



Міністерство освіти і науки України  
Державний університет «Житомирська політехніка»  
Факультет інформаційно-комп'ютерних технологій  
Кафедра інженерії програмного забезпечення

**ПОЯСНЮВАЛЬНА ЗАПИСКА**  
до кваліфікаційної магістерської роботи  
на тему: «Мобільний додаток «Щоденник  
самоконтролю цукрового діабету»»

Виконав студент 2-го курсу, групи ПІ-50м  
спеціальності 121 «Інженерія програмного  
забезпечення»

 Д.С. Моргунов  
Керівник, старший викладач кафедри КН

 В.Л. Левківський


Рецензент, асистент кафедри ІПЗ  
 І.А. Толстой

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»  
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ  
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

СПЕЦІАЛЬНІСТЬ 121 «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри інженерії програмного  
забезпечення

к.т.н., доцент  І.В. Пулеко  
« 04 » 09 20 19 р.

**ЗАВДАННЯ**

на кваліфікаційну магістерську роботу

Студента

Моргунова Дениса Сергійовича

Тема роботи:

Мобільний додаток «Щоденник самоконтролю цукрового діабету»»






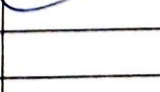
Затверджена Наказом університету від «04» вересня 2019 р. № 206с

Термін здачі студентом закінченої роботи 01.12.2019

Вихідні дані роботи (зазначається предмет і об'єкт дослідження)

Об'єктом дослідження є процес компенсації цукрового діабету. Предметом дослідження є використання інформаційних технологій для спрощення самоконтролю хворих цукровим діабетом.

Консультанти з кваліфікаційної роботи магістра із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Левківський В.Л.	 05.09.19р.	 30.09.19р.
2	Левківський В.Л.	 30.09.19р.	 08.11.19р.
3	Левківський В.Л.	 01.11.19р.	 15.11.19р.

Керівник

  
(підпис)

### Календарний план

№ з/п	Назва етапів кваліфікаційної магістерської роботи	Термін виконання етапів роботи	Примітка
1	Дослідження теми	5 вересня 2019 – 15 вересня 2019	Виконано
2	Постановка задачі	16 вересня 2019 – 21 вересня 2019	Виконано
3	Пошук, огляд та аналіз аналогічних розробок	22 вересня 2019 – 26 вересня 2019	Виконано
4	Формулювання технічного завдання	27 вересня 2019 – 30 вересня 2019	Виконано
5	Опрацювання літературних джерел	1 жовтня 2019 – 10 жовтня 2019	Виконано
6	Проектування структури	11 жовтня 2019 – 15 жовтня 2019	Виконано
7	Написання програмного коду, розробка бази даних	16 жовтня 2019 – 8 листопада 2019	Виконано
8	Тестування	9 листопада 2019 – 15 листопада 2019	Виконано
9	Написання пояснювальної записки	16 листопада 2019 – 30 листопада 2019	Виконано

Студент \_\_\_\_\_  
(підпис)

Керівник \_\_\_\_\_  
(підпис)



## РЕФЕРАТ

Кваліфікаційна магістерська робота складається з мобільного додатку «Щоденник самоконтролю цукрового діабету» та пояснювальної записки. Пояснювальна записка містить 76 сторінок, 19 ілюстрацій та 4 таблиць.

Метою роботи є реалізація мобільного додатку, що містить модулі для ведення щоденнику самоконтролю, роботи з записами, синхронізацією даних, модулі для створення аналітики.

В пояснювальній записці розглянуто всі етапи розробки додатку. В випускній роботі проаналізовано аналогічні мобільні додатки, визначено основні завдання для реалізації системи. Визначено та обґрунтовано архітектуру майбутнього додатку. Також в записці описано структура програмного додатку, проаналізовано та обґрунтовано вибір бази даних та стаку технологій, на базі яких буде реалізовано додаток. Детально розглянуто структуру бази даних, описано алгоритми взаємодії деяких класів системи, продемонстровано об'єктну структуру додатку, алгоритми роботи модулів.

**КЛЮЧОВІ СЛОВА:** ЦУКРОВИЙ ДІАБЕТ, МОБІЛЬНИЙ ДОДАТОК, FLUTTER

## ABSTRACT

The thesis consists of a mobile application for self-control of diabetes and an explanatory note. The explanatory note to the graduation thesis contains 76 pages, 19 illustrations and 4 tables.

The purpose of the work is the implementation of a mobile application containing modules for diary self-control, work with records, data synchronization, modules for creating analytics.

The explanatory note covers all stages of application development. In the final work similar mobile applications are analyzed, the main tasks for implementation of the system are defined. The architecture of the future application has been identified and substantiated. The note also describes the structure of the software application, analyzed and substantiated the choice of database and technology stack on which the application will be implemented. The structure of the database is described in detail, the algorithms of interaction of some classes of the system are described, the object structure of the application, the algorithms of module operation are demonstrated.

**Keywords:** SUGAR DIABETES, MOBILE APPLICATION, FLUTTER

## ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП .....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ДОДАТКУ «ЩОДЕННИК САМОКОНТРОЛЮ ЦУКРОВОГО ДІАБЕТУ».....	10
1.1 Актуальність теми дослідження .....	10
1.2 Аналіз методів збору вхідних даних про стан пацієнта .....	14
1.3 Аналіз та дослідження аналогів програмного продукту .....	18
1.4 Постановка задачі.....	22
Висновки до першого розділу.....	24
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ «ЩОДЕННИК САМОКОНТРОЛЮ ЦУКРОВОГО ДІАБЕТУ».....	25
2.1. Визначення варіантів використання та вимог до додатка .....	25
2.2 Архітектура та узагальнена структуру, програмного комплексу. ....	28
2.3 Обґрунтування вибору інструментальних засобів .....	32
2.4. Розробка бази даних системи.....	38
2.5 Реалізація функціональних частин додатку.....	40
Висновки до другого розділу .....	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МОБІЛЬНИМ ДОДАТКОМ «ЩОДЕННИК САМОКОНТРОЛЮ ЦУКРОВОГО ДІАБЕТУ».....	49
3.1. Структура інтерфейсу. Інтерфейс та порядок роботи з додатком .....	49
3.2. Тестування роботи мобільного додатку .....	55
3.3 Аналіз та моделювання даних на базі записів додатку .....	58
Висновки до третього розділу.....	60
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	63
ДОДАТКИ.....	66

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD – create, read, update, delete (з англ. створити, прочитати, оновити, видалити);

ГК – глюкоза в крові;

БД – база даних;

ЦД – цукровий діабет;

UX – User Experience (з англ. користувацький досвід);

СКБД – система керування базами даних;

## ВСТУП

**Актуальність теми.** Темою кваліфікаційної магістерської роботи є розробка мобільного додатку «Щоденник самоконтролю цукрового діабету». Використання сучасних інформаційних технологій створює нові можливості для оперативного отримання інформації про стан і тенденції перебігу такої хвороби, як цукровий діабет. Наприклад, такий напрям як телемедицина, коли пацієнт может через інтернет провести консультацію з лікарем. Також люди з цукровим діабетом можуть покращити свої показники компенсації за рахунок ведення щоденнику самоконтролю, і технології, які можуть зберігати та системазувати ці дані, дозволять своєчасно виробити і прийняти необхідні дії для покращення перебігу хвороби.

Актуальність обраної теми полягає в тому, що все більше людей в світі хворіють на цукровий діабет, а розроблюваний додаток значною мірою може покращити рівень життя людей з даним захворюванням.

**Метою випускної роботи** є проектування архітектури, розробка алгоритмів роботи та реалізація мобільного додатку по самоконтролю цукрового діабету

**Завдання роботи.** Встановлена мета обумовлює наступні завдання:

- проведення аналізу функцій додатків для самоконтролю;
- визначення архітектури та узагальненої структури системи;
- обґрунтування та вибір засобів та інструментів реалізації додатку;
- проектування структурних складових та алгоритмів роботи системи;
- реалізація додатку

**Об'єктом дослідження** є процес компенсації цукрового діабету.

**Предметом дослідження** є використання інформаційних технології для спрощення самоконтролю хворих цукровим діабетом.

**Практична значимість роботи.** Розробка програмного продукту який може значною мірою покращити процес компенсації цукрового діабету.



**Методи досліджень.** В роботі було використано методи теорії систем і методи об'єктно орієнтованого проектування та програмування.

**Апробації та публікації.**

1. Моргунов Д.С., Левківський В.Л. Аналіз методів та метрик для моделювання системи автоматизованого контролю цукрового діабету. Тези доповідей учасників X Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології – 2019» - Житомир: ЖДТУ, 18-20 квітня 2019 р. - с.55-56.
2. Моргунов Д.С. Київський національний університет імені Тараса Шевченка, II тур Всеукраїнського конкурсу студентських наукових робіт з інженерії програмного забезпечення, отримано диплом другого ступеню з роботою "Інсулін-глюкоза"

## РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ДОДАТКУ «ЩОДЕННИК САМОКОНТРОЛЮ ЦУКРОВОГО ДІАБЕТУ»

### 1.1 Актуальність теми дослідження

Актуальність проблеми цукрового діабету зумовлена значною поширеністю захворювання, тим що він є базою для розвитку складних супутніх захворювань та ускладнень, ранньої інвалідності та смертності. Основу їх складають діабетичні мікроангіопатії та нейропатії. У хворих на діабет значний ризик атеросклерозу та ішемічної хвороби серця. Більше 40% ампутацій нижніх кінцівок є наслідком синдрому діабетичної стопи. Цукровий діабет також найчастіша причина сліпоты у людей.

Все вищезазначене призводить до значних матеріальних витрат, спрямованих на лікування цукрового діабету та його ускладнень.

Цукровий діабет — це захворювання обміну речовин, в основі якого лежить хронічна гіперглікемія. Поширеність цукрового діабету складає близько 4—5% у розвинених країнах світу. Відомо, що у осіб після 65 років захворюваність на діабет складає близько 10—15%. У 2000 році кількість хворих на діабет становила 175 млн., згідно з прогнозом міжнародного інституту діабету на 2010 рік — 240 млн., а на 2030 рік ця цифра складе 300 млн. чол.

Відповідно до патогенезу діабет є захворюванням гетерогенним, але в клініці чітко виділяють два характерних типи: I (інсулінозалежний) та II (інсулінозалежний). Цукровий діабет I типу є аутоімунним захворюванням, пускову роль в якому відіграє вірусна інфекція. Віруси викликають деструкцію інсуліно-продукуючих клітин підшлункової залози, активують продукцію прозапальних цитокінів. На цій основі формується абсолютна інсулінова недостатність.

Цукровий діабет II типу — гетерогенне захворювання, в основі якого лежить інсулінорезистентність та недостатність функції бета-клітин.

Інсулінорезистентність у переважної більшості хворих генетично обумовлена. порушення інсуліносекреції проявляється втратою раннього піку секреції інсуліну, базальної секреції, амплітуди пульсових імпульсів. Певну роль в цьому відіграють білки-транспортери глюкози, особливо ГЛЮТ-4.

Діагноз цукрового діабету встановлюється завдяки клінічним показникам і підтверджується лабораторними даними. Враховуючи велику поширеність захворювання, важливість його раннього виявлення, використовують апробовані, прості і доступні методи визначення рівня глікемії і прирівнюють їх до відповідних міжнародних критеріїв. Вони є своєрідним «золотим стандартом» у діагностиці цукрового діабету. Відомо, що нормальний рівень глікемії у здорової людини коливається у межах 3,3—5,5 ммоль/л (80—120 мг) натщесерце, а впродовж доби від 4 до 8—9 ммоль/л.

Згідно з останніми критеріями ВООЗ (1999) діагноз цукрового діабету встановлюють, якщо рівень глікемії натщесерце 6,1 ммоль/л і більше або 11,1 ммоль/л і більше випадково серед доби. Для підтвердження діагнозу аналізу необхідно провести 2—3 рази в інші дні [7].

Другим варіантом гіперглікемії є порушена толерантність до глюкози (ТТГ). Вона визначається за наявності сумнівних, невизначених результатів під час діагностики цукрового діабету. Показанням для проведення ТТГ є: наявність цукрового діабету в родині або в анамнезі жінок, що народили мертвих дітей чи дітей з масою 4,5 кг, надмірна вага, артеріальна гіпертензія, глюкозурія вагітних, патологічна вагітність і пологи, випадкова гіперглікемія після приймання їжі, реактивна гіпоглікемія, хронічні інфекції та ускладнення, характерні для цукрового діабету. Порушеною толерантністю до глюкози згідно з критеріями ВООЗ вважається наявність глікемії натщесерце до 6,1 ммоль/л та через 2 години після навантаження до 7,8 ммоль/л. Ця категорія гіперглікемії вводиться для

більш диференційованої оцінки доклінічних порушень вуглеводного обміну і ранньої первинної профілактики цукрового діабету [6].

З метою визначення рівня глікемії використовують глюкозооксидазний метод або портативні глюкометри, які працюють на методі "сухої" хімії за допомогою тест-смужок ("Глюкофот", "Глюкокард", "Ван Тач" та інші).

Певне діагностичне значення має визначення глюкозурії. Її наявність є приводом для визначення рівня глікемії або проведення ТТГ.

З метою ранньої діагностики цукрового діабету та оцінки якості лікування використовується визначення рівня глікованих протеїнів. Вони представляють собою білки з глюкозою, приєднаною неферментативним шляхом. Серед них найбільше клінічне значення мають глікований гемоглобін та фруктозамін. Рівень глікованого гемоглобіну в нормі складає 4–7%, для діабету є характерним його підвищення. Ступінь глікування гемоглобіну залежить від компенсації вуглеводного обміну, є ранньою ознакою порушення обміну вуглеводів і свідчить про ступінь компенсації діабету протягом останніх 90 діб.

Важливе діагностичне значення має визначення рівня фруктозаміну, продукту глікування деяких білків плазми і, в першу чергу, альбуміну. Рівень фруктозаміну у здорових людей складає до 0,285 ммоль/л, при цукровому діабеті він значно перевищує цей показник. Період напівжиття фруктозаміну складає 14 діб, тому його рівень дає можливість оцінити порушення вуглеводного обміну за цей період.

Діагностичне значення за цукрового діабету має також визначення антитіл до компонентів інсулінопродукуючих клітин, рівня цитокінів, показників ліпідного обміну, інсуліну та С-пептиду.

Центральне місце в лікуванні цукрового діабету займає компенсація обміну речовин, яка передбачає досягнення нормоглікемії, аглюкозурії, нормальних показників глікованого гемоглобіну, ліпідів, білків, мінералів, артеріального тиску. Насправді ситуація з компенсацією цукрового діабету на сьогодні є

незадовільною. Зокрема, лише 16% хворих на цукровий діабет у Європі знаходяться у стані компенсації, у 43% спостерігається нормоліпідемія та у 41% — нормальні показники артеріального тиску.

Згідно з дослідженням UKPDS, проведеним у Великій Британії, на 5102 хворих цукровим діабетом (1998), яке проводилось протягом 20 років, при зниженні рівня глікованого гемоглобіну з 7,9% до 7,0% досягнуто зниження усіх ускладнень діабету на 12%, смертності, пов'язаної з діабетом, — на 25%, загальної смертності — на 17%, інфаркту міокарда — на 16%, інсульту — на 15%, мікроангіопатії, враховуючи ретинопатію та нефропатію, — на 25%.

Контроль за артеріальним тиском, проведений у рамках вищезазначеного дослідження свідчить, що зниження його рівня з 154/87 мм рт. ст. до 144/82 мм рт. ст. дає змогу знизити загальну смертність на 32%, ускладнення цукрового діабету — на 24%, захворюваність на інфаркт міокарда — на 21%, інсульт — на 44%, серцеву недостатність — на 56%, мікроангіопатію — на 37%.

Багатоцентрове дослідження, а також досвід практичної медицини свідчить, що компенсація цукрового діабету є основним принципом його лікування, профілактики гострих та хронічних ускладнень, забезпечення нормального розвитку та росту дітей, збереження і відновлення працездатності, нормалізації маси тіла, забезпечення достатньої соціальної адаптації та якості життя хворих.

Основою лікування цукрового діабету I і II є дієтотерапія. Вона має бути фізіологічною, адекватною до енерговитрат, розрахованою на "ідеальну" масу тіла з урахуванням статі, віку, професії. Передбачається обмеження рафінованих вуглеводів, стабільний режим фізичної активності та харчування, достатнє вживання харчових волокон, обмеження жирів тваринного походження.

Особливістю дієтотерапії хворих на цукровий діабет I типу є постійність добової енергетичної цінності, співвідношення основних компонентів їжі. Дієта при діабеті II типу з надмірною масою тіла має бути низькокалорійною, особливо за рахунок жирів тваринного походження [7].

На практиці для зручності розрахунку дієти використовують систему хлібних одиниць (ХО). Відомо, що одна хлібна одиниця дорівнює 12 г вуглеводів і еквівалентна 20 г білого хліба. У відповідності з добовою енергетичною потребою за умови легкої праці рекомендовано 12 ХО, середньої — 17 ХО, тяжкої — 23—27 ХО, а людям з наявністю ожиріння 6—9 ХО.

Важливим компонентом терапії цукрового діабету є дозоване фізичне навантаження. Воно сприяє засвоєнню глюкози, поліпшує кровообіг, активує схуднення, знижує дозу цукрознижувальної терапії. Фізичні навантаження мають бути під лабораторним контролем, а також коригуватись дозою цукрознижувальних засобів, приємними для хворого і не обтяжливими для їх виконання.

Найпотужнішим, ефективним і надійним методом лікування цукрового діабету є інсулінотерапія. Інсулін має цукрознижувальну, анаболічну та антикатаболічну дію. Його застосовують для лікування хворих на цукровий діабет I типу, при кетоацидозі, гіперкетонемічній комі, значній втраті маси тіла, наявності інфекційних та інτερкурентних захворювань, оперативних втручаннях, вагітності та лактації [6].

Особливої уваги заслуговує застосування інсуліну в терапії хворих на цукровий діабет II типу. Необхідно пам'ятати, що при цьому типі діабету за відсутності ефекту від пероральної цукрознижувальної терапії призначення інсуліну обов'язкове. Основною у терапії різних типів діабету є повна компенсація обміну відповідно до вищенаведених критеріїв. Від цього залежить якість життя хворого і прогноз захворювання.

## 1.2 Аналіз методів збору вхідних даних про стан пацієнта

Для успішної розробки мобільного додатку «Щоденник самоконтролю цукрового діабету», необхідно розглянути та реалізувати методи для збору вхідних даних про стан пацієнта.



Ці дані можна розділити на дві групи: потокові — які потрібно постійно вводити в режимі потоку; сталі — задаються один раз при налаштуванні системи.

До сталих даних можна віднести такі показники, як:

- Вік — цей показник є одним з ключових при моделюванні динаміки ГК. З віком швидкість метаболізму зменшується, що в свою чергу призводить до збільшення часу засвоєння вуглеводів.;

- Стать — різниця в гормонах може впливати на підрахунок динаміки ГК;

- Тип діабету — впливає на підрахунок необхідної дози ліків;

Потокові дані включають в себе:

- ГК — являє собою один з головних показників, від якого залежить прорахунок необхідної дози ліків;

- Алкоголь — має безпосередній вплив на ГК;

- Вуглеводи (розділені по рівням ГІ) — являє собою один з головних показників, від якого залежить прорахунок необхідної дози ліків. При цьому повинен враховуватися — Глікемічний індекс (ГІ) — це показник, який відображає, з якою швидкістю той чи інший харчовий продукт розщеплюється в організмі людини і перетворюється на глюкозу — головне джерело енергії. Чим швидше розщеплюється продукт, тим вищий його глікемічний індекс. За еталон була прийнята глюкоза, глікемічний індекс якої дорівнює 100. Всі інші показники порівнюються з глікемічним індексом глюкози.;

- Вага — від даного показника залежить метаболізм пацієнта;

- Інсулін — являє собою один з головних показників, який використовується як головний інструмент при моделюванні системи автоматичного управління ЦД. Так як інсулін з різних областей тіла має різну швидкість всмоктування, необхідно дотримуватися правил проведення ін'єкцій. Якщо інсулін вводиться в м'язи - швидкість всмоктування буде швидше, що може

привести до більш низьких показників цукру крові, ніж при звичайному підшкірному введенні.;

- Час та дата – від даного показника залежить коефіцієнт чутливості до ліків;

- Місце введення ліків - від даного показника залежить швидкість утилізації ліків;

- Фізична активність – даний показник впливає на коефіцієнт чутливості до ліків;

- Стрес та хвороби - даний показник може знижувати чутливість до ліків;

Далі потрібно розглянути методи отримання приведених вище даних. Деякі з цих даних можна отримати лише шляхом їх ручного вводу пацієнтом:

- Вуглеводи;
- Алкоголь;
- Інсулін;
- Місце введення ліків;
- Стрес та хвороби;

Також є дані які можна отримати іншими шляхами, окрім ручного вводу: ГК, вага, фізична активність, дата та час. Розглянемо способи отримання цих показників.

### *Глюкоза в крові*

Інформацію про ГК можна отримати двома шляхами:

- 1) Глюкометр – після вимірювання ГК за допомогою глюкометра, пацієнт може або ввести дані власноруч, або синхронізувати дані з системою використовуючи технологію Bluetooth (якщо її підтримує глюкометр);

- 2) Системи безперервного моніторингу глюкози - проводить вимірювання цукру крові кожні 10 секунд і робить запис середнього значення кожні 5 хвилин. Результати вимірювань після комп'ютерної обробки можуть бути

представлені як у вигляді цифрових даних (288 вимірювань в добу із зазначенням часу, меж коливання і середніх значень рівня глікемії, а також рівня глікемії за день і за три доби), так і у вигляді графіків, на яких відзначені коливання рівня глікемії за час дослідження.

Даний метод дослідження глікемічного профілю у хворих на цукровий діабет дає можливість лікарю уникнути систематичних помилок в призначенні цукрознижуючої терапії, зокрема безперервних нічних гіпоглікемій, підібрати оптимальні індивідуальні дози цукрознижуючих препаратів, що дає можливість підібрати оптимальний режим терапії для досягнення компенсації цукрового діабету.

Добовий моніторинг глікемії дозволяє виявити приховані коливання рівня глюкози крові, які не можуть бути виявлені простим визначенням рівня глюкози протягом дня або вимірюванням глікозильованого гемоглобіну (HbA1c).

#### *Вага*

Після вимірювання ваги, пацієнт може або ввести дані власноруч, або синхронізувати дані з системою використовуючи технологію Bluetooth (якщо її підтримують ваги);

#### *Фізична активність*

Інформація даного типу має різний характер збору, один з них, синхронізація даних з побічного пристрою, такого як фітнес трекер - це гаджет, який - в більшості випадків - одягається на руку і має вбудовані датчики, які відслідковують активність протягом дня, включаючи: кількість кроків, пульс, сон, спалені калорії і т.д.; аналогом фітнес трекеру являється збір використовуючи додаток встановлений на смартфон користувача, або користувач може просто ввести дані про денну активність власноруч.

#### *Дата та час*

У випадку введення даних власноруч, час та дата вносяться таким самим чином. Якщо дані надходять від інших пристроїв або додатків, то вони вже несуть в собі інформацію про час та дату.

### 1.3 Аналіз та дослідження аналогів програмного продукту

Щоденник самоконтролю цукрового діабету дозволяє забезпечити оперативне отримання інформації про стан і тенденції перебігу хвороби; своєчасно виробити і прийняти необхідні дії для покращення перебігу хвороби. На сучасному ринку програмних засобів представлено велике різноманіття програм, що вирішують дану задачу. Для розгляду аналогів, за критеріями популярності та позитивного рейтингу були обрані додатки наведені в табл. 1.1.

Опишемо кожен із розглянутих програмних систем.



Рис.1.1 – Головний екран додатку Diabetes

«Diabetes». Перевагами цієї програми є зручний та зрозумілий інтерфейс додатку, синхронізація між декількома пристроями, експорт даних з додатку у вигляді таблиці, додаток доступний як на платформі Android так і на iOS. Недоліками програми є відсутність аналітики, введення тільки показників глюкози.

«Дневник сахара». Перевагами цього додатку є можливість додати свої ліки, система тегів, створення нагадування, експорт даних. До недоліків можна віднести застарілий та заплутаний користувацький інтерфейс, присутність реклами у додатку, повільна робота додатку, статистика надає надто мало інформації.



Рис.1.2 Головний екран

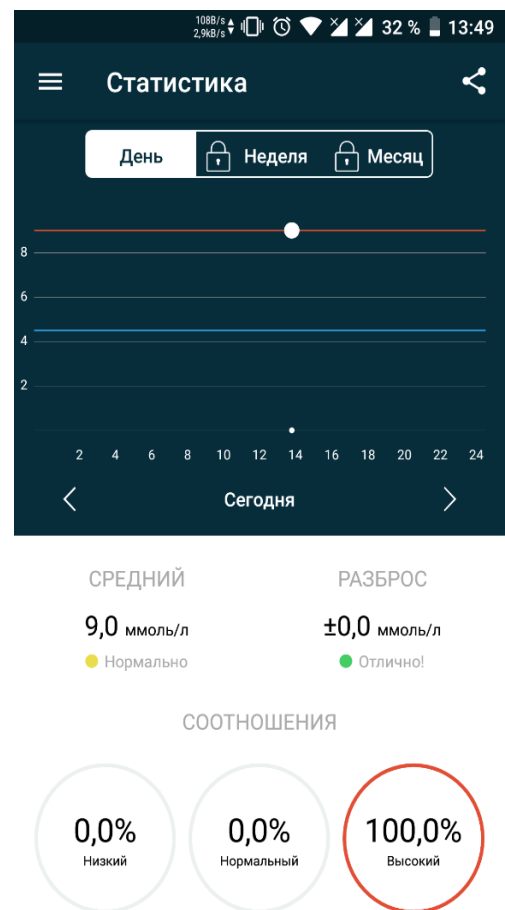


Рис.1.3 Экран статистики додатку  
«DiaMeter»

«DiaMeter». Перевагами є гарний та сучасний інтерфейс додатку, синхронізація даних, вбудований довідник «Школа Діабету» з партнерством провідної компанії Sanofi, статистика та можливість встановити цілі, додаток доступний як на платформі Android так і на iOS. Недоліками додатку є вбудована реклама, можливості додатку є лімітованими в безкоштовній версії, а платна версія коштує дорого, деякі елементи інтерфейсу не є зручними з точки зору UX.

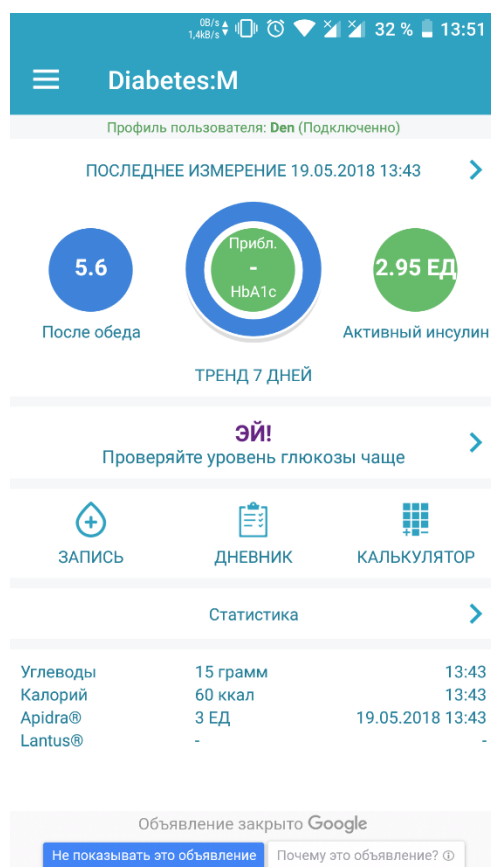


Рис. 1.4 – Головний екран додатку «Diabetes:M»

«Diabetes:M». Даний додаток є найбільш передовим серед аналогів. Переваги: розширена статистика (графіки, діаграми), широкий вибір ліків, нагадування, синхронізація даних, функція знаходження шаблонів в щоденнику та створення на їх основі рекомендацій. Недоліки: вбудована реклама, складний та застарілий інтерфейс, більшість функцій є платними.



Таблиця 1.1

Перелік аналогів мобільного додатку  
«Щоденник самоконтролю цукрового діабету»

Назва	Розробник	Платформа	Умови ліцензії
Diabetes	MedM Inc.	Android, iOS	Безкоштовний
Дневник сахара	mEl Studio	Android	Умовно безкоштовний, повна версія 80 грн/міс
DiaMeter	MeteorIT	Android, iOS	Умовно безкоштовний, повна версія 90 грн/міс
Diabetes:M	Sirma Medical System	Android, iOS	Умовно безкоштовний, повна версія 140 грн/міс

Розглянемо функціональні можливості обраних аналогів.

Таблиця 1.2.

Функціональні можливості

Додаток	Статистика	Синхронізація	Калькулятор	Експорт	Теги
Diabetes	Ні	Так	Ні	Так	Ні
Дневник Сахара	Так	Ні	Ні	Ні	Так
DiaMeter	Так	Так	Ні	Ні	Ні
Diabetes:M	Так	Так	Так	Так	Ні

Переглянувши список аналогів розроблюваного додатку можна зробити висновки, що майже всі програми мають стандартний набір функцій таких, як: робота з записами в щоденнику (створення, видалення, редагування записів), синхронізація даних між пристроями, створення статистики на основі введених

записів в щоденнику, вибір ліків, перегляд статистики використання ліків в реальному часі. В розібраних додатках головним недоліком є те що вони обмежують користувачів в функціоналі, доступ до якого є платним та показують рекламу, а ті додатки що повністю безкоштовні мають бідний функціонал.

Таким чином, основним рисами розроблюваного додатку мають бути всі вище перераховані функції та він має бути повністю безкоштовним без реклами, не обмежуючи користувача в доступних функціях, як це зроблено в деяких аналогах. Також головним нововведенням додатку буде можливість вести спілкування зі своїм лікарем з приводу компенсації хвороби, задля покращення показників.

#### 1.4 Постановка задачі

Користувач додатку матиме можливість вести щоденник самоконтролю, а саме вносити та редагувати дані про фізичну активність, ліки та їжу; переглядати аналітику щодо перебігу хвороби, створену на основі введених даних та синхронізувати дані з сервером додатку. Люди зможуть більш ефективно вести лікування, тим самим покращити перебіг хвороби. Вести спілкування зі своїм лікарем з приводу компенсації хвороби.

Для реалізації поставленого завдання, основними етапами є:

##### 1. Розробити дизайн мобільного додатку.

1.1. Спроекувати загальну концепцію дизайну для головної та внутрішніх сторінок додатку: головна сторінка з основною інформацією про перебіг хвороби; сторінка з аналітикою, для оцінки компенсації за обраний період часу; сторінка з усіма записами в щоденник; сторінка з налаштуваннями (персональні налаштування, ліки, налаштування рівнів глюкози) тощо.

##### 1.2. Створити шаблони екранів додатку.

##### 2. Реалізувати ядро системи управління контентом.

## 2.1. Написати програмні модулі для:

- авторизації користувачів в системі;
- ведення щоденнику самоконтролю (додавання, редагування та видалення записів);
- можливість перегляду аналітики з компенсації хвороби за обраний період часу;
- модуль налаштувань в профілі користувача;
- модуль для відображення графіків активності ліків;
- модуль для відображення основної інформації про перебіг хвороби;

## 2.2. Реалізувати CRUD операції для всіх колекції в базі даних.

## 3. Реалізувати модулі генерації аналітики та статистики:

- 3.1. Аналітичні дані за обраний користувачем період.
- 3.2. Статистичні дані по використанню ліків в реальному часі.
- 3.3. Статистичні дані по рівням цукру в крові в реальному часі.

## 4. Реалізувати структуру збереження даних наступного виду:

- 4.1. Дані про записи користувачів (дата, час, ліки, активність, їжа)
- 4.2. Дані про налаштування користувачів (персональні, рівні глюкози, ліки)
- 4.3. Дані про користувачів (електронна пошта та пароль)

## 5. Створити демонстраційний та тестовий контент для додатку.

Результатом реалізації поставленого завдання є мобільний додаток для ведення щоденнику самоконтролю цукрового діабету, що містить в своєму складі серверну структуру збереження даних, багатокористувацький клієнтський додаток для реалізації функціональних можливостей.

## Висновки до першого розділу

При виконанні даного розділу кваліфікаційної роботи було проаналізовано поставлене завдання та визначено схему подальшої розробки програмного забезпечення. Проаналізовано існуюче програмне забезпечення, а саме щоденники самоконтролю цукрового діабету на мобільних пристроях.

Було визначено основний функціонал розроблюваного додатку та засоби виконання поставлених задач. Розроблюваний додаток буде мати такі функції як: синхронізація, додавання записів в щоденник, вибір ліків, отримання статистики за обраний період, графік активності ліків. Також додаток має працювати як на мобільній системі Android, так і на iOS.

## РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ «ЩОДЕННИК САМОКОНТРОЛЮ ЦУКРОВОГО ДІАБЕТУ»

### 2.1. Визначення варіантів використання та вимог до додатка

Мобільний додаток «Щоденник самоконтролю цукрового діабету» створюється з метою спрощення контролю за перебігом хвороби людям з цукровим діабетом. Користувач додатку буде мати можливість вести щоденник самоконтролю, а саме вносити та редагувати дані про фізичну активність, ліки та їжу; переглядати аналітику щодо перебігу хвороби, створену на основі введених даних та синхронізувати дані з сервером додатку.

#### **Вимоги користувачів**

Зовнішні користувачі мають доступ до наступних функцій:

1. Внесення та редагування даних про ліки, фізичну активність та їжу;
2. Отримання аналітики, створеної на основі введених користувачем даних в щоденник;
3. Синхронізація даних між пристроями;
4. Реєстрація в додатку;
5. Отримання інформації про активність ліків;
6. Можливість проведення консультацій з лікарем

Внутрішні користувачі – адміністратори мають таких функцій, як:

1. Редагування БД;
2. Зміна дизайну інтерфейсу додатку;

#### **Характеристика об'єкта комп'ютеризації:**

##### **Функціональні вимоги**

1. **Аутентифікація користувачів в системі:** В системі має бути представлена можливість реєстрації користувача;
2. **Можливість збереження інформації:** Система повинна зберігати інформацію і надавати можливість користувачу керувати нею;

3. **Ведення журналу самоконтролю:** Внесення та редагування даних про фізичну активність, ліки та їжу;
4. **Перегляд аналітики:** Система повинна надавати можливість перегляду аналітики та статистики;
5. **Консультація з лікарем:** Система повинна забезпечити можливість проведення консультації пацієнта з лікарями.

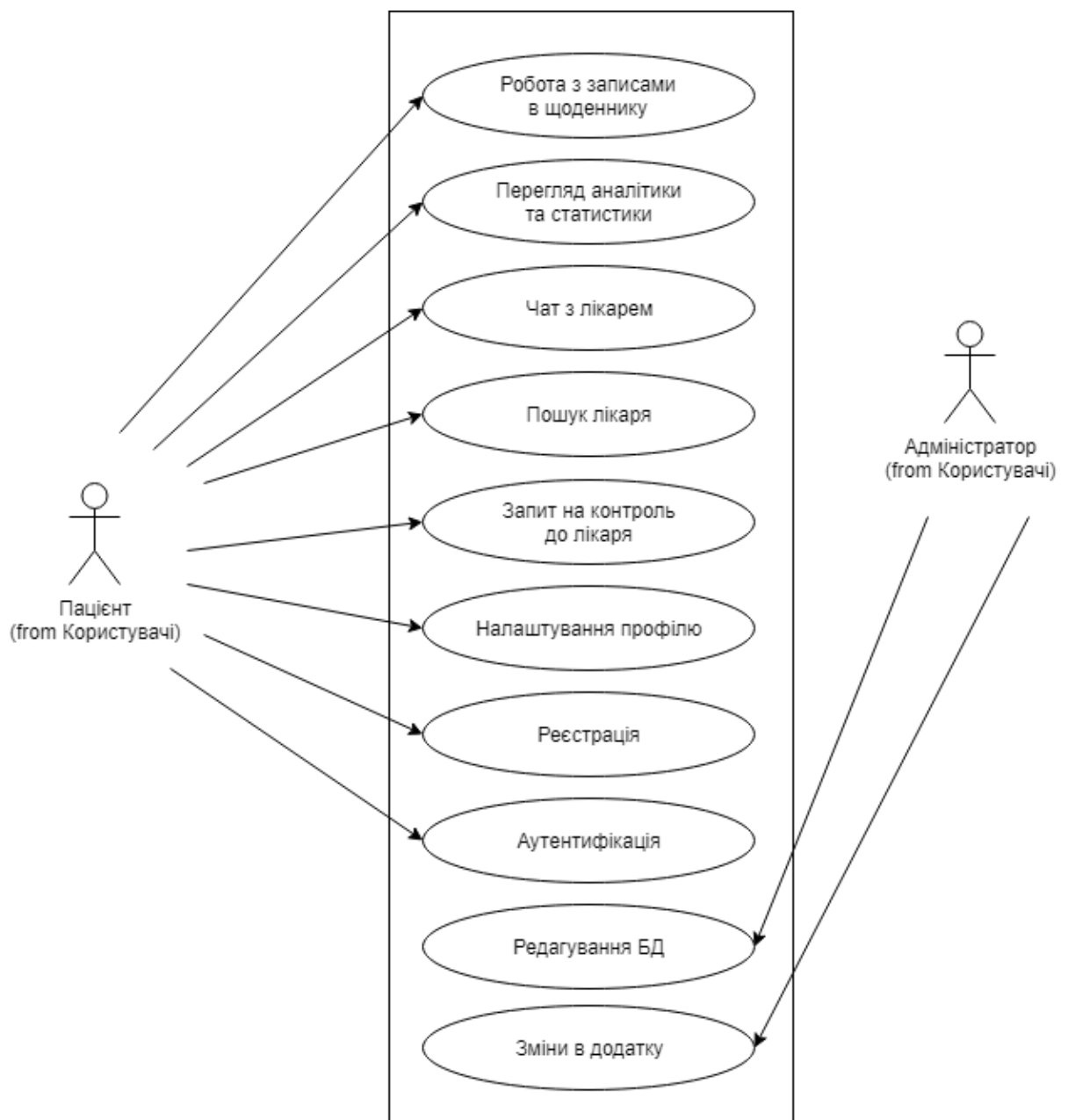


Рис. 2.1. Варіанти використання розроблюваного додатку



## **Нефункціональні вимоги**

### **1. Сприйняття**

- Час, потрібний для навчання інструментами роботи з інформаційною системою для звичайних користувачів - 3 хвилини, а для досвідчених - 1 хвилина;
- Час відповіді системи для звичайних запитів не повинен перевищувати 5 секунд, а для більш складних запитів - 10 сек.;
- Інтерфейс представлення додатку повинен бути інтуїтивно зручним для користувача та не вимагати від нього додаткової підготовки;

### **2. Надійність**

- Доступність – час, потрібний для обслуговування системи не повинен перевищувати 1% від загального часу роботи.
- Середній час безперервної роботи – 30 робочих днів.
- Максимальна норма помилок та дефектів в роботі системи – 1 помилка на 10000 запитів користувача.

### **3. Продуктивність**

Система повинна підтримувати мінімум 500 одночасно працюючих користувачів.

### **4. Можливість експлуатації**

- Масштабування – система повинна мати можливість збільшувати потужності (продуктивність), зі збільшенням користувачів таким чином, щоб це ні як негативно не відобразилося на її роботі;
- Оновлення версій – Оновлення версій повинно здійснюватися в автоматичному режимі залежно від вподобань користувачів;

## **Системні вимоги**

### **1. Вимоги до середовища виконання:**

Мінімальні вимоги до мобільного пристрою:

- 512Мб RAM;

- 50Mб ROM;
- Процесор з тактовою частотою 1,1GHz;
- Операційна система Android 5.0.1+ або iOS 10+;

## 2.2 Архітектура та узагальнена структура, програмного комплексу.

Для реалізації додатку будемо використовувати мову програмування Dart та її framework Flutter. Backend буде реалізовано за допомогою мови Node.js та framework Nest.js. Даний стек технологій зручно масштабувати, та використовувати додаток на різного роду платформах. На базі Flutter існують різні типи архітектури системи. Проаналізуємо їх та виділимо основні плюси та мінуси. Розглянемо деякі з цих підходів.

### *Native state*

Робота з Flutter складається з віджетів. Віджети в свою чергу можуть бути видимими, невидимими, містити дочірні віджети і взаємодіяти між собою. Можуть бути як віджетом без стану (Stateless Widget), так і віджетом, у якого є стан (Stateful Widget). Основна відмінність — можливість повторно відображати віджети під час виконання програми. Stateless Widget використовується тільки один раз і є незмінним. Stateful Widget може використовуватися безліч разів в залежності від зміни внутрішнього стану віджета.

Перевагами такого підходу є простота і швидкість впровадження в додаток з невеликою кількістю екранів, яка виражається у відсутності будь-яких додаткових бібліотек.

З недоліків можна виділити такі:

View та бізнес-логіка ніяк не розділені.

Складність в модульному тестуванні.

Складність в масштабуванні і підтримки.

Велика кількість коду, який неможливо використати повторно

### *Provider (Scoped Model)*

Provider тип архітектури дозволяє винести бізнес-логіку з Widget і дає можливість використовувати цю логіку в різних модулях системи. Робота системи здійснюється за допомогою створення моделей та реагуванням підписаних віджетів на її зміну.

Для локального стану віджета насамперед необхідно створити модель з усіма полями, які будуть використовуватися у віджеті. Після зміни кожного поля (або полів) потрібно повідомити підписані віджети, що модель змінилася, і виконати візуалізацію підписаних віджетів. Щоб підписати віджет на модель, використовується клас `ChangeNotifierProvider`, який є частиною бібліотеки `provider`. Підписка відбувається безпосередньо до того віджету, який буде залежати від даних зі створеної моделі.

Глобальний стан нічим не відрізняється від локального, крім того, що модель підписується до самого верхнього віджету додатку. Сама модель повинна мати унікальний клас, щоб не було конфліктів при її пошуку в дочірніх віджетах.

З переваг можна виділити поділ бізнес-логіки та інтерфейсу за допомогою створення моделей і event-based-архітектури. Тестувати такі моделі легше, ніж в Native state, за рахунок відсутності додаткових зусиль на створення віджетів, в яких цей стан використовується. Цей тип архітектури підтримується Google. Одна з основних проблем такої архітектури — складність в розумінні того, які властивості було змінено.

### *BLoC*

BLoC (Business Logic Component) — шаблон, створений Google для управління складним станом додатків, ґрунтується на реактивній парадигмі. Архітектура BLoC (business logic components) зображена на рисунку 2.2.

Основна ідея полягає в тому, що наш додаток розбитий на модулі, що реалізують бізнес-логіку. Кожен модуль має одну або кілька Sink (труб), які є деяким вхідним потоком для агрегування подій ззовні. В якості вихідних даних

виступає Stream (потік), який визначає асинхронний формат даних для наших віджетів. Щоб скористатися модулем на рівні керування, застосовують StreamBuilder, який керує потоком даних і автоматично вирішує проблеми підписки і перебуванням дочірнього дерева віджетів. Незважаючи на це, використовувати BLoC в чистому вигляді — досить складна робота, оскільки треба застосовувати бібліотеку RxDart для маніпуляції з потоками, власноруч відписуватися від потоків, інакше можна отримати серйозний витік пам'яті на великих додатках.

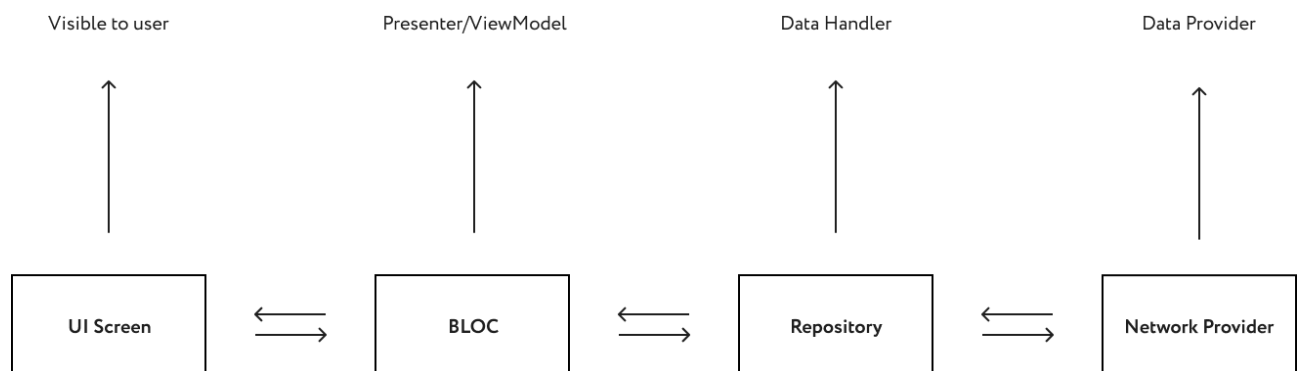


Рис. 2.2 - Схема архітектури BloC

Сам підхід цікавий і має велику кількість переваг:

- API при роботі з потоками, що дозволяє їх легко групувати, поєднувати і трансформувати;
- угрупування логіки в одному місці;
- легкість в тестуванні стану з сайд-ефектами за рахунок вбудованого в Dart API тестування потоків.

В проєкті використаємо тип архітектури BloC, він дозволяє легко і швидко впровадити необхідний бізнесу функціонал, окрім цього цей тип архітектури підтримується Google.

### *Концепція «клієнт-сервер»*

Для розробки повноцінного програмного комплексу, який буде включати в себе розроблюваний мобільний додаток та додатки для лікарів, буде використана «клієнт-серверна» архітектура з RESTful API для роботи зі спільною БД.

Концепція «клієнт-сервер» пов'язана з комп'ютерами спільного користування (серверами), які керують спільними ресурсами, і надають доступ до цих ресурсів як до сервісу своїм клієнтам. Обчислювальні мережі, побудовані на основі концепції «клієнт-сервер», дають змогу: реалізувати кооперативне управління ресурсами ЕОМ; виробити розподіл доступу до даних і процесів їх оброблення між множиною робо-чих станцій та сервером.

Сервер — одно- або багатопроцесорна персональна чи віртуальна ЕОМ з розподілюваною пам'яттю, розподілюваним обробленням даних, розподілюваними комунікаційними засобами та засобами управління периферійним обладнанням.

Одночасний доступ багатьох користувачів до інтегрованої бази даних (БД) реалізується в концепції «клієнт-сервер», згідно з якою, серверу належить більш активна роль. Запит на оброблення даних посилається клієнтом по мережі на сервер. На сервері здійснюється пошук даних і їх оброблення засобами системи керування БД. Оброблені дані передаються по мережі від серверу до клієнта. Специфікою архітектури «клієнт-сервер» є використання мови структурованих запитів SQL до БД, що забезпечує роботу зі спільними даними з різнотипних додатків у мережі.

Клієнт — робоча станція, що взаємодіє з користувачем, здатна виконувати потрібні обчислення і забезпечує приєднання до обчислювальних ресурсів та БД, засобів їх оброблення, а також засобів організації інтерфейсів.

Концепція «клієнт-сервер» означає, що кожна технологічна процедура потребує наявності трьох елементів: клієнта, який запитує інформацію; серверу, що цю інформацію надає; власне мережі. Сервер можна розглядати як

програмний компонент, що надає спільний функціональний сервіс іншим програмним компонентам. Клієнта можна розглядати як додаток, що формує і спрямовує запит до серверу. Він відповідає за оброблення, виведення інформації та передачу запитів серверу. Програма-сервер приймає запит, обробляє його і відправляє результат клієнту. Користувач взаємодіє тільки з програмою-клієнтом. При цьому в концепції «клієнт-сервер» програми клієнта та його запити зберігаються окремо від системи керування БД.

Схема «клієнт-сервер» наведена на рисунку 2.3.

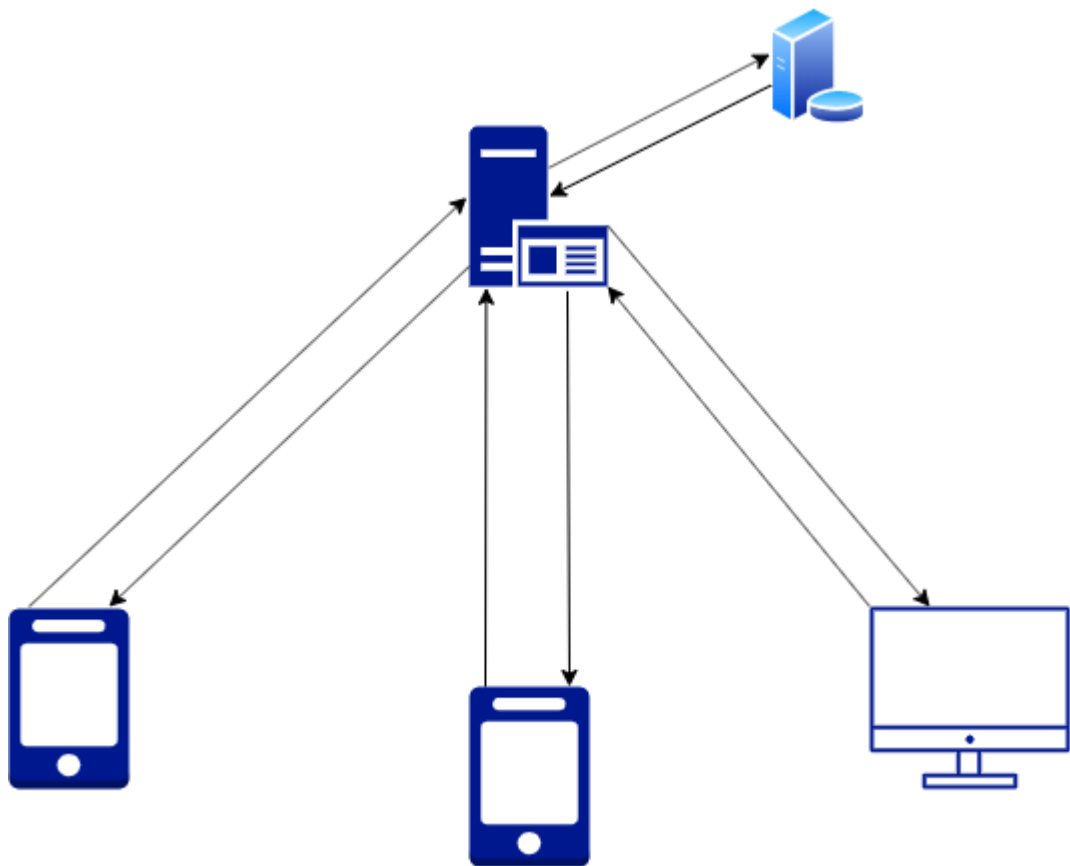


Рис.2.3. – Схема «клієнт-серверної» архітектури додатку

### 2.3 Обґрунтування вибору інструментальних засобів

Для створення мобільного додатку, який буде працювати, як на мобільній системі Android так і на iOS, доцільно застосувати: фреймворк Flutter, мову



програмування Dart, Node.JS, Javascript фреймворк Nest.JS, онлайн-сервіс для розробки інтерфейсів Figma.

### *Dart*

Dart - це об'єктно-орієнтована мова, яка за бажанням може бути перекомпільована в JavaScript. Вона підтримує різноманітний набір програмних засобів, таких як інтерфейси, класи, колекції, абстрактні класи, статичну типізацію.

Dart можна широко використовувати для створення односторінкових веб-програм, додатків для Android та iOS. Односторінкові програми дозволяють переходити між різними екранами веб-сайту без завантаження іншої веб-сторінки. Dart може використовуватися як для клієнтської частини, так і для серверної частини програмних продуктів.

В своєму розвитку Dart пережив вплив таких більш ранніх мов програмування, як Smalltalk, Java, JavaScript. Його синтаксис схожий на синтаксис других C-подібних мов програмування. [21]

### *Flutter*

Flutter - це SDK з відкритим кодом, який може використовувати для створення додатків Android та iOS. Flutter існує вже з 2015 року, коли Google представив його та залишився на стадії бета-версії до офіційного запуску в грудні 2018 року. [26]

Центральна ідея Flutter - це використання віджетів. Саме завдяки комбінуванню різних віджетів розробники можуть створювати весь користувацький інтерфейс. Кожен із цих віджетів визначає структурний елемент (наприклад, кнопку або меню), стилістичний елемент (шрифт або колірну схему) та багато інших.

Flutter не використовує віджети OEM, але він має вбудовані власні готові віджети, як для Android або iOS-додатків (за матеріалами Material Design або Cupertino). Звичайно розробники також можуть створювати власні віджети.

Flutter використовує Dart. Він компілює Dart ahead-of-time (AOT) у нативний код для декількох платформ.

Однією з найцікавіших особливостей Flutter є мова, якою він користується: Dart. Як і інші системи, що використовують reactive view, Flutter оновлює дерево перегляду для кожного нового кадру. Для цього він створює багато об'єктів, які можуть жити не більше одного кадру. Dart використовує generational garbage collector, який виявився дуже ефективним для систем такого типу.

Крім того, у Dart є компілятор «tree shaking», який включає лише необхідний код у вашій програмі. Навіть якщо вам потрібен лише віджет або два, ви можете вільно користуватися його великою бібліотекою віджетів.

### *Node.js*

Node.js представляє з себе середовище для виконання коду на JavaScript, яка побудована на основі движка JavaScript Chrome V8, який дозволяє транслювати виклики на мові JavaScript в машинний код. Node.js перш за все призначений для створення серверних додатків на мові JavaScript. Хоча також існують проекти по написанню десктопних додатків (Electron) і навіть зі створення коду для мікроконтролерів.

Перевагою використання Node.JS є масштабування. При одночасному підключенні до сервера тисяч користувачів Node працює асинхронно, тобто ставить пріоритети і розподіляє ресурси грамотніше. Java ж, наприклад, виділяє на кожне підключення окремий потік. [24]

### *Nest.js*

Nest.js - це масштабоване серверне середовище JavaScript, створене на основі TypeScript, яка зберігає сумісність з JavaScript, що робить його ефективним інструментом для створення продуктивних і надійних серверних додатків. Він має модульну архітектуру, яка забезпечує структурний шаблон проектування для розробки Node.js.

Він заснований на концепціях захисту, каналів і перехоплювачів і поставляється з вбудованою підтримкою WebSockets і gRPC. Nest.js використовує Express і повністю сумісний з великою кількістю популярних бібліотек. Він в основному використовує класи, декоратори, відображення метаданих для більшості своїх абстракцій.

### *Figma*

Figma — кросплатформний, векторний графічний редактор від компанії Figma. Призначений для проектування та створення інтерфейсів, також створює інтерактивні прототипи. Працює у двох форматах: у браузері та як клієнтський додаток на десктопі користувача. Зберігає онлайн-версії файлів, з якими працював користувач.

Figma є безкоштовною для індивідуальних користувачів і платною для фахових команд.

Даний редактор підходить як для створення простих прототипів і дизайн-систем, так і складних проектів (мобільні додатки, портали). У 2018 році платформа стала одним із тих інструментів для розробників і дизайнерів, що найбільш швидко розвиваються.

Може відкривати популярні графічні формати, зокрема: Scalable Vector Graphics (SVG), SKETCH, також експортувати у ці формати PNG, JPG та SVG.

### *Visual Studio Code*

Visual Studio Code — засіб для створення, редагування та зневадження сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки стало першим крос-платформовим продуктом у лінійці Visual Studio.

За основу для Visual Studio Code використовуються напрацювання вільного проекту Atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js. Примітно, що про використання напрацювань вільного проекту Atom на сайті Visual Studio Code і в прес-релізі і в офіційному блозі не згадується.

Редактор містить вбудований зневаджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки. Серед підтримуваних мов і технологій: JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, DockerFile, Coffee Script, Java, HandleBars, R, Objective-C, PowerShell, Luna, Visual Basic, Markdown, JSON, HTML, CSS, LESS і SASS, Haxe.

### *Adobe Photoshop*

Adobe Photoshop - багатофункціональний графічний редактор, розроблений і поширюваний фірмою Adobe Systems. В основному працює з растровими зображеннями, проте має деякі векторні інструменти. Продукт є лідером ринку в області комерційних засобів редагування растрових зображень і найбільш відомою програмою фірми Adobe.

В даний час Photoshop доступний на платформах macOS, Windows, в мобільних системах iOS, Windows Phone і Android. Також існує версія Photoshop Express для Windows Phone 8 і 8.1.

### *СКБД*

Порівняльний аналіз СКБД дозволяє раціонально вибрати систему керування базами даних для проекту. У якості альтернатив будуть розглянуті наступні СКБД: **PostgreSQL, MySQL**.

**MySQL** - це найпопулярніша з усіх великих серверних БД. Розібратися в ній дуже просто, та й в мережі про неї можна знайти велику кількість інформації. Хоча MySQL і не намагається повністю реалізувати SQL-стандарти, вона пропонує широкий функціонал. Додатки спілкуються з базою даних через процес-демон.

**Відомі обмеження:** по визначенню, MySQL не може зробити все, що завгодно, і в ній присутні певні обмеження функціональності. Наприклад, в ній не має повноцінного FULL JOIN запиту. Деякі операції реалізовані менш надійно, ніж в інших СКБД. Застій в розробці: хоча MySQL і є open-source продуктом, робота над нею сильно загальмована. Проте, існує кілька БД, повністю заснованих на MySQL (наприклад, MariaDB).

**Переваги:** в MySQL вбудовано багато функцій безпеки. MySQL може працювати з дійсно великими обсягами даних, і непогано підходить для масштабованих додатків. Швидкість: нехтування деякими стандартами дозволяє MySQL працювати продуктивніше.

**PostgreSQL** - це сама просунута СКБД, що орієнтується в першу чергу на повну відповідність стандартам і розширюваність. PostgreSQL, або Postgres, намагається повністю відповідати SQL-стандартам ANSI / ISO.

PostgreSQL відрізняється від інших СКБД тим, що володіє об'єктно-орієнтованим функціоналом, в тому числі повною підтримкою концепту ACID (Atomicity, Consistency, Isolation, Durability).

**Недоліки:** В простих операціях читання PostgreSQL може поступатися своїм суперникам в швидкодії. Через свою складність інструмент не дуже популярний.

**Переваги:** Повна SQL-сумісність. PostgreSQL використовується в багатьох інструментах, пов'язаних з СКБД. PostgreSQL можна програмно розширити за рахунок збережених процедур. PostgreSQL - не тільки реляційна, а й об'єктно-орієнтована СУБД.

Після порівняння таких СКБД, як **PostgreSQL**, **MySQL** можна сказати, що оптимальним вибором буде PostgreSQL. Дана СКБД має більший список функцій та надійність і цілісність даних у порівнянні з конкурентами є кращою.

## 2.4. Розробка бази даних системи

Реалізована база даних PostgreSQL складається з 16 таблиць, які містять у собі всі дані для роботи додатку. Структура бази даних наведена на рис. 2.4.

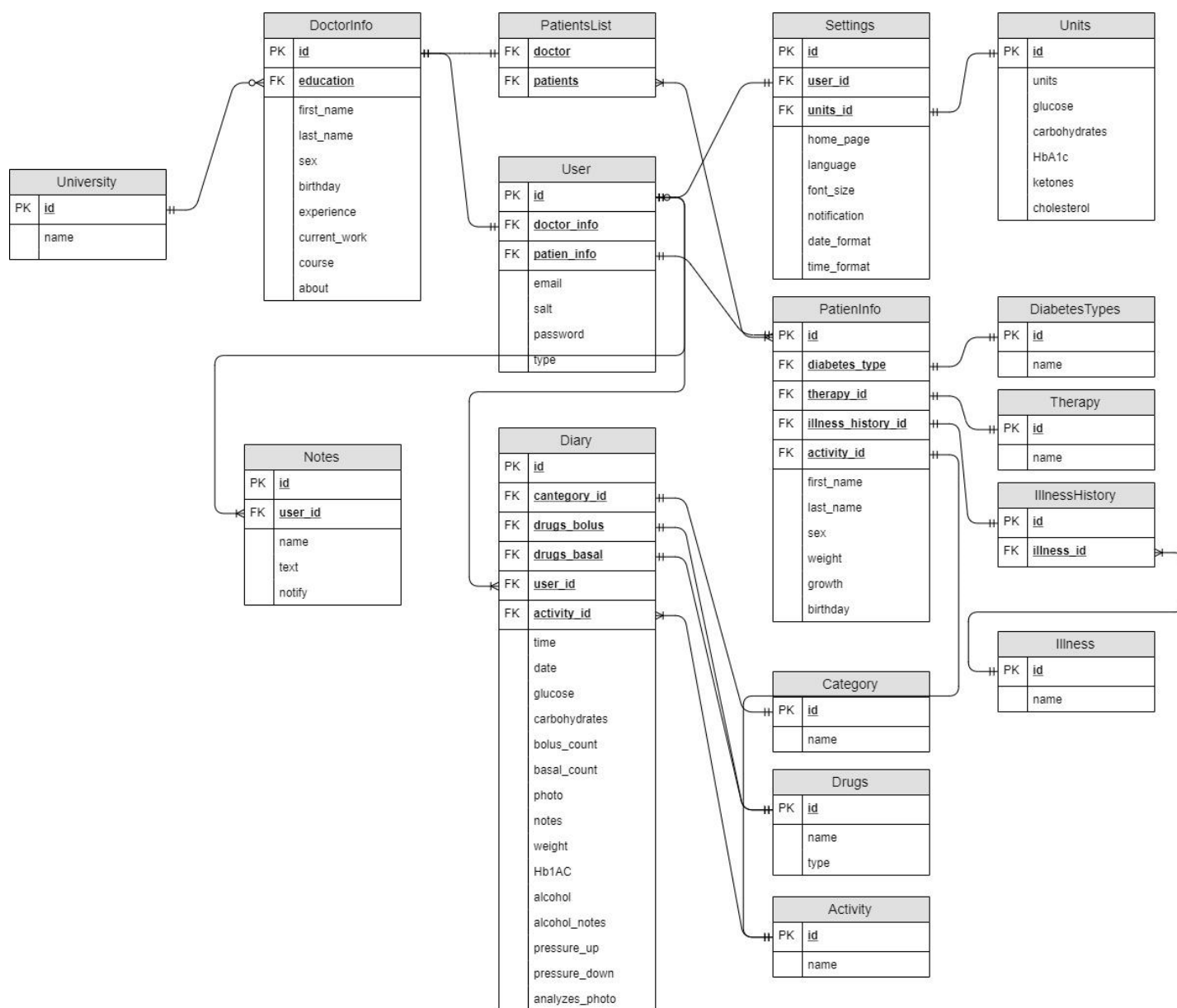


Рис. 2.4. Структурна схема БД

Далі наведемо опис основних таблиць бази даних.

Таблиця «User» призначена для зберігання головної інформації користувачів додатку. В даній таблиці зберігається пароль, пошта та тип користувача: лікар або пацієнт.

Клас який описує дану таблицю:

#### **User**

```
@Entity()
@Unique(['email'])
export class User extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  email: string;

  @Column()
  password: string;

  @Column()
  salt: string;

  @Column()
  type: string;

  @OneToOne(type => Patient)
  @JoinColumn()
  patient_info: Patient;

  async validatePassword(password: string): Promise<boolean> {
    const hash = await bcrypt.hash(password, this.salt);
    return hash === this.password;
  }
}
```

Таблиця «Settings» призначена для зберігання налаштувань самого додатку користувачів.

Таблиця «Diary» призначена для зберігання записів користувача. Запис в даній таблиці містить ряд важливих показників, які вносить пацієнт.

Таблиця «PatientInfo» містить в собі інформацію про пацієнта, таку як: ім'я, вік, стать, вагу, ріст, історію хвороб, терапію пацієнта, тип діабету.

Також база даних містить в собі таблиці, які створені для оптимізації структури бази. До цих таблиць входять: «Units», «DiabetesTypes», «Therapy», «Illnes», «IllnesHistory», «Category», «Drugs», «Activity».

Отже реалізована база даних у відповідності до вимог реляційної моделі, яка забезпечує збереження та доступ до інформації додатку. Реалізація функціональних частин додатку

## 2.5 Реалізація функціональних частин додатку

Мобільний додаток було реалізовано за допомогою Flutter, розглянемо головний файл додатку – main.dart.

```
Future main() async {
  await DotEnv().load('.env');
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider.value(
          value: Auth()
        ),
      ],
      child: Consumer<Auth>(
        builder: (ctx, auth, _) => MaterialApp(
          theme: ThemeData(
            fontFamily: 'Circe',
            brightness: Brightness.light,
            primaryColor: Color.fromRGBO(47, 163, 156, 1),
```



```

        primaryColorDark: Color.fromRGBO(31, 31, 31, 1),
    ),
    home: auth.isAuth ? InfoScreen() : FutureBuilder(
        future: auth.tryAutoLogin(),
        builder: (ctx, authResultSnapshot) =>
authResultSnapshot.connectionState == ConnectionState.waiting ?
SplashScreen() : AuthMainScreen(),
    ),
    routes: {
    },
),
),
);
}
}

```

В цьому файлі знаходиться головна функція `main()`, яка власне ініціалізує додаток. Також в цій функції завантажуються змінні оточення. Окрім головної функції в цьому файлі знаходиться головний клас додатку – `MyApp`, він наслідує базовий клас `StatelessWidget`. В цьому класі ми реєструємо провайдери, які необхідні для роботи з даними додатку і задаємо базові налаштування зовнішнього вигляду додатку.

Для взаємодії з API використовуються дата-провайдери. Принцип роботи дата-провайдеру розглянемо на прикладі `auth` провайдера. У цьому класі зберігається інформація по авторизованого користувача, а саме `_token`, `_expiryDate`, `_userId`, `isAuth`. `Auth` провайдер реалізує такі функції – `signup`, `signin`, `tryAutoLogin`, `logout`, `autoLogout`.

```

class Auth with ChangeNotifier {
    final baseUrl = DotEnv().env['BASE_URL'];
    String _token;
    DateTime _expiryDate;
    int _userId;
    Timer _authTimer;

    bool get isAuth {
        return token != null;
    }
}

```

```

String get token {
    if (_expiryDate != null && _expiryDate.isAfter(DateTime.now()) &&
_token != null) {
        return _token;
    }
    return null;
}

Future<bool> tryAutoLogin() async {
    final prefs = await SharedPreferences.getInstance();
    if (!prefs.containsKey('userData')) {
        return false;
    }
    final executedUsedData = json.decode(prefs.getString('userData')) as
Map<String, Object>;
    final expiryDate = DateTime.parse(executedUsedData['expiryDate']);

    if (expiryDate.isBefore(DateTime.now())) {
        return false;
    }

    _token = executedUsedData['token'];
    _userId = executedUsedData['userId'];
    _expiryDate = expiryDate;
    notifyListeners();
    _autoLogout();
    return true;
}

void logout() async {
    _token = null;
    _userId = null;
    _expiryDate = null;
    if (_authTimer != null) {
        _authTimer.cancel();
        _authTimer = null;
    }
    notifyListeners();
    final prefs = await SharedPreferences.getInstance();
    prefs.remove('userData');
    prefs.clear();
}

void _autoLogout() {
    if (_authTimer != null) {
        _authTimer.cancel();
    }
    final timeToExpire = _expiryDate.difference(DateTime.now()).inSeconds;
    _authTimer = Timer(
        Duration(

```

```

        seconds: timeToExpire,
    ),
    logout
);
}
}

```

Розглянемо схему роботи блока аутентифікації AuthBloc (Рис. 2.5). В залежності від екрану на якому було створено подію, вона потрапляє до Sync (Труби) в AuthBloc, де ця подія оброблюється, та на виході через Stream (Потік) змінюється State (Стан) в блоці. В залежності від того, який був State, відкривається відповідний екран.

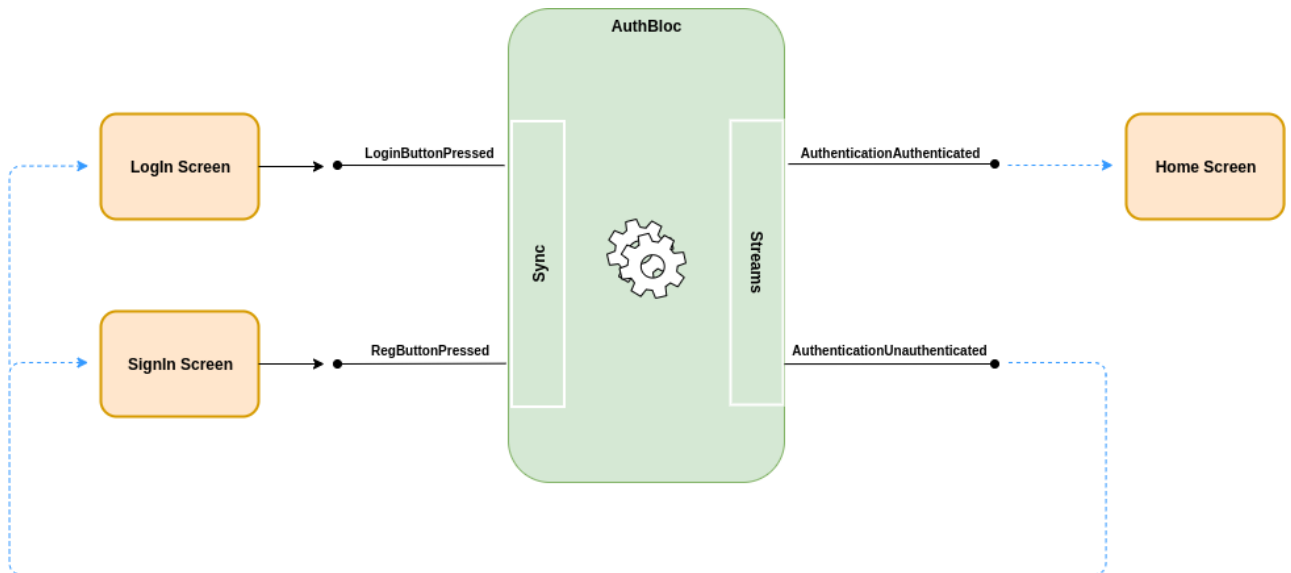


Рис. 2.5 – Схема роботи блока аутентифікації AuthBloc

Signup – використовується для реєстрації користувача в системі. В тілі функції йде звернення до API. В разі успішного запиту йде виклик функції signin. Якщо відповідь від сервера приходить з помилкою, то це все оброблюється і користувачу показується відповідна помилка.

```

Future<void> signup(String email, String password) async {
    final url = '$baseUrl/auth/signup';
    try {
        final response = await http.post(
            url,

```

```

        body: json.encode({'email': email, 'password': password,}),
        headers: {
            HttpHeaders.contentTypeHeader: 'application/json',
        }
    );
    if (response.statusCode != 201) {
        final responseData = json.decode(response.body);
        if (responseData['message'] is List<dynamic>) {
            throw
HttpException(responseData['message'][0]['constraints']['matches']);
        }
        throw HttpException(responseData['message']);
    }
    signin(email, password);
} catch (error) {
    throw error;
}
}

```

Signin – використовується для авторизації користувача в системі. В тілі функції йде звернення до API і у разі успішного запиту дані про користувача записуються в провайдер, а також ці дані зберігаються на пристрої користувача. Це необхідно для повторної авторизації користувача в фоновому режимі.

```

Future<void> signin(String email, String password) async {
    final url = '$baseUrl/auth/signin';
    try {
        final response = await http.post(
            url,
            body: json.encode({'email': email, 'password': password,}),
            headers: {
                HttpHeaders.contentTypeHeader: 'application/json',
            }
        );
        if (response.statusCode != 200) {
            final responseData = json.decode(response.body);
            if (responseData['message'] is List<dynamic>) {

```

```

        throw
HttpException(responseData['message'][0]['constraints']['matches']);
    }
    throw HttpException(responseData['message']);
}
final data = json.decode(response.body);
_token = data['accessToken'];
_userId = data['userId'];
_expiryDate = DateTime.now().add(
    Duration(
        seconds: int.parse(
            data['expireIn']
        ),
    ),
);
_autoLogout();
notifyListeners();
final pref = await SharedPreferences.getInstance();
final userData = json.encode({
    'token': _token,
    'userId': _userId,
    'expiryDate': _expiryDate.toIso8601String(),
});
pref.setString('userData', userData);
} catch (error) {
    print(error);
    throw error;
}
}

```

Розглянемо реалізацію нижньої панелі навігації. Рівень презентації включає в себе набір компонентів, які слугують для відображення елементів навігації. Для динамічного створення елементів навігації розроблено клас `NavigationItem`

```

class NavigationItem {
    final Icon icon;
    final Text title;
    NavigationItem(this.icon, this.title);}

```

Клас `NavigationItem` включає в себе поля `icon` типу `Icon` та `text` типу `Text`. Цей клас використовується для генерації елементів навігації. За генерацію елементів відповідає функція `_buildItem`. Вхідними параметрами функція приймає `item` та `isSelected`.

```
Widget _buildItem(NavigationItem item, bool isSelected) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      IconTheme(
        data: IconThemeData(
          size: 32,
          color: isSelected ? Theme.of(context).primaryColor :
Theme.of(context).primaryColorLight,
        ),
        child: item.icon,
      ),
      SizedBox(
        width: isSelected ? 13 : 0,
      ),
      AnimatedDefaultTextStyle(
        duration: Duration(milliseconds: 200),
        child: item.title,
        style: isSelected ? TextStyle(fontSize: 12, fontWeight:
FontWeight.bold, color: Theme.of(context).primaryColor) :
TextStyle(fontSize: 0),
      ),
    ],
  );
}
```

В тілі функції на основі вхідних параметрів генерується елемент навігації. Елемент навігації має такі складові - іконка та текста. Також, в залежності від того, чи елемент навігації активний змінюється колір для іконки і відображається текст.

На бізнес рівні панелі навігації знаходиться BottomNavigationBloc. Він відповідає за зміну стану навігації і в залежності від стану навігації змінюється поточна сторінка. Розглянемо стани, які може приймати модуль навігації.

Модуль навігації може приймати 6 станів. Кожний стан відповідає за відображення певної сторінки.

```
abstract class BottomNavigationState extends Equatable {
    const BottomNavigationState();
}
class InitialBottomNavigationState extends BottomNavigationState {
    @override
    List<Object> get props => [];
}
class HomePageLoaded extends BottomNavigationState {
    @override
    List<Object> get props => [];
}
class DiaryPageLoaded extends BottomNavigationState {
    @override
    List<Object> get props => [];
}
class AnalyticsPageLoaded extends BottomNavigationState {
    @override
    List<Object> get props => [];
}
class DoctorPageLoaded extends BottomNavigationState {
    @override
    List<Object> get props => [];
}
class SettingPageLoaded extends BottomNavigationState {
    @override
    List<Object> get props => [];
}
```

Також у цьому модулі присутній абстрактний клас BottomNavigationState, який слугує для задання методів та властивостей, які мають бути реалізовані в класах, які його наслідують. В даному випадку це властивість props, яку реалізують всі класти станів.

## Висновки до другого розділу

Даний розділ описує обрану архітектуру додатку та технології для її реалізації. Для написання додатку будуть використовуватися наступні технології: мова програмування Dart, Framework Flutter, Node.js, Nest.js, Середовище керування базами даних PostgreSQL.

В даному розділі була реалізована база даних у відповідності до вимог реляційної моделі, яка забезпечує збереження та доступ до інформації додатку.

Було розглянуто декілька алгоритмів роботи додатку та наведено лістинг їх коду. В останньому пункті наводиться детальний розгляд коду деяких компонентів реалізованого додатку. Основну увагу було приділено головному файлу додатку. Також було розглянуто принцип роботи дата-провайдерів на прикладі auth провайдера.



## РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МОБІЛЬНИМ ДОДАТКОМ «ЩОДЕННИК САМОКОНТРОЛЮ ЦУКРОВОГО ДІАБЕТУ»

### 3.1. Структура інтерфейсу. Інтерфейс та порядок роботи з додатком

Після авторизації користувач потрапляє на головний екран. На ньому можна переглянути статистику основних показників, а саме: кількість активного болюсного інсуліну та час до його повної утилізації, останнє вимірювання рівня глюкози, кількість вуглеводів за день, середній рівень ГК за 7 днів, середнє відхилення ГК за 7 днів та час проведений в рамках допустимих рівнів ГК. Також є діаграма яка показує розподіл рівнів ГК по рівням за 3 останні місяці, де кожен сектор з відповідним кольором відображає певний рівень цукру. Щоб переглянути історію зміни рівнів ГК, користувачу доступних графік змін рівнів ГК (рисунок 3.1).



Рис. 3.1. - Таблица основных показателей stanu компенсації



Рис. 3.2 - Діаграма стану та графік змін ГК

На головному екрані присутня кнопка для створення нового запису в щоденник. Натиснувши на неї, відкривається модальне вікно з формою (рис. 3.4). Навігація по додатку здійснюється через 5 вкладок: Головний екран, Щоденник, Аналітика, Чат з лікарем, Налаштування.

На екрані «Щоденник» (рис.3.3), користувач може переглянути всі записи відсортовані та згруповані по даті та часу. Записи відображаються у вигляді карток з введеними показниками.

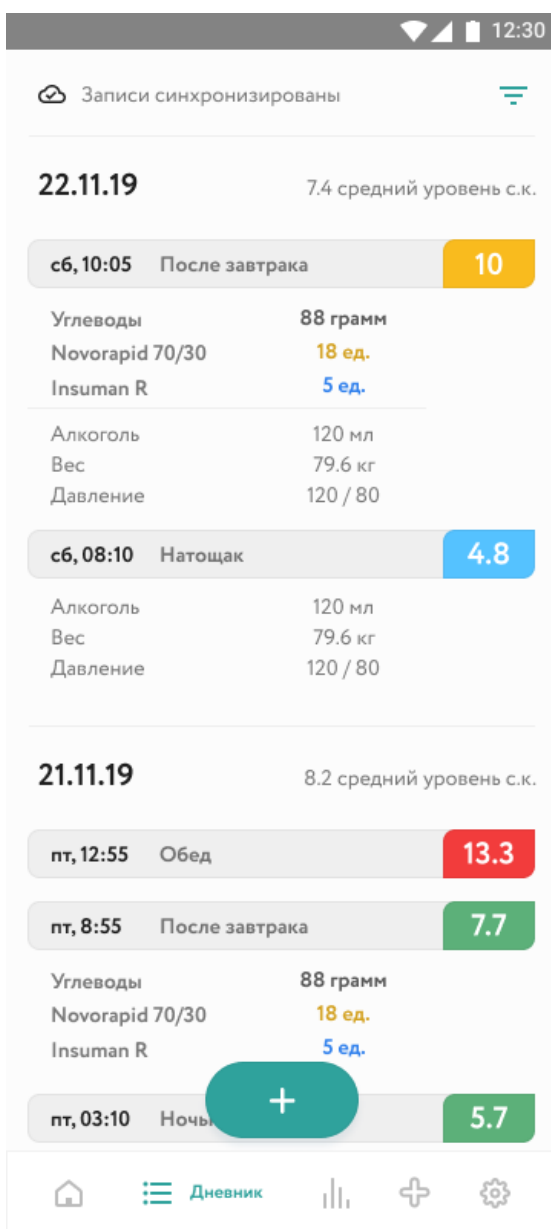


Рис. 3.3 - Экран щоденнику

Натиснувши на будь який запис, користувач може відкрити вікно для того щоб відредагувати або видалити запис. (рис. 3.5 - 3.6). В даному вікні є форма з

усіма потрібними полями для запису: ліки, їжа, активність, рівень цукру. Також користувач може обрати час та дату запису. В режимі редагування запису, в навігацій панелі з'являється кнопка для видалення запису, якщо на неї натиснути, то з'явиться діалогове вікно для підтвердження видалення запису.

Назад СОХРАНИТЬ

Основные Дополнительно

Дата 20 05 2019 + -

Время 12 : 25

Категория После завтрака

Глюкоза 00.00 mmol/L

Углеводы 00 грамм

Аpidra 00.00 ед.

Лantus 00.00 ед.

ДОБАВИТЬ ЗАПИСЬ

Примечания

Дневник

Рис. 3.5 - Форма запису - Основное

Форма запису поділена дві частини, це основна інформація та додатку. До основної входять показники рівню ГК, кількість вуглеводів, кількість, тип та марка ліків, які прийняв пацієнт. До додаткової частини входить фізична активність, вага, аналіз Hb1Ac, кількість алкоголю та заміри тиску, це ті показники які будуть рідше використані при створенні записів в щоденник.

← Назад СОХРАНИТЬ

Основные **Дополнительно**

Вес 78.2 кг

Hb1AC 8.4 mmol/L

Физ. актив. Тренажерный зал ▼

Алкоголь 0 мл

Примечания

Запись до 100 символов...

**ДОБАВИТЬ ЗАПИСЬ**

Верхнее 000 mmHg

**Дневник**

Рис. 3.6 - Форма запису - Додаткове

Перейшовши до екрану Аналітика, користувач може переглянути статистику по різноманітним показникам: підрахунок рівнів ГК, критичні показники ГК за певні періоди, кількість використаних ліків, оцінка шансів на гіпоглікемію або гіперглікемію тощо. Також користувачу доступні графіки, де можна побачити як змінювалась ГК на протязі певного періоду; діаграми з розподілом рівнів глюкози по категоріям записів (натщесерце, обід, вечер тощо) (рис. 3.7).

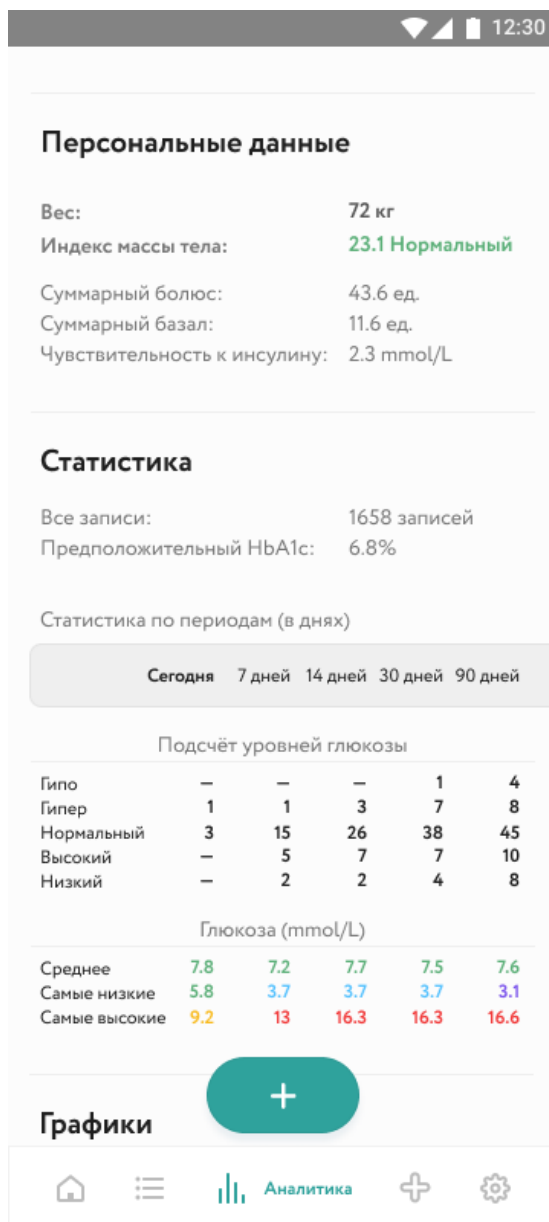


Рис. 3.7. Экран аналітики

Однією з головних функцій додатку є пошук та спілкування з лікарем. Для цього передбачено екран пошуку лікаря (рис. 3.9.) та профіль лікаря (рис. 3.10), де можна переглянути детальну інформацію про лікаря: місце навчання, стаж роботи, спеціалізація, місце поточної роботи, завантаженість лікаря тощо. Якщо лікар влаштовує пацієнта, то він може або написати лікарю (рис. 3.8), або відразу записатися на контроль до обраного лікаря натиснувши на кнопку «Записатися к доктору» (лікар має підтвердити цю заявку). Після підтвердження заявки, лікар

матиме доступ до аналітики та щоденника пацієнта. А у пацієнта замість пошуку лікарів тепер з'явиться чат з обраним лікарем.

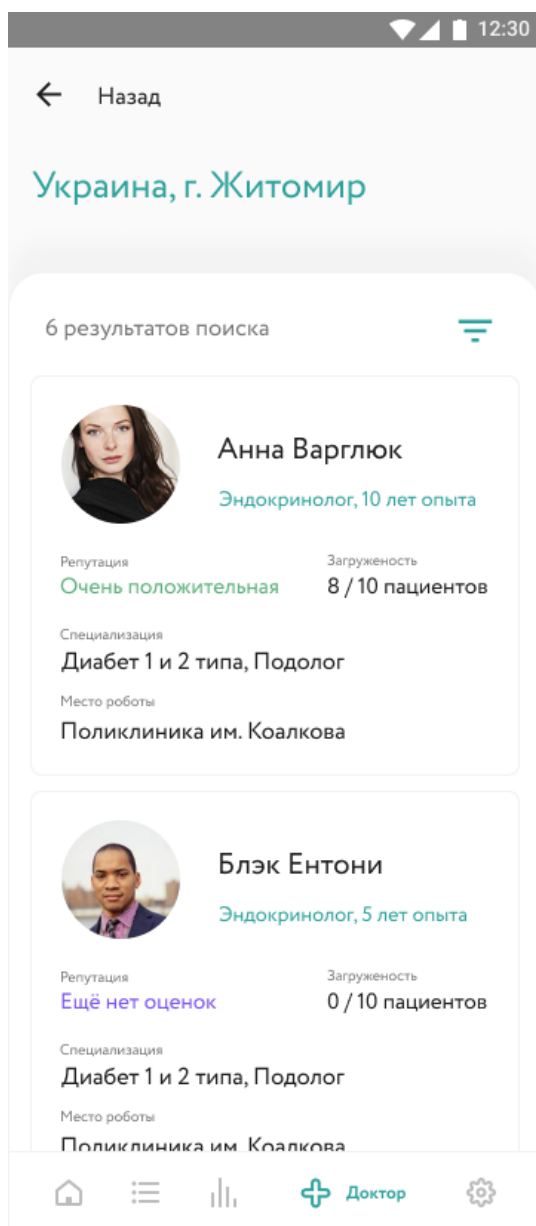


Рис. 3.8 – Форма пошуку лікарів

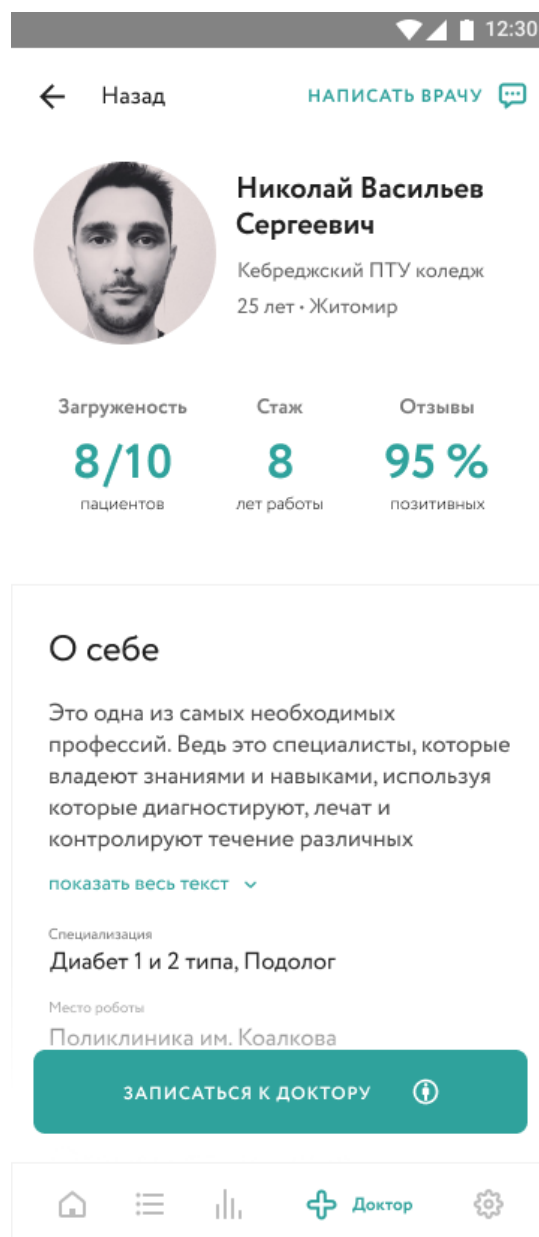


Рис.3.9 – Профіль лікаря

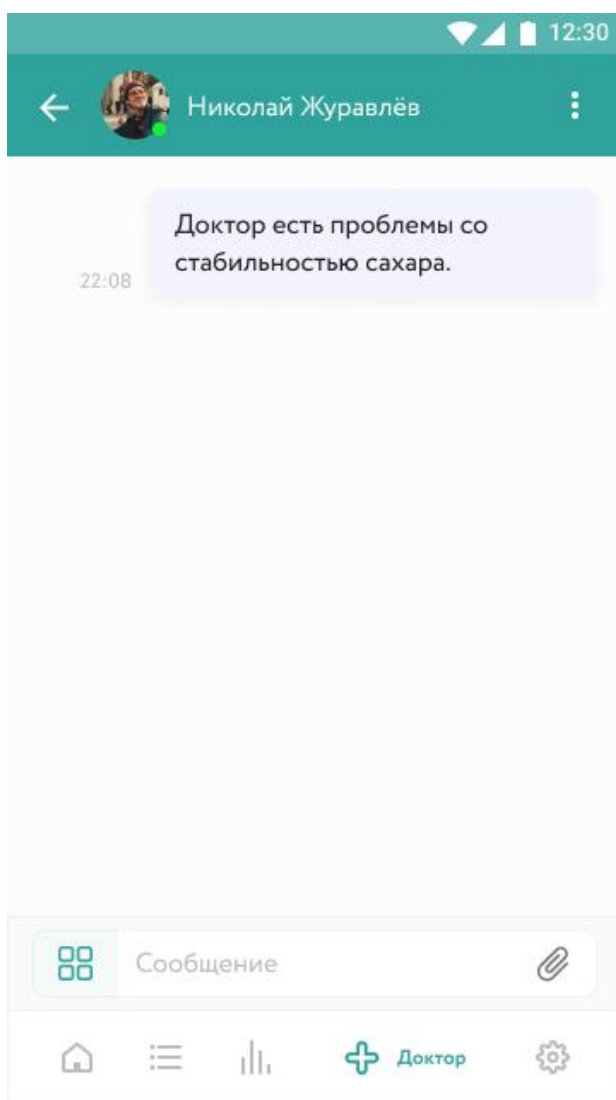


Рис.3.10 – Профіль лікаря

### 3.2. Тестування роботи мобільного додатку

При розробці мобільного додатку «Щоденник самоконтролю цукрового діабету» було проведене системне тестування, функціональне тестування та тестування графічного інтерфейсу. Під час розробки додатку та його тестування було створено, налаштовано та наповнено тестову базу даних. Далі розглянемо детальніше деякі види тестування.

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття

якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог.

Тестування програмного забезпечення можна розділити на декілька груп:

Functional testing (Функціональне тестування) — передбачає аналіз функціональних характеристик додатка та перевірку на невідповідності між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і бізнес-вимоги.

Non-functional testing (Нефункціональне тестування) — тестування властивостей, які не належать до функціональності системи. Які характеризують продукт з таких сторін, як: надійність, продуктивність, зручність, безпека.

Regression testing (Регресійне Тестування) — тестування вже раніше протестованої частини програми, після будь-якої зміни в коді, для того щоб переконатися що виправлений баг або оновлені налаштування сервера, або системи не вплинули на стару функціональність.

Кросплатформеність – здатність програмного забезпечення працювати більш, ніж на одній апаратній платформі.

#### **Тестування в певному середовищі.**

Мета: Перевірити коректну роботу і дизайн додатку на різних пристроях, з різними характеристика та рівнями API системи. Опис конфігурацій (табл.3.1)

Таблиця 3.1

Параметри конфігураційного тестування

Пристрій	Рівень API
Nexus 4	19
Nexus 4	21
Nexus 5	23
Nexus 6P	24



Nexus 6P	25
Pixel 2	26
Pixel 2	27
Pixel 2	28

Дане тестування було проведено за допомогою TestLab – це сервіс від компанії Google, який дозволяє проводити автоматичне (або по заданому сценарію) тестування мобільних додатків на різних пристроях та різних версіях операційної системи Android. Результати тестування (рис.3.10).

Выполнение теста	Продолжительность	Региональные настройки	Ориентация	Проблемы
✓ Pixel 2, уровень API 26	—	русский (Россия)	Вертикальная	—
✓ Nexus 4, Virtual, уровень API 21	—	русский (Россия)	Вертикальная	—
✓ Pixel 2, уровень API 28	—	русский (Россия)	Вертикальная	—
✓ Nexus 5, Virtual, уровень API 23	—	русский (Россия)	Вертикальная	—
✓ Nexus 6P, Virtual, уровень API 24	—	русский (Россия)	Вертикальная	—
✓ Nexus 6P, Virtual, уровень API 25	—	русский (Россия)	Вертикальная	—
✓ Nexus 4, Virtual, уровень API 19	—	русский (Россия)	Вертикальная	—
✓ Pixel 2, уровень API 27	—	русский (Россия)	Вертикальная	—

Рис.3.10 - Результати автоматичного тестування в TestLab

Отже було проведено комплексне тестування розробленого мобільного додатку. Було проведено наступні типи тестування: функціональне, системне, тестування сумісності та тестування графічного інтерфейсу. За результатами тестування було виявлено 10 функціональних помилок середнього пріоритету та

6 помилок в інтерфейсі. Всі помилки було виправлено, на даний час додаток працює коректно та виконує всі поставлені задачі.

### 3.3 Аналіз та моделювання даних на базі записів додатку

В даному розділі розглянемо статистичні графіки по хворобі цукровий діабет, що базуються на даних зібраних додатком.

Для проведення аналізу даних та моделювання прогнозу буде використовуватись мова програмування Python та відповідні бібліотеки, для роботи з даними та їх візуалізації у вигляді графіків.

Бібліотеки Python – це файли з шаблонами коду. Вони існують для того, щоб не доводилося щоразу заново набирати один і той же код: бібліотеки просто відкривають файл, вставляють свої дані і отримують потрібний результат.

**Pandas** — високорівнева Python бібліотека для аналізу даних. Вона побудована поверх більш низькорівневої бібліотеки NumPy (написана на Сі), що є великим плюсом в продуктивності. В екосистемі Python, pandas є найбільш просунутим інструментом. Це гарний варіант для обробки неповних, непорядкованих і немаркованих даних (як раз такі найчастіше і зустрічаються в житті). Pandas дозволяє замінити досить складні операції з даними на одну-дві команди. Містить багато готових методів групування, фільтрації, об'єднання даних, а також можливість розпізнавання різних типів джерел. У бібліотеці можна об'єднувати таблиці за аналогією з SQL JOIN. При цьому дані беруться прямо з файлів, завдяки чому зникає необхідність організації баз даних. Ще одна особливість Pandas – швидкість роботи.

**Seaborn** - бібліотека візуалізації даних Python, яка базується на matplotlib. Вона забезпечує інтерфейс високого рівня для малювання привабливої та інформативної статистичної графіки.

Bokeh створює інтерактивні і масштабовані графіки в браузері, використовуючи віджети JavaScript. Має три рівні інтерфейсу, від високого, який дозволяє швидко створювати складні графіки, до низького.

Basemap використовується для створення карт. На її основі зроблена бібліотека Folium, за допомогою якої створюють інтерактивні карти в інтернеті.

Під час роботи було здійснено аналіз та моделювання даних за наступними показниками пацієнтів:

- Розподіл рівнів цукру по категоріям
- Розподіл середніх рівнів глюкози по користувацьким категоріям

Таблиця 3.2

Розподіл середніх рівнів глюкози по користувацьким категоріям

Категорія	Середній рівень ГК
Натщесерце	8.66
Перед сніданком	7.51
Сніданок	5.52
Після сніданку	6.74
Перед обідом	6.35
Обід	6.88
Після обіду	7.89
Перед вечерею	5.89
Вечеря	5.74
Після вечері	6.35
Перед сном	5.99
Ніч	6.35
Інше	7.98

В таблиці показаний середній рівень глюкози в крові по категоріям запису, які вказують коли було зроблено замір. Можна зробити висновок, що найбільші

рівні цукру знаходяться в категоріях «натщесерце» та «після обіду». Це вказує на те, що хворий повинен переглянути програму лікування та скореговувати дози ліків на ці проміжки часу.

Розподіл рівнів цукру по категоріям

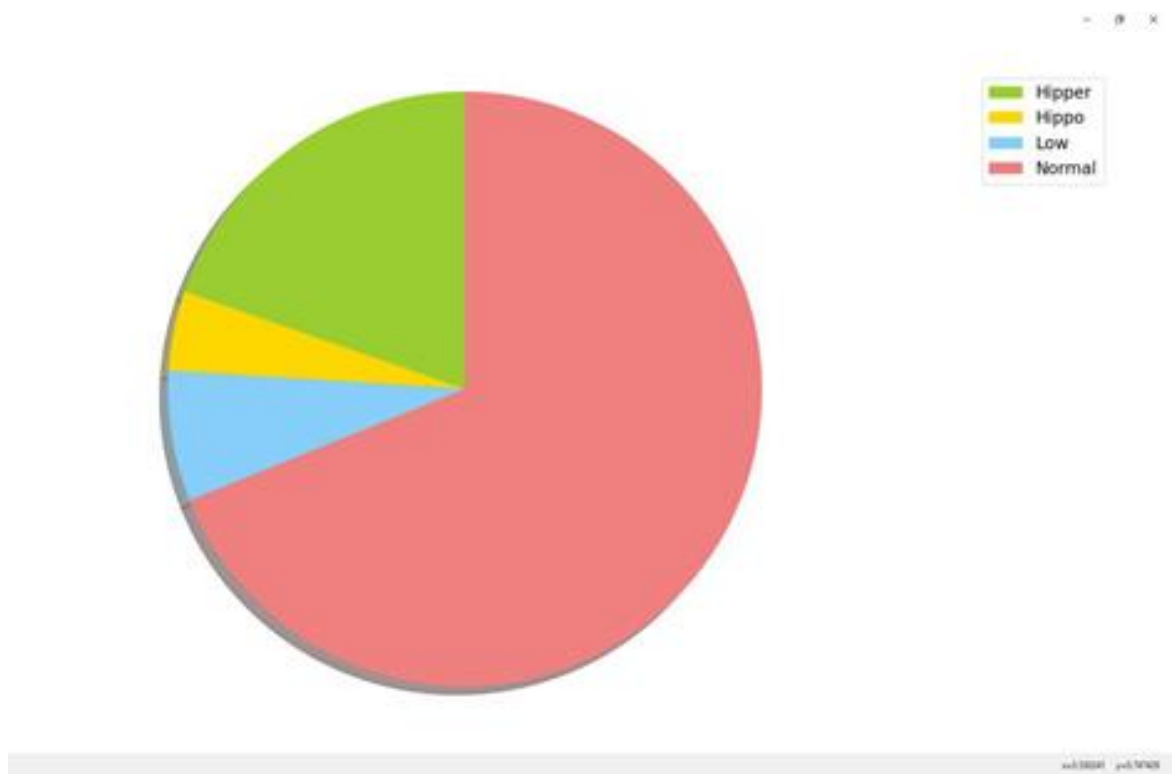


Рис. 3.11 – Розподіл рівнів цукру по категоріям

На даному графіку відображено розподіл рівнів цукру пацієнта за категоріями. І ми можемо зробити висновок, що у пацієнта більшу частину часу глюкоза в крові знаходиться в нормі, але має деяку схильність до гіперглікемії, тобто високого рівню глюкози в крові. А прояви гіпоглікемії є рідким явищем для даного пацієнта.

### Висновки до третього розділу

Даний розділ містить інформацію про інтерфейс додатку. Детально описано кожен екран функціональних модулів розробленого додатку.

В даній частині кваліфікаційної роботи також проведено комплексне тестування розробленого мобільного додатку. Було проведено наступні типи

тестування: функціональне, системне, тестування сумісності та тестування графічного інтерфейсу. За результатами тестування було виявлено ряд помилок, які наразі виправлені, та додаток виконує всі поставлені задачі.

На основі зібраних додатком даних було проведено статистичний аналіз одного випадкового обраного пацієнта. Наприклад, даний аналіз дозволив виявити час доби в який рівень ГК має тенденцію до гіперглікемії, це спостереження може бути конвертоване в рекомендації пацієнту що до перегляду його терапій в дані відрізки часу.

## ВИСНОВКИ

Результатом кваліфікаційної магістерської роботи є спроектований додаток «Щоденник самоконтролю цукрового діабету», для його реалізації було виконано ряд наступних етапів.

В першому розділі кваліфікаційної роботи було проаналізовано поставлене завдання та визначено схему подальшої розробки програмного забезпечення. Проаналізовано існуюче програмне забезпечення, а саме щоденники самоконтролю цукрового діабету на мобільних пристроях.

Було визначено основний функціонал розроблюваного додатку та засоби виконання поставлених задач. Розроблюваний додаток буде мати такі функції як: синхронізація, додавання записів в щоденник, вибір ліків, отримання статистики за обраний період, графік активності ліків. Також додаток має працювати як на мобільній системі Android, так і на iOS.

Даний розділ описує обрану архітектуру додатку та технології для її реалізації. Для написання додатку будуть використовуватися наступні технології: мова програмування Dart; framework мови Dart Flutter; Node.js; Nest.js, середовище керування базами даних PostgreSQL.

Наведено детальний розгляд коду деяких компонентів реалізованого додатку. Основну увагу було приділено головному файлу додатку. Було розглянуто принцип роботи дата-провайдерів на прикладі auth провайдера.

Третій розділ демонструє та описує користувацький інтерфейс додатку. Також було проведено комплексне тестування розробленого мобільного додатку. На основі зібраних додатком даних було проведено статистичний аналіз одного випадкового обраного пацієнта.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Натан Адам: JavaScript. Подробное руководство— Символ-Плюс, 2013. — 1205 с.
2. Крис Баккет: Dart в действии, ДМК-Пресс, 2016, 568 с.
3. М. Фаулер, К. Скотт. UML в кратком изложении. Применение стандартного языка объектного моделирования. М: «Мир», 2010, 191 с.
4. П.Козловский, П. Бэкон, Разработка веб-приложений с использованием AngularJS: ДМК, 2013, 394с.
5. Майк Кантелон, Марк Хартер, Натан Райлих, TJ Головайчук. Node.js в действии / Питер, 2015. – 458 с.
6. А.С Асметов. Сахарный диабет 2 типа. Проблемы и решения. Учебное пособие / ГЭОТАР-Медиа, 2014. – 1080 с.
7. Павло Фадєєв. Цукровий діабет / Навчальна книга - Богдан, 2011.— 168с.
8. Разработка веб-приложений с использованием AngularJS: ДМК, 2013, 394с.
9. Стив Круг. Не заставляйте меня думать / Эксмо., 2015. – 216 с.
10. Alessandro B. Flutter for Beginners / Biessek Alessandro., 2019. – 566 с.
11. G. Krishna, The Best Is No Interface: The simple path to brilliant technology: New Riders, 2015, 302 с.
12. Pacini G., Bergman R.N. MINMOD: a computer program to calculate insulin sensitivity and pancreatic responsivity from the frequently sampled intravenous glucose tolerance test. Methods Programs Biomed. 1986; 23 (2): 113–122
13. Ollerton R.L. Application of optimal control theory to diabetes mellitus. Int. J. Control. 1989; 50 (6): 2503–2522. DOI: 10.1080/00207178908953512

14. Fisher M.E. A semiclosed loop algorithm for the control of blood glucose levels in diabetics. *IEEE Transact. Biomed. Engineering*. 1991; 38 (1): 57–61. DOI: 10.1109/10.68209.
15. Roy A., Parker R.S. Dynamic modeling of free fatty acid, glucose, and insulin: an extended «minimal model». *Diabetes Technol. Ther.* 2006; 8 (6): 617–626. DOI: 10.1089/dia.2006.8.617.
16. Roy A., Parker R.S. Dynamic modeling of exercise effects on plasma glucose and insulin levels. *J. Diabet. Sci Technol.* 2007; 1 (3): 338–347.
17. Quon M.J., Cochran C., Taylor S.I., Eastman R.C. Non insulin mediated glucose disappearance in subjects with IDDM. Discordance between experimental results and minimal model analysis. *Diabetes*. 1994; 43: 890–896.
18. Saad M.F., Anderson R.L., Laws A., Laws A., Watanabe R.M., Kades W.W., Chen Y.D., Sands R.E. A comparison between the minimal model and the glucose clamp in the assessment of insulin sensitivity across the spectrum of glucose tolerance. *Insulin Resistance Atherosclerosis Study. Diabetes*. 1994; 43: 1114–1121.
19. Berger M., Rodbard D. Computer simulation of plasma insulin and glucose dynamics after subcutaneous insulin injection. *Diabet Care*. 1989; 12 (10): 725–736. DOI: 10.2337/diacare.12.10.725.
20. Актуальні питання діагностики та лікування цукрового діабету [Електронний ресурс]. Доступ за посиланням: <https://m-1.com.ua/?aid=24>
21. Что такое Dart. Первая программа [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: <https://metanit.com/dart/tutorial/1.1.php>.
22. Что такое Node.js и где он пригодится [Електронний ресурс] / Нетология. – 2018. – Режим доступу до ресурсу: <https://netology.ru/blog/node>.
23. Обзор архитектур управления состоянием на Flutter: [Електроний ресурс]. Доступ за посиланням: <https://dou.ua/lenta/articles/flutter-architecture/>
24. Node.JS: [Електроний ресурс]. Доступ за посиланням: <https://uk.wikipedia.org/wiki/Node.js>



25. Flutter — Cookbook [Электронный ресурс]. – 2019. – Режим доступа до ресурсу: <https://flutter.dev/docs/cookbook>.

26. What is Flutter? Here is everything you should know [Электронный ресурс] / Concise Software. – 2019. – Режим доступа до ресурсу: <https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f>.

27. Flutter: [Электронный ресурс]. Доступ за посиланням: <https://fluttersamples.com/>

28. Nest.js: [Электронный ресурс]. Доступ за посиланням: <https://docs.nestjs.com/>

29. Yablonski J. Laws of UX [Электронный ресурс] / Jon Yablonski. – 2019. – Режим доступа до ресурсу: <https://lawsofux.com/>.

## ДОДАТКИ

```
import 'package:bloc/bloc.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:flutter_secure_storage/flutter_secure_storage.dart';
import 'package:patient/repositories/repositories.dart';
import 'package:patient/screens//UserInfo/InfoScreen.dart';
import 'package:patient/screens/InitialPage.dart';
import 'package:patient/screens/SplashScreen.dart';
import 'package:patient/screens/Auth/MainScreen.dart';
import 'package:http/http.dart' as http;

import 'blocs/blocs.dart';

class SimpleBlocDelegate extends BlocDelegate {
  @override
  void onEvent(Bloc bloc, Object event) {
    super.onEvent(bloc, event);
    print(event);
  }

  @override
  void onTransition(Bloc bloc, Transition transition) {
    super.onTransition(bloc, transition);
    print(transition);
  }

  @override
  void onError(Bloc bloc, Object error, StackTrace stacktrace) {
    super.onError(bloc, error, stacktrace);
    print(error);
  }
}

class LoadingIndicator extends StatelessWidget {
  @override
  Widget build(BuildContext context) => Center(
    child: CircularProgressIndicator(),
  );
}

Future main() async {
  await DotEnv().load('.env');
  BlocSupervisor.delegate = SimpleBlocDelegate();
  final authRepository = AuthRepository(
    authApiClient: AuthApiClient(
      httpClient: http.Client(),
    ),
  );
  runApp(
    BlocProvider<AuthBloc>(
      builder: (context) {
        return AuthBloc(authRepository: authRepository,)
          ..add(AppStarted());
      },
      child: App(authRepository: authRepository),
    ),
  );
}
```

```

    )
  );
}

class App extends StatelessWidget {
  final AuthRepository authRepository;

  App({Key key, @required this.authRepository}) : super(key: key);

  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        fontFamily: 'Circe',
        brightness: Brightness.light,
        primaryColor: Color.fromRGBO(47, 163, 156, 1),
        primaryColorDark: Color.fromRGBO(31, 31, 31, 1),
        primaryColorLight: Color.fromRGBO(31, 31, 31, 0.3),
        backgroundColor: Color.fromRGBO(245, 245, 245, 1),
      ),
      home: BlocBuilder<AuthBloc, AuthState>(
        builder: (context, state) {
          final storage = new FlutterSecureStorage();
          // storage.delete(key: 'token');
          // storage.delete(key: 'id');
          // storage.delete(key: 'filled');
          if (state is AuthenticationUninitialized) {
            return SplashPage();
          }
          if (state is AuthenticationAuthenticated) {
            return state.user.filled ? InitialPage() : InfoScreen();
          }
          if (state is AuthenticationUnauthenticated) {
            return AuthMainScreen(authRepository: authRepository,);
          }
          if (state is AuthenticationLoading) {
            return LoadingIndicator();
          }
          return Center(
            child: Text('Hello 1'),
          );
        },
      ),
    );
  }
}

```

Додаток Б

Лістинг файлу auth\_block.dart

```

import 'package:meta/meta.dart';
import 'package:bloc/bloc.dart';
import 'package:patient/blocs/auth/blocs.dart';

import 'package:patient/models/user.dart';
import 'package:patient/repositories/repositories.dart';

class AuthBloc extends Bloc<AuthEvent, AuthState> {
  final AuthRepository authRepository;

```

```

User user;

AuthBloc({@required this.authRepository})
  : assert(authRepository != null);

@override
AuthState get initialState => AuthenticationUninitialized();

@override
Stream<AuthState> mapEventToState(AuthEvent event) async* {
  if (event is AppStarted) {
    final bool hasToken = await authRepository.hasToken();
    user = await authRepository.getUser();

    yield hasToken ? AuthenticationAuthenticated(user: user) :
AuthenticationUnauthenticated();
  }

  if (event is LoggedIn) {
    yield AuthenticationLoading();
    user = event.user;
    await authRepository.persistInfo(event.user);
    yield AuthenticationAuthenticated(user: event.user);
  }

  if (event is LoggedOut) {
    yield AuthenticationLoading();
    await authRepository.delete();
    yield AuthenticationUnauthenticated();
  }
}
}

```

Додаток В

Лістинг файлу auth\_event.dart

```

import 'package:meta/meta.dart';
import 'package:equatable/equatable.dart';
import 'package:patient/models/user.dart';

abstract class AuthEvent extends Equatable {
  const AuthEvent();

  @override
  List<Object> get props => [];
}

class AppStarted extends AuthEvent {}

class LoggedIn extends AuthEvent {
  final User user;

  const LoggedIn({@required this.user});

  @override

```

```

    List<Object> get props => [user];
  }

class LoggedOut extends AuthEvent {}

```

Додаток Г

Лістинг файлу auth\_state.dart

```

import 'package:meta/meta.dart';
import 'package:equatable/equatable.dart';
import 'package:patient/models/user.dart';

abstract class AuthState extends Equatable {
  const AuthState();

  @override
  List<Object> get props => [];

  User get currentUser => null;

  String get token => null;

  int get patinet_info => null;
}

class AuthenticationUninitialized extends AuthState {}

class AuthenticationAuthenticated extends AuthState {
  final User user;

  const AuthenticationAuthenticated({@required this.user})
    : assert(user != null);

  @override
  List<Object> get props => [user];

  @override
  User get currentUser => user;

  @override
  String get token => currentUser.token;

  @override
  int get patinet_info => currentUser.patient_info;
}

class AuthenticationUnauthenticated extends AuthState {}

class AuthenticationLoading extends AuthState {}

export 'auth_event.dart';
export 'auth_state.dart';
export 'auth_bloc.dart';

```

Додаток Д

Лістинг файлу diary\_block.dart

```

import 'dart:async';

```

```

import 'dart:math';
import 'package:meta/meta.dart';
import 'package:bloc/bloc.dart';
import 'package:patient/blocs/blocs.dart';
import 'package:patient/models/models.dart';
import 'package:patient/repositories/category/repositories.dart';
import 'package:patient/repositories/diary/diary_api_client.dart';
import 'package:patient/repositories/diary/diary_repository.dart';
import './bloc.dart';
import 'package:http/http.dart' as http;

class DiaryBloc extends Bloc<DiaryEvent, DiaryState> {
  AuthBloc authBloc;
  final CategoryRepository _categoryRepository = CategoryRepository(categoryApiClient:
CategoryApiClient(httpClient: http.Client()));
  final DiaryRepository _diaryRepository = DiaryRepository(diaryApiClient:
DiaryApiClient(httpClient: http.Client()));

  DiaryBloc({this.authBloc});

  @override
  DiaryState get initialState => InitialDiaryState();

  @override
  Stream<DiaryState> mapEventToState(
    DiaryEvent event,
  ) async* {
    if (event is FetchDiaryLists) {
      yield* _mapCategoriesToState();
    }
    if (event is CreateDiaryRecordEvent) {
      yield FetchDiaryInfoState();
      try {
        await this._diaryRepository.create(token: authBloc.state.token, record:
event.record);
        yield* _mapCategoriesToState();
      } catch(error) {
        print(error);
        yield DiaryFailure(error: error.toString());
      }
    }
  }

  Stream<DiaryState> _mapCategoriesToState() async* {
    try {
      List<Category> categories = await _categoryRepository.list(authBloc.state.token);
      List<Drugs> drugs = await _categoryRepository.drugs(authBloc.state.token);
      List<Activity> activities = await
_categoryRepository.activities(authBloc.state.token);
      List<Diary> diary = await _diaryRepository.fetch(token: authBloc.state.token);
      yield DiaryListsFetched(categories, drugs, activities, diary);
    } catch(error) {
      yield DiaryFailure(error: error.toString());
    }
  }
}

```

## Лістинг файлу diary\_event.dart

```
import 'package:equatable/equatable.dart';
import 'package:meta/meta.dart';
import 'package:patient/models/diary.dart';

abstract class DiaryEvent extends Equatable {
  const DiaryEvent();

  @override
  List<Object> get props => [];
}

class FetchDiaryLists extends DiaryEvent {}

class CreateDiaryRecordEvent extends DiaryEvent {
  final String record;

  CreateDiaryRecordEvent({@required this.record});
}
```

## Додаток Є

## Лістинг файлу diary\_state.dart

```
import 'package:equatable/equatable.dart';
import 'package:patient/models/models.dart';
import 'package:meta/meta.dart';

abstract class DiaryState extends Equatable {
  final List<Category> categories = [];
  final List<Drugs> drugs = [];
  final List<Activity> activities = [];
  final List<Diary> diaries = [];

  DiaryState([List props = const []]);

  @override
  List<Object> get props => [];
}

class InitialDiaryState extends DiaryState {}

class FetchDiaryInfoState extends DiaryState {}

class DiaryListsFetched extends DiaryState {
  final List<Category> categories;
  final List<Drugs> drugs;
  final List<Activity> activities;
  final List<Diary> diary;

  DiaryListsFetched(this.categories, this.drugs, this.activities, this.diary) :
    super([categories, drugs, activities, diary]);

  @override
  List<Object> get props => [categories, drugs, activities];
}
```



```

List<Category> get categoriesList => categories;

List<Drugs> get drugsList => drugs;

List<Activity> get activitiesList => activities;

List<Diary> get diaryList => diary;
}

class DiaryFailure extends DiaryState {
  final String error;

  DiaryFailure({@required this.error});

  @override
  List<Object> get props => [error];
}

```

Додаток Ж

Лістинг файлу category\_api\_client.dart

```

import 'dart:io';
import 'package:meta/meta.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

import 'package:patient/models/models.dart';

class CategoryApiClient {
  static final baseUrl = DotEnv().env['BASE_URL'];
  final http.Client httpClient;

  CategoryApiClient({
    @required this.httpClient,
  });

  Future<List<Category>> list(
    @required String token,
  ) async {
    final url = '$baseUrl/category';
    List<Category> categoriesList = [];
    try {
      final response = await this.httpClient.get(
        url,
        headers: {
          HttpHeaders.contentTypeHeader: 'application/json',
          'Authorization': 'Bearer $token',
        },
      );
      final categories = json.decode(response.body);
      for (var category in categories) {
        categoriesList.add(Category.fromJson(category));
      }
      return categoriesList;
    } catch (error) {
      throw error;
    }
  }
}

```

```

    }
  }

Future<List<Drugs>> drugs(
  @required String token,
) async {
  final url = '$baseUrl/drugs';
  List<Drugs> drugsList = [];
  try {
    final response = await this.httpClient.get(
      url,
      headers: {
        HttpHeaders.contentTypeHeader: 'application/json',
        'Authorization': 'Bearer $token',
      }
    );
    final drugs = json.decode(response.body);
    for (var category in drugs) {
      drugsList.add(Drugs.fromJson(category));
    }
    return drugsList;
  } catch(error) {
    throw error;
  }
}

Future<List<Activity>> activities(
  @required String token,
) async {
  final url = '$baseUrl/activity';
  List<Activity> activitiesList = [];
  try {
    final response = await this.httpClient.get(
      url,
      headers: {
        HttpHeaders.contentTypeHeader: 'application/json',
        'Authorization': 'Bearer $token',
      }
    );
    final activities = json.decode(response.body);
    for (var category in activities) {
      activitiesList.add(Activity.fromJson(category));
    }
    return activitiesList;
  } catch(error) {
    throw error;
  }
}
}

```

Додаток 3

Лістинг файлу category\_repository.dart

```

import 'package:meta/meta.dart';
import 'package:patient/models/models.dart';
import 'package:patient/repositories/category/category_api_client.dart';

class CategoryRepository {
  final CategoryApiClient categoryApiClient;

```

```

CategoryRepository({
  @required this.categoryApiClient,
});

Future<List<Category>> list(
  String token,
) async {
  return await this.categoryApiClient.list(token);
}

Future<List<Drugs>> drugs(
  String token,
) async {
  return await this.categoryApiClient.drugs(token);
}

Future<List<Activity>> activities(
  String token,
) async {
  return await this.categoryApiClient.activities(token);
}
}

```

Додаток И

Лістинг файлу diary\_api\_client.dart

```

import 'dart:io';
import 'package:meta/meta.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:http/http.dart' as http;
import 'package:patient/helpers/functions.dart';
import 'dart:convert';

import 'package:patient/models/models.dart';

class DiaryApiClient {
  static final baseUrl = DotEnv().env['BASE_URL'];
  final http.Client httpClient;

  DiaryApiClient({@required this.httpClient});

  Future<List<Diary>> fetchDiaryRecords({@required String token, params = const[]})
  async {
    String url = '$baseUrl/diary';
    List<Diary> diaryList = [];
    try {
      if (params.length > 0) {
        url += '?${generateParamsString(params)}';
      }
      final response = await this.httpClient.get(
        url,
        headers: {
          HttpHeaders.contentTypeHeader: 'application/json',
          'Authorization': 'Bearer $token',
        }
      );
      final diaries = json.decode(response.body);
      for (var diary in diaries) {

```

```

        diaryList.add(Diary.fromJson(diary));
    }
    return diaryList;
} catch (error) {
    throw error;
}
}

Future<void> create({@required String token, @required String record}) async {
    final url = '$baseUrl/diary';
    try {
        final response = await this.httpClient.post(
            url,
            headers: {
                HttpHeaders.contentTypeHeader: 'application/json',
                'Authorization': 'Bearer $token',
            },
            body: record,
        );
    } catch(error) {
        throw error;
    }
}
}

```