Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка мобільного застосунку для персонального медичного консультування на платформі Android з використанням технологій Kotlin і Firebase

Виконав: студент 4 курсу
групи ЗПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Лейбак Д.В.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Снігур А.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ О.Н. Романюк
«10» 06 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 12 » березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Лейбаку Денісу Володимировичу

1. Тема роботи – Розробка мобільного застосунку для персонального медичного консультування на платформі Android з використанням технологій Kotlin і Firebase.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доц. каф ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: тип програми – мобільний застосунок; модель розробки ітеративна; засоби організаціїї даних програми – Firebase; вхідні дані: персональні дані реєстрації користувачів, протипоказання фізичної активності користувачів; вихідні дані: відображення тренувань користувачів, нотатки; середовище розробки – Android Studio; мова програмування – Kotlin; програмне забезпечення для симуляції мобільних пристроїв – Android Emulator.

4. Зміст текстової частини: обґрунтування вибору методу розробки та постановка задач дослідження, розробка структури та алгоритмів роботи програмного продукту, розробка програмного забезпечення, тестування програми; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження; порівняльний аналіз аналогів; новизна і практична цінність отриманих результатів; архітектура мобільного застосунку; блок-схема алгоритму створення рекомендацій; блок-схема алгоритму створення нотаток; екрани реєстрації та авторизації; домашній екран; екран «Workout»; екрани проведення тренувань; екрани створення нотаток; тестування мобільного застосунку; апробація та публікація матеріалів бакалаврської роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Катєльніков Д.І., к.т.н., доцент каф ПЗ	13.03.24	31.05.24

7. Дата видачі завдання _____13 березня 2024_____

КАЛЕНДАРНИЙ ПЛАН

№ п\п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Прим
1	Аналіз стану мобільних застосунків в галузі охорони здоров'я	13.03.24 – 22.03.24	Вик
2	Розробка структури мобільного застосунку	13.03.24 – 22.03.24	Вик
3	Розробка алгоритмів роботи програмного забезпечення	25.03.24 – 19.04.24	Вик
4	Розробка програмного забезпечення	22.04.24 – 10.05.24	Вик
5	Тестування програмного забезпечення	13.05.24 – 24.05.24	Вик
6	Оформлення матеріалів до захисту БДР	27.05.24 – 03.06.24	Вик

Студент _____ **Д. В. Лейба**
(підпис) (ініціали та прізви

Керівник бакалаврської дипломної роботи _____ **Д. І. Катєльніко**
(підпис) (ініціали та прізви

АНОТАЦІЯ

УДК 613:004.4

Лейбак Д.В. Розробка мобільного застосунку для персонального медичного консультування на платформі Android з використанням технологій Kotlin і Firebase : бакалаврська дипломна робота зі спеціальності 121 Інженерія програмного забезпечення. Вінниця : ВНТУ, 2024. 61 с.

На укр. мові. Бібліогр. : 23 назв ; рис. : 37; табл. 2.

У бакалаврській дипломній роботі було визначено, практичну цінність та мету розробки, яка полягає у вдосконалені процесу видання рекомендацій користувачу, щодо тренувань, залежно від вказаних протипоказань.

Розроблено алгоритми для проведення реєстрації та авторизації користувачів у системі для подальшого збереження даних у базі даних. Створено функціональний модуль, який призначений для персоналізації планів тренувань, в залежності від доданих протипоказань користувача. Додано можливість вести історію проведених тренувань або створювати власні записи.

Програмне забезпечення розроблено із використанням середовища розробки Android Studio та мови програмування Kotlin. Під час розробки було використано вбудовані бібліотеки для інтерфейсу користувача Material Components, для побудови навігації – Navigation Component, для зберігання даних – Firebase. У результаті виконання бакалаврської дипломної роботи розроблено програмний застосунок, працездатність і правильність роботи якого провірено, провівши тестування функціональних модулів, підготовлено інструкцію користувача.

Отримані в бакалаврській дипломній роботі результати можна використовувати для створення особистого плану тренувань фізичних вправ за допомогою мобільного застосунку.

Ключові слова: Android-застосунок, тренування, плани тренувань, рекомендації, ведення записів.

ABSTRACT

UDC 613:004.4

Leibak D.V. Development of a mobile application for personal medical consultation on the Android platform using Kotlin and Firebase technologies: bachelor's thesis on the specialty 121 Software Engineering. Vinnytsia: VNTU, 2024. 61 p.

In Ukrainian. Bibliography: 23 titles; Figures: 37; Table 2.

The bachelor's thesis determined the practical value and purpose of the development, which consists in improving the process of issuing recommendations to the user regarding training, depending on the indicated contraindications.

Algorithms have been developed for registration and authorization of users in the system for further saving of data in the database. A functional module has been created, which is designed to personalize training plans, depending on the added contraindications of the user. The ability to keep a history of training sessions or create your own records has been added.

The software is developed using the Android Studio development environment and the Kotlin programming language. During development, built-in libraries were used for the Material Components user interface, Navigation Component for building navigation, and Firebase for data storage. As a result of completing the bachelor's thesis, a software application was developed, the functionality and correctness of which was checked by testing the functional modules, and a user manual was prepared.

The results obtained in the bachelor thesis can be used to create a personal exercise training plan using a mobile application.

Keywords: Android application, training, training plans, recommendations, record keeping.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження.

З інформатизацією та поширенням мобільних технологій зростає потреба людей у доступній та зручній медичній допомозі. Такі застосунки можуть забезпечити користувачам доступ до інформації про стан здоров'я, лікарських рецептів, порад здорового способу життя, а також дозволити вести електронний журнал здоров'я та нагадувати про прийом ліків чи візити до лікаря. Проте, важливо враховувати виклики, пов'язані з розробкою та впровадженням таких застосунків, такі як забезпечення конфіденційності медичної інформації, точність наданих рекомендацій та доступність для різних категорій користувачів, включаючи людей похилого віку чи з обмеженими можливостями.

Кожен рік приносить значні зміни на ринку мобільних пристроїв. Із розвитком як апаратного, так і програмного забезпечення розробники мають можливість створювати програмні продукти, які спрощують повсякденне життя, оптимізують складні процеси та скорочують час на їх виконання. Це робить користування мобільним пристроєм зручним, не втрачаючи при цьому доступного функціоналу або можливостей персоналізації.

Мобільні застосунки, що спрямовані на підтримку здоров'я, є наявними сьогодні, але вони мають ряд недоліків. Деякі з них не відповідають усім потребам користувачів або роблять це тільки частково. Інші мають необхідні функції, але не завжди надають точну інформацію або не мають інтегрованої системи персоналізації. Тому розробка власних мобільних додатків для здоров'я залишається актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження.

Метою бакалаврської дипломної роботи є підвищення доступності інформації про дотримання здорового образу життя і точності порад щодо тренувань, залежно від вказаних показань та протипоказань, шляхом розробки мобільного застосунку для персонального медичного консультування на платформі Android з використанням технологій Kotlin і Firebase.

Основними задачами роботи є:

- здійснити аналіз стану питання та методів розв’язання задачі;
- розробити архітектуру та загальний алгоритм роботи системи;
- розробка модуля графічного інтерфейсу, призначеного для відображення контенту та його управлінням;
- розробка модуля для зберігання користувацьких даних у базу даних;
- розробка модуля для створення власних записів та їх управлінням;
- розробка модуля для відображення та використання планів тренувань;
- розробка модуля відбору планів тренувань, залежно від вказаних протипоказань;
- проведення тестування системи;
- розробити інструкцію користувача.

Об’єкт дослідження – процес розробки програмних засобів мобільного застосунку в галузі охорони здоров’я.

Предмет дослідження – програмні засоби та алгоритми для реалізації мобільного застосунку персонального консультанта в галузі охорони здоров’я.

Методи дослідження.

Протягом дослідження було використано: теорія людино-машинної взаємодії для побудови інтерфейсу, теорія баз даних для організації зберігання інформації, методи об’єктно-орієнтованого програмування для розробки архітектури класів та модулів програмного забезпечення.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод композиції планів тренувань, у якому на відміну від існуючих методів композиції, враховуються не тільки показання і цільові установки, але й протипоказання, що дозволило обирати більш безпечні плани тренувань.

2. Подальшого розвитку отримав метод протоколювання тренувань, у якому на відміну від існуючих методів протоколювання, застосовується детальна фіксація кількості підходів кожної окремої вправи та кількості її повторень, що дозволяє більш точно слідкувати за навантаженнями і приймати більш виважені рішення щодо зміни плану тренувань.

Практична цінність отриманих результатів.

Практична цінність отриманих результатів полягає в розробці програмних засобів мобільного застосунку, який надає користувачам покращити можливості використання планів тренувань, вести їх історію, створювати нотатки або нагадування.

Апробація матеріалів бакалаврської дипломної роботи.

Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на XXIV Всеукраїнській науково-технічній конференції «СТАН, ДОСЯГНЕННЯ ТА ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ» [1].

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану мобільних застосунків в галузі охорони здоров’я

Швидке впровадження інтелектуальних пристроїв в останнє десятиліття значною мірою сприяло перспективі використання мобільних технологій для оздоровлення. Також з’явився термін «мобільна охорона здоров’я», який був визначений ВООЗ як: «практика медичного та громадського здоров’я, що підтримується мобільними пристроями, зокрема мобільними телефонами, пристроями спостереження за пацієнтами, персональними цифровими помічниками та іншими бездротовими пристроями».

Мобільні застосунки пропонують нові методи для постійного моніторингу біологічних, поведінкових чи екологічних даних, показників здоров’я та тенденцій, пов’язаних із поведінкою людини. Мобільні застосунки можуть допомогти змінити ставлення та поведінку, розподіляючи, збираючи, обробляючи та інтерпретуючи інформацію, пов’язану зі здоров’ям, та дозволяючи різнотипові впливи. Тому різні цілі можуть бути досягнуті за допомогою мобільних застосунків, орієнтованих на широкий спектр груп користувачів. Можна розробити застосунки, орієнтовані на медичних працівників, одержувачів медичної допомоги та широку громадськість.

Мобільні пристрої та застосунки використовуються для швидкої доставки клінічної інформації медичним працівникам для оснащення сільського медичного персоналу актуальною інформацією як у розвинених, так і в слаборозвинених країнах .

Необхідність розуміння та вирішення проблем кінцевих користувачів є критично важливою для забезпечення широкого використання мобільних програм в галузі охорони здоров’я. Сприйняття функціональності, продуктивності, надійності, простоти використання, наприклад, інтерфейсу, часу для вивчення тощо, а також проблеми конфіденційності, впливають на

використання мобільних застосунків як в галузі охорони здоров’я так і в цілому.

Отже, розробка програмних модулів для створення мобільного застосунку у галузі охорони здоров’я є критично важливим завданням у сучасному цифровому світі. Використовуючи досвід спеціалізованих компаній з розробки програмного забезпечення, компанії можуть створювати інноваційні рішення, які покращують користувацький досвід та надають конкурентні переваги.

1.2 Порівняльний аналіз аналогів

Незважаючи на велику кількість програм для охорони здоров’я, що рекламуються на ринках застосунків, наприклад, Google Play Store, широке впровадження мобільних застосунків для охорони здоров’я та електронного здоров’я все ще очікується. Розуміння категорій, на які орієнтовані популярні застосунки, та відповідних функцій, що надаються користувачам, призводить до кращої оцінки користувачів та зростання кількості завантажень. Тому дослідження мобільних застосунків у галузі охорони здоров’я є актуальним напрямком сучасних досліджень.

До найбільш відомих рішень відносять:

- AdaHealth;
- MyFitnessPal;
- Fitbit.

Застосунок Ada пропонує користувачам можливість отримати персоналізовану медичну консультацію та діагноз щодо їх симптомів та стану здоров'я (див. рисунок 1.1) [2]. Ada використовує штучний інтелект та алгоритми машинного навчання для аналізу інформації, яку користувачі надають про свої симптоми, і надає рекомендації щодо можливих причин та дій для полегшення стану здоров'я. Крім того, додаток може надавати користувачам інформацію про захворювання, медичні процедури та лікування,

що допомагає їм краще зрозуміти їх стан здоров'я та взаємодіяти з медичними фахівцями.

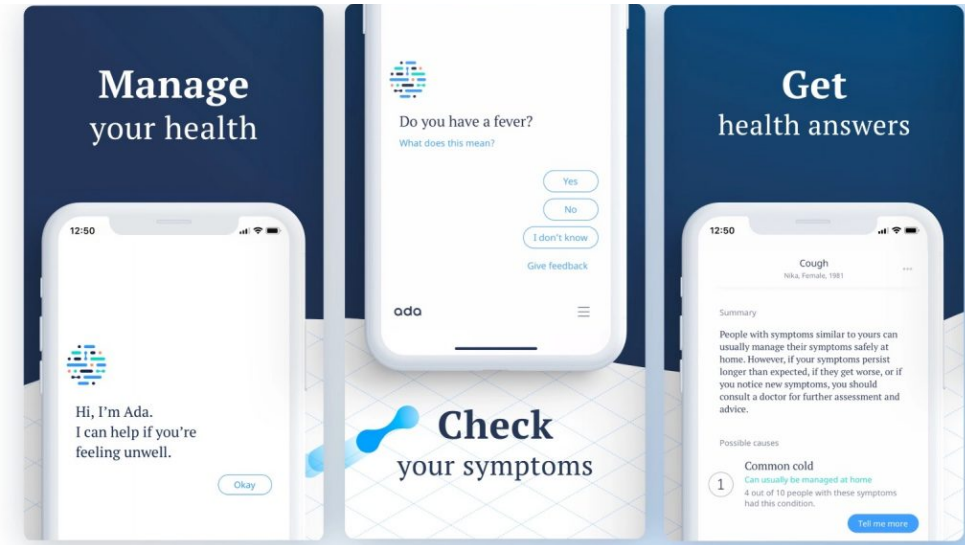


Рисунок 1.1 – Зображення мобільного застосунку «Ada Health»

MyFitnessPal – це мобільний додаток та веб-сервіс, який допомагає користувачам вести здоровий спосіб життя, контролювати харчування та ведення активного способу життя [3]. Додаток став популярним інструментом для здорового харчування та фітнесу. MyFitnessPal має активну спільноту користувачів, де можна отримати поради, підтримку та мотивацію від інших людей, які діляться своїм досвідом у здоровому способі життя (див. рисунок 1.2).

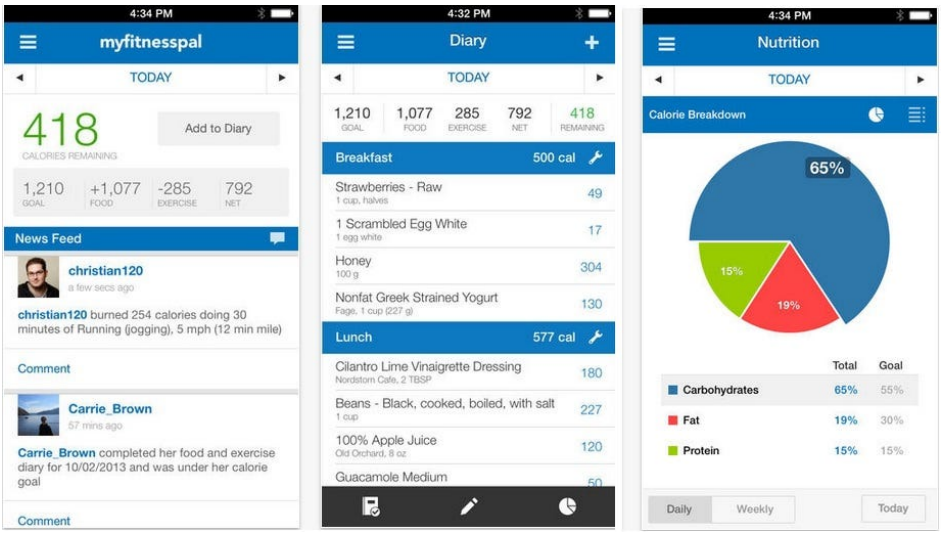


Рисунок 1.2 – Зображення мобільного застосунку «MyFitnessPal»

Fitbit App – це мобільний додаток, який дозволяє користувачам синхронізувати дані з їхніх пристроїв Fitbit (таких як смарт-годинники, фітнес-браслети) зі смартфонами або планшетами. Цей застлсунок дозволяє користувачам відстежувати різноманітні показники, такі як кількість кроків, відстань, кількість спожитих калорій, час активності (див. рисунок 1.3) [4]. Крім того, Fitbit App має додаткові функції, такі як створення цілей фізичної активності, моніторинг харчування, можливість конкурувати з друзями або членами родини у фітнес-викликах, а також отримання персоналізованих рекомендацій щодо покращення здоров'я та фізичної форми.

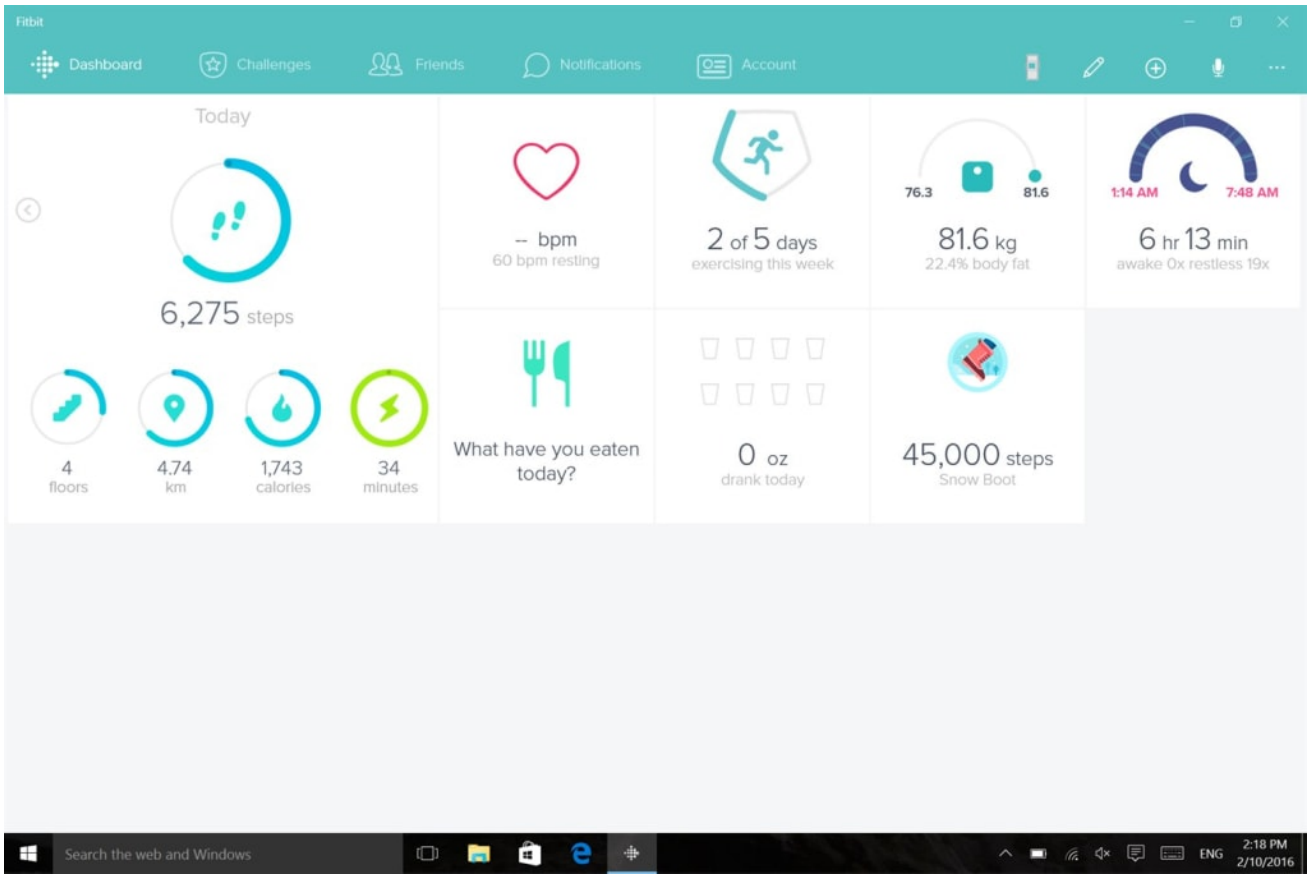


Рисунок 1.3 – Зображення застосунку «Fitbit»

Результати порівняння аналогів наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

	AdaHealth	MyFitnessPal	Fitbit	Власна розробка
Персоналізовані консультації вправ	-	-	+	+
Доступ до даних через віджети робочого столу	-	-	+	+
Конфіденційність та безпека	+	-	-	+
Ведення історії	-	+	+	+
Створення нотаток	-	+	+	+
	2	2	3	5

Після порівняння власіного програмного продукту з іншими мобільними застосунками у галузі охорони здоров'я можна зробити наступні висновки:

Власна розробка є кращим рішенням у цій галузі завдяки своїм різноманітним функціям та користувацькому досвіду. Він дозволяє користувачам моніторити широкий спектр показників здоров'я, включаючи фізичну активність, харчування та загальний стан здоров'я. Крім того, додаток має інтуїтивний і легкий у використанні інтерфейс, який робить процес відстеження та аналізу даних доступним і зручним для широкого кола користувачів.

1.3 Аналіз методів розв’язання задач

Під час розробки мобільних застосунків існує кілька методів розв'язання задачі, які можуть варіюватися в залежності від конкретних потреб проекту, вимог замовника та власних вподобань розробників. Ось кілька основних методів, які можуть бути використані:

Нативна розробка мобільних програм. На даний момент це розробка під дві основні платформи Android та iOS [5]. Кожна з платформ має свою специфіку, але загалом має схожий функціонал. Написання програм під ці платформи дозволяє забезпечити найкращий UX, що відповідає актуальним гайдлайнам кожної з платформ, але писати код програми доведеться двічі – під Android та під iOS.

Однією з основних переваг кросплатформної розробки додатків є можливість охоплення ширшої аудиторії [6]. Розробляючи єдину програму, яка може працювати на кількох платформах, підприємства можуть націлюватися на користувачів на різних пристроях, зокрема iOS, Android і Windows. Це розширення охоплення ринку може значно підвищити видимість програми та потенціал залучення користувачів.

Гібридна розробка – це тип розробки програмного забезпечення, який поєднує в собі нативні та веб-технології [7]. Цей підхід дозволяє розробникам створювати додатки з таким же зовнішнім виглядом, відчуттям і продуктивністю, як традиційні нативні додатки, використовуючи переваги економії коштів і швидкості веб-розробки.

Нативна розробка є найкращим рішенням для розробки мобільних застосунків, оскільки дозволяє отримати повний контроль над функціональністю та виглядом додатку на кожній платформі, забезпечуючи найвищу продуктивність, доступ до всіх функцій пристрою та максимальну інтеграцію з екосистемою платформи. Використання мов програмування та інструментів, специфічних для кожної платформи, дозволяє розробникам створювати оптимізовані та швидкі додатки, які найкращим чином відповідають вимогам користувачів і бізнесу. Тому було прийнято рішення використовувати нативну розробку для мобільного застосунку.

1.4 Постановка задач

Проаналізувавши переваги та недоліки існуючих мобільних застосунків у галузі охорони здоров’я, було визначено наступні завдання, які необхідно виконати:

- розробка модуля графічного інтерфейсу, призначеного для відображення контенту та його управлінням;
- розробка модуля для зберігання користувацьких даних;
- розробка модуля для створення власних записів та їх управлінням;
- розробка модуля для відображення та використання планів тренувань;
- інтеграція розроблених модулів у єдиний засіб мобільного застосунку.

1.5 Висновки

У першому розділі було розглянуто стан мобільних застосунків у галузі охорони здоров’я. Було проведено порівняння відомих застосунків у цій сфері, таких як: Fitbit, AdaHealth та MyFitnessPal.

У результаті дослідження було виявлено, що існуючі застосунки набули значної популярності серед учасників ринку завдяки своїй зручності та доступності. Проаналізувавши їх переваги та недоліки, було з’ясовано, що вони не надають усіх необхідних можливостей. Таким чином було прийнято рішення щодо розробки власного мобільного застосунку. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власних програмних засобів.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

Для реалізації програмних засобів мобільного застосунку персонального консультанта в галузі охорони здоров’я будуть використані мова програмування Kotlin та інтегроване середовище розробки Android Studio. Kotlin є сучасною мовою програмування, яка прийшла на заміну традиційній Java, для розробки мобільних додатків на платформі Android. В свою чергу, Android Studio створене спеціально для розробки програмного забезпечення для платформи Android. Це основний інструмент для розробників Android, розроблений компанією Google.

Розробка програмних засобів мобільного застосунку персонального консультанта в галузі охорони здоров’я передбачає декілька послідовних етапів.

Початковим етапом створення програмного засобу є правильне написання інтерфейсу. Інтерфейс – це перше, що бачить користувач, після застосунку, тому слід приділити увагу, щоб інтерфейс буд доволі простим, однак був лаконічним і зрозумілм для користувача, зберігав його основний функціонал. Наступним кроком є створення графу навігації. Це елемент, завдяки якому застосунок будує схеми і переходи між екранами, або налаштовує правильний порядок використання фрагментів інтерфейсу користувача, під час зміни контенту після переходів.

Наступним етапом є розробка модуля для реєстрації та авторизації. Цей модуль буде використовувати зв’язок із базою даних для забезпечення безпеки та конфіденційності користувачів, забезпечуючи доступ до особистих даних тільки тим, кому вони належать, і обмежуючи функції додатка відповідно до повноважень користувача.

Протяг третього етапу буде розроблено модуль, який дозволить користувачам вести нотатки, щодо тренувань або зауважень щодо здоров’я.

Створені записи будуть зберігатися у базі даних та синхронізуватися між пристроями, зберігаючи їх конфіденційність.

Завершальним етапом розробки буде створення планів тренувань різних видів та категорій. Таким чином, користувачі можуть обирати ті вправи, які найбільше їм підходять або до вподоби. Додатково, буде створюватися історія проведення тренувань.

Отже, розробка програмних засобів мобільного застосунку персонального консультанта в галузі охорони здоров'я, використовуючи Kotlin та Android Studio є завданням комплексним, яке включає декілька послідовних етапів розробки. Для досягнення поставлених цілей потрібно уважно спроектувати та ретельно протестувати програмні модулі для того, щоб гарантувати безперебійну працездатність програмного продукту та досягти максимальної ефективності.

2.2 Розробка структури мобільного застосунку

Розробка структури мобільного застосунку - це процес проектування та організації всіх складових частин програми для мобільних пристроїв. Це охоплює визначення функціональних можливостей, взаємодію з користувачем, архітектуру даних, перехід між екранами, інтеграцію з зовнішніми сервісами, безпеку та інші аспекти. Структура мобільного застосунку має бути добре продуманою і зручною для користувача, а також ефективною з точки зору ресурсів пристрою та мережі.

Одним із важливих етапів розробки мобільного застосунку є вибір його архітектури [8]. Архітектура MVVM (Model-View-ViewModel) стала досить популярною для розробки Android-додатків через свою здатність до відокремлення логіки бізнес-логіки від інтерфейсу користувача [9]. Вона складається із таких компонентів:

- Модель (Model). Це представлення даних вашого додатку. Вона відповідає за роботу з даними, такими як збереження, оновлення та видалення.

Ця частина не залежить від жодного іншого компонента і може використовуватися в різних частинах мобільного застосунку.

- Вид (View). Це інтерфейс користувача розроблюваного мобільного застосунку. Він відображає дані, представлені моделлю, та реагує на дії користувача, такі як натискання кнопок чи введення тексту.
- ViewModel. ViewModel відповідає за збереження і обробку даних, які відображаються на екрані. Вона слугує посередником між моделлю та видом. ViewModel може містити логіку, яка відповідає за обробку введених даних, перетворення їх у формат, зрозумілий моделі, та відображення результуючих даних в інтерфейсі користувача.

Переваги використання MVVM на Android включають полегшення тестування, зменшення залежності між компонентами, покращення читабельності та підтримку асинхронного програмування.

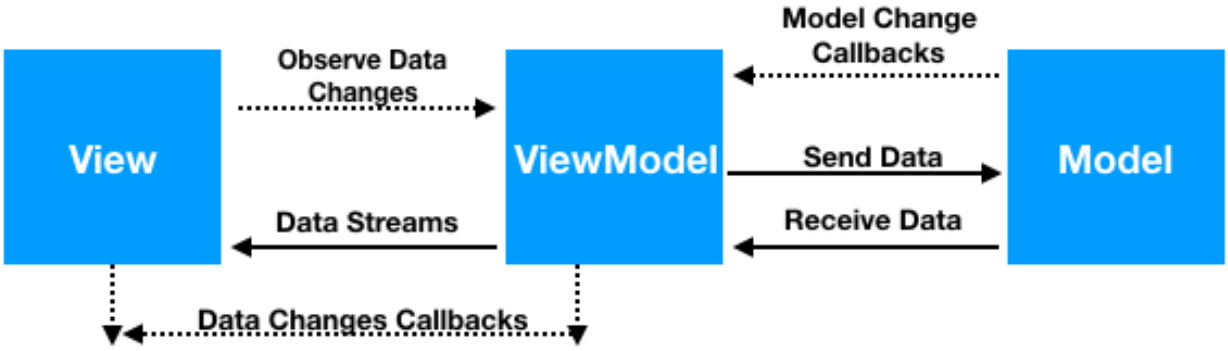


Рисунок 2.1 – Схема архітектури MVVM

2.3 Розробка загального алгоритму роботи програми

При створенні складної програмної системи, критично важливо мати чітке уявлення про її повну функціональність та взаємодію компонентів. Це означає ретельне визначення робочих процесів, потоку даних і взаємодії між різними частинами програми. Один із ефективних методів досягнення цієї мети

полягає у створенні загального алгоритму, який відображає загальний принцип роботи системи та визначає послідовність дій.

Вивчення загального алгоритму може сприяти глибокому розумінню функціональності застосунку, а також виявленню можливих проблем та вузьких місць, які можуть виникнути під час розробки. Це дозволяє оптимізувати дизайн програми, підвищити її ефективність та результативність в майбутньому.

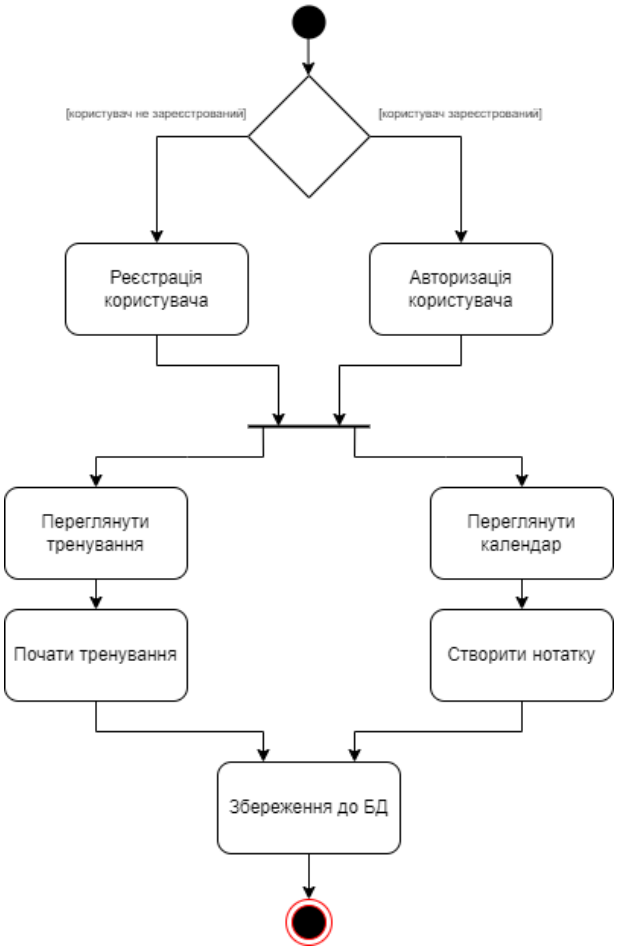


Рисунок 2.2 – UML-діаграма роботи мобільної системи

Діаграма діяльності дозволяє продемонструвати загальну роботу застосунку, при цьому не вдаючись до деталізації робочих процесів, для того щоб полегшити сприйняття та розуміння.

2.4 Розробка методу протоколювання тренувань

Однією із ключових функціональних можливостей мобільного застосунку є проведення тренувань з його персоналізацією. Для кращого розуміння роботи модуля, який відповідає за цю можливість, потрібно розробити алгоритм, який визначає його функціональні можливості. Цей алгоритм зображено на рисунку 2.3.

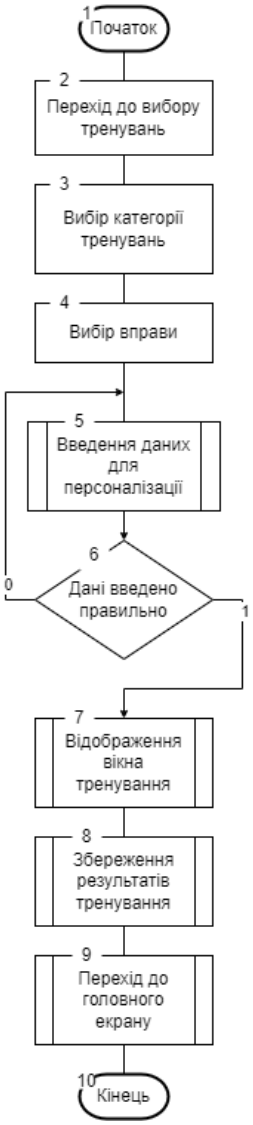


Рисунок 2.3 – Екран проведення тренувань

Згідно цього алгоритму, спочатку користувач взаємодіє із інтерфейсом мобільного застосунку для того, щоб перейти на екран тренувань. Після цього слід обрати категорію тренувань, а потім із списку вправ, обрати певну вправу.

На екрану мобільного пристрою з’являється вікно персоналізації тренування, в якому користувачеві потрібно ввести кількість підходів вправи та кількість її повторень. Після натискання кнопки «Продовжити», користувачеві відображається вікно, яке демонструє анімацію правильного виконання вибраної вправи та дає можливість стежити за часом її виконання. Після завершення вправи, введені дані для персоналізації та витрачений час зберігаються до історії виконання вправ, а користувача перекидає до головного екрану мобільного застосунку

2.5 Висновки

У розділі було проаналізовано вхідні та вихідні дані мобільного застосунку для персонального медичного консультування, розроблено архітектуру системи. Також було розроблено загальний алгоритм роботи мобільної системи та розроблено алгоритм обробки дій користувача та відображення планів тренувань та їх проведення,

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Інтегровані середовища розробки (IDE) - це комплексні програмні системи, які надають усі необхідні інструменти для створення, тестування та розгортання програмного забезпечення [10]. У контексті розробки додатків для Android доступні декілька IDE, які можна використовувати.

Android Studio [11], яке є офіційним середовищем розробки для Android, розроблене компанією Google. Це базується на IntelliJ IDEA і надає широкий набір інструментів для створення додатків для Android. У свою чергу, Android Studio підтримує мови програмування Kotlin, C++ та Java.

Eclipse - це інтегроване середовище розробки (IDE), яке існує протягом значного часу [12]. Воно є популярним серед розробників на Java, а також надає підтримку для створення додатків для Android за допомогою плагіну Android Development Tools (ADT). Eclipse забезпечує розширену підтримку мов програмування Java та C++, має вбудований емулятор мобільних пристроїв для тестування додатків та надійний інструмент для налагодження коду.

IntelliJ IDEA, ще одне відоме середовище для розробки програм для мобільних пристроїв, має аналогічний набір функцій, що й Android Studio, оскільки воно розроблене тією ж компанією [13]. Воно підтримує Java, Kotlin та інші мови програмування і включає в себе вбудований емулятор для тестування додатків, а також підтримує автоматизацію збірки за допомогою Gradle [14].

Порівняльний аналіз на основі можливостей згаданих IDE наведено в таблиці 3.1.

Як видно з порівняльної таблиці, Android Studio видається кращим варіантом для запланованої розробки, порівняно з Eclipse та IntelliJ IDEA. По-перше, Android Studio офіційно підтримується Google, що гарантує регулярні оновлення та поліпшення. По-друге, вона використовує систему збірки на основі Gradle, яка є більш гнучкою та простою у використанні, ніж та, що

використовується в Eclipse. По-третє, в Android Studio вбудований емулятор, що дозволяє розробникам швидко та зручно тестувати свої програми. Отже, Android Studio видається оптимальним вибором для розробки трекера читання.

Таблиця 3.1 – Порівняння середовищ розробки

Особливість	**Android Studio**	**Eclipse**	**IntelliJ IDEA**
Офіційно підтримується Google	+	–	–
Система збірки на основі Gradle	+	–	+
Вбудований емулятор	+	+	+
Інтеграція з Git	+	+	+
Інструменти профілювання	+	–	+
Миттєваий запуск для тестування	+	–	+
Екосистема плагінів	+	+	+
Підсумок	7	3	6

Для створення мобільних додатків використовуються різні мови програмування. Кожна з цих мов має свої переваги та обмеження, і вибір конкретної залежить від потреб і цілей проекту.

Kotlin – це сучасна мова програмування, що працює на платформі JVM (Java Virtual Machine), яка була розроблена компанією JetBrains. Kotlin поєднує в собі функціональні та об'єктно-орієнтовані підходи до програмування, а також має безліч корисних функцій, які полегшують розробку [15]. Вона є однією з альтернатив Java, проте Kotlin можна використовувати не лише для розробки Android-додатків, а й для серверних застосувань, веб-розробки та багатьох інших областей. Kotlin має зрозумілу синтаксичну структуру та вбудовану підтримку корутин для асинхронного програмування, що робить її привабливим вибором для багатьох розробників.

Java була однією з перших мов програмування, яка використовувалася для розробки Android-додатків і залишається популярним вибором серед розробників [16]. Вона має велику спільноту розробників, величезну кількість доступної документації та різноманітні бібліотеки, що спрощують розробку. Java дозволяє створювати додатки для різних типів пристроїв Android та забезпечує високу стабільність і сумісність. Ця мова також підтримується Android Studio, офіційним інтегрованим середовищем розробки для Android, що сприяє зручності розробки та відладки. Однак, Java може мати більший обсяг коду порівняно з сучаснішими мовами, такими як Kotlin, і менше експресивності, що може призвести до більшої кількості рутинної роботи.

C++ може бути використаний для розробки Android-додатків завдяки Android NDK (Native Development Kit), який дозволяє розробникам писати частину коду на C++ та компілювати його в нативні бібліотеки, що можуть бути використані в Java або Kotlin застосунках [17]. Це особливо корисно для ситуацій, де необхідно досягти високої продуктивності, наприклад, у випадку ігрових додатків або додатків, які використовують складні алгоритми обробки даних. Використання C++ також дозволяє перевикористовувати існуючий код на C++ та здійснювати інтеграцію зі сторонніми бібліотеками, що розширює можливості розробки Android-додатків. Однак варто враховувати, що використання C++ може бути складнішим для розробки порівняно з Java або Kotlin, і вимагає ретельного контролю над пам'яттю та безпекою, щоб уникнути можливих проблем з пам'яттю та витоків.

Порівняльний аналіз описаних мов програмування наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння мов програмування

Особливість	**Java**	**Kotlin**	**C++**
Компільована мова	+	+	+
Збір сміття	+	+	–
Кросплатформеність	+	+	–

Продовження таблиці 3.2 – Порівняння мов програмування

Приведення типів	–	+	+
Функції розширення	–	+	–
Null-безпека	–	+	–
Перевантаження операторів	–	+	+
Контроль ресурсів пристрою	–	–	+
Підсумок	3	7	4

Для запланованої розробки, Kotlin виявляється більш перспективним вибором з декількох причин. По-перше, ця мова має кращу читабельність та більш компактний синтаксис порівняно з Java та C++, що сприяє швидшому та ефективнішому процесу розробки. По-друге, Kotlin надає кращу підтримку null-безпеки, що допомагає уникнути збоїв. По-третє, мова має розширену підтримку функціонального програмування, що робить код більш зручним у плані обслуговування.

3.2 Розробка модуля реєстрації та авторизації

Для початку розробки модуля реєстрації та авторизації користувачів, потрібно створити та ініціалізувати посилання для підключення мобільного застосунку до бази даних. Тому було створено клас FirebaseFirestoreService (див. рисунок 3.1).

Рисунок 3.1 – Діаграма класів підключення до бази даних

За допомогою методу createProfileDetail() мобільний застосунок спочатку зв’язується із віддаленою базою даних, встановлюючи перше підключення, та валідуючи користувача в системі, використовуючи параметр dataMap. Блок схема алгоритму цього методу показано на рисунку 3.2.

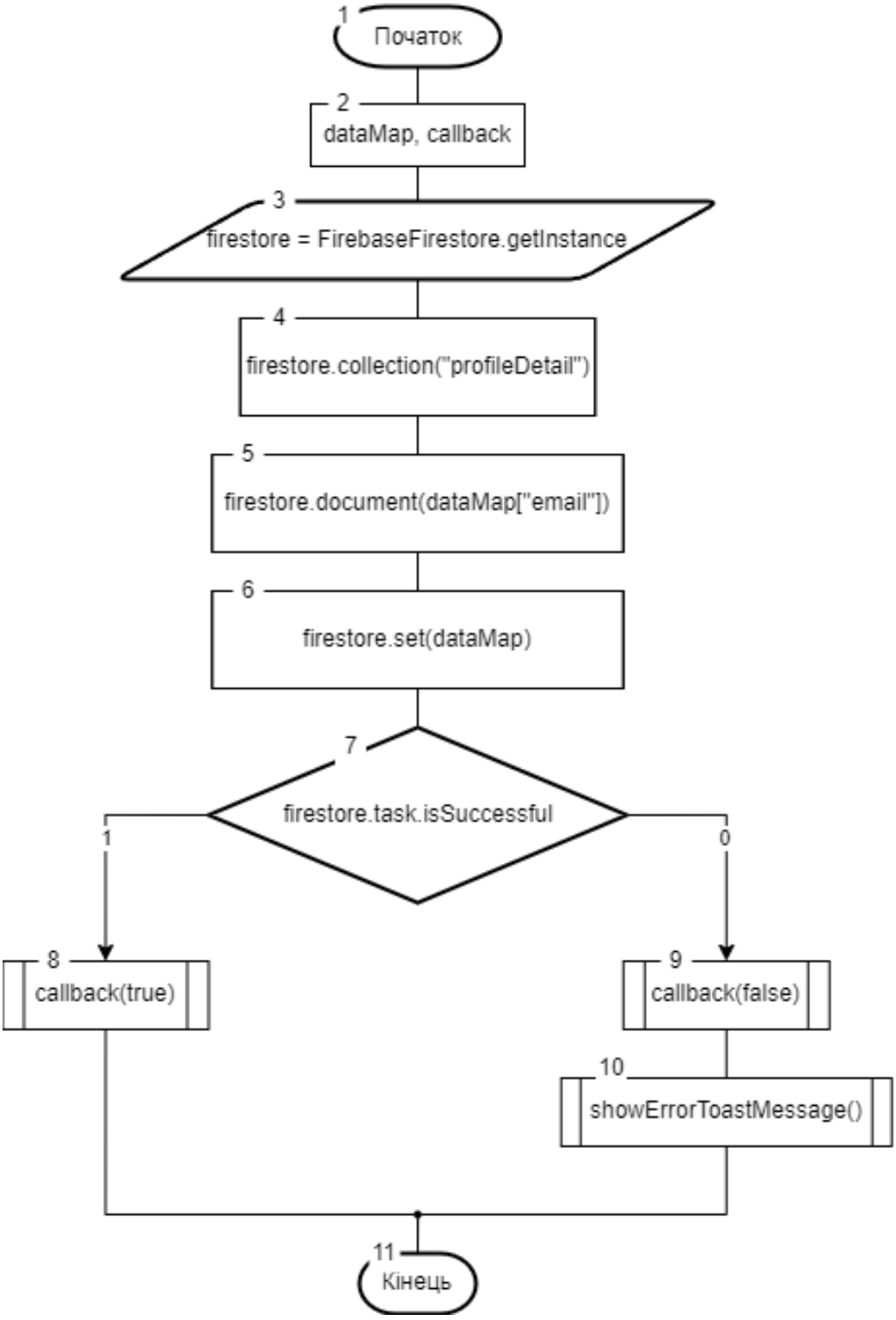


Рисунок 3.2 – Блок схема алгоритму методу createProfileDetail

1. Виклик методу createProfileDetail().
2. В якості аргументів методу передаються змінні dataMap і callback.
3. Створюється та ініціалізується змінна класу firestore, викликаючи вбудований метод getInstance() класу FirebaseFirestore.
4. За допомогою методу firestore.collection(“profileDetail”) витягується дані профілів із таблиці “Profiles” із бази даних.
5. Викликаючи метод firestore.document(dataMap[“email”]), отримуються усі записи елекронних пошт.
6. Метод firestore.set(dataMap) присвоює змінній firestore значення, яке містить усі дані про користувача, електронна пошта якого була сказана у змінній dataMap.
7. За допомогою вбудованої властивості task та її властивості isSuccessful перевіряється, чи отримані дані надійшли без помилок, повертаючи значення «true» або «false».
8. Якщо було повернено значення «true», виклається вбудований метод callback(), якому передається значення «true», щоб завершити з’єднання із базою даних і зберегти отримані дані у змінній firestore.
9. Якщо було повернено значення «false», callback(), якому передається значення «false», щоб завершити з’єднання із базою даних і не зберігати отримані дані у змінній firestore.
10. Викликається метод вбудований метод showErrorToastMessage(), який повідомляє, що сталася помилка і відображає сповіщення типу Toast, яке вбудоване у систему Android.
11. Вихід із методу createProfileDetail().

Однак, для відправки та збереження оновлених даних використовується метод updateProfileDetail(). Блок схема алгоритму цього методу показано на рисунку 3.3.

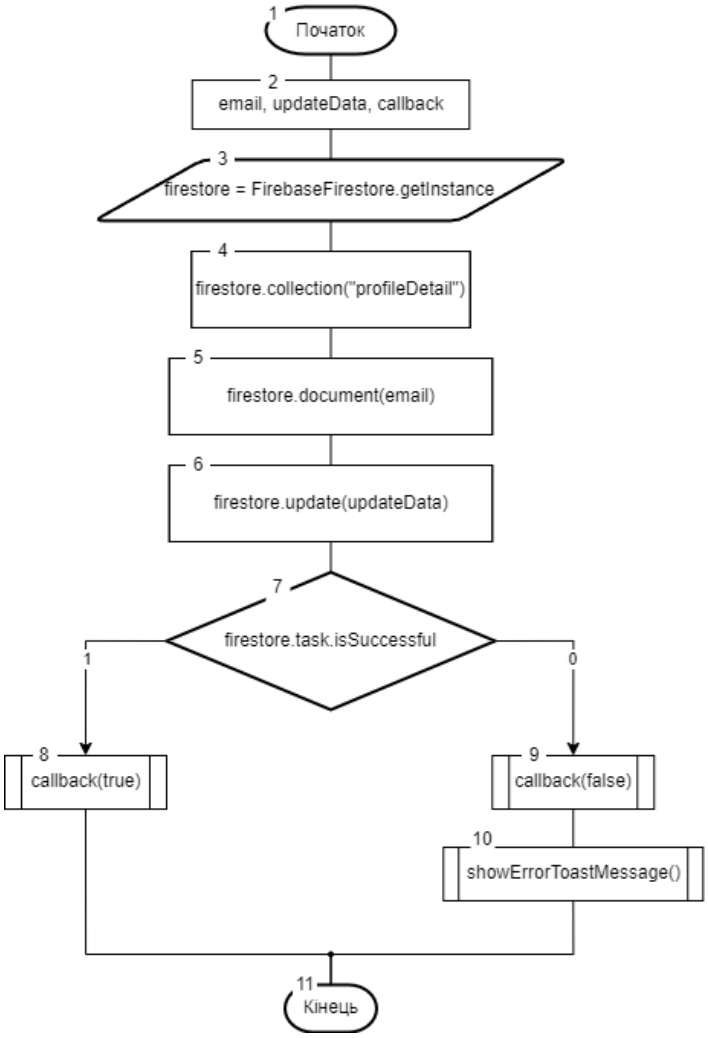


Рисунок 3.3 – Блок схема алгоритму методу updateProfileDetails

1. Виклик методу updateProfileDetail().

2. В якості аргументів методу передаються змінні email, що містить елекронну пошту користувача, updateData, що містить дані, які необхідно оновити і callback, змінна, яка буде повертати значення «true» або «false» для завершення операції з’єднання.

3. Створюється та ініціалізується змінна класу firestore, викликаючи вбудований метод getInstance() класу FirebaseFirestore.

4. За допомогою методу firestore.collection(“profileDetail”) витягується дані профілів із таблиці “Profiles” із бази даних.

5. Викликаючи метод firestore.document(email), отримуються усі записи користувача, із вказаною елекронною поштою.

6. Метод firestore.update(updateData) присвоює змінній firestore значення, яке містить усі оновлені дані про користувача, які були записані у змінну.

7. За допомогою вбудованої властивості task та її властивості isSuccessful перевіряється, чи отримані дані надійшли без помилок, повертаючи значення «true» або «false».

8. Якщо було повернено значення «true», виклається вбудований метод callback(), якому передається значення «true», щоб завершити з'єднання із базою даних і зберегти отримані дані у змінній firestore.

9. Якщо було повернено значення «false», callback(), якому передається значення «false», щоб завершити з'єднання із базою даних і не зберігати отримані дані у змінній firestore.

10. Викликається метод вбудований метод showErrorToastMessage(), який повідомляє, що сталася помилка і відображає сповіщення типу Toast, яке вбудоване у систему Android.

11. Вихід із методу createProfileDetail().

Використовуючи створений ініціалізатор бази даних, потрібно створити клас, який буде відповідати за авторизацію, тобто вхід, а також за реєстрацію – створення нового аккаунту. Для таких цілей було створено клас FirebaseAuthService (див. рисунок 3.4).

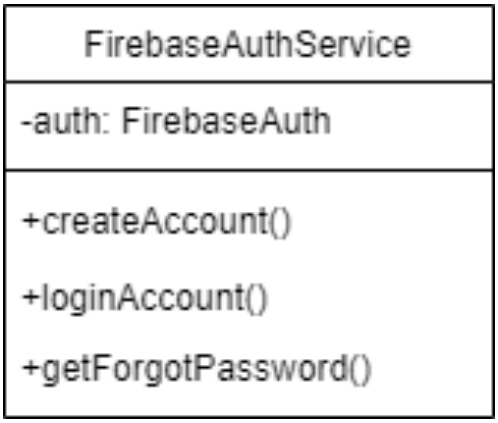


Рисунок 3.4 – Діаграма класів класу FirebaseAuthService

Для проведення авторизації користувача у мобільний застосунок використовується метод класу loginAccount(), який приймає аргументи електронну пошту та пароль, а потім з допомогою об’єкту auth звіряє введені дані з даними у базі даних, і якщо подібні записи є – повертає булеве значення true. Блок схема алгоритму цього методу показана на рисунку 3.5.

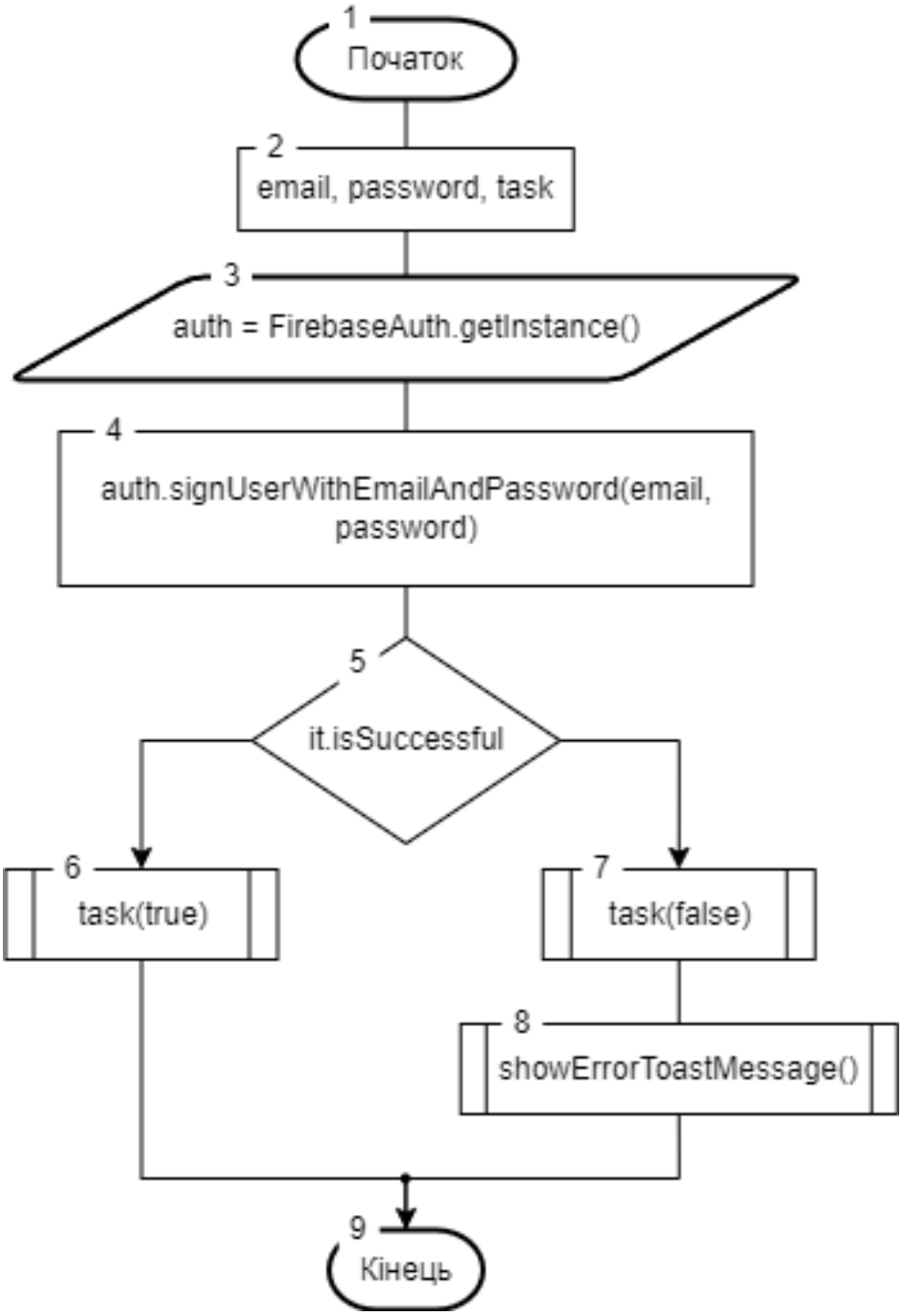


Рисунок 3.5 – Блок-схема алгоритму методу loginAccount

1. Виклик методу loginAccount().
2. В якості аргументів методу передаються змінні email, що містить елекронну пошту користувача, password, що містить пароль до аккаунту користувача і callback, змінна, яка буде повертати значення «true» або «false» для завершення операції аутентифікації.
3. Ініціалізується змінна класу auth, викликаючи вбудований метод getInstance() класу FirebaseAuth для підключення до бази даних.
4. До зміни класу auth викликається вбудований метод sugnUserWithEmailAndPassword(), яка приймає аргументи email – електронна пошта аккаунту користувача, та password – пароль до цього аккаунту.
5. За допомогою вбудованої властивості it.isSuccessful перевіряється, чи була аутентифікація успішною та повертає булеве значення.
6. Якщо аутентифікація пройшла успішно, змінній task передається значення «true», яке завершує з’єднання із базою даних, підвантаживши всі дані аккаунту користувача.
7. Якщо аутентифікація невдалася, змінній task передається значення «false», яке завершує з’єднання із базою даних без отримання даних аккаунту користувача.
8. Викликається вбудований метод showErrorToastMessage(), який повідомляє, що сталася помилка і відображає сповіщення типу Toast, яке вбудоване у систему Android.
9. Вихід із методу loginAccount().

Для проведення реєстрації використовується метод createAccount() якому також передаються аргументи електронна пошта та пароль. Однак, на відміну від попереднього методу, цей метод повертає булеве значення true, якщо дані були успішно збережені у базі даних. Блок-схема алгоритму цього методу показано на рисунку 3.6

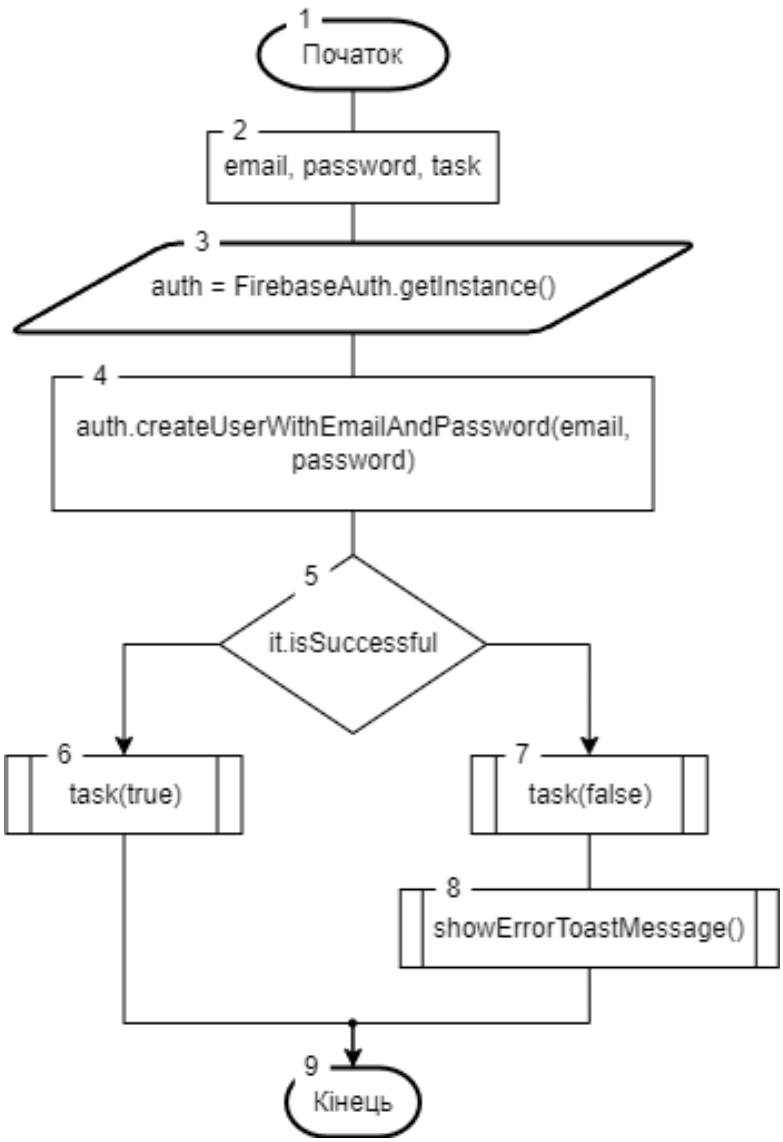


Рисунок 3.6 – Блок Блок-схема алгоритму методу createAccount

1. Виклик методу createAccount().

2. В якості аргументів методу передаються змінні email, що містить елекронну пошту, за допомогою якої буде створено аккаунт користувача, password, що містить пароль до аккаунту користувача і callback, змінна, яка буде повертати значення «true» або «false» для завершення операції реєстрації.

3. Ініціалізується змінна класу auth, викликаючи вбудований метод getInstance() класу FirebaseAuth для підключення до бази даних.

4. До зміни класу auth викликається вбудований метод createUserWithEmailAndPassword(), яка приймає аргументи email – електронна пошта аккаунту користувача, та password – пароль до цього аккаунту.

5. За допомогою вбудованої властивості it.isSuccessful перевіряється, чи була реєстрація була успішною та повертає булеве значення.

6. Якщо реєстрація пройшла успішно, змінній task передається значення «true», яке завершує з’єднання із базою даних, зберігаючи аккаунт користувача.

7. Якщо реєстрація невдалася, змінній task передається значення «false», яке завершує з’єднання із базою даних без зберігання аккаунту користувача у базі даних.

8. Викликається вбудований метод showErrorToastMessage(), який повідомляє, що сталася помилка і відображає сповіщення типу Toast, яке вбудоване у систему Android.

9. Вихід із методу loginAccount().

3.3 Створення методу композиції планів тренувань

Першим кроком для створення модуля відображення планів тренувань є створення методу deleteRecordsWithContraindiations(), який буде отримувати всі записи із тренуваннями та очищувати всі записи із колекції, які містять протипоказання. Блок-схема алгоритму зображена на рисунках 3.7 та 3.8.

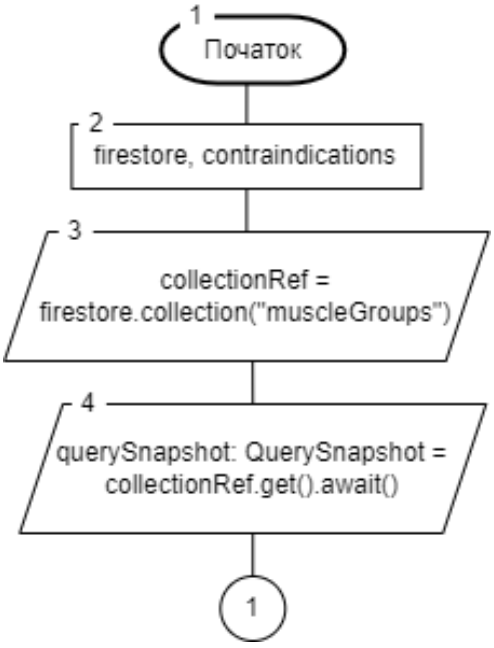


Рисунок 3.7 – Блок-схема алгоритму методу deleteRecordsWithContraindiations

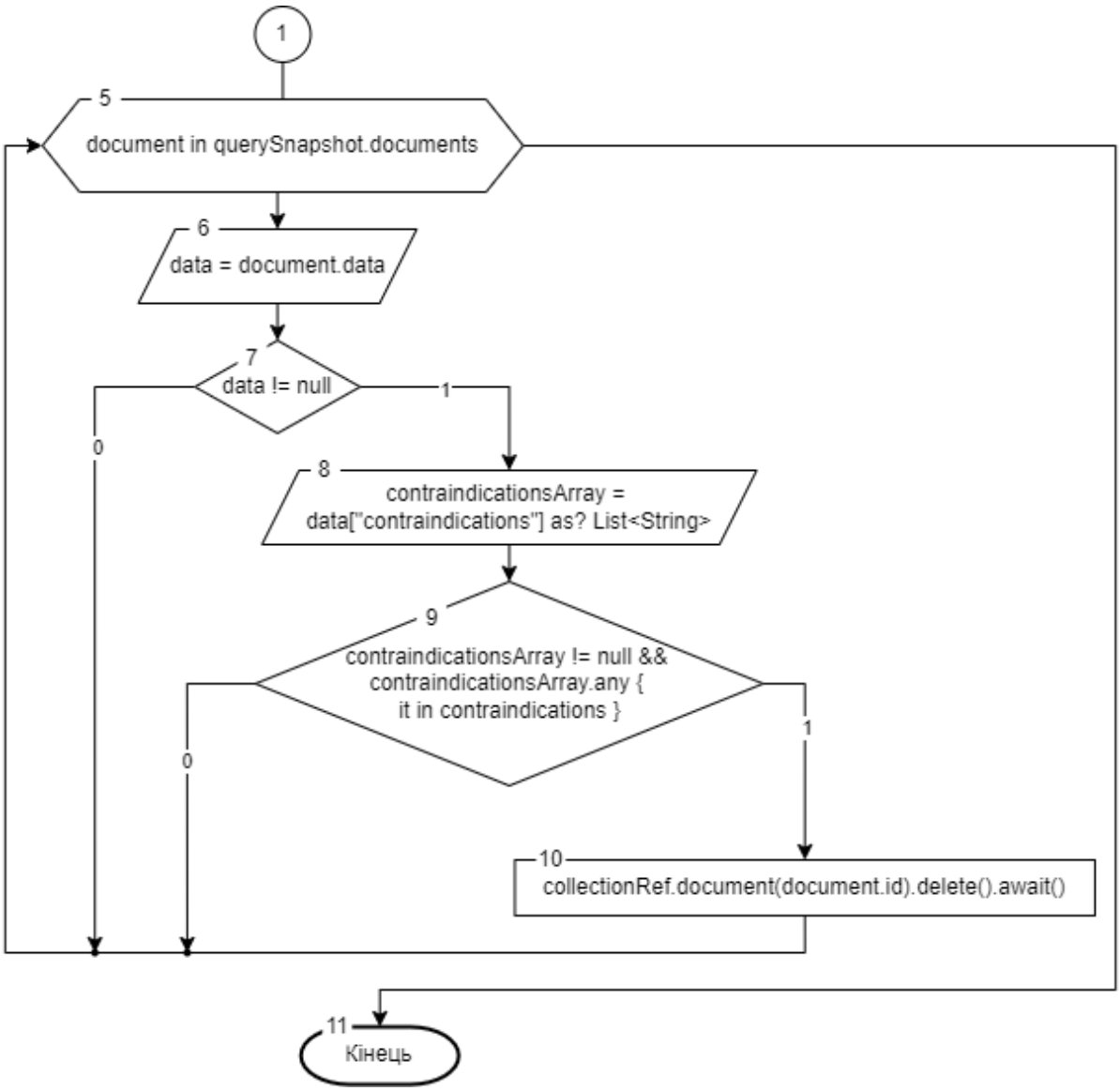


Рисунок 3.8 – Блок-схема алгоритму методу deleteRecordsWithContraindiations

1. Виклик методу deleteRecordsWithContraindiations().

2. В якості аргументів передаються firestore – езкемпляр Firestore, contraindications – список протипоказань.

3. Ініціалізується змінна collectionRef, що зберігає посилання на колекцію «muscleGroups».

4. Ініціалізується змінна querySnapshot, що містить усі записи колекції тренувань, викликаючи метод get() змінної collectionRef, перетворивши запит на асинхронний методом await().

5. Цикл, що перебирає кожний елемент результату запиту.

6. Ініціалізується змінна data, в яку записуються дані елементу у вигляді пар «ключ-значення».

7. Умова, що перевіряє, чи не є елемент запиту порожнім.

8. Ініціалізується змінна contraindicationsArray, у яку записується значення поля «contraindications» з даних елемента та перетворює його на список рядків.

9. Умова, що перевіряє, чи масив протипоказань не є пустим, і чи містить він будь-який елемент зі списку «contraindications».

10. Асинхронно видаляє елемент за його ID з колекції «muscleGroups».

11. Вихід із методу deleteRecordsWithContraindiations().

Для отримання тренувань, які будуть відображатися безпосередньо на екрані користувача, було створено метод getMuscleGroups(). Блок схема алгоритму роботи методу показана не рисунках 3.9 та 3.10.

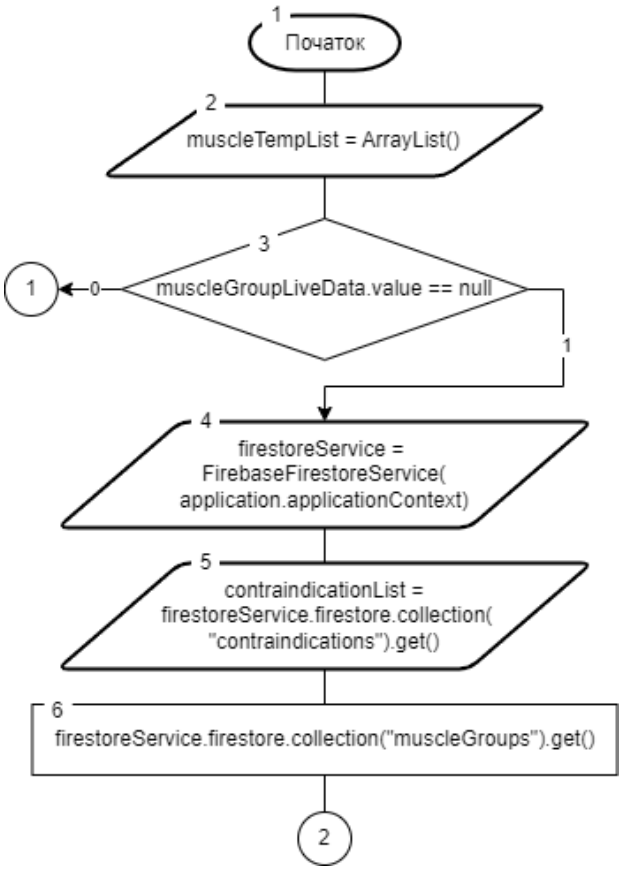


Рисунок 3.9 – Блок-схема алгоритму методу getMuscleGroups()

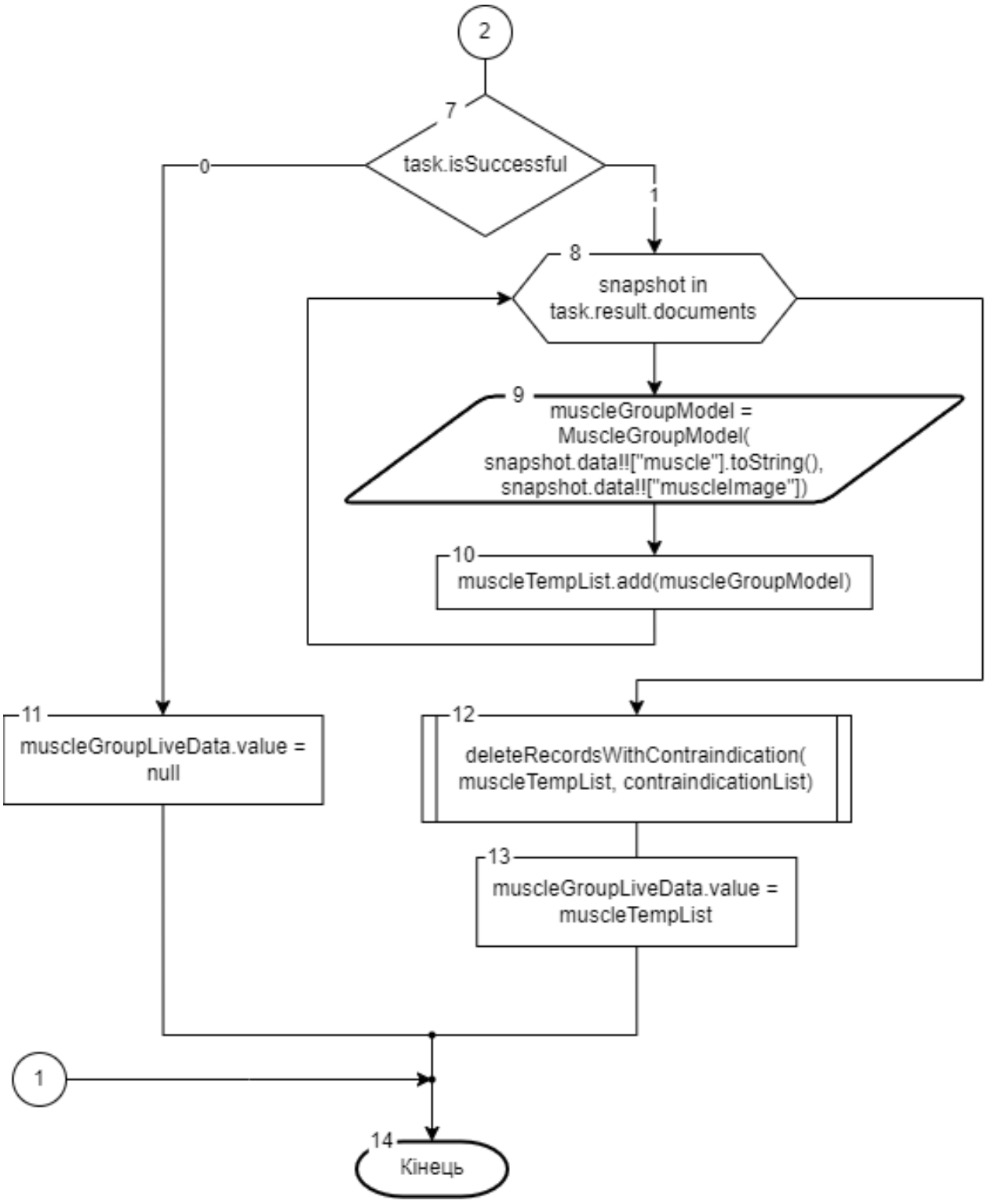


Рисунок 3.10 – Блок-схема алгоритму методу getMuscleGroups()

1. Виклик методу getMuscleGroups().
2. Ініціалізується змінна muscleTempList як список ArrayList.

3. Умова, що перевіряє, чи значення muscleGroupLiveData є null. Якщо так, то це означає, що дані ще не завантажені, і можна продовжити процес отримання даних з Firestore.

4. Встоновлення з’єднання із базою даних, та зберігає екземпляр для підключення у змінну firestoreService.

5. Викликає метод для отримання всіх елементів з колекції «muscleGroups».

6. Викликає метод для отримання всіх елементів з колекції «contraindication».

7. Перевіряється, чи запит виконано успішно за допомогою вбудованої властивості task.isSuccessful.

8. Проходить через кожний елемент, отриманий від запиту.

9. Створюється новий об'єкт MuscleGroupModel, де snapshot.data!!["muscle"] і snapshot.data!!["muscleImage"] отримують відповідні поля з документа та перетворюються на рядки.

10. Додає створену модель до тимчасового списку muscleTempList.

11. Якщо запит не вдався, встановлює значення muscleGroupLiveData в null, тобто, оновлювати або додавати нічого не потрібно.

12. Виклик методу deleteRecordsWithContraindication(), який видаляє елементи, в яких присутні протипоказання із списку contraindicationList.

13. Оновлює значення LiveData, додаючи записи на екран пристрою, після завершення обробки всіх елементів.

14. Вихід із методу getMuscleGroups().

3.4 Розробка модуля створення нотаток

Для створення користувачем нотаток було розроблено метод addToDoItem(), який в якості аргумента приймає todoArray – масив із данними, які були введені користувач для створення власного запису, та callback – посилання на булеве значення для отримання відгуку про успішне виконання методу. Алгоритм роботи цього методу показано на рисунку 3.11.

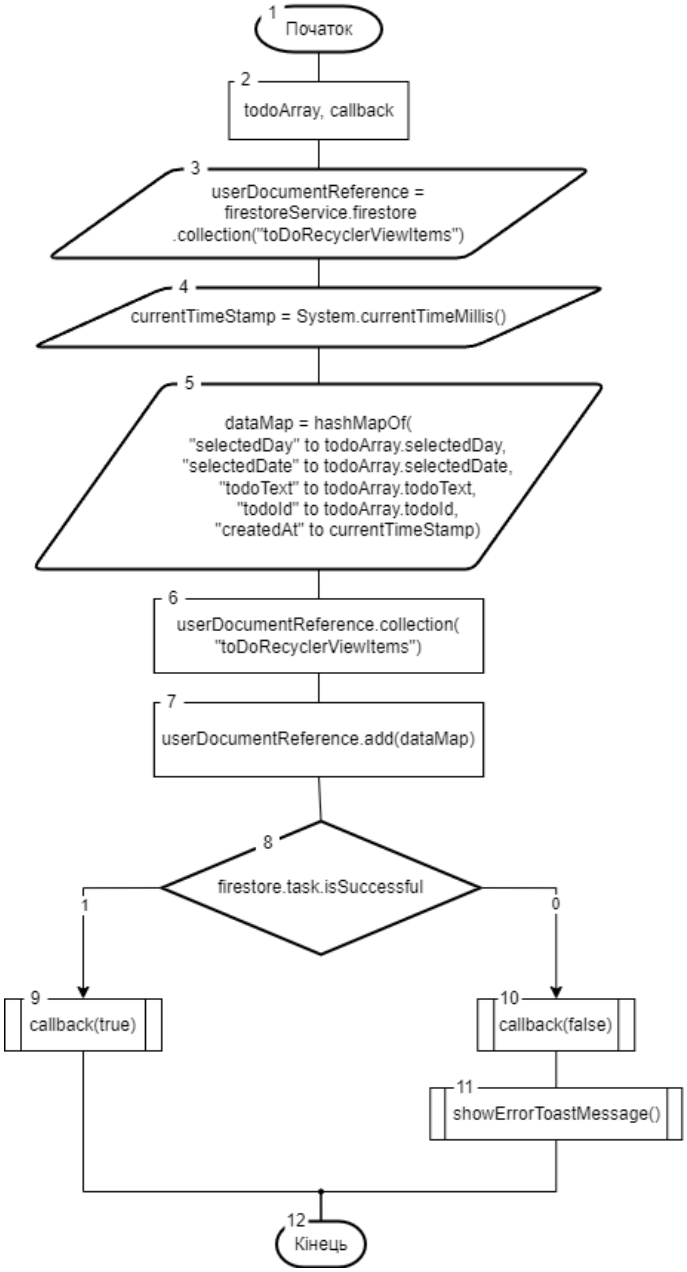


Рисунок 3.11 – Блок-схема алгоритму методу addToDoItem()

1. Виклик методу addToDoItem().

2. Ініціалізація зміних todoArray та callback переданими значення аргументів.

3. У змінну userDocumentReference зберігається посилання для підключення до бази даних Firebase.

4. У змінну currentTimeStamp зберігається час, в який була створена нова нотатка.

5. Ініціалізується змінна dataMap, яка являється набором даних у вигляді пар «ключ-значення», і записуються такі дані, як обраний день на який створюється нотатка, повна дата створення нотатки, текст запису, ідентифікатор запису та час створення нотатки.

6. Відбувається підключення до таблиці із даними про нотатки.

7. Додається новостворена до набору даних таблиці нотаток.

8. За допомогою вбудованої властивості it.isSuccessful перевіряється, чи була реєстрація була успішною та повертає булеве значення.

9. Якщо реєстрація пройшла успішно, змінній task передається значення «true», яке завершує з’єднання із базою даних, зберігаючи аккаунт користувача.

10. Якщо реєстрація невдалася, змінній task передається значення «false», яке завершує з’єднання із базою даних без зберігання аккаунту користувача у базі даних.

11. Викликається вбудований метод showErrorToastMessage(), який повідомляє, що сталася помилка і відображає сповіщення типу Toast, яке вбудоване у систему Android.

12. Вихід із методу addToDoItem().

3.5 Розробка інтерфейсу мобільного застосунку

Розробка інтерфейсу мобільного додатку - це ключовий етап в процесі створення програмного забезпечення, оскільки саме через нього користувач сприймає та взаємодіє з додатком. Правильно спроектований інтерфейс забезпечує комфортну, зрозумілу та зручну роботу з додатком, що, в свою чергу, підвищує задоволення користувача та збільшує ймовірність використання додатку.

Враховуючи це, було вирішено використовувати нижню навігаційну панель як основу для розробки інтерфейсу програмного застосунку. При використанні такого методу навігації, найчастіше структура інтерфейсу складається із безпосередньо панелі навігації, області, де буде розташовай

основний контент програмного застосунку, а також додаткові елементи, які використовуються для навігації між вікнами застосунку.

Нижня панель навігації розташовується внизу екрану та швидко дозволяє отримати доступ до ключових функцій. Зазвичай вона складається з іконок з або без написів, які відображають розділи програми. Коли користувач торкається іконки, відбувається перехід до відповідного розділу. Вміст розділу відображається над панеллю навігації і займає більшу частину екрану (див. рисунок 3.12). Крім того, розділи можуть містити верхню панель навігації як додатковий засіб навігації, що дозволяє отримати доступ до функціоналу розділу або переходу до інших екранів програми.

Рисунок 3.12 – Структура інтерфейсу на основі нижньої панелі навігації

Структура інтерфейсу програмного застосунку буде побудована на основі принципу єдиної активності з використанням Navigation Component [18]. Це означає, що додаток матиме одну активність – базовий компонент, через який

користувач буде взаємодіяти з програмою. Ця активність буде відповідальна за відображення вікна додатку. Для відображення контенту використовуються фрагменти, за допомогою яких, екрани і будуть наповнюватися інформацією. Для навігації між різними екранами буде використовуватися Navigation Component.

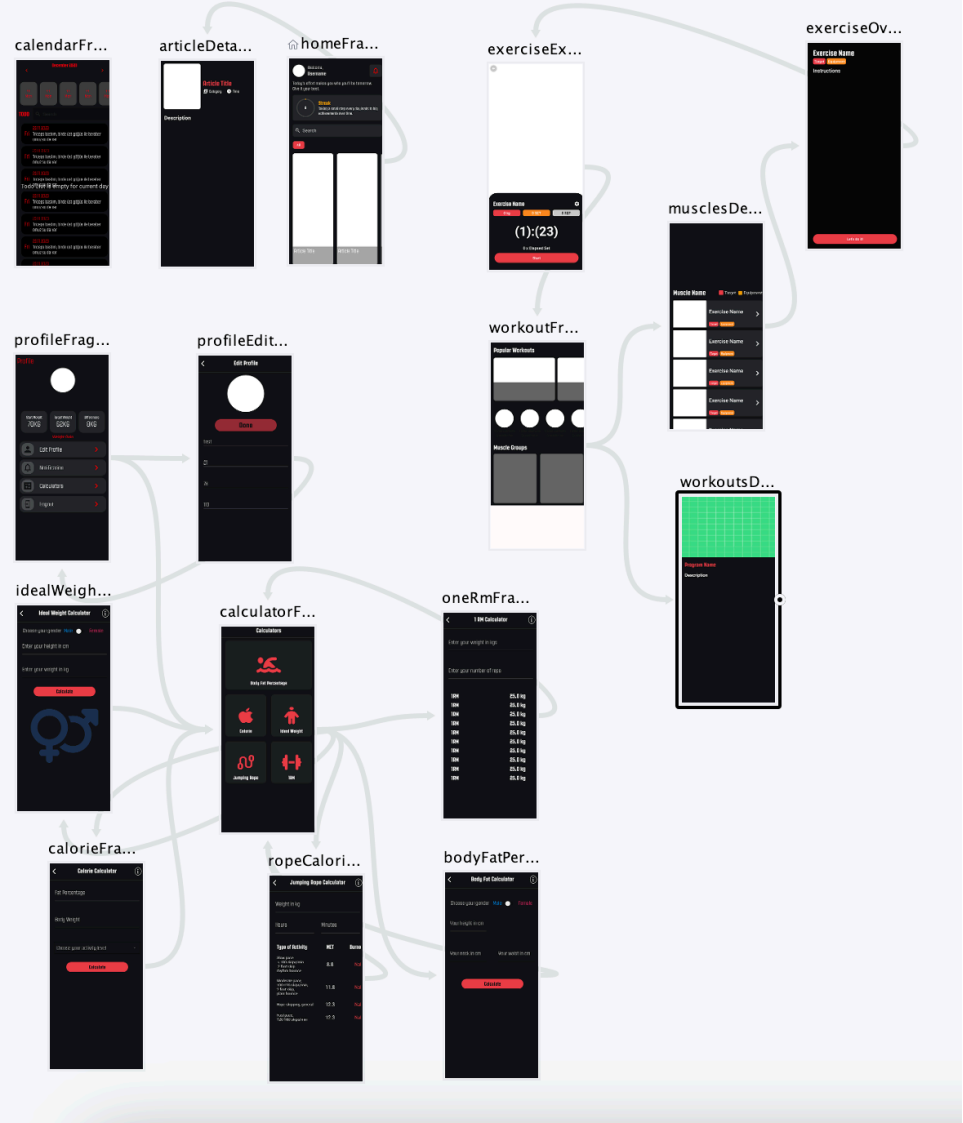


Рисунок 3.13 – Навігаційний граф програмного застосунку

Android Studio, інтегроване середовище розробки для Android, дозволяє за допомогою бібліотеки Navigation Component генерувати візуальне відображення навігаційних графів [19]. Навігаційний граф зображено на рисунку 3.13.

Нижня панель навігації буде використана для переходу між чотирма основними екранами програмного застосунку: домашній екран, екран тренувань, календар та профіль користувача.

Так як мобільний застосунок працює із реєстрацію користувачів, було спроектовано та розроблено екрани для реєстрації та авторизації користувачів. Під час перебування на екрані авторизації, користувач повинен ввести електронну пошту та пароль, якщо він був зареєстрованим раніше (див. рисунок 3.14). Якщо ж користувач запускає застосунок вперше, йому необхідно пройти процес реєстрації на відповідному екрані (див. рисунок 3.15), де необхідно вказати ім’я користувача, електронну пошту, пароль та вибрати стать, та натиснути кнопку «Continue».

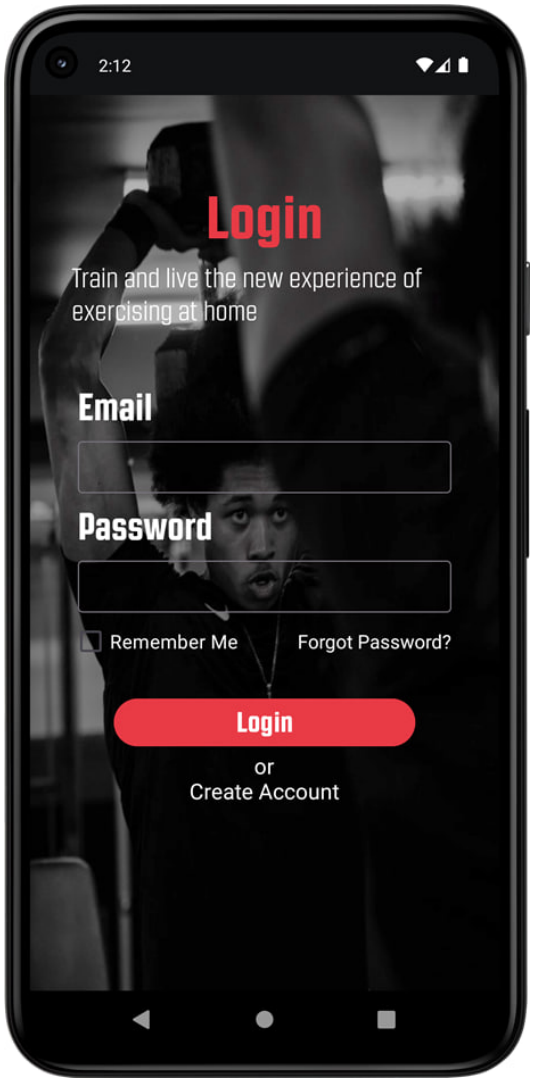


Рисунок 3.14 – Екран авторизації користувача

Якщо користувач обрав реєстрацію, то після введення основних даних користувача, система попросить користувача обрати протипоказання, що стосуються здоров’я. Щоб обрати протипоказання слід натиснути на кнопку типу Checkbox. Користувач може обирати декілька протипоказань, щоб підлаштувати плани тренувань максимально під себе. Із усіх можливих протипоказань є «Серцеві захворювання», «Респіраторні захворювання», «Опорно-руховий апарат», «Метаболічні та ендокринні захворювання», «Неврологічні захворювання», «Інфекційні захворювання», «Психіатричні розлади».

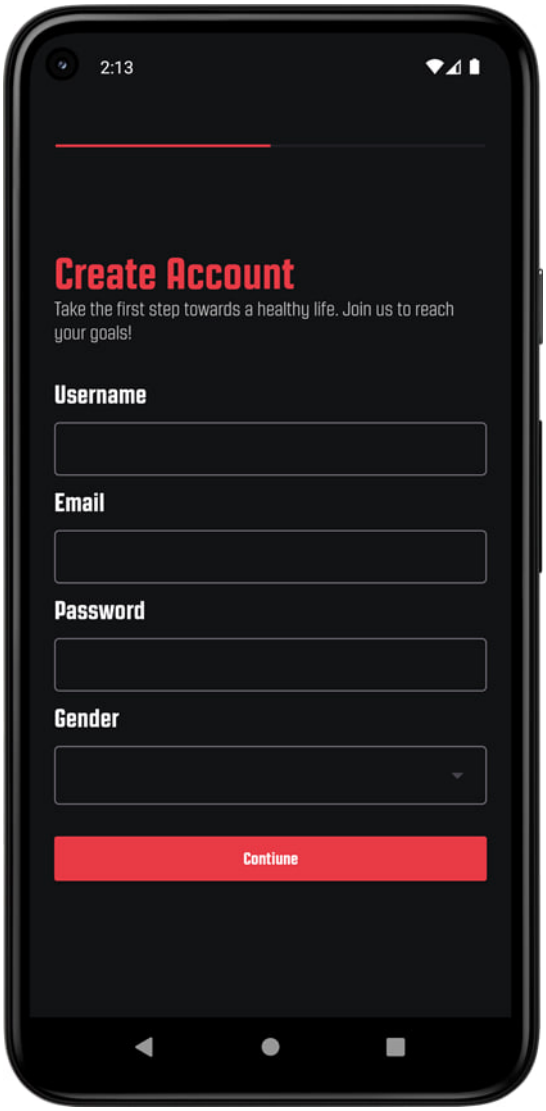


Рисунок 3.15 – Екран реєстрації користувача

Після проходження авторизації або реєстрації, користувач потрапляє на головний екран мобільного застосунку (див. рисунок 3.16). На ньому відображається такі елементи, як ім’я користувача, яке було введено під час реєстрації, віджет, який повідомляє про кількість фізичних вправ, які були проведені впродовж поточного дня. Поле для пошуку допомагає швидко знайти основну інформацію про види тренувань або способи навантаження, і безпосередньо область, яка показує всі можливі навантаження, а при введеному запиті у полі пошуку – відповідні елементи.

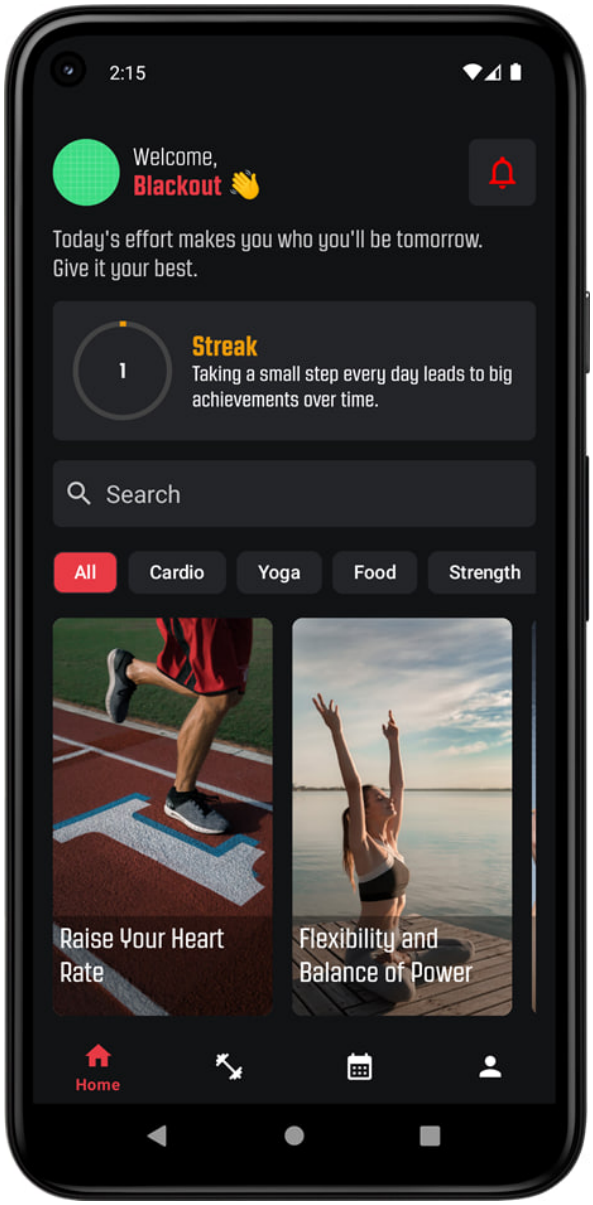


Рисунок 3.16 – Головний екран

Також, внизу усіх основних екранів розташована панель навігації, яка допомагає переміщатися між екранами із найважливішими функціями мобільного застосунку.

Натиснувши на другу кнопку у панель навігації, яка додатково зображена у вигляді гантелі, користувач потрапляє на екран «Workout», який надає йому можливість обирати фізичні вправи певної категорії (див. рисунок 3.17). Серед категорій фізичних вправ є «Спина», «Кардіо», «Груди», «Ноги», «Нижня частина рук», «Верхня частина рук» та «Плечі». Залежно від того, які протипоказання були обрані користувачем, категорії тренувань, які можуть бути недоступні для користувача .Також, на цьому екрані користувачеві показано тренування, які отримали більшу популярність за останній час.

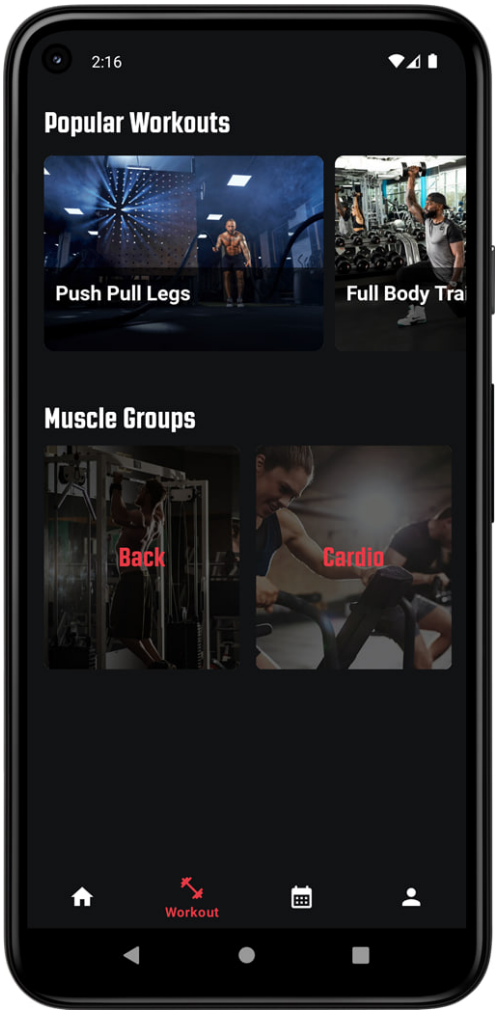


Рисунок 3.17 – Екран «Workout»

Під час перебування на екрані «Workout», користувач може обрати одну із доступних категорій тренувань, натиснувши на кнопку із зображення та назвою категорії, після чого він буде перенаправлений на екран «Category». На цьому екрані відображаються усі тренування у вигляді списку, які стосуються обраної категорії. За аналогію з категоріями, деякі вправи можуть бути недоступні, залежно від обраних користувачем протипоказань (див. рисунок 3.18).

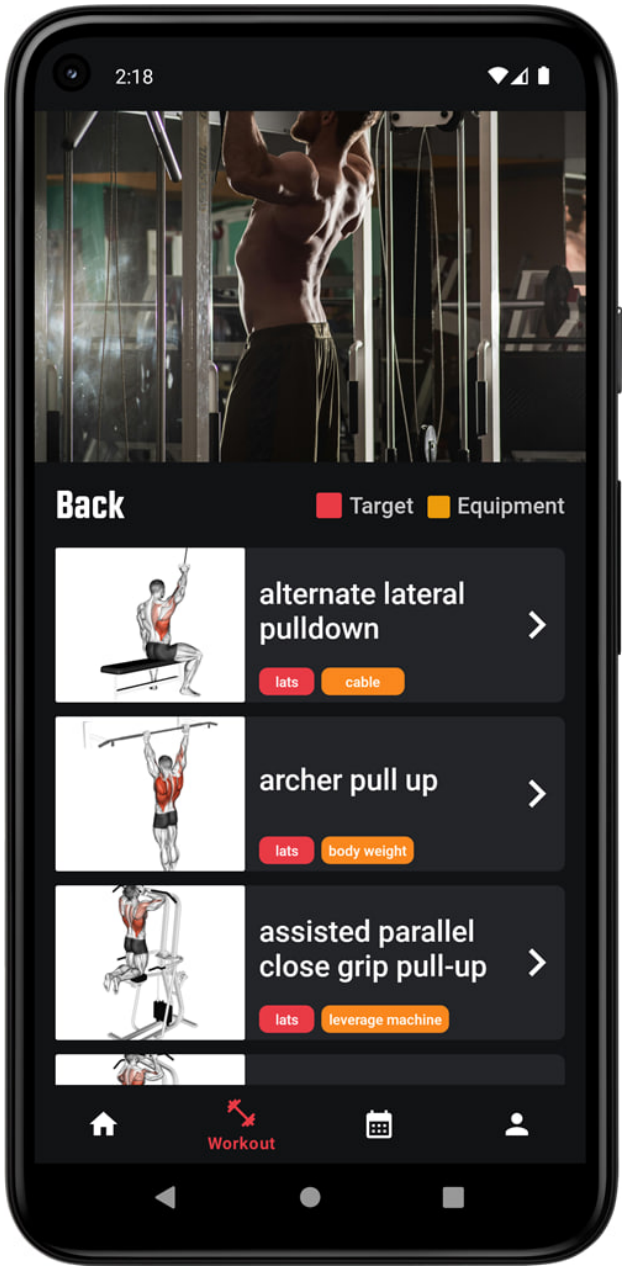


Рисунок 3.18 – Екран «Category»

Для того, щоб обрати тренування, необхідно натиснути на елемент списку, який відповідає фізичній вправі із назвою тренування та малюнком. Після чого, користувачу буде показано модальне вікно з обраним тренуванням та його описом. Додатково, щоб краще персоналізувати обрану вправу, користувачу необхідно ввести вагу навантаження – Weight, одиниці виміру ваги – Type, вказати кількість підходів – Sets та кількість повторів – Sets (див. рисунок 3.19).

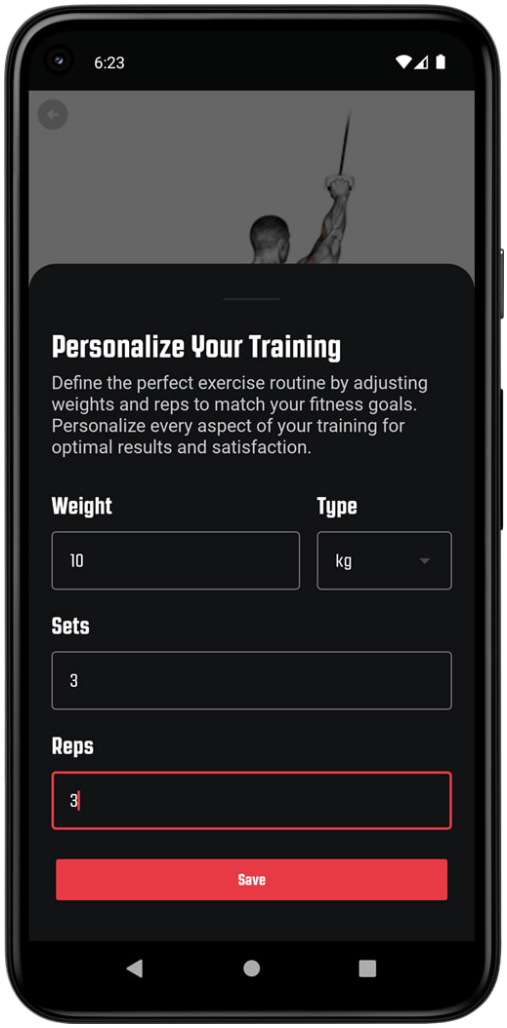


Рисунок 3.19 – Вікно обраного тренування

Після введення необхідних даних модальне вікно закривається, а користувачу відображається екран на якому показується анімація, яка демонструє правильне виконання обраної вправи (див. рисунок 3.20). В низу екрану відображається основна введена інформація така як назва вправи,

обрана вага, кількість підходів та повторів. Нижче додано секундомір, який слугує для інформування часу, витраченого на проведення обраної вправи. Для управління секундоміром створена кнопка «Play/Pause», яка призначена для зупинки або продовження виконання вправи. Також, поруч розташована кнопка, яка дозволяє завершити поточний підхід виконання вправи та перейти до наступного. Після звершення усієї вказаної кількості підходів модальне вікно закривається і користувач переноситься на екран із списком раніше обраної категорії вправ.

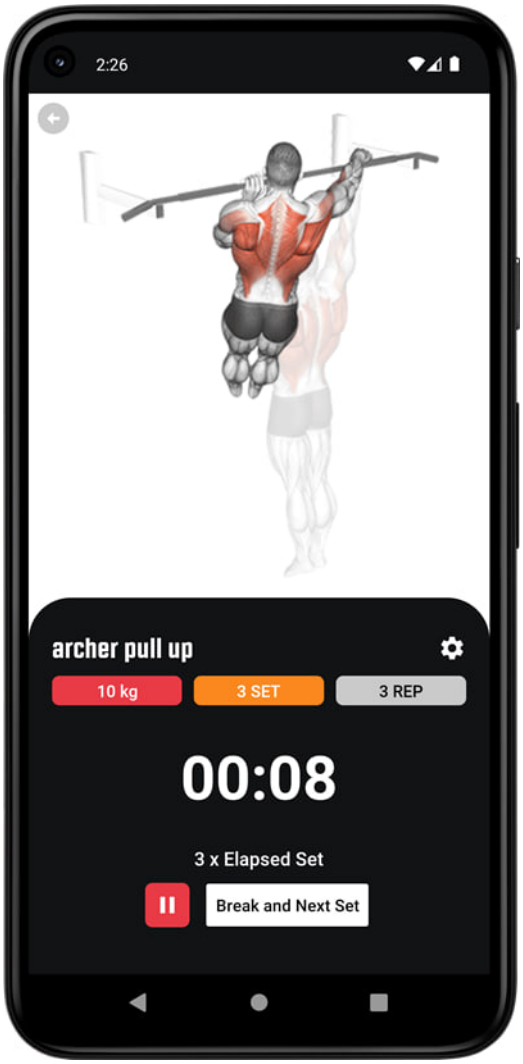


Рисунок 3.20 – Модальне вікно виконання вправи

Однією із основних можливостей мобільного застосунку є створення нотаток та ведення історії виконання фізичних навантажень. Для того, щоб

відкрити історію необхідно у нижній панель навігації обрати третій пункт під назвою «Calendar» (див. рисунок 3.21). На цьому екрані відображається поточний місяць і річ. Нижче розташовані кнопки-дати, навігація по яких відбувається шляхом пролистуванням вліво або вправо.

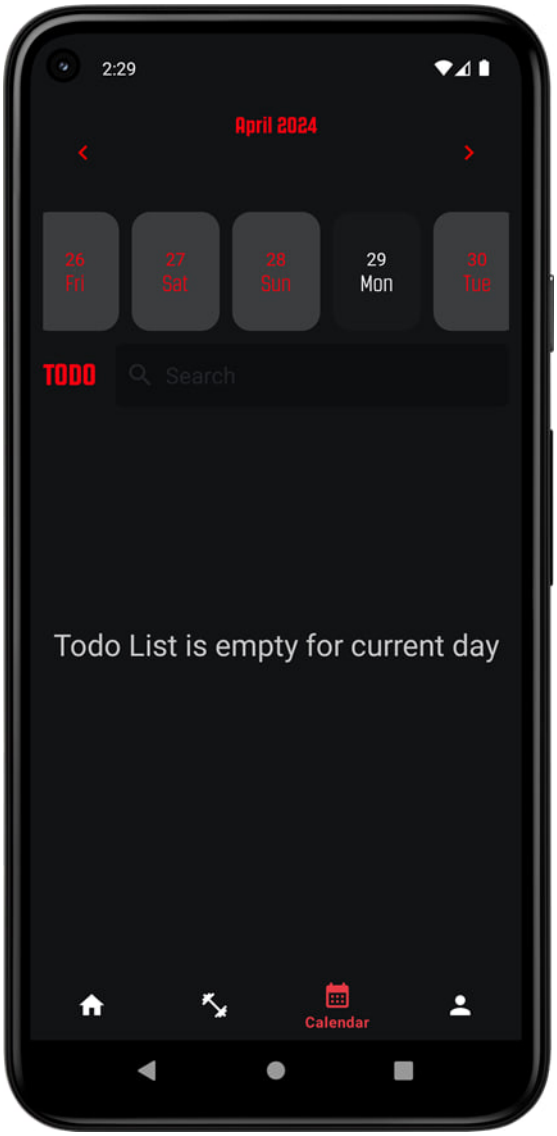


Рисунок 3.21 – Екран «Calendar»

Якщо натиснути на одну із кнопок, в області знизу будуть відображені створені нотатки або історія виконання фізичних вправ, якщо така існує.

Під датами розташована кнопка «TODO», яка дозволяє створювати нові нотатки для обраного дня після натискання (див. рисунок 3.22).

Рисунок 3.22 – Історія тренувань

Розробка користувацького графічного інтерфейсу мобільного застосунку була проведена у середовищі розробки Android Studio, використовуючи вбудовані технології Android Jetpack Compose [20] та ViewBinding [21].

3.6 Висновки

У третьому розділі було обґрунтовано вибір мови програмування та технологій для розробки модулів мобільного застосунку. Проаналізувавши потреби програмного продукту було обрано мову програмування Kotlin. Для зручності розробки графічного інтерфейсу було обрано Jetpack Compose та ViewBinding. В якості бази даних було обрано Firebase.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування системи

Тестування програмного забезпечення (ПЗ) – це процес перевірки, валідації та верифікації програм для забезпечення їх якості, правильності та відповідності вимогам. Цей процес є важливою складовою розробки ПЗ і має на меті виявлення помилок та недоліків у програмі до її впровадження.

Важливість проведення тестування ПЗ важко переоцінити. Ось кілька ключових причин:

- Тестування допомагає впевнитися, що програмне забезпечення відповідає вимогам щодо якості та функціональності.

- Виявлення помилок. Через тестування виявляються помилки та дефекти в програмі, що дозволяє їх виправити перед випуском продукту на ринок.

- Економія коштів. Виявлення помилок на ранніх етапах розробки є набагато дешевшим, ніж виправлення їх пізніше в процесі експлуатації.

- Покращення репутації компанії. Висока якість програмного забезпечення допомагає підвищити репутацію компанії серед користувачів.

Види тестування можна класифікувати за різними критеріями, такими як методологія, тип дефектів, область застосування тощо [22]. Ось кілька загальновживаних видів тестування:

- Функціональне тестування. Перевірка функціональних вимог програми, тобто тестування функціональності .

- Нефункціональне тестування. Тестування, що не стосується конкретної функціональності програми, але важливе для забезпечення якості, наприклад, тестування продуктивності, надійності, безпеки тощо.

- Модульне тестування. Тестування окремих модулів або компонентів програми [23].

- Інтеграційне тестування. Тестування взаємодії між різними компонентами або модулями програми.

- Системне тестування. Перевірка системи як цілісності, включаючи всі її компоненти та їх взаємодію.

- Приймальне тестування. Тестування з метою підтвердження відповідності програми вимогам замовника або стандартам якості.

Під час розробки мобільного застосунка персонального консультанта в галузі охорони здоров'я були створені класи та функції для проведення юніт-тестування, використовуючи бібліотеку Junit. Ці класи слугують для перевірки та валідації електронної пошти та паролю під час реєстрації нового користувача. Діаграми класів цих класів зображені на рисунках 4.1 та 4.2.

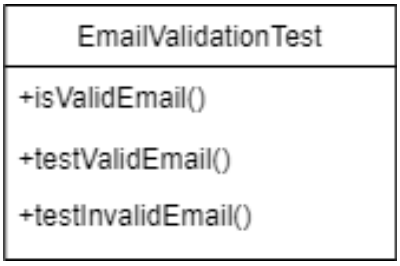


Рисунок 4.1 – Клас тестування електронної пошти

У класі тестування введеної електронної пошти – EmailValidationTest, використовується RegEx (Регулярні вирази), для перевірки, чи записана електронна пошта відповідно стандартам.

Основним методом для перевірки елекронної пошти є метод isValidEmail(). Блок-схема алгоритму цього методу зображена на рисунку 4.2.

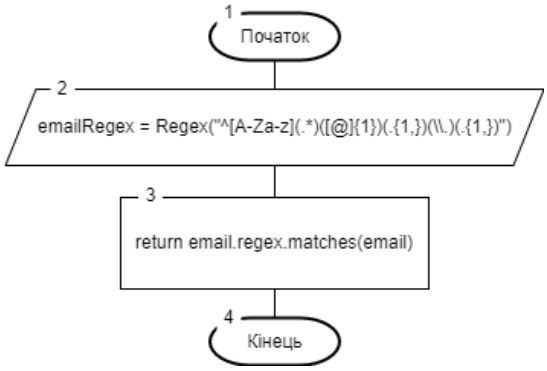


Рисунок 4.2 – Метод перевірки електронної пошти за шаблоном

1. Виклик методу тестування isValidEmail().
2. Ініціалізація змінної та збереження шаблону для перевірки електронної пошти.
3. Повертає значення «true» або «false», відповідно до того, чи відповідає введена електронна пошта шаблону, чи ні, використовуючи метод matches().
4. Вихід з методу isValidEmail().

Для проведення unit-тестування на правильність введеного користувачем паролю, було створено метод testValidEmail() – перевіряє значення «example@example.com», що в результаті повинно повертатися значення «true».

Блок-схема цього методу зображена на рисунку 4.3.

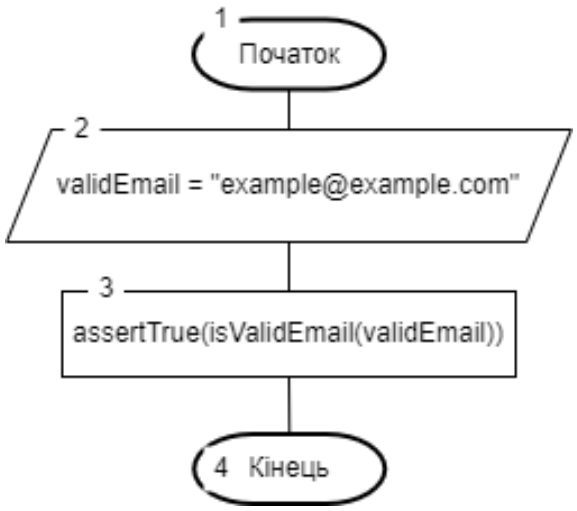


Рисунок 4.3 – Блок схема перевірки методу на правильність електронної пошти

1. Виклик методу тестування testValidEmail().
2. Ініціалізація змінної validEmail та збереження введеної електронної пошти
3. Використовується вбудований метод для тестування assertTrue(), який приймає аргументи або методи, які повинні повернути значення «true». Якщо

при перевірці було повернено значення «false», метод assertTrue() повертає «false», і тест буде не пройдений.

4. Вихід з методу isValidEmail().

Для проведення unit-тестування на неправильність введеного користувачем паролю, було створено метод testInvalidEmail() – перевіряє значення «example@.com», що в результаті повинно повертатися значення «false». Блок-схема цього методу зображена на рисунку 4.4.

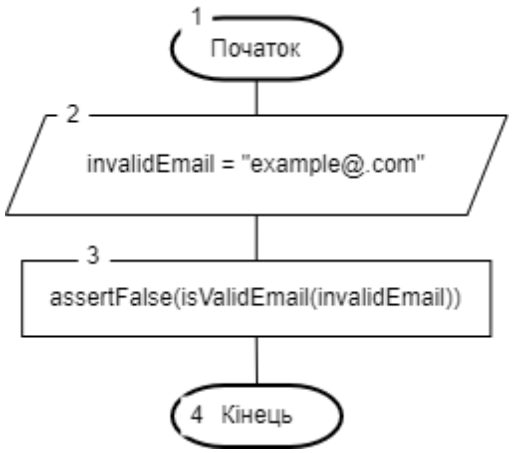


Рисунок 4.4 – Блок схема алгоритму методу testInvalidEmail

1. Виклик методу тестування testInvalidEmail().

2. Ініціалізація змінної invalidEmail та збереження введеної електронної пошти.

3. Використовується вбудований метод для тестування assertFalse(), який приймає аргументи або методи, які повинні повернути значення «false». Якщо при перевірці було повернено значення «true», метод assertFalse() повертає «false», і тест буде непройдений.

4. Вихід з методу isInvalidEmail().

У класі тестування введеного паролю, однак в цьому випадку, використовуються три шаблони, які перевіряють, чи не містить пароль символів

кирилиці, чи містить хоча б одну цифру і одну літеру у великому регістрі. Діаграма класу, для тестування введеного паролю зображена на рисунку 4.5.

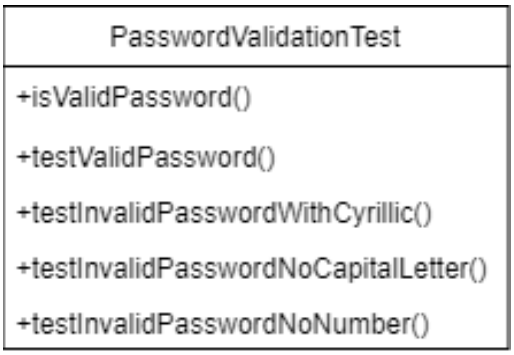


Рисунок 4.5 – Діаграма класу PasswordValidationTest

Основним методом для перевірки паролю є метод isValidPassword(). Блок-схема алгоритму цього методу зображена на рисунку 4.6.

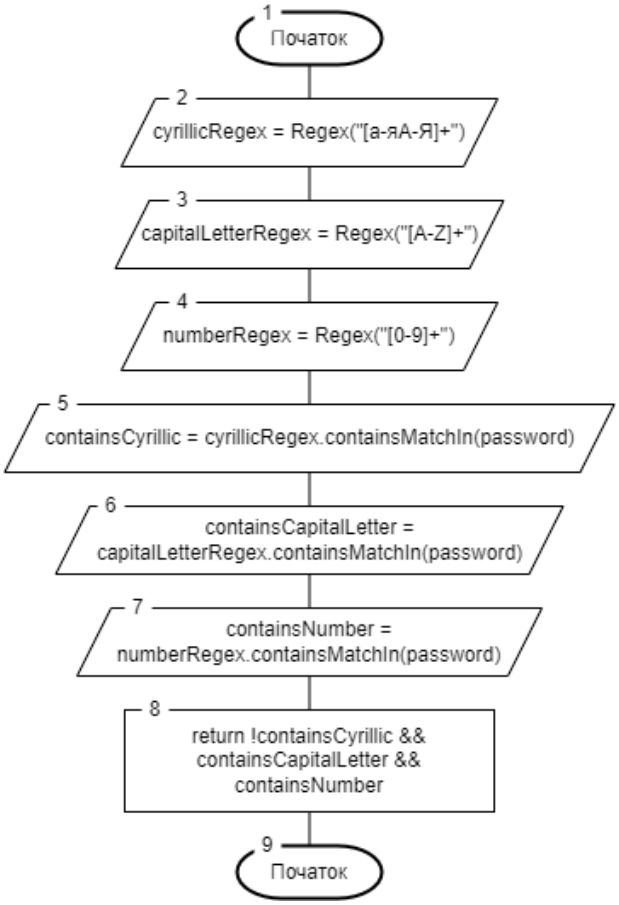


Рисунок 4.6 – Блок-схема матоду isValidPassword()

1. Виклик методу тестування isValidPassword().

2. Створюється регулярний вираз для перевірки наявності кириличних символів у рядку.

3. Створюється регулярний вираз для перевірки наявності великих латинських літер у рядку.

4. Створюється регулярний вираз для перевірки наявності цифр у рядку.

5. Перевіряється, чи містить рядок password кириличні символи.

6. Перевіряється, чи містить рядок password великі латинські літери.

7. Перевіряється, чи містить рядок password цифри.

8. Повертається true, якщо рядок не містить кириличних символів і рядок містить великі латинські літери, і рядок містить цифри.

9. Вихід з методу isValidPassword().

Для проведення базової перевірки паролю був створений метод testValidPassword(). Блок-схема алгоритму цього методу показана на рисунку 4.7.

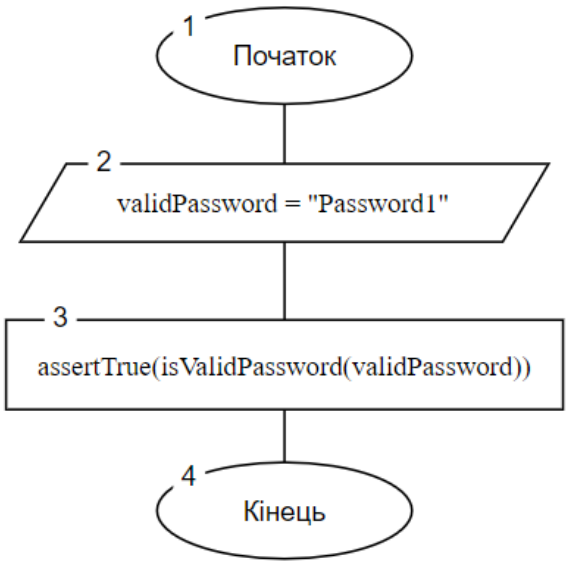


Рисунок 4.7 – Блок-схема алгоритму методу testValidPassword()

1. Виклик методу тестування testValidPassword().

2. Створюється локальна змінна validPassword типу String з присвоєним значенням "Password1".

3. Використовується метод assertTrue() для перевірки, чи повертає метод isValidPassword() значення «true» для пароля "Password1".

4. Вихід з методу testValidPassword().

Для перевірки введеного паролю на наявність символів кирилиці був створений метод testInvalidPasswordWithCyrillic(). Блок-схема алгоритму методу показана на рисунку 4.8.

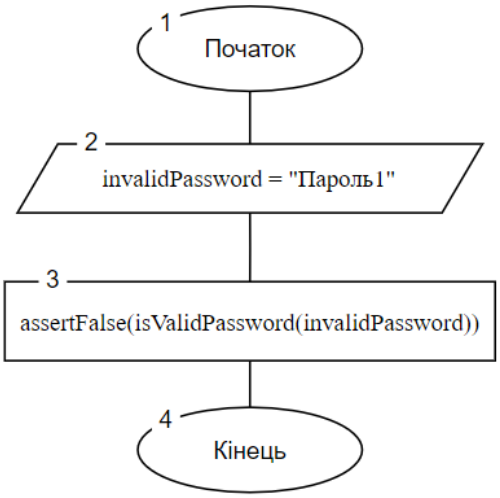


Рисунок 4.8 – Блок-схема алгоритму методу testInvalidPasswordWithCyrillic()

1. Виклик методу тестування testInvalidPasswordWithCyrillic().

2. Оголошується константа invalidPassword і їй присвоюється значення рядка "Пароль1".

3. Це виклик функції assertFalse, яка перевіряє, що вираз в дужках – метод isValidPassword(), поверне «false».

4. Вихід з методу testInvalidPasswordWithCyrillic().

4.2 Розробка інструкції користувача

Інструкція користувача – це документ, який містить посібник або набір інструкцій, які допомагають користувачеві ефективно використовувати певний продукт чи сервіс. Вона призначена для того, щоб надати користувачеві необхідну інформацію та допомогти йому зрозуміти, як правильно використовувати продукт, уникнути помилок та досягти бажаного результату. Важливість інструкції користувача важко переоцінити, особливо у сфері програмного забезпечення чи складних технічних пристроїв.

Інструкція для використання мобільного застосунку:

1. Запустити мобільний застосунок.
2. Якщо користувач не зареєстрований у системі, на початковому екрані обрати опцію «Get started». Якщо користувач зареєстрований перейти до пункту 4.
3. Пройти реєстрацію у системі, ввівши дані у поля «Usename», «Email», «Password» і «Gender». Натиснути на кнопку «Continue».
4. На початковому екрані обрати опцію «Login».
5. Ввести поля «Email» та «Password». Натиснути кнопку «Login».
6. Для початку тренувань перейти на екран «Workout», натиснувши на кнопку у вигляді гантелі у нижньому меню.
7. Обрати одну категорію із секції «Muscle Groups» та натиснути на неї.
8. В новому вікні обрати та натиснути на бажаний елемент тренування у списку.
9. Прочитати опис та натиснути на кнопку «Let’s do it!».
10. У модальному вікні вказати бажану вагу – «Weight», обрати одиниці вимірювання – «Type», кількість підходів – «Sets» та кількість повторів – «Reps». Потім натиснути на кнопку «Save».
11. В новому екрані натиснути на кнопку «Start» для початку тренувань.
12. Для призупинення/продовження тренування натиснути на кнопку «Play/Pause».
13. Для завершення тренування натиснути на кнопку «Break and next Set».

4.3 Висновки

У четвертому розділі було проведено тестування системи та наведено приклади розроблених тестів. Також було розроблено інструкцію користувача для пояснення порядку роботи із мобільним застосунком.

ВИСНОВКИ

У бакалаврській дипломній роботі був розроблений мобільний застосунок для персонального медичного консультування у галузі охорони здоров’я. Розроблене програмне забезпечення має на меті покращити якість процесу надання рекомендацій користувачам щодо програм тренувань, враховуючи їхні індивідуальні медичні протипоказання

Було розглянуто стан питання мобільних застосунку на платформі Android та методи розв’язання задачі, проведено порівняльний аналіз аналогів моблільних застосунків, описано напрямки вдосконалення таких програмних продуктів та підходи створення.

Було розроблено архітектуру мобільного застосунку, використовуючи структурний шаблон MVVM. Було розроблено загальний алгоритм роботи мобільного застосунку та методи подання планів тренувань для користувачів з урахуванням вказаних ними протипоказань.

Було проведено порівняльний аналіз переваг та недоліків популярних мов програмування та середовищ розробки. На основі аналізу було обрано мову програмування Kotlin для розробки мобільного застосунку та інтегроване середовище розробки Andoroid Studio. Описано процес розробки основних функціональних модулів, таких як: інтерфейс користувача, модуль відображення персоналізованих планів тренувань та модуль створення власних записів користувача.

На основі наданої теми було розглянуто актуальність впровадження тестувань у програмні застосунки та різновиди тестувань. З огляду на це, обґрунтовано доцільність впровадження функціонального тестування для забезпечення працездатності та ефективності роботи системи. Було створено тестові класи для перевірки основного функціоналу системи. За результатами тестування підтверджено працездатність та ефективність роботи системи. Також було описано інструкцію користувача мобільного застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Проблеми та перспективи використання програмних застосунків у галузі охорони здоров‘я. Лейбак Д.В., Катєльніков Д.І. (Вінницький національний технічний університет).

2. AdaHealth [Електронний ресурс] – Режим доступу до ресурсу: https://ada.com/.

3. MyFitnessPal [Електронний ресурс] – Режим доступу до ресурсу: https://www.myfitnesspal.com.

4. The Fitbit App [Електронний ресурс] – Режим доступу до ресурсу: https://www.fitbit.com/global/us/technology/fitbit-app

5. Типи мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://smile-ukraine.com/ua/mobile-apps/mobile-apps-types

6. Кросплатформова розробка додатків [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/krossplatformennaya-razrabotka-prilozhenij

7. Гібридні мобільні додатки [Електронний ресурс] – Режим доступу до ресурсу: https://web4u.in.ua/blog/g-bridn-mob-l-n-dodatki-ta-h-perevagi-18

8. Архітектура мобільного застосунку [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/arhitektura-mobilnogo-prilozheniya

9. Логіка відображення стану View в Android: проектуємо і тестуємо [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/33022/

10. Інтегроване середовище розробки [Електронний ресурс] – Режим доступу до ресурсу: https://w3schoolsua.github.io/hyperskill/ide.html#gsc.tab=0

11. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio

12. Eclipse https://www.eclipse.org/downloads/

13. IntellijIdea [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/

14. Gradle [Електронний ресурс] – Режим доступу до ресурсу: https://gradle.org/

15. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/

16. Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/

17. Огляд і основи мови програмування C++ [Електронний ресурс] – Режим доступу до ресурсу: http://www.znannya.org/?view=Cpp_basics

18. Navigation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/guide/navigation

19. Design your navigation graph [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/guide/navigation/design

20. Build better apps faster with Jetpack Compose [Електронний ресурс] - Режим доступу до ресурсу: https://developer.android.com/develop/ui/compose

21. ViewBinding [Електронний ресурс] - Режим доступу до ресурсу: https://developer.android.com/topic/libraries/view-binding

22. Види тестування програмного забезпечення [Електронний ресурс] - Режим доступу до ресурсу: https://sqa.lviv.ua/yaki-ye-typy-testuvannya

23. Модульне тестування [Електронний ресурс] - Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/modulne-testuvannya/

ДОДАТКИ

63

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
О. Н. Романюк _____
«12» березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка мобільного застосунку для персонального медичного консультування на платформі Android з використанням технологій Kotlin і Firebase»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доцент Д. І. Катєльніков
«12» березня 2024 р.
Виконав:
_____ студент гр. ЗПІ-20б Д. В. Лейбак
«12» березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мобільного застосунку для персонального медичного консультування на платформі Android з використанням технологій Kotlin і Firebase».

Галузь застосування – спортивна медицина.

2. Підстава для розробки

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки

Метою дослідження полягає в тому, щоб покращити якість процесу надання рекомендацій користувачам щодо програм тренувань, враховуючи їхні індивідуальні медичні протипоказання.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Singh Ajit. Android Application Development. Сіетл, США : Amazon Publishing, 2022. 318 с.

2. А.М. Пєтух, О.В. Романюк, О.Н. Романюк. Бази даних. Мови запитів, управління транзакціями, розподілена обробка даних. Вінниця : ВНТУ, 2016. 97 с.

3. O'Regan Gerard. Concise Guide to Software Testing. Хам, Швейцарія : Springer International Publishing, 2019. 293 с.

5. Технічні вимоги

Тип програми – мобільний застосунок; модель розробки ітеративна; засоби організаціїї даних програми – Firebase; вхідні дані: персональні дані

реєстрації користувачів, протипоказання фізичної активності користувачів; вихідні дані: відображення тренувань користувачів, нотатки; середовище розробки – Android Studio; мова програмування – Kotlin; програмне забезпечення для симуляції мобільних пристроїв – Android Emulator.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинги програми

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

9. Стадії та етапи розробки:

№ п\п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану мобільних застосунків в галузі охорони здоров'я	13.03.24 –22.03.24
2	Розробка структури мобільного застосунку	13.03.24 –22.03.24
3	Розробка алгоритмів роботи програмного забезпечення	25.03.24 –19.04.24
4	Розробка програмного забезпечення	22.04.24 –10.05.24
5	Тестування програмного забезпечення	13.05.24 – 24.05.24
6	Оформлення матеріалів до захисту БДР	27.05.24 – 03.06.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

67

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільного застосунку для персонального медичного консультування на платформі Android з використанням технологій Kotlin і Firebase

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк О.Н.

Оригінальність	91.3%
Схожість	8.7%

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформленні коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цілісності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховання недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Лейбак Д.В.

Керівник роботи Катєльніков Д.І.

Додаток В. Лістинг програми

FirebaseAuthService.kt

```
package com.app.ebfitapp.service

import android.app.Dialog
import android.content.Context
import android.widget.Toast
import com.app.ebfitapp.utils.CustomProgress
import com.google.firebase.auth.FirebaseAuth

class FirebaseAuthService(private val context: Context) {
    private var auth: FirebaseAuth = FirebaseAuth.getInstance()
    fun createAccount(email: String, password: String, task: (Boolean) -> Unit) {
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener {
            if (it.isSuccessful) task(true)
        }.addOnFailureListener {
            task(false)
            showErrorToastMessage(it.message.toString())
        }
    }

    fun loginAccount( email: String,
                      password : String,
                      task: (Boolean) -> Unit) {
        auth.signInWithEmailAndPassword(email,password)
            .addOnCompleteListener {
                if (it.isSuccessful) task(true)
                else task(false)
            }.addOnFailureListener() {
                task(false)
                showErrorToastMessage(it.message.toString())
            }
    }

    fun getForgotPassowrd(sPassword : String , dialog: Dialog, customProgress: CustomProgress) {
        auth.sendPasswordResetEmail(sPassword)
            .addOnSuccessListener {
                customProgress.dismiss()
                showErrorToastMessage("Please Check Your Email Address")
                dialog.dismiss()
            }.addOnFailureListener {
                customProgress.dismiss()
                showErrorToastMessage(it.message.toString())
            }
    }

    fun signOut() {
        auth.signOut()
    }

    fun getCurrentUserEmail(): String {
        return auth.currentUser?.email.toString()
    }

    private fun showErrorToastMessage(error: String) {
        Toast.makeText(context, error, Toast.LENGTH_LONG).show()
    }
}
```

FirebaseFirestoreService.kt

```
package com.app.ebfitapp.service
```

```
import android.content.Context
import android.widget.Toast
import com.google.firebase.firestore.FirebaseFirestore

class FirebaseFirestoreService(private val context: Context) {
    var firestore: FirebaseFirestore = FirebaseFirestore.getInstance()
    fun createProfileDetail(dataMap: HashMap<String,Any?>, callback: (Boolean) -> Unit) {
        firestore.collection("profileDetail").document(dataMap["email"].toString())
            .set(dataMap).addOnCompleteListener { task ->
            if (task.isSuccessful) callback(true)
        }.addOnFailureListener {
            callback(false)
            showErrorToastMessage(it.message.toString())
        }
    }

    fun updateProfileDetail(email: String,
                            updateData: HashMap<String,Any>,
                            callback: (Boolean) -> Unit) {
        firestore.collection("profileDetail")
            .document(email).update(updateData)
            .addOnCompleteListener { task ->
            if (task.isSuccessful) callback(true)
        }.addOnFailureListener {
            callback(false)
            showErrorToastMessage(it.message.toString())
        }
    }

    private fun showErrorToastMessage(error: String) {
        Toast.makeText(context, error, Toast.LENGTH_LONG).show()
    }

}
```

CalendarViewModel.kt

```
package com.app.ebfitapp.viewmodel

import android.app.Application
import android.widget.Toast
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.MutableLiveData
import com.app.ebfitapp.model.ToDoModel
import com.app.ebfitapp.service.FirebaseAuthService
import com.app.ebfitapp.service.FirebaseFirestoreService

class CalendarViewModel(private val application: Application) : AndroidViewModel(application) {

    val indexExists = MutableLiveData<Boolean>()

    private val firestoreService = FirebaseFirestoreService(application.applicationContext)
    private val firebaseAuthService = FirebaseAuthService(application.applicationContext)
    private val currentEmail = firebaseAuthService.getCurrentUserEmail()
    private                    val                    userDocumentReference                    =
firestoreService.firestore.collection("toDoRecyclerViewItems").document(currentEmail)

    fun addToDoItem(todoArray : ToDoModel, callback: (Boolean) -> Unit) {

        val currentTimeStamp = System.currentTimeMillis()
        val dataMap = hashMapOf(
            "selectedDay" to todoArray.selectedDay,
            "selectedDate" to todoArray.selectedDate,
            "todoText" to todoArray.todoText,
            "todoId" to todoArray.todoId,
            "createdAt" to currentTimeStamp
        )

        userDocumentReference.collection("toDoRecyclerViewItems")
```

```
            .add(dataMap)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    callback(true)
                } else {
                    callback(false)
                }
            }
    }
    fun getToDoItems(callback: (List<ToDoModel>) -> Unit) {
        userDocumentReference.collection("toDoRecyclerViewItems")
            .get()
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    val toDoList = mutableListOf<ToDoModel>()

                    for (document in task.result!!) {
                        val selectedDay = document.getString("selectedDay") ?: ""
                        val selectedDate = document.getString("selectedDate") ?: ""
                        val todoText = document.getString("todoText") ?: ""
                        val todoId = document.getString("todoId") ?: ""
                        val createdAt = document.getLong("createdAt")
                        val    toDoItem    =    ToDoModel(selectedDay,    selectedDate,    todoText,
todoId,createdAt)
                        toDoList.add(toDoItem)
                    }

                    callback(toDoList)
                } else {
                    showErrorToastMessage(task.toString())
                    callback(emptyList())
                }
            }
    }
    fun deleteToDoItem(todoId: String?) {
        if (todoId != null) {
            userDocumentReference.collection("toDoRecyclerViewItems")
                .whereEqualTo("todoId" ,todoId)
                .get()
                .addOnCompleteListener { querySnapshot ->
                    if (querySnapshot.isSuccessful) {
                        for (document in querySnapshot.result!!) {
                            document.reference.delete()
                        }

                    } else {

                    }
                }
        } else {

        }
    }
    private fun showErrorToastMessage(error: String) {
        Toast.makeText(application.applicationContext, error, Toast.LENGTH_LONG).show()
    }
}
```

WorkoutViewModel.kt

```
package com.app.ebfitapp.viewmodel

import android.app.Application
import android.widget.Toast
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.MutableLiveData
import com.app.ebfitapp.model.BestTrainersModel
import com.app.ebfitapp.model.MuscleGroupModel
import com.app.ebfitapp.model.PopularWorkoutsModel
import com.app.ebfitapp.service.FirebaseFirestoreService
```

```
class WorkoutViewModel(private val application: Application) : AndroidViewModel(application) {

    val muscleGroupLiveData = MutableLiveData<ArrayList<MuscleGroupModel>?>()
    val bestTrainersLiveData = MutableLiveData<ArrayList<BestTrainersModel>?>()
    val popularWorkoutsLiveData = MutableLiveData<ArrayList<PopularWorkoutsModel>?>()

    private var muscleTempList: ArrayList<MuscleGroupModel> = ArrayList()
    private var trainersTempList: ArrayList<BestTrainersModel> = ArrayList()
    private var workoutsTempList: ArrayList<PopularWorkoutsModel> = ArrayList()
    private val firestoreService = FirebaseFirestoreService(application.applicationContext)

    fun getMuscleGroups() {
        if (muscleGroupLiveData.value == null) {
            firestoreService.firestore.collection("muscleGroups").get()
                .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    for (snapshot in task.result.documents) {
                        val                             muscleGroupModel                             =
MuscleGroupModel(snapshot.data!!["muscle"].toString(),
                            snapshot.data!!["muscleImage"].toString())
                        muscleTempList.add(muscleGroupModel)
                    }
                    muscleGroupLiveData.value = muscleTempList
                } else {
                    muscleGroupLiveData.value = null
                }
            }.addOnFailureListener {
                muscleGroupLiveData.value = null
                showErrorToastMessage(it.message.toString())
            }
        }

    }

    fun getBestTrainers() {
        if (bestTrainersLiveData.value == null) {
            firestoreService.firestore.collection("bestTrainers").get().addOnCompleteListener   {
query ->
                if (query.isSuccessful) {
                    for (snapshot in query.result.documents) {
                        val                             bestTrainersModel                             =
BestTrainersModel(snapshot.data!!["profileImageURL"].toString(),
snapshot.data!!["username"].toString(), snapshot.data!!["specialization"].toString())
                        trainersTempList.add(bestTrainersModel)
                    }
                    bestTrainersLiveData.value = trainersTempList
                    //trainersTempList.clear()
                } else bestTrainersLiveData.value = null

            }.addOnFailureListener {
                bestTrainersLiveData.value = null
                showErrorToastMessage(it.message.toString())
            }
        }
    }

    fun getPopularWorkouts() {
        if (popularWorkoutsLiveData.value == null) {
            firestoreService.firestore.collection("popularWorkouts").get()
                .addOnCompleteListener { querySnapshot ->
                if (querySnapshot.isSuccessful) {
                    for (snapshot in querySnapshot.result.documents) {
                        val popularWorkouts = PopularWorkoutsModel(
                            snapshot.data!!["name"].toString(),
                            snapshot.data!!["description"].toString(),
                            snapshot.data!!["imageURL"].toString()
                        )
                        workoutsTempList.add(popularWorkouts)
```

```
                    }
                    popularWorkoutsLiveData.value = workoutsTempList
                    //workoutsTempList.clear()
                } else popularWorkoutsLiveData.value = null
            }.addOnFailureListener {
                popularWorkoutsLiveData.value = null
                showErrorToastMessage(it.message.toString())
            }
        }
    }

    private fun showErrorToastMessage(error: String) {
        Toast.makeText(application.applicationContext, error, Toast.LENGTH_LONG).show()
    }
}
```

WorkoutFragment.kt

```
package com.app.ebfitapp.fragment

import android.annotation.SuppressLint
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
import com.app.ebfitapp.R
import com.app.ebfitapp.adapter.BestTrainersAdapter
import com.app.ebfitapp.adapter.MuscleGroupsAdapter
import com.app.ebfitapp.adapter.PopularWorkoutsAdapter
import com.app.ebfitapp.databinding.FragmentWorkoutBinding
import com.app.ebfitapp.model.BestTrainersModel
import com.app.ebfitapp.model.MuscleGroupModel
import com.app.ebfitapp.model.PopularWorkoutsModel
import com.app.ebfitapp.viewmodel.WorkoutViewModel
import com.google.android.material.bottomnavigation.BottomNavigationView

class WorkoutFragment : Fragment() {

    private lateinit var muscleGroups: ArrayList<MuscleGroupModel>
    private lateinit var bestTrainers: ArrayList<BestTrainersModel>
    private lateinit var popularWorkouts: ArrayList<PopularWorkoutsModel>

    private lateinit var bestTrainersAdapter: BestTrainersAdapter
    private lateinit var muscleGroupsAdapter: MuscleGroupsAdapter
    private lateinit var popularWorkoutsAdapter: PopularWorkoutsAdapter

    private val workoutViewModel: WorkoutViewModel by viewModels()
    private lateinit var fragmentWorkoutBinding: FragmentWorkoutBinding

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View? {
        fragmentWorkoutBinding = FragmentWorkoutBinding.inflate(layoutInflater)
        activity?.findViewById<BottomNavigationView>(R.id.bottomNavigation)?.visibility =
View.VISIBLE

        muscleGroups = ArrayList()
        //bestTrainers = ArrayList()
        popularWorkouts = ArrayList()

        //bestTrainersAdapter = BestTrainersAdapter(bestTrainers)
        muscleGroupsAdapter = MuscleGroupsAdapter(muscleGroups)
        popularWorkoutsAdapter = PopularWorkoutsAdapter(popularWorkouts)

        //fragmentWorkoutBinding.bestTrainersRecycler.adapter = bestTrainersAdapter
        fragmentWorkoutBinding.muscleGroupsRecycler.adapter = muscleGroupsAdapter
```

```
        fragmentWorkoutBinding.popularWorkoutsRecycler.adapter = popularWorkoutsAdapter

        getWorkoutAllData()

        return fragmentWorkoutBinding.root
    }

    @SuppressLint("NotifyDataSetChanged")
    private fun observeMuscleGroups() = with(fragmentWorkoutBinding) {
        workoutViewModel.muscleGroupLiveData.observe(viewLifecycleOwner,        Observer        {
liveMuscleGroups ->
            if (liveMuscleGroups != null) {
                muscleGroups.clear()
                muscleGroups.addAll(liveMuscleGroups)
                muscleGroupsRecycler.adapter?.notifyDataSetChanged()
            }
        })
    }

    @SuppressLint("NotifyDataSetChanged")
    private fun observeBestTrainers() = with(fragmentWorkoutBinding) {
        workoutViewModel.bestTrainersLiveData.observe(viewLifecycleOwner,        Observer        {
liveBestTrainers ->
            if (liveBestTrainers != null) {
                bestTrainers.clear()
                bestTrainers.addAll(liveBestTrainers)
                bestTrainersRecycler.adapter?.notifyDataSetChanged()
            }
        })
    }

    @SuppressLint("NotifyDataSetChanged")
    private fun observePopularWorkouts() = with(fragmentWorkoutBinding) {
        workoutViewModel.popularWorkoutsLiveData.observe(viewLifecycleOwner,      Observer      {
livePopularWorkouts ->
            if (livePopularWorkouts != null) {
                popularWorkouts.clear()
                popularWorkouts.addAll(livePopularWorkouts)
                popularWorkoutsRecycler.adapter?.notifyDataSetChanged()
            }
        })
    }

    private fun getWorkoutAllData() {
        workoutViewModel.getPopularWorkouts()
        observePopularWorkouts()
        //workoutViewModel.getBestTrainers()
        //observeBestTrainers()
        workoutViewModel.getMuscleGroups()
        observeMuscleGroups()
    }
}
```

LoginFragment.kt

```
package com.app.ebfitapp.fragment

import android.app.Dialog
import android.content.Intent
import android.graphics.drawable.ColorDrawable
import android.os.Bundle
import android.view.Gravity
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.view.Window
import android.widget.Button
```

```
import android.widget.EditText
import android.widget.Toast
import androidx.navigation.Navigation
import com.app.ebfitapp.R
import com.app.ebfitapp.databinding.FragmentLoginBinding
import com.app.ebfitapp.service.FirebaseAuthService
import com.app.ebfitapp.utils.AppPreferences
import com.app.ebfitapp.utils.CustomProgress
import com.app.ebfitapp.view.MainActivity
import com.google.firebase.auth.FirebaseAuth

class LoginFragment : Fragment() {

    private var email: String? = null
    private var password: String? = null
    private lateinit var appPreferences: AppPreferences
    private lateinit var customProgress: CustomProgress
    private lateinit var firebaseAuthService: FirebaseAuthService
    private lateinit var fragmentLoginBinding: FragmentLoginBinding

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View? {
        fragmentLoginBinding = FragmentLoginBinding.inflate(layoutInflater)

        customProgress = CustomProgress(requireContext())
        appPreferences = AppPreferences(requireContext())
        firebaseAuthService = FirebaseAuthService(requireContext())

        return fragmentLoginBinding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        with(fragmentLoginBinding) {
            onLoginButton.setOnClickListener() {
                email = loginEmailText.text.toString()
                password = loginPasswordText.text.toString()

                if (email.isNullOrEmpty() || password.isNullOrEmpty()) {
                    Toast.makeText(requireContext(),
getString(R.string.please_fill_in_the_empty_fields), Toast.LENGTH_LONG).show()
                } else {
                    customProgress.show()
                    firebaseAuthService.loginAccount(email = email!!, password = password!!) {
task ->
                        if (task) {
                            customProgress.dismiss()
                            checkRememberMe()
                            requireActivity().finish()
                            val intent = Intent(requireContext(), MainActivity::class.java)
                            startActivity(intent)
                        } else {
                            customProgress.dismiss()
                        }
                    }
                }
            }

            forgotPasswordText.setOnClickListener() {
                showDialog()
            }
            createAccountText.setOnClickListener()
            {
                    val action = LoginFragmentDirections.actionLoginFragmentToSignUpFragment()
                    Navigation.findNavController(requireView()).navigate(action)
            }

        }
```

```
    }

    private fun checkRememberMe() {
        if                                  (fragmentLoginBinding.rememberMeCheckBox.isChecked)
appPreferences.saveBoolean("rememberMe", true)
        else appPreferences.saveBoolean("rememberMe", false)
    }

    private fun showDialog() {

        val dialog = Dialog(requireActivity())
        dialog.requestWindowFeature(Window.FEATURE_NO_TITLE)
        dialog.setContentView(R.layout.bottomsheetlayout)

        val resetEmailAddress = dialog.findViewById<EditText>(R.id.resetEmailAddress)
        val resetButton = dialog.findViewById<Button>(R.id.resetButton)

        resetButton?.setOnClickListener {

            if (!resetEmailAddress?.text.isNullOrEmpty()) {
                customProgress.show()
                val sPassword = resetEmailAddress?.text.toString()
                firebaseAuthService.getForgotPassowrd(sPassword, dialog, customProgress)
            } else {
                Toast.makeText(requireContext(),       "Fill       the       blank       please",
Toast.LENGTH_SHORT).show()
            }

        }

        dialog.show()
        dialog.window?.setLayout(
            ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT
        )
        dialog.window?.setWindowAnimations(R.style.DialogAnimation)
        dialog.window?.setGravity(Gravity.BOTTOM)
        dialog.window?.setBackgroundDrawable(ColorDrawable(android.graphics.Color.TRANSPARENT))

    }
}
```

SignupFragment.kt

```
package com.app.ebfitapp.fragment

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.navigation.Navigation
import com.app.ebfitapp.R
import com.app.ebfitapp.databinding.FragmentSignUpBinding
import com.app.ebfitapp.service.FirebaseAuthService
import com.app.ebfitapp.service.FirebaseFirestoreService
import com.app.ebfitapp.utils.CustomProgress

class SignUpFragment : Fragment() {

    private var username: String? = null
    private var email: String? = null
    private var password: String? = null
    private var gender: String? = null

    private lateinit var customProgress: CustomProgress
```

```
    private lateinit var firebaseAuthService: FirebaseAuthService
    private lateinit var firebaseFirestoreService: FirebaseFirestoreService
    private lateinit var fragmentSignUpBinding: FragmentSignUpBinding

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View? {
        fragmentSignUpBinding = FragmentSignUpBinding.inflate(layoutInflater)

        settings()
        customProgress = CustomProgress(requireContext())
        firebaseAuthService = FirebaseAuthService(requireContext())
        firebaseFirestoreService = FirebaseFirestoreService(requireContext())

        return fragmentSignUpBinding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        with(fragmentSignUpBinding) {

            genderDropDown.setOnItemClickListener { adapterView, view, i, l ->
                gender = adapterView.getItemAtPosition(i).toString()
            }

            contiuneButton.setOnClickListener {
                username = usernameText.text.toString()
                email = emailText.text.toString()
                password = passwordText.text.toString()

                if (username.isNullOrEmpty() || email.isNullOrEmpty() || password.isNullOrEmpty()
|| gender.isNullOrEmpty()) {
                    Toast.makeText(requireContext(),
getString(R.string.please_fill_in_the_empty_fields), Toast.LENGTH_LONG).show()
                } else {
                    customProgress.show()
                    firebaseAuthService.createAccount(email = email!!, password = password!!) {
task ->
                        if (task) {
                            firebaseFirestoreService.createProfileDetail(hashMapOf("username"  to
username!!, "email" to email!!, "gender" to gender!!, "profileImageURL" to null, "age" to null,
                                "height"  to  null,  "weight"  to  null,  "targetWeight"  to  null,
"goal" to null)) { callback ->
                                if (callback) {
                                    customProgress.dismiss()
                                    val                          action                          =
SignUpFragmentDirections.actionSignUpFragmentToProfileDetailFragment(email!!)
                                    Navigation.findNavController(requireView()).navigate(action)
                                } else {
                                    customProgress.dismiss()
                                }
                            }
                        } else {
                            customProgress.dismiss()
                        }
                    }
                }
            }

        }

    }

    private fun settings() {
        val    genderItems    =    listOf(getString(R.string.male),    getString(R.string.female),
getString(R.string.prefer_not_to_say))
        val adapter = ArrayAdapter(requireContext(), R.layout.dropdown_items, genderItems)
        fragmentSignUpBinding.genderDropDown.setAdapter(adapter)
    }
```

```
}
```

CalendarFragment.kt

```
package com.app.ebfitapp.fragment

import android.annotation.SuppressLint
import android.app.Dialog
import android.graphics.Color
import android.graphics.drawable.ColorDrawable
import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import kotlin.random.Random
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.view.animation.AlphaAnimation
import android.view.animation.Animation
import android.widget.Button
import android.widget.EditText
import android.widget.ImageView
import android.widget.TextView
import android.widget.Toast
import androidx.fragment.app.viewModels
import androidx.recyclerview.widget.ItemTouchHelper
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.LinearSnapHelper
import androidx.recyclerview.widget.RecyclerView
import androidx.recyclerview.widget.SnapHelper
import com.app.ebfitapp.R
import com.app.ebfitapp.adapter.CalendarAdapter
import com.app.ebfitapp.adapter.CalendarToDoAdapter
import com.app.ebfitapp.databinding.FragmentCalendarBinding
import com.app.ebfitapp.model.CalendarDateModel
import com.app.ebfitapp.model.ToDoModel
import com.app.ebfitapp.service.FirebaseAuthService
import com.app.ebfitapp.utils.CustomProgress
import com.app.ebfitapp.viewmodel.CalendarViewModel
import com.google.android.material.bottomnavigation.BottomNavigationView
import java.text.SimpleDateFormat
import java.util.*
import kotlin.collections.ArrayList

class CalendarFragment : Fragment(), CalendarAdapter.onItemClickListener {

    private lateinit var recyclerView: RecyclerView
    private lateinit var tvDateMonth: TextView
    private lateinit var ivCalendarNext: ImageView
    private lateinit var ivCalendarPrevious: ImageView
    private lateinit var toDoAdapter: CalendarToDoAdapter
    private val calendarViewModel: CalendarViewModel by viewModels()
    private lateinit var firebaseAuthService: FirebaseAuthService
    private lateinit var customProgress: CustomProgress
    var selectedDate: String? = null
    var selectedDay: String? = null
    var isSelected: Boolean = false
    var todoArray = arrayListOf<ToDoModel>()


    private val sdf = SimpleDateFormat("MMMM yyyy", Locale.ENGLISH)
    private val cal = Calendar.getInstance(Locale.ENGLISH)
    private val currentDate = Calendar.getInstance(Locale.ENGLISH)
    private val dates = ArrayList<Date>()
    private lateinit var adapter: CalendarAdapter
    private val calendarList2 = ArrayList<CalendarDateModel>()
```

```
private lateinit var fragmentCalenderBinding: FragmentCalendarBinding


override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    customProgress = CustomProgress(requireContext())
    firebaseAuthService = FirebaseAuthService(requireContext())
    fragmentCalenderBinding = FragmentCalendarBinding.inflate(layoutInflater)
    return fragmentCalenderBinding.root
}


override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    with(fragmentCalenderBinding)
    {
        this@CalendarFragment.tvDateMonth = textDateMonth
        this@CalendarFragment.recyclerView = recyclerView
        this@CalendarFragment.ivCalendarNext = ivCalendarNext
        this@CalendarFragment.ivCalendarPrevious = ivCalendarPrevious
        emptyText.visibility = View.GONE
        setUpAdapter()
        setUpClickListener()
        setUpCalendar()
        isEmptyText.visibility = View.GONE
        observeIndexExists()
        calendarSearch()

        toDoAdapter = CalendarToDoAdapter(todoArray, calendarViewModel)
        todoRecyclerView.layoutManager = LinearLayoutManager(this@CalendarFragment.context)
        todoRecyclerView.adapter = toDoAdapter

        val itemTouchHelper = ItemTouchHelper(toDoAdapter.SwipeToDeleteCallback())
        itemTouchHelper.attachToRecyclerView(todoRecyclerView)

        toDoAdapter.notifyDataSetChanged()

        todoText.setOnClickListener() {
            if (isSelected)
                showToDoDialog()
            else Toast.makeText(
                this@CalendarFragment.context,
                "You should select a day first",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}

override fun onItemClick(text: String, day: String) {
    isSelected = true
    selectedDay = day
    selectedDate = text
    customProgress.show()

    calendarViewModel.getToDoItems { toDoList ->
        val filteredToDoList = if (isSelected && selectedDate != null) {
            toDoList.filter { it.selectedDate == selectedDate }

        } else {
            emptyList()
        }

        if (filteredToDoList.isNotEmpty()) {
            //There is something on the screen
            fragmentCalenderBinding.isEmptyText.visibility = View.GONE
            fragmentCalenderBinding.emptyText.visibility = View.GONE
```

```
                val sortedToDoList = filteredToDoList.sortedBy { it.createdAt }
                val arrayListToDoList = ArrayList(sortedToDoList)

                toDoAdapter.todoArray.clear()
                toDoAdapter.todoArray.addAll(arrayListToDoList)
                toDoAdapter.notifyDataSetChanged()
            } else {
                //There is not item current day is null can be shown
                //if user uses search bar currenday has to be remove and if items show up
                //emptyText should be viewGone otherwise show empty text
                toDoAdapter.todoArray.clear()
                fragmentCalenderBinding.isEmptyText.visibility = View.VISIBLE
                fragmentCalenderBinding.emptyText.visibility = View.GONE
            }

            customProgress.dismiss()
        }

        todoTextAnimation()
    }


    private fun setUpClickListener() {
        ivCalendarNext.setOnClickListener()
        {
            cal.add(Calendar.MONTH, 1)
            setUpCalendar()
        }
        ivCalendarPrevious.setOnClickListener()
        {
            cal.add(Calendar.MONTH, -1)
            if (cal == currentDate)
                setUpCalendar()
            else
                setUpCalendar()
        }
    }

    private fun setUpAdapter() {
        val snapHelper: SnapHelper = LinearSnapHelper()
        snapHelper.attachToRecyclerView(recyclerView)
        adapter  =  CalendarAdapter(fragmentCalenderBinding.recyclerView)  {  calendarDateModel:
CalendarDateModel, position: Int ->
            calendarList2.forEachIndexed { index, calendarModel ->
                calendarModel.isSelected = index == position
            }
            adapter.setData(calendarList2)
            adapter.setOnItemClickListener(this@CalendarFragment)
        }
        recyclerView.adapter = adapter
    }

    private fun setUpCalendar() {
        val calendarList = ArrayList<CalendarDateModel>()
        tvDateMonth.text = sdf.format(cal.time)
        val monthCalendar = cal.clone() as Calendar
        val maxDaysInMonth = cal.getActualMaximum(Calendar.DAY_OF_MONTH)
        dates.clear()
        monthCalendar.set(Calendar.DAY_OF_MONTH, 1)
        while (dates.size < maxDaysInMonth) {
            dates.add(monthCalendar.time)
            calendarList.add(CalendarDateModel(monthCalendar.time))
            monthCalendar.add(Calendar.DAY_OF_MONTH, 1)
        }
        calendarList2.clear()
        calendarList2.addAll(calendarList)
        adapter.setOnItemClickListener(this@CalendarFragment)
        adapter.setData(calendarList)
```

```
    adapter.scrollToPosition()
}


private fun showToDoDialog() {
    val rootView = requireActivity().window.decorView.rootView
    val overlay = View(requireContext())

    overlay.setBackgroundColor(Color.parseColor("#80000000"))
    val layoutParams = ViewGroup.LayoutParams(
        ViewGroup.LayoutParams.MATCH_PARENT,
        ViewGroup.LayoutParams.MATCH_PARENT
    )
    (rootView as ViewGroup).addView(overlay, layoutParams)

    val dialog = Dialog(requireActivity())
    dialog.setContentView(R.layout.todo_dialog_box)
    dialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))
    dialog.window?.setLayout(
        ViewGroup.LayoutParams.MATCH_PARENT,
        ViewGroup.LayoutParams.WRAP_CONTENT
    )
    dialog.setCancelable(false)

    val dialogDate = dialog.findViewById<TextView>(R.id.dialogDate)
    val dialogDay = dialog.findViewById<TextView>(R.id.dialogDay)

    selectedDay?.let {
        dialogDay.text = it
    }

    selectedDate?.let {
        dialogDate.text = it
    }
    val dialogEditText = dialog.findViewById<EditText>(R.id.dialogEditText)
    val dialogCancelBtn = dialog.findViewById<Button>(R.id.dialogCancelBtn)
    val dialogSaveBtn = dialog.findViewById<Button>(R.id.dialogSaveBtn)
    dialog.show()
    dialogCancelBtn.setOnClickListener {
        rootView.removeView(overlay)
        dialog.dismiss()
    }

    dialogSaveBtn.setOnClickListener {
        val dialogEditText = dialogEditText.text.toString()
        val uniqueId = UUID.randomUUID().toString().substring(0, 12)
        val currentTimeStamp = System.currentTimeMillis()
        val newTodoItem =
            ToDoModel(selectedDay, selectedDate, dialogEditText, uniqueId, currentTimeStamp)
        calendarViewModel.addToDoItem(newTodoItem) { isSuccess ->
            if (isSuccess) {
                fragmentCalenderBinding.isEmptyText.visibility = View.GONE
                toDoAdapter.todoArray.add(newTodoItem)
                toDoAdapter.notifyDataSetChanged()
                rootView.removeView(overlay)
                dialog.dismiss()
            } else {
            }
        }
    }
}


private fun calendarSearch() = with(fragmentCalenderBinding) {
    calendarSearchView.addTextChangedListener(object : TextWatcher {
        override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) { }

        override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) { }
```

```
            @SuppressLint("NotifyDataSetChanged")
            override fun afterTextChanged(p0: Editable?) {
                val searchText = p0.toString().trim()

                if (searchText.isNotEmpty()) {
                    calendarViewModel.getToDoItems { toDoList ->
                        val filteredToDoText = if (isSelected)
                            toDoList.filter  {  it.todoText!!.contains(searchText,  ignoreCase  =
true) }
                        else arrayListOf()

                        if (filteredToDoText.isNotEmpty()) {
                            emptyText.visibility = View.GONE
                            isEmptyText.visibility = View.GONE

                            val sortedToDoList = filteredToDoText.sortedBy { it.todoText }
                            val arrayListToDoList = ArrayList(sortedToDoList)

                            if (toDoAdapter.todoArray != arrayListToDoList) {
                                toDoAdapter.todoArray.clear()
                                toDoAdapter.todoArray.addAll(arrayListToDoList)
                                toDoAdapter.notifyDataSetChanged()
                            }
                        } else {
                            toDoAdapter.todoArray.clear()
                            toDoAdapter.notifyDataSetChanged()
                            emptyText.visibility = View.VISIBLE
                            isEmptyText.visibility = View.GONE
                        }

                        customProgress.dismiss()
                    }
                } else {
                    if (toDoAdapter.todoArray.isNotEmpty()) {
                        toDoAdapter.todoArray.clear()
                        toDoAdapter.notifyDataSetChanged()
                    }

                    emptyText.visibility = View.VISIBLE
                    isEmptyText.visibility = View.GONE
                }
            }
        })
    }



    private fun observeIndexExists() {
        calendarViewModel.indexExists.observe(
            viewLifecycleOwner,
            androidx.lifecycle.Observer { indexExists ->
                if (!indexExists){
                    fragmentCalenderBinding.isEmptyText.visibility = View.VISIBLE
                }
                else fragmentCalenderBinding.isEmptyText.visibility = View.GONE
            })
    }

    private fun todoTextAnimation()
    {
        val blinkAnimForToDoText = AlphaAnimation(1f,0f)
        blinkAnimForToDoText.duration = 500
        blinkAnimForToDoText.repeatMode = Animation.REVERSE
        blinkAnimForToDoText.repeatCount = Animation.INFINITE

        fragmentCalenderBinding.todoText.startAnimation(blinkAnimForToDoText)
    }
}
```

fragment_calendar.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    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"
    android:background="@color/gray"
    android:orientation="vertical"
    android:padding="10dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <ImageView
            android:id="@+id/iv_calendar_previous"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="20dp"
            android:src="@drawable/left_arrow" />

        <TextView
            android:id="@+id/text_date_month"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:fontFamily="@font/revolutiongothic_extra_bold"
            android:gravity="center"
            android:text="December 2020"
            android:textColor="#FF0010"
            android:textSize="18dp" />

        <ImageView
            android:id="@+id/iv_calendar_next"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="20dp"
            android:src="@drawable/right_arrow" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recyclerView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:orientation="horizontal"
            app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
            tools:listitem="@layout/date_layout" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/todoText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
```

```
            android:fontFamily="@font/revolutiongothic_extra_bold"
            android:text="TODO"
            android:textColor="#FF0010"
            android:textSize="24sp" />
        <EditText
            android:id="@+id/calendarSearchView"
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:layout_marginStart="15dp"
            android:background="@drawable/round_6dp_background"
            android:backgroundTint="#48000000"
            android:drawableStart="@drawable/baseline_search_24"
            android:drawablePadding="8dp"
            android:drawableTint="@color/light_gray"
            android:hint="@string/search"
            android:inputType="text"
            android:maxLines="1"
            android:paddingVertical="8dp"
            android:paddingStart="8dp"
            android:paddingEnd="8dp"
            android:textColor="@color/white"
            android:textColorHint="@color/light_gray" />

    </LinearLayout>

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/todoRecyclerView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="5dp"
            android:layout_marginHorizontal="10dp"
            android:orientation="vertical"
            tools:listitem="@layout/item_calendar_to_do" />

        <TextView
            android:id="@+id/isEmptyText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginBottom="30dp"
            android:gravity="center"
            android:text="@string/todo_list_is_empty_for_current_day"
            android:textColor="@color/silver"
            android:textSize="24sp" />

        <TextView
            android:id="@+id/emptyText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:gravity="center"
            android:visibility="gone"
            android:text="@string/no_results_found_according_to_your_search"
            android:textColor="@color/silver"
            android:textSize="18sp" />
    </FrameLayout>
</LinearLayout>
```

Додаток Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ПЕРСОНАЛЬНОГО МЕДИЧНОГО КОНСУЛЬТУВАННЯ НА ПЛАТФОРМІ ANDROID З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ KOTLIN І FIREBASE

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська дипломна робота на тему:
«Розробка мобільного застосунку для персонального медичного консультування на платформі Android з використанням технологій Kotlin і Firebase»

Автор: ст. гр. ЗПІ-20б Лейбак Д.В.
Науковий керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.

Вінниця - 2024

Рисунок Г.1 – Титульний слайд

Актуальність теми

- **Зростаючий попит на медичні послуги**: Мобільні застосунки допомагають задовольнити підвищений попит на медичні консультації без необхідності фізичного відвідування лікарень.
- **Розвиток технологій**: Сучасні мобільні пристрої дозволяють інтегрувати функції для точнішої діагностики та моніторингу здоров'я.
- **Персоналізація медичних послуг**: Мобільні застосунки аналізують дані користувачів для надання персоналізованих рекомендацій та підвищення ефективності лікування.
- **Підвищення обізнаності пацієнтів**: Застосунки надають інформацію про здоров'я, рекомендації щодо здорового способу життя, що підвищує обізнаність і самостійність користувачів.

Рисунок Г.2 – Актуальність теми

Мета, об’єкт та предмет дослідження

Основною метою є створення інструментів, які допоможуть вдосконалити процес видання рекомендацій користувачу, щодо тренувань, залежно від вказаних протипоказань.

Об’єкт дослідження – процес розробки програмних засобів мобільного застосунку в галузі охорони здоров’я.

Предмет дослідження – програмні засоби та алгоритми для реалізації мобільного застосунку персонального консультанта в галузі охорони здоров’я.

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

Задачі дослідження

- здійснити аналіз стану питання та методів розв’язання задачі;
- розробити архітектуру та загальний алгоритм роботи системи;
- розробка модуля графічного інтерфейсу, призначеного для відображення контенту та його управлінням;
- розробка модуля для зберігання користувацьких даних у базу даних;
- розробка модуля для створення власних записів та їх управлінням;
- розробка модуля для відображення та використання планів тренувань;
- розробка модуля відбору планів тренувань, залежно від вказаних протипоказань;
- проведення тестування системи;
- розробити інструкцію користувача.

Рисунок Г.4 – Задачі дослідження

Порівняльний аналіз аналогів

	AdaHealth	MyFitnessPal	Fitbit	Власна розробка
Персоналізовані консультації вправ	-	-	+	+
Доступ до даних через віджети робочого столу	-	-	+	+
Конфіденційність та безпека	+	-	-	+
Ведення історії	-	+	+	+
Створення нотаток	-	+	+	+
	2	2	3	5

Рисунок Г.5 – Порівняльний аналіз аналогів

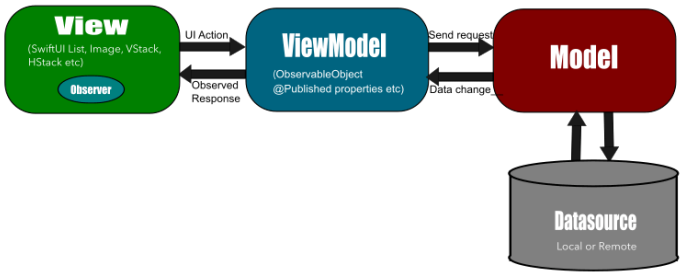
Новизна і практична цінність отриманих результатів

Подальшого розвитку отримав алгоритм відбору планів тренувань персонально для користувача мобільного застосунку, відповідно до вказаних ним протипоказань.

Подальшого розвитку отримав алгоритм створення власних записів користувача щодо тренувань або нагадувань про прийом ліків та ведення історії тренувань користувача.

Практична цінність отриманих результатів полягає в розробці програмних засобів мобільного застосунку, який надає користувачам покращити можливості використання планів тренувань, вести їх історію, створювати нотатки або нагадування.

Рисунок Г.6 – Новизна і практична цінність отриманих результатів



Архітектура мобільного застосунку

Рисунок Г.7 – Архітектура мобільного застосунку

Блок-схема алгоритму створення рекомендацій

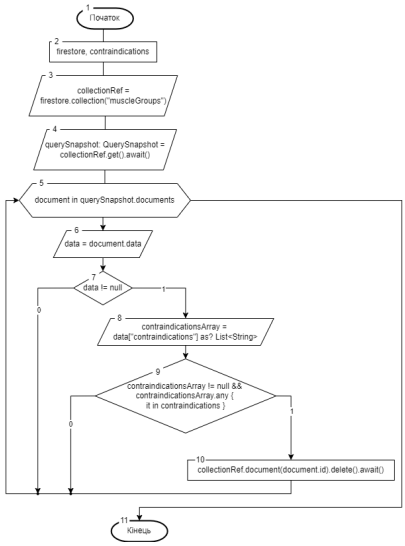


Рисунок Г.8 – Блок-схема алгоритму створення рекомендацій

Блок-схема алгоритму створення нотаток

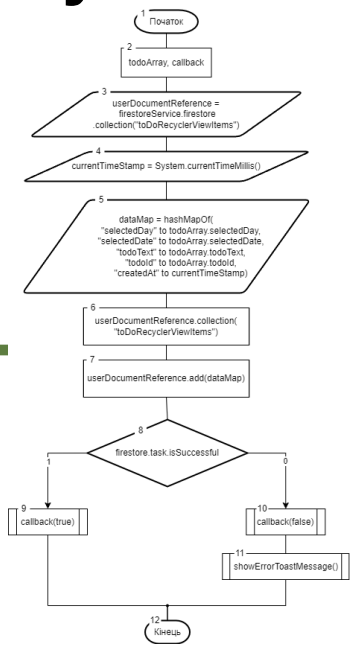


Рисунок Г.9 – Блок-схема алгоритму створення нотаток

Екрани реєстрації та авторизації

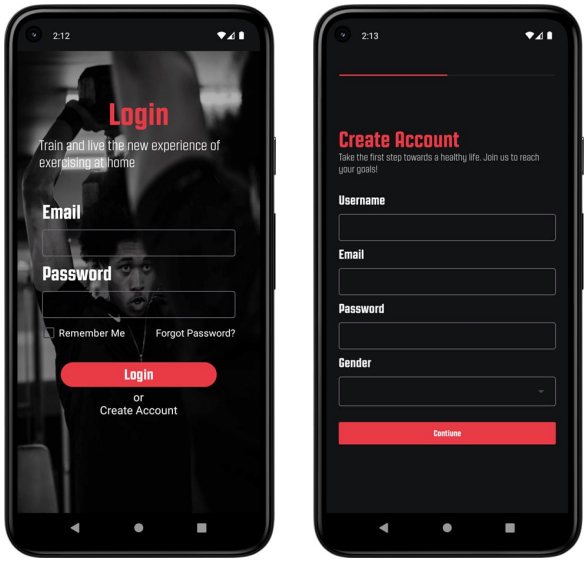


Рисунок Г.10 – Екрани реєстрації та авторизації

Екрани вибору протипоказань

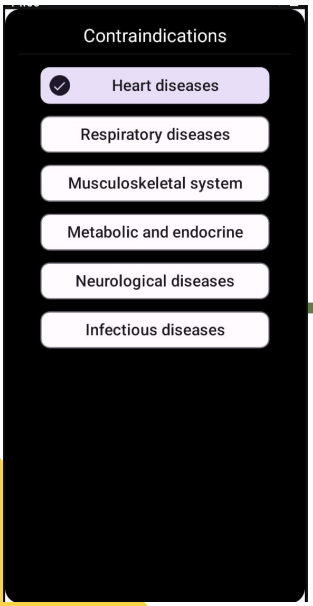


Рисунок 11 – Екран вибору протипоказань

Домашній екран застосунку

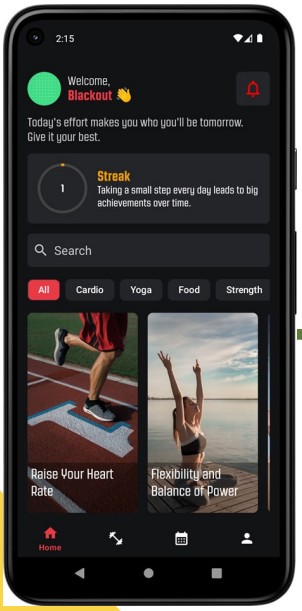


Рисунок Г.12 – Домашній екран

Екран «Workout»

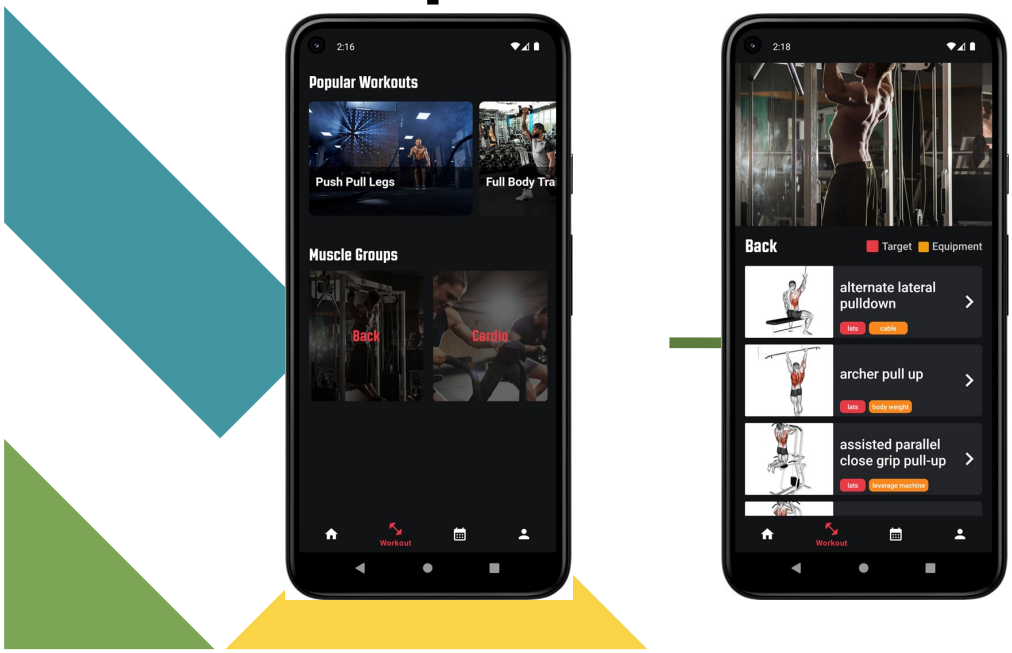


Рисунок Г.13 – Екран «Workout»

Екрани проведення тренування

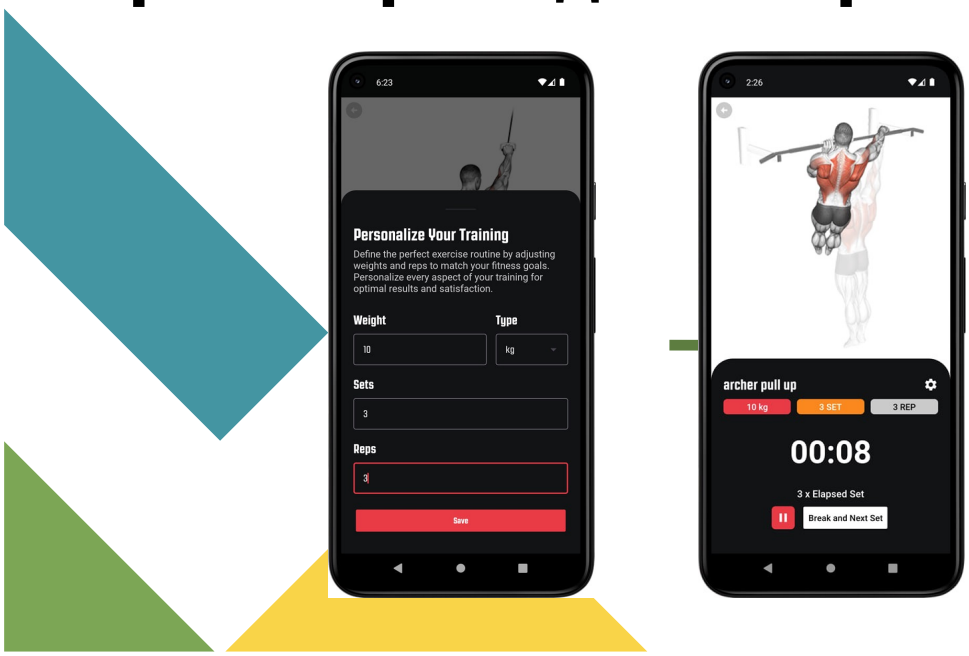


Рисунок Г.14 – Екрани проведення тренувань

Екрани створення нотаток

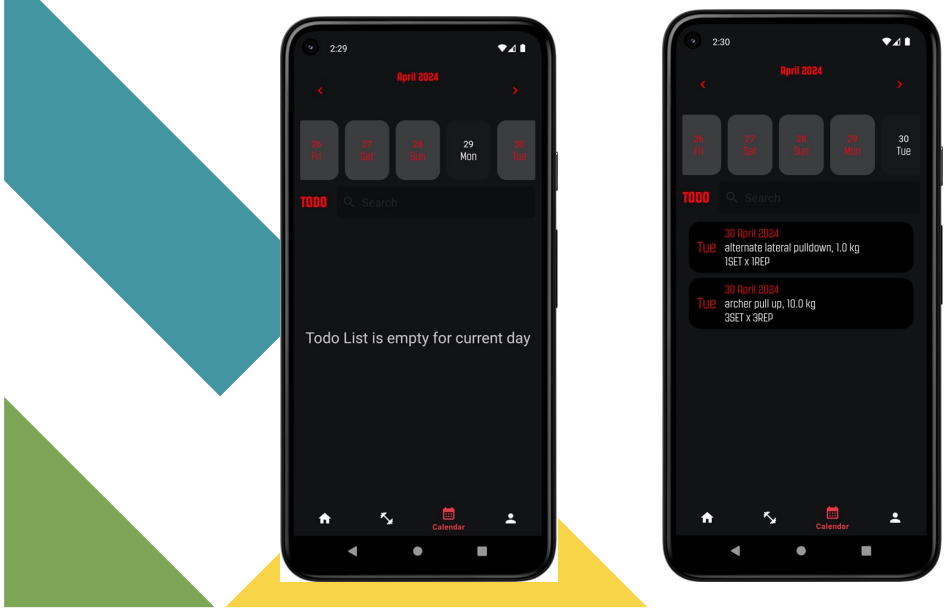


Рисунок Г.15 – Екрани створення нотаток

Тестування мобільного застосунку

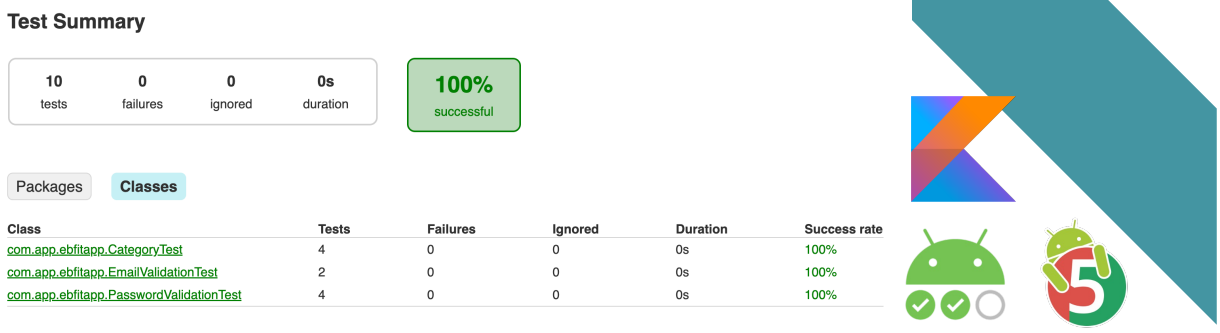


Рисунок Г.16 – Тестування мобільного застосунку

Апробація та публікація матеріалів бакалаврської роботи

Проблеми та перспективи використання програмних застосунків у галузі охорони здоров'я. Лейбак Д.В., Катєльніков Д.І. (Вінницький національний технічний університет).

За тематикою дослідження опублікована 1 наукова праця у збірниках матеріалів конференції.

Рисунок Г.17 – Апробація та публікація матеріалів бакалаврської роботи

Дякую за увагу!

Рисунок Г.18 – Фінальний слайд