

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему: ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ-
ТРЕКЕРУ НАПАДУ МІГРЕНІ У ЛЮДЕЙ
SOFTWARE IMPLEMENTATION OF A MIGRAINE
ATTACKS TRACKER

Виконав: студент(ка) 4 курсу, групи КНТ-128
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Колесникова М.В.

(прізвище та ініціали)

Керівник Федорончак Т.В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (КОД І НАЙМЕНУВАННЯ)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Колесникової Марини Віталіївни

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація мобільного застосунку-трекеру нападу мігрені у людей Software Implementation of a Migraine Attacks Tracker
 керівник проєкту (роботи) Федорончак Тетяна Василівна, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 13 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка завдання. 2. Опис програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

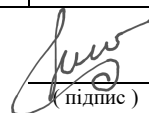
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Федорончак Т.В., доцент		
Нормоконтроль	Белова А.В., асистент		

7. Дата видачі завдання “28” лютого 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 – 3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми	5 – 6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи	9 тиждень	

Студент(ка)



Колесникова М.В.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Федорончак Т.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
102 с., 52 рис., 16 табл., 3 дод., 13 джерел.

НАПАД МІГРЕНІ, МОБІЛЬНИЙ ЗАСТОСУНОК, FLUTTER, DART,
CLOUD FIREBASE, ПОГОДА, ТРЕКЕР, BLOC.

Об'єкт дослідження – облік нападів мігрені.

Предмет дослідження – кроссплатформний мобільний застосунок-трекер нападів мігрені у людей «MigTracker».

Мета роботи – створення кроссплатформного мобільного застосунку-трекеру «MigTracker» для обліку нападів мігрені у людей.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, кроссплатформний фреймворк Flutter, мультипарадигмальна мова програмування Dart, сховище Cloud Firestore, персональний комп'ютер Mac mini (M1, 2020), операційна система macOS Monterey.

У результаті було розроблено кроссплатформний застосунок-трекер нападів мігрені у людей.

Висновки. Розроблений програмний продукт дозволяє вести облік нападів мігрені у зручний спосіб за допомогою смартфона, отримувати поради та передбачення, конвертувати записи у PDF.

Галузь використання – користувачі смартфонів, що страждають на напади мігрені.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 102 pages, 52 figures, 16 tables, 3 appendices, 13 sources.

MIGRAINE ATTACK, MOBILE APPLICATION, FLUTTER, DART, CLOUD FIREBASE, WEATHER, TRACKER, BLOC.

Object of the study is the tracking of migraine attacks.

Subject of the study is the cross-platform mobile application migraine attacks-tracker "MigTracker".

The aim of the work is creation of the cross-platform mobile application-tracker "MigTracker" for tracking migraine attacks in humans.

Materials, methods and technical means: object-oriented programming, cross-platform Flutter framework, multi-paradigm programming language Dart, Cloud Firestore storage, personal computer Mac mini (M1, 2020), macOS Monterey operating system.

Results. Cross-platform application of a migraine attack tracker in humans has been developed.

Conclusions. The developed software product allows you to keep track of migraine attacks in a convenient way with the help of a smartphone, gettind advices and predictions, convert entries to PDF.

Field of use - smartphone users suffering from migraine attacks.

ЗМІСТ

С.

Перелік скорочень та умовних позначок	8
1 Аналіз проблеми та постановка завдань роботи	10
1.1 Аналіз предметної області	10
1.2 Аналіз аналогів	10
1.2.1 Migraine Buddy	10
1.2.2 Migraine and headache diary	12
1.2.3 Migraine Monitor	13
1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів	14
1.3 Постановка завдання	16
1.4 Висновки за 1 розділом	17
2 Опис програми	18
2.1 Функціональні вимоги до програми	18
2.2 Вибір та обґрунтування програмних ресурсів	19
2.3 Вибір та обґрунтування сховища для зберігання даних	21
2.4 Керування станом	23
2.5 Архітектура застосунку	25
2.7 Висновки за 2 розділом	28
3 Розробка програми	30
3.1 Алгоритм функціонування розробленої програми	30
3.1.1 Алгоритм авторизації	31
3.1.2 Алгоритм реєстрації	31
3.1.3 Алгоритм авторизації та реєстрації за допомоги сторонніх сервісів	32
3.1.4 Алгоритм пошуку записів	33
3.1.5 Сортування записів	34
3.1.6 Алгоритм створення запису	34

3.1.7 Алгоритм підтвердження завершення нападу	36
3.1.8 Алгоритм редагування запису	37
3.1.9 Алгоритм видалення запису	38
3.1.10 Алгоритми відображення записів у календарю	38
3.1.11 Алгоритм передбачення нападу	39
3.1.12 Алгоритм конвертації нападів у pdf формат	42
3.2 Архітектура застосунку на основі предметної області	42
3.3 Зберігання даних	51
3.4 Інтерфейс застосунку	58
3.5 Висновки за 3 розділом	62
4 Експлуатація, тестування та експрементальне дослідження програми	63
4.1 Призначення та умови виконання програми	63
4.2 Характеристики програми	64
4.3 Звернення для програми	65
4.3 Тестування	69
4.4 Висновки за 4 розділом	73
Висновок	74
Перелік посилань.	75
Додаток А	77
Додаток Б	82
Додаток В	93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

BLoC – Business Logic Component;

PDF – Portable Data File;

REST - REpresentational State Transfer;

SDK – Software Development Kit;

SQL – Structured Query Language;

БД – база даних;

ОС – операційна система;

ПЗ – програмний засіб.

ВСТУП

За даними Всесвітньої Організації Охорони Здоров'я близько 4% дорослих людей страждають на регулярні напади мігрені [1]. Ця хвороба спричиняє головний біль, порушення координації, зору та погіршує якість життя людини.

Через те, що причиною мігрені можуть бути різні чинники, є необхідним вести облік нападів, занотовувати симптоми, передумови та ліки. Такий щоденник мігренів є рекомендованим лікарями для виявлення тригерів нападу та ефективності вживаних ліків.

Найзручніший спосіб вести записи із допомоги смартфона, що завжди знаходиться під рукою. Саме тому актуальності набувають застосунки-трекери, що передбачають інтерфейс для зручного занотовування та перегляду записів. та мають єдину структуру запису, що полегшує аналіз даних.

Метою даного проекту є створення кросплатформного мобільного застосунку «MigTracker» для:

- створення та зберігання докладних записів нападів мігрені;
- перегляду записів у вигляді списку або у календарі;
- діаграми для відслідковування найбільш вживаних препаратів, тривалості нападів, найчастіших симптомів;
- надання рекомендацій щодо полегшення та попередження нападів.

Всі необхідні дані будуть зберігатися у віддаленому хмарному сховищі Firestore Database.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОБОТИ

1.1 Аналіз предметної області

Розробка застосунку, який присвячено такому напрямку як мігрень передбачає аналіз його предметної області, до якого входять специфічні поняття: напад мігрені, симптом, лікування, локалізація болю та тригер.

Напад мігрені - напад головного болю, що має пульсуючий характер, та може супроводжуватись іншими симптомами.

Симптом - характерний прояв певного захворювання [2].

Лікування - засоби, медичні препарати та заходи, що сприяють полегшенню симптомів та припиненню захворювання [2]. Для того, щоб визначити ефективність препаратів необхідно зазначити їх у записі атаки.

Знання про локалізацію допоможе лікарю визначити тип головного болю та його причину, у тому числі й відрізнити мігрень від іншого захворювання. Тому під локалізацією болю маються на увазі певні області голови пацієнта.

Через те, що головний біль може бути спричинений зовнішніми факторами - тригерами: шумом, нестачею повітря, вживанням алкоголю чи фізичним впливом, їх потрібно занотовувати для того, щоб уникати.

У роботі також будуть використовуватись поняття рівня болю та рівня стресу, які визначають ступінь величини болю та стресу, який відчуває користувач застосунку перед або під час нападу.

1.2 Аналіз аналогів

1.2.1 Migraine Buddy

Застосунок Migraine Buddy (рис. 1.1) - комплексний застосунок для відслідковування та вивчення головного болю [3].

Migraine Buddy є застосунком-трекером, нападів мігрені та сну.

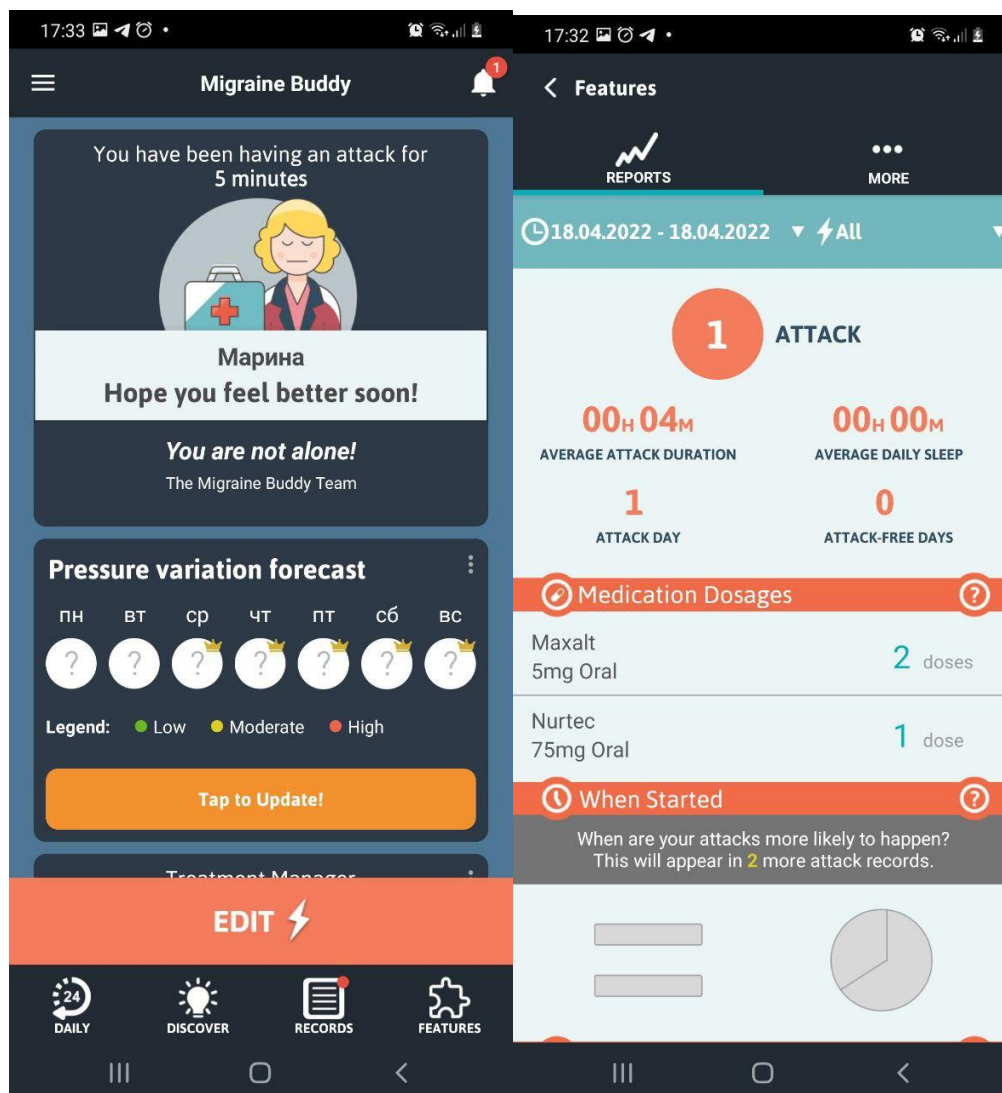


Рисунок 1.1 - Застосунок Migraine Buddy

Основні переваги сервісу:

- трекінг нападів мігрені включаючи тригери, симптоми медичні препарати, локалізацію болю та інтенсивність;
- нагадування про візит до лікаря та прийом ліків;
- новини та програми, що стосуються лікування мігрені.

Недолікі:

- перенасичений інтерфейс, що ускладнює користування застосунком;
- немає погодних умов у конструкторі записів;

- дизайн застосунку має світлі сторінки та яскраві елементи, що можуть викликати додатковий дискомфорт під час нападів у людей зі світлочутливістю;
- лише англійська мова застосунку.

1.2.2 Migraine and headache diary

Migraine and headache diary (рис. 1.2) - простий інструмент для реєстрації нападів мігрені [4].

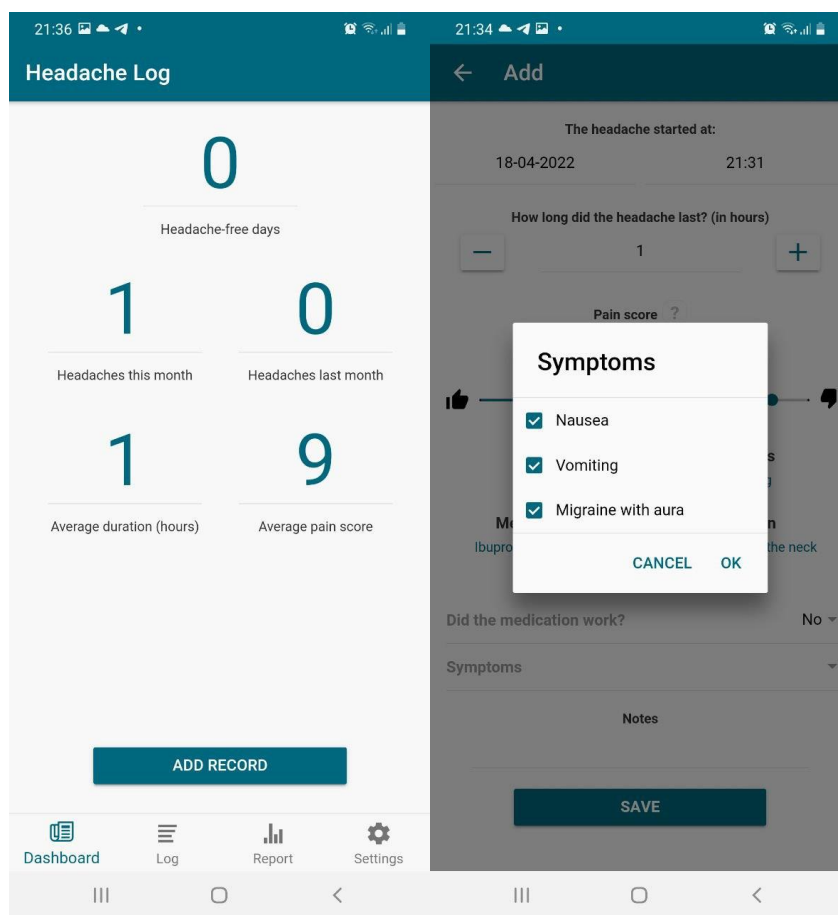


Рисунок 1.2 - Застосунок Migraine and headache diary

Основні переваги застосунку:

- простий, зрозумілий інтерфейс.

Недоліки:

- світлий дизайн;
- відсутній інтерактивний вибір локалізації головного болю;
- відсутність реєстрації
- відсутність збереження записів у віддаленому сховищі;
- мале число симптомів (3 симптоми);
- відсутність візуалізації статистики (графіків частоти виникнення нападів);
- лише англійська мова застосунку;
- погодні умови зазначаються без деталізації.

1.2.3 Migraine Monitor

Migraine Monitor (рис. 1.3) - застосунок для моніторингу нападів мігрені та відправки статистики особистому лікарю.

Основні переваги застосунку:

- автоматичне заповнення погодних умов;
- графічне зображення статистики;
- можливість надсилати статистику лікарю;
- зручний конструктор записів;
- можливість генерувати pdf файл.

Недоліки:

- дизайн застосунку має світлі сторінки та яскраві елементи, що можуть викликати додатковий дискомфорт;
- лише англійська мова застосунку.

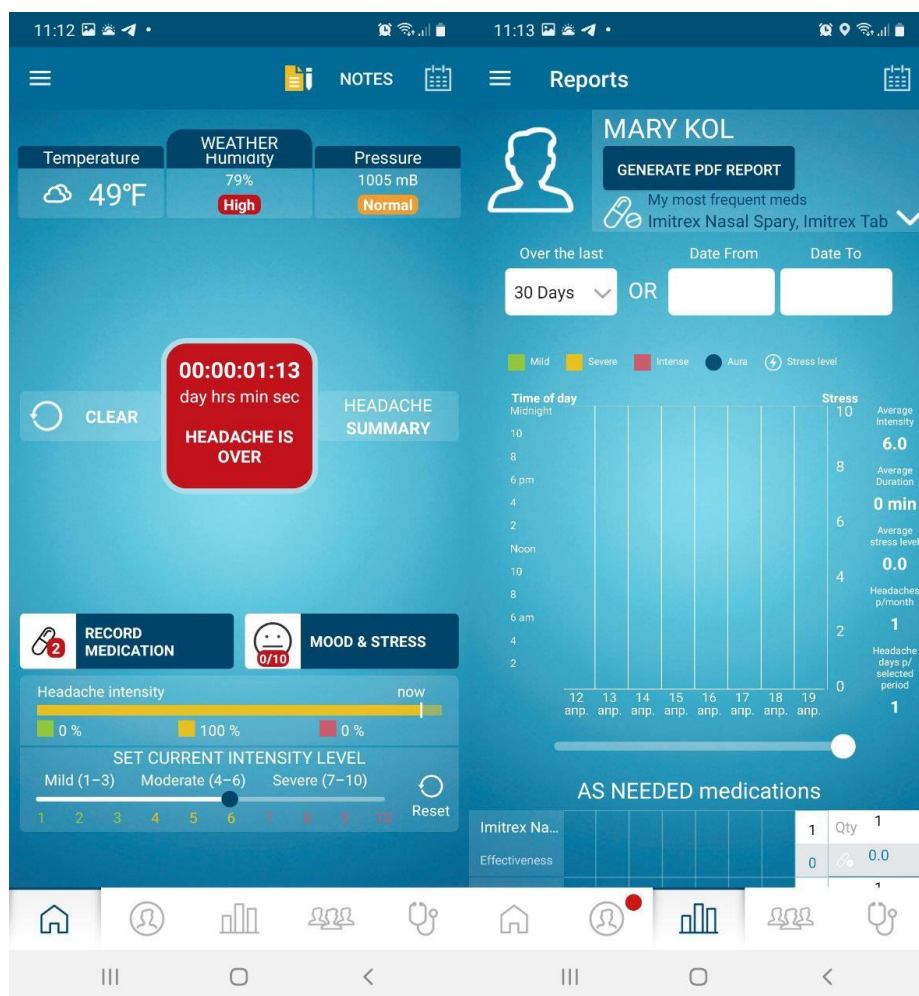


Рисунок 1.3 - Застосунок Migraine Monitor

1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів

Проаналізувавши існуючі аналоги, усі вони дозволяють створювати та зберігати записи про напади мігрені та отримувати статистичну інформацію.

Особливу увагу привернув застосунок Migraine Monitor, що суттєво відрізняється від аналогів своїм широким функціоналом. З нього можна перейняти такі функції, як автоматичне отримання погодних умов згідно гео-локації та конвертація записів у PDF форматі.

Застосунок, що планується розробляти враховує слабкі та сильні сторони

аналогів та буде мати наступні переваги:

- темна кольорова схема застосунку;
- наявність української мови;
- простий, не перенасичений конструктор записів;
- автоматичне визначення та співставлення погодних умов;
- конвертація записів у формат PDF;
- сортування записів.

В табл. 1.1 проведено порівняння вже існуючих програмних продуктів та програми «MigTracker», яка буде розроблена в ході виконання роботи.

Таблиця 1.1 – Порівняння аналогів

Критерій	Migraine Buddy	Migraine and headache diary	Migraine Monitor	MigTracker
1	2	3	4	5
Наявність конструктору для створення записів	+	+	+	+
Кросплатформність (Android та IOS)	+	+	+	+
Темна кольорова схема	±	-	-	+
Мультимовність	-	-	-	+
Сортування записів (дата, інтенсивність болю, тощо)	-	-	-	+
Конвертація у PDF	-	-	+	+
Представлення записів у вигляді календаря	+	+	+	+
Зрозумілий, не перенасичений інтерфейс	-	+	±	+

Продовження таблиці 1.1

1	2	3	4	5
Автоматичне зіставлення погодних умов	-	-	+	+
Прогнозування нападу на основі погодних умов	-	-	-	+
Візуалізація статистики	+	-	+	+

1.3 Постановка завдання

Виходячи з проаналізованих аналогів застосунок виконувати наступні функції:

- створення, редагування та зберігання записів про напади мігрені з єдиною структурою;
- візуалізація статистичних даних з тривалості нападів, найчастіших тригерів, найбільш застосовуваних ліків;
- відображення записів у вигляді списку;
- відображення записів у календарі;
- конвертація записів до файлу формату PDF;
- сортування записів;
- надання рекомендації користувачу щодо полегшення симптомів;
- можливість передбачення нападів за погодними умовами;
- можливість зміни мови застосунку.

Застосунок буде розроблено для мобільних девайсів з операційними системами (OC) Android чи IOS.

1.4 Висновки за першим розділом

В першому розділі було розглянуто проблемну область застосунку, проаналізовано існуючі аналоги програм, таких як: «Migraine Monitor», «Migraine and headache diary» та «Migraine Buddy».

Загалом, існуючі аналоги мають структуровані конструктори для трекінгу, графічне зображення статистики за частотою виникнення та найрозповсюдженіші тригери, відображення записів у списку та у вигляді календаря.

В ході аналізу було виявлено ряд недоліків: яскравий дизайн, що може викликати дискомфорт у людей з фоточутливістю, відсутність локалізації та неповний збір інформації, що відноситься до причин виникнення головного болю (погодні умови, симптоми).

Аналоги було порівняно з застосунком «MigTracker» та зроблено висновок, що розробка цього мобільного застосунку є актуальною та доцільною. Це доведено низкою функцій, що буде передбачати «MigTracker» на відміну від аналогів.

2 ОПИС ПРОГРАМИ

2.1 Функціональні вимоги до програми

До мобільного застосунку висунуті наступні функціональні вимоги:

- реєстрація та авторизація із допомоги поштової адреси або із допомоги сервісів Google та Facebook;
- створення запису про напад мігрені із допомоги конструктору до якого входять наступні параметри: дата та час початку, тривалість, симптоми, рівень болю, локалізація болю, застосовані медичні засоби, погодні умови, рівень стресу, кількість годин сну, тригери (вживання алкоголю, травма голови, вплив шуму, вплив задухи);
- зберігання записів у віддаленому сховищі;
- перегляд записів у вигляді списку з можливістю сортування за датою, за кількістю симптомів, за рівнем болю;
- перегляд записів у вигляді календарю;
- створення PDF файлів з записами про атаки мігрені за місяць та за весь час;
- редагування та видалення записів про атаки мігрені;
- відображення графіків частоти виникнення нападів, найчастіших тригерів, найбільш застосовуваних ліків;
- передбачення нападів за погодними умовами;
- відображення рекомендації користувачу щодо полегшення симптомів;
- зміна мови застосунку;
- вихід з облікового запису.

Наведено діаграму прецедентів користувача (рис. 2.1).



Рисунок 2.1 - Діаграма прецедентів користування застосунком клієнта

2.2 Вибір та обґрунтування програмних ресурсів

Для виконання поставленого завдання було проаналізовано існуючі фреймворки (табл. 2.1) та обрано універсальний фреймворк Flutter. Цей набір засобів розробки та фреймворк підходить для створення мобільних застосунків як для платформи Android так і для IOS, а також веб та десктоп з ОС Windows. Крім того Flutter підтримує технологію швидкого перезавантаження, що

мінімізує час очікування обробки змін коду.

Таблиця 2.1 – Порівняння фреймворків

Критерій	Xamarin	React Native	Flutter
Android, IOS розробка	+	+	+
Продуктивність на рівні нативної	-	-	+
Повністю безкоштовний	-	+	+
Засоби для швидкісної розробки (hot reload)	-	+	+
Оптимізація коду та продуктивності	-	+	+
IDE	Visual Studio	Atom	IntelliJ idea, Android Studio, Visual Studio
Мова програмування	C#	Java Script	Dart

Сильною стороною Flutter є засоби створення користувацького інтерфейсу однакового на різних платформах, продуктивність на рівні нативної та наявність потужних засобів розробки.

Також, було обрано об'єктно-орієнтовану мову програмування Dart, тому що вона:

- кросплатформна;
- підтримує потоки;
- оптимізована для створення користувацьких інтерфейсів;
- структурована та гнучка;

- висока продуктивність одержуваних програм;
- наявність великої кількості готових інструментів для підтримки розробки мовою Dart;
- підтримка бібліотек, що дозволяють, як використовувати готові рішення, так і створювати окремо власні, та підключати їх до проекту;
- Flutter було розроблено саме для Dart.

У якості середовища розробки обрано Android Studio базуючись на результатах порівняльної табл. 2.2.

Таблиця 2.2 – Порівняння середовищ розробки

Критерій	Android Studio	IntelliJ Idea	Visual Studio Code
Призначений для мобільної розробки	+	-	-
Підтримка системи контролю версій	+	+	+
Пропонує способи виправлення помилок	+	+	-
Безкоштовний	+	-	+
Вбудований відлагоджувач	+	+	-
Вбудований емулятор смартфона	+	-	-
Авто-доповнення коду	+	+	+

В результаті порівняння середовищ розробки лідером стала Android Studio, завдяки наданню інструментів саме для мобільної розробки.

2.3 Вибір та обґрунтування сховища для зберігання даних

Для зберігання та обміну даними, необхідним є створення серверу або

налаштування хмарної бази даних (БД).

При виборі віддаленого сховища вибір впав на сервіси корпорації «Google» через те, що вона також є розробником фреймворку Flutter, що означає наявність стабільного інтерфейсу між ними.

У якості сервісу з надання вільних сховищ обрано Google Firebase, який має два типи сховищ: FirestoreDatabase та RealtimeDatabase. Їх порівняння наведено у табл. 2.3.

Таблиця 2.3 - Порівняння сховищ FirestoreDatabase та RealtimeDatabase

Критерій	Firestore Database	Realtime Database
Модель даних	Дані зберігаються у вигляді документів, що схожі за структурою на json.	Зберігає дані у дереві json.
Оффлайн підтримка	Підтримується на Android, Apple та web клієнтах	Підтримується на Android та Apple клієнтах
Запити	Індексовані запити з можливістю складного сортування та фільтрації	Глибокі запити, що повертають все піддерево, з обмеженою можливістю сортування та фільтрації
Продуктивність та надійність	БД реплікується у різних регіонах, що робить її більш стійкою до збоїв, а також підтримує масштабованість.	Репліки БД зберігаються у одному регіоні. Низька затримка відповіді підходить для програм з частою синхронізацією.
Автоматична масштабованість	+	-
Можливість відстеження клієнта онлайн	-	+

Отже, порівнюючи ці БД можна зробити наступний висновок: Realtime

Database підходить для застосунків, що знаходяться в одному географічному регіоні, потребують частих транзакцій та відстеження статусу клієнта. Не потребують складних запитів з фільтрацією та сортуванням. Тому, для застосунку MigTracker було обрано Firestore Database через можливості автоматичної масштабованості, високої відмовостійкості та засоби сортування, фільтрації даних у запитах.

2.4 Керування станом

У мобільних застосунках існують два типи станів: глобальний стан та локальний. Вони необхідні для зберігання, у глобальному даних, що необхідні від сесії до сесії (дозволи на встановлення інтернет, bluetooth зв'язку, отримання геолокації, використання камери, дані про авторизації та ін.). Локальні стани - це стан застосунку у поточній сесії (стани сторінок, прогрес запущених анімацій, поточна сторінка застосунку) [5].

Засоби керування станом зазвичай вирішують ще одну проблему - відокремлення бізнес логіки від презентаційного рівня застосунку.

Тому необхідним є обрати засіб керування станом застосунку.

Flutter має вбудовані засоби стейт-менеджменту, вони реалізовані за допомогою віджетів, що самі можуть керувати своїм станом та зберігати його. Але коли стан віджету має синхронізуватися з іншими, таке рішення є громіздким та не гнучким, тому підходить лише для маленьких застосунків.

Flutter має безліч пакетів, за допомогою яких можна виконувати стейт-менеджмент, але найрозповсюдженіші це BLoC, Cubit, Provider та Redux.

BLoC, Cubit та Provider використовують принцип реактивного програмування - розповсюдження змін шляхом прослуховування потоків даних.

Redux - засновано на принципі функціонального програмування.

Порівняння цих пакетів наведено у табл. 2.4

Таблиця 2.4 - Порівняння засобів керування станом застосунку

Критерій	BLoC	Cubit	Provider	Redux
Основні компоненти	Bloc, Event та State	Bloc, Event та State	ChangeNotifier, ChangeNotifier Provider та Consumer	Reducer, Action та Store
Послідовність відповідей у тому ж порядку, що й виклик подій	+	-	-	+
Сховища для різних за призначенням даних	Необмежена кількість	Необмежена кількість	Необмежена кількість	Одне сховище для всіх даних
Можливість відслідковування зміни окремих властивостей	+	+	-	+
Використання у застосунках	Середньої та великої складності	Прості застосунки (Minimal view prototype)	Прості застосунки (Minimal view prototype)	Середньої складності

Проаналізувавши вищезгадані пакети для керування станом застосунку було обрано BLoC, через його потужні засоби реагування на зміни стану, можливість створення різних сховищ, для дотримання принципу єдиної відповідальності. А також доречність такої архітектури для застосунків середньої та великої складності, що надасть можливість дороблювати та розширювати функціонал MigTracker без зміни на іншу, більш потужну.

2.5 Архітектура застосунку

Виходячи з того, що у розділі 2.4 було обрано BLoC, як засіб керування станом застосунку, а також відокремлення бізнес логіки застосунку, то застосунок буде побудований на основі чистої архітектури (Clean architecture).

Чиста архітектура передбачає розділення презентаційного рівня, рівня бізнес логіки та рівня даних. Ця архітектура є необхідною для надання коду більш зрозумілого вигляду, уникнення надлишкової зв'язності компонентів та можливості перевикористання коду. Усе це робить кодову базу застосунку легкою у розумінні, а отже і у підтримці, а також розширенні функціоналу.

BLoC працює у застосунку за наступним принципом. Спочатку створюється клас, що реалізує поведінку абстрактного класу із пакету. У такому класі, його також називають блок (bloc), можна задекларувати сервіси або сховища, до яких він буде звертатися. За ним закріплюється стан. Стан (state) це клас, що містить властивості, які використовує bloc. Наступним кроком створюється подія (event), або декілька, в залежності від потреби. Події закріплені за певним блоком і лише він може відповідати на їх виклик. Події мають доступ до властивостей стану, але не можуть змінювати їх. Для того, щоб змінити стан, необхідно додати у потік (stream) новий стан.

Для кращого розуміння чистої архітектури наведено рисунок 2.2. З нього видно, що коли у презентаційному рівні, тобто у віджеті на екрані, відбувається дія (натискання кнопки, закриття екрану, перехід, тощо), на яку потрібно виконати якусь бізнес-логіку, то викликається подія. Події попередньо визначені у рівні BLoC. Подія виконує необхідну логіку, наприклад, виклик функції сервісу для запиту до БД. Коли подія завершується, тобто код, що вона містить виконано, то вона додає оновлений стан у потік. Оновлений стан не передається у презентаційний рівень буквально. Цей рівень може слухати зміни у потоці та реагувати на них.

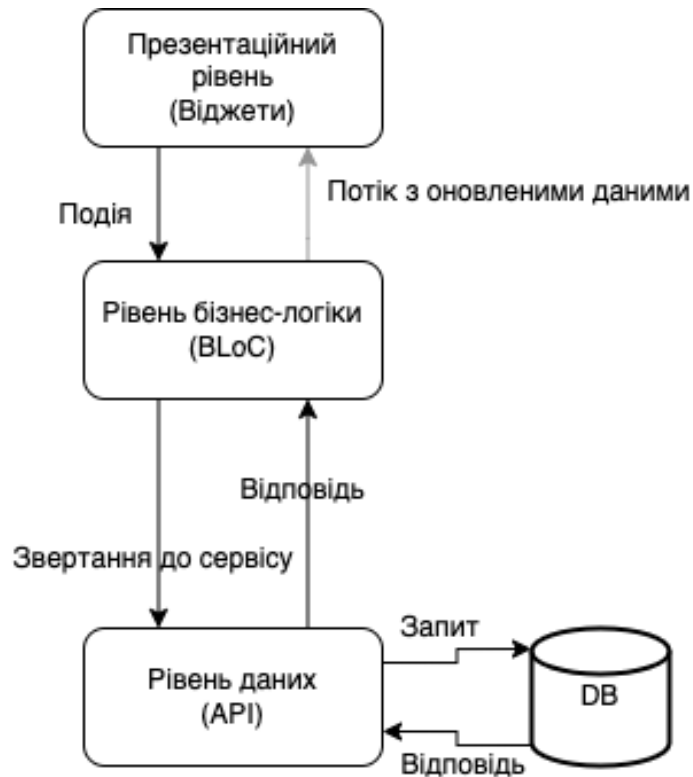


Рисунок 2.2 - Діаграма обраної архітектури

2.6 Структурна схема програми

При створенні застосунку Flutter автоматично створюється структура папок та файлів. До них входять:

- .dart_tool - файли цієї папки входять файли, що використовуються інструментами розробника (DartDevTools) [6];
- .idea - файли, що використовує середовище розробки;
- android - файли та папки, за допомоги яких можна виконувати застосунок для ОС Android;
- ios - файли та папки, за допомоги яких можна виконувати застосунок для ОС IOS;
- lib - папки та файли безпосередньо з вихідним кодом застосунку. В ній автоматично генерується файл з вхідною точкою проекту main.dart;

– test - папка у яку додаються файли для тестування застосунку. Автоматично генерується файл `widget_test.dart`, де міститься вхідна точка для виконання тестів;

– web - файли та папки, за допомоги яких можна виконувати застосунок як веб застосунок та містить автоматично згенерований файл з вхідною точкою `index.html`;

– windows - файли та папки, за допомоги яких можна виконувати застосунок для ОС Windows та містить автоматично згенерований файл `CMakeLists.txt`, що зберігає конфігурацію збірки;

– .gitignore - файл з переліком файлів, що мають ігноруватися системою контролю версій;

– .metadata - файл, що зберігає артефакти для оновлення версії Flutter;

– .packages - файл, містить інформацію про встановлені пакети;

– .analysis options.yaml - файл, що містить правила для перевірки коду на відповідність їм. Цей файл може розширюватись для дотримання розробниками однієї стилістики коду;

– flutter_application_1.iml - файл-модуль, необхідний для Android;

– pubspec.lock - містить інформацію про версії встановлених пакетів;

– pubspec.yaml - файл, що містить інформацію про назву, опис проекту, версію проекту та версію мови Dart. В нього також додають залежності проекту, пакети, шрифти, шляхи до медіафайлів;

– README.md - супроводжуючий проект файл, що заповнюється розробником, як інструкція до запуску проекту.

Структура проекту наведено на рис. 2.3.

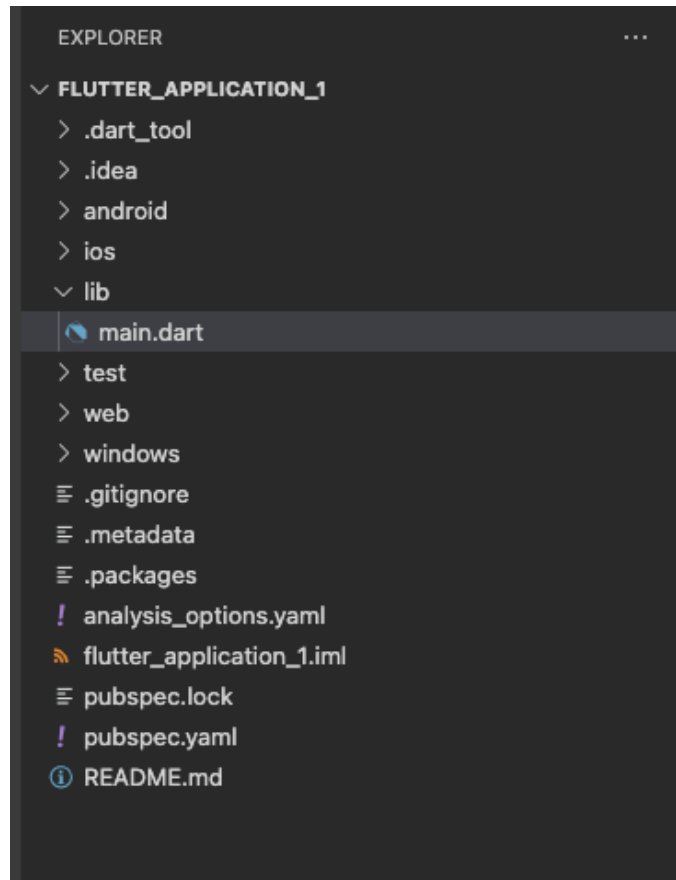


Рисунок 2.3 - Структура Flutter застосунку

До проекту зазвичай додають папки `assets` та `fonts`, що будуть знаходитись на рівні з `lib`. У них зберігаються медіафайли проекту та шрифти.

2.7 Висновки за 2 розділом

У другому розділі виконано опис програми з визначенням мови програмування та детальним розглядом фреймворку Flutter. Також було проведено аналіз середовища програмування, та за низкою переваг (наявність вбудованого емулятору та необхідних для Android розробки інструментів) обрано середовище розробки Android Studio.

У розділі 2.3 було розглянуто та порівняно віддалені сховища для зберігання даних від компанії Google, та обрано Firestore Database. Також було

розглянуто найпопулярніші засоби керування станами у Flutter, та за результатами порівняльної таблиці обрано BLoC.

Було також визначено архітектуру та проаналізовано структуру застосунку.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування розробленої програми

Основні алгоритми, які можуть виконуватися при роботі із розробленою програмою зображені на рис. 3.1.

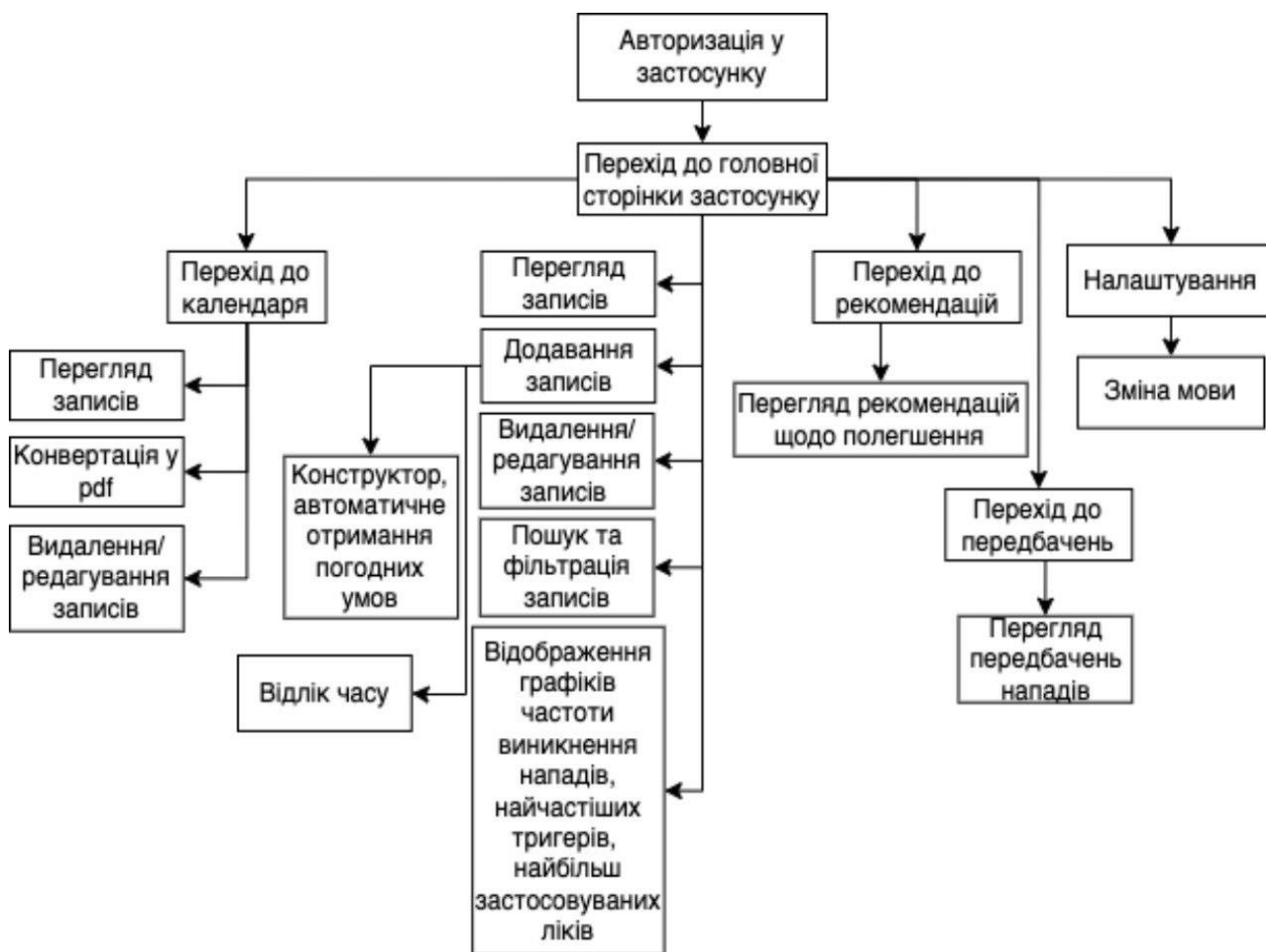


Рисунок 3.1 – Функціональні можливості розробленої програми

Як видно зі схеми під час розробки застосунку «MigTracker», були розроблені наступні алгоритми:

- авторизація;
- реєстрація;

- алгоритм реєстрації/ авторизації із допомоги сторонніх сервісів;
- пошук записів;
- сортування записів;
- створення та редагування записів;
- підтвердження завершення нападу;
- видалення записів;
- відображення записів у календарю;
- алгоритм передбачення нападу;
- збереження записів у pdf формат.

3.1.1 Алгоритм авторизації

Далі буде наведено алгоритм авторизації користувача.

Крок 1. Дані введені користувачем передаються у подію `LoginEvent()` авторизаційного блоку `AuthBloc`.

Крок 2. Виконується запит до `Firebase` з даними, що були введені користувачем.

Крок 3. Якщо запит повертає токен, то запис існує то крок 4, якщо ні крок 5.

Крок 4. Користувача переадресовує на головну сторінку.

Крок 5. Користувач отримує попередження про некоректні дані.

Крок 6. Кінець алгоритму.

3.1.2 Алгоритм реєстрації

Далі буде наведено алгоритм реєстрації користувача.

Крок 1. Дані введені користувачем передаються у подію `SignUpEvent()`,

авторизаційного блоку AuthBloc.

Крок 2. Виконується запит до Firebase з даними, що були введені користувачем.

Крок 3. Якщо запит повертає токен, то запис успішно створений то крок 4, якщо ні крок 5.

Крок 4. Користувача переадресовує на головну сторінку, перехід на крок 6.

Крок 5. Користувач отримує попередження про вже зареєстрований під цими даними обліковий запис.

Крок 6. Кінець алгоритму.

3.1.3 Алгоритм авторизації та реєстрації за допомоги сторонніх сервісів

У якості сторонніх сервісів використовується Facebook та Google. Алгоритм авторизації та реєстрації у данному випадку співпадають, та наведено нижче.

Крок 1. Користувач знаходиться на авторизаційній чи реєстраційній сторінці. Натискає на одну з запропонованих кнопок. Виконується подія `SocialSignInEvent()` блоку AuthBloc.

Крок 2. Відкривається вікно для авторизації на сервісі, користувач вводить дані та підтверджує авторизацію, виконується запит до сервісу, якщо введені дані валідні, то запит повертає токен, то крок 3, якщо ні - крок 4.

Крок 3. Виконується запит до Firebase з токеном, що було отримано у другому кроці, якщо операція пройшла успішно, то повертається авторизаційний токен та перехід на крок 4, якщо ні то крок 5.

Крок 4. Користувача переадресовує на головну сторінку, перехід на крок 6.

Крок 5. Користувач отримує попередження про некоректні дані.

Крок 6. Кінець алгоритму.

3.1.4 Алгоритм пошуку записів

Далі наведено алгоритм пошуку записів.

Крок 1. Запит введений користувачем для пошуку, передається у подію SearchAttackEvent() блоку AttackBlock.

Крок 2. Виконується функція searchAttack() для записів, що зберігаються у AttackState.

Крок 3. Для кожного оголошення перевіряється наявність фрази для пошуку цілком у описі запису. Перехід на крок 5.

Крок 4. Якщо співпадіння знайдене – перехід на крок 6, якщо ні – крок 8.

Крок 5. Оголошення додається до списку-результату. Перехід на крок 8.

Крок 6. Якщо досягнуто кінця, перехід на крок 7, якщо ні повернення на крок 4.

Крок 7. Для кожного оголошення перевіряється наявність фрази у всіх параметрах.

Крок 8. Якщо співпадіння знайдене – перехід на крок 9, якщо ні – крок 11.

Крок 9. Перевіряється наявність цього оголошення у списку-результату, якщо знайдене – крок 13, якщо ні крок 12.

Крок 10. Оголошення додається до списку-результату. Перехід на крок 13.

Крок 11. Якщо досягнуто кінця, перехід на крок 12, якщо ні повернення на крок 7.

Крок 12. На основі вказаного режиму сортування, відбувається сортування списку-результату оголошень до часткового вигляду.

Крок 13. Кінець алгоритму.

3.1.5 Сорткування записів

Сорткування записів може відбуватися за кількома критеріями: дата, кількість симптомів, величина болю. За замовчанням сорткування відбувається лише за датою. Нижче наведено алгоритм сорткування записів.

Крок 1. Користувач обирає один з параметрів сорткування. Викликається подія `SortAttackListEvent()` блоку `AttackBlock` з обраним параметром у якості аргументу.

Крок 2. Одночасно сорткування може відбуватися лише за одним параметром, тому перевіряється який параметр було обрано попередній раз. Якщо той самий, то перехід на крок 4, інакше попередні стають неактивними.

Крок 3. Викликається функція сорткування за потрібним параметром за спадаючою, повертається результуючий список, перехід до крок 5.

Крок 4. Викликається функція сорткування за потрібним параметром за зростаючою, повертається результуючий список.

Крок 5. Кінець алгоритму.

3.1.6 Алгоритм створення запису

Далі представлено алгоритм створення запису про напад.

Крок 1. Користувач викликає конструктор для створення записів.

Крок 2. Користувач заповнює дані про погодні умови самотужки, перехід до крок 4. Користувач використовує автозаповнення - крок 3.

Крок 3. Викликається подія `FetchWeatherEvent` блоку `AttackBlock`, яка виконує функцію `fetchWeather()`, що повертає дані про погодні умови. `AttackBlock` оновлює свій стан.

Крок 4. Якщо користувач створює запис про напад, що уже завершився, то він натискає підтвердження і виконується крок 8, якщо користувач

використовує відкладене завершення нападу то крок 5.

Крок 5. Дані про напад передаються у подію `SetCurrentAttackEvent` блоку `AttackBlock`. Якщо поле `currentAttack` не дорівнює `null`, тобто поточний напад вже існує, перехід до крок 6, інакше - крок 8.

Крок 6. Користувач отримує повідомлення, де йому пропонується підтвердити закінчення попередньої атаки, або скасувати процедуру відкладеного завершення. Якщо він обирає підтвердити завершення - перехід до крок 7, інакше крок 4.

Крок 7. Для попереднього нападу, що був підтверджений як завершений, виконується алгоритм підтвердження завершення нападу (розд. 3.1.7).

Крок 8. Подія `SetCurrentAttackEvent` оновлює стан блоку із оновленим полем `currentAttack` у який поміщено поточний напад.

Крок 9. У подію `PostAttackEvent` блоку `AttackBlock` передаються дані події, що завершилась із полем `isFinished` (в залежності від стану нападу), викликається метод `postNewAttack()`, що додає новий запис до віддаленого сховища, якщо успішно - перехід на крок 11, якщо ні, користувач отримує повідомлення, перехід на крок 10.

Крок 10. Запис додається до загального списку нападів, перехід на крок 11.

Крок 11. Кінець алгоритму.

Для кращого розуміння роботи алгоритму на рисунку 3.2 зображено блок-схему створення запису про напад мігрені.

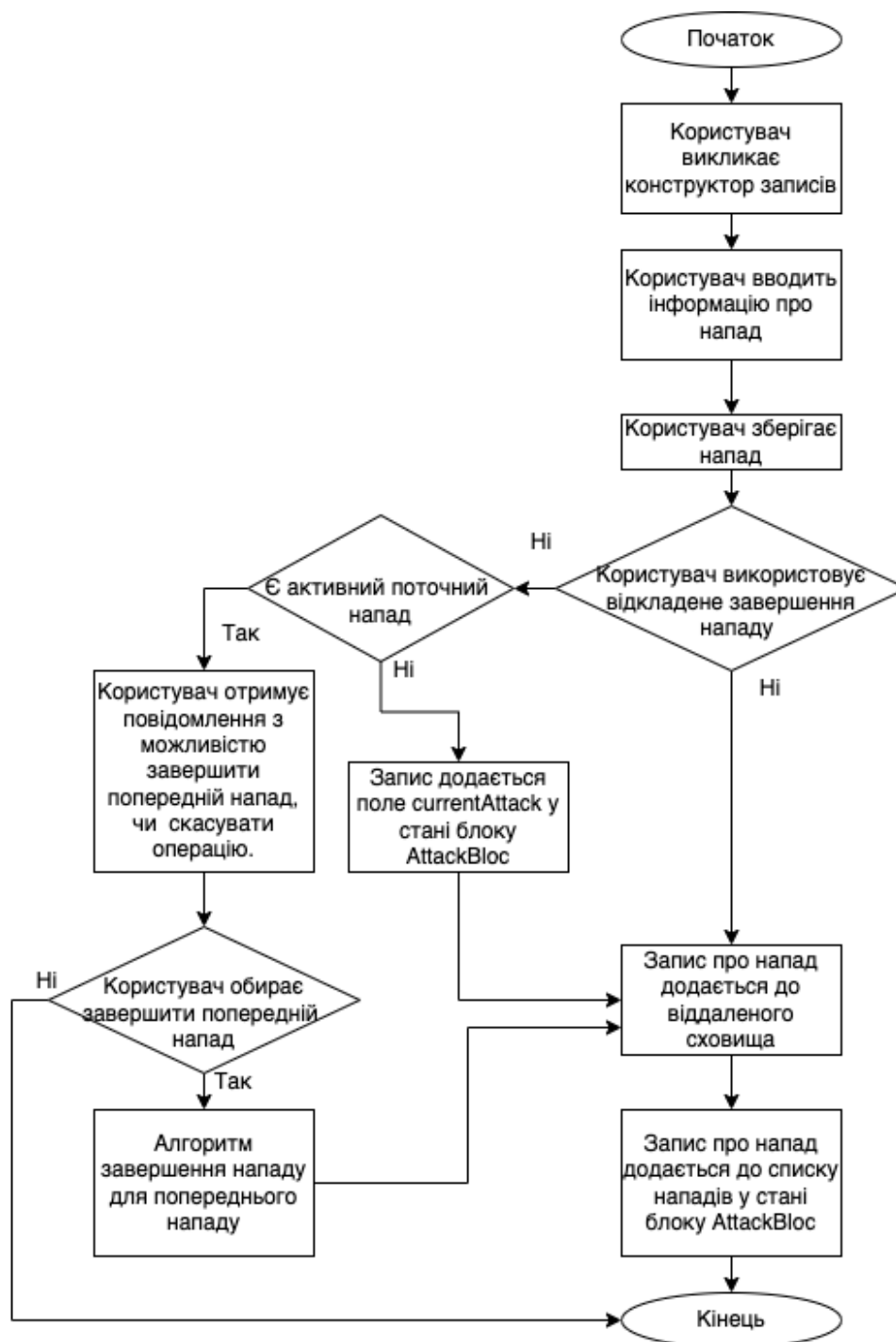


Рисунок 3.2 - Блок-схема алгоритма створення запису

3.1.7 Алгоритм підтвердження завершення нападу

Для того щоб завершити напад необхідно його підтверити. Нижче

наведено алгоритм підтвердження завершення нападу.

Крок 1. Користувач натискає кнопку завершення на поточному нападі, та викликає подію `FinishAttackEvent` блоку `AttackBlock`.

Крок 2. Викликається метод `putFinishedAttack()`, що змінює у віддаленому сховищі змінну `isFinished` на `true`.

Крок 3. У загальному списку нападів, що зберігається у стані блоку `AttackBlock` поточний запис замінюється на той же, але з полем `isFinished`, що дорівнює `true`.

Крок 4. Поле `currentAttack`, що знаходиться у стані блоку `AttackBlock` приймає `null`.

Крок 5. Кінець алгоритму.

3.1.8 Алгоритм редагування запису

Нижче наведено алгоритм редагування запису.

Крок 1. Користувач вибирає редагування запису, викликається подія `SetCurrentAttackEvent`, у яку передається обраний запис.

Крок 2. Перед користувачем розгортається конструктор до якого він вводить виправлені дані та підтверджує зміни, перехід до крок 3, або скасовує, перехід на крок 5.

Крок 3. До події `EditAttackEvent()` блоку `AttackBlock` надходить оновлений запис та індекс за яким він знаходиться у списку нападів, виконується запит до віддаленого сховища, де старий запис оновлюється новим. Якщо запит успішний, перехід до крок 4, якщо ні, користувач отримує повідомлення, перехід на крок 5.

Крок 4. Новий запис замінює старий у списку нападів. Змінений список оновлює стан блоку.

Крок 5. Кінець алгоритму.

3.1.9 Алгоритм видалення запису

Нижче наведено алгоритм видалення запису.

Крок 1. Користувач вибирає запис для видалення, викликається подія `DeleteAttackEvent`, у яку передається обраний запис.

Крок 2. Виконується запит до віддаленого сховища, де вибраний запис видаляється. Якщо запит успішний, перехід до крок 3, якщо ні, користувач отримує повідомлення, перехід на крок 4.

Крок 3. Запис видаляється зі списку нападів. Змінений список оновлює стан блоку.

Крок 4. Кінець алгоритму.

3.1.10 Алгоритми відображення записів у календарю

Алгоритм відображення записів у календарю.

Крок 1. При завантаженні застосунку, викликається запит `FetchAttacksEvent` блоку `AttackBloc` виконується запит до віддаленого сховища, який повертає записи про напади, що додаються до списку у стані блоку.

Крок 2. При підключенні до плагіну календаря, у відповідність до кожної дати ставляться записи, якщо один чи декілька записів існує на поточну дату, то крок 3, якщо ні, то крок 4.

Крок 3. Список з записів відсортовується за часом виникнення за спаданням.

Крок 4. Список з записами (або пустий) повертається як результат функції. Плагін календарю визиває таку функцію, передаючи у якості аргумента дату.

Крок 5. Кінець алгоритму.

3.1.11 Алгоритм передбачення нападу

Виходячи з того, що напади мігрені можуть бути спровоковані певними погодними умовами або їх стрімкою зміною, відповідно до статті Дебори Фридман [7] та наукових праць Альберта С. Янга, Чон-Линг Фу [8] та багатьох інших науковців, то на основі цих знань можна передбачити напад.

У науковій праці Хирохиса Окума [8] було доведено що при зменшенні атмосферного тиску на 6–10 hPa (4.5 - 7.5 мм р.ст.) від середнього 1013 hPa (760 мм р.ст.) виникає великий ризик нападу мігрені. А також зміна температури більш ніж на 5 °C за менш ніж 24 години теж може викликати головний біль у пацієнтів, що страждають на мігрень, згідно до праці Кеннета Дж. Мукамаля [10].

У статті докторів медичних наук Вэньюань Ли Да та Сюзан М. Бертиш [11] виведено залежність, що у теплий період часу більш ніж 13.3 °C (56.9 °F) при вологості більшої ніж 67.3% ризик виникнення мігрені.

А також виникнення мігрені пов'язують із низькими температурами (менше за -10 °C) та можливим переохолодженням людини, що страждає на мігрень [12].

У застосунку є алгоритм передбачення нападу на основі погодних умов, що буде наведено нижче.

Крок 1. Для отримання передбачення користувач при першому запуску застосунку у вкладці «Передбачення» дає дозвіл на отримання його геолокації.

Крок 2. Кожний день застосунок буде викликати подію `FetchDailyWeatherEvent()` блоку `AttackBloc` та підтягувати погодні умови.

Крок 3. Викликається функція `fetchWeather()`, отримані температура, вологість та атмосферний тиск формують об'єкт типу `Weather`, який додається у клас `PredictionWeather`, що складається з двох об'єктів типу `ShortWeatherModel`, один з них поточні погодні умови, другий попередні, а також поточної дати та відповіді користувача чи справдилося передбачення.

Крок 4. До віддаленого сховища надсилається об'єкт типу PredictionWeather, який замінює собою попередній.

Крок 5. На основі вищезазначеної інформації, перевіряється відхилення тиску від норми, у разі наявності - визначається ризик головного болю.

Крок 6. Вираховується різниця між температурами поточних та попередніх погодних умов, якщо різниця склала більш ніж 5 °C - визначається ризик головного болю.

Крок 7. Якщо поточна температура менше ніж -10 °C - визначається ризик головного болю.

Крок 8. Якщо температура більша за 13.3 °C, а вологість більша за 67.3% - визначається ризик головного болю.

Крок 9. Якщо ризик головного болю відзначається 2 чи більше разів, то він є підвищеним, результат повертається у стан AttackBloc у вигляді однієї з констант increasedRisk об'єкту enum RiskPrediction. Якщо ризик визначився 1 раз - risk, якщо 0, то noRisk.

Крок 10. Якщо ризик існує, у вкладці з передбаченнями з'явиться повідомлення, у якому буде вказано чи існує ризик головного болю сьогодні, та з якої причини.

Крок 11. Кінець алгоритма.

Для кращого розуміння роботи алгоритму на рис. 3.3 зображено блок схему передбачення нападу.

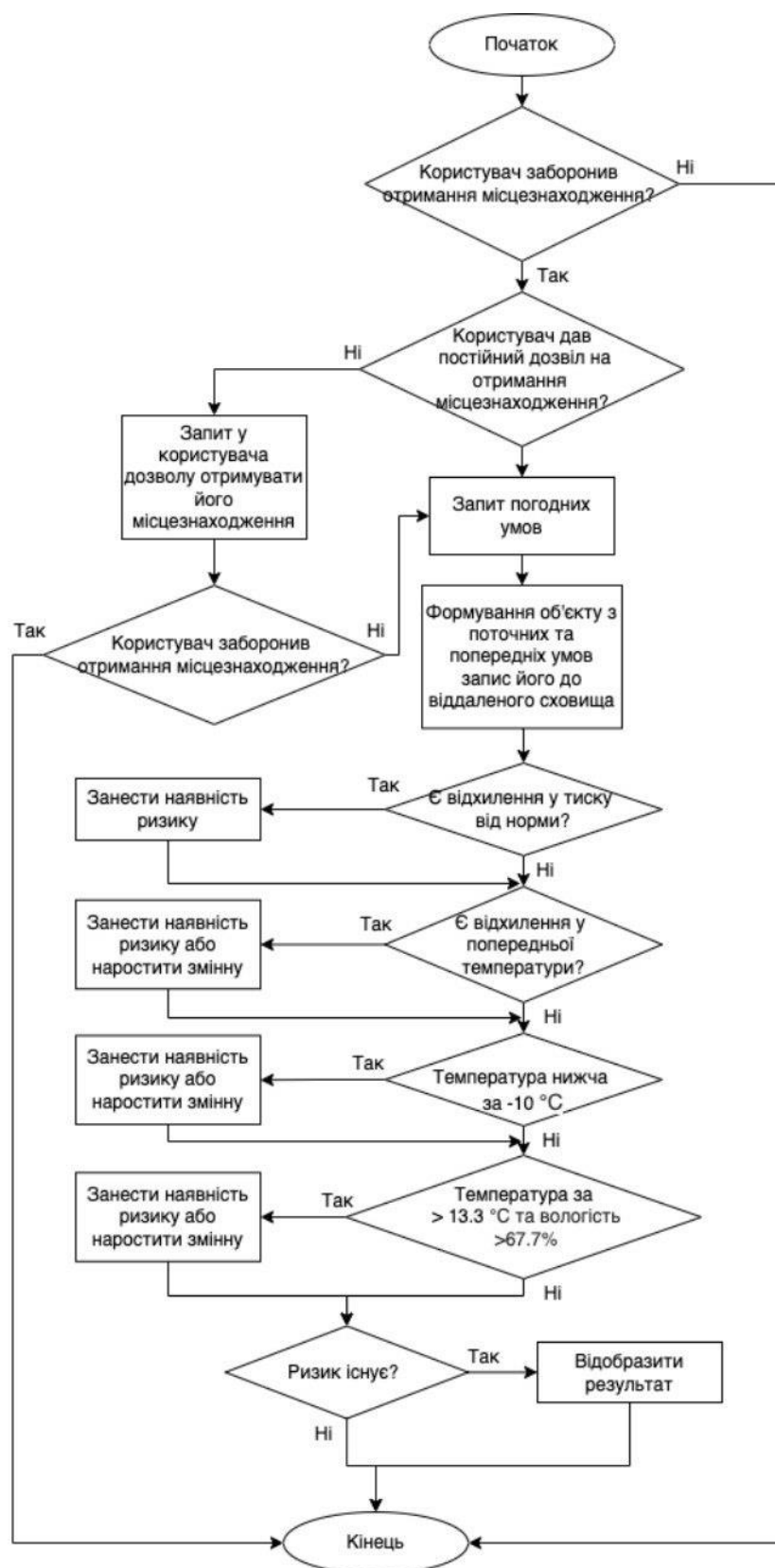


Рисунок 3.3 - Блок-схема алгоритма передбачення нападу

3.1.12 Алгоритм конвертації нападів у pdf формат

Нижче буде представлено алгоритм для збереження записів у PDF форматі.

Крок 1. Користувач на вкладці «Календар» натискає кнопку «Зберегти у pdf», у спливаючому меню обирає один із запропонованих періодів (збереження записів за поточний день, поточний місяць або за весь час).

Крок 2. До події SaveToPdfEvent блоку AttackBloc передається обраний період, із списку записів вибираються ті, що відповідають періоду.

Крок 3. За допомоги функції convertToPdf(), що приймає у якості параметру список записів, та заздалегідь визначає структуру файлу, створюється файл .pdf.

Крок 4. Файл зберігається у пам'яті телефону.

Крок 5. Кінець алгоритму.

3.2 Архітектура застосунку на основі предметної області

На основі обраної у розділі 2.5 архітектури було побудовано компоненти до яких входять класи, віджети, та сервіси. Їх побудовано за принципом великої зв'язності у середині компонентів та низької між компонентами.

У застосунку розроблено компонент Authorization, до якого входять класи: AuthApi, що реалізує інтерфейс між FirebaseAuth та застосунком AuthRepositoryUnauthorized – клас, що містить функціонал, для авторизації - зв'язок з AuthApi, а також авторизація за допомоги сторонніх сервісів.

Клас AuthService є абстракцією над класами AuthServiceImpl та AuthServiceMock. AuthServiceImpl є реалізацією безпосередньо для авторизації та AuthServiceMock є реалізацією класу для тестування. За допомоги AuthServiceImpl застосунок звертається до AuthRepositoryUnauthorized.

Клас `SignUpDto` - містить модель об'єкту для передачі реєстраційних даних користувача.

Клас `LoginDto` – містить модель об'єкту для передачі авторизаційних даних користувача.

Клас `SocialTokenDto` – містить модель об'єкту для передачі токена від сторонніх сервісів.

Клас `AuthBloc` – реалізує авторизаційний блок (компонент бізнес логіки), що використовується у якості менеджера станів для сторінки авторизації.

Клас `AuthState` створено для зберігання стану авторизаційного блоку. Містить властивості, стан необхідно зберігати під час роботи з авторизаційним блоком.

Клас `LoginEvent` – подія авторизаційного блоку, що виконує необхідні операції для логіну користувача та повертає оновлений стан авторизаційного блоку.

Клас `SignUpEvent` – подія авторизаційного блоку, що виконує необхідні операції для реєстрації користувача у застосунку.

Клас `SocialSignInEvent` – подія, що виконує операції для реєстрації та авторизації користувача у застосунку із допомоги сторонніх сервісів.

Клас `AuthorizationScreen` – реалізує структурну одиницю – віджет, що використовується для створення користувацького інтерфейсу екрану авторизації.

До нього входять інші класи-віджети (`LoginScreen`, `SingUpScreen`, `SignUpFieldsList`, `LoginFields`), які реалізують менші компоненти інтерфейсу.

Для того, щоб винести з віджетів елементи логіки, виклики функцій та інший код, що ускладнює розуміння коду використано міксини. Вони також є класами, але не мають конструктору та можуть розширювати інші класи власними полями та методами. `LoginScreenMixin` – розширює `LoginScreen`, `SingUpScreenMixin` - `SingUpScreen`.

Наступний розроблений компонент програми - це компонент `Attack`. До нього входять наведені нижче класи.

Клас WeatherApi слугує для виконання REST API викликів до сервісу, що надає погодні умови, відповідно до заданих координатів.

Клас AttackRepository містить методи для отримання та відправки записів про напади мігрені у віддалене сховище, отримання довідкових даних, а також методи, що звертаються до класу WeatherApi та обробляють отримані дані.

Клас AttackService є абстракцією над класами AttackServiceImpl та AttackServiceMock, у ньому описані методи, які використовуються класами для додавання, отримання записів, отримання погоди, довідкових даних. AttackServiceImpl звертається до методів класу AttackRepository, а AttackServiceMock необхідний для повернення з методів статичних, попередньо визначених даних для тестування компоненту.

Клас AttackBloc – реалізує блок нападів, що використовується у якості менеджера станів для сторінок, що залежать від даних стосовно нападів.

Клас AttackState – зберігає стан AttackBloc, тобто дані, що стосуються нападів мігрені і використовуються у застосунку.

Класи, що будуть зазначені нижче є подіями блоку AttackBloc. Події виконують операції над даними з AttackState та можуть їх оновлювати.

FetchAttacksEvent – подія завантаження записів про напади з віддаленого сховища.

FetchTreatmentsEvent та FetchSymptomsEvent – завантаження довідкових даних про ліки та симптоми.

PostAttackEvent – відправка запису про атаку у віддалене сховище.

SetUnsetSymptomsEvent – подія для застосування та відхилення симптомів та ліків при створенні нового запису про напад.

SearchAttackEvent та SortAttackListEvent події, виконують пошук та сортування записів.

FetchDailyWeatherEvent – отримує поточні погодні умови.

SetCurrentAttackEvent – встановлює об'єкт нападу, що було передано у

подію у якості поточного (ще не завершеного) нападу.

`FinishAttackEvent` – подія, що викликається при підтвердженні завершення нападу.

`DeleteAttackEvent` – подія для видалення запису. Надсилає запит на видалення до віддаленого сховища.

`SaveToPdfEvent` – викликає сервіс для збереження записів про напади у форматі .pdf.

`EditAttackEvent` – подія, що виконується при редагуванні існуючого запису про напад.

Клас `AttackDTO` – містить модель об'єкту для передачі та отримання запису про напад у віддаленого сховищу.

Клас `AttackModel` – містить модель запису нападу та використовується у застосунку.

Клас `ForecastDTO` – містить модель об'єкту для отримання погодних умов.

Клас `EditableSymptomsModel` – модель для використання симптому або лікарського засобу у віджеті.

Абстрактний клас `PickingGroupAbstract` – модель, для отримання групи симптомів або лікарських засобів. Він необхідний для уніфікації аргументів для віджетів, що можуть приймати як модель симптому так і модель ліків.

Клас `SymptomsGroup` наслідує `PickingGroupAbstract` та є моделлю для отримання та зберігання груп симптомів.

Клас `MedicinesGroup` наслідує `PickingGroupAbstract` та є моделлю для отримання та зберігання груп лікарських засобів.

Класи `Symptom` та `Medicines` – це модель для отримання та зберігання у застосунку симптому та лікарського засобу відповідно.

Клас `TemperatureDTO` – використовується як модель температури, входить до складу `ForecastDTO` як одне з полей.

Клас WeatherDTO – це допоміжний клас, що дозволяє виокремити з класу ForecastDTO основні погодні умови, та помістити їх у клас WeatherModel.

Класи, що відносяться до презентаційного шару компоненту нападу це віджет MainScreen у якому відображено список нападів, реалізовано елементи інтерфейсу для пошуку та сортування списку нападів, відображення графіків. До віджету входять інші, дрібні віджети: AttackItem – запис нападу, AttackList – список записів нападів, ChartCurrency – графік частоти виникнення нападів, ChartPanel – панель у якій відображуються графіки, PieChartSymptoms – круговий графік частоти симптомів. Також розроблену міксину ChartPanelMixin, що надає поля та методи віджету ChartPanel.

Віджет AttackCreation є реалізацією конструктору для створення запису про напад мігрені. До нього входять такі віджети як: FirstCreationPage, що містить віджет Pickers – елементи для встановлення дати, часу та тривалості нападу, CustomPicker – віджет для вибору лікарських засобів. SecondCreationPage містить PainLocalization – віджет для вибору локалізацій болю, а також ще CustomPicker але для вибору медичних препаратів. ThirdCreationPage містить StressSlider – віджет для встановлення рівню стресу, SleepingTimePicker використовується для встановлення кількості часів сну, WeatherConditions – погодні умови та TriggerButton – кнопки для вибору факторів, що могли спричинити головний біль.

Віджет Calendar також входить до компоненту атаки, та є відображенням списку записів про напади у вигляді календарю.

Наступний розроблений компонент – це компонент Advices, щодо поліпшення симптомів та попередження нападів. До нього входять наведені нижче класи.

Клас AdviceRepository – сервіс для отримання та відправки списку порад.

Клас AdviceService - абстрактний клас, що визначає поля та методи для класів AdviceServiceImpl - що використовується безпосередньо для звернення до AdviceRepository, а AdviceServiceMock - використовується для тестування.

Клас `AdviceBloc` - реалізує блок, який використовується для контролю стану – `AdviceState` та для відділення логіки завантаження порад від презентаційного шару. У `AdviceState` зберігається список порад, що використовує застосунок.

Клас `FetchAvicesEvent` – це подія блоку `AdviceBloc`, що виконує операції для підвантаження списку порад, використовуючи `AdviceService` та оновлює стан `AdviceState`.

Клас `AdviceDto` – це модель об'єкту для отримання сутності поради, та роботи з нею у застосунку.

До компоненту порад відносяться наступні віджети: `AdvicesScreen` - що є сторінкою з порадами та `AdviceItem` – що є записом поради.

У застосунку присутній ще компонент `Prediction`. До нього входять класи, що наведено нижче.

Клас `PredictionRepository` використовується для отримання та передачів передбачень у віддалене сховище.

Клас `PredictionService` є абстрацією над `PredictionServiceImpl` та `PredictionServiceMock`. Де перший з них використовується у застосунку для звернення у `PredictionRepository`, а другий для надання статичних даних при тестуванні.

Клас `PredictionBloc` контролює стан `PredictionState`, що містить поля які використовуються для створення та зберігання передбачення у застосунку.

Клас `FetchPreviousPredictionEvent` – подія, що здійснює операції для завантаження попереднього передбачення та оновлення стану `PredictionBloc`.

Клас `FetchDailyWeatherEvent` – подія, що виконує підвантаження погодних умов. Потрібно зазначити, що методи, які подія викликає з сервісу `PredictionServiceImpl` звертаються до класу `WeatherApi`, що відноситься до компоненту нападу.

`DeletePredictionEvent` та `CreatePredictionEvent` – класи, що є подіями

видалення та створення і відправки передбачення до віддаленого сховища. Це дозволяє валідувати точність передбачення, що у подальшому може використовуватись для аналізу та покращення алгоритму передбачення нападу мігрені для окремого користувача.

Клас `ConfirmPredictionEvent` – подія, яка використовується для підтвердження справджених передбачення, а клас `DeclinePredictionEvent` - для несправджених.

Клас `PredictionDTO` необхідний для передачі та отримання об'єкту з моделлю передбачення з віддаленого сховища.

`PredictionWeather` - клас, що зберігає погодні умови за поточний та попередній дні, на основі яких будується передбачення, об'єкт `enum`, що визначає наявність ризику та відповідь користувача на те, чи справдилось передбачення.

`ShortWeatherPrediction` – клас, що зберігає значення температури, вологості та тиску.

До компоненту передбачень також відноситься об'єкт `enum RiskPrediction`, що містить константи `increasedRisk`, `risk`, `noRisk` та використовується для позначення ступеню ризику виникнення нападу мігрені.

Презентаційний рівень компоненту `Prediction` представлений віджетами `PredictionScreen` - що є сторінкою з передбаченнями та `PredictionItem` - що є записом передбачення.

Окрім вищезазначених компонентів у застосунку також є допоміжні, що будуть наведені далі.

`Layouts` - для уніфікації макетів екранів. У ньому визначено фон застосунку, та поведінку, що може бути притаманна декільком екранам.

`Resources` - ресурси застосунку, константи, які зберігають: кольори, шрифти, словники, шляхи до веб-сторінок та мультимедіа файлів, що зазвичай використовуються презентаційним шаром.

`Utils` - це спеціалізовані класи, що виконують специфічну логіку, у

данному випадку валідацію за регулярними виразами та конвертацію одиниць виміру температури та тиску, й можуть використовуватись повсемісно у застосунку. Вони, зазвичай, не є специфічними для кожного застосунку, тому їх з легкістю можна перевикористовувати.

Services - виконують функціонал, який відноситься до застосунку загалом. У даному випадку – один з сервісів перехоплює та обробляє помилки, що надходять при веб-запитах.

Extensions - це файли, що містять розширення функціоналу до окремих типів даних. У застосунку використовується розширення до типу DateTime, що додає до нього можливість порівняння двох об'єктів лише за датою.

Для того, щоб наглядно зобразити взаємодію компонентів та шарів на основі обраної архітектури, наведено діаграму компонентів розробленого застосунку (рис.3.4).

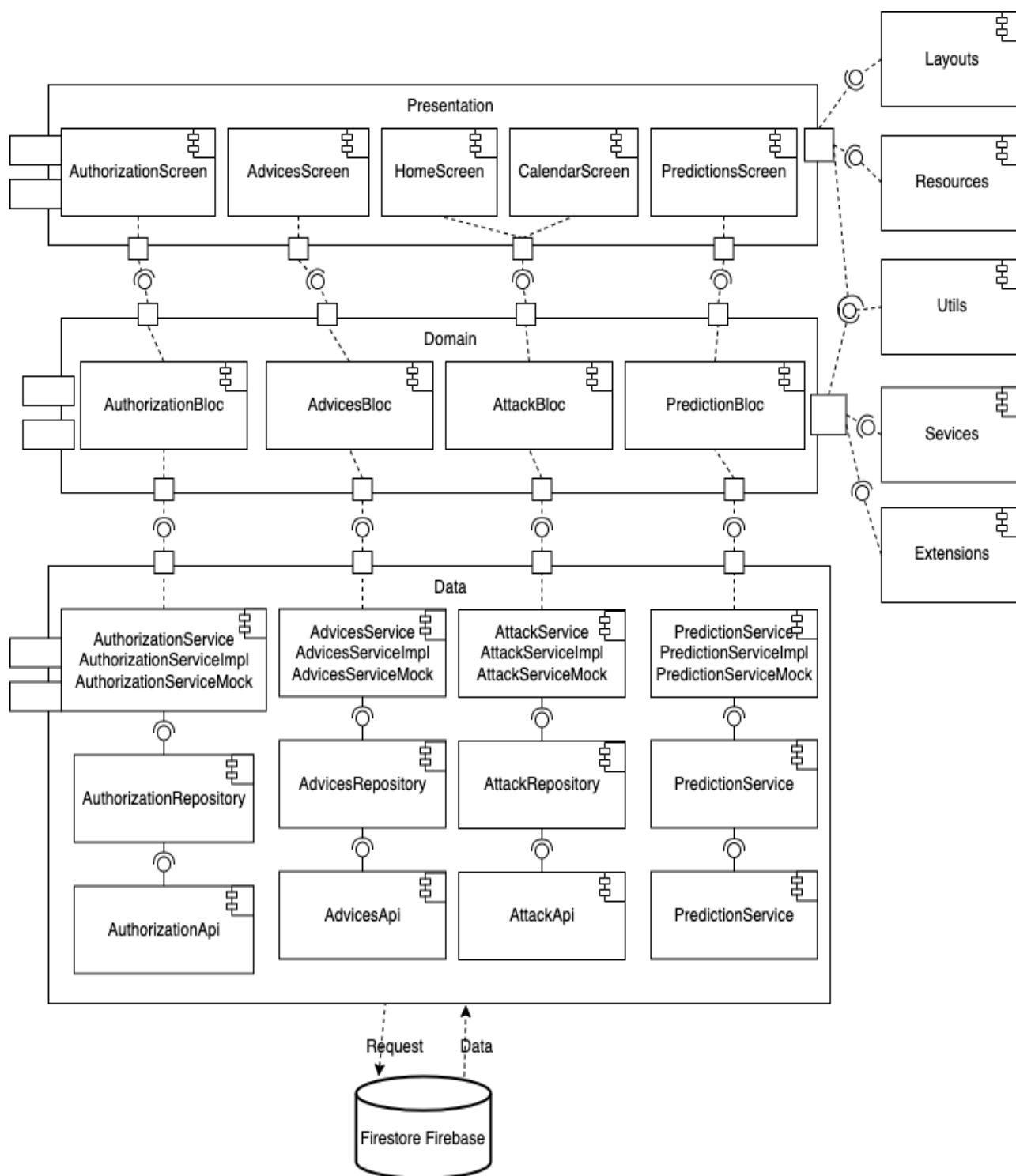


Рисунок 3.4 – Діаграма компонентів розроблюваної системи

3.3 Зберігання даних

У розділі 2.3 для зберігання даних було обрано віддалене сховище Cloud Firestore. Воно відноситься до NoSQL систем керування базами даних, тобто доступ до даних може здійснюватись не тільки засобами мови SQL. Замість звичайних таблиць у реляційних базах даних, ця технологія зберігає дані у документах, які містять поля, яким зіставлені значення. Такі документи зберігаються у колекціях, що безпосередньо є контейнерами для них [13].

Для того, щоб почати роботу зі сховищем, було створено Firebase проект (рис.3.5) та додано до нього застосунки для цільових платформ. У даному випадку це Android та IOS.

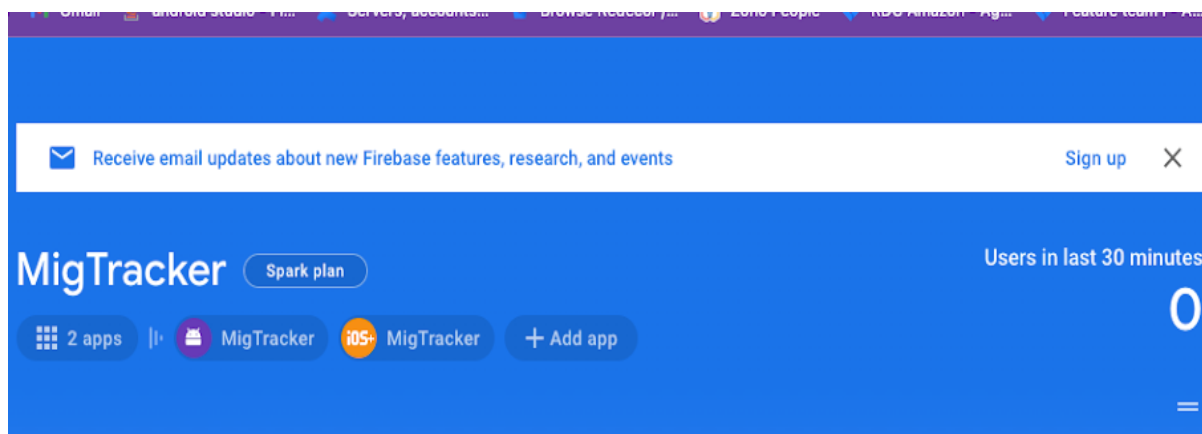


Рисунок 3.5 - Головна сторінка проекту MigTracker

Після цього було налаштовано способи авторизації у застосунку (рис. 3.6). Було обрано авторизацію за допомоги поштової адреси та паролю, а також авторизацію сервісами Google та Facebook, що дозволяють користувачу прив'язати свій обліковий запис, до вже існуючого на одній з платформ.

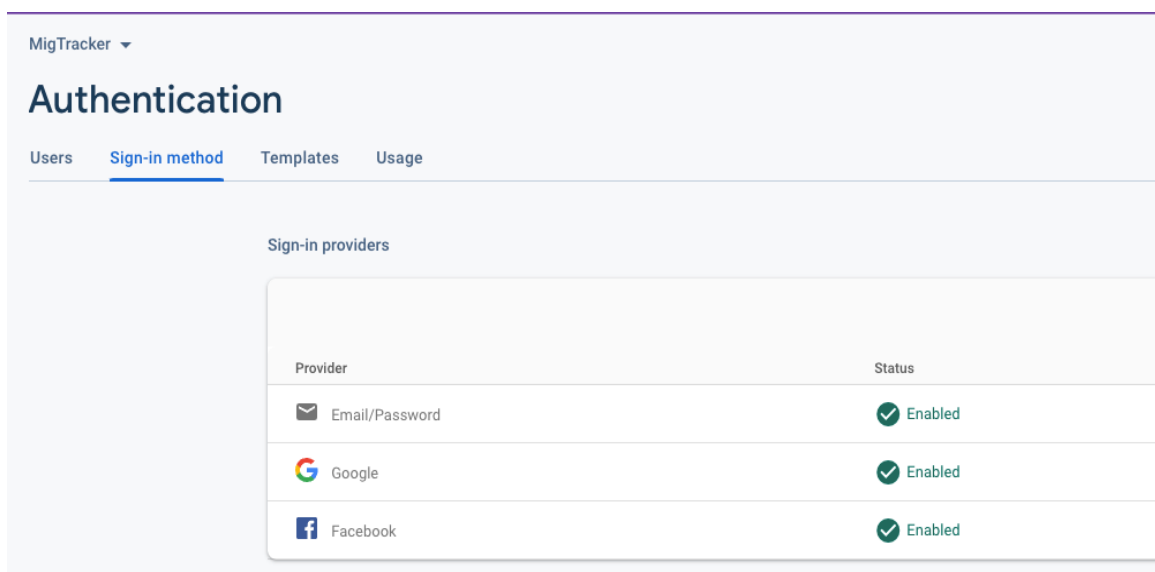


Рисунок 3.6 - Способи авторизації у застосунку

Маніпулювання даними між застосунком та сховищем, тобто відправка та отримання відбуваються у форматі json.

Для взаємодії застосунку та Cloud Firestore використовуються бібліотеки «Cloud Firestore» та «Auth Firestore». Перша з них надає інтерфейс для обміну даними з документами у сховищі, друга дозволяє авторизовувати користувача, та надає йому доступ до сховища.

На рис. 3.7 зображено відношення сутностей у сховищі, проте необхідно зазначити, що безпосередньо у сховищі, цих зв'язків не побудовано вони реалізовані на рівні застосунку. Це зроблено для того, щоб не прив'язувати моделі до id документів і у разі міграції до іншою бази даних не перероблювати їх. Бо зробити посилання на певну сутність іншого документу можна лише знаючи його id, який у більшості випадків генерується автоматично. У якості прикладу наведено рис. 3.8.

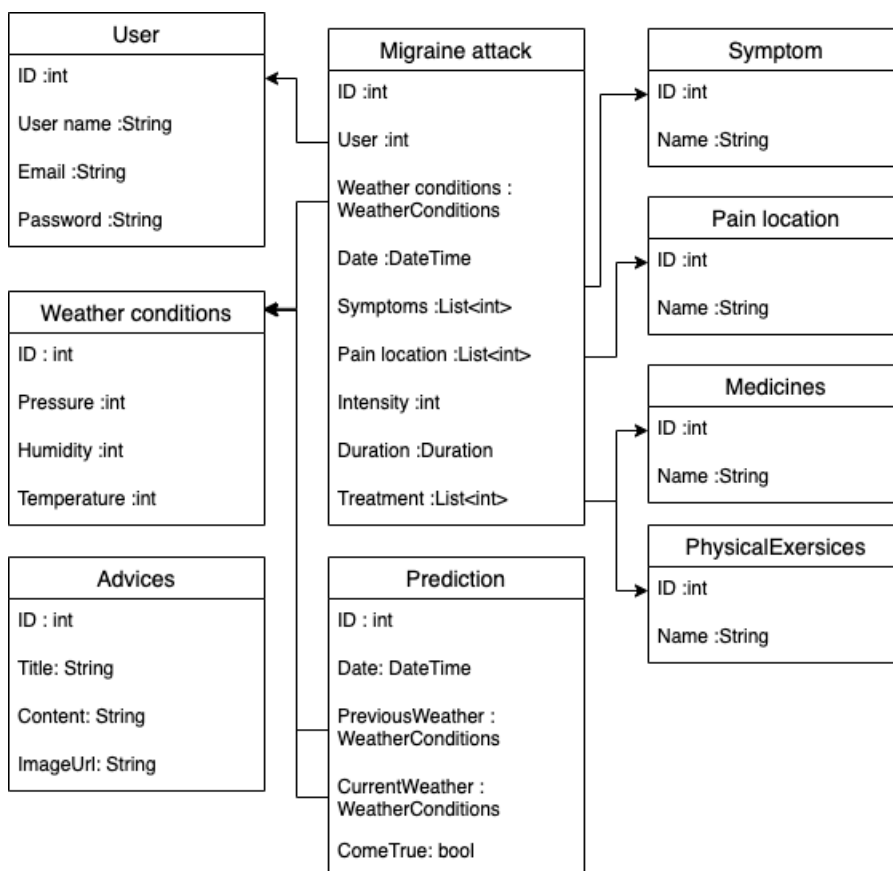


Рисунок 3.7 - Схема зв'язків моделей даних у застосунку

Field Type

= reference

Document path

Ex: collection/document001/

Cancel Add

Рисунок 3.8 - Створення поля таблиці з посиланням на документ

Також, потрібно зазначити, що допоміжної таблиці WeatherConditions - не існує, дані безпосередньо додано до записів, бо вірогідність їх дублювання дуже мала. Приклад такої вкладеності одного об'єкта у інший відображено на рис. 3.9,

де погодні умови вкладено у сутність передбачення.

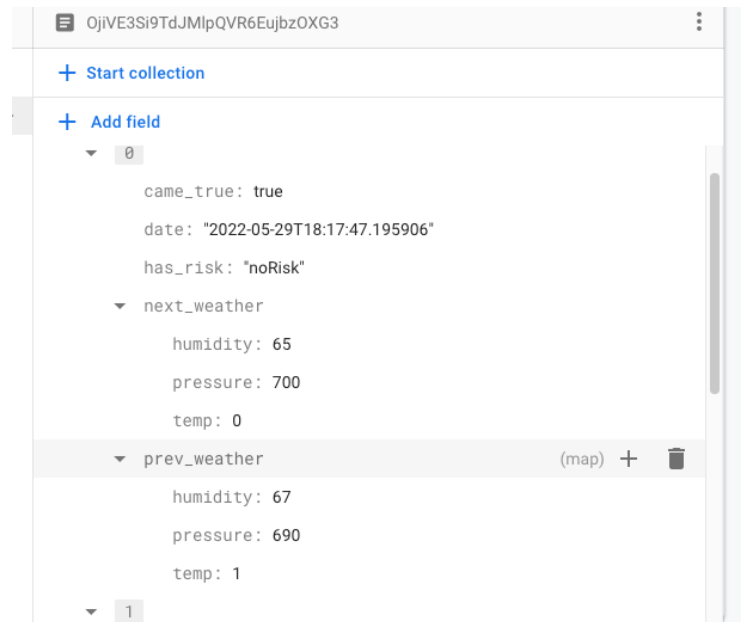


Рисунок 3.9 - Приклад вкладеності об'єктів

База даних застосунку представлена наступними колекціями:

- `advices` – зберігає поради, що надаються користувачу застосунку, для полегшення симптомів та попередження нападів;
- `attack` – зберігає записи про напади мігрені у користувачів;
- `medicines` – зберігає перелік препаратів, які можуть бути застосовані при нападі мігрені;
- `physical_exersices` – зберігає перелік фізичних вправ, що можуть полегшувати симптоми;
- `syntoms` – зберігає перелік симптомів, що можуть виникати під час нападу;
- `predictions` – зберігає передбачення щодо нападів мігрені у користувача;
- `users` – зберігає дані про користувача.

Сутність «`advice`», містить такі поля як: `adviceId`, `content`, `img_url` та `title` представлено табл. 3.1.

Таблиця 3.1 – Опис сутності «advice»

№	Назва	Тип
1	adviceId	int
2	content	String
3	img_url	String
4	title	String

Сутність «attack», містить такі поля як: alcohol_consumption, date, description, duration, head_injuring, noise_impact, pain_level, sleeping_time, stress_level, symptoms, treatments та weather_conditions представлено таблицею 3.2.

Таблиця 3.2 – Опис сутності «attack»

№	Назва	Тип
1	alcohol_consumption	bool
2	date	DateTime
3	description	String
4	duration	int
5	head_injuring	bool
6	noise_impact	bool
7	pain_level	int
8	sleeping_time	int
9	stress_level	int
10	symptoms	List<int>
11	treatments	List<int>
12	weather_conditions	WeatherConditions

Сутність «attack», використовує клас WeatherConditions, що містить такі поля як: temperature, pressure, humidity (табл. 3.3).

Таблиця 3.3 – Опис сутності «WeatherConditions»

№	Назва	Тип
1	2	3
1	temperature	int
2	pressure	int

Продовження таблиці 3.3

1	2	3
3	humidity	int

Сутність «prediction» містить user_id, prediction_list (табл. 3.4).

Таблиця 3.4 – Опис сутності «prediction»

№	Назва	Тип
1	user_id	String
2	prediction_list	List<PredictionModel>

У «prediction» використовується список, що складається з сутностей PredictionModel. Ця сутність містить наступні поля: came_true, date, has_risk, next_weather, prev_weather (табл. 3.5).

Таблиця 3.5 – Опис сутності PredictionModel

№	Назва	Тип
1	came_true	bool
2	date	DateTime
3	has_risk	RiskPrediction
4	next_weather	WeatherConditions
5	prev_weather	WeatherConditions

У «symptoms», «medicines», «physical_exercises» та «pain_location» однакова структура, вони містять поля: id, name (табл. 3.6).

Таблиця 3.6 – Опис сутностей «symptoms», «medicines», «physical_exercises» та «pain_location»

№	Назва	Тип
1	id	int
2	name	String

Підставні документи типу Symptoms, Medicines, PainLocation та PhysicalExercises стягуються із віддаленого сховища при завантаженні застосунку. До моделі Attack додаються лише їх id, щоб уникнути дублювання даних.

Застосунок також використовує Firebase Storage для зберігання медіафайлів - зображень, що використовуються для ілюстрації порад (рис.3.10). Він надається при створенні проекту у Firebase та легкий у використанні: файл після завантаження у сховище повертає посилання на себе. Це посилання додається у сутність документу «advices» як посилання на зображення.

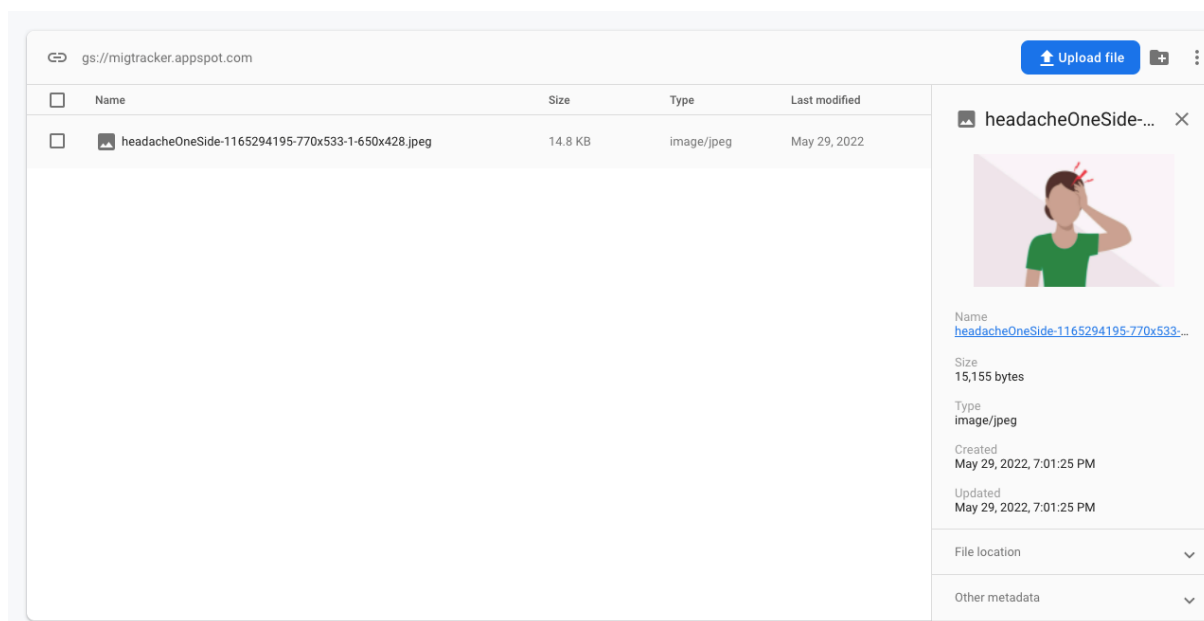


Рисунок 3.10 - Додане зображення у Firebase Storage

Перевагами зберігання даних у подібних віддалених сховищах, є зручність адміністрування, можливість встановлення окремих правил доступу до кожного з документів (рис. 3.11), а також отримання аналітики із користування сховищем, що необхідно для комерційного використання застосунку.

```

1  rules_version = '2';
2  service cloud.firestore {
3    match /databases/{database}/documents {
4      // True if the user is signed in or the requested data is 'public'
5      function signedInOrPublic() {
6        return request.auth != null && request.auth.uid != null || resource != null &&
7          resource.data.visibility == 'public';
8      }
9      match /users/{user} {
10       allow read, write: if signedInOrPublic();
11     }
12     match /medicines/{document=**} {
13       allow read, create: if signedInOrPublic();
14     }
15     match /physical_exercise/{document=**} {
16       allow read, create: if signedInOrPublic();
17     }
18     match /attack/{document=**} {
19       allow read, create: if signedInOrPublic();
20     }
21     match /predictions/{document=**} {
22       allow read, create: if signedInOrPublic();
23     }
24   }

```

Рисунок 3.11 - Правила, встановлені для документів у Cloud Firestore

3.4 Інтерфейс застосунку

Завдяки тому, що фреймворк Flutter сам відмальовує об'єкти, користувацький інтерфейс на всіх платформах виглядає однаково та не потребує налагодження для окремих нативних компонентів. Завдяки цьому інтерфейс застосунку може бути виконаний універсальним та відповідно до дизайнерського задуму.

Користувацький інтерфейс застосунку MigTracker представлений екранами, що наведені нижче.

Авторизаційний та реєстраційний екран містить поля для введення даних користувача, кнопку підтвердження та кнопки реєстрації за допомоги сторонніх сервісів (рис. 3.12).

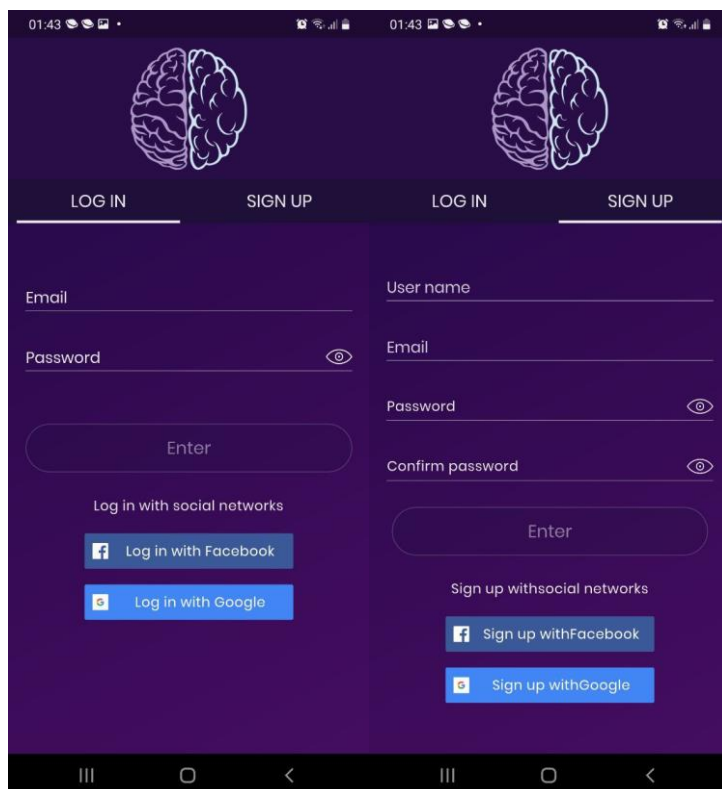


Рисунок 3.12 – Авторизаційна та реєстраційна сторінки застосунку

Після вдалої авторизації користувач потрапляє на домашню сторінку (рис.3.13), що відображує список нападів, графіки, що відображають статистику користувача за тривалістю нападів, найчастіших тригерів та найвживаніших препаратів (рис. 3.14), елементи для пошуку та сортування, кнопку додавання нападу.

При натисканні кнопки додавання нападу відкривається конструктор створення нового запису про напад (рис. 3.15). Він представлений трьома екранами: перший містить елементи для заповнення дати, часу, тривалості нападу та медикаментів, що були застосовані; другий екран містить елемент у формі головного мозку для визначення зони болю, а також елементи для вказання рівня болю та симптомів; третій екран відображує елементи для заповнення кількості годин сну, погодних умов та впливу зовнішніх факторів, а також кнопки для простого додавання запису та початку нападу мігрені.

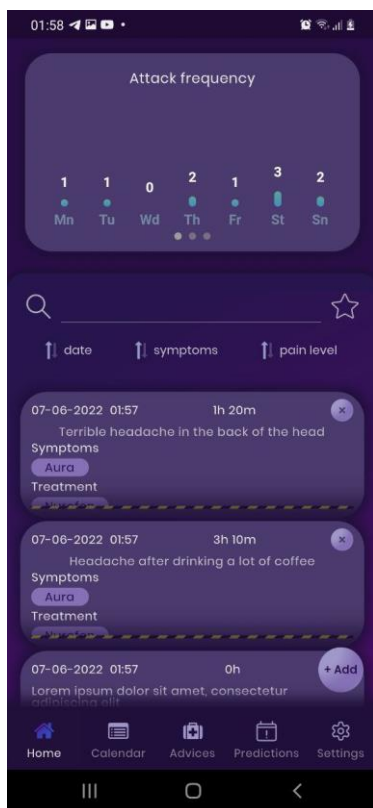


Рисунок 3.13 – Домашня сторінка застосунку

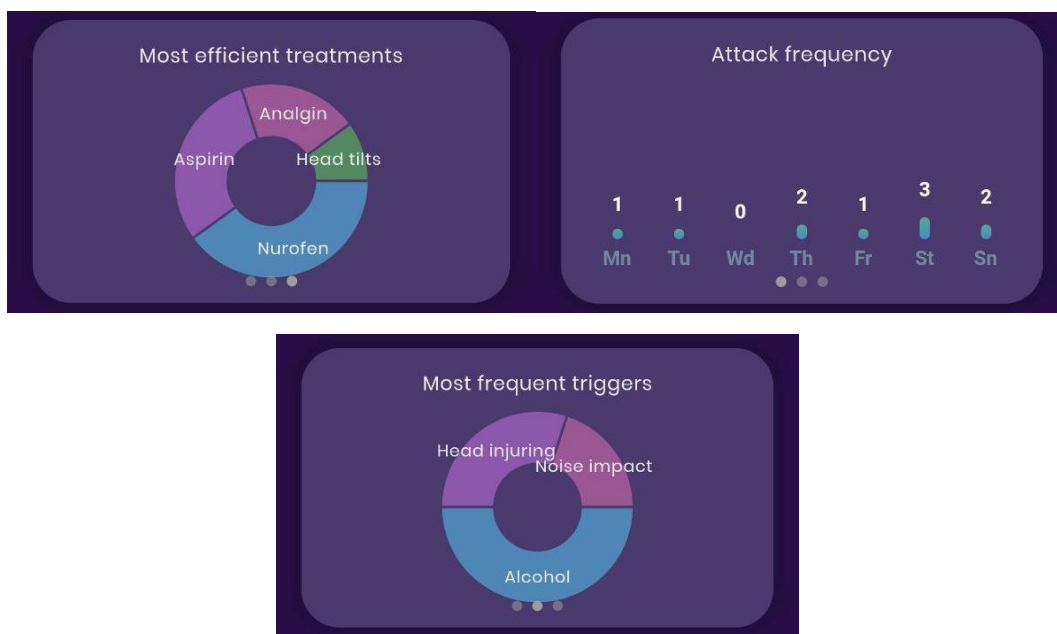


Рисунок 3.14 – Графіки, що візуалізують статистичні дані

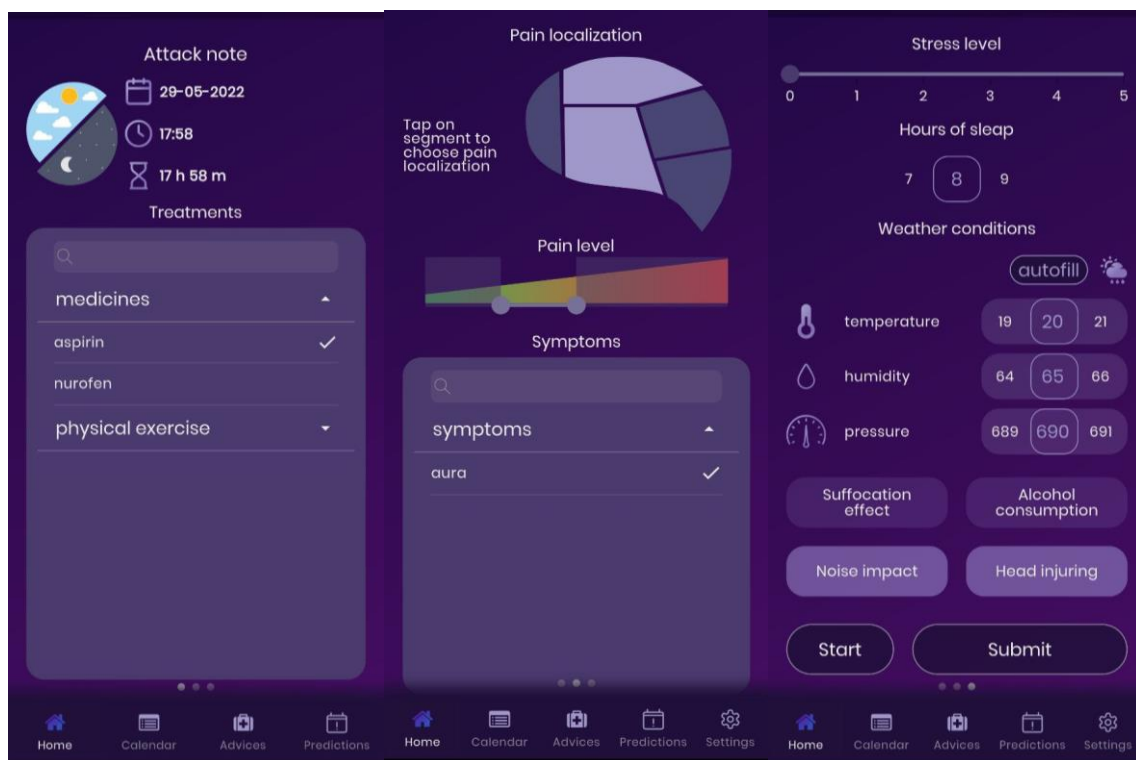


Рисунок 3.15 – Сторінки конструктору нападу

Після потрапляння на домашню сторінку користувач для навігації у застосунку може використовувати елемент, що називається таб-бар (рис. 3.16) та знаходиться у нижній частині сторінки. За його допомоги можна перейти на сторінки з календарем, порадами, передбаченнями та у налаштування.

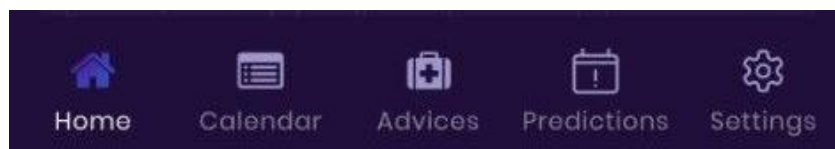


Рисунок 3.16 – Елемент навігації у застосунку

Для відображення навігаційної структури застосунку наведено схему інформаційної архітектури у якій відображено переходи між сторінками застосунку (рис. 3.17).

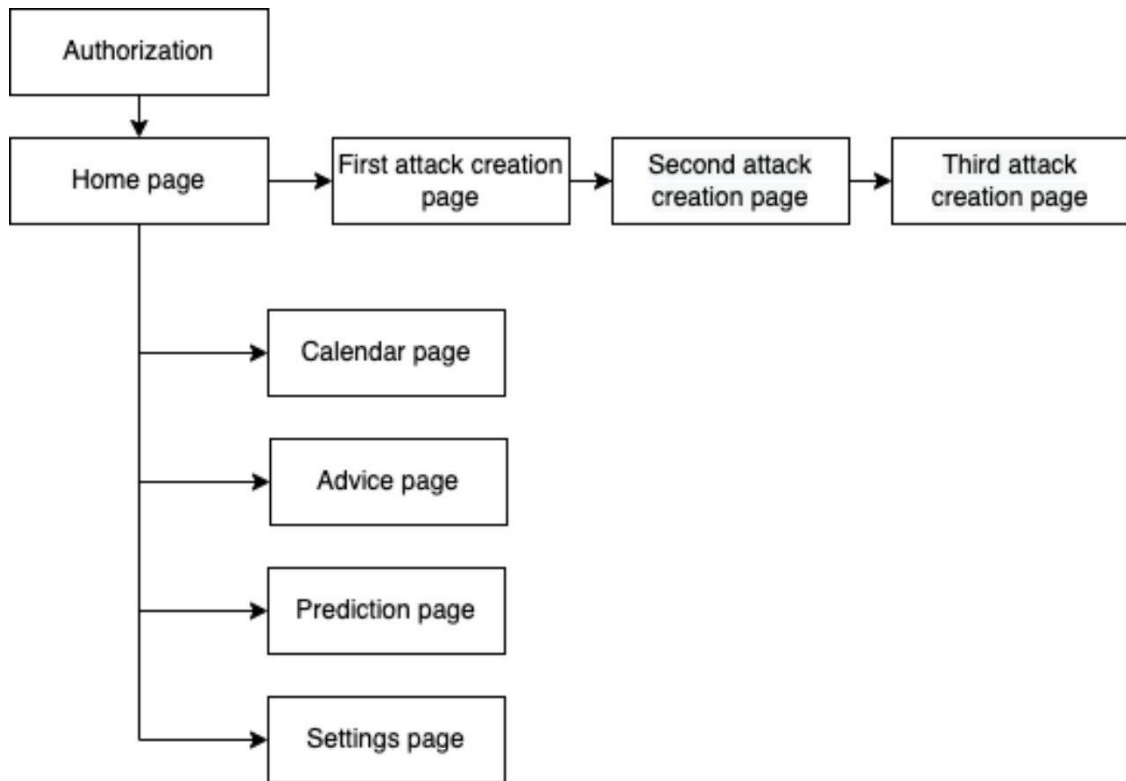


Рисунок 3.17 – Інформаційна архітектура застосунку

3.5 Висновки за 3 розділом

У третьому розділі було наведено опис основних алгоритми застосунку. Враховуючи особливості предметної області та обраної архітектури було спроектовано взаємодію компонентів та наведено їх опис.

Також пояснено спосіб зберігання даних у віддаленому сховищі Firestore, та особливості авторизації користувачів, як за допомоги поштової адреси, так і сторонніх сервісів. Також наведено опис сутностей та правил доступу до документів.

Розроблено інтерфейс користувача, розглянуто його складові та їх призначення, наведено інформаційну архітектуру застосунку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення та умови виконання програми

Призначенням програми «MigTracker» є облік записів про напади мігрені, ведення статистики, надання користувачу порад та попередження про напади. Це дозволить користувачу полегшити ведення щоденнику за нападів мігрені, а також краще розуміти частоту та причину їх виникнення.

Програма виконує наступні функції:

- авторизація та реєстрація користувачів;
- створення запису про мігрень;
- встановлення нового запису про напад, як того що триває;
- перегляд, сортування та пошук записів;
- зберігання записів у вигляді календаря;
- конвертація записів у файл формату .pdf;
- відкриття записів для перегляду детальної інформації;
- надання статистики у вигляді графіків;
- передбачення нападу за погодними даними;
- надання користувача порад щодо полегшення симптомів та попередження нападу;
- зміна мови застосунку.

Для нормального функціонування програми, повинні виконуватися наступні умови до апаратної частини клієнта:

- мінімальна версія Android – 7.0, або IOS – 9.0, проте цільові версії це 11.0 та 13.0 та вище відповідно;
- не менше 4 гб оперативної пам'яті;
- не менше 100 мб вільної пам'яті девайсу;

- екран девайсу не менше 1331x750;
- підтримка Google сервісів;
- наявність інтернет зв'язку.

Для виконання програми у середовищі розробки необхідні наступні умови до апаратної та програмної частин персонального комп'ютера:

- ОС Windows не нижче 10 версії (тільки для запуску Android) або MacOS не нижче 11 версії;
- не менше 4 гб оперативної пам'яті;
- не менше 4 гб вільної пам'яті девайсу;
- встановлене одне з середовищ розробки: AndroidStudio, VSCode, IntelliJ Idea;
- встановлений Flutter SDK.

Для запуску програми перший раз, за всіх попередньо зазначених умов, розробник має виконати команду ``flutter pub get`` - для завантаження бібліотек, що використовує застосунок, потім побудувати залежності програми командою ``flutter pub run build_runner build --delete-conflicting-outputs``. Після цього застосунок може бути виконано або за допомоги інтерфейсу середовища розробки або командою ``flutter run``.

4.2 Характеристики програми

Проект виконано за допомогою мови програмування високого рівню Dart та фреймворку Flutter. У якості шаблону проектування обрано чисту архітектуру. Застосунок створювався у середовищі розробки Android Studio Bumblebee 2021. Застосунок містить папку `lib`, що безпосередньо зберігає код програми, а також папки та файли, що описані у розділі 2.6. Окремо потрібно зазначити папку `assets`, яка містить файли, що використовуються застосунком у якості медіафайлів.

Assets містить в собі папку images, що містить файли .png розширення, lottie – зберігає файли анімації формату .json та icons – файли іконок у форматі .svg.

4.3 Звернення до програми

Для звернення до програми користувачам, що використовують девайс з ОС Android необхідно встановити на нього файл збірки застосунку з розширенням .apk. Після встановлення, необхідно запустити програму та пройти авторизацію. Для власників девайсів з системою IOS, встановлення застосунку не через офіційні джерела, як TestFlight та Appstore, є неможливим через політику безпеки. Для того, щоб викласти застосунок у Appstore необхідно мати акаунт розробника Apple, створити на офіційному порталі проект, завантажити туди збірку з розширенням .ipa та додати супроводжуючу документацію. Після цього застосунок буде перевірено та розглянуто до публікації.

Після завантаженні застосунку у користувача запитується дозвіл на використання геолокації (рис. 4.1).

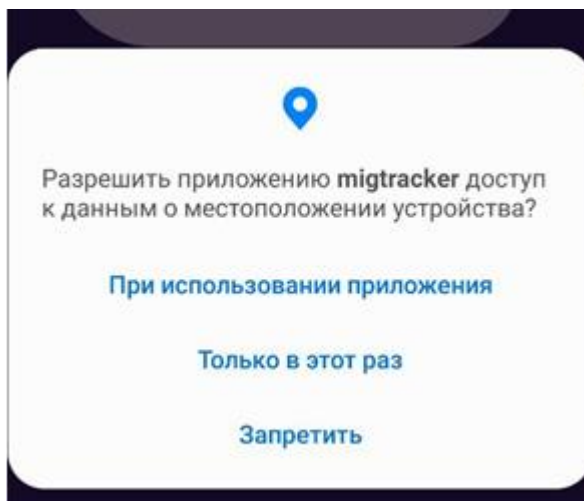


Рисунок 4.1 – Нативне вікно запиту дозволу на геолокацію

Після авторизації користувач переходить на головний екран застосунку

(рис. 3.12, розд. 3). Для того щоб створити запис про напад необхідно натиснути кнопку додавання нападу (рис 4.2) у нижньому правому кутку.



Рисунок 4.2 – Кнопка додавання нападу

Викликається конструктор нападів (рис. 3.13, розд. 3), а після заповнення користувач може почати або додати напад за допомоги двох кнопок (рис. 4.3).



Рисунок 4.3 – Кнопки початку та додавання нападу

Якщо натиснути кнопку «Start», на головному екрані з'явиться блок з поточним нападом (рис. 4.4).

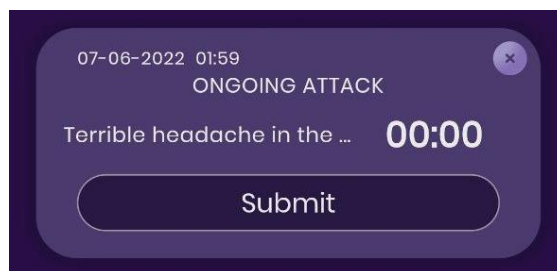


Рисунок 4.4 – Відображення поточного нападу на головному екрані

При закінчення нападу, користувач отримає повідомлення, що запитує підтвердження дії (рис. 4.5).

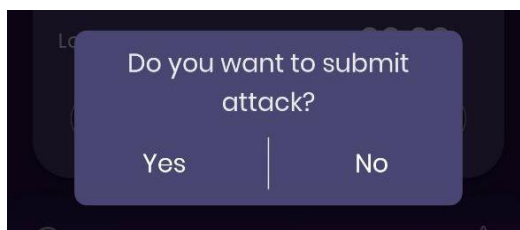


Рисунок 4.5 – Повідомлення користувачу про підтвердження нападу

Якщо користувач хоче отримати виписку своїх нападів у форматі .pdf, йому необхідно перейти на екран «Календар» (рис. 4.6) та натиснути кнопку «to pdf» й отримати повідомлення з вибором періоду (рис. 4.7).

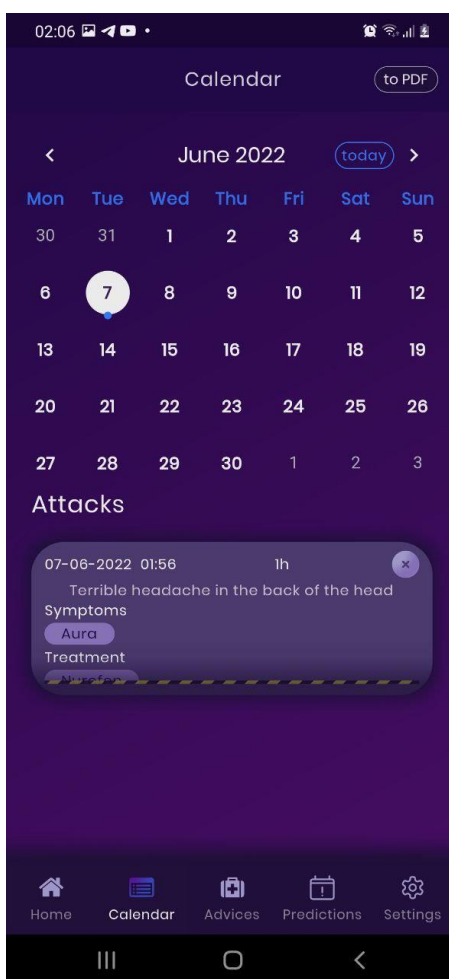


Рисунок 4.6 – Екран календаря



Рисунок 4.7 – Повідомлення з вибором періоду нападів для конвертації

Також користувач може перейти на сторінку «Поради» (рис. 4.8) або сторінку «Передбачення» (рис. 4.8). На сторінці передбачення, дійсність передбачення можна підтвердити або відхилити за допомоги кнопок «Relevant» та «Not relevant» (рис.4.9).

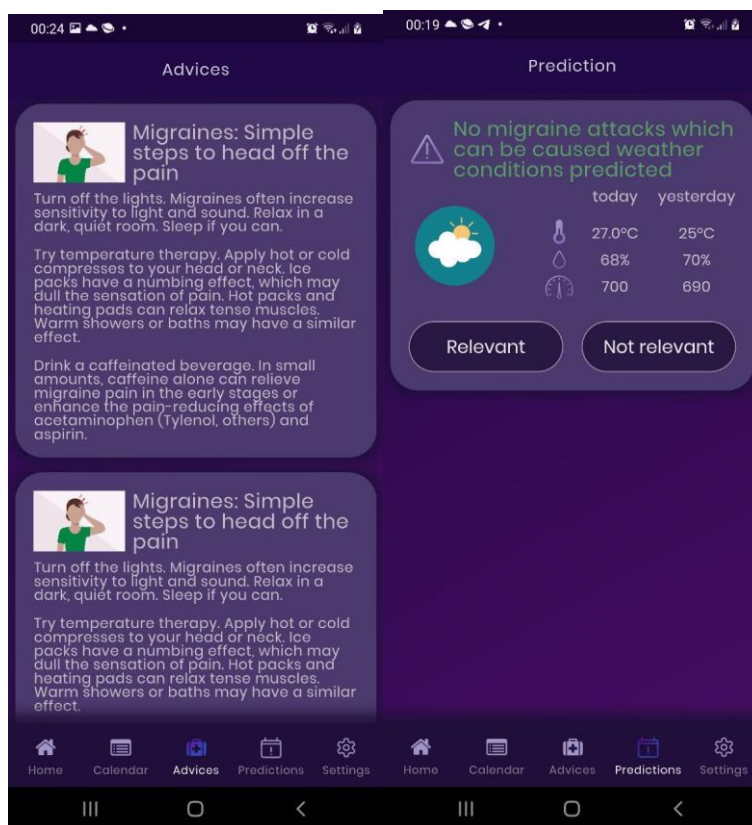


Рисунок 4.8 – Екран порад та передбачень



Рисунок 4.9 – Кнопки підтвердження дійсності передбачення

Якщо користувачу необхідно змінити мову, то необхідно перейти на екран налаштувань (рис. 4.10) та переключити мову за допомогою перемикача. Також з цього екрану можна здійснити вихід з застосунку.



Рисунок 4.10 – Екран налаштувань

4.3 Тестування

Тестування у Flutter може відбуватися за допомогою юніт-тестування, віджет-тестування, інтеграційного-тестування. Юніт-тестування полягає у тестуванні окремих логічних компонентів. При віджет-тестуванні перевіряються

окремі елементи інтерфейсу, а інтеграційні поєднують як тестування логічних компонентів, так і віджетів.

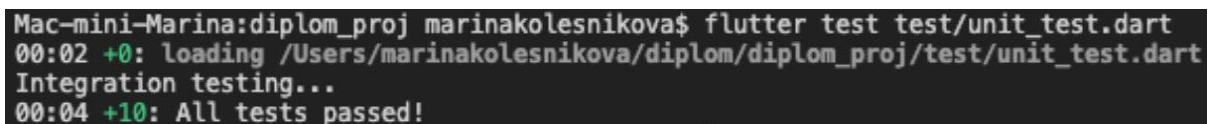
У застосунку MigTracker юніт-тестами покрито наступні компоненти:

- text_validation_util.dart – утиліта для валідації даних;
- units_convertation_util.dart – утиліта для конвертації одиниць виміру температури із F у °C та тиску із Pa у мм ртутного стовпця;
- login_event.dart – подія авторизації користувача;
- registration_event.dart – подія реєстрації користувача;
- fetch_attack_event.dart – подія для отримання даних із віддаленого сховища;
- set_new_attack_event.dart – подія для створення нападу;
- create_prediction_event.dart – подія для створення передбачення на основі погодних умов;
- init_app_event.dart – подія ініціалізації застосунку;
- set_settings_event.dart – подія для встановлення постійних налаштувань застосунку.

Також було розроблено інтеграційні тести для перевірки роботи логіки та віджетів. Інтеграційні віджети було написано для:

- auth_integration_test.dart – для віджетів авторизаційного компоненту;
- create_attack_integration_test.dart – для віджетів, що приймають участь у створенні.

Результати виконання тестів наведено на рис. 4.11.



```
Mac-mini-Marina:diplom_proj marinakolesnikova$ flutter test test/unit_test.dart
00:02 +0: loading /Users/marinakolesnikova/diplom/diplom_proj/test/unit_test.dart
Integration testing...
00:04 +10: All tests passed!
```

Рисунок 4.11 – Результати виконання тестування

Тестування застосунку також проведено за наступним чек-листом (табл. 4.1 – 4.5).

Таблиця 4.1 – Чек-лист тестування функціональності авторизації користувача

Перевірка	Android	IOS
Введення авторизаційних даних	+	+
Авторизація за допомоги поштової адреси	+	+
Авторизація за допомоги сервісів Google	+	+
Авторизація за допомоги сервісів Facebook	+	+
Реєстрація користувача за допомоги авторизаційних даних	+	+
Реєстрація користувача за допомоги сервісів Google	+	+
Реєстрація користувача за допомоги сервісів Facebook	+	+
Валідація даних	+	+
Перехід на головну сторінку застосунку	+	+

Таблиця 4.2 – Чек-лист тестування функціональності створення нападу

Перевірка	Android	IOS
Відкриття конструктору для створення нападів	+	+
Заповнення даних про напад	+	+
Підтвердження завершення нападу	+	+
Відкладене завершення нападу	+	+
Редагування нападу	+	+

Таблиця 4.3 – Чек-лист тестування функціональності домашньої сторінки

Перевірка	Android	IOS
Відображення списку нападів	+	+
Пошук нападів	+	+
Сортування нападів	+	+
Видалення нападу	+	+
Відображення статистичних даних	+	+
Відкриття конструктору для створення нападів	+	+

Таблиця 4.4 – Чек-лист тестування функціональності сторінки календаря

Перевірка	Android	IOS
Відображення календаря	+	+
Вибір дати	+	+
Повернення на поточну дату	+	+
Перехід між місяцями	+	+
Завантаження файлу у pdf форматі	+	+

Таблиця 4.5 – Чек-лист тестування передбачень

Перевірка	Android	IOS
Відображення передбачення в залежності від наявності ризику	+	+
Підтвердження передбачення	+	+
Відхилення передбачення	+	+

Усі тест-кейси було пройдено вдало, поведінка застосунку є передбачуваною на обох платформах, зокрема через відсутність платформно-залежної функціональності. Тестування показало стабільність роботи застосунку

та його відповідність висунутим до нього функціональним умовам.

4.4 Висновки за 4 розділом

У червертому розділі розглянуто призначення програми, мінімальні необхідні вимоги до апаратно-програмного забезпечення користувача. Описано налаштування середовища розробки проекту.

Також було вказано характеристики програми та спосіб звернення до неї. Наведено приклад використання основного функціоналу застосунку та способи діалогу за користувачем.

Для перевірки надійності було проведено тестування бізнес-логіки та презентаційного рівня програми було проведено юніт- та інтеграційне-тестування, а також перевірку за допомогою чек листу.

ВИСНОВОК

За результатами виконання дипломної роботи було розглянуто проблемну область застосунку, проаналізовано існуючі аналоги програм, таких як: «Migraine Monitor», «Migraine and headache diary» та «Migraine Buddy».

Аналоги було порівняно з застосунком «MigTracker» та доведено, що розробка цього мобільного застосунку є актуальною та доцільною. Виходячи цього, було проведено аналіз існуючих технологій за результатами якого було обрано мову програмування Dart та фреймворк Flutter. Також було порівняно середовища програмування, та за низкою переваг обрано середовище розробки Android Studio.

У якості джерала даних для застосунку було обрано віддалене сховище Firestore Database від компанії Google. Також було розглянуто найпопулярніші засоби керування станами у Flutter, та за результатами аналіз обрано BLoC.

У третьому розділі було наведено опис основних алгоритми застосунку. Враховуючи особливості предметної області та обраної архітектури було спроектовано взаємодію компонентів та наведено їх опис.

Розроблено інтерфейс користувача, розглянуто його складові та їх призначення, наведено інформаційну архітектуру застосунку.

У результаті було розроблено мобільний застосунок «Migtracker», що надає функціонал для створення записів про напади, їх відображення та конвертацію, надання статистичних даних. А також відображення порад для попередження нападів та надання передбачень на основі погодних умов.

В подальшому планується підвищити зручність застосунку шляхом впровадження спливаючих повідомлень, як про можливість нападу, так і нагадування про наявність нападу, що триває.

ПЕРЕЛІК ПОСИЛАНЬ

1. World Health Organization [Electronic resource]. – Access mode: <https://www.who.int/news-room/fact-sheets/detail/headache-disorders>
2. Словник української мови: в 11 тт. АН УРСР. Інститут мовознавства / [за ред. І. К. Білодіда]. — К.: Наукова думка, — Т. 9. — 1978. — 177 с.
3. GooglePlaymigraineapp [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.healint.migraineapp>
4. GooglePlayheadachelog [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.samantharoobol.headachelog>
5. Flutter docs - Differentiate between ephemeral state and app state [Electronic resource]. – Access mode: <https://docs.flutter.dev/development/data-and-backend/state-mgmt/ephemeral-vs-app>
6. Dart dev - What not to commit [Electronic resource]. – Access mode: <https://dart.dev/guides/libraries/private-files#the-rules>
7. Friedman D. Migraine and the Environment / D. Friedman, T. De Ver // Headache: The Journal of Head and Face Pain. – 2009. – Vol. 49. – P. 941-952.
8. Yang A. C. Temporal Associations between Weather and Headache : Analysis by Empirical Mode Decomposition / A. C. Yang, J. Fuh, N. Huang, B. Shia, Ch. Peng, Sh. Wang // PLoS One. – 2011. – Vol. 4. – 790 p.
9. Okuma H. Examination of fluctuations in atmospheric pressure related to migraine / H. Okuma, Y. Okuma, Y. Kitagawa // Springerplus. – 2015. – Vol. 4. – P. 790-796.
10. Mukamal K. Weather and air pollution as triggers of severe headaches / K. Mukamal, G. Wellenius, H. Suh, M. Mittleman. // Neurology. – 2009. – Vol. 72. – P. 922 – 927.
11. Wenyuan L. Weather, ambient air pollution, and risk of migraine headache onset among patients with migraine / L. Wenyuan, S. Bertisch, E. Mostofsky //

Environment International. – 2019. – Vol. 132.

12. Yang A. Patients with migraine are right about their perception of temperature as a trigger: time series analysis of headache diary data / A. Yang, J. Fuh, N. Huang, B. Shia // The Journal of Headache and Pain. – 2015. – Vol. 16. – P. 533-542.

13. Firebase Documentation - Cloud Firestore [Electronic resource]. – Access mode: <https://firebase.google.com/docs/firestore>

ДОДАТОК А
Технічне завдання

Вступ

У зв'язку з всесвітньою тенденцією до диджиталізації в усіх сферах життя, а також тим, що смартфон майже завжди є під рукою, найзручніший спосіб вести записи із допомоги смартфона, що завжди знаходиться під рукою. Саме тому актуальності набувають застосунки-трекери, що передбачають інтерфейс для зручного занотовування та перегляду записів. та мають єдину структуру запису, що полегшує аналіз даних.

Програмний засіб (ПЗ) призначено для обліку записів про напади мігрені, ведення статистики, надання користувачу порад та попередження про напади. Це дозволить користувачу полегшити ведення щоденнику за нападів мігрені, а також краще розуміти частоту та причину їх виникнення.

A.1 Назва програми

Мобільний застосунок має назву «MigTracker».

A.2 Підстави для розробки

Підставою для розробки є дипломна кваліфікаційна робота бакалавра з теми «Програмна реалізація мобільного застосунку-трекера нападів мігрені у людей» згідно наказу №102 від 27 квітня 2022 р.

A.3 Призначення розробки

Застосунок призначено для обліку нападів мігрені, отримання аналітичних даних, порад та передбачень на основі погодних умов, а також отримання виписки за нападами у форматі .pdf.

A.4 Вимоги до програми

A.4.1 Вимоги до функціональних характеристик

До застосунку висунуто наступні функціональні вимоги:

- створення, редагування та зберігання записів про напади мігрені з єдиною структурою;
- візуалізація статистичних даних з тривалості нападів, найчастіших тригерів, найбільш застосовуваних ліків;
- відображення записів у вигляді списку;
- відображення записів у календарі;
- конвертація записів до файлу формату PDF;
- сортування записів;
- надання рекомендації користувачу щодо полегшення симптомів;
- можливість передбачення нападів за погодними умовами;
- можливість зміни мови застосунку.

A.4.2 Вимоги до надійності

Застосунок «MigTracker» має вміти оброблювати ситуації некоректної авторизації користувача – повідомляти про існуючий акаунт або некоректні дані. Також необхідним є підтвердження для користувачів операцій, що є незворотними, наприклад, завершення нападу, що тривав – користувач має отримати повідомлення та підтвердити або скасувати операцію. А через те, що деякий функціонал застосунку залежить від наявності інтернет зв'язку, то за його відсутності користувач має бути попереджений про це.

A.4.3 Умови експлуатації

Для нормального функціонування застосунку, до девайсу користувача висунуто наступні програмно-апаратні вимоги:

- мінімальна версія Android – 7.0, або IOS – 9.0;
- не менше 4 гб оперативної пам'яті;
- не менше 100 мб вільної пам'яті девайсу;
- екран девайсу не менше 1331x750;
- підтримка Google сервісів;
- наявність інтернет зв'язку.

A.5 Стадії і етапи розробки

A.5.1 Стадії розробки

Розробка програмного продукту пройшла 4 стадії:

- аналіз вимог;
- проектування;
- розробка програми;
- розробка документації.

A.5.2 Етапи розробки

На першій стадії – «Аналіз вимог» виконують наступні етапи:

- збір вимог;
- аналіз та систематизування вимог;
- встановлення та вирішення суперечностей.

На стадії «Проектування»:

- аналіз існуючих технологій;
- вибір архітектури застосунку;
- проектування БД;
- вивчення можливостей програмної реалізації структур даних та інтерфейсу користувача;
- аналіз вимог до ПЗ.

На стадії «Розробка програми»:

- розробка алгоритмів;
- написання програмного коду;
- тестування ПЗ.

На стадії «Розробка документації»:

- розробка опису програми;
- розробка супроводжуючої документації.

A.6 Порядок контролю та приймання

Контроль та приймання застосунку здійснюються згідно порядку приймання дипломних кваліфікаційних робіт бакалаврів. Програмний продукт має відповідати усім заявленим у розділі A.4 вимогам.

ДОДАТОК Б
Код програми

Authorization_screen.dart

```
import 'package:diplom_proj/presentation/authorization/log_in/login_screen.dart';
import 'package:diplom_proj/presentation/authorization/sign_up/sign_up_screen.dart';
import
'package:diplom_proj/presentation/shared/scroll_component/scroll_clean_behavior.dart';
import
'package:diplom_proj/presentation/shared/widgets/bru_top_tab_bar/models/tab_bar_model.dart';
import
'package:diplom_proj/presentation/shared/widgets/bru_top_tab_bar/top_tab_bar_builder.dart';
import 'package:diplom_proj/presentation/shared/widgets/logo/logo.dart';
import 'package:diplom_proj/resources/resources.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:diplom_proj/src/shared/layouts/main_layout.dart';
import 'package:diplom_proj/src/auth/domain/bloc/auth_bloc.dart';

class AuthorizationScreen extends StatelessWidget {
  const AuthorizationScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final AuthDictionary dict = dictionaryManager.getSelectedData.auth;
    final Widget sliverAppBar = SliverAppBar(
      expandedHeight: 160.0,
      backgroundColor: LightColors.backgroundColor,
      automaticallyImplyLeading: false,
      flexibleSpace: const FlexibleSpaceBar(
        background: Logo(),
        collapseMode: CollapseMode.none,
      ),
    );

    return BlocProvider(
      create: (ctx) => AuthBloc.byGetIt(),
      child: MainLayout.withFocusLayout(
        resizeToAvoidBottomInset: true,
        child: ScrollConfiguration(
          behavior: ScrollCleanBehavior(),
          child: TopTabBarBuilder.sliverTopTabBar(
            header: sliverAppBar,
            elements: [
              TabBarModel(
                title: dict.login,
                page: LogInScreen(),
              ),
              TabBarModel(
                title: dict.signUp,
                page: SignUpScreen(),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Login_screen.dart

```
import
'package:diplom_proj/presentation/authorization/log_in/mixins/log_in_screen_mixin.dart';
import
'package:diplom_proj/presentation/authorization/log_in/widgets/login_fields.dart';
import
'package:diplom_proj/presentation/shared/scroll_component/scroll_clean_behavior.dart';
import
'package:diplom_proj/presentation/shared/widgets/buttons/outlined_button.dart/light_outl
ined_button.dart';
import
'package:diplom_proj/presentation/shared/widgets/sign_with_buttons/builders/sign_with_bu
ilder.dart';
import 'package:diplom_proj/src/auth/domain/bloc/auth_bloc.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:auto_route/src/router/auto_router_x.dart';
import 'package:diplom_proj/config/modules/injection_config.dart';
import 'package:diplom_proj/resources/resources.dart';

import 'package:flutter_form_builder/flutter_form_builder.dart';

class LogInScreen extends StatefulWidget {
  const LogInScreen({Key? key}) : super(key: key);

  @override
  State<LogInScreen> createState() => _LogInScreenState();
}

class _LogInScreenState extends State<LogInScreen> with LogInScreenMixin {
  bool isClicked = false;
  @override
  Widget build(BuildContext context) {
    return FormBuilder(
      key: formKey,
      child: Center(
        child: ScrollConfiguration(
          behavior: ScrollCleanBehavior(),
          child: BlocBuilder<AuthBloc, AuthState>(
            builder: (context, state) {
              return ListView(
                physics: ClampingScrollPhysics(),
                shrinkWrap: true,
                children: [
                  LoginFields(
                    loginController: loginController,
                    passwordController: passwordController,
                    onChange: onChange,
                    isError: errorText != null,
                  ),
                  if (errorText != null)
                    Padding(
                      padding: const EdgeInsets.only(top:
20.0).add(EdgeInsets.symmetric(horizontal: 20.0)),
                      child: Text(
                        errorText!,
                        key: const Key('errorText'),
```

```

        style: LightTextStyles.poppinsS12W400(color:
LightColors.errorColor),
      ),
    ),
    Align(
      alignment: Alignment.bottomCenter,
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal:
20.0).add(EdgeInsets.only(top: 60.0)),
        child: LightOutlineButton(
          color: LightColors.text,
          isDisable: isButtonDisable,
          onTap: onLogin,
          text: dict.enter,
        ),
      ),
    ),
    SignWithBuilder(
      signWithType: SignWithType.logIn,
      padding: const EdgeInsets.symmetric(horizontal: 87.0).add(const
EdgeInsets.only(top: 18.0)),
    ),
  ],
);
},
),
),
),
);
}

void onLogin() {
  final authBloc = context.read<AuthBloc>();
  authBloc.add(
    LoginEvent(
      email: loginController.text,
      password: passwordController.text,
      onLoginCompleted: () {
        context.router.replaceNamed(AppRoutes.homeScreen);
      },
      onError: onError,
    ),
  );
}
}

```

Auth_repository.dart

```

import 'package:dio/dio.dart';
import 'package:diplom_proj/config/app_config.dart';
import 'package:diplom_proj/config/modules/dio_module_config.dart';
import 'package:diplom_proj/presentation/messages/scaffold_messenger.dart';
import 'package:diplom_proj/presentation/shared/mixins/logout.dart';
import 'package:diplom_proj/src/auth/data/api/auth_api.dart';
import 'package:diplom_proj/src/auth/entity/dto/email_dto/email_dto.dart';
import 'package:diplom_proj/src/auth/entity/dto/login_dto/login_dto.dart';
import 'package:diplom_proj/src/auth/entity/dto/sign_up_dto/sign_up_dto.dart';

```

```

import 'package:diplom_proj/src/auth/entity/dto/social_token/social_token_dto.dart';
import 'package:diplom_proj/src/auth/entity/models/social_token/social_token.dart';
import 'package:diplom_proj/src/auth/entity/models/token/token.dart';
import 'package:diplom_proj/src/shared/interfaces/i_base_repository.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:injectable/injectable.dart';

@lazySingleton
class AuthRepositoryUnauthorized extends IBaseRepository<AuthApi> {
  AuthRepositoryUnauthorized(
    @Named(unauthorized) Dio dio,
    this.config,
  ) : super(AuthApi(dio, baseUrl: config.baseUrl));

  final AppConfig config;

  Future<User?> login(LoginDTO body) async {
    try {
      final _auth = FirebaseAuth.instance;
      final userCredential = await _auth.signInWithEmailAndPassword(
        email: body.email,
        password: body.password,
      );
      return userCredential.user;
    } on FirebaseAuthException catch (e) {
      if (e.code == 'account-exists-with-different-credential') {
        InfoMessages.accountExistsMessage();
      } else if (e.code == 'invalid-credential' || e.code == 'wrong-password') {
        InfoMessages.wrongCredentialsMessage();
      }
      return null;
    } catch (e) {
      return null;
    }
  }

  Future<User?> signup(SignUpDTO body) async {
    try {
      final _auth = FirebaseAuth.instance;
      final userCredential = await _auth.createUserWithEmailAndPassword(
        email: body.email,
        password: body.password,
      );
      return userCredential.user;
    } on FirebaseAuthException catch (e) {
      if (e.code == 'account-exists-with-different-credential') {
        InfoMessages.accountExistsMessage();
      } else if (e.code == 'invalid-credential') {
        InfoMessages.wrongCredentialsMessage();
      }
      return null;
    } catch (e) {
      return null;
    }
  }

  Future<void> refreshToken() async {
    await FirebaseAuth.instance.currentUser?.getIdTokenResult(true);
  }

  Future<void> logout() async {

```

```

    await FirebaseAuth.instance.signOut();
  }
}

```

Main_screen.dart

```

import 'package:auto_route/auto_route.dart';
import 'package:diplom_proj/config/app_router.gr.dart';
import 'package:diplom_proj/presentation/main/widgets/active_attack.dart';
import 'package:diplom_proj/presentation/main/widgets/attack_list.dart';
import 'package:diplom_proj/presentation/main/widgets/charts_panel.dart';
import
'package:diplom_proj/presentation/shared/widgets/non_animated_inkwell.dart/non_animated_
inkwell.dart';
import 'package:diplom_proj/presentation/shared/widgets/prefered_size_container.dart';
import
'package:diplom_proj/presentation/shared/widgets/search_field/search_sliver_header_deleg
ate.dart';
import 'package:diplom_proj/resources/resources.dart';
import 'package:diplom_proj/src/attack/domain/attack_bloc.dart';
import
'package:diplom_proj/src/attack/domain/events/fetch_tretments_symptoms_event.dart';
import 'package:diplom_proj/src/attack/domain/events/set_new_attack_event.dart';
import 'package:diplom_proj/src/attack/entities/attack_model/attack_model.dart';
import 'package:diplom_proj/src/shared/layouts/main_layout.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class MainScreen extends StatelessWidget {
  const MainScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<AttackBloc, AttackState>(builder: (context, state) {
      final ongoingAttack = AttackModel.mock();
      //state.ongoingAttack;
      final additionalHeight = ongoingAttack != null ? 200.0 : 0;
      return MainLayout.withFocusLayout(
        floatingActionButton: getFloatingButton(context),
        child: Container(
          decoration: getContainerDecoration,
          child: CustomScrollView(
            slivers: [
              SliverAppBar(
                backgroundColor: Colors.transparent,
                expandedHeight: 270.0 + additionalHeight,
                bottom: PreferredSizeContainer(
                  isShadow: true,
                  height: 20.0,
                  color: LightColors.lightUndercover,
                ),
              ),
              flexibleSpace: FlexibleSpaceBar(
                background: ColoredBox(
                  color: LightColors.backgroundColor,
                  child: Padding(
                    padding: const EdgeInsets.symmetric(
                      vertical: 8.0,

```

```

        horizontal: 20.0,
      ),
      child: Column(
        children: [
          ChartsPanel(),
          if (ongoingAttack != null) ActiveAttack(ongoingAttack:
ongoingAttack),
        ],
      ),
    ),
  ),
),
),
),
SliverPersistentHeader(
  pinned: true,
  delegate: SearchSliverHeaderDelegate(),
),
SliverPadding(
  padding: const EdgeInsets.symmetric(horizontal: 15.0, vertical: 8.0),
  sliver: AttackList(),
)
],
),
),
);
});
}

```

```
Widget getFloatingActionButton(BuildContext context) {
  return NonAnimatedInkWell(
    onTap: () {
      context.read<AttackBloc>().add(
        FetchTreatmentsSymptomsEvent(
          onCompleted: (() {
            context.read<AttackBloc>().add(
              SetNewAttackEvent(
                onCompleted: (() {
                  context.pushRoute(AttackCreationRoute());
                }
              ),
            ),
          ),
        ),
      ),
    ),
  );
},
child: Container(
  height: 50.0,
  width: 50.0,
  decoration: BoxDecoration(
    gradient: RadialGradient(
      center: Alignment.topLeft,
      radius: 1,
      colors: LightColors.onPanelIcon,
      tileMode: TileMode.mirror,
      stops: [0.5, 0.9],
    ),
    borderRadius: BorderRadius.circular(50.0),
  ),
  child: Center(
    child: Text(
      '+ Add',
    ),
  ),
),
);
}
```

```

        style: LightTextStyles.poppinsW700(
          color: LightColors.backgroundColor,
          fontSize: 11.0,
        ),
      ),
    ),
  ),
);
}

BoxDecoration get getContainerDecoration {
  return BoxDecoration(
    gradient: LinearGradient(
      colors: LightColors.panelGradient,
      begin: Alignment.topLeft,
      end: Alignment.bottomRight,
      stops: [0.4, 0.9],
    ),
  );
}
}

```

Attack_creation_screen.dart

```

import
'package:diplom_proj/presentation/attack_creation/creation_pages/first_creation_page.dart';
import
'package:diplom_proj/presentation/attack_creation/creation_pages/second_creation_page.dart';
import
'package:diplom_proj/presentation/attack_creation/creation_pages/third_creation_page.dart';
import 'package:diplom_proj/presentation/shared/mixins/page_view_mixin.dart';
import 'package:diplom_proj/src/attack/domain/attack_bloc.dart';
import 'package:diplom_proj/src/shared/layouts/main_layout.dart';
import 'package:flutter/material.dart';

class AttackCreation extends StatefulWidget {
  const AttackCreation({Key? key}) : super(key: key);

  @override
  State<AttackCreation> createState() => _AttackCreationState();
}

class _AttackCreationState extends State<AttackCreation> with PagedViewMixin {
  List<Widget> pages(BuildContext context) {
    final treatments = context.read<AttackBloc>().state.treatmentsGroup;
    return [
      FirstCreationPage(
        pickedGroups: treatments,
      ),
      SecondCreationPage(),
      ThirdCreationPage(),
    ];
  }
}

```

```

late final PageController controller;

@override
void initState() {
  super.initState();
  controller = PageController();
}

@override
Widget build(BuildContext context) {
  return MainLayout.withFocusLayout(
    child: Expanded(
      child: Stack(
        children: <Widget>[
          buildPageView(
            pages(context),
            controller,
          ),
          buildCircleIndicator(pages(context).length),
        ],
      ),
    ),
  );
}

```

Weather_api.dart

```

import 'package:dio/dio.dart';
import 'package:diplom_proj/src/attack/entities/forecast_dto/forecast_dto.dart';
import 'package:diplom_proj/src/attack/entities/weather_dto/weather_dto.dart';
import 'package:diplom_proj/src/shared/interfaces/i_base_api.dart';
import 'package:retrofit/http.dart';
part 'weather_api.g.dart';

@RestApi()
abstract class WeatherApi extends IBaseApi {
  factory WeatherApi(Dio dio, {String? baseUrl}) = _WeatherApi;

  @GET('/weather?lat={lat}&lon={lon}&appid={keyAPI}')
  Future<WeatherDTO> getCurrentWeather({
    @Path() required String lon,
    @Path() required String lat,
    @Path() required String keyAPI,
  });

  @GET('/onecall?lat={lat}&lon={lon}&exclude=current,minutely,hourly&appid={keyAPI}')
  Future<ForecastDTO> getForecastWeather({
    @Path() required String lon,
    @Path() required String lat,
    @Path() required String keyAPI,
  });
}

```

Fetch_weather_event.dart

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:diplom_proj/src/shared/interfaces/i_bloc.dart';
import 'package:geolocator/geolocator.dart';
import '../attack_bloc.dart';

class FetchWeatherEvent extends IBlocEvent<AttackState> {
  const FetchWeatherEvent({
    required this.onCompleted,
    required this.permissionDenied,
  });

  final VoidCallback onCompleted;
  final VoidCallback permissionDenied;

  @override
  Stream<AttackState> action(covariant AttackBloc bloc) async* {
    final state = bloc.state;
    yield state.copyWith(isLoading: true);
    LocationPermission permission = await Geolocator.checkPermission();
    if (permission == LocationPermission.denied) {
      permission = await Geolocator.requestPermission();
      if (permission == LocationPermission.denied) {
        permissionDenied.call();
      }
    }

    final Position pos = await Geolocator.getCurrentPosition();
    final weather = await bloc.attackService.getForecastWeather(
      lon: pos.longitude.toString(),
      lat: pos.latitude.toString(),
    );
    final updatedModel = state.currentModel!.copyWith(weather: weather);

    yield state.copyWith(
      isLoading: false,
      currentModel: updatedModel,
    );
    onCompleted();
  }
}
```

CreatePredictionEvent.dart

```
import 'dart:async';

import
'package:diplom_proj/src/predictions/domain/events/fetch_daily_weather_event.dart';
import 'package:diplom_proj/src/predictions/domain/prediction_bloc.dart';
import
'package:diplom_proj/src/predictions/entities/prediction_weather/prediction_weather.dart
';
import
```

```

'package:diplom_proj/src/predictions/entities/short_weather_model/short_weather_model.dart';
import 'package:flutter/material.dart';
import 'package:diplom_proj/src/shared/interfaces/i_bloc.dart';

class CreatePredictionEvent extends IBlocEvent<PredictionState> {
  const CreatePredictionEvent ({
    required this.onCompleted,
    required this.permissionDenied,
  });

  final VoidCallback onCompleted;
  final VoidCallback permissionDenied;

  @override
  Stream<PredictionState> action(covariant PredictionBloc bloc) async* {
    bloc.add(
      FetchDailyWeatherEvent(
        onCompleted: () async* {
          final weatherModel = bloc.state.weather;
          if (weatherModel != null) {
            final previosWeather = bloc.state.currentPredictions.nextWeather;
            RiskPrediction newPrediction = RiskPrediction.noRisk;
            int riskCounter = 0;
            if (760 - weatherModel.pressure > 6) {
              riskCounter++;
            }
            if ((weatherModel.temp.morn - previosWeather.temperature).abs() > 5) {
              riskCounter++;
            }
            if (weatherModel.temp.morn < -10) {
              riskCounter++;
            }
            if (weatherModel.temp.morn > -13.3 && weatherModel.humidity > 67.7) {
              riskCounter++;
            }
            if (riskCounter == 1) {
              newPrediction = RiskPrediction.risk;
            } else if (riskCounter > 1) {
              newPrediction = RiskPrediction.increasedRisk;
            }
            final newPredictionModel = PredictionWeather(
              hasRisk: newPrediction,
              date: DateTime.now(),
              nextWeather: ShortWeatherModel(
                humidity: weatherModel.humidity,
                temperature: weatherModel.temp.morn,
                pressure: weatherModel.pressure,
              ),
              prevWeather: previosWeather,
              cameTrue: null,
            );
            final List<PredictionWeather> list =
            bloc.state.predictionList..add(newPredictionModel);
            bloc.predictionService.postPredictions(list);
            yield bloc.state.copyWith(
              currentPredictions: newPredictionModel,
              predictionList: list,
            );
          }
        },),), ); }}

```

ДОДАТОК В
Слайди презентації

Програмна реалізація мобільного застосунку-трекеру нападів мігрені у людей

Дипломна робота

Виконала:
ст. гр. КНТ-128

М. В. Колесникова

Керівник:
к. т. н. доцент

Т. В. Федорончак

Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

Рисунок В.1 – Слайд 1

Мета роботи

- створення кросплатформного мобільного застосунку-трекеру «MigTracker» для обліку нападів мігрені у людей

Об'єкт дослідження

- Облік нападів мігрені

Предмет дослідження

- Мобільний застосунок-трекер нападів мігрені у людей «MigTracker»

Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

Рисунок В.2 – Слайд 2

Постановка завдання

Завдання дипломної роботи, це програмна реалізація мобільного застосунку-трекера нападу мігрені у людей.

Вимоги до застосунку :

- створення, редагування та зберігання записів про напади мігрені з єдиною структурою;
- візуалізація статистичних даних з кількості нападів, найчастіших тригерів, найбільш застосовуваних ліків;
- відображення записів у вигляді списку;
- відображення записів у календарі;
- конвертація записів до файлу формату PDF;
- сортування записів;
- надання рекомендації користувачу щодо полегшення симптомів;
- можливість передбачення нападів за погодними умовами;
- можливість зміни мови застосунку.

Активация Windows

Чтобы активировать Windows, перейдите в раздел "Параметры".

3

Рисунок В.3 – Слайд 3

Порівняння існуючих аналогів



Рисунок 1 - Застосунок Migraine Buddy

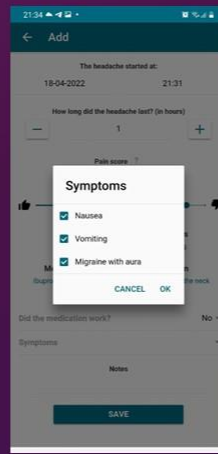


Рисунок 2 - Застосунок Migraine and headache diary



Рисунок 3 - Застосунок Migraine Monitor

4

Рисунок В.4 – Слайд 4

Порівняння існуючих аналогів

Критерій	Migraine Buddy	Migraine and headache diary	Migraine Monitor	MigTracker
Наявність конструктору для створення записів	+	+	+	+
Кросплатформність (Android та IOS)	+	+	+	+
Темна кольорова схема	±	-	-	+
Мультимовність	-	-	-	+
Сортування записів (дата, інтенсивність болю)	-	-	-	+
Конвертація у PDF	-	-	+	+
Представлення записів у вигляді календаря	+	+	+	+
Зрозумілий, не перенасичений інтерфейс	-	+	±	+
Автоматичне зіставлення погодних умов	-	-	+	+
Прогнозування нападу на основі погодних умов	-	-	-	+
Візуалізація статистики	+	-	+	+

5

Рисунок В.5 – Слайд 5

Аналіз існуючих технологій

Порівняння фреймворків

Критерій	Xamarin	React Native	Flutter
Android, IOS розробка	+	+	+
Продуктивність на рівні нативної	-	-	+
Повністю безкоштовний	-	+	+
Засоби для швидкісної розробки (hot reload)	-	+	+
Оптимізація коду та продуктивності	-	+	+
IDE	Visual Studio	Atom	IntelliJ idea, Android Studio, Visual Studio
Мова програмування	C#	Java Script	Dart

6

Рисунок В.6 – Слайд 6

Аналіз існуючих технологій

Вибір мови програмування

Обрано об'єктно-орієнтовану мову програмування Dart, за рядом причин:

- кросплатформна;
- підтримує потоки;
- оптимізована для створення користувацьких інтерфейсів;
- структурована та гнучка;
- висока продуктивність одержуваних програм;
- наявність великої кількості готових інструментів для підтримки розробки мовою Dart;
- підтримка бібліотек, що дозволяють, як використовувати готові рішення, так і створювати окремо власні, та підключати їх до проекту;
- Flutter було розроблено саме для Dart.

7

Рисунок В.7 – Слайд 7

Аналіз існуючих технологій

Порівняння середовищ розробки

Критерій	Android Studio	IntelliJ Idea	Visual Studio Code
Призначений для мобільної розробки	+	-	-
Підтримка системи контролю версій	+	+	+
Пропонує способи виправлення помилок	+	+	-
Безкоштовний	+	-	+
Вбудований відлагоджувач	+	+	-
Вбудований емулятор смартфона	+	-	-
Авто-доповнення коду	+	+	+

8

Рисунок В.8 – Слайд 8

Зберігання даних

Порівняння віддалених сховищ

Критерій	Firestore Database	Realtime Database
Модель даних	Дані зберігаються у вигляді документів, що схожі за структурою на json.	Зберігає дані у дереві json.
Офлайн підтримка	Підтримується на Android, Apple та web клієнтах	Підтримується на Android та Apple клієнтах
Запити	Індексовані запити з можливістю складного сортування та фільтрації	Глибокі запити, що повертають все піддерево, з обмеженою можливістю сортування та фільтрації
Продуктивність та надійність	БД реплікується у різних регіонах, що робить її більш стійкою до збоїв, а також підтримує масштабованість.	Репліки БД зберігаються у одному регіоні. Низька затримка відповіді підходить для програм з частою синхронізацією.
Автоматична масштабованість	+	-
Можливість відстеження клієнта онлайн	-	+

9

Рисунок В.9 – Слайд 9

Архітектура застосунку



Рисунок 4 - Діаграма обраної архітектури

10

Рисунок В.10 – Слайд 10

Функціональна схема розробленої програми

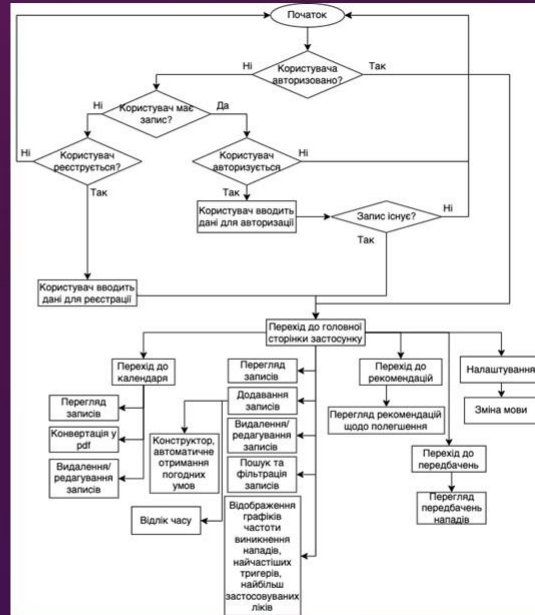


Рисунок 5 - Функціональна схема розробленої програми 11

Рисунок В.11 – Слайд 11

Компонентна схема застосунку

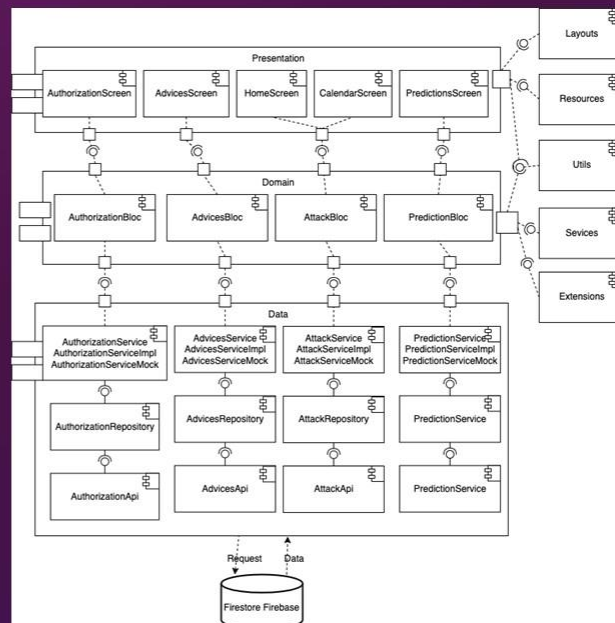


Рисунок 6 – Діаграма компонентів розроблюваної системи 12

Рисунок В.12 – Слайд 12

Зберігання даних

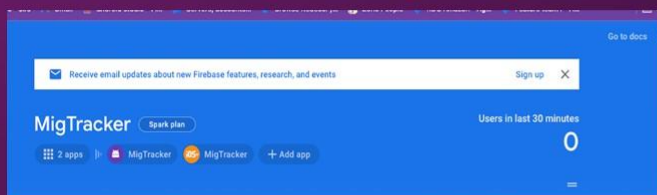


Рисунок 7 - Головна сторінка проекту MigTracker

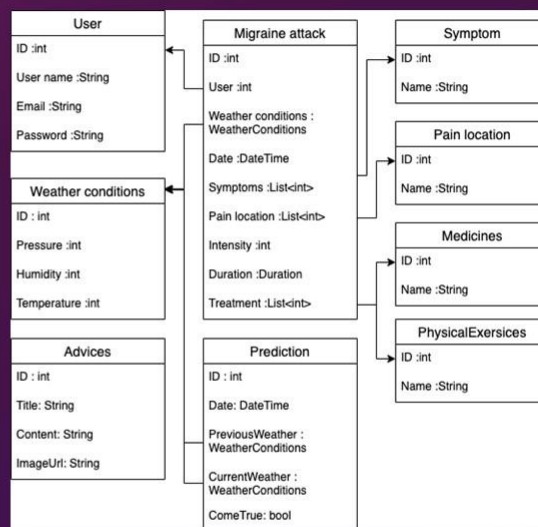


Рисунок 8 - Схема зв'язків моделей даних у застосунку

13

Рисунок В.13 – Слайд 13

Приклад інтерфейсу застосунку

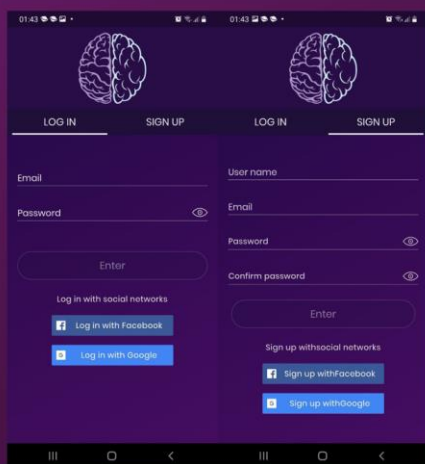


Рисунок 9 – Авторизаційна та реєстраційна сторінки застосунку

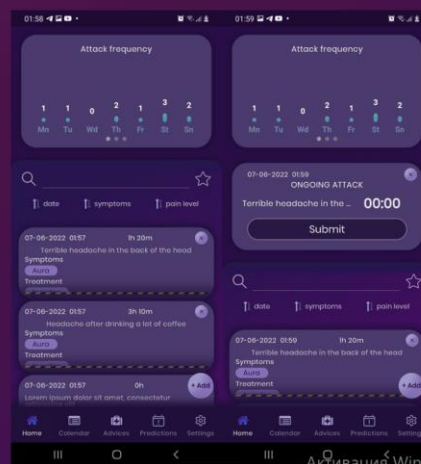


Рисунок 10 – Домашня сторінка застосунку

14

Рисунок В.14 – Слайд 14

Приклад інтерфейсу застосунку

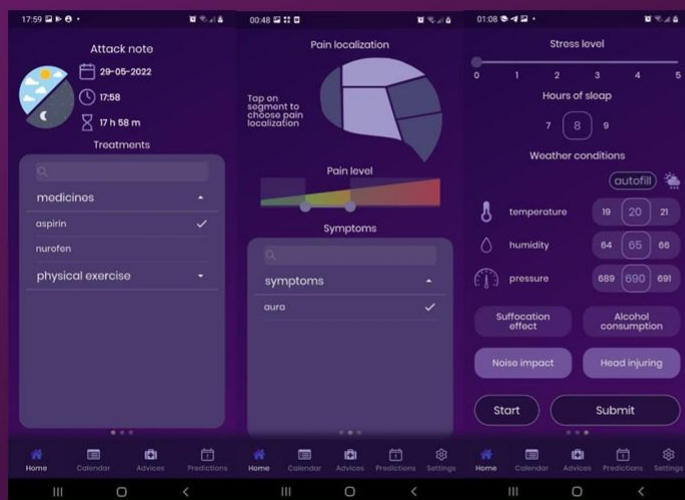


Рисунок 11 – Сторінки конструктору нападу



Рисунок 12 – Сторінка календаря

15

Рисунок В.15 – Слайд 15

Приклад інтерфейсу застосунку



Рисунок 13 – Сторінка порад



Рисунок 14 – Сторінка передбачень



Рисунок 15 – Сторінка налаштувань

16

Рисунок В.16 – Слайд 16

Висновки

У результаті дипломної роботи:

- Було розглянуто проблемну область застосунку, проаналізовано існуючі аналоги програм;
- Визначено вимоги до застосунку, проаналізовано існуючі технології, обрано архітектуру;
- Розроблено та протестовано програмний продукт “MigTracker”, що відповідає заявленим до нього вимогам, та стане зручним засобом вести облік нападів.

17

Рисунок В.17 – Слайд 17

Дякую за увагу!

18

Рисунок В.18– Слайд 18