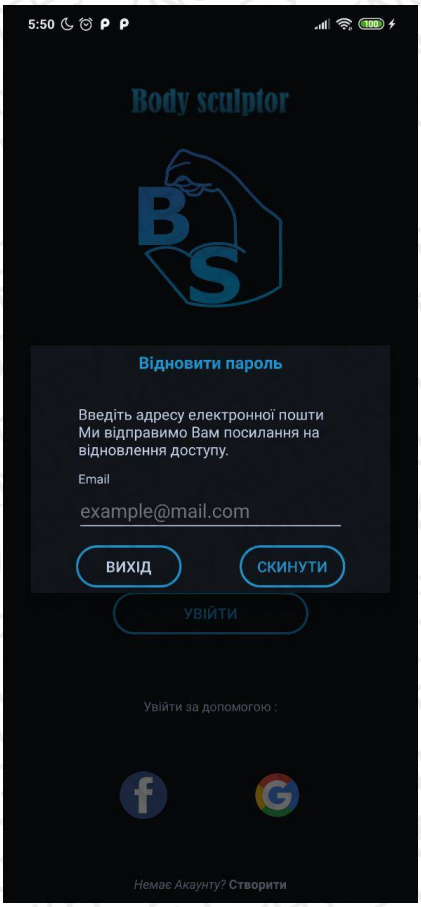

Рис. 3.23 Дизайн Activity_ settings.

3.8 Інструкція застосування програми для фітнес-студії «Body Sculptor».

Після скачування програми потрібно натиснути на програмний продукт з назвою «Body Sculptor». Відкривається перше вікно програми (Рис. 3.24).

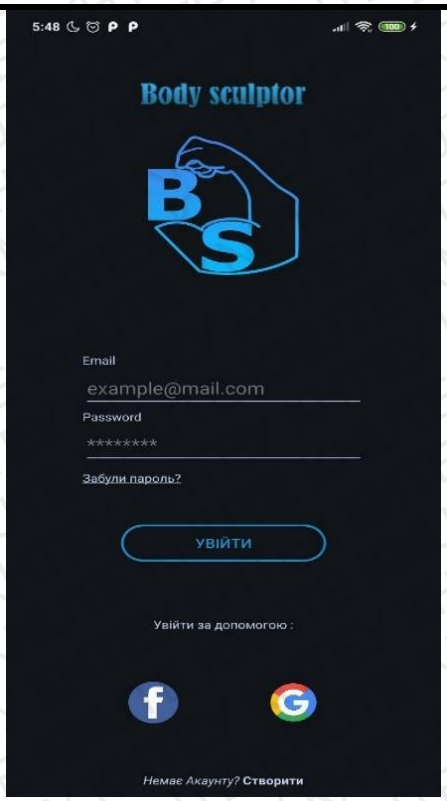

Рис. 3.24. Головне вікно.

Для початку потрібно натиснути на текст «Створити» для відкриття вікна реєстрації користувача (Рис. 3.25).

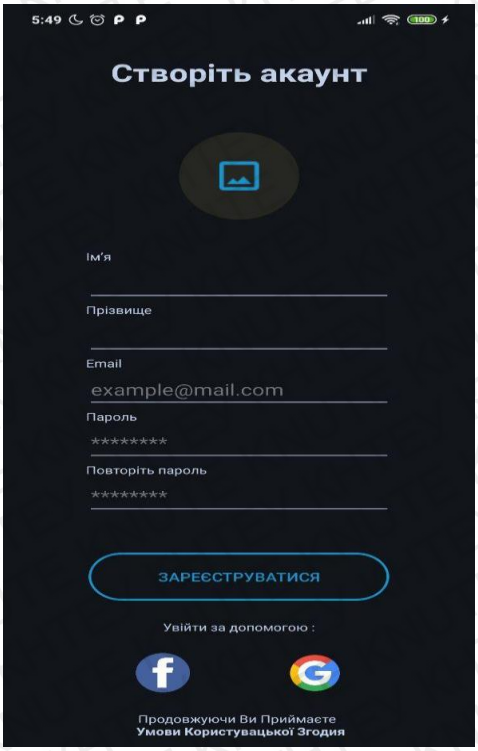

Рис. 3.25 Вікно реєстрації.

Після заповнення натискаємо кнопку «Зареєструватися». Користувача перекине до головного вікна.

Тепер потрібно підтвердити електронну адресу,яку було введено при реєстрації, після цього вводимо дані від облікогового записа, та натискаємо увійти. Потрабляємо до системи там можемо розпочинати роботу. Відкриється вікно де будуть відображення тренування які ще потрібно провести в даний день (Рис. 3.26).

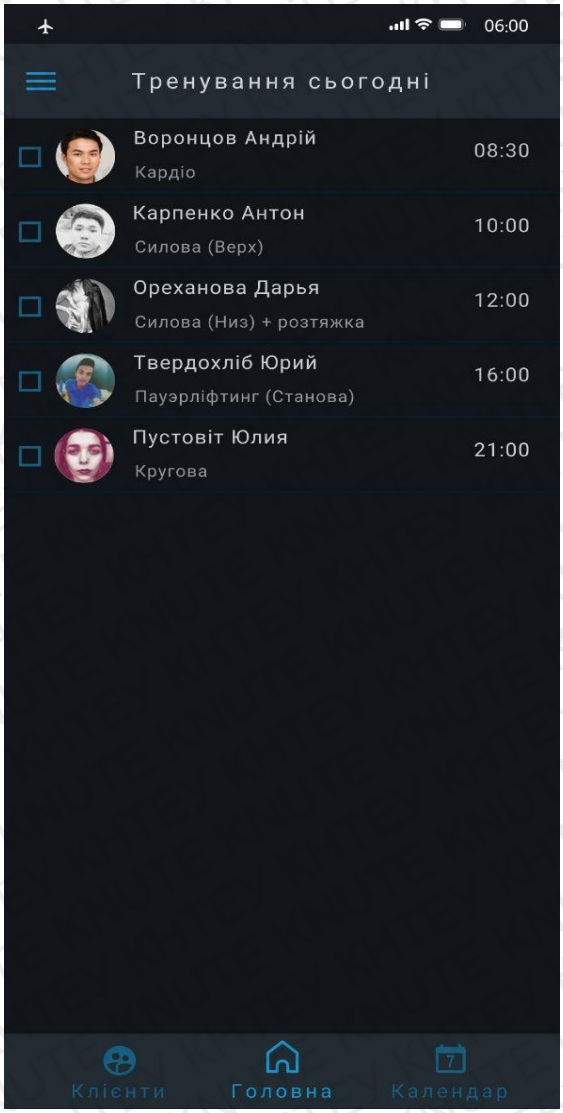

Рис. 3.26 Fragment Головна.

Перейдемо після цього на вкладку клієнти, клацнувши по елементу в нижній панелі навігації. Тут зображені всі клієнти, колом навколо вказано стан абонименту. В вехній панелі іконки для пошуку та фільтру клієнтів. Зелене коло

– більше 50% тренувань, жовте – лишився тиждень тренувань та червоний, коли абонимент закінчився (Рис. 3.27).

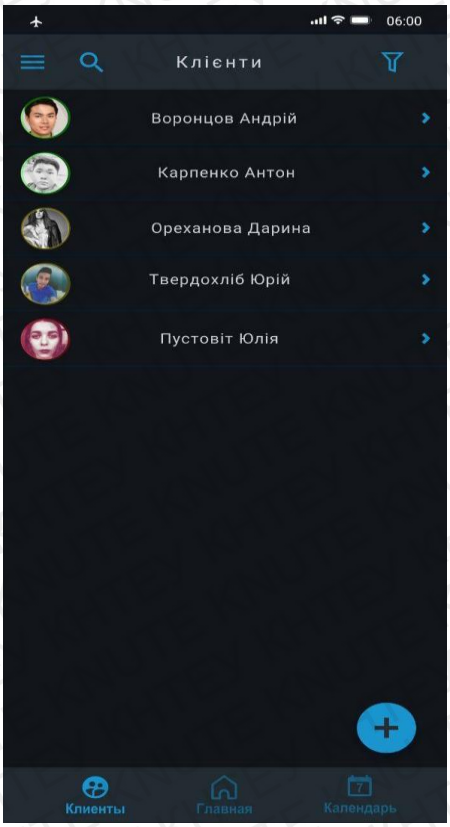

Рис . 3.27 Fragment Клієнти.

При натисканні на певного користувача відкривається основна інформація про нього (Рис. 3.28).

Рис. 3.29 Карточка клієнта.

Натиснемо далі на іконку замірів та откримаємо акутальну інформації про вагу та заміри (Рис.3.30).

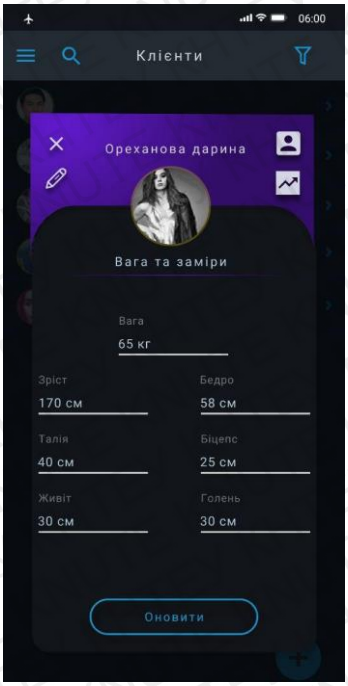

Рис. 3.30 Антропометричні дані.

Перейшовши по значку графік – побачимо візуальне відображення прогресу (Рис. 3.31).

Рис.3.31 Графік результатів клієнта.

Для створення нового клієнта необхідно повернутися на вкладку клієнти, та натиснути на кнопку з символом плюс. З’явиться діалог, в який потрібно ввести ім’я, прізвище, дату народження, емейл, вагу, вибраати тариф та ціль, завантажити фотографію клієнта та нажати створити (Рис. 3.33).

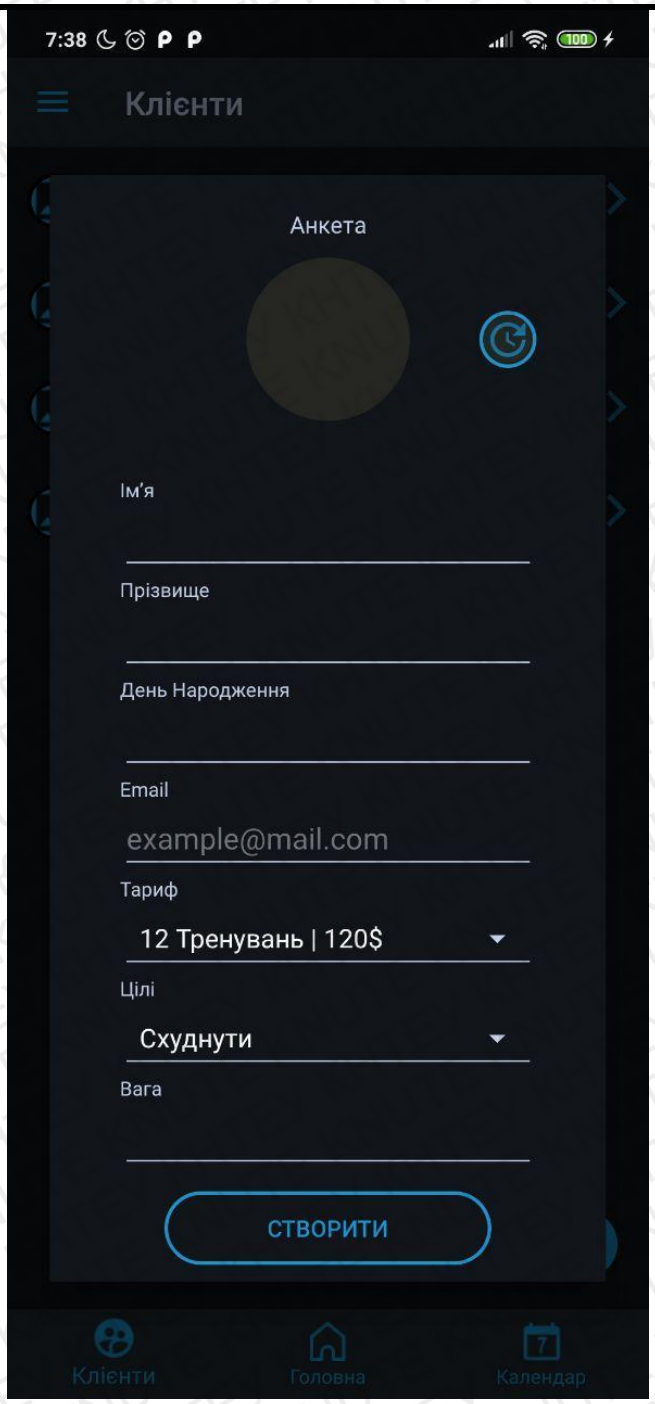

Рис.3.33 Діалог створення клієнта.

3.9 Висновок до розділу 3.

Отже, в даному розділі було описано створення програмного додатку з використанням Android studio на мові Kotlin. Описано створення основних класів, activity та fragment. Поетапно описано інструкцію використання програми для фітнес-студії.. Було зроблено:

1. Клас Clients.

2. Клас ClientAdapter.

3. Клас CanvasClass.

4. Activity Main.

5. Activity Registration.

6. Activity StartProgram.

7. Fragment Clients.

8. Fragment Today.

9. Fragment Calendar.

10. Fragment Tarifs

11. Fragment Goals.

12. Fragment Programs.

13. Fragment Reference.

14. Fragment Settings.

15. Layout ResetPasswordDialog

16. Layout DialogCreateClient.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Одже, було розроблено мобільний додаток для фітнес-студії «Body Sculptor» на операційну систему Android. Даний програмний продукт це мобільний журнал тренера. Це дозволить проаналізувати пророблену роботу, порівняти результати, зібравши всі необхідні дані в однмоу місці. З початку було розроблена архітектура програми з якої потім було розроблено базу даних, класи для воду та виводу інформації та інтерфейс програми.

Програма виконує:

1 Запис до бази даних користувачів, клієнтів.
2 Вивід інформації щодо тренувань.
3 Авторизацію за допомогою Firebase.
4 Редагування даних в базі даних.
5 Видалення з бази даних.

В першому розділу було описано технічне завдання у вимогах для створення програмного продукту. Існує лише декілька програм для фітнес-центрів та тренерів. Не всі програми доступні для певних верст населення. Не кожна людина, зможе користуватися програмою з більш складними функціями, та інтерфейсомю.

Також розписано вимоги до створення бази даних, бібліотеки керування БД. Розглянута ідея та вирішення проблеми.

У другому розділу описано класи в UML діаграмах та розглянута архітектура проекту. Третьому розділу було описано створення програмного додатку з використанням Android studio. Описано створення основних класів, activity та fragment. Поетапно описано інструкцію використання програми «Body Sculptor».

Було зроблено:

1. Клас Clients.
2. Клас ClientAdapter.
3. Клас CanvasClass.
4. Activity Main.
5. Activity Registration.
6. Activity StartProgram.
7. Fragment Clients.
8. Fragment Today.
9. Fragment Calendar.
10. Fragment Tarifs
11. Fragment Goals.
12. Fragment Programs.
13. Fragment Reference.
14. Fragment Settings.
15. Layout ResetPasswordDialog
16. Layout DialogCreateClient.

В програмі обліку особистих фінансів була задача зробити більш доступний інтерфейс та щоб вона виконувала всі необхідні функції.

При дослідженні всіх існуючих програм, була представлена більш зручна в користуванні інтерфейсу, функцій програми для широкого використані.

Пропозиції для програми, це додати функції чатування з клієнтами, щоденник харчування, створення десктопної версії

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Berglund, Tim; McCullough, Matthew (July 2011). Building and Testing with Gradle. Foreword by Hans Dockter (First ed.). O'Reilly Media. p. 116. IP ПРОГРАМНИХ ЗАСОБІВ ТА АРХІТЕКТУРИSBN 978-1-4493-0463-8.

2. IP ПРОГРАМНИХ ЗАСОБІВ ТА АРХІТЕКТУРИkkink, Hubert (November 2012). Gradle Effective IP ПРОГРАМНИХ ЗАСОБІВ ТА АРХІТЕКТУРИmplementation Guide (First ed.). Packt Publishing. p. 382. IP ПРОГРАМНИХ ЗАСОБІВ ТА АРХІТЕКТУРИSBN 978-1849518109.

3. Berglund, Tim; McCullough, Matthew (May 2013). Gradle DSLs (First ed.). O'Reilly Media. pp. 50 est. IP ПРОГРАМНИХ ЗАСОБІВ ТА АРХІТЕКТУРИSBN 978-1-4493-0467-6.

4. Muschko, Benjamin (Fall 2013). Gradle IP ПРОГРАМНИХ ЗАСОБІВ ТА АРХІТЕКТУРИn Action (First ed.). Manning Publications. p. 390. IP ПРОГРАМНИХ ЗАСОБІВ ТА АРХІТЕКТУРИSBN 9781617291302.

5. Галина Острякова, Хто поклав грощі в тумбу, Острякова Г. Ф., Київ. 11. Effective java Thrid Edition, Joshua Bloch/ М.: Лори, 2002. — 224 с

6. Java Puzzlers: Traps, Pitfalls, and Corner Cases, ISBN 0-321-33678-X, 200513. Date C. J. Introduction to Database Systems = Introduction to Database Systems. - 8th ed. - M.: Williams, 2005 .-- 1328 p.

7. Kuznetsov S. D. Fundamentals of databases. - 2nd ed. - M .: Internet University of Information Technologies; Laboratory of Knowledge, 2007. - 484 p.

8. Kogalovsky M.R. Encyclopedia of database technologies. - M.: Finance and Statistics, 2002. - 800 p.

9. Garcia-Molina G., Ulman J., Widom J. Database systems. Full course. - M.: Williams, 2003 .-- 1088 p.

10. Kogalovsky M. R. Encyclopedia of database technologies. - M.: Finance and Statistics, 2002. - 800 p.

11. Tsikritzis D., Lochowski F. Data models - D. Tsichritzis, F. Lochovsky. Data Models. Prentice Hall, 1982. - 344 p.

12. Ben Fort. Learn your own SQL query language / Per. from English - 3rd ed. - M .: Dialectics, 2005 .-- 288 p.

13. Paul Wilton, John Colby. SQL query language for beginners / Per. from English - M .: Dialectics, 2005 .-- 496 p.

14. K.J. Date. Introduction to database systems / Per. from English - 8th ed. - M.: Williams, 2005 .-- 1328 p.

15. Kevin Kline. SQL Directory. - M .: Kudits-Obraz, 2006 .-- 832 p.

16. Yuriy Bekarevich. Microsoft Access 2014. Self-proprietor of “The Doorway of the Tribute” / U. Bekarevich, N. Pushkina. 2014.–89 p.

0

ДОДАТКИ

Дотаток А

Лістинг класу MainAcivity.kt

```
package com.example.bodyscupltor

import android.content.Intent
import android.content.pm.PackageManager
import android.os.Bundle
import android.text.TextUtils
import android.util.Base64
import android.util.Log
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import com.facebook.AccessToken
import com.facebook.CallbackManager
import com.facebook.FacebookCallback
import com.facebook.FacebookException
import com.facebook.login.LoginManager
import com.facebook.login.LoginResult
import com.google.android.gms.auth.api.signin.GoogleSignIn
import com.google.android.gms.auth.api.signin.GoogleSignInAccount
import com.google.android.gms.auth.api.signin.GoogleSignInClient
import com.google.android.gms.auth.api.signin.GoogleSignInOptions
import com.google.android.gms.common.api.ApiException
import com.google.android.gms.tasks.Task
```

```
import com.google.firebase.auth.FacebookAuthProvider
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.GoogleAuthProvider
import kotlinx.android.synthetic.main.activity_main.*
import java.security.MessageDigest
import java.security.NoSuchAlgorithmException
import java.util.*
const val RC_SIGN_IN: Int = 1
lateinit var mGoogleSignInClient: GoogleSignInClient
lateinit var mGoogleSignInOptions: GoogleSignInOptions
lateinit var firebaseAuth: FirebaseAuth
lateinit var callbackManager: CallbackManager
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      configureGoogleSignIn()
      setupUI()
      firebaseAuth = FirebaseAuth.getInstance()
      callbackManager = CallbackManager.Factory.create()
      iv_fb.setOnClickListener {
         facebookLogin()
      }
      tv_create_acc.setOnClickListener {
         val intent = Intent(this, RegistrationActivity::class.java)
         startActivity(intent)
         finish()
      }
      b_login.setOnClickListener {
```

```
        loginUser()
    }
    forgotPassword.setOnClickListener {
        Toast.makeText(this, "Forgot press", Toast.LENGTH_LONG)
            .show()
        val dialog = AlertDialog.Builder(this)
        val dialogView = layoutInflater.inflate(R.layout.reset_password_dialog, null)
        dialog.setView(dialogView)
        val customDialog = dialog.create()
        customDialog.show()
        val resetButton = dialogView.findViewById<Button>(R.id.b_reset)
        val exitButton = dialogView.findViewById<Button>(R.id.b_exit)
        val emailInput = dialogView.findViewById<EditText>(R.id.et_email)
        resetButton.setOnClickListener {
            Toast.makeText(this, "Reset pressed ", Toast.LENGTH_LONG).show()
            firebaseAuth.sendPasswordResetEmail(emailInput.text.toString())
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        Toast.makeText(this, "Check Email",
Toast.LENGTH_LONG).show()
                    }
                    else {
            val message = task.exception!!.toString()
            Toast.makeText(this, "Error: $message", Toast.LENGTH_LONG).show()
        }
                    }
        }
        exitButton.setOnClickListener {
            customDialog.dismiss()
```

```
        }
    }
    printHash()
}

fun facebookLogin() {
    LoginManager.getInstance()
        .logInWithReadPermissions(this, Arrays.asList("public_profile", "email"))
    LoginManager.getInstance()
        .registerCallback(callbackManager, object : FacebookCallback<LoginResult>
{
            override fun onSuccess(result: LoginResult?) {
                //Second step
                handleFacebookAccessToken(result?.accessToken)
            }
            override fun onCancel() {
            }

            override fun onError(error: FacebookException?) {
            }
        })
}
private fun handleFacebookAccessToken(accessToken: AccessToken?) {
    val credential = FacebookAuthProvider.getCredential(accessToken!!.token)
    firebaseAuth.signInWithCredential(credential)
        .addOnFailureListener { e ->
            Toast.makeText(this, e.message, Toast.LENGTH_LONG)
                .show()
        }
```

```
            .addOnSuccessListener { result ->
                val email = result.user?.email
                Toast.makeText(this, "login success", Toast.LENGTH_LONG)
                    .show()
                val intent = Intent(this, StartProgramActivity::class.java)
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)
                startActivity(intent)
                finish()
            }
    }
    private fun setupUI() {
        imageView.setOnClickListener {
            signIn()
        }
    }
    private fun signIn() {
        val signInIntent: Intent = mGoogleSignInClient.signInIntent
        startActivityForResult(signInIntent, RC_SIGN_IN)
    }
    private fun configureGoogleSignIn() {
        mGoogleSignInOptions =
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(getString(R.string.default_web_client_id))
            .requestEmail()
            .build()
        mGoogleSignInClient = GoogleSignIn.getClient(this, mGoogleSignInOptions)
    }

    fun printHash() {
```

```
        try {
            val info = packageManager.getPackageInfo(
                "com.example.bodyscupltor",
                PackageManager.GET_SIGNATURES
            )
            for (signature in info.signatures) {
                val md = MessageDigest.getInstance("SHA")
                md.update(signature.toByteArray())
                Log.e("KeyHash:", Base64.encodeToString(md.digest(),
Base64.DEFAULT))
            }
        } catch (e: PackageManager.NameNotFoundException) {

        } catch (e: NoSuchAlgorithmException) {


        }
    }
    private fun loginUser() {
        val email = et_email.text.toString()
        val password = et_password.text.toString()

        when {
            TextUtils.isEmpty(email) -> Toast.makeText(
                this,
                "Email is Empty",
                Toast.LENGTH_LONG
            ).show()
            TextUtils.isEmpty(password) -> Toast.makeText(
                this,
```

```
            "Password is Empty",
            Toast.LENGTH_LONG
        ).show()

        else -> {
            val mAuth: FirebaseAuth = FirebaseAuth.getInstance()

            mAuth.signInWithEmailAndPassword(email, password)
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        checkVerificationEmail()

                    } else {
                        Toast.makeText(this, "error", Toast.LENGTH_LONG).show()
                    }
                }
        }

    }
}
private fun checkVerificationEmail() {
    val user = FirebaseAuth.getInstance().currentUser

    if (user != null) {
        if (user.isEmailVerified()) {
            val intent = Intent(this, StartProgramActivity::class.java)
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)
            startActivity(intent)
            finish()
```

```
0
        } else {
            Toast.makeText(this, "Email is not verified",
Toast.LENGTH_LONG).show()
        }

    }
  }
  private fun firebaseAuthWithGoogle(acct: GoogleSignInAccount) {
    val credential = GoogleAuthProvider.getCredential(acct.idToken, null)
    firebaseAuth.signInWithCredential(credential).addOnCompleteListener {
      if (it.isSuccessful) {
        val intent = Intent(this, StartProgramActivity::class.java)
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)
        startActivity(intent)
        finish()
      } else {
        Toast.makeText(this, "Google sign in failed:(",
Toast.LENGTH_LONG).show()
      }
    }
  }
  override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    callbackManager.onActivityResult(requestCode, resultCode, data)
    if (requestCode == RC_SIGN_IN) {
      val task: Task<GoogleSignInAccount> =
GoogleSignIn.getSignedInAccountFromIntent(data)
      try {
        val account = task.getResult(ApiException::class.java)
        if (account != null) {
```

```
                firebaseAuthWithGoogle(account)
            }
        } catch (e: ApiException) {
            Toast.makeText(this, "Google sign in failed:(",
Toast.LENGTH_LONG).show()
        }
    }
}
```

Додаток Б

Лістинг RegistrationActivity.kt

```
package com.example.bodyscupltor

import android.content.Intent
import android.os.Bundle
import android.text.TextUtils
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.google.android.gms.auth.api.signin.GoogleSignIn
import com.google.android.gms.auth.api.signin.GoogleSignInAccount
import com.google.android.gms.auth.api.signin.GoogleSignInOptions
import com.google.android.gms.common.api.ApiException
import com.google.android.gms.tasks.Task
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.GoogleAuthProvider
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import kotlinx.android.synthetic.main.activity_registration.*
class RegistrationActivity : AppCompatActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_registration)
        b_register.setOnClickListener() {
            createAccount()
        }
        configureGoogleSignIn()
        setupUI()
        firebaseAuth = FirebaseAuth.getInstance()
    }
    private fun setupUI() {
        imageView.setOnClickListener {
            signIn()
        }
    }
    private fun signIn() {
        val signInIntent: Intent = mGoogleSignInClient.signInIntent
        startActivityForResult(signInIntent, RC_SIGN_IN)
    }
    private fun configureGoogleSignIn() {
        mGoogleSignInOptions =
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(getString(R.string.default_web_client_id))
            .requestEmail()
            .build()
        mGoogleSignInClient = GoogleSignIn.getClient(this, mGoogleSignInOptions)
    }
    private fun firebaseAuthWithGoogle(acct: GoogleSignInAccount) {
        val credential = GoogleAuthProvider.getCredential(acct.idToken, null)
```

```
        firebaseAuth.signInWithCredential(credential).addOnCompleteListener {
            if (it.isSuccessful) {
                val intent = Intent(this, StartProgramActivity::class.java)
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)
                startActivity(intent)
                finish()
            } else {
                Toast.makeText(this, "Google sign in failed:(",
Toast.LENGTH_LONG).show()
            }
        }
    }
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == RC_SIGN_IN) {
            val task: Task<GoogleSignInAccount> =
GoogleSignIn.getSignedInAccountFromIntent(data)
            try {
                val account = task.getResult(ApiException::class.java)
                if (account != null) {
                    firebaseAuthWithGoogle(account)
                }
            } catch (e: ApiException) {
                Toast.makeText(this, "Google sign in failed:(",
Toast.LENGTH_LONG).show()
            }
        }
    }
    private fun createAccount() {
        val firstName = et_name.text.toString()
```

```
val secondName = et_second_name.text.toString()
val email = et_email.text.toString()
val password = et_password.text.toString()
val repPassword = et_repeat_password.text.toString()
when {
  TextUtils.isEmpty(firstName) -> Toast.makeText(
    this,
    "First Name is Empty",
    Toast.LENGTH_LONG
  ).show()
  TextUtils.isEmpty(secondName) -> Toast.makeText(
    this,
    "Second Name is Empty",
    Toast.LENGTH_LONG
  ).show()
  TextUtils.isEmpty(email) -> Toast.makeText(
    this,
    "Email is Empty",
    Toast.LENGTH_LONG
  ).show()
  TextUtils.isEmpty(password) -> Toast.makeText(
    this,
    "Password is Empty",
    Toast.LENGTH_LONG
  ).show()
  TextUtils.isEmpty(repPassword) -> Toast.makeText(
    this,
    "Repeat password",
    Toast.LENGTH_LONG
```

```
        ).show()
        repPassword != password ->
            Toast.makeText(this, "Passwords do not match",
Toast.LENGTH_LONG).show()
        else -> {
            val mAuth: FirebaseAuth = FirebaseAuth.getInstance()
            mAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        saveUserInfo(firstName, secondName, email)
                        sendEmailVerificatin()
                        Toast.makeText(this, "Account created, Virify Your Email",
Toast.LENGTH_LONG).show()
                        val intent = Intent(this, MainActivity::class.java)
                        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)
                        startActivity(intent)
                        finish()
                    } else {
                        val message = task.exception!!.toString()
                        mAuth.signOut()
                        Toast.makeText(this, "Error: $message",
Toast.LENGTH_LONG).show()
                    }
                }
        }
    }
  }
  private fun sendEmailVerificatin() {
    val user = FirebaseAuth.getInstance().currentUser
    user?.sendEmailVerification()
```

```
0

    }
    private fun saveUserInfo(firstName: String, secondName: String, email: String) {
        val currentUserID = FirebaseAuth.getInstance().currentUser!!.uid
        val usersRef: DatabaseReference =
FirebaseDatabase.getInstance().reference.child("Users")
        val userMap = HashMap<String, Any>()
        userMap["uid"] = currentUserID
        userMap["firstname"] = firstName
        userMap["secondname"] = secondName
        userMap["email"] = email
        usersRef.child(currentUserID).setValue(userMap)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                } else {
                    Toast.makeText(this, "error", Toast.LENGTH_LONG).show()
                }
            }
    }
}
```

Додаток В

Лістинг StartProgramActivity.kt

```
package com.example.bodyscupltor

import android.os.Bundle
import android.view.MenuItem
import android.view.View
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
```

```
import androidx.core.view.GravityCompat
import androidx.fragment.app.FragmentTransaction
import com.google.android.material.bottomnavigation.BottomNavigationView
import com.google.android.material.navigation.NavigationView
import kotlinx.android.synthetic.main.activity_start_program.*
import kotlinx.android.synthetic.main.toolbar.*
class StartProgramActivity : AppCompatActivity(),
NavigationView.OnNavigationItemSelectedListener {
    lateinit var ClientsFragment: ClientsFragment
    lateinit var TodayFragment: TodayFragment
    lateinit var calendarFragment: CalendarFragment
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_start_program)
        val bottomNavigationView: BottomNavigationView =
findViewById(R.id.bottom_nav_view)
        TodayFragment = TodayFragment()
        toolbar.title = getString(R.string.train_today)
        supportFragmentManager
            .beginTransaction()
            .replace(R.id.fl_4_fragment, TodayFragment)
            .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN)
            .commit()
        bottomNavigationView.setOnNavigationItemSelectedListener { item ->
            when (item.itemId) {
                R.id.clients_but -> {
                    ClientsFragment = ClientsFragment()
                    toolbar.title = getString(R.string.clients)
                    supportFragmentManager
```

```
                .beginTransaction()
                .replace(R.id.fl_4_fragment, ClientsFragment)
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN)
                .commit()
            }
            R.id.main_but -> {
                TodayFragment = TodayFragment()
                toolbar.title = getString(R.string.train_today)
                supportFragmentManager
                .beginTransaction()
                .replace(R.id.fl_4_fragment, TodayFragment)
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN)
                .commit()
            }
            R.id.calendar_but -> {
                calendarFragment = CalendarFragment()
                toolbar.title = getString(R.string.calendar_navigation)
                supportFragmentManager
                .beginTransaction()
                .replace(R.id.fl_4_fragment, calendarFragment)
                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN)
                .commit()
            }
        }
        true
    }
    val toolbar = findViewById<View>(R.id.toolbar) as Toolbar
    setSupportActionBar(toolbar)
    val toggle = ActionBarDrawerToggle(
```

```
        this, drawer_layout, toolbar, R.string.open, R.string.close
    )
    drawer_layout.addDrawerListener(toggle)
    toggle.syncState()
    nvView.setNavigationItemSelectedListener(this)
    displayScreen(-1)
}
override fun onBackPressed() {
    if (drawer_layout.isDrawerOpen(GravityCompat.START)) {
        drawer_layout.closeDrawer(GravityCompat.START)
    } else {
        super.onBackPressed()
    }
}
fun displayScreen(id: Int) {
    // val fragment =  when (id){
    when (id) {
        R.id.menu_tariffs -> {
            toolbar.title = getString(R.string.tariffs)
            supportFragmentManager.beginTransaction()
                .replace(R.id.fl_4_fragment, TartifFragment()).commit()
        }
        R.id.menu_programs -> {
            toolbar.title = getString(R.string.programs)
            supportFragmentManager.beginTransaction()
                .replace(R.id.fl_4_fragment, ProgramsFragment()).commit()
        }
        R.id.menu_goals -> {
            toolbar.title = getString(R.string.goals)
```

```
            supportFragmentManager.beginTransaction()
                .replace(R.id.fl_4_fragment, GoalsFragment()).commit()
        }
        R.id.menu_reference -> {
            toolbar.title = getString(R.string.reference)
            supportFragmentManager.beginTransaction()
                .replace(R.id.fl_4_fragment, ReferenceFragment()).commit()
        }
        R.id.menu_settings -> {
            toolbar.title = getString(R.string.settings)
            supportFragmentManager.beginTransaction()
                .replace(R.id.fl_4_fragment, SettingsFragment()).commit()
        }
    }
}
override fun onNavigationItemSelected(item: MenuItem): Boolean {
    // Handle navigation view item clicks here.

    displayScreen(item.itemId)

    drawer_layout.closeDrawer(GravityCompat.START)
    return true
}
}
```

Додаток Г

Лістинг ClientFragment.kt

```
package com.example.bodyscupltor

import android.app.Activity
import android.app.DatePickerDialog
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.Button
import android.widget.Spinner
import android.widget.TextView
import androidx.appcompat.app.AlertDialog
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.storage.FirebaseStorage
import de.hdodenhof.circleimageview.CircleImageView
import kotlinx.android.synthetic.main.fragment_clients.*
import java.util.*
```

```
class ClientsFragment : Fragment() {
    var selectedPhotoUri: Uri? = null
    lateinit var mRecyclerView : RecyclerView
    lateinit var mDatabase: DatabaseReference
    private val clients = listOf(
        simpleClientInfo("1987","Raising ","A rizona"),
        simpleClientInfo("19247","test1 ","A test"),
        simpleClientInfo("325235","test2 ","A testtte"),
        simpleClientInfo("565645","boss ","A rwefwegwe")
    )
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val v = inflater.inflate(R.layout.fragment_clients, container, false)
        retainInstance = true
        val btn = v.findViewById<View>(R.id.ib_add_clients)
        btn.setOnClickListener {
            showDialog()
        }
        return v
        }
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        lv_clients.apply{
            layoutManager = LinearLayoutManager(activity)
            adapter = ClientAdapter(clients)
        }
```

```
0

    }
    fun showDialog() {
        Log.d("Clients", "Press show Dialog")
        val dialog = AlertDialog.Builder(activity!!)
        val dialogView = layoutInflater.inflate(R.layout.dialog_create_client, null)
        dialog.setView(dialogView)
        val customDialog = dialog.create()
        customDialog.show()
        val createbtn = dialogView.findViewById<Button>(R.id.btn_create)
        val nameClient = dialogView.findViewById<TextView>(R.id.et_name)
        val secNameClient =
dialogView.findViewById<TextView>(R.id.et_second_name)
        val emailClient = dialogView.findViewById<TextView>(R.id.et_email)
        val date = dialogView.findViewById<TextView>(R.id.et_date_bday)
        val tarif = dialogView.findViewById<Spinner>(R.id.s_tarifs)
        val goal = dialogView.findViewById<Spinner>(R.id.s_goals)
        val weight = dialogView.findViewById<TextView>(R.id.et_weight)
        val imageUpload =
dialogView.findViewById<CircleImageView>(R.id.imageButton)
        val updatePhoto =
dialogView.findViewById<CircleImageView>(R.id.updatePhoto)
//Spinners
        val goals = resources.getStringArray(R.array.Goals)
        val tarifs = resources.getStringArray(R.array.Tarifs)
        val spinner = dialogView.findViewById<Spinner>(R.id.s_goals)
        if (spinner != null) {
            val adapter = ArrayAdapter(activity!!,
                android.R.layout.simple_spinner_item, goals)

adapter.setDropDownViewResource(android.R.layout.simple_dropdown_item_1line)
```

```
        spinner.adapter = adapter
      }
      val spinner2 = dialogView.findViewById<Spinner>(R.id.s_tarifs)
      if (spinner2 != null) {
        val adapter = ArrayAdapter(activity!!,
          android.R.layout.simple_spinner_item, tarifs)
  adapter.setDropDownViewResource(android.R.layout.simple_dropdown_item_1line)
        spinner2.adapter = adapter
      }
//
//Photo
      imageUpload.setOnClickListener {
        Log.d("Dialog_create_CLient", "Press imageButton")
        val intent = Intent(Intent.ACTION_PICK)
        intent.type ="image/*"
       startActivityForResult(intent,1)
      }
      updatePhoto.setOnClickListener {
        if (selectedPhotoUri == null) return@setOnClickListener
        imageUpload.setImageURI(selectedPhotoUri)
      }
      createbtn.setOnClickListener{
createClient(nameClient.text.toString().trim(),secNameClient.text.toString().trim(),dat
e.text.toString().trim(),emailClient.text.toString().trim(),tarif.selectedItem.toString(),g
oal.selectedItem.toString(),weight.text.toString().trim().toFloat())
        customDialog.dismiss()
      }
      date.setOnClickListener{
        // Calendar
```

```
        val c= Calendar.getInstance()
        val year= c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)
        var dpd = DatePickerDialog(activity!! ,DatePickerDialog.OnDateSetListener {
view, mYear,mMonth , mDay ->
            val mmMonth = mMonth+1
            val data = "$mDay/$mmMonth/$mYear"
            date.setText(data)
        },year,month,day)
        dpd.show()
    }
  }
  override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == 1 && resultCode  == Activity.RESULT_OK && data !=
null){
        Log.d("Dialog_create_CLient", "Photo was selected")
       selectedPhotoUri = data?.data!!
       // val bitmap = data?.extras?.get("data") as Bitmap
        //val bitmap =   MediaStore.Images.Media.getBitmap(contentResolver,
selectedPhotoUri)
    }
  }
  private fun uploadClinetImage (clientID : String?, currentUserID : String){
    if (selectedPhotoUri == null) return
    val filename = "$clientID||||$currentUserID"
    val ref = FirebaseStorage.getInstance().getReference("/images/$filename")
    ref.putFile(selectedPhotoUri!!)
        .addOnSuccessListener {
```

```
            Log.d("Create Client", "Upload image correct")
        }
        .addOnFailureListener {
            Log.d("Create CLient", "Upload image fail")
        }
    }
    fun createClient(fName: String, sName: String, date: String, email: String, tarif:
String = "", goal:String = "", weight: Float) {
        Log.d("Dialog_create_CLient", "Press create butt")
        val ref = FirebaseDatabase.getInstance().getReference("Clients")
        val currentUserID = FirebaseAuth.getInstance().currentUser!!.uid
        val clientID = ref.push().key
        val client =
            Client(clientID, currentUserID,
                "$clientID||||$currentUserID", fName, sName, date, email, tarif, goal,
weight)
        uploadClinetImage(clientID,currentUserID)
        if (clientID != null) {
            ref.child(clientID).setValue(client)
        }
    }
    class ClientsViewHolder(itemView: View?) :
RecyclerView.ViewHolder(itemView!!){
    }
}
```

0

Додаток Д

Лістинг Clients.kt

```
package com.example.bodyscupltor

import java.util.*

class Client(
    val cid: String?,
    val uid: String,
     val imid: String,
    val firstName: String,
    var secondName: String,
    val date: String,
    var email: String,
    var tarif: String = "",
    var goal: String = "",
    var weight: Float = 0.0f)
```

Додаток Е

Лістинг ClientAdapter.kt

```
package com.example.bodyscupltor

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
```

```
import androidx.recyclerview.widget.RecyclerView
import com.squareup.picasso.Picasso
class ClientAdapter(private var clients: List<simpleClientInfo>):
    RecyclerView.Adapter<clientViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
clientViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        return clientViewHolder(inflater, parent)
    }

    override fun onBindViewHolder(holder: clientViewHolder, position: Int) {
        val client: simpleClientInfo = clients[position]
        holder.bind(client)
    }
    override fun getItemCount(): Int = clients.size
}
```