

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

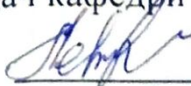
ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної магістерської роботи
на тему: «Веб-платформа «Система контролю пацієнтів
з цукровим діабетом»»

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

 Н.В. Штоюнда

Керівник старший викладач кафедри ІПЗ, к.т.н.

 А.Ю. Левченко

Рецензент доцент кафедри ІПЗ, к.т.н., доцент

 Ю.М. Єфремов

Житомир – 2019

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. Пулко
« 04 » 20 19 р.

ЗАВДАННЯ

на кваліфікаційну магістерську роботу

Студента

Штоюнди Назара Вікторовича

Тема роботи:

Веб-платформа «Система контролю пацієнтів з цукровим діабетом»


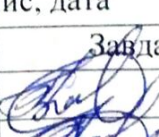


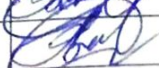

Затверджена Наказом університету від «04» вересня 2019 р. № 206с

Термін здачі студентом закінченої роботи 01.12.2019

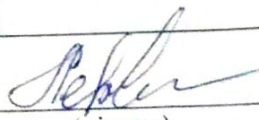
Вихідні дані роботи (зазначається предмет і об'єкт дослідження)

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

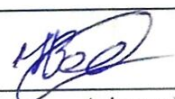
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	В.Л. Левківський	 01.09.19р.	 30.08.19р.
2	В.Л. Левківський	 30.09.19р.	 08.11.19р.
3	В.Л. Левківський	 08.11.19р.	 15.11.19р.

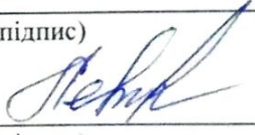
Керівник


(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

Випускна кваліфікаційна магістерська робота складається з платформи «Система контролю пацієнтів з цукровим діабетом» та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить в собі 76 сторінок, 19 ілюстрацій та 8 таблиць.

Метою проекту є розробка платформи, що містить модулі для відслідковування показників рівня цукру в щоденнику пацієнта, перегляд статистичних даних пацієнта, можливість прямого спілкування у вигляді чату, та модуль створення нотаток і нагадувань.

В пояснювальній записці вказано основні завдання на розробку додатку, здійснено аналіз існуючих на ринку додатків. Обґрунтовано вибір архітектурного шаблону для реалізації системи – Flutter BLoC. Наведено сценарії роботи програмного комплексу, його загальну структуру, описано базу даних системи — СКБД PostgreSQL, проведено опис алгоритмів взаємодії окремих класів системи, описано об'єктну структуру додатку, алгоритми роботи основних модулів, реалізованих з використанням мови програмування Dart.

КЛЮЧОВІ СЛОВА: ЦУКРОВИЙ ДІАБЕТ, ЕНДОКРИНОЛОГІЯ, FLUTTER, КОМПЕНСАЦІЯ ДІАБЕТУ

ABSTRACT

The diploma thesis consists of the Diabetes Patient Control System platform and an explanatory note. The explanatory note for the thesis contains 76 pages, 19 illustrations and 8 tables.

The aim of the project is to develop a web-based platform containing modules for tracking sugar levels in a patient's diary, viewing patient statistics, chatting directly, and a module for taking notes and reminders.

The explanatory note outlines the main tasks for the application development, analyzes the existing applications in the market. The choice of the architectural template for implementation of the system - Flutter BLoC is substantiated. The scenarios of the work of the software complex, its general structure, the system database - PostgreSQL DBMS are described, the algorithms of interaction of individual classes of the system are described, the object structure of the application, the algorithms of the main modules implemented using the Dart programming language are described.

KEYWORDS: SUGAR DIABETES, FLUTTER, ENDOCRINOLOGY, DIABETES COMPENSATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ДОДАТКУ «СИСТЕМА КОНТРОЛЮ ПАЦІЄНТІВ З ЦУКРОВИМ ДІАБЕТОМ»	10
1.1 Актуальність теми дослідження	10
1.2 Методи оцінки стану пацієнта.....	14
1.3 Аналіз та дослідження аналогів програмного продукту	18
1.4 Постановка задачі	22
Висновки до першого розділу	23
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПЛАТФОРМИ «СИСТЕМА КОНТРОЛЮ ПАЦІЄНТІВ З ЦУКРОВИМ ДІАБЕТОМ»	24
2.1 Визначення варіантів використання та вимог до платформи.....	24
2.2 Архітектура та узагальнена структуру, програмного комплексу.	26
2.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	31
2.4 Розробка бази даних системи	36
2.5 Реалізація функціональних частин додатку	40
Висновки до другого розділу	46
РОЗДІЛ 3. ДОСЛІДЖЕННЯ ПРАЦЕЗДАТНОСТІ ТА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ДОДАТКУ	47
3.1 Структура інтерфейсу. Інтерфейс та порядок роботи з додатком	47
3.2 Тестування роботи розробленого додатку	51
3.3 Аналіз та моделювання даних на базі записів додатку	54
Висновки до третього розділу	57
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	59
ДОДАТКИ.....	62

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. Тема кваліфікаційної роботи — розробка платформи «Система контролю пацієнтів з цукровим діабетом».

Поширеність цукрового діабету в популяції, у середньому, становить 1-8,6%, а серед дітей та підлітків — близько 0,1-0,3%. З урахуванням не діагностованих випадків, у деяких країнах поширеність може сягати 6%. За даними IDF, у світі мешкає до 183 млн осіб із не діагностованим цукровим діабетом, що становить 50% від діагностованих випадків. 2011 року діабет став причиною 4,6 млн смертей.

Використовуючи додаток, лікар має змогу контролювати стан пацієнтів, слідкувати за їх записами та тенденціями зміни глюкози в крові, як результат миттєво надавати поради для хворих без необхідності очної зустрічі.

Для нормального функціонування, людям хворим на діабет необхідний постійний контроль та компенсація хвороби. Додаток суттєво спрощує цей процес, пацієнт має змогу вільного спілкування з лікуючим лікарем, отримувати рекомендації та різного роду поради, а інша сторона — лікар, має постійний доступ до показників пацієнта.

Метою випускної роботи є проектування архітектури, розробка алгоритмів роботи та реалізація додатку «Система контролю пацієнтів з цукровим діабетом»

Завдання роботи. Встановлена мета обумовлює наступні завдання:

- проведення аналізу ринку;
- вибір та обґрунтування засобів та інструментів реалізації додатку;
- проектування складових та алгоритмів роботи системи;
- реалізація додатку

Об'єктом дослідження є процес компенсації цукрового діабету.

Предметом дослідження є використання інформаційних технологій 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. Левківський В.Л., Штоюнда Н.В., Левченко А.Ю. Математичне моделювання систем регуляції глікемії пацієнтів з цукровим діабетом. Тези всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки. – Житомир : ЖДТУ, 2019. – с. 87

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ДОДАТКУ «СИСТЕМА КОНТРОЛЮ ПАЦІЄНТІВ З ЦУКРОВИМ ДІАБЕТОМ»

1.1 Актуальність теми дослідження

Цукровий діабет — група ендокринних захворювань, що розвиваються внаслідок абсолютної чи відносної недостатності гормону інсуліну, внаслідок чого виникає стійке підвищення рівня глюкози в крові — гіперглікемія. Захворювання характеризується хронічним перебігом і порушенням усіх видів обміну речовин: вуглеводного, жирового, білкового, мінерального і водно-сольового. Характерними симптомами є невгамовна спрага (полідипсія) та надмірне сечовиділення (поліурія), однак ці симптоми можуть бути слабо вираженими, якщо рівень глюкози в крові підвищений помірно.

Поширеність цукрового діабету у світі станом на 2000 рік (на 1,000 осіб). Середній світовий показник 2,8 %. За даними Міжнародної федерації діабету 2011 року кількість хворих на цукровий діабет у світі досягла рекордної цифри — 366 мільйонів, а в 2030 році становитиме 552 мільйони.

Поширеність цукрового діабету в популяції, у середньому, становить 1-8,6%, а серед дітей та підлітків — близько 0,1-0,3%. З урахуванням не діагностованих випадків, у деяких країнах поширеність може сягати 6%. За даними IDF, у світі мешкає до 183 млн осіб із не діагностованим цукровим діабетом, що становить 50% від діагностованих випадків. 2011 року діабет став причиною 4,6 млн смертей.

Слід зазначити неоднорідність захворюваності на цукровий діабет залежно від раси. Цукровий діабет 2-го типу найпоширеніший серед монголоїдів; так, у Великобританії серед осіб монголоїдної раси старше 40 років 20% страждають на цукровий діабет 2-го типу, на другому місці перебувають люди негроїдної раси: серед осіб, старших 40 років, частка хворих на цукровий діабет становить 17%.

Також неоднорідна частота ускладнень. Належність до монголоїдної раси підвищує ризик розвитку діабетичної нефропатії та ішемічної хвороби серця, але знижує ризик виникнення синдрому діабетичної стопи. Для осіб негроїдної раси характерний розвиток гестаційного цукрового діабету. За даними Міжнародної федерації діабету 2011 року найбільшу кількість хворих на цукровий діабет зареєстровано в Китаї — 90 млн, Індії — 63,1 млн та США — 23,7 млн.

Існує ряд класифікацій цукрового діабету за різними ознаками. У сукупності вони входять в структуру діагнозу і дозволяють досить точно описати стан хворого на діабет.

Таблиця 1.1

Порівняння діабету першого та другого типу

Ознака	Діабет 1-го типу	Діабет 2-го типу
Початок	раптовий	поступовий
Вік, коли починається	переважно у дітей	переважно у дорослих
Статура	худа або нормальна	часто ожиріння
Кетоацидоз	поширений	не часто
Аутоантитіла	зазвичай присутні	відсутні
Рівень інсуліну	низький	нормальний, зменшений або підвищений
Конкордантність ідентичних близнюків	у 50%	90%
Частка хворих	~10%	~90%

Цукровий діабет 1-го типу

В основі патогенетичного механізму розвитку діабету 1-го типу лежить недостатність синтезу і секреції інсуліну в-клітинами підшлункової залози, внаслідок їх відмирання під впливом вірусної інфекції, стресів, аутоімунної агресії тощо). Поширеність цукрового діабету 1-го типу в популяції становить від 10 до 15 % всіх випадків цукрового діабету. Найчастіше це захворювання маніфестується в дитячому або підлітковому віці з швидким розвитком ускладнень на тлі декомпенсації вуглеводного обміну. Основним методом лікування є ін'єкції інсуліну. За відсутності лікування діабет 1-го типу швидко прогресує та призводить до виникнення важких ускладнень, таких як кетоацидоз та діабетична кома.[7]

Цукровий діабет 2-го типу

В основі патогенезу цього типу захворювання лежить зниження чутливості інсулінозалежних тканин до дії інсуліну (інсулінорезистентність). На початковій стадії хвороби інсулін синтезується в звичайних або навіть підвищених кількостях. Дотримання дієти та зниження маси тіла на початкових стадіях захворювання допомагають нормалізувати вуглеводний обмін, відновити чутливість тканин до дії інсуліну і знизити синтез глюкози на рівні печінки. Проте в ході прогресування захворювання біосинтез інсуліну в-клітинами підшлункової залози знижується, що зумовлює необхідність призначення замісної гормональної терапії препаратами інсуліну.

Діабет 2-го типу становить від 85% до 90% усіх випадків цукрового діабету у дорослого населення і найчастіше маніфестується серед осіб старше 40 років, як правило виникає на тлі ожиріння. Захворювання розвивається повільно, перебіг легкий. У клінічній картині переважають супутні симптоми; кетоацидоз розвивається рідко. Стійка гіперглікемія з роками приводить до розвитку мікро- та макроангіопатії, нефро- і нейропатії, ретинопатії та інших ускладнень.

Класифікація за важкістю перебігу захворювання

Легкий перебіг

Легку (І ступінь) форму хвороби характеризує не високий рівень глікемії, яка не перевищує 8 ммоль/л натще, коли немає значних коливань вмісту цукру в крові протягом доби, незначна добова глюкозурія (від слідів до 20 г/л). Стан компенсації підтримується за допомогою дієтотерапії. При легкій формі діабету у хворого можуть діагностуватися ангіонейропатії доклінічної і функціональної стадії.

Середньої ступені тяжкості

Для середньої (ІІ ступінь) тяжкості цукрового діабету, як правило, характерна глікемія натще не вище 14 ммоль/л, коливання глікемії протягом доби, добова глюкозурія не вище 40 г/л, можливий епізодичний розвиток кетозу або кетоацидозу. Компенсація діабету досягається дієтою та прийомом цукрознижувальних пероральних засобів або введенням інсуліну (в разі розвитку вторинної сульфамідорезистентності) у дозах, які не перевищують 40 ОД на добу. У цих хворих можуть виявлятися діабетичні ангіонейропатії різної локалізації та функціональної стадії.

Важкий перебіг

Важка (ІІІ ступінь) форма діабету характеризується високими рівнями глікемії (понад 14 ммоль/л), значними коливаннями вмісту цукру в крові протягом доби, високим рівнем глюкозурії (понад 40-50 г/л) та різними діабетичними ангіонейропатіями. Хворі потребують постійної інсулінотерапії в дозах 60 ОД і більше.

На сьогоднішній день вважається доведеною генетична схильність до цукрового діабету. Вперше подібну гіпотезу висловили 1896 року, на той час вона підтверджувалася тільки результатами статистичних спостережень. 1974 року J. Nerup і співавтори, A. G. Gudworth та J. C. Woodrow, виявили зв'язок В-локусу лейкоцитарних антигенів гістосумісності і цукрового діабету 1-го типу та відсутність їх в осіб із діабетом 2-го типу.

Згодом було виявлено ряд генетичних варіацій, що значно частіше зустрічаються в геномі хворих на діабет, ніж в іншій популяції. Так, приміром, наявність у геномі одночасно B8 і B15 збільшувало ризик захворювання приблизно в 10 разів. Наявність маркерів Dw3/DRw4 збільшує ризик захворювання в 9,4 рази. Близько 1,5 % випадків діабету пов'язані з мутацією A3243G мітохондріального гена MT-TL1.

Однак слід зазначити, що при діабеті 1-го типу спостерігається генетична гетерогенність, тобто захворювання може викликатися різними групами генів. Лабораторно-діагностичною ознакою, що дозволяє визначити 1-й тип діабету, є виявлення в крові антитіл до β -клітин підшлункової залози. Характер успадкування наразі не зовсім зрозумілий, складність прогнозування спадкування пов'язана з генетичною гетерогенністю цукрового діабету, побудова адекватної моделі успадкування вимагає додаткових статистичних та генетичних досліджень.

Прогноз при всіх типах цукрового діабету умовно сприятливий, за умови адекватно проведеного лікування і дотриманні режиму харчування зберігається працездатність. Прогресування ускладнень значно сповільнюється або повністю припиняється. Однак слід зазначити, що в більшості випадків в результаті лікування причина захворювання не усувається, і терапія носить лише симптоматичний характер.

1.2 Методи оцінки стану пацієнта

Для розробки платформи віддаленої консультації та компенсації глюкози в крові хворого цукровим діабетом, потрібно ввести поняття показників та методи їх збору, що аналізуватимуться та узагальнюватимуть інформацію користувача і для лікаря відповідно. Даних значення, лікар буде використовувати для аналізу даних та статистики пацієнта.

Список введених показників:

Time in range(TIR) — час в межах цільових показників. За даними American Diabetes Association, допустимі межі глюкози в крові — від 3,8 ммоль/л до 10 ммоль/л. Час, проведений в діапазоні цих показників, в ідеалі має бути більше 50%. Час, проведений з рівнем глюкози 3,8 ммоль/л і нижче - не більше 5%.

SD — дисперсія значень глюкози, відхилення від середнього значення в межах однієї доби. Стандартне відхилення рахується відштовхуючись від середнього значення ЦК (mean glucose) — дисперсія показника SD має бути менше 1/3 значення mean glucose. Для більш точного значення SD на показник рівня ЦК вводиться коефіцієнт відхилення (CV). Даний показник вираховується по формулі (SD * середнє значення ЦК) / 100. Поріг нормального CV знаходиться в межах 36%. При CV> 36%, потрібно переглянути процес компенсації хвороби.

Окрім цього показник стандартного відхилення і його коефіцієнт залежать ще й від віку. За дослідженнями, у дітей від 2 до 6 років значення CV може несуттєво виходити за межі до 40%.

Glucose Variability (GV) — показник вказує наскільки дані ГК відхиляються від середнього значення глюкози, як часто показники знаходяться в діапазоні гіперглікемії або гіпоглікемії.

$$S = \sqrt{\frac{\sum_{i=1}^n (x_i - x_{\text{ср}})^2}{n - 1}}$$

Рис. 1.1 – Формула для обчислення стандартного відхилення

1. Знаходимо середнє арифметичне вибірки.
2. Від кожного значення вибірки віднімаємо середнє арифметичне.
3. Кожну отриману різницю зводимо в квадрат.
4. Підсумовуємо отримані значення квадратів різниць.
5. Ділимо на розмір вибірки мінус 1.
6. Знаходимо квадратний корінь.

GVI (Glycemic Variability Index) — індекс вказує співвідношення довжини цукрової кривої за певний проміжок часу до довжини ідеальної цукрової кривої за такий же самий відрізок. Співвідношення цих довжин буде позначати індекс глікемічний варіабельності;

PGS (Patient Glycemic Status) — глікемічний статус хворого на цукровий діабет. Повний показник рівня компенсації, що враховує GVI, середній рівень цукру і відсоток часу в цільових показниках. Даний показник обчислюється за формулою $(GVI * \text{середній ЦК} * (1\% \text{ TIR}))$;

Середнє значення ГК з N днів;

Відхилення за N днів — відхилення показує, на скільки сильно рівні ГК відхилялися від середнього значення за останні N днів. Максимальне допустиме відхилення обраховується за формулою: $(\text{Максимальний рівень ГК} - \text{Мінімальний рівень ГК}) / 2$.

Таблиця 1.2

Норми цукру в крові

HbA1c, %	Середній цукор, ммоль/л	HbA1c, %	Середній цукор, ммоль/л
4	3,8	10	13,4
4,5	4,6	10,5	14,2
5	5,4	11	14,9
5,5	6,2	11,5	15,7
6	7	12	16,5
6,5	7,8	12,5	17,3
7	8,6	13	18,1

Продовження таблиці 1.2

7,5	9,4	13,5	18,9
8	10,2	14	19,7
8,5	11	14,5	20,5
9	11,8	15	21,3
9,5	12,6	15,5	22,1

Таблиця 1.3

Компенсація хвороби

Показники	Компенсація	Субкомпенсація	Декомпенсація
Цукор в крові натщесерце	3,9–6,2	6,3–7,8	>7.8
Цукор в крові після їжі	5,5–8,0	8,1–10,0	>10
Цукор в сеча(%)	0	<0,5	>0,5
Глікований гемоглобін (%) Норма до 6,0%	<6,2	6,5–7,5	>7,5
Загальний холестерин	<5,2	5,2–6,5	>6,5
Тригліцериди	<1,7	1,7–2,2	>2,2
Індекс маси тіла у чоловіків	<25	25–27	>27
Індекс маси тіла у жінок	<24	24–26	>27
Артеріальний тиск	<140/85	<160/95	>160/95

1.3 Аналіз та дослідження аналогів програмного продукту

В наш час ринок програмних засобів спрямованих на так звану - «телемедицину» або ж консультацію лікаря через інтернет, надає широкий вибір, але разом с тим, систем, які були б спрямовані саме на людей з цукровим діабетом на жаль немає. Тому для людей з діабетом та лікарів ендокринологів це може стати унікальним інструментом для покращення прогнозу компенсації цукрового діабету.

Щоб зрозуміти, якими можливостями, має володіти розроблюваний програмний продукт, потрібно провести порівняльний аналіз представлених на ринку аналогів. Але слід зауважити, що повноцінного аналога не має, а обрані системи серед спільного з розроблюваним додатком мають головну функцію — консультація лікаря. Для цього за критеріями популярності та позитивного рейтингу було обрано додатки наведені в таблиці 1.1.

Опишемо кожен з обраних додатків.

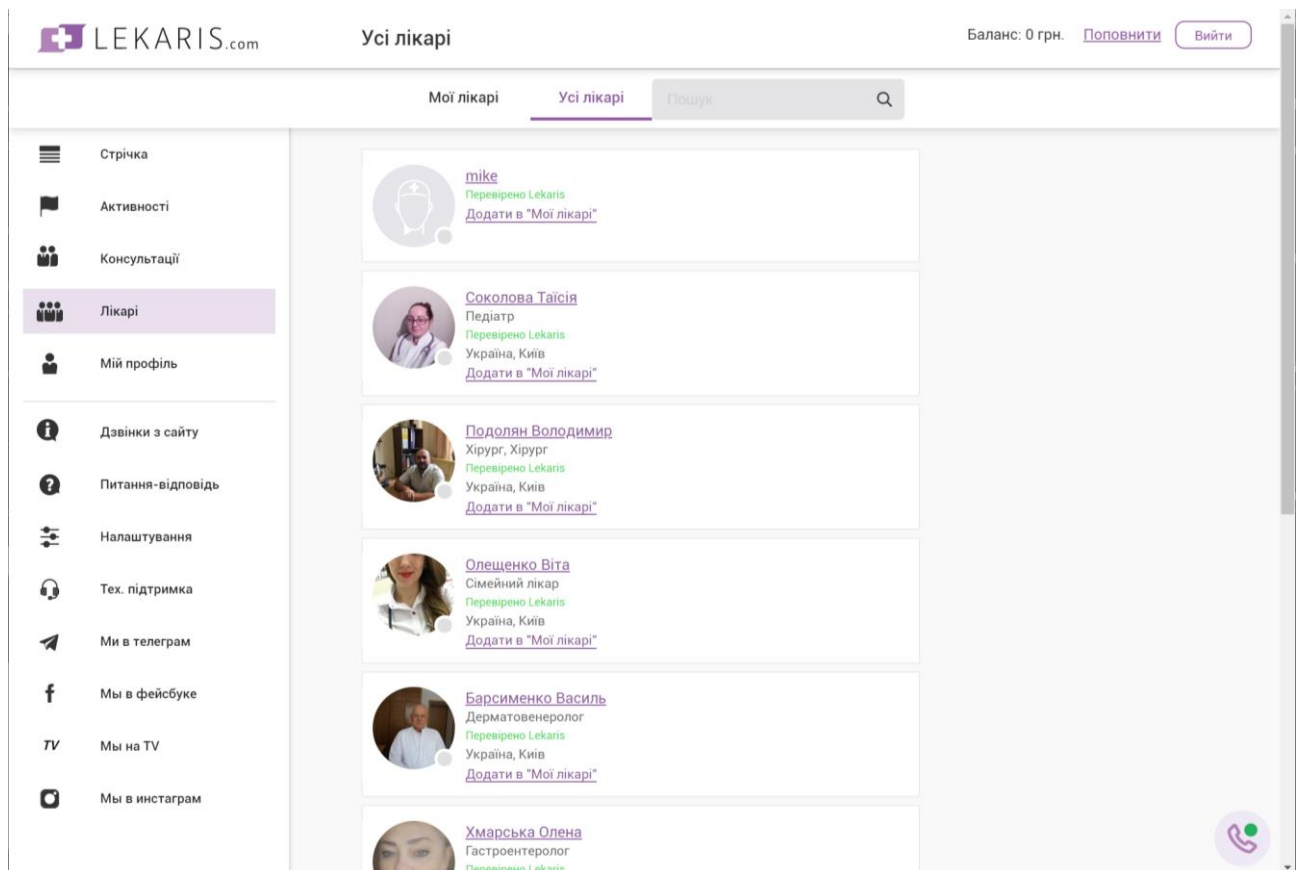


Рис.1.2 – Екран веб-кабінету «Lekaris»

Хто консультує. Тільки зареєстровані лікарі, які мають медичну практику в лікувальні заклади України. У базі сайту більше 150 лікарів. Їх список можна подивитися в окремому розділі

Як швидко відповідають. Від питання до відповіді може пройти до одних-двох діб.

Особливості. Рейтинги громадської думки з тих чи інших питань.

Недоліки. Деякі розділи працюють нестабільно. Наприклад, в даний момент неможливо подивитися додаткову інформацію про лікарів. Також лікарі не мають повної інформації про стан пацієнта, що особливо важливо для людей з діабетом.

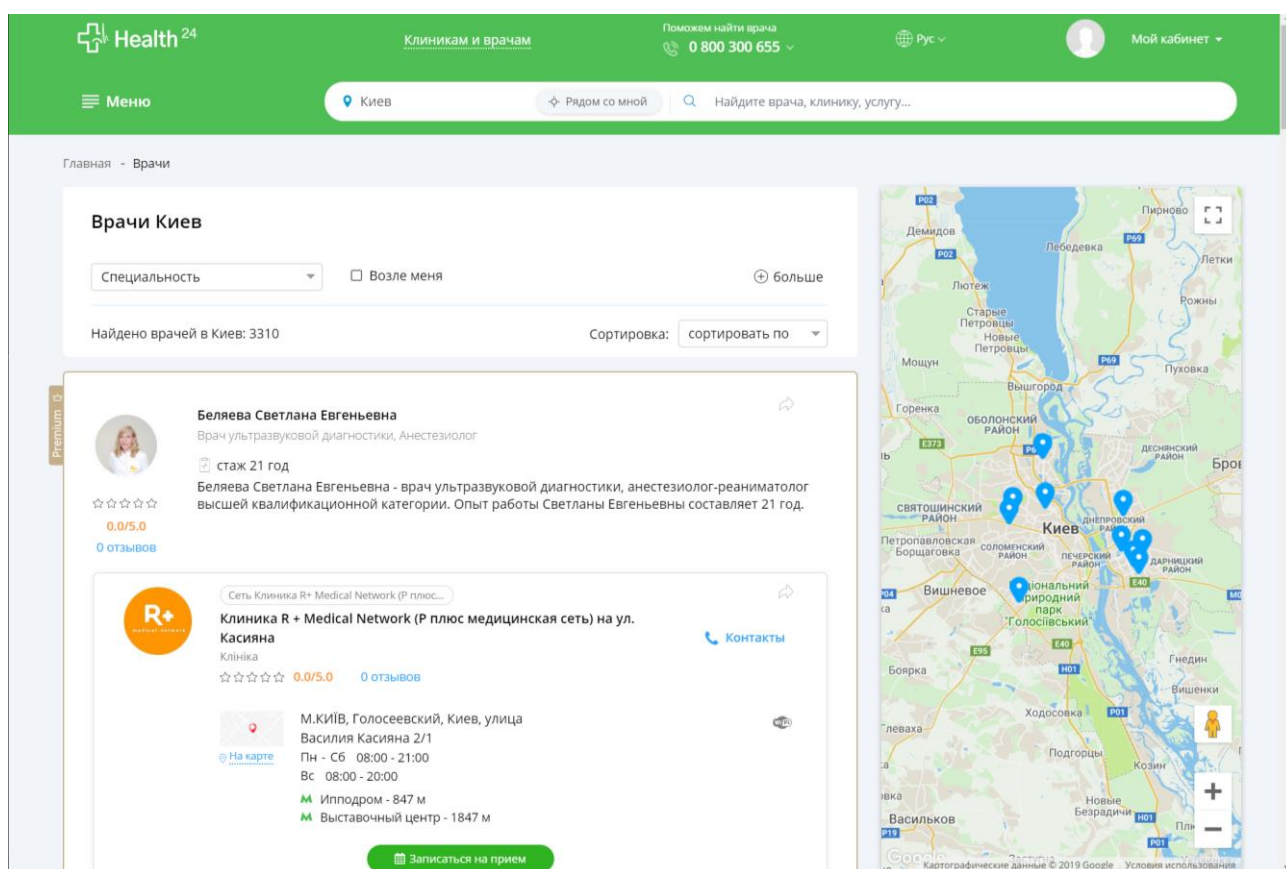


Рис.1.3 – Экран веб-кабинету «Health24»

Хто консультує. На цьому сайті консультації надають провідні лікарі України.

Як швидко відповідають. Протягом доби. Іноді до двох.

Особливості. До свого питання можна прикріпити історію хвороби, яку можна вести прямо на цьому сайті.

Недоліки. Для пацієнтів з діабетом не передбачено функції щоденника самоконтролю, що погано для об'єктивної оцінки стану пацієнта.

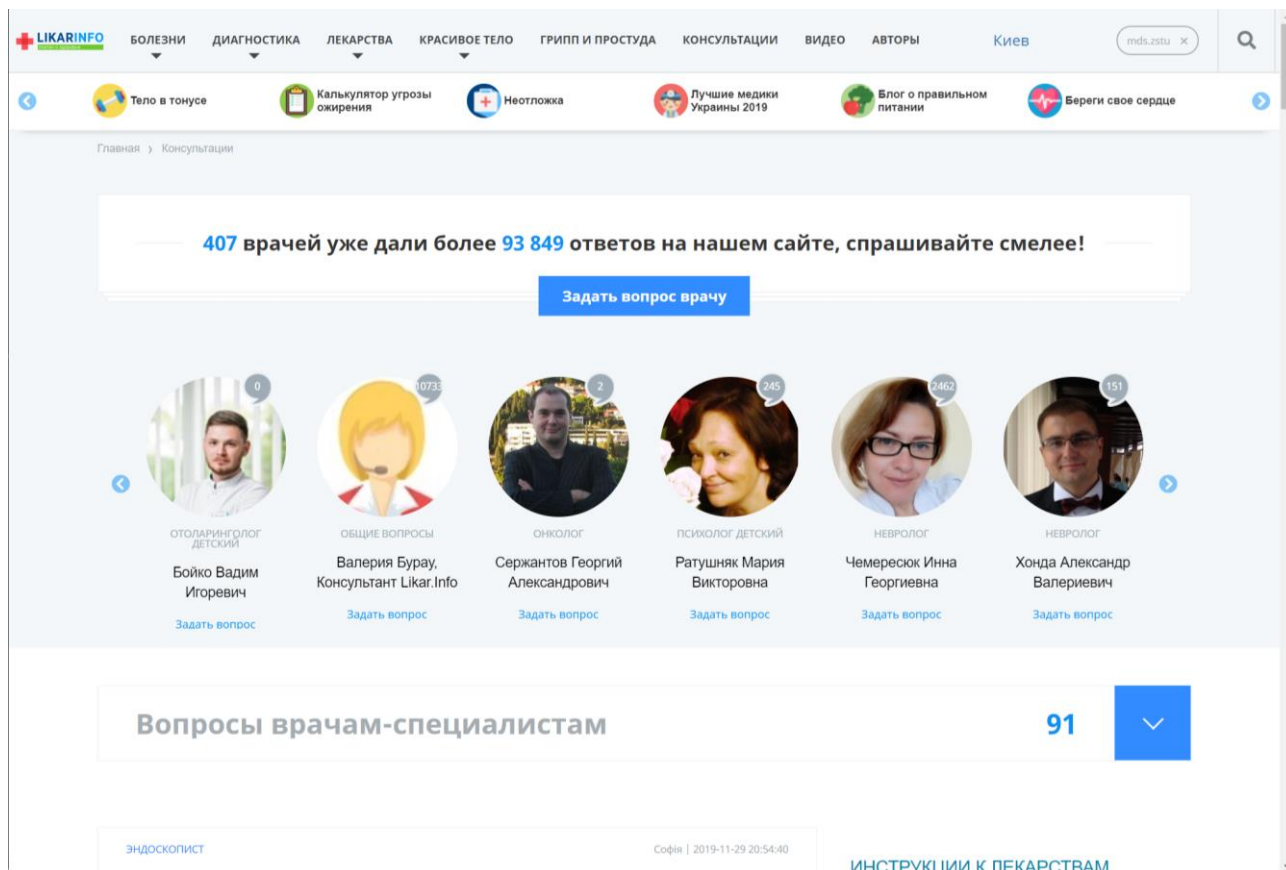


Рис.1.4 – Экран веб-кабинету «Likar.Info»

Хто консультує. На цьому сайті на питання можуть відповісти не тільки лікарі, а й інші відвідувачі сайту. Однак при цьому, прізвище зареєстрованого лікаря відображається червоним кольором і є посиланням на його картку.

Як швидко відповідають. Протягом доби. Іноді до двох.

Особливості. Форум для всіх, форум для лікарів. Спеціальні тематичні розділи, наприклад, "Здоровий спосіб життя". До свого питання можна прикріпити історію хвороби, яку можна вести прямо на цьому сайті.

Недоліки. Щоб задати питання лікарю необхідна реєстрація на сайті. Відсутня можливість поділитися щоденником самоконтролю для людей з діабетом.

Таблиця 1.4

Перелік аналогів платформи «Панель керування для лікарів ендокринологів по компенсації діабету у пацієнтів»

Назва	Розробник	Платформа	Умови ліцензії
Health24	Health24.Life	Web	Умовно безкоштовний
Likar.Info	SIA Creative Media Invest	Web	Безкоштовний
Lekaris	Sergey Cherskoi	Android, iOS, Web	Умовно безкоштовний

Таблиця 1.5

Функціональні можливості

Додаток	Безкоштовна консультація	Фільтр лікарів	Щоденник хвороби	Планування консультації
Health24	Так	Так	Ні	Так
Likar.Info	Ні	Так	Ні	Так
Lekaris	Так	Так	Ні	Так

Можна зробити висновок що всі платформи схожі по функціональності та орієнтовані більш на масовий ринок, як результат в додатках слабо реалізований контроль та компенсація хвороби пацієнтів саме з цукровим діабетом. Майбутній додаток окрім того що буде реалізований на десктоп також має підтримувати основний функціонал описаних платформ.

Головна перевага додатку, повний контроль лікарем процесу контролю та компенсації цукрового діабету його пацієнтів, як результат миттєва реакція на будь які відхилення поведінки глюкози в крові.

1.4 Постановка задачі

Головна мета роботи створення платформи «Система контролю пацієнтів з цукровим діабетом» — спрощення контролю та компенсації хвороби цукровий діабет шляхом постійного зв'язку між хворим та лікарем. Лікар постійно отримує останні актуальні заміри у щоденнику самоконтролю пацієнта. Бачить тенденції зміни рівня глюкози в крові, та таким чином знаходить певні закономірності поведінки глюкози. Дві сторони мають змогу прямої комунікації у вигляді чату. Так пацієнт може швидко отримувати відповіді на свої запитання, а лікар надавати рекомендації та поради. Сукупність такого функціоналу породжує інструмент для дистанційного контролю та компенсації хвороби.

Для реалізації поставленого завдання, основними етапами є:

1. Розробити дизайн платформи додатку.

1.1. Проектування користувацького інтерфейсу додатку. Розробка дизайн концепції головної та внутрішніх сторінок додатку. Створення бібліотеки компонентів дизайн-системи.

1.2. Створити шаблони екранів додатку.

2. Реалізувати ядро системи управління контентом.

2.1. Написати програмні модулі для:

- авторизації та автентифікація користувача в системі;
- модуль для перегляду щоденника та аналітики пацієнта;
- розробка спеціалізованого чату та належних до нього функцій;
- модуль налаштувань профілю користувача;
- модуль для ведення нотаток з нагадуваннями;
- модуль для оцінки перебігу хвороби у пацієнта;

- 2.2. Реалізувати CRUD операції для обробки запитів до БД додатку.
3. Реалізувати структуру збереження даних наступного виду:
 - 3.1. Дані про записи в щоденник самоконтролю пацієнтів (дата, час, ліки, активність, їжа тощо);
 - 3.2. Дані про налаштування користувачів додатку;
 - 3.3. Дані для авторизації користувачів (електронна пошта та пароль).
4. Створити демонстраційний та тестовий контент для додатку.

Результатом реалізації поставленого завдання є додаток що спростить контролю та компенсації хвороби цукровий діабет шляхом постійного зв'язку між хворим та лікарем. Використовуючи додаток, лікар має змогу контролювати стан пацієнтів, слідкувати за їх записами та тенденціями зміни глюкози в крові, як результат миттєво надавати поради для хворих без необхідності очної зустрічі.

Висновки до першого розділу

В даному розділі кваліфікаційної магістерської роботи було проведено визначено актуальність обраної теми, проведено порівняльний аналіз існуючих на ринку аналогів, додатки для дистанційної консультації лікаря, та визначено їх основні переваги та недоліки. На основі цього аналізу було поставлене завдання на розробку програмного продукту.

Були визначені основні можливості додатку, які мають бути реалізовані: спеціалізований чат між пацієнтом та лікарем, модулі для нотатко лікаря з вбудованою системою сповіщень, модуль для перегляду записів зі щоденнику самоконтролю пацієнта та модуль перегляду аналітики і статистика терапії пацієнта. Також було визначено, що додаток має працювати на операційних системах Android та iOS.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПЛАТФОРМИ «СИСТЕМА КОНТРОЛЮ ПАЦІЄНТІВ З ЦУКРОВИМ ДІАБЕТОМ»

2.1 Визначення варіантів використання та вимог до платформи

Розробка платформи «Система контролю пацієнтів з цукровим діабетом» ставить за мету спрощення контролю за перебігом хвороби людям з цукровим діабетом. Використовуючи додаток, лікар має змогу контролювати стан пацієнтів, слідкувати за їх записами в щоденник самоконтролю та тенденціями зміни глюкози в крові, як результат, це створює можливості для лікарів створення плану лікуванню, який буде максимально скоригований під конкретного пацієнта без необхідності очної зустрічі.

Вимоги користувачів

Зовнішні користувачі мають доступ до наступних функцій:

1. Перегляд даних про стан пацієнтів;
2. Отримання доступу до щоденників самоконтролю пацієнтів та їх аналітики;
3. Синхронізація даних між пристроями;
4. Реєстрація в системі;
5. Можливість ведення нотатків;
6. Можливість проведення консультацій з пацієнтами.

Внутрішні користувачі – адміністратори мають таких функцій, як:

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

Характеристика об'єкта комп'ютеризації:

Функціональні вимоги

Аутентифікація користувачів в системі: В системі має бути представлена можливість реєстрації користувача;

Можливість збереження інформації: Система повинна зберігати інформацію і надавати можливість користувачу керувати нею;

Ведення нотатків: Внесення та редагування нотатків лікаря;

Перегляд аналітики: Система повинна надавати можливість перегляду аналітики та статистики пацієнтів, що записані до лікаря;

Консультація пацієнтів: Система повинна забезпечити можливість проведення консультації записаних до лікаря пацієнтів.

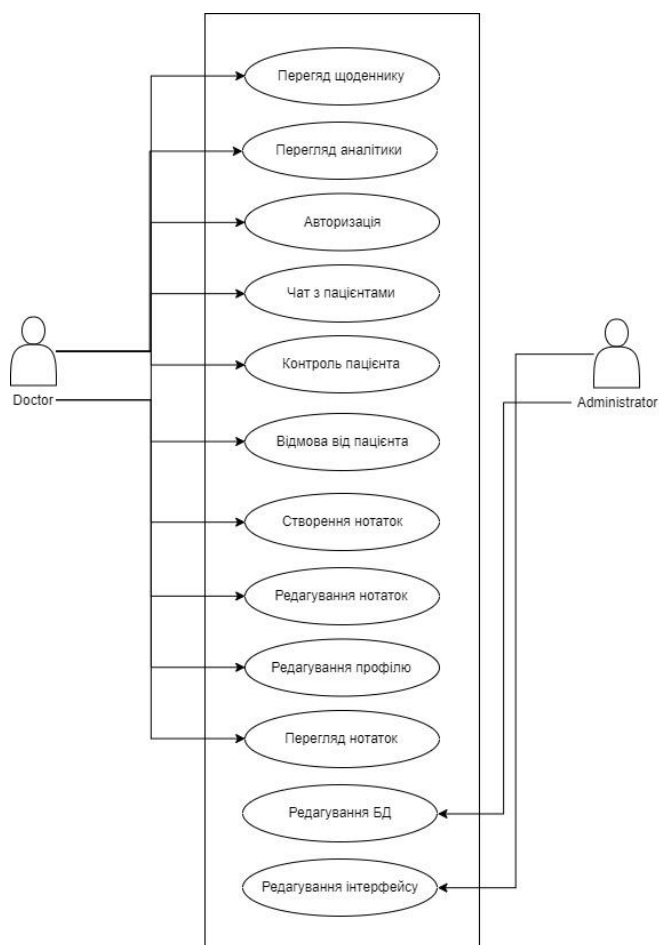


Рис. 2.1 – Варіанти використання розроблюваного додатку

Нефункціональні вимоги

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;

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.

Продуктивність

Система повинна підтримувати мінімум 500 одночасно працюючих користувачів.

1. 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 пристрою:

- 1024Mб RAM;
- 125Mб ROM;
- Процесор з тактовою частотою 2,2 GHz;
- Операційна система Windows 10, MacOS 10.10, Ubuntu 14.04+;

2.2 Архітектура та узагальнена структура, програмного комплексу.

Для реалізації кваліфікаційної роботи ми будемо використовувати наступний стек технологій. Для написання Frontend частини обрано мову

програмування Dart та її framework Flutter. Backend буде реалізовано на мові Node.js та framework Nest.js. Такий стек технологій легко та зручно масштабувати, та розгортати під різні платформи Android, iOS, desktop тощо. Далі На базі Flutter існують різні типи архітектури системи. Проаналізуємо їх та виділимо основні плюси та мінуси. Розглянемо деякі з цих підходів.

Native state

Основа роботи з Flutter складається з віджетів. В свою чергу віджети можуть бути видимими, невидимими, містити в собі дочірні віджети та взаємодіяти між собою. Також віджети можуть бути без стану (Stateless Widget), так і містити у собі стан (Stateful Widget). Головна відмінність між ними — можливість повторно відображати віджети в ході роботи програми. Віджети без стану використовуються лише один раз і являється незмінним. Stateful Widget можуть використовуватися будь яку кількість разів в залежності від змін стану віджета.

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

Основні недоліки:

- Бізнес-логіка та View не розділені;
- Труднощі в модульному тестуванні;
- Складність в масштабуванні і підтримці.

BloC

Business Logic Component являє собою шаблон, створений Google для роботи з складним станом додатків та базується ґрунтується на реактивній парадигмі. Головна ідея підходу полягає в тому що додаток розбитий на модулі, що реалізують бізнес-логіку.

Кожен з цих модулів містить одну або кілька вхідних потоків для агрегування зовнішніх подій. Вихідними даними виступає Stream, який визначає

асинхронний формат даних для віджетів. Щоб скористатися модулем, застосовують StreamBuilder, який маніпулює потоком даних і на автоматичному рівні вирішує проблеми підписки і перебудовою дочірнього дерева віджетів.

Використовувати BLoC в чистому вигляді — не проста робота, оскільки для маніпуляції з потоками треба застосовувати бібліотеку RxDart, та для безпеки інформації потрібно власноруч відписуватися від потоків. Підхід має велику кількість переваг:

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

Provider (Scoped Model)

Даний тип архітектури дозволяє винести бізнес-логіку з Widget та надає можливість використовувати цю логіку в різних модулях додатку. Робота додатку базується на створенні моделей та реагуванням підписаних віджетів на її зміну.

Для віджета локального стану необхідно створити модель з усіма полями, що будуть використовуватися у віджеті. Після зміни будь якого поля (або полів) потрібно повідомити підписані на модель віджети та виконати візуалізацію підписаних віджетів. Підписка відбувається безпосередньо до того віджету, який залежить від даних зі створеної моделі.

Глобальний стан не відрізняється від локального, окрім того, що модель підписується до найвищого віджету додатка.

Перевага підходу полягає у поділу бізнес-логіки та інтерфейсу за допомогою створення моделей і event-based-архітектури. Також процес тестування таких моделей легше, ніж в Native state, за рахунок відсутності додаткових зусиль на створення віджетів, в яких цей стан використовується.

Даний тип архітектури підтримується Google. Одна з основних проблем такої архітектури — складність в розумінні того, які властивості було змінено.

Для написання проекту використаємо тип архітектури BloC, вона дозволяє легко і швидко впровадити необхідний бізнесу функціонал та легка у тестуванні. Архітектура BloC (business logic components) зображена на рисунку 2.1

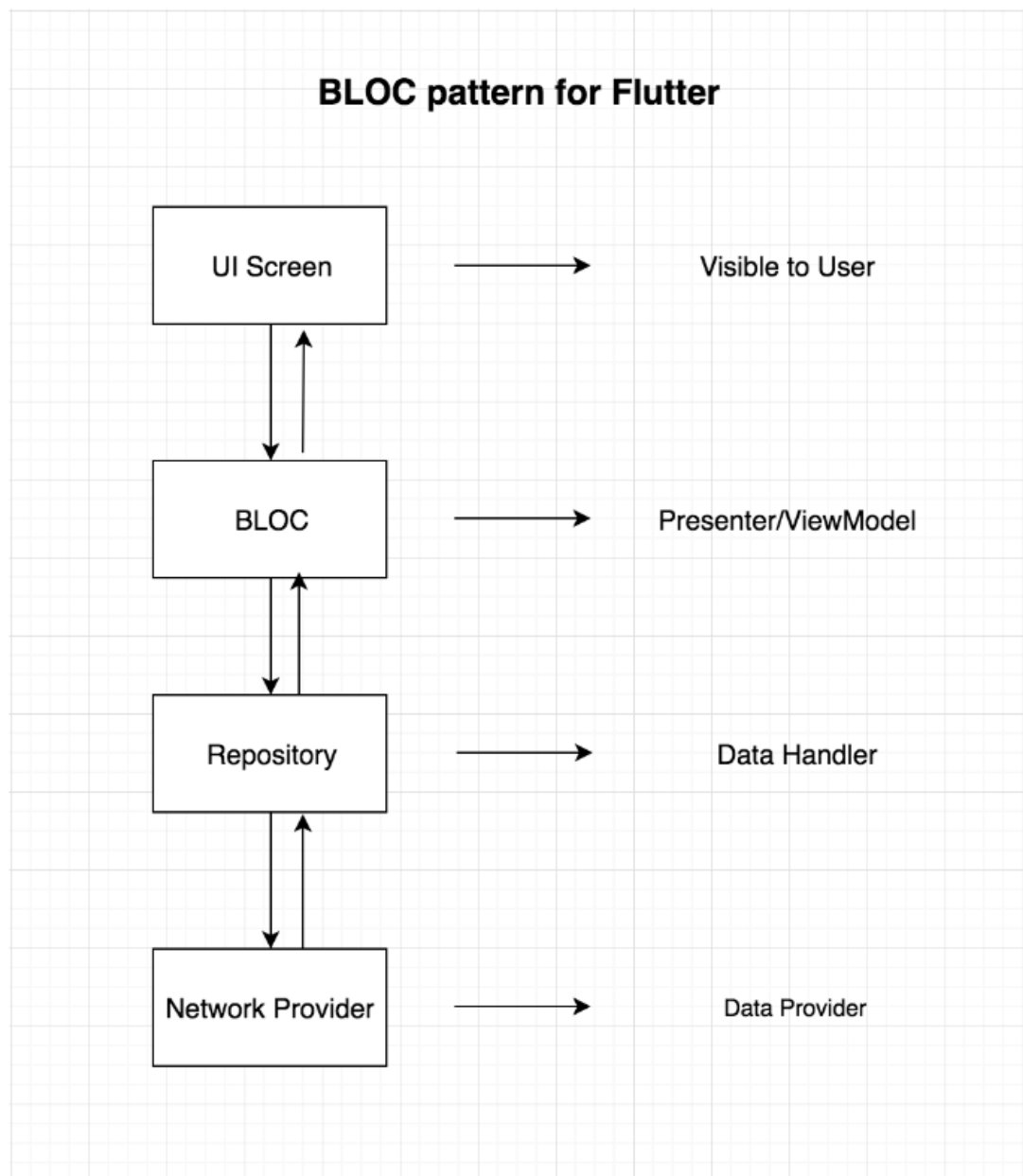


Рис. 2.2 – Схема архітектури BloC

Для розробки повноцінного програмного комплексу, який буде включати в себе розроблювану платформу та мобільні додатки «щоденник самоконтролю»,

«контроль пацієнтів для лікаря ендокринолога», 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 робочих станцій та сервером.

REST (RESTful) - це загальні принципи організації взаємодії додатка / сайту з сервером за допомогою протоколу HTTP. Особливість REST в тому, що сервер не запам'ятовує стан користувача між запитами - в кожному запиті передається інформація, що ідентифікує користувача (наприклад, token, отриманий через OAuth-авторизацію) і всі параметри, необхідні для виконання операції.

Все взаємодія з сервером зводиться до 4 операцій:

1. отримання даних з сервера (зазвичай у форматі JSON, або XML)
2. додавання нових даних на сервер
3. модифікація існуючих даних на сервері
4. видалення даних на сервері

Операція отримання даних не може приводити до зміни стану сервера.

Для кожного типу операції використовується свій метод HTTP-запиту:

1. отримання - GET
2. додавання - POST
3. модифікація - PUT
4. видалення – DELETE

Схема даної архітектури зображено на рисунку 2.2.

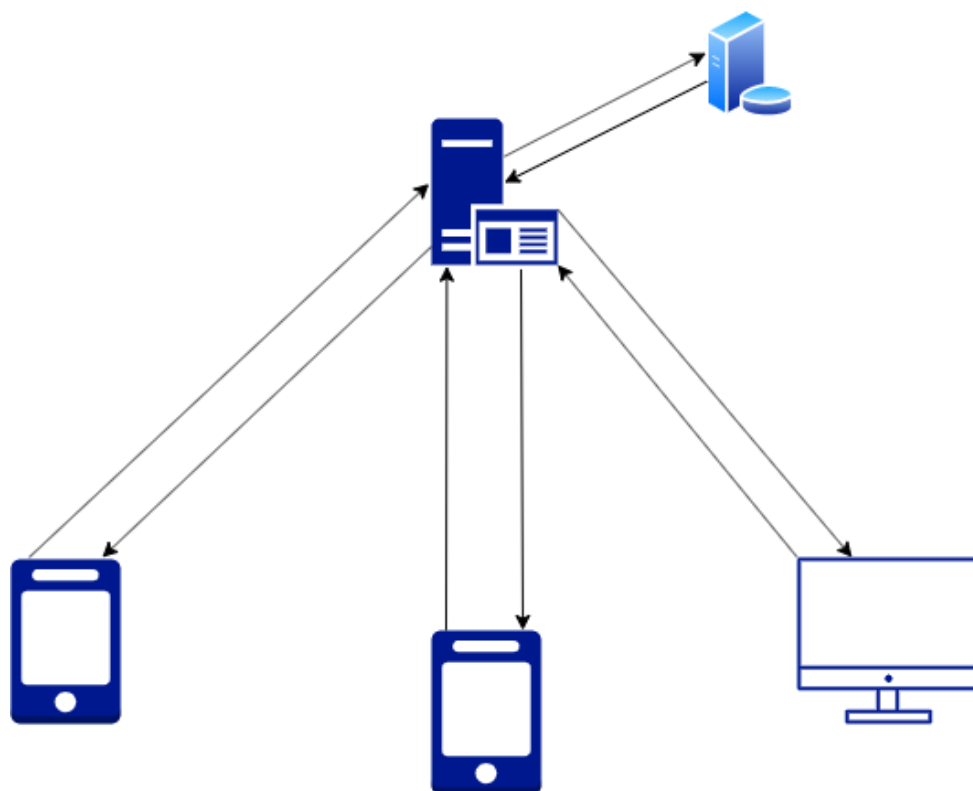


Рис 2.3 – Клієнт-серверна архітектура додатку

2.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Як було описано вище, для реалізації додатку ми будемо використовувати наступний стек технологій: Dart, Flutter(framework), Node.js, Nest.js.

Опишемо кожну технологію детальніше.

Dart

Мову, знання якої може завтра стати обов'язковим пунктом в резюме будь-якого поважаючого себе веб-розробника. Причина тому: він був створений і активно просувається Google. У 2011 році офіційно почалася розробка мови Google Dart, який за задумом авторів, повинен повністю замінити JavaScript. Причиною тому послужили «фундаментальні вади» останнього (зі слів Марка Міллера, одного з розробників Dart), які виявилось просто неможливо виправити еволюційним шляхом. У 2013 була представлена перша офіційна стабільна версія. Крім синтаксису, є маса зручностей прискорюють, як розробку коду, так і

його подальше виконання на машині. Наприклад, статична типізація з усіма наслідками, що впливають перевагами за швидкістю і можливістю відловити переважну кількість помилок ще на стадії компіляції [21].

Flutter

Молода, але дуже перспективна платформа, вже привернула до себе увагу великих компаній, які запустили свої додатки. Цікава ця платформа своєю простотою порівнянної з розробкою веб-додатків, і швидкістю роботи на рівні з нативними додатками. Висока продуктивність програми і швидкість розробки досягається за рахунок декількох шляхів.

На відміну від багатьох відомих на сьогоднішній день мобільних платформ, Flutter не використовує JavaScript ні в якому вигляді. В якості мови програмування для Flutter вибрали Dart, який компілюється в бінарний код, за рахунок чого досягається швидкість виконання операцій порівнянна з Objective-C, Swift, Java, або Kotlin.

Flutter не використовує нативні компоненти, знову ж таки, ні в якому вигляді, так що не доводиться писати ніяких прошарків для комунікації з ними. Замість цього, він будує весь інтерфейс самостійно. Кнопки, текст, медіа-елементи, фон — все це будується всередині графічного движка в самому Flutter. [26]

Node.js

Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Якщо раніше Javascript застосовувався для обробки даних в браузері користувача, то node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їх виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

Node.js має наступні властивості:

- асинхронна одно-нитева модель виконання запитів;

- неблокуючий ввід/вивід;
- система модулів CommonJS;
- рушій JavaScript Google V8;

Nest.js

Nest — це основа для створення ефективних масштабованих додатків на сервері Node.js. Він використовує прогресивний JavaScript, створений і повністю підтримує TypeScript і поєднує елементи ООР (об'єктно-орієнтоване програмування), FP (функціональне програмування) та FRP (функціональне реактивне програмування). Під кришкою Nest використовує надійні рамки HTTP-сервера, такі як Express (за замовчуванням), і, за бажанням, можна налаштувати і для Fastify!

Nest забезпечує рівень абстракції над цими загальними рамками Node.js (Express / Fastify), але також піддає свої API безпосередньо розробнику. Це дозволяє розробникам свободу використовувати безліч сторонніх модулів, доступних для базової платформи.

Figma

Figma - це графічний редактор для веб-дизайну, за допомогою нього можна швидко і легко розробляти:

- інтерактивні прототипи сайтів і мобільних додатків;
- спеціальні елементи інтерфейсу - іконки, кнопки, меню, вікна, форми зворотного зв'язку;
- векторні зображення.

Документи програми як і в Google Docs зберігаються в хмарі. Тому одночасно над проектом можуть працювати кілька співробітників, при цьому немає необхідності завантажувати файли для редагування. Вхід в Figma можна здійснити через браузер або ж завантажити програму на свій ПК. Програма працює на Windows і Mac OS. Десктопна версія дозволяє працювати офлайн, а при підключенні інтернету синхронізувати дані для подальшої роботи.

Основні переваги Figma:

- зберігання початкових кодів в хмарі
- командний доступ до оригіналів макета і можливості для спільної роботи.

Доступні аналогічні функції, що і в Google Docs - загальний доступ на перегляд і редагування файлів, паралельна робота групи людей. Інструмент безкоштовний для особистого використання. Для старту роботи необхідна лише реєстрація.

Visual Studio Code

Visual Studio Code — редактор вихідного коду, розроблений Microsoft для Windows, Linux і macOS. Позиціонується як «легкий» редактор коду для кроссплатформенної розробки веб-і хмарних додатків. Включає в себе відладчик, інструменти для роботи з Git, підсвічування синтаксису, IntelliSense і засоби для рефакторинга. Має широкі можливості для кастомізації: призначені для користувача теми, поєднання клавіш і файли конфігурації. Розповсюджується безкоштовно, розробляється як програмне забезпечення з відкритим вихідним кодом, але готові збірки розповсюджуються під пропріетарною ліцензією.

Visual Studio Code заснований на Electron - фреймворк, що дозволяє з використанням Node.js розробляти настільні додатки, які працюють на движку Blink. Незважаючи на те, що редактор заснований на Electron, він не використовує редактор Atom. Замість нього реалізується веб-редактор Monaco, розроблений для Visual Studio Online.

Adobe Photoshop

Adobe Photoshop - це растровий графічний редактор, розроблений та опублікований компанією Adobe Inc. для Windows та macOS. Спочатку він був створений у 1988 році Томасом та Джоном Ноллом. З цього часу це програмне забезпечення стало галузевим стандартом не лише в редагуванні растрової графіки, але й у цифровому мистецтві в цілому.

Photoshop може редагувати та створювати растрові зображення в декількох шарах і підтримує маски, альфа-композиції та кілька кольорових моделей, включаючи RGB, CMYK, CIELAB, кольоровий шар та дуетон. Photoshop використовує власні формати файлів PSD та PSB для підтримки цих функцій.

Окрім растрової графіки, це програмне забезпечення має обмежені можливості редагування чи візуалізації тексту та векторної графіки (особливо через відсічний контур для останнього), а також тривимірної графіки та відео. Набір функцій можна розширити плагінами; програми, розроблені та розповсюджені незалежно від Photoshop, які працюють всередині нього та пропонують нові або розширені функції.

СКБД

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

Обрані системи керування базами даних підходять для проведення аналізу й порівняння тому що реалізують реляційну модель даних.

Таблиця 2.1

Аналіз загальних показників СКБД

	PostgreSQL	MySQL
ANSI SQL сумісність	Близька до стандартів ANSI SQL	Слідую деяким стандартам ANSI SQL
Швидкодія	Швидше	Повільніше
Вкладені селекти	Так	Ні
Транзакції	Так	Так, але має бути тип таблиці InnoDB

Відповідь БД	Так	Так
Зовнішні ключі	Так	Ні
Представлення	Так	Ні
Процедури	Так	Ні
Тригери	Так	Ні
Unions	Так	Ні
Повні Joins	Так	Ні
Обмежувачі цілісності	Так	Ні
ODBC	Так	Так
JDBC	Так	Так

Проведений аналіз таких СКБД, як **PostgreSQL**, **MySQL**. показав, що для виконання поставлених задач найбільш підходящими є PostgreSQL. Ця СКБД є більш гнучкою та має у порівнянні розширений список функцій.

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. 2.1.

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

Таблиця «PatientInfo». В ній зберігається основна інформація про пацієнта, його вік, стать, стаж, вага, терапія тощо.

Таблиця «DoctorInfo». В ній зберігається основна інформація про лікаря, його вік, місце навчання, стаж роботи, спеціалізація тощо.

Таблиця «PatientList». В даній таблиці зберігається інформація про те які пацієнти за якими лікарями закріплені.

Таблиця «Settings». В ній зберігаються дані про налаштування додатку користувачів.

Таблиця «Diary». В ній зберігаються записи пацієнтів в щоденник самоконтролю, який містить в собі такі відомості, як: рівень глюкози, кількість введених ліків (базал, болус), активність, кількість вуглеводів тощо.

Для оптимізації структури бази даних було створено ще ряд таблиць. До цих таблиць входять: «Units», «University», «Illnes», «DiabetesTypes», «Therapy», «Drugs», «IllnesHistory», «Category», «Activity».

DiabetesTypes

@Entity()

```
export class DiabetesTypes extends BaseEntity {  
  @PrimaryGeneratedColumn()  
  id: number;  
  @Column()  
  name: string;  
}
```

Illness

@Entity()

```
export class Illness extends BaseEntity {  
  @PrimaryGeneratedColumn()  
  id: number;  
  
  @Column()  
  name: string;  
  
  @ManyToOne(type => IllnessHistory, history => history.illness_ids)  
  history: IllnessHistory;  
}
```

IllnessHistory

@Entity()

```

export class IllnessHistory extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;
  @OneToMany(type => Illness, illness => illness.history)
  illness_ids: Illness[];
}

```

Therapy

```

@Entity()
export class Therapy extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;
  @Column()
  name: string;
}

```

Patient

```

@Entity()
export class Patient extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({
    nullable: true
  })
  first_name: string;
  @Column({
    nullable: true
  })
  last_name: string;
  @Column({
    nullable: true
  })
  sex: string;
  @Column({
    nullable: true
  })
  weight: number;
  @Column({
    nullable: true
  })
  growth: number;
}

```

```

@Column({
  nullable: true
})
birthday: Date;

@OneToOne(type => DiabetesTypes)
@JoinColumn()
diabetes_type: DiabetesTypes | null;

@OneToOne(type => Therapy)
@JoinColumn()
therapy_id: Therapy | null;

@OneToOne(type => IllnessHistory)
@JoinColumn()
illness_history_id: IllnessHistory | null;

@OneToOne(type => Activity)
@JoinColumn()
activity_id: Activity | null;
}

```

В результаті була реалізована реляційна база даних, яка забезпечує збереження та доступ до інформації додатку.

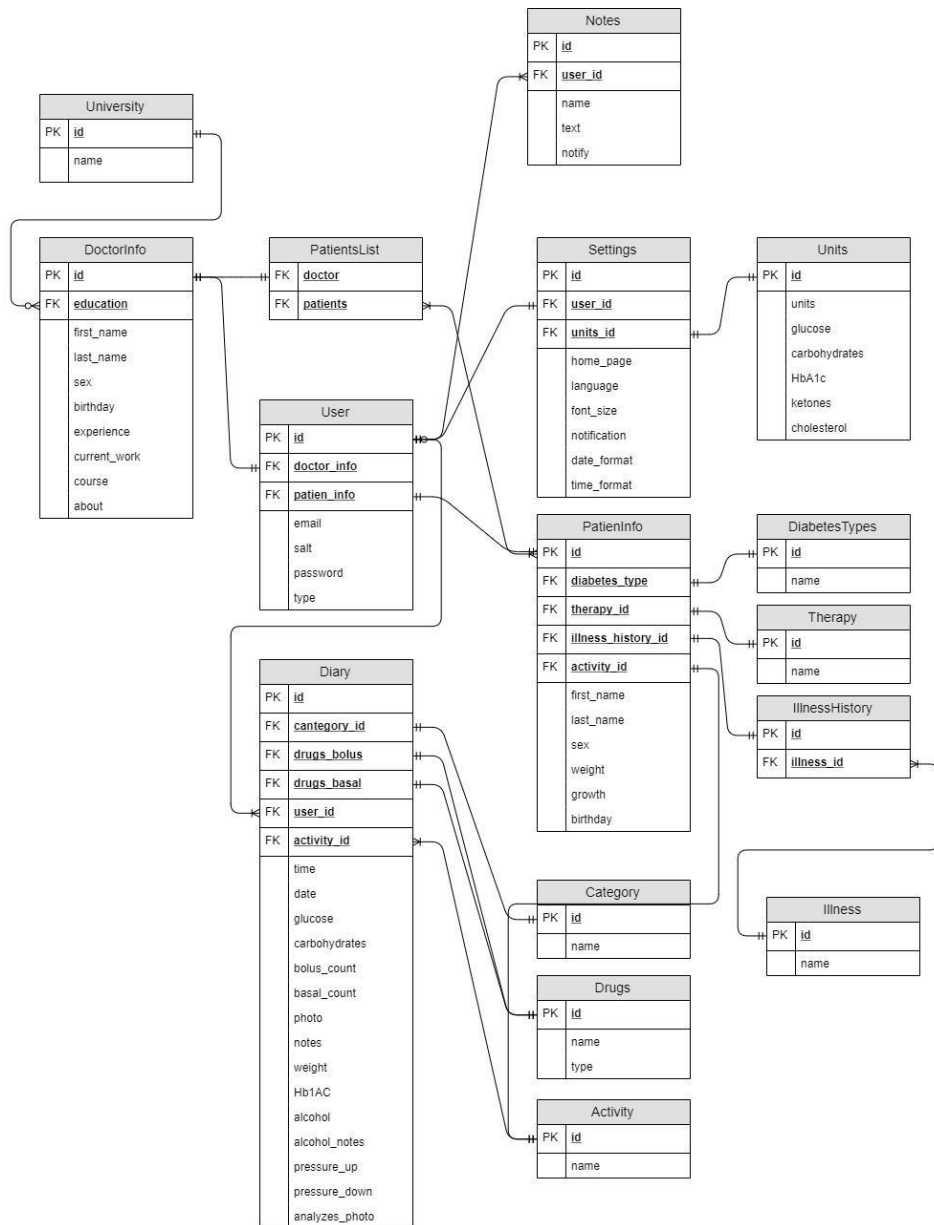


Рис.2.4 – Структурна схема БД

2.5 Реалізація функціональних частин додатку

Розглянемо модуль пацієнта. Основним класом, який задає параметри запитів, а також валідацію цих параметрів, являється PatientInfoDto.

```

export class PatientInfoDto {
    @IsString()
    @IsNotEmpty()
    first_name: string;
    @IsString()
    @IsNotEmpty()

```

```

last_name: string;
  @IsString()
  @IsNotEmpty()
sex: string;
  @ApiModelProperty()
  @IsNumberString()
  @IsNotEmpty()
weight: number;
  @IsNumberString()
  @IsNotEmpty()
growth: number;
  @IsDateString()
  @IsNotEmpty()
birthday: Date;
  @IsEmpty()
diabetes_type: DiabetesTypes | null;
  @IsEmpty()
therapy_id: Therapy | null;
  @IsEmpty()
illness_history_id: IllnessHistory | null;
  @IsEmpty()
activity_id: Activity | null;
}

```

PatientModule – модуль, який відповідає за взаємодію з інформацією про пацієнта. В цьому модулі йде підключення необхідних для роботи сервісів, а також репозиторія (необхідний для роботи з базою даних).

```

@Module({
  imports: [
    ConfigModule,
    TypeOrmModule.forFeature([PatientRepository]),
  ],
  providers: [PatientInfoService],
  controllers: [PatientInfoController],
})
export class PatientInfoModule {}

```

PatientController – відповідає за опис маршрутів, для взаємодії з інформацією пацієнта. Контролер реалізує методи для створення та оновлення записів.

```
@Controller('patient-info')
export class PatientInfoController {
  constructor(
    private patientInfoService: PatientInfoService,
  ) {}
  @UseGuards(AuthGuard('jwt'))
  @Post('/')
  create(
    @Body(ValidationPipe) patientInfoDto: PatientInfoDto,
  ): Promise<Patient> {
    Logger.verbose(`POST /patient-info/ with params
${JSON.stringify(patientInfoDto)}`);
    return this.patientInfoService.create(patientInfoDto);
  }

  @UseGuards(AuthGuard('jwt'))
  @Put('/:id')
  update(
    @Param('id', ParseIntPipe) id: number,
    @Body(ValidationPipe) patientInfoDto: PatientInfoDto,
  ): Promise<Patient> {
    Logger.verbose(`PUT /patient-info/${id} with params
${JSON.stringify(patientInfoDto)}`);
    return this.patientInfoService.update(id, patientInfoDto);
  }
}
```

PatientService – відповідає за взаємодію з репозиторієм, тобто саме в сервісі йде звернення до методів репозиторію, які в свою чергу виконують маніпуляції з даними. В сервісі представлено два методи – create, update.

```
@Injectable()
export class PatientInfoService {
  constructor(
    @InjectRepository(PatientRepository)
```

```

        private patientRepository: PatientRepository,
    ) {}
}

```

Create – метод слугує для створення інформації про пацієнта. Вхідними параметрами методу є patientInfoDto з типом PatientInfoDto. В тілі методу у нас йде звернення до базового методу репозиторію create.

```

create(patientInfoDto: PatientInfoDto): Promise<Patient> {
    return this.patientRepository
        .create(patientInfoDto)
        .save();
}

```

Update – метод слугує для оновлення інформації про пацієнта. Вхідними параметрами методу є patientInfoDto з типом PatientInfoDto, також вхідним параметром є id типу number. Цей параметр необхідний для пошуку необхідного запису в базі даних. В тілі методу у нас йде звернення до базового методу репозиторію update.

```

async update(id: number, patientInfoDto: PatientInfoDto): Promise<Patient>
{
    await this.patientRepository.update({id}, patientInfoDto);
    return await this.patientRepository.findOne({ id });
}

```

На головній сторінці додатку лікаря відображається загальна кількість пацієнтів, а також кількість пацієнтів їх рівнем глюкози. Фільтрування за рівнем глюкози відбувається за допомогою методу масиву where, який вхідними параметрами приймає callback-функцію. Розглянемо це на прикладі.

```

int badLength = patients.where((patient) =>
calculateA1c(calculateGlucoseAvg(patient.diary)) > 11 ||
calculateA1c(calculateGlucoseAvg(patient.diary)) < 4).length;
int needControllength = patients.where((patient) =>
calculateA1c(calculateGlucoseAvg(patient.diary)) >= 8 &&

```

```

calculateA1c(calculateGlucoseAvg(patient.diary)) < 11 ||
calculateA1c(calculateGlucoseAvg(patient.diary)) >= 4 &&
calculateA1c(calculateGlucoseAvg(patient.diary)) < 5).length;
    int goodLength = patients.where((patient) =>
calculateA1c(calculateGlucoseAvg(patient.diary)) >= 5 &&
calculateA1c(calculateGlucoseAvg(patient.diary)) < 8).length;

```

В якості callback ми передаємо функцію, в якій перевіряємо входження рівня глюкози пацієнта в певну межу. В змінну ми записуємо кількість пацієнтів, які потрапляють під цю умову.

Реалізація навігації в додатку. Виходячи з архітектури додатку, навігація відбувається за допомогою NavigationBloc і в залежності від стану, який приймає цей блок, ми відображаємо необхідну сторінку.

```

@override
Stream<LeftNavigationState> mapEventToState(
    BottomNavigationEvent event,
) async* {
    if (event is PageTapped) {
        index = event.index;
        switch(event.index) {
            case 1:
                yield NotesPageLoaded();
                break;
            case 2:
                yield SettingsPageLoaded();
                break;
            default:
                yield HomePageLoaded();
        }
    }
}

```

Функція mapEventToState відповідає за відловлювання події і в залежності від цієї події змінює стан панелі навігації. За допомогою події PageTapped користувач змінює поточну сторінку. В тілі обробника події за допомогою

конструкції switch/case на основі індексу сторінки ми змінюємо стан на необхідний. Розглянемо як це виглядає на рівні презентації.

```
child: BlocBuilder<LeftNavigationBloc, LeftNavigationState >(
  builder: (context, state) {
    if (state is HomePageLoaded) {
      return PatientsWidget();
    } else if (state is NotesPageLoaded) {
      return NotesList();
    }
    return Container();
  },
),
```

На рівні презентації перевіряється стан панелі навігації за допомогою конструкції if/else. В залежності від стану повертається необхідний компонент.

Розглянемо схему роботи блоку LeftNavigationBloc. Даний блок відповідає за навігацію між сторінками. Блок Sync отримує подію, обробляє її, та на виході через блок Stream змінюється State. В залежності від State(стану):HomePageLoaded, NotesPageLoaded, SettingPageLoaded. додаток відкриває відповідний екран.

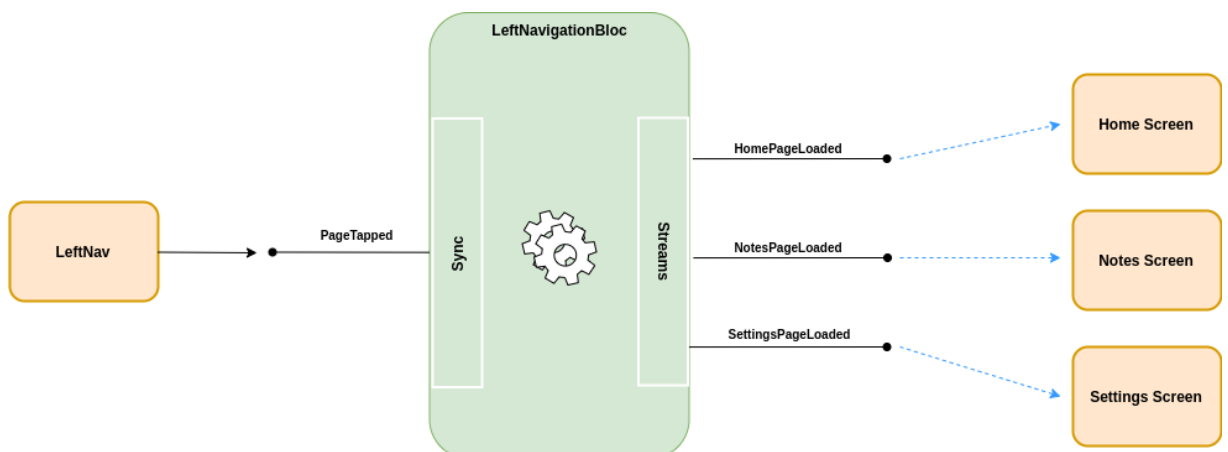


Рис. 2.5 – Схема роботи блока навігації LeftNavigationBloc

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 їх лістинг.

РОЗДІЛ 3. ДОСЛІДЖЕННЯ ПРАЦЕЗДАТНОСТІ ТА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ДОДАТКУ

3.1 Структура інтерфейсу. Інтерфейс та порядок роботи з додатком

В даному підрозділу описано та розглянуто інтерфейс реалізованої системи, та порядок роботи з системою.

Головний екран додатку розбито на 3 частини. Ліва колонка містить інформацію загальну інформацію для лікаря — його профіль, нотатки та навігація для роботи з різними модулями системи. Центральна колонка, призначена для відображення інформації про пацієнтів лікаря: список хворих, їх кількість, стан, компенсація хвороби пацієнтами.

Права колонка, відповідає за процес комунікації з пацієнтами. Ліва та права колонки фіксовані та не змінюються при роботі з іншими модулями системи. Інтерфейс головного екрану зображено на рис. 3.1

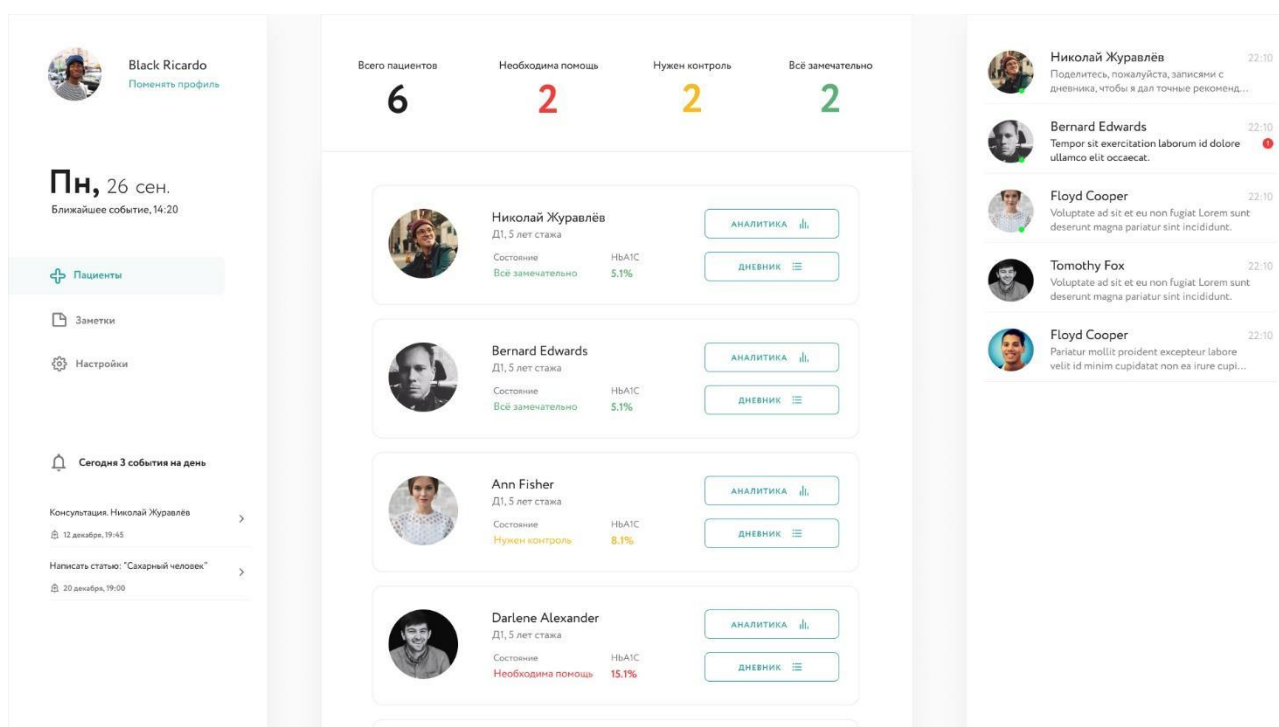


Рис. 3.1 – Головний екран додатку

Обравши потрібного пацієнта на головному екрані лікар переходить в профіль даного пацієнта, де відображаються показники компенсації хвороби

цукровий діабет. Окрім цього, профіль містить персональні дані: вагу, вік, вид терапії пацієнта, суміжні з діабетом хвороби, тощо(рис.3.2). Якщо виникає потреба в консультації хворого, лікар може прямо з профілю користувача перейти до чату.

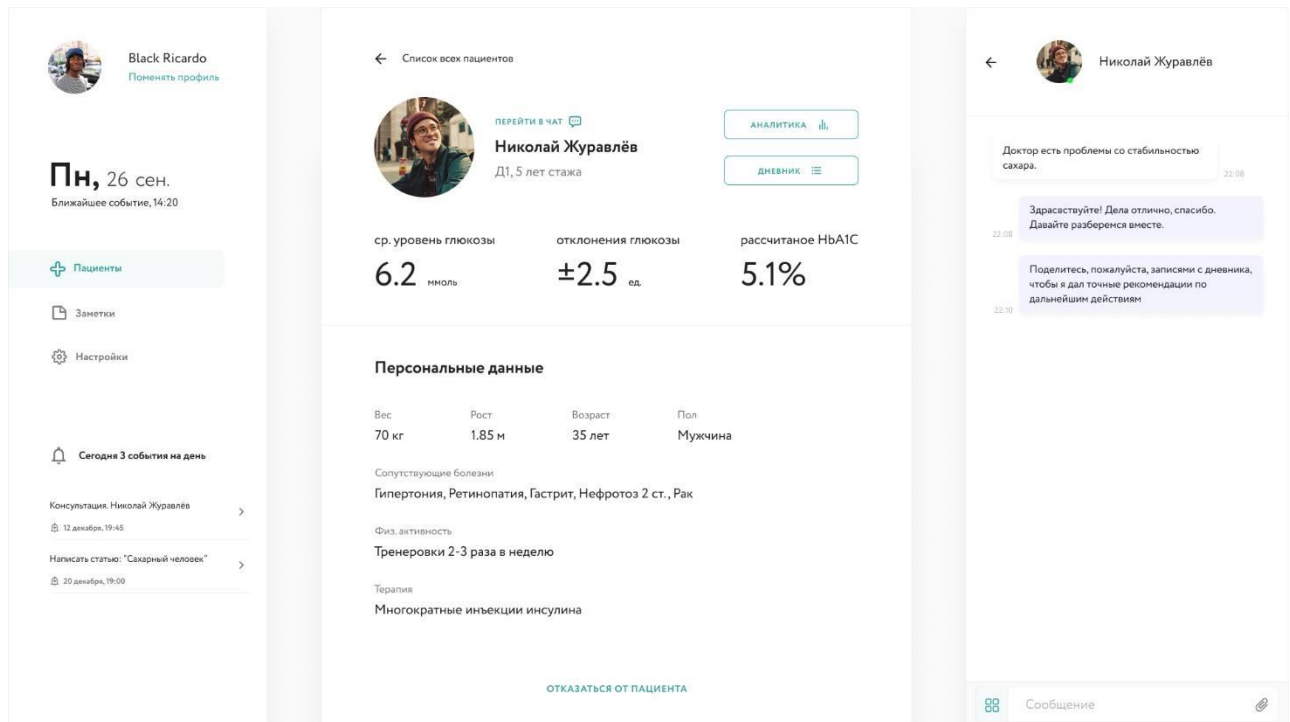


Рис. 3.2 – Екран профілю пацієнта

В профілю користувача, лікар може переглядати щоденник записів пацієнта та його аналітику. Та на базі цих даних надавати консультації хворому. Сторінка з щоденником зображено на рис.3.3, аналітика рис.3.4.

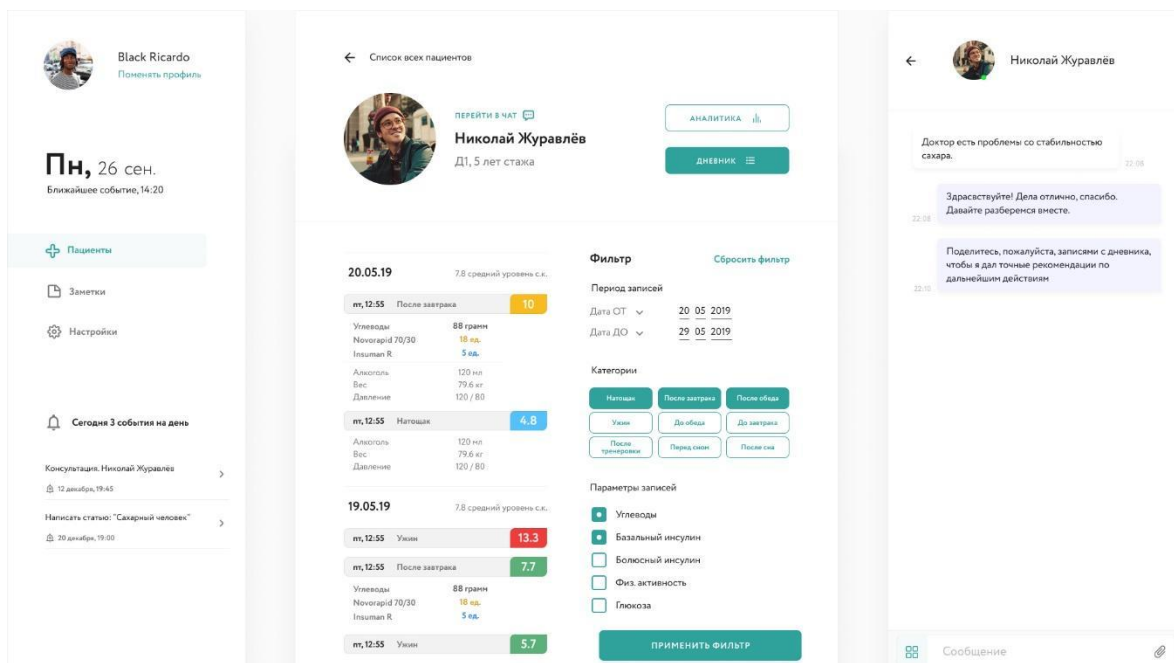


Рис. 3.3 – Щоденник записів пацієнта

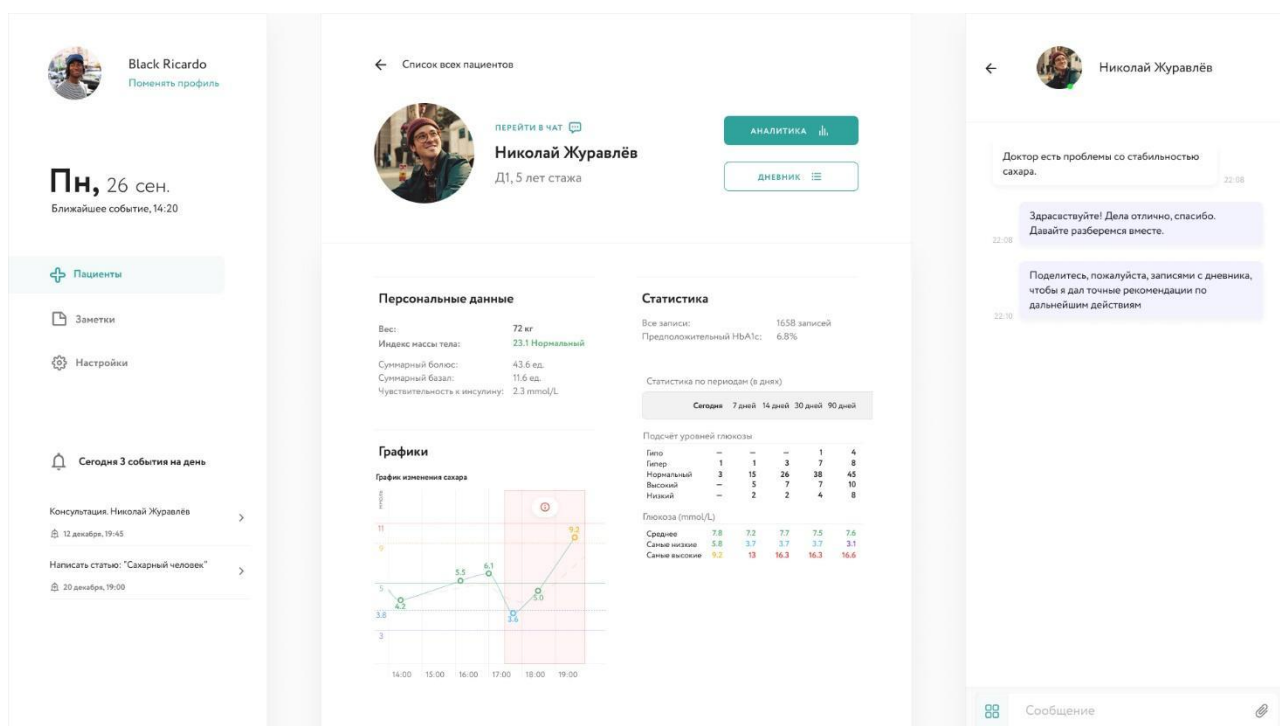


Рис. 3.4 – Аналітика по даним пацієнта

Для зручності лікаря, в додаток інтегровано модуль з нотатками. Лікар може створювати різного роду нотатки та нагадування. На сторінці відображаються існуючі додатки, та поле введення, що являє собою створення нової нотатки(рис.3.5).

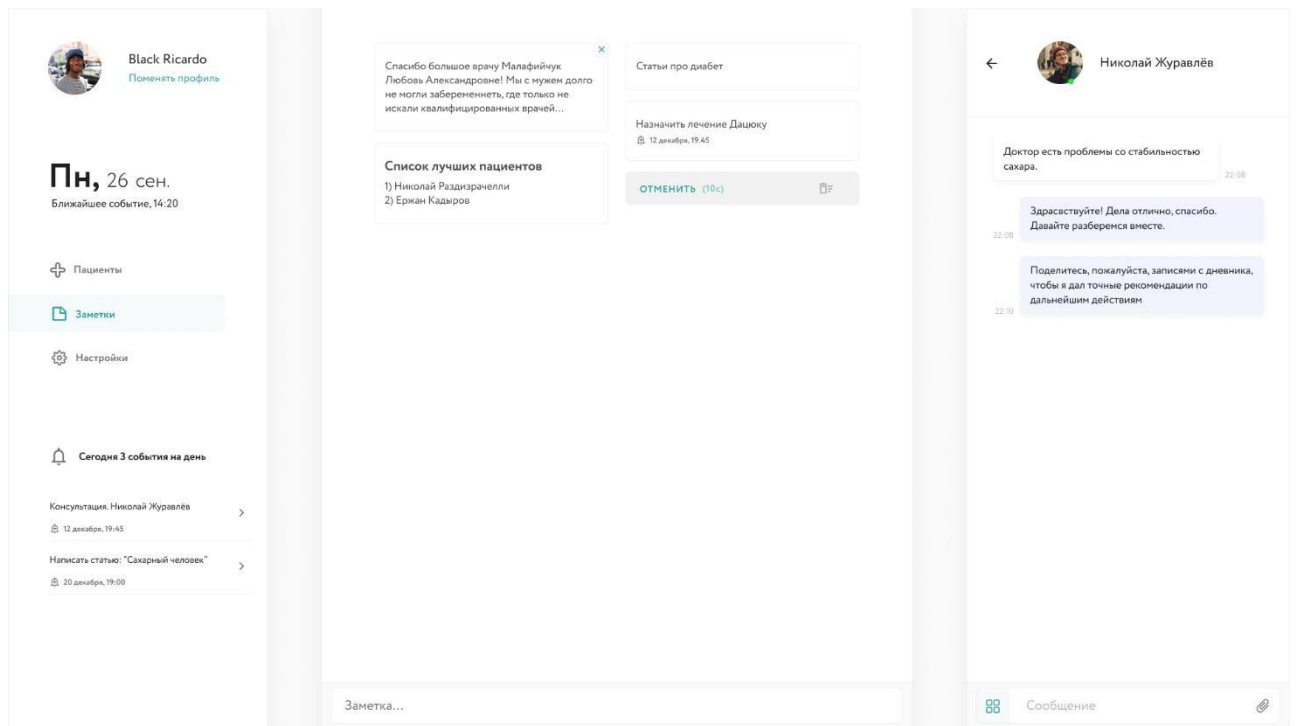


Рис. 3.5 – Сторінка нотаток

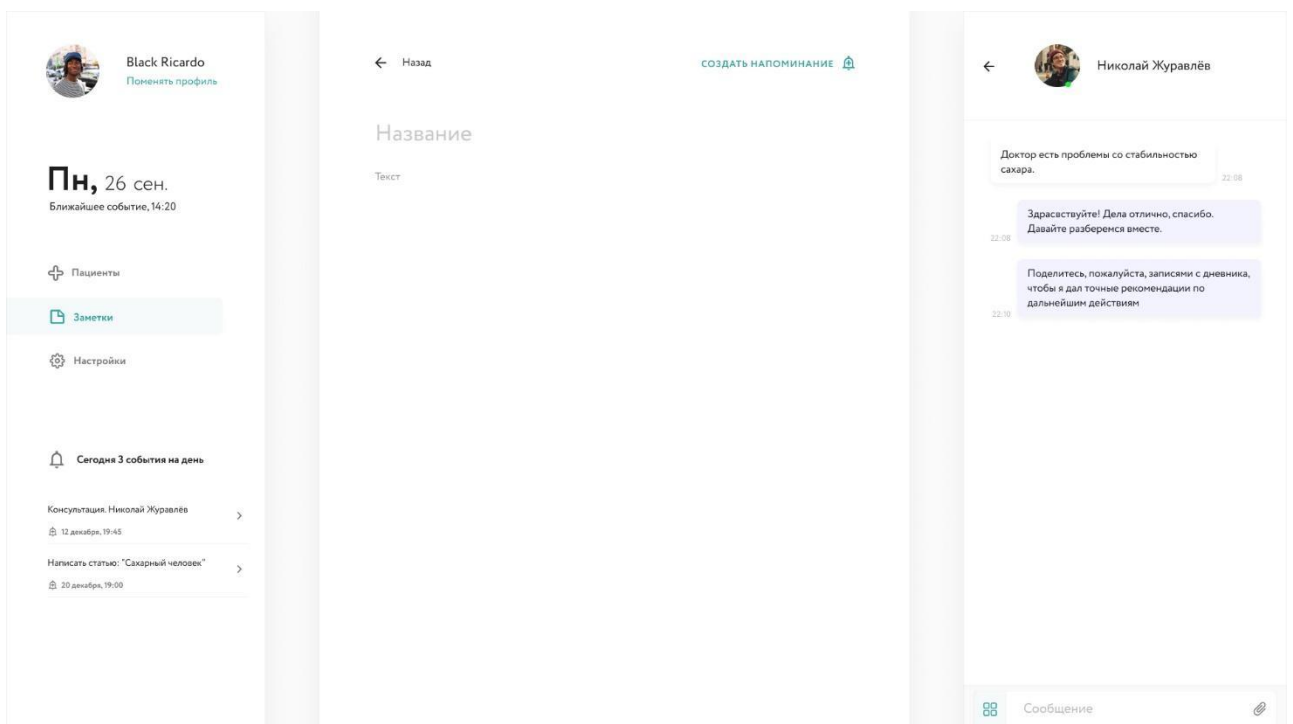


Рис. 3.6 – Сторінка створення нової нотатки

Модуль налаштування додатку містить в собі, можливість змінити персональні дані лікаря, та налаштування роботи самого додатку. Сторінка налаштування зображена на рис. 3.7

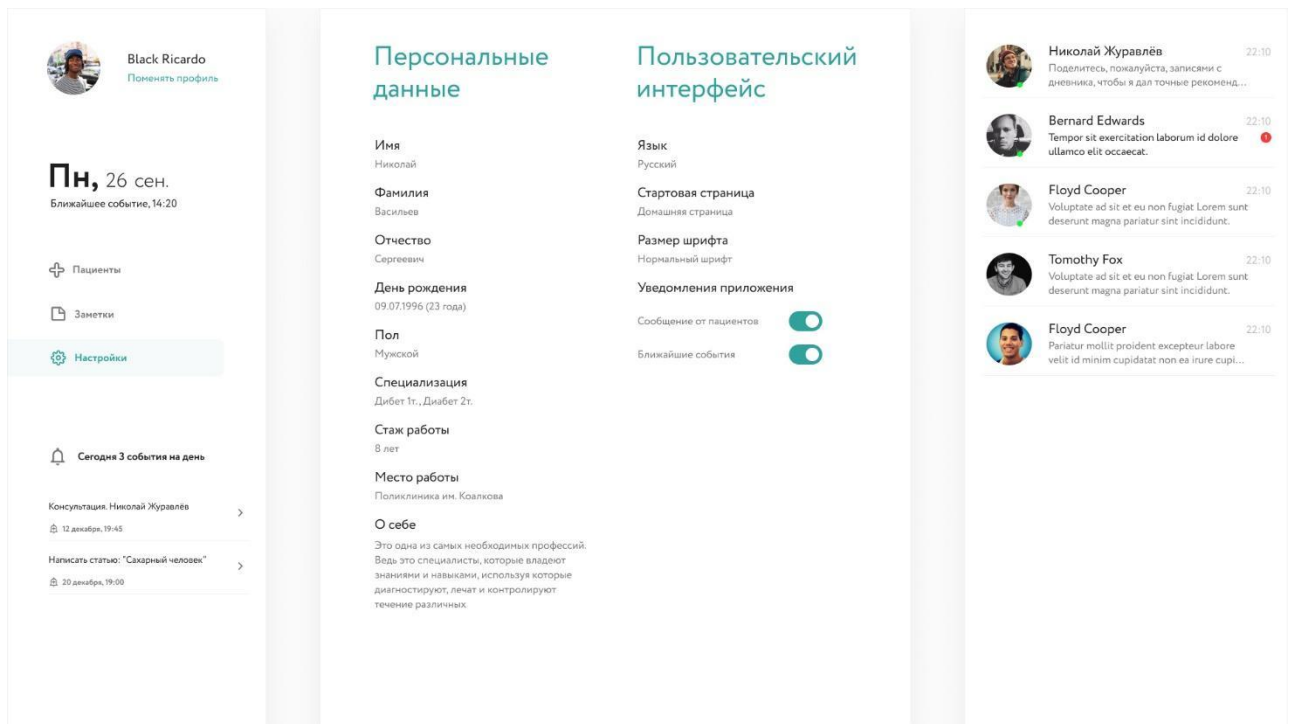


Рис. 3.7 – Сторінка налаштування додатку

3.2 Тестування роботи розробленого додатку

Після розробки платформи «Система контролю пацієнтів з цукровим діабетом» було здійснено ряд різного роду тестувань. До них відноситься тестування інтерфейсу користувача, функціональне тестування та системне тестування. Для проведення адекватного тестування, база даних додатку заповнено схожими до реальних даними.

Тестування програмного забезпечення — це процес, що використовується для виміру якості розроблюваного програмного забезпечення.

В залежності від переслідуваних цілей види тестування можна умовно розділити на наступні типи:

- Функціональні (Functional testing)
- Нефункціональні (Non-functional testing)
- Регресійне тестування (Regression 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. Функціональне тестування фактично імітує фактичне використання системи.

Нефункціональне тестування — тестування властивостей, які не належать до функціональності системи.

Дані властивості визначаються нефункціональними вимогами, які характеризують продукт з таких сторін, як:

Надійність (реакція системи на непередбачені ситуації).

Продуктивність (Працездатність системи під різними навантаженнями).

Зручність (Дослідження зручності роботи з додатком з точки зору користувача).

Безпека (Захищеність призначених для користувача даних).

Кросплатформне тестування — здатність системи працювати на різних платформах. Забезпечується завдяки використанню високорівневих мов програмування, середовищ розробки і виконання, що підтримують умовну компіляцію, компоновку і виконання коду для різних платформ.

Таблиця 3.1

Тестування роботи на різних платформах

Платформа	Тестування	Результат тестування
Windows	Функціональне	Працює стабільно
Mac OS	Тестування навантаженням	Працює стабільно
Linux	Тестування безпеки	Додаток демонструє стабільну роботу

Функціональне тестування — пошук відмінностей між поточною поведінкою написаних функцій і очікуваною поведінкою.

Таблиця 3.2

Функціональне тестування додатку

Модуль	Результат тестування
Авторизація	Модуль працює згідно прогнозу
Модуль консультації пацієнта	Модуль працює згідно прогнозу
Модуль роботи з нотатками	Модуль працює згідно прогнозу
Налаштування системи	Модуль працює згідно прогнозу
Робота з щоденником пацієнта	Модуль працює згідно прогнозу
Робота з аналітикою пацієнта	Модуль працює згідно прогнозу

Для повноцінної реалізації автоматичних тестів був запущений окремий сервер для запуску і проходження тестів, завдяки чому вдалося своєчасно виявляти дефекти і бачити актуальну картину стану продукту.

За підсумками проведення тестів продуктивності і навантажувального тестування були виявлені слабкі місця в додатку, що дозволило домогтися швидкої і стабільної роботи сервісу.

3.3 Аналіз та моделювання даних на базі записів додатку

Даний розділ описує статистичні порівняння даних по хворобі, 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.

NumPy — це бібліотека мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих (і дуже швидких) математичних функцій для операцій з цими масивами. Містить готові методи для найрізноманітніших операцій: від створення, зміни форми, множення і розрахунку детермінанта матриць до рішення лінійних рівнянь і сингулярного розкладання.

SciPy ґрунтується на NumPy і розширює її можливості. Включає методи лінійної алгебри і методи для роботи з ймовірними розподілами, інтегральним обчисленням і перетвореннями Фур'є. Matplotlib – низькорівнева бібліотека для створення двовимірних діаграм і графіків. З її допомогою можна побудувати будь-який графік. Однак для складної візуалізації потрібно більше коду, ніж в розвинених бібліотеках.

Scikit-learn ґрунтується на NumPy і SciPy. Надає алгоритми для машинного навчання та інтелектуального аналізу даних: кластеризації, регресії і класифікації. Одна з найкращих бібліотек для компаній, що працюють з величезним обсягом даних. Її використовують Evernote, OKCupid, Spotify і Birchbox.

Seaborn - це бібліотека для створення статистичної графіки в Python. Він побудований поверх matplotlib і тісно інтегрований із структурами даних панд.

Ось деякі з функцій, які пропонує Seaborn:

- API, орієнтований на набір даних, для вивчення зв'язків між декількома змінними
- Спеціалізована підтримка використання категоричних змінних для показу спостережень або сукупної статистики
- Варіанти візуалізації унівариантних або біваріантних розподілів та порівняння їх між підмножинами даних
- Автоматичне оцінювання та побудова лінійних моделей регресії для різних змінних, що залежать від типу
- Зручне уявлення про загальну структуру складних наборів даних
- Абстракції високого рівня для структурування сіток з декількома ділянками, які дозволяють легко будувати складні візуалізації
- Короткий контроль над оформленням фігури matplotlib з кількома вбудованими темами
- Інструменти для вибору кольорової палітри, які вірно розкривають візерунки у ваших даних

Seaborn прагне зробити візуалізацію центральною частиною вивчення та розуміння даних. Її функції, орієнтовані на набір графіків, функціонують на фреймах даних та масивах, що містять цілі набори даних, і внутрішньо виконують необхідне семантичне відображення та статистичну агрегацію для отримання інформативних графіків.

В даному підрозділі було здійснено моделювання даних та аналіз за наступними показниками хворих на діабет:

Розподіл хворих за гендерною ознакою

- Розподіл хворих за віковою ознакою
- Кореляція між кількістю віком пацієнта та кількістю випадків коли ГК

була за межами норми

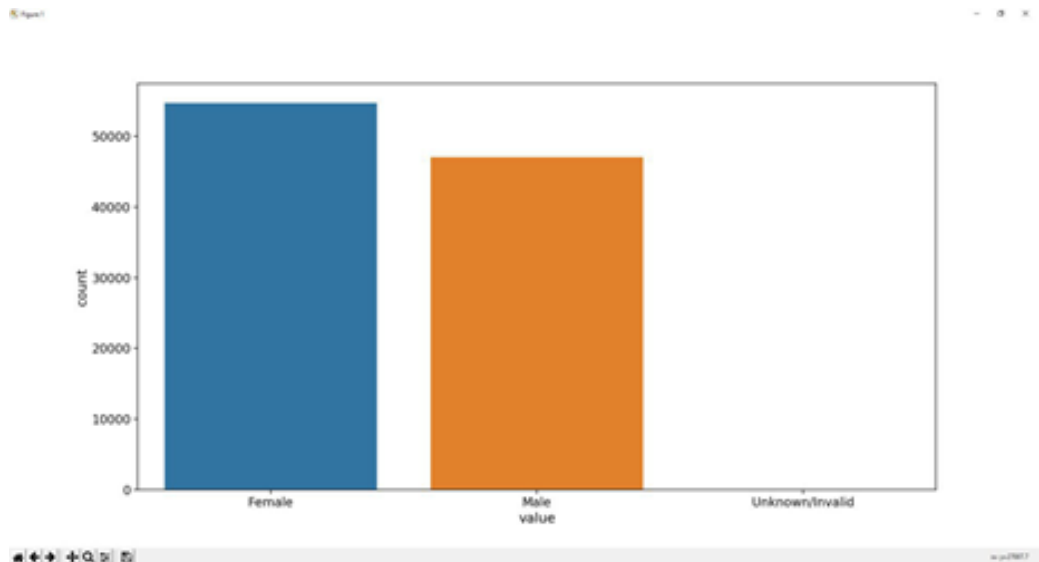


Рис. 3.8 – Графік розподілу хворих за гендером

За графіком чітко видно розподіл хворих за гендерними ознаками. Згідно результату аналізу можна зробити висновок, що жіноча стать більш вразлива до цукрового діабету.

Розподіл хворих за віковою ознакою

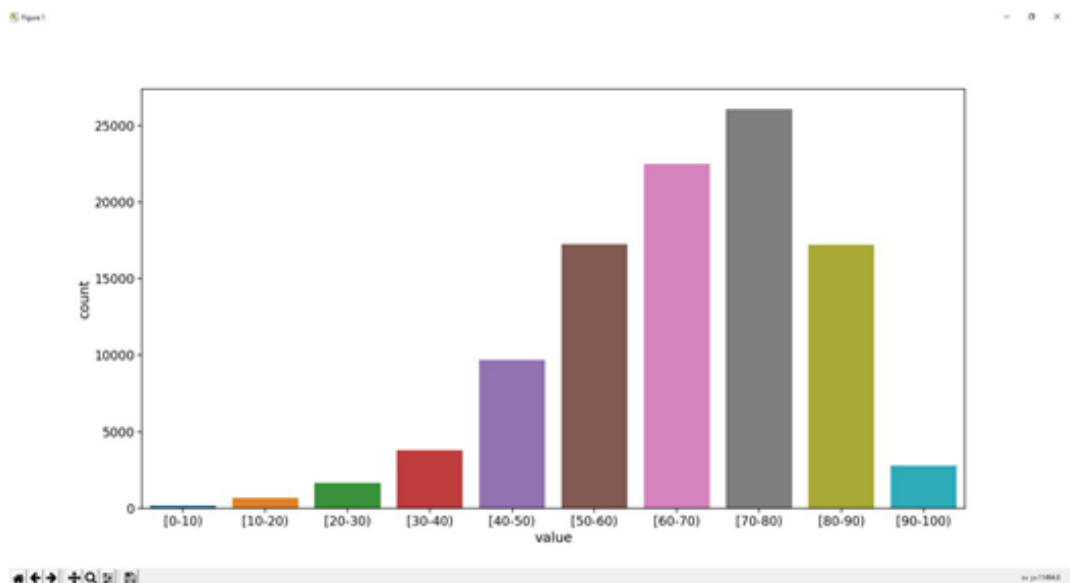


Рис. 3.9 – Графік розподілу хворих за віком

На графіку відображено розподіл хворих за віковою ознакою. По даним показникам можна побачити, зростання кількості хворих з віком. Тому можна зробити висновок — шанс захворіти на цукровий діабет зростає з віком.

Тепловая карта кількості випадків коли ГК була за межами норми та віком пацієнта

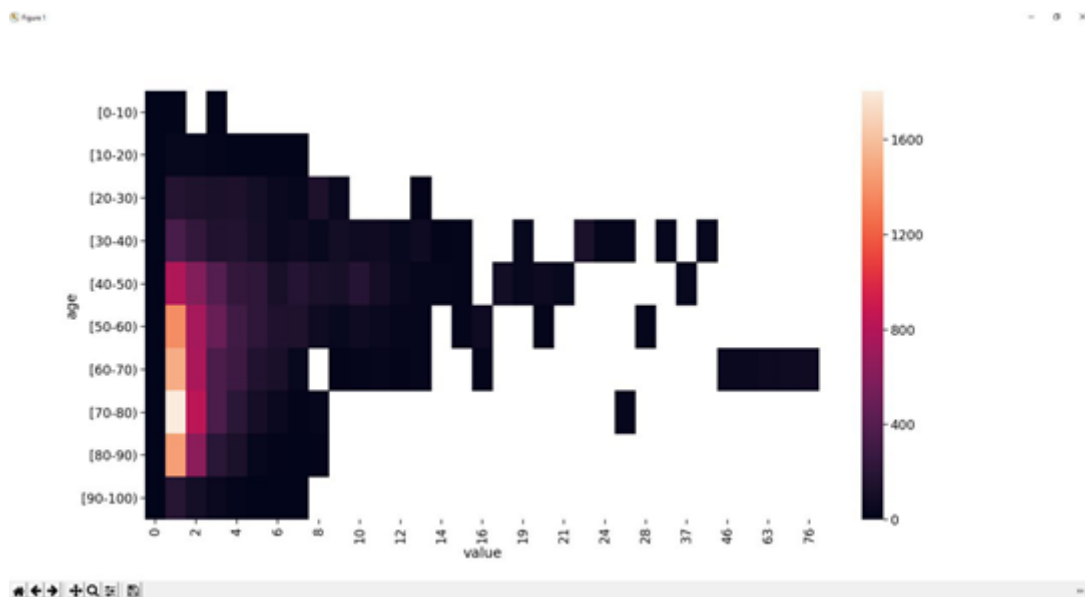


Рис. 3.10 – Тепловая карта між кількістю консультацій у лікаря та віком

На графіку відображено кореляція між віком та кількістю випадків коли ГК виходить за діапазон норми. За рис.3.10 можна побачити, що кількість виходу за межі норми збільшується з віком. Та сягає свого піку у період 70-80 років.

Висновки до третього розділу

В даному розділі було розглянуто роботу зі створеною платформою, а саме її користувацький інтерфейс: головний екран зі списком всіх пацієнтів, що закріплені за лікарем; модуль для роботи з нотатками; профіль пацієнта - щоденник самоконтролю та аналітика; спеціалізований чат для спілкування з пацієнтами.

Після чого було проведено тестування платформи. Яке включало в себе такі типи тестування, як: функціональне, не функціональне, системне та тестування інтерфейсу. Далі було проведено статистичний аналіз на базі даних. Наприклад, було виявлено, що кількість хворих жінок на цукровий діабет є дещо вищою в порівнянні з чоловіками.

ВИСНОВКИ

Результатом кваліфікаційного магістерського проекту — спроектований додаток «Система контролю пацієнтів з цукровим діабетом», для цього було виконано наступні етапи.

В ході першого етапу здійснено аналіз поставлених перед проектом задач та приведено поетапний процес розробки майбутнього програмного забезпечення.

Проведено аналіз ринку додатків даного типу та здійснено їх порівняння з метою вибірки кращих рішень та їх інтеграції в майбутній додаток.

Визначено основний функціонал додатку та засоби виконання поставлених задач. Розроблюваний додаток буде мати такі функції як:

- реалізація функціоналу контролю щоденника та аналітики пацієнта
- розробка спеціалізованого чату та належних до нього функцій;
- модуль налаштувань профілю користувача;
- функціонал створення та контролю нотаток
- модуль для відображення основної інформації про перебіг хвороби;

Другий розділ описує обрану архітектуру додатку та технології для її реалізації.

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 їх лістинг. В останньому пункті наводиться детальний розгляд коду деяких компонентів реалізованого додатку.

В третьому розділі продемонстровано інтерфейс додатку, описано порядок роботи з додатком. Здійснено тестування розробленого проекту. Останній підрозділ демонструє статистичні порівняння даних зібраних додатком.

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 Фадеев. 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.Balakrishnan N.P., Rangaiah G.P., Samavedham L. Review and analysis of blood glucose (BG) models for type 1 diabetic patients. Ind. Eng. Chem. Res. 2011; 50 (21): 12041–12066. DOI: 10.1021/ie2004779
- 13.Doyle F.J., Jovanovic L., Seborg D.E., Parker R.S., Bequette B.W. A tutorial on biomedical process control. J. Process Control. 2007; 17: 571–572. DOI: 10.1016/j.jprocont.2007.01.012.

14. Bremer T., Gough D.A. Is blood glucose predictable from previous values? A solicitation for data. *Diabetes*. 1999; 48: 445–451. DOI: 10.2337/diabetes.48.3.445.
15. Reifman J., Rajaraman S., Gribok A., Ward W.K. Predictive monitoring for improved management of glucose levels. *J. Diabet. Sci*
16. Sparacino G., Zanderigo F., Corazza S., Maran A., Facchinetti A., and Cobelli C. Glucose concentration can be predicted ahead in time from continuous glucose monitoring sensor time series. *IEEE Transact. Biomed. Engineer.* 2007; 54 (5): 931–937. DOI: 10.1109/TBME.2006.889774.
17. Van Herpe T., Espinoza M., Pluymers B., Wouters P., De Smet F., Van Berghe G., et al. Development of a critically ill patient input output model. *Proceedings 14th IFAC Symposium on System Identification (SYSID 2006)*. Newcastle. Australia. 2006: 481–486. DOI: 10.3182/20060329-3-AU-2901.00073.
18. Van Herpe T., Espinoza M., Pluymers B., Goethals I., Wouters P., Van den Berghe G., and De Moor B. An adaptive input output modeling approach for predicting the glycemia of critically ill patients. *Physiol. Meas.* 2006; 27 (11): 1057–1069. DOI: 10.1088/0967-3334/27/11/001.
19. Finan D.A., Doyle F.J., Palerm C.C., Bevier W.C., Zisser H.C. Experimental evaluation of a recursive model identification technique for type 1 diabetes. *J. Diabetes Sci Technol.* 2009; 3 (5): 1192–1202. DOI: 10.1177/193229680900300526.
20. Актуальні питання діагностики та лікування цукрового діабету [Електронний ресурс]. Доступ за посиланням: <https://m-l.com.ua/?aid=24>
21. Что такое Dart. Первая программа [Електронний ресурс]. – 2019. – Режим доступа до ресурсу: <https://metanit.com/dart/tutorial/1.1.php>.
22. Что такое Node.js и где он пригодится [Електронний ресурс] / Нетология. – 2018. – Режим доступа до ресурсу: <https://netology.ru/blog/node>.

- 23.Обзор архитектур управления состоянием на Flutter: [Электронный ресурс].
Доступ за посиланням: <https://dou.ua/lenta/articles/flutter-architecture/>
- 24.Node.JS: [Электронный ресурс]. Доступ за посиланням:
<https://uk.wikipedia.org/wiki/Node.js>
- 25.Flutter — Cookbook [Электронный ресурс]. – 2019. – Режим доступу до ресурсу: <https://flutter.dev/docs/cookbook>.
- 26.What is Flutter? Here is everything you should know [Электронный ресурс] / Concise Software. – 2019. – Режим доступу 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 доступу до ресурсу: <https://lawsofux.com/>.

ДОДАТКИ

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:flutter_secure_storage/flutter_secure_storage.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

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

class AuthApiClient {
  static final baseUrl = DotEnv().env['BASE_URL'];
  final http.Client httpClient;
  final storage = new FlutterSecureStorage();

  AuthApiClient({
    @required this.httpClient,
  }) : assert(httpClient != null);

  Future<User> signup({
    @required String email,
    @required String password,
  }) async {
    final url = '$baseUrl/auth/signup';
    try {
      final response = await this.httpClient.post(
        url,
        body: json.encode({'email': email, 'password': password, 'type': 'patient'}),
        headers: {
          HttpHeaders.contentTypeHeader: 'application/json',
        }
      );
      if (response.statusCode != 201) {
        final responseData = json.decode(response.body);
        print(responseData);
        if (responseData['message'] is List<dynamic>) {
          throw HttpException(responseData['message'][0]['constraints']['matches']);
        }
        throw HttpException(responseData['message']);
      }
      return await signin(email: email, password: password);
    } catch (error) {
      throw error;
    }
  }
}
```

```

}

Future<User> signin({
    @required String email,
    @required String password,
}) async {
    final url = '$baseUrl/auth/signin';
    try {
        final response = await this.httpClient.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);
        return User(
            id: data['userId'],
            token: data['accessToken'],
            filled: data['filled'],
            patient_info: data['patient_info'],
        );
    } catch (error) {
        throw error;
    }
}

Future<void> deleteToken() async {
    await storage.delete(key: 'token');
    await storage.delete(key: 'id');
    await storage.delete(key: 'filled');
    await storage.delete(key: 'patient_info');
    return;
}

Future<void> persistId(int id) async {
    await storage.write(key: 'id', value: id.toString());
    return;
}

```

```

Future<void> persistToken(String token) async {
  await storage.write(key: 'token', value: token);
  return;
}

Future<void> persistInfo(User user) async {
  await storage.write(key: 'id', value: user.id.toString());
  await storage.write(key: 'token', value: user.token);
  await storage.write(key: 'filled', value: user.filled.toString());
  await storage.write(key: 'patient_info', value: user.patient_info.toString());
  return;
}

Future<bool> hasToken() async {
  String token = await storage.read(key: 'token');
  return token != null;
}

Future<User> getUser() async {
  String token = await storage.read(key: 'token');
  String id = await storage.read(key: 'id');
  String filled = await storage.read(key: 'filled');
  String patient_info = await storage.read(key: 'patient_info');

  if (token == null) return User(
    id: 0,
    token: null,
    filled: false,
    patient_info: null,
  );

  return User(
    id: int.parse(id),
    token: token,
    filled: filled == 'true',
    patient_info: int.parse(patient_info),
  );
}
}

```

Додаток Б
Лістинг файлу auth_repository.dart

```

import 'dart:async';
import 'package:meta/meta.dart';

```

```

import 'package:patient/repositories/auth/auth_api_client.dart';
import 'package:patient/models/models.dart';

class AuthRepository {
  final AuthApiClient authApiClient;

  AuthRepository({@required this.authApiClient})
    : assert(authApiClient != null);

  Future<User> signup(String email, String password) async {
    return await authApiClient.signup(email: email, password: password);
  }

  Future<User> signin(String email, String password) async {
    return await authApiClient.signin(email: email, password: password);
  }

  Future<void> delete() async {
    return await authApiClient.deleteToken();
  }

  Future<void> persistToken(String token) async {
    return await authApiClient.persistToken(token);
  }

  Future<void> persistId(int id) async {
    return await authApiClient.persistId(id);
  }

  Future<void> persistInfo(User user) async {
    return await authApiClient.persistInfo(user);
  }

  Future<bool> hasToken() async {
    return await authApiClient.hasToken();
  }

  Future<User> getUser() async {
    return await authApiClient.getUser();
  }
}

```

Додаток В
Лістинг файлу LoginScreen.dart

```

import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:meta/meta.dart';
import 'package:flutter/material.dart';
import 'package:patient/blocs/blocs.dart';
import 'package:patient/repositories/repositories.dart';
import 'package:patient/widgets/Auth/LoginForm.dart';

class LoginScreen extends StatelessWidget {
  static const routeName = '/login';
  final AuthRepository authRepository;

  LoginScreen({Key key, @required this.authRepository})
    : assert(authRepository != null),
      super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).primaryColor,
      appBar: AppBar(
        elevation: 0,
        titleSpacing: 0,
        automaticallyImplyLeading: false,
        title: FlatButton.icon(
          label: Text(
            'Назад',
            style: TextStyle(
              color: Colors.white,
              fontSize: 16,
            ),
          ),
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ),
          onPressed: () => Navigator.of(context).pop(),
        ),
      ),
      body: BlocProvider(
        builder: (context) {
          return LoginBloc(
            authBloc: BlocProvider.of<AuthBloc>(context),
            authRepository: authRepository,
          );
        },
      ),
    );
  }
}

```

```

        child: SingleChildScrollView(
          child: Container(
            width: double.infinity,
            padding: EdgeInsets.fromLTRB(15, MediaQuery.of(context).size.height * 0.05,
15, 15),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Text(
                  'Диабет привіт',
                  style: TextStyle(
                    fontSize: 42,
                    fontWeight: FontWeight.bold,
                    color: Colors.white,
                  ),
                ),
                LoginForm()
              ],
            ),
          ),
        ),
      ),
    );
  }
}

```

Додаток Г

Лістинг файлу AuthMainScreen.dart

```

import 'package:flutter/material.dart';
import 'package:patient/repositories/repositories.dart';
import 'package:patient/screens/Auth/LoginScreen.dart';
import 'package:patient/screens/Auth/RegistrationScreen.dart';
import 'package:patient/widgets/CommonButton.dart';

class AuthMainScreen extends StatelessWidget {
  static const routeName = '/auth';
  final AuthRepository authRepository;

  AuthMainScreen({Key key, @required this.authRepository})
    : assert(authRepository != null),
      super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(

```



```

backgroundColor: Theme.of(context).primaryColor,
body: Container(
  width: double.infinity,
  padding: EdgeInsets.only(top: MediaQuery.of(context).size.height * 0.16),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.center,
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: <Widget>[
      Text(
        'DiaBET',
        style: TextStyle(
          fontSize: 64,
          fontWeight: FontWeight.w900,
          color: Colors.white,
        ),
      ),
      Container(
        padding: EdgeInsets.fromLTRB(15, 0, 15, 15),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            Padding(
              padding: EdgeInsets.only(bottom: 5),
              child: CommonButton(
                onPressed: () => Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) =>
LoginScreen(authRepository: authRepository,))
                ),
                text: 'Войти',
                isPrimary: true,
              ),
            ),
            CommonButton(
              onPressed: () => Navigator.push(
                context,
                MaterialPageRoute(builder: (context) =>
RegistrationScreen(authRepository: authRepository,))
              ),
              text: 'Зарегистрироваться',
            ),
          ],
        ),
      ),
    ],
  ),
)

```

```

    );
  }
}

```

Додаток Д

Лістинг файлу RegistrationScreen.dart

```

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:patient/blocs/blocs.dart';
import 'package:patient/repositories/repositories.dart';
import 'package:patient/widgets/Auth/RegistrationForm.dart';

class RegistrationScreen extends StatelessWidget {
  static const routeName = '/reg';
  final AuthRepository authRepository;

  RegistrationScreen({Key key, @required this.authRepository})
    : assert(authRepository != null),
      super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        backgroundColor: Colors.white,
        elevation: 0,
        titleSpacing: 0,
        automaticallyImplyLeading: false,
        title: FlatButton.icon(
          label: Text(
            'Назад',
            style: TextStyle(
              color: Theme.of(context).primaryColorDark,
              fontSize: 16,
            ),
          ),
          icon: Icon(
            Icons.arrow_back,
            color: Theme.of(context).primaryColorDark,
          ),
          onPressed: () => Navigator.of(context).pop(),
        ),
      ),
      body: BlocProvider(
        builder: (context) {

```

```

        return LoginBloc(
          authBloc: BlocProvider.of<AuthBloc>(context),
          authRepository: authRepository,
        );
      },
      child: SingleChildScrollView(
        child: Container(
          width: double.infinity,
          padding: EdgeInsets.fromLTRB(15, MediaQuery.of(context).size.height * 0.05,
15, 15),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              Text(
                'Регистрация',
                style: TextStyle(
                  fontSize: 42,
                  fontWeight: FontWeight.bold,
                  color: Theme.of(context).primaryColor,
                ),
              ),
              RegistrationForm(),
            ],
          ),
        ),
      ),
    ),
  );
}
}

```

Додаток Е

Лістинг файлу LoginForm.dart

```

import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:patient/blocs/blocs.dart';
import 'package:patient/widgets/CommonButton.dart';

class LoginForm extends StatefulWidget {
  @override
  _LoginFormState createState() => _LoginFormState();
}

class _LoginFormState extends State<LoginForm> {

```

```

final _emailController = TextEditingController();
final _passwordController = TextEditingController();
final _form = GlobalKey<FormState>();
FocusNode _emailFocus;
FocusNode _passwordFocus;
bool _passwordVisible;
Map<String, String> _authData = {'email': '', 'password': ''};

@override
void initState() {
  super.initState();

  _emailFocus = new FocusNode();
  _passwordFocus = new FocusNode();
  _passwordVisible = false;
}

@override
void dispose() {
  _emailFocus.dispose();
  _passwordFocus.dispose();

  super.dispose();
}

@override
Widget build(BuildContext context) {
  Orientation orientation = MediaQuery.of(context).orientation;
  __onLoginButtonPressed() {
    if (!_form.currentState.validate()) {
      return;
    }
    _form.currentState.save();
    BlocProvider.of<LoginBloc>(context).add(
      LoginButtonPressed(
        email: _authData['email'],
        password: _authData['password'],
      )
    );
  }
  return BlocListener<LoginBloc, LoginState>(
    listener: (context, state) {
      if (state is LoginFailure) {
        Scaffold.of(context).showSnackBar(
          SnackBar(
            content: Text('${state.error}'),

```

```

        backgroundColor: Colors.red,
      ),
    );
  }
  if (state is LoginInitial) {
    Navigator.of(context).pop();
  }
},
child: BlocBuilder<LoginBloc, LoginState>(
  builder: (context, state) {
    return Form(
      key: _form,
      child: SingleChildScrollView(
        child: Container(
          height: orientation == Orientation.landscape
            ? MediaQuery.of(context).size.height * 0.8
            : MediaQuery.of(context).size.height * 0.7,
          padding: EdgeInsets.fromLTRB(
            0, (MediaQuery.of(context).size.height * 0.6) * 0.14, 0, 5),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: <Widget>[
              Container(
                child: Column(
                  children: <Widget>[
                    TextFormField(
                      focusNode: _emailFocus,
                      style: TextStyle(
                        fontSize: 16,
                        color: Colors.white,
                      ),
                      decoration: InputDecoration(
                        contentPadding: EdgeInsets.symmetric(vertical: 10),
                        labelText: 'Baw email',
                        labelStyle: TextStyle(
                          color: _emailFocus.hasFocus
                            ? Color.fromRGBO(255, 255, 255, 0.6)
                            : Colors.white,
                          fontSize: 18),
                        enabledBorder: UnderlineInputBorder(
                          borderSide: BorderSide(
                            color: Colors.white,
                          ),
                        ),
                        focusedBorder: UnderlineInputBorder(
                          borderSide: BorderSide(

```

```
color: Colors.white,
    ),
  ),
),
textInputAction: TextInputAction.next,
controller: _emailController,
onFieldSubmitted: (_) {
  FocusScope.of(context).requestFocus(_passwordFocus);
},
onSaved: (value) {
  _authData['email'] = value;
},
validator: (value) {
  if (value.isEmpty || !value.contains('@')) {
    return 'Invalid email!';
  }
},
),
),
SizedBox(
  height: 30,
),
TextFormField(
  focusNode: _passwordFocus,
  obscureText: !_passwordVisible,
  style: TextStyle(
    fontSize: 16,
    color: Colors.white,
  ),
  decoration: InputDecoration(
    contentPadding: EdgeInsets.symmetric(vertical: 10),
    labelText: 'Ваш пароль',
    labelStyle: TextStyle(
      color: _passwordFocus.hasFocus
        ? Color.fromRGB(255, 255, 255, 0.6)
        : Colors.white,
      fontSize: 18),
    enabledBorder: UnderlineInputBorder(
      borderSide: BorderSide(
        color: Colors.white,
      ),
    ),
    focusedBorder: UnderlineInputBorder(
      borderSide: BorderSide(
        color: Colors.white,
      ),
    ),
    suffixIcon: IconButton(
```

```

        icon: Icon(
          _passwordVisible
            ? Icons.visibility
            : Icons.visibility_off,
          color: Colors.white,
        ),
        onPressed: () {
          _passwordFocus.unfocus();
          setState(() {
            _passwordVisible = !_passwordVisible;
          });
        },
      ),
    ),
    controller: _passwordController,
    validator: (value) {
      if (value.isEmpty || value.length < 5) {
        return 'Invalid password!';
      }
    },
    onSave: (value) {
      _authData['password'] = value;
    },
  ),
  SizedBox(
    height: 10,
  ),
  Row(
    children: <Widget>[
      Text(
        'Забыли пароль?',
        style: TextStyle(
          fontSize: 16,
          color: Color.fromRGBO(255, 255, 255, 0.6),
        ),
      ),
      FlatButton(
        color: Colors.transparent,
        child: Text(
          'Восстановить',
          style: TextStyle(
            fontSize: 16,
            color: Color.fromRGBO(255, 255, 255, 0.9)),
        ),
        onPressed: () {},
      )
    ],
  ),
],

```

```

        ),
      ],
    ),
  ),
  CommonButton(
    onPressed: state is! LoginLoading ? __onLoginButtonPressed : null,
    text: 'Войти',
    isPrimary: true,
  ),
],
),
),
),
),
);
},
),
);
}
}

```