Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

На правах рукопису

УДК 004.42

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: “Розробка мікросервісного ПЗ для обробки медичних даних методом багатокрокової генерації з предметно-орієнтованої моделі”
Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент
_____Олексій БОЯРСЬКИЙ_____
(підпис) (розшифровка підпису) (дата)

Науковий керівник
доц._____Світлана ПОПЕРЕШНЯК_____
(посада) (підпис) (розшифровка підпису) (дата)

Допускається до захисту
з питань нормоконтролю
Завідувач кафедри

_____Олексій БИЧКОВ_____
(підпис) (розшифровка підпису) (дата)

Київ 2022

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

Олексія БОЯРСЬКОГО

захищена з оцінкою

_____

Голова Екзаменаційної комісії №2

професор, доктор техн. наук Андрій БОНДАРЧУК

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
_____(Олексій БИЧКОВ)
“___”_____ 20___р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Боярському Олексію Володимировичу

(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної магістерської роботи: “Розробка мікросервісного ПЗ для обробки медичних даних методом багатокрокової генерації з предметно-орієнтованої моделі”

затверджена наказом вищого навчального закладу від „21” грудня 2021р. №8

2. Строк здачі студентом закінченої роботи 10 травня 2022р.
3. Вихідні дані до роботи

   Дані про принципи проектування й конструювання генераторів коду; засади реалізації мікросервісного ПЗ; положення клінічних протоколів та варіанти їх використання на рівні клієнта за класифікацією ВООЗ; макет графічного користувацького інтерфейсу.
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
   - Визначення мету, цілі, задачі та коло проблемних питань до вирішення
   - Аналіз існуючих рішень на рівні клієнта, виявлення переваг та недоліків
   - Визначення вимог до розроблюваного ПЗ для обробки медичних даних
   - Проектування архітектури та генератора мікросервісного ПЗ
   - Розробка генератора мікросервісного ПЗ у складі системи обробки медчних даних, демонстрація можливостей та проведення розгортання й тестування
   - Визначення практичної цінності розробленого ПЗ та шляхів його покращення.
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)
   - Діаграма варіантів використання кінцевим користувачем
   - Діаграма компонентів системи
   - Діаграма розгортання
   - Схема процесу неперервної інтеграції
   - Схема процесу покрокової генерації з предметно-орієнтованої моделі.

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1. Аналіз існуючих рішень	Світлана ПОПЕРЕШНЯК	22.12.2021	14.01.2022
Розділ 2. Специфікація вимог та опис архітектури	Світлана ПОПЕРЕШНЯК	15.01.2022	09.02.2022
Розділ 3. Розробка генератора та мікросервісного ПЗ	Світлана ПОПЕРЕШНЯК	10.02.2022	19.04.2022
Розділ 4. Тестування й розгортання мікросервісного ПЗ	Світлана ПОПЕРЕШНЯК	20.04.2022	05.05.2022

7. Дата видачі завдання 22 грудня 2021р.

Керівник _____ /Світлана ПОПЕРЕШНЯК
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ /Олексій БОЯРСЬКИЙ/
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів магістерської роботи	Термін виконання етапів роботи	Примітка
Підбір і вивчення літератури по мікросервісному ПЗ й генерувальним підходам	28.12.2021	виконано
Аналіз існуючих рішень на рівні клієнта для моніторингу стану здоров’я	5.01.2022	виконано
Вивчення можливостей відображення клінічних протоколів на предметно-орієнтовану модель	15.01.2022	виконано
Проектування генератора мікросервісного ПЗ	03.02.2022	виконано
Тестування генератора й розгортання отриманого ПЗ	20.04.2022	виконано
Аналіз отриманих результатів, визначення шляхів подальшого розвитку	05.05.2022	виконано

Студент – магістр _____ /Олексій БОЯРСЬКИЙ/
(підпис) (розшифровка підпису)

Керівник роботи _____ /Світлана ПОПЕРЕШНЯК/
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Випускна кваліфікаційна магістерська робота: 84 с., 24 рис., 20 джерел, 7 додатків.

Тема: “Розробка мікросервісного ПЗ для обробки медичних даних методом багатокрокової генерації з предметно-орієнтованої моделі”.

Об'єкт дослідження – програмне забезпечення первинної профілактичної медицини.

Мета роботи – розробка мікросервісного ПЗ, що надає рекомендацій для покращення стану здоров’я на основі зібраних медико-біологічних параметрів.

Предмет дослідження – генерація ПЗ з предметно-орієнтованої моделі для обробки даних медичного призначення на рівні клієнта.

Результати дослідження: Визначено переваги та недоліки існуючого ПЗ, а також вимоги до створюваної мікросервісної програмної системи. Здійснено проектування й розробку предметно-орієнтованої мови програмування для опису у вигляді моделей профілактичних положень клінічних протоколів медичним персоналом. Розроблено генератор мікросервісного ПЗ, який на основі предметно-орієнтованої моделі створює компонент системи, який використовується для надання рекомендацій користувачам.

Висновок: Упровадження розробленого мікросервісного ПЗ дозволяє користувачам відслідковувати власні медико-біологічні параметри та отримувати рекомендації, щодо покращення стану здоров’я на основі положень клінічних протоколів. Також може застосовуватись як платформа для незалежних експериментів, що фокусуються на вдосконалення та розширення профілактичних рекомендацій за допомогою розробленої предметно-орієнтованої мови програмування для можливості швидкої демонстрації чи отриманих результатів.

ОБРОБКА МЕДИЧНИХ ДАНИХ, МЕДИКО-БІОЛОГІЧНІ ПАРАМЕТРИ, ПРЕДМЕТНО-ОРІЄНТОВАНА МОВА ПРОГРАМУВАННЯ, ГЕНЕРАЦІЯ МІКРОСЕРВІСНОГО ПЗ.

ANNOTATION

Final qualification master's thesis: 84 pages., 24 figures., 20 sources, 7 attachments.

Topic: "Development of microservice software for processing medical data rusing multi-step generation from the domain-specific model."

The object of study - the software of primary preventive medicine.

The aim of the work is to develop micro-service software that provides recommendations for improving health based on the collected medical and biological parameters.

The subject of research is the generation of software from a subject-oriented model for processing medical data at the client level.

Research results: The advantages and disadvantages of existing software are identified, as well as the requirements for the created microservice software system. The design and development of DSL for description in the form of models of preventive provisions of clinical protocols by medical staff. A microservice software generator has been developed, which creates a component of the system based on a domain-specific model, which is used to provide recommendations to users.

Conclusion: The implementation of the developed microservice software allows users to monitor their own medical and biological parameters and receive recommendations for improving health based on the provisions of clinical protocols. It can also be used as a platform for independent experiments focusing on improving and expanding prevention recommendations using the developed DSL to enable rapid demonstration of results.

MEDICAL DATA PROCESSING, MEDICAL AND BIOLOGICAL PARAMETERS, DOMAIN-SPECIFIC PROGRAMMING LANGUAGE, GENERATION OF MICROSERVICE SOFTWARE.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ

HDSL	Розроблена в рамках роботи предметно-орієнтована мова програмування
MVC	Model-View-Controller, архітектурний патерн, який розділяє бізнес логіку, зберігання даних та відображення користувачу
PWA	Progressive Web Application, може встановлюватись на платформах, де наявний веб-браузер як самостійний застосунок й працювати offline
REST	Representational State Transfer, підхід до реалізації веб програмного інтерфейсу
МБП	Медико-біологічні параметри, наприклад пульс, артеріальний тиск та ін

ВСТУП

За останні роки з’явилось багато пристроїв носимої електроніки, які дозволяють вимірювати тиск та пульс. Також пристрої для вимірювання рівня глюкози, гемоглобіну та ін. показників крові стали більш доступнішими не тільки за ціною, а й можливістю придбання звичайними людьми, а не медустановами. Загалом рівень обізнаності людей про своє здоров’я зріс, а здоровий спосіб життя в екологічно чистому середовищі зі здоровим харчуванням турбує дедалі більшу частку населення. Тому розробка програмного забезпечення, що дозволить проводити збір медико-біологічних параметрів й аналіз з надаванням рекомендацій щодо покращення стану здоров’я, є актуальним.

Процес збору даних медичного призначення досліджувався Прасол І. В., Єрошенко О. А. [1], процес обробки даних лікарями в спеціальних пакетах програм розглядався Гойко О. В. [2], проблеми обробки первинних медико-біологічних даних пацієнтів сімейним лікарем досліджувалися Сітнікова О. А., Почебут М. В. [3], застосування предметно-орієнтованої мови програмування для запитів до даних медичного призначення з БД описано в патенті[4], де вона пропонується як заміна SQL. У дослідженні[5] Llahm Omar Ben Dalla пропонує мову моделювання, що специфічна для предметної області, для представлення медичних рецептів, потоку рецепту між різними користувачами та різними частинами системи. Stefan Kraus[6] в своїй статті узагальнює концепти й конструкції мови розмітки медичних даних Arden Syntax, щоб отримати можливість її використання у клінічних алгоритмах.

Проте вищезгадані роботи не розглядали можливість збору медико-біологічних параметрів на рівні клієнта й їх аналізу на основі профілактичних положень клінічних протоколів, що дасть змогу виявляти захворювання ще до візиту до лікаря.

Об’єктом дипломної роботи є програмне забезпечення первинної профілактичної медицини.

Предмет дослідження – генерація ПЗ з предметно-орієнтованої моделі для обробки даних медичного призначення на рівні клієнта.

Мета дослідження – розробка мікросервісного ПЗ, що надає рекомендацій для покращення стану здоров’я на основі зібраних медико-біологічних параметрів.

В процесі дослідження вирішувались наступні завдання:

- аналіз існуючого програмного забезпечення;
- формування вимог до програмної системи;
- проектування й розробка компонентів та генератора;
- тестування програмної системи та генератора;
- розгортання розробленої системи;
- визначення необхідних заходів для подальшого вдосконалення й розвитку ПЗ.

В роботі застосовувалися евристичні методи інженерії програмного забезпечення, а саме: методи об’єктно-орієнтованого проектування з використанням UML; методи об’єктно-орієнтованого та генерувального програмування.

Наукова новизна справжнього дослідження полягає в наступному:

- надання профілактичних рекомендацій без залучення медичного персоналу;
- створення предметно-орієнтованої мови програмування для опису профілактичних положень клінічних протоколів;
- генерація готового до інтеграції мікросервісного ПЗ з предметно-орієнтованої моделі.

Розроблюване ПЗ може використовуватись користувачами для збору й обробки власних медико-біологічних параметрів. Також можливе використання предметно-орієнтованої мови медичним персоналом шляхом додавання нових моделей профілактичних положень клінічних протоколів для підвищення якості проведеного аналізу медико-біологічних параметрів.

За темою роботи було опубліковано статтю в збірнику матеріалів конференції ISDMCI 2021[7], що проіндексована науко-метричною базою Scopus. Також були опубліковані тези на Всеукраїнській науково-технічній конференції «Застосування

програмного забезпечення в інфокомунікаційних технологіях»[8] та на Східно-Європейській конференції “Математичні та програмні технології Internet of Everything”[9].

За темою магістерської роботи була оформлена конкурсна робота, яка прийняла участь у другому турі Всеукраїнського конкурсу студентських наукових робіт з природничих, технічних та гуманітарних наук у 2020-2021 навчальному році в галузі «Біомедична інженерія», де виборола **перше** місце (див. Додаток А).

Результати роботи були впроваджені на 2-х підприємствах, що підтверджується відповідними актами наведеними у додатку Б.

Відповідно до завдання дипломної роботи необхідно розробити мікросервісне ПЗ для обробки медичних даних шляхом генерації з предметно-орієнтованої моделі.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Класифікація додатків в галузі охорони здоров’я

Застосування інформаційних технологій в галузі охороні здоров’я значно збільшилось за останні роки в України та за її межами. На теперішній час на ринку представлено багато мобільних додатків та веб-застосунків у сфері охорони здоров’я, тому постала необхідність чітко класифікувати такі додатки, а також сформулювати основні проблеми, які вони вирішують.

Сфера охорони здоров’я традиційно вважається доволі консервативною та повільною в процесах адаптації нових технологій, адже будь-яка помилка може мати необернені наслідки для життя і здоров’я пацієнтів. Проте в цифрову епоху – проникнення ІТ в будь-яку сферу людської життєдіяльності є невідворотним, оскільки тільки так можна досягти подальшого розвитку інформаційного суспільства. Класифікація втручань цифрового здоров'я (англ. DHI – Digital Health Intervention) була створена Всесвітньою організацією охорони здоров’я (англ. WHO – World Health Organization) саме для вирішення цього питання.

ІТ згідно DHI WHO використовуються на чотирьох рівнях: рівень клієнта; рівень провайдерів медичних послуг; рівень системи охорони здоров’я; рівень сервісів даних [10].

Для реалізації поставленого завдання розроблюване ПЗ буде відноситися до рівня клієнта, а саме до підпункту 1.4 Personal Health Tracking (укр. Персональне відслідковування здоров’я), що зображено на рисунку 1.1. Таким чином користувач зможе самостійно слідкувати за станом свого здоров’ям.

Оскільки користувачі даного ПЗ у більшості випадків не матимуть спеціалізованої освіти та складного обладнання для аналізу свого стану здоров’я, то найбільш раціональним і зручним механізмом особистого моніторингу можна вважати дані медико-біологічних параметрів (далі – МБП) користувача, які можна з

легкістю виміряти навіть у побутових умовах, або вони є очевидними. Мова йде про такі показники, як: пульс, артеріальний тиск, температура, стать, вік та ін.

Рис. 1.1 Рівень клієнта класифікації DHI [10]

Пропонована система не є лише електронним щоденником, в який користувач вносить дані вимірювань своїх показників. Це програмне забезпечення також буде надавати функцію аналізу біологічних показників користувача за заданими періодами часу відповідно до клінічних протоколів, що надасть можливість вчасно ідентифікувати потенційні захворювання та провести необхідні профілактичні чи лікувальні заходи.

1.2 Медико-біологічні показники моніторингу

Згідно зі звітом Всесвітньої організації охорони здоров’я за 2018 рік в Україні смертність від неінфекційних хвороб склала близько 91% від загальної кількості смертей [11]. До неінфекційних хвороб відносять: серцево-судинні захворювання, рак, хронічні респіраторні захворювання та ін. Відсоток смертності від цих захворювань зображено у вигляді кругової діаграми на рисунку 1.2.

Серед основних факторів ризику передчасної смерті виділяють паління, ожиріння, малорухливий спосіб життя, а також підвищений артеріальний тиск.

Оскільки смертність від захворювань серцево-судинної системи є найбільшою й враховуючи фактори ризику, було вирішено, що в розроблюваному ПЗ користувачу буде запропоновано відслідковувати такі МБП: артеріальний тиск, пульс, вага, температура, рівень глюкози в крові.

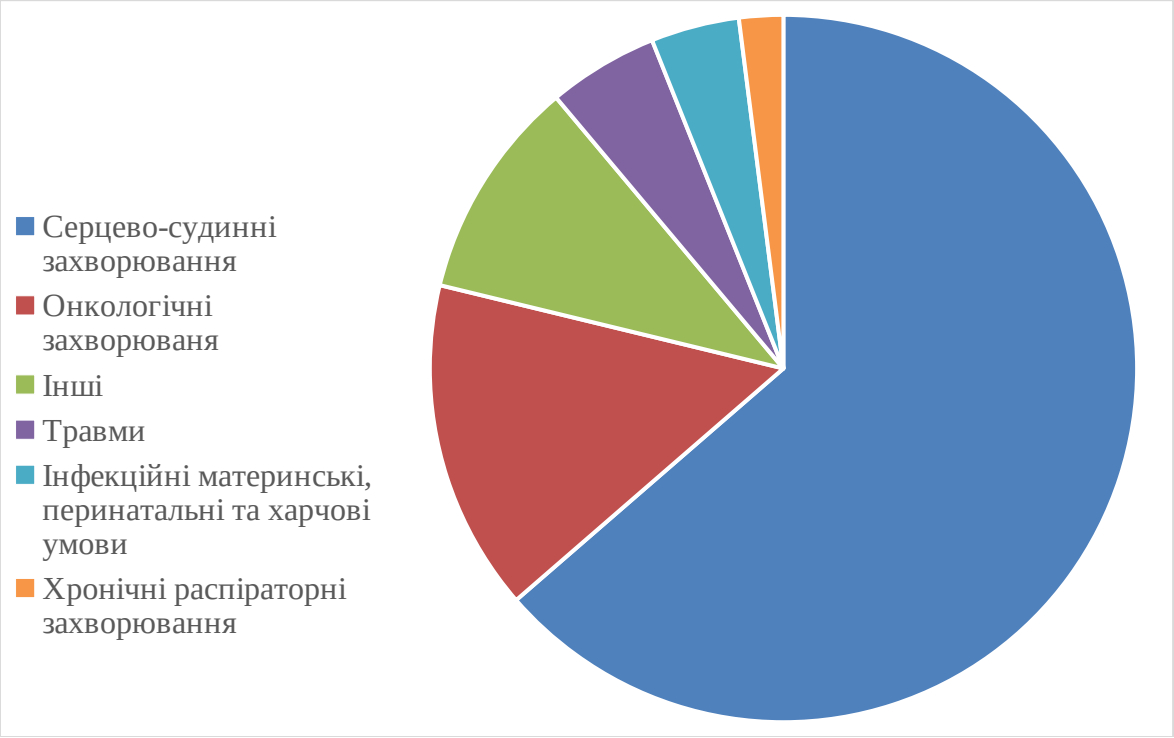


Рис. 1.2 Діаграма рівня смертності від неінфекційних захворювань

Дані вимірювань МБП артеріального тиску детально характеризуються клінічною настановою №00069 «Гіпертензія: обстеження та стартове лікування» [12]. Згідно з приведеною настановою артеріальний тиск вважається вище за допустимий, якщо систолічний артеріальний тиск вище 140, а диастолічний артеріальний тиск вище 90 міліметрів ртутного стовпця.

Дані вимірювань МБП частоти серцевих скорочень характеризується одразу декількома клінічними настановами [13][14]. Це пов’язано в першу чергу з тим, що значення пульсу однозначно не може свідчити про наявність певного захворювання без проведення аналізу електрокардіограми. Оскільки лікування й точне встановлення діагнозу не передбачається в рамках розробки мікрсервіного ПЗ, то

порогом тахікардії будемо вважати пульс, значення якого досягає більше 100 ударів в хвилину (нижній поріг синусової та шлуночкової тахікардій).

Для визначення нормальної маси тіла прийнято використовувати індекси маси тіла (ІМТ). Щоб визначити нормальну масу тіла, буде використовуватись ІМТ, розроблений бельгійським статистиком Адольфом Кетле. Формула розрахунку приведена нижче:

$$I = \frac{m}{h^2} \qquad (1.1)$$

де: m – маса тіла, кг

h – зріст, м

I – індекс маси тіла, кг/м^2

Саме застосування цієї формули передбачається клінічною настановою для оцінки пацієнтів з ожирінням [15].

Після визначення індексу маси тіла необхідно інтерпретувати його за таблицею 1.1.

Таблиця 1.1

Індекс маси тіла та його інтерпретація

Індекс	Клас
<18.5	Дефіцит маси тіла
18.5-25	Норма
25-30	Надлишкова маса тіла
30-35	Ожиріння 1 ступеня
35-40	Ожиріння 2 ступеня
>40	Ожиріння 3 ступеня

Індекс маси тіла не є абсолютно точним, бо не враховує вік, расу, статуру та ін. Тому як альтернативу було розроблено компанією Select Research індекс об’єму тіла, який дає конкретні дані про ризики ожиріння для кожного пацієнта, а також враховує

розподіл маси тіла (кількість кісток чи м’язів). Для застосування даного метода використовується тривимірний сканер.

Індекс Кетле було обрано як метод оцінки маси тіла, який застосовується найширше в медичній практиці та не потребує додаткового обладнання для калькуляції.

Значення вимірювання МБП температури тіла коливається, коли в організмі людини розвиваються інфекції, запальні процеси та великий спектр інших хвороб. Аналіз температури в відриві від наявних симптомів у людини не зможе дати значимих результатів, тому в розроблюваному ПЗ такий аналіз проводиться не буде. Доцільність такого рішення також підтверджується ще й тим, що не було знайдено клінічних протоколів, на основі положень яких це можна було б реалізувати.

Коливання рівня глюкози в крові може бути причиною розвитку багатьох захворювань, але серед найбільш поширеним є цукровий діабет, що одночасно є фактором ризику розвитку інших неінфекційних захворювань, наприклад серцево-судинних. Значення рівня глюкози в крові натще характеризується клінічною настановою 00486 [16], що й буде використовуватись в розроблюваному ПЗ для аналізу.

1.3 Порівняльна характеристика існуючих додатків

На теперішній момент існує низка мобільних додатків, використовуючи які можна відстежувати деякі показники свого здоров’я. Це як і звичайні журнали, так і повноцінні додатки з функціями контролю активності, споживання їжі та води. Розглянемо характеристики кожного з них та проведемо їх порівняльну характеристику. Всі наведені нижче додатки були завантажені з сервісу Google Play та встановлені на смартфон під керуванням операційної системи Android 10.

Розглянемо додаток Health Infinity. Адреса для завантаження – https://play.google.com/store/apps/details?id=com.droidinfinity.healthplus.

Додаток підтримує реєстрацію за допомогою облікового запису Google та за допомогою адреси електронної пошти та пароля. Для початку використання потрібно заповнити профіль користувача.

Доступ до більшості функцій здійснюється з основного екрану додатку, де можна отримати інструкції зі здорового харчування та фізичних вправ. Основний екран додатку приведено на рисунку 1.3а. Додаток дозволяє записувати дані вимірювань пульсу та ваги. Екран додавання ваги зображено на рисунку 1.4б. Для виміряти пульсу необхідно використовувати камеру смартфона, на об’єктив якої помістити палець. Це дозволяє провести вимірювання без додаткових пристроїв, проте не є точним: в тестовому випадку похибка склала 40 ударів за хвилину. З розповсюдженням оптичних пульсометрів, якими обладнані майже всі смартбраслети та смартгодинники, така функція є більше засобом розваги, чим вирішенням задачі вимірювання.

Загалом додаток можна охарактеризувати як аналог Google Fit, Samsung Health або Mi Fit з деякими додатковими опціями як таблиця витрати калорій за активностями, рекомендований денний рівень споживання калорій, води та ін.

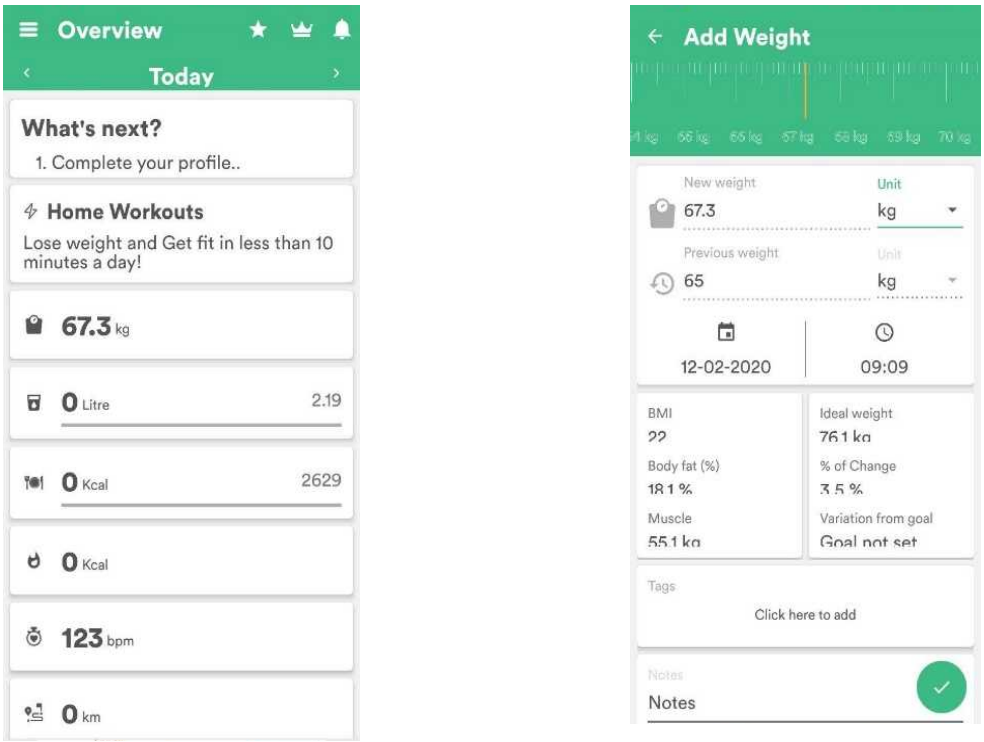


Рисунок 1.3 а) Основий екран. б) Екран додавання ваги

Основні недоліки додатку Health Infinity:

- необхідність встановлення сторонніх додатків для використання всіх заявлених функцій – отримувати рекомендації щодо фізичних вправ допускається за умови встановлення окремого додатоку.

- велика кількість реклами, яка доволі часто з’являється одразу на весь екран, що суттєво впливає на позитивний досвід використання.

Розглянемо наступний додаток Health Diary. Адреса для завантаження – https://play.google.com/store/apps/details?id=com.ahssoft.healthdiary.

Він дозволяє моніторити рівень глюкози в крові, артеріальний тиск та пульс.

Додаток доступний без реєстрації і зберігає інформацію локально на смартфоні, тому при його перевстановленні чи зміні смартфона вся введена інформацію буде втрачена.

З основного екрану, що зображено на рисунку 1.4а, можна записувати вимірювання пульсу та артеріального тиску. Створення нового запису приведено на рисунку 1.4б.

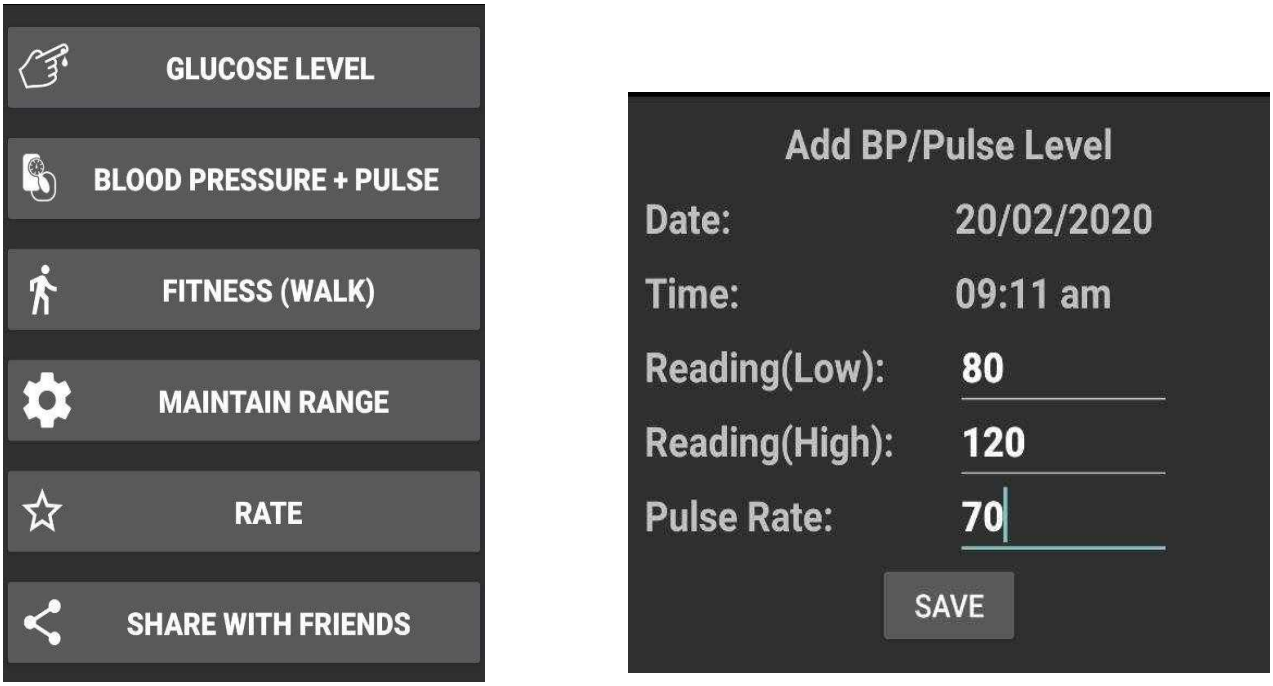


Рис. 1.4 а) Основний екран. б) Додавання вимірювання пульсу та артеріального тиску

Після внесення даних доступна можливість перегляду у вигляді графіка, який показує динаміку зміни значень вимірювань. Верхня та нижня межі значень будь-якого вимірювання налаштовуються також з основного екрану.

Основні недоліки додатку Health Diary:

- відсутність підказок при встановленні верхніх, нижніх допустимих границь для значень вимірювань глюкози, тиску та пульсу;
- застарілий користувацький інтерфейс;
- відсутність можливості експортувати дані для переносу їх на інший пристрій.

Розглянемо наступний додаток I Health Diary. Адреса для завантаження – https://play.google.com/store/apps/details?id=com.eserhealthcare.ihealthapp.

Для початку користування передбачається створення облікового запису, де необхідно вказати логін, пароль, вік, вагу, зріст та стать. Доступ до основних функцій здійснюється з основного екрану, де відображаються дані профілю, кнопки переходу до вимірювання артеріального тиску, пульсу, глюкози, гемоглобіну. Основний екран додатку зображено на рисунку 1.5а.

Вимірювання всіх доступних МБП здійснюється приладом, який необхідно придбати в спеціалізованому магазині та під’єднати за допомогою Bluetooth. Здійснити вимірювання без прилада не можливо. Перегляд історії вимірювань здійснюється у вигляді графіка на екрані, що приведено на рисунку 1.5б.

Основні недоліки додатку I Health Diary:

- відсутність можливості вносити дані вимірювань вручну або з застосуванням вимірювальних приладів інших виробників;
- відсутність підказок при встановленні допустимих меж показників вимірювання;
- відсутність нагадувань про вимірювання.

Розглянемо додаток Diabetes Journal. Адреса для завантаження – https://play.google.com/store/apps/details?id=com.suderman.diabeteslog.

Додаток націлений на людей, хворих цукровим діабетом, проте й для інших він може запропонувати вимірювання багатьох інших показників, окрім глюкози, наприклад: артеріальний тиск, пульс, рівень гемоглобіну.

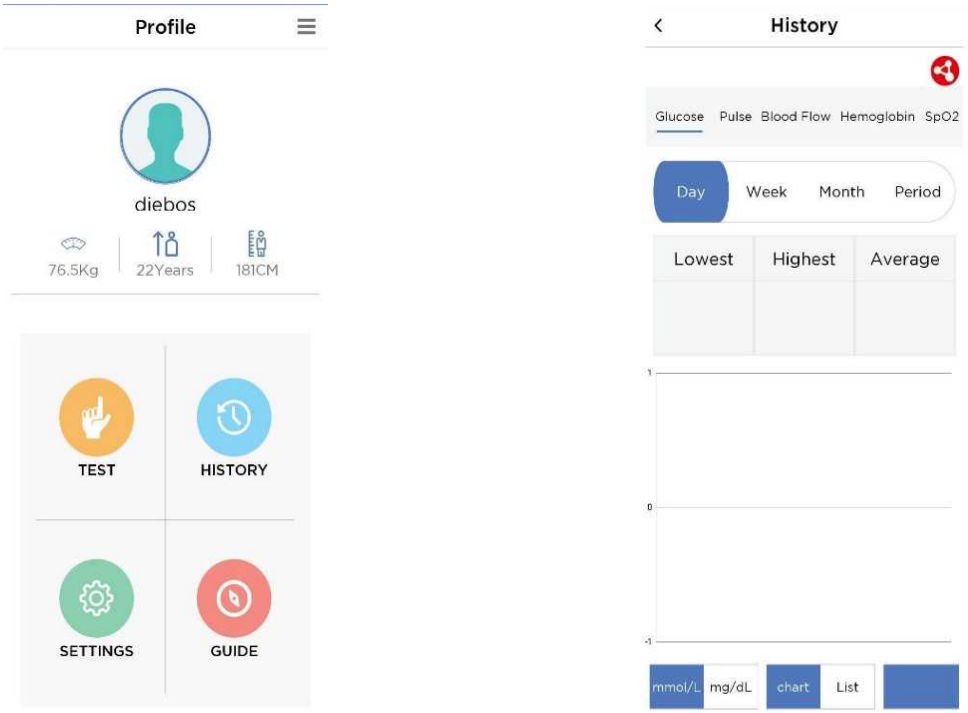


Рис. 1.5 а) Основний екран б) Екран перегляду історії вимірювань

Для початку роботи необхідно створити новий профіль по аналогії з розглянутими вище додатками. Для хворих діабетом є можливість вказати стадію хвороби. Додаток підтримує декілька профілів, між якими можна здійснювати перехід. На основному екрані виводяться дані про останні здійснені вимірювання, як показано на рисунку 1.6а. Додаток надає можливості перегляду даних вимірювань у вигляді графіка, а також – створення нагадування про необхідність провести вимірювання. Перегляд даних вимірювань в хронологічній послідовності приведено на рисунку 1.6б. Додаток підтримує можливість експорту даних в формат CSV для кожного з профілів окремо.

Основні недоліки додатку Diabetes Journal:

- наявність реклами в нижній частині кожного екрану;

- спрямованість на людей хворих на цукровий діабет, що в більшості випадків зробить майже не можливим потенційне використання додатку, зважаючи на його назву.

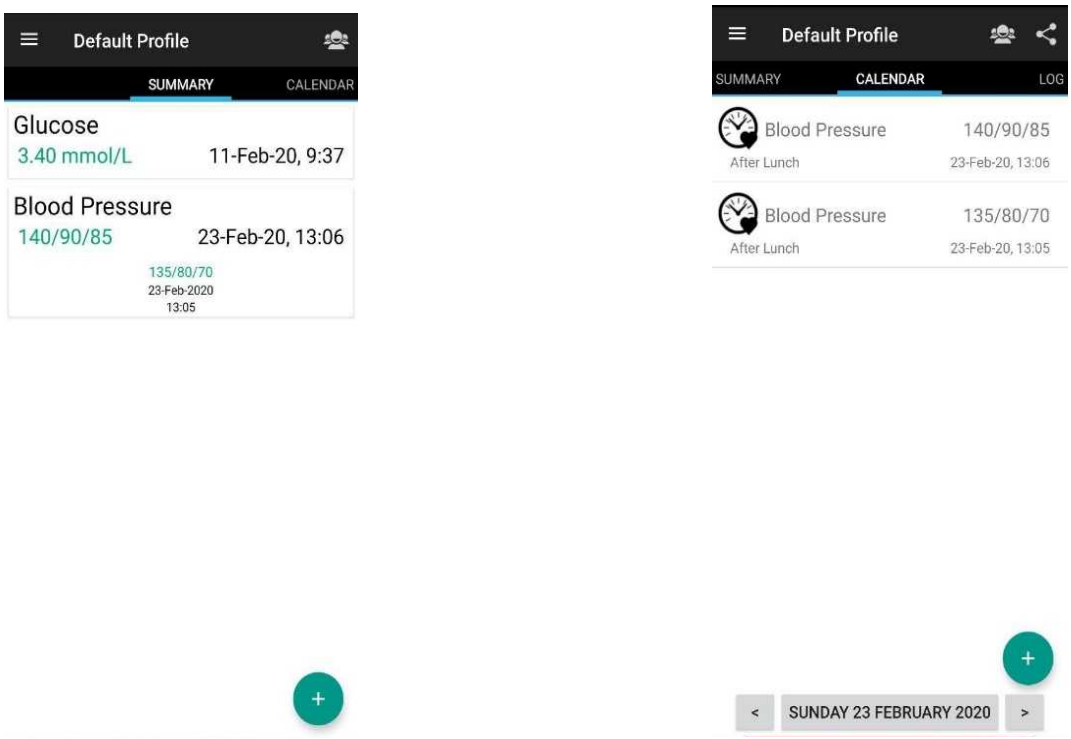


Рис. 1.6 а) Основний екран. б) Екран перегляду даних вимірювань

Узагальнення аналізу розглянутих додатків в розрізі підтримуваних вимірювань МБП подано в таблиці 1.2.

Таблиця 1.2

Порівняльна характеристика аналогів за кількістю вимірюваних показників

Додаток	Глюкоза	Гемоглобін	Пульс	Артеріальний тиск	Вага
Health Infinity	-	-	+	-	+
Health Diary	+	-	+	+	-
I Health Diary	+	+	+	+	+
Diabetes Journal	+	-	+	+	-

Враховуючи дані проведеного аналізу вважається за доцільне надати користувачеві можливість відслідковувати такі показники як: пульс, артеріальний тиск, вагу, рівень глюкози. Рівень гемоглобіну вимірювався тільки в одному додатку,

тому в пропонованій системі відслідковування цього показника буде недоступним. Натомість пропонується відслідковувати температуру тіла, адже її можна легко виміряти в домашніх умовах з високою точністю, а в періоди сезонного збільшення кількості респіраторних захворювань – ПЗ може слугувати пацієнтам для запису даних вимірювань температури з подальшим використанням сімейним лікарем для призначення чи корекції необхідного лікування.

Для забезпечення регулярності та змістовності даних МБП є доцільним забезпечити користувачів функцією зі створення нагадувань про необхідність здійснити вимірювання. Також слід забезпечити, у відповідності з міжнародним законодавством у сфері охорони даних медичного призначення, контроль доступу до даних, їх безпечне збереження й обробку з можливістю для власника цих даних запросити їх повне знищення чи видачу в обраному форматі.

1.4 Огляд приладів вимірювання МБП

Важливим аспектом при вимірюванні МБП є прилади, якими ці вимірювання здійснюються. Прилади можуть відрізнятися за рівнем точності, механізмом вимірювання та призначенням, але найголовнішою характеристикою в рамках даної роботи є здатність приладів до передачі даних вимірювань для їх зберігання та аналізу за допомогою ПЗ.

На даний момент серед приладів, що є у вільному доступі на ринку для приватних клієнтів, найпоширенішим протоколом бездротової передачі даних виступає Bluetooth. Цей протокол підтримується приладами компаній Beurer, Microlife, Omron та ін. Такі моделі в середньому коштують дорожче, але користувач отримує можливість встановити спеціальний додаток для синхронізації з приладом через Bluetooth, що дозволяє не вводити дані вимірювань вручну. Наприклад, у компанії Beurer є додаток для ОС Android «Beurer HealthManager». Він дещо схожий з тими додатками, що розглядалися в розділі 1.3, але відрізняється можливістю зчитування даних вимірювань через Bluetooth з сумісних пристроїв виробництва цієї компанії. Недоліком такого підходу для кінцевого користувача є недоступність

додатку на смартфонах під управлінням IOS та неможливість використання вимірювальних приладів інших компаній.

Прилади для вимірювання МБП виробляються й в Україні. Наприклад, Науково-технічний центр «ХАІ-Медика» пропонує комплексні рішення у сфері телемедицини, а також окремі прилади такі як реографи, електрокардіографи, холтеровські монітори. Ці прилади направлені більше на використання медичними закладами, про що свідчить велика кількість доступних функцій, застосування у складі єдиної телемедичної системи та висока ціна. Останнє робить прилади «ХАІ-Медика» майже недоступними для використання звичайними пацієнтами.

Для забезпечення максимального охоплення приладів вимірювання МБП необхідно використовувати програмні інтерфейси або бібліотеки, які забезпечать абстракцію над ними з метою створення універсального ПЗ. В процесі пошуку існуючих рішень було встановлено, що для цих цілей можна використовувати SDK, що розробляється компанією з США MedM. Оскільки ця компанія не надає прямого доступу до вихідного коду, то можна зробити висновок, що SDK може накладати ліцензійні обмеження таким чином, що буде необхідно або сплачувати ліцензійні платежі, або буде неможливо здійснити відкриту публікацію вихідного коду розроблюваного ПЗ. Ця думка підтверджується також й тим, що MedM надає своїм клієнтам послуги у сфері телемедицини. Тому в рамках даної роботи було вирішено не проводити інтеграцію з приладами вимірювання МБП, але це не виключає реалізації цієї функції в процесі подальшого розвитку.

1.5 Розробка вимог до ПЗ обробки медичних даних

Метою даної дипломної роботи є створення автоматизованої системи збору й обробки даних медичного призначення. Для цього в рамках роботи повинні бути вирішені наступні проблеми, які були виявлені в уже наявних аналогах:

- користувач повинен мати можливість вивантажити дані своїх медичних показників для використання в особистих цілях;
- користувач повинен мати можливість імпортувати раніше експортовані дані;

- для використання системи необхідно створити обліковий запис;
- записувати дані вимірювань можна вручну в зрозумілих одиницях вимірювання;
- дані вимірювань можна отримувати з Bluetooth-пристроїв;
- дані вимірювань можна переглядати у вигляді графіків;
- відсутність реклами.

Додатково пропонується реалізувати такі функції:

- аналіз даних вимірювань кожного користувача, використовуючи медичні протоколи для надання йому рекомендацій для покращення стану здоров’я;
- можливість входу в додаток за допомогою існуючого облікового запису Google, що значно зменшить час з моменту встановлення додатку до початку його використання;
- можливість створення нагадувань про необхідність здійснити вимірювання.

Код всіх програмних компонентів повинно бути опубліковано під ліцензією MIT, ща надасть можливості для інших розробників з адаптації та використання вихідного коду ПЗ в своїх проектах. Також дана ліцензія дозволяє публікацію та розповсюдження вихідного коду у складі комерційного програмного забезпечення зі збереженням авторського права. Важливою особливістю ліцензії MIT окрім відкритості є також те, що на розробника не покладається гарантійних обов’язків перед суб’єктами, що будуть використовувати вихідний код, а сам код надається у такому вигляді, в якому є (англ. as is). Для кінцевих користувачів передбачається надання одиничних невиключних ліцензій на використання ПЗ в особистих цілях без обов’язкових ліцензійних платежів.

Вимоги до розроблюваної системи представлені на діаграмі варіантів використання на рисунку 1.7.

Приведена діаграма варіантів використання відображає лише ті функції ПЗ, які важливі для кінцевого користувача. Оскільки для в процесі реалізації ПЗ виникла необхідність здійснювати моніторинг підсистем, то було сформовано вимоги до окремої підсистеми, якою будуть користуватися системні

адміністратори та інженери ПЗ, що й було відображено діаграмою варіантів використання на рисунку 1.8.

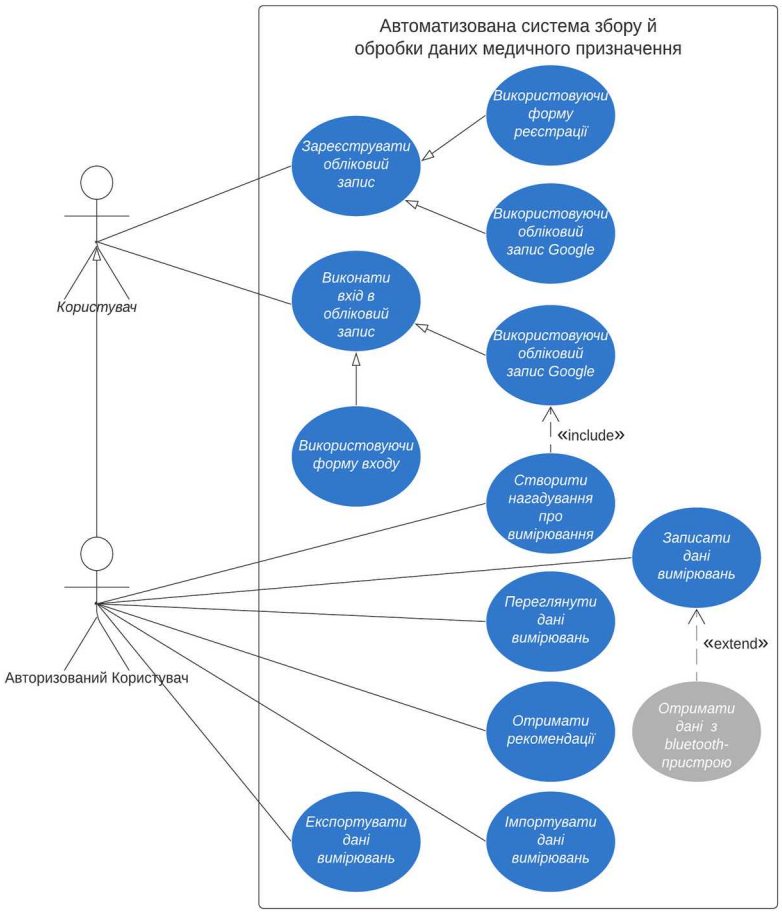


Рис. 1.7 – Діаграма варіантів використання АС

Системний адміністратор повинен мати можливість перевірити статус підсистем, тобто їх активність. Важливою складовою є також й версія підсистем, адже таким чином можна буде зрозуміти, який набір функцій підлягає верифікації з боку інженерів з ручного й автоматичного тестування. Відображення версії бази даних для адміністратора необхідно для розуміння стану поточної моделі даних та необхідності її оновлення.

Обсяг використання апаратних ресурсів таких як центральний процесор та оперативна пам'ять для адміністратора є вихідними даними для прийняття заходів щодо необхідності вертикального або горизонтального масштабування.

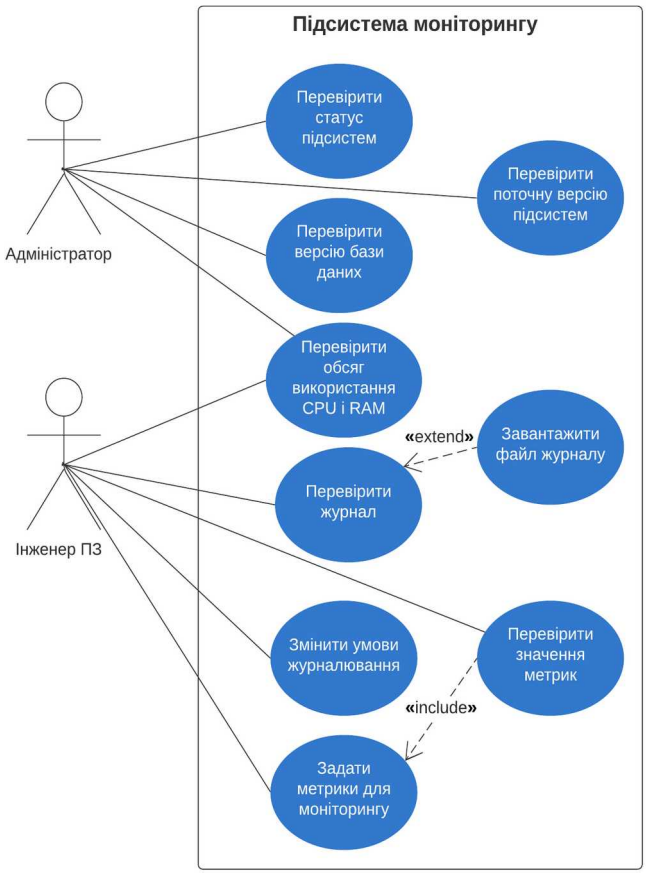


Рис. 1.8 . – Діаграма варіантів використання підсистеми моніторингу

Для інженера ПЗ також передбачається можливість перевірки обсягу використання апаратних ресурсів, проте для інших цілей – з метою виявлення витоків пам’яті, зміни конфігурації використовуваної структури пам’яті, перевірки ефективності роботи збирача сміття та ін. Важливим в роботі інженера ПЗ під час відтворення дефектів є також і перегляд журналу, де підсистеми записують результати під час роботи. Для уникнення зміни конфігурації систем та їх перезапуску передбачається встановлення й зміна рівнів журналювання . Для оцінки оптимальності прийнятих рішень для інженера ПЗ повинно бути доступна можливість перевірки метрик роботи підсистем.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ТА ОПИС АРХІТЕКТУРИ

2.1 Специфікація вимог

Для формалізованого опису вимог до розроблюваної ПЗ було створено специфікацію вимог до програмного забезпечення згідно зі стандартом SO/IEC/IEEE 29148:2011 [17]. Деякі розділи такі як «Цілі», «Читацька аудиторія», «Посилання» та ін. було опущено, адже вони або були визначені раніше в даній роботі, або не мають особливого значення для розроблюваного ПЗ.

2.1.1 Межі проекту

Розроблювана система призначена для збору МБП користувачів для їх подальшої обробки з наданням рекомендацій на основі профілактичних положень клінічних протоколів.

Обсяг проектних робіт включає в себе:

- розробку підсистеми графічного інтерфейсу ПЗ, який може використовуватись у вигляді нативних додатків платформи Android, IOS або у вигляді веб-додатку (далі – PWA);
- розробку предметно-орієнтованої мови програмування, що дозволить експертам предметної області створити моделі положень клінічних протоколів, що будуть використовуються для надання рекомендацій (далі – HDSL);
- розробку підсистеми, що забезпечує перевірку даних вимірювань МБП згідно з положеннями клінічних протоколів (далі – Protocol Engine);
- розробку основної підсистеми ПЗ, що реалізовує основні функції (далі - Api Service);
- розробку підсистеми моніторингу (далі – Admin);
- тестування й розгортання реалізованих компонентів.

2.1.2 Особливості продукту

- імпорт/експорт вимірювань медико-біологічних параметрів;
- запис вимірювань медико-біологічних параметрів;

- графічне відображення вимірювань медико-біологічних параметрів;
- аналіз даних вимірювань за допомогою положень клінічних протоколів;
- реєстрація за допомогою облікового запису Google;
- нагадування про необхідність здійснити вимірювання.

2.1.3 Класи користувачів та їх характеристика

Потенційних користувачів можна розподілити на наступні класи:

1. З хронічними захворюваннями;

2. Які ведуть здоровий спосіб життя і хочуть попередити поширені неінфекційні захворювання;

3. Які здійснюють вимірювання МБП в домашніх умовах за рекомендацією/призначенням лікаря.

Перша група буде використовувати ПЗ кожного дня, друга – декілька разів на тиждень або місяць, а третя – лише при рекомендації здійснення вимірювань МБП під час лікування або уточнення діагнозу.

Використання ПЗ не повинно вимагати від користувачів великої медичної або технічної обізнаності, залежати від рівня освіти чи віку.

2.1.4 Робоче середовище

Графічний інтерфейс ПЗ повинен працювати в браузері Google Chrome версії 84+, а також на смартфонах під управлінням Android 7+ й IOS 13+.

Підсистема збору й зберігання медико-біологічних показників, підсистема аналізу, підсистема моніторингу, сервер графічного інтерфейсу повинні працювати всередині контейнерів Docker на віртуальних машинах провайдера хмарних сервісів Microsoft Azure та не залежати від операційної системи.

2.1.5 Обмеження проектування й реалізації

Реалізація користувацького інтерфейсу повинна здійснюватися за допомогою фреймворку Angular з використанням мови програмування Typescript.

Реалізація підсистеми збору й зберігання МБП, підсистеми моніторингу, підсистеми обробки повинна здійснюватися з використанням мови програмування Java.

Для опису профілактичних положень клінічних протоколів слід реалізувати предметно-орієнтовану мову програмування. Протоколи описані даною мовою експертами предметної області повинні збиратися у вигляді незалежної бібліотеки, яку необхідно використовувати для аналізу медико-біологічних параметрів в підсистемі обробки.

Збір й обробка даних в ПЗ повинна відповідати положенням міжнародної конвенції GDPR [19].

Реалізовані компоненти ПЗ мають бути не доступними за межами локальної мережі, окрім компонентів, які відповідають за роботу графічного інтерфейсу.

В процесі розробки використання пропрієтарних компонентів, бібліотек, вихідного коду заборонено, оскільки розроблюваного ПЗ повинна ґрунтуватися на принципах вільного програмного забезпечення для забезпечення безперешкодності подальшого розвитку самого ПЗ або проведення наукових досліджень з її використанням суб’єктами, які не приймали участі в початковій реалізації цього ПЗ.

2.1.6 Функції системи

Наведено в підрозділі 1.7.

2.1.7 Вимоги до користувацького інтерфейсу

Користувацький інтерфейс повинен бути реалізований відповідно до вимог Material Design [18]. Інтерфейс повинен бути доступним на платформах Android, IOS і у вигляді веб-додатку.

У випадках мобільних платформ елементи інтерфейсу повинні відображатися однаково. Не допускаються будь-які відмінності в розмітці, шрифтах, блоках контенту та ін. компонентів користувацького інтерфейсу на мобільних платформах.

Навігація веб-інтерфейсом за допомогою клавіатури не передбачається.

Вспливаючі вікна, форми, діалоги повинні використовуватись лише для відображення необов’язкових деталей контенту або у випадку виконання певних дій, які вимагають повторного підтвердження, таких як видалення облікового запису користувача.

2.1.8 Вимоги до програмного інтерфейсу

Графічний інтерфейс повинен взаємодіяти з підсистемою збору й зберігання медико-біологічних параметрів використовуючи REST API з механізмом автентифікації та авторизації користувача на основі JWT. Підсистема обробки медико-біологічних також повинна відповідати принципам REST; реалізація автентифікації та авторизації підсистеми збору й збереження даних в підсистемі обробки даних є опціональною у випадку, якщо підсистема обробки опрацьовує анонімізовані дані, не зберігає стан процесу аналізу та недоступна поза межами локальної мережі розгортання ПЗ.

2.1.9 Вимоги до інтерфейсу комунікації

Взаємодія між підсистемами ПЗ повинна відбуватися з використанням протоколу HTTP версії 1.1 або 2.0.

Запис та читання даних з реляційної БД повинна здійснюватися за протоколом JDBC.

Збір даних підсистемою моніторингу повинен здійснюватися за протоколом HTTPS або JMX.

2.1.10 Вимоги до продуктивності

При розгорнутому 1 екземплярі кожної підсистеми ПЗ повинна забезпечити одночасну роботу 1000 користувач.ів з часом обробки запиту:

- не більше 1 секунди на запити, які включають підсистему графічного інтерфейсу та підсистему збору й збереження медико-біологічних параметрів;
- не більше 10 секунд на запити, які задіюють підсистему обробки медико-біологічних параметрів;
- не більше 2 секунд на запити, які задіюють треті системи (Google Calendar, Google OAuth2).

2.1.11 Вимоги до безпеки

Комунікація всіх підсистем ПЗ повинна здійснюватись з використанням шифрування згідно з криптографічним протоколом TLS версії 1.2 та вище. Використання шифрування також поширюється на взаємодію з будь-якими

зовнішніми системами (Google OAuth2, Google Calendar), базами даних (SQL, NoSQL) та брокерами повідомлень (ActiveMQ, Microsoft Azure Eventhubs, Apache Kafka).

2.2 Опис архітектури та проектування

Реалізацію ПЗ буде проводитися у відповідності з принципами клієнт-сервісної мікросервісної архітектури. Це означає, що кожна підсистема буде являти собою самостійний компонент, який виконуватиме одне або групу споріднених завдань. Підсистеми будуть розгортатися як самостійні сутності, що забезпечить незалежну еволюцію від інших компонентів. Такий підхід дозволяє розробляти кожну підсистему окремими командами розробників та знижує кількість необхідних знань про ПЗ для початку розробки інженером. Детально архітектура описана в Додатку З в документі SAD. Оскільки в розроблюваній системі передбачається розробка декількох підсистем (компонентів), то доцільно розглянути їх взаємодію у вигляді діаграми на рисунку 2.1.

З приведеної діаграми зрозуміло, що центральним компонентом системи виступає Api Service, який забезпечує автентифікацію й авторизацію користувача та зберігання даних вимірювань МБП. Для взаємодії користувача з системою відповідає компонент Progressive Web Application, який використовує Api Service за допомогою програмного інтерфейсу реалізованого за принципами REST. Api Service для виконання своїх функцій використовує:

базу даних через інтерфейс JDBC;

- сховище бінарних файлів через REST-інтерфейс, де зберігаються зображення з профіля користувача;
- компонент Protocol Engine через REST-інтерфейс для отримання рекомендацій;
- інтерфейс Google для автентифікації й авторизації користувача та доступу до календаря, де зберігаються нагадування.

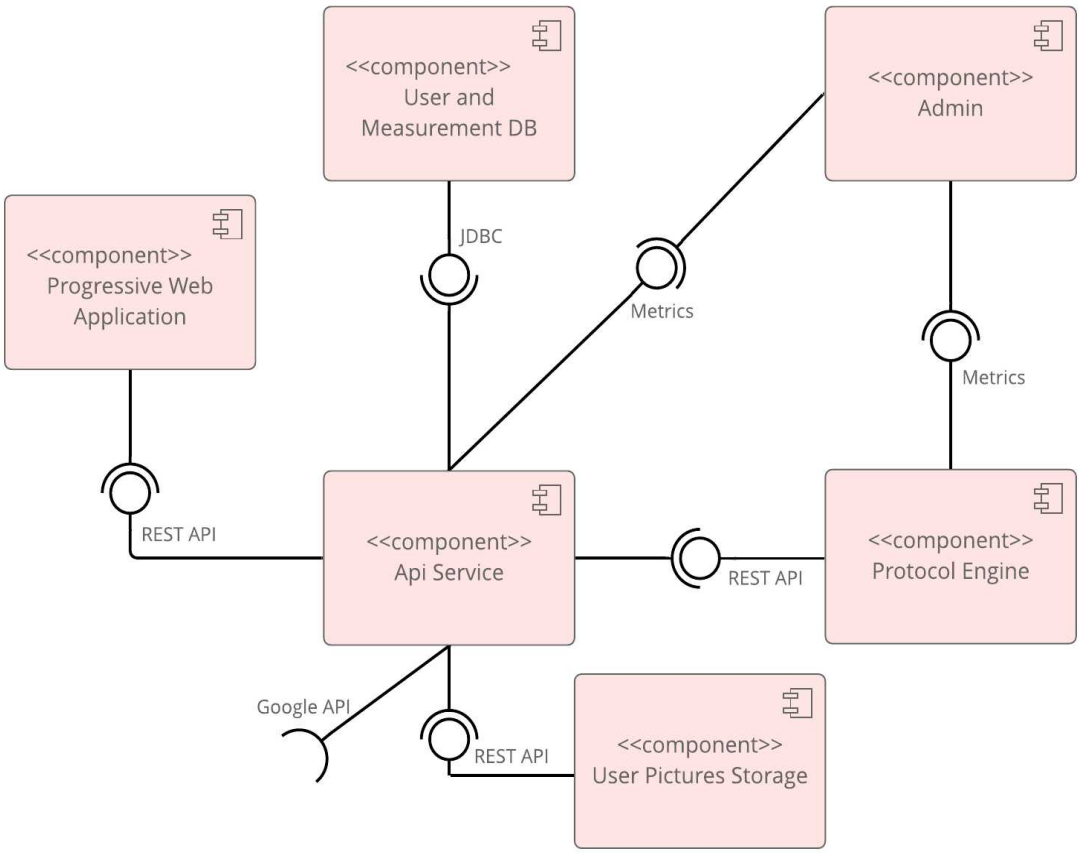


Рис. 2.1 Діаграма компонентів ПЗ

Api Service також реалізовує наступні інтерфейси:

- REST-інтерфейс, що використовується компонентом PWA;
- інтерфейс збору метрик роботи компонента.

Компонент Protocol Engine забезпечує процес перевірки даних вимірювань медико-біологічних показників з наданням рекомендацій й реалізовує наступні інтерфейси:

- REST-інтерфейс, що використовується компонентом Api Service;
- інтерфейс збору метрик роботи компонента.

Компонент Admin відповідає за збір метрик через спеціальні інтерфейси з компонентів Api Service та PWA. Також здійснює візуалізацію зібраних даних через власний користувацький інтерфейс.

Для забезпечення незалежності конфігурації підсистеми моніторингу від підсистем, що підлягають такому моніторингу, необхідно реалізувати механізм самостійної реєстрації таких підсистем на моніторинг. Таким чином у разі горизонтального масштабування підсистем, що підлягають моніторингу, нові екземпляри здійснять самостійну реєстрацію без необхідності зміни конфігурації підсистеми моніторингу та її перезапуску. Цей процес зображено на діаграмі діяльності, що приведена в додатку В.

Спочатку Admin очікує підключення підсистем, які підлягають моніторингу. Одразу після запуску Api Service та Protocol Engine виконують самореєстрацію в Admin. Після цього Admin починає в користувацькому інтерфейсі відображати підключені підсистеми, а також з певним інтервалом перевіряти їх статус. Якщо зареєстровані підсистеми відповідають позитивно на перевірку, то вони продовжують відображатися як активні. У випадку коли підсистеми не відповідають – Admin відображає їх як такі, що відключилися. В такому випадку адміністратору необхідно вжити заходів щодо виявлення причин такої поведінки та здійснити перезавантаження проблемних підсистем.

Реєстрація та вхід користувача за допомогою облікового запису Google буде здійснюватись за допомогою протоколу OAuth2. Цей протокол підтримує декілька варіантів реалізації в залежності від типу додатку, який буде його використовувати. Оскільки в даній роботі за автентифікацію та авторизацію відповідає Api Service, то реалізовувати OAuth2 протокол необхідно у вигляді Authorization Code Flow. Процес виконання реєстрації та входу приведений на діаграмі послідовності в додатку Д.

Користувач натискає на кнопку реєстрації, PWA здійснює обробку й надсилає перенаправлення через браузер на Api Service. Api Service в свою чергу зберігає деталі запиту на авторизацію в cookie та здійснює перенаправлення через браузер на сервер авторизації Google, вказавши при цьому свій код клієнта (англ. – client id).

Сервер авторизації Google перевіряє чи виконав поточний користувач вхід раніше. Якщо не виконав, то користувачу необхідно буде пройти процедуру автентифікації відповідно до налаштувань свого облікового запису. Після цього

сервер авторизації Google відобразить діалог підтвердження авторизації Api Service. Після підтвердження авторизації користувачем сервер авторизації Google здійснює генерацію кода авторизації й перенаправляє через браузер на Api Service. Api Service виконує запит токена доступу за допомогою отриманого кода авторизації, кода клієнта й секретного кода (англ. - client secret). Сервер авторизації Google перевіряє правильність цих даних та відповідає на запит токеном доступу. Api Service здійснює запит даних про користувача у сервера ресурсів Google. Отримавши дані користувача виконується їх зберігання в базу даних. Після зберігання виконується генерація JWT, який перенаправленням через браузер передається для обробки PWA. PWA виконує обробку JWT та здійснює відображення головного екрану.

Вхід користувача через звичайну форму з введенням адреси електронної пошти та пароля також реалізовується за допомогою генерації JWT, але не задіює треті системи, оскільки не є імплементацією протоколу OAuth2.

Оскільки PWA необхідно для виконання входу користувача лише JWT, що отримується від Api Service, то для здійснення виходу з облікового запису достатньо лише очистити з локального сховища браузера (англ. – local storage) значення JWT та здійснити перехід на екран входу.

З метою реалізації «права бути забутим» статті 17 Загального регламенту про захист даних [6] необхідно реалізувати механізм видалення даних про користувача. Його реалізація приведена у вигляді діаграми послідовності на рисунку 2.2.

Користувач натискає на кнопку видалення облікового запису, PWA здійснює обробку й відправляє запит на видалення до Api Service. Api Service виконує:

- видалення всіх даних вимірювань медико-біологічних параметрів;
- видалення інформації про обліковий запис користувача;
- видалення зображення профіля користувача.

Після успішного завершення всіх вищезгаданих операцій видалення Api Service відповідає PWA, що видалення виконано успішно. В свою чергу PWA виконує вихід для поточного користувача та здійснює перехід на екран входу. Після

цієї операції користувач не зможе користуватися програмним забезпеченням без повторної реєстрації.

За повторної реєстрації з використанням того ж самого облікового запису Google або адреси електронної пошти, користувачу не будуть доступні попередні дані вимірювань та зображення з профіля.

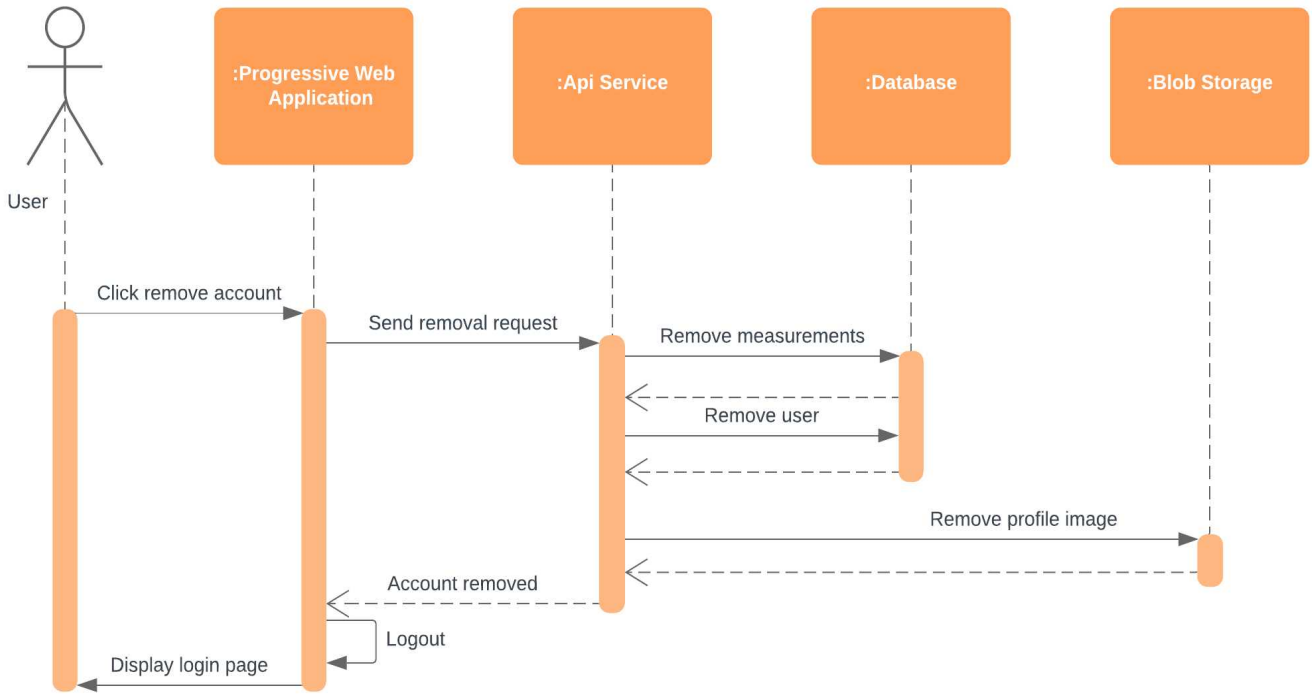


Рис. 2.2 Діаграма послідовності видалення даних про користувача АС

Для створення нагадування про необхідність здійснити вимірювання заданих медико-біологічних показників передбачається застосування Google Calendar, що полегшує реалізацію нагадувань на різних платформах й звільняє систему від необхідності власної реалізації календаря. Процес створення нового нагадування зображено діаграмою діяльності на рисунку 2.3.

Користувач переходить на екран з нагадуваннями. Якщо це користувач зареєстрований за допомогою облікового запису Google, то відображається список нагадувань на теперешній день, якщо ні – відображається повідомлення про недоступність цієї функції. При першому запиті нагадувань Api Service створює окремий календар та асоціює його з поточним користувачем. Далі цей календар

використовується для додавання нагадувань. Для створення нагадування користувач натискає на кнопку й заповнює форму. Правильно заповнену форму PWA відправляє Api Service для створення нагадування. Api Service у випадку повторюваного нагадування задає день повторення та відправляє запит Google Calendar для створення нагадування. Api Service відповідає, що нагадування успішно створено, а PWA відображає користувачу повідомлення.

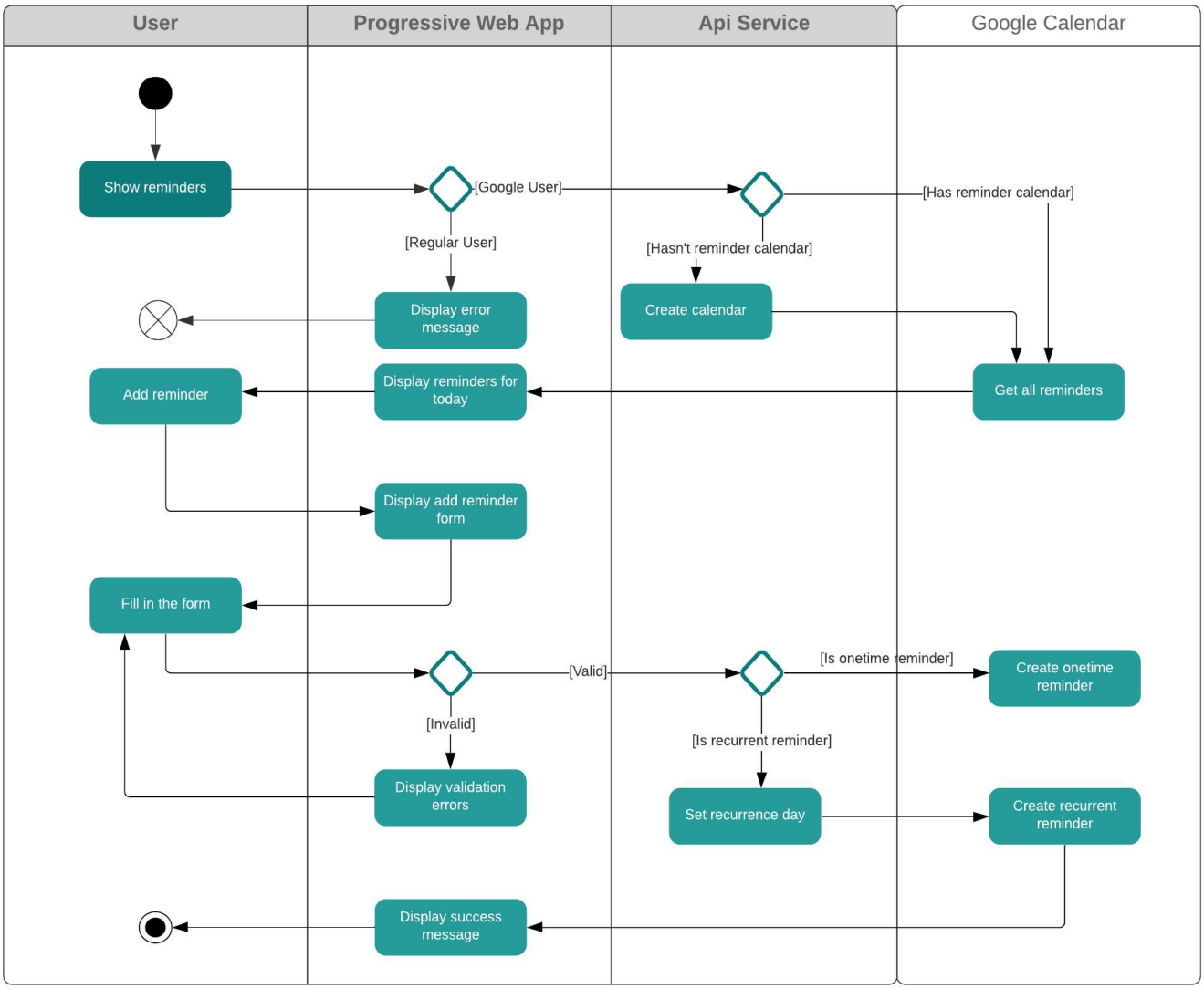


Рис. 2.3 Діаграма діяльності створення нагадування

Процес отримання рекомендацій користувачем щодо покращення стану здоров’я зображено діаграмою діяльності на рисунку 2.4. Користувач переходить на екран рекомендацій. PWA запрошує вибрати період, за який необхідно провести

аналіз медико-біологічних показників. Після цього PWA здійснює запит до API Service, який знаходить всі дані вимірювань за вибраний період та відправляє їх на аналіз. Protocol Engine здійснює аналіз даних вимірювань згідно з профілактичними положеннями клінічних протоколів, збирає результати та відправляє їх Api Service. Api Service виконує їх обробку та надсилає рекомендації PWA для відображення.

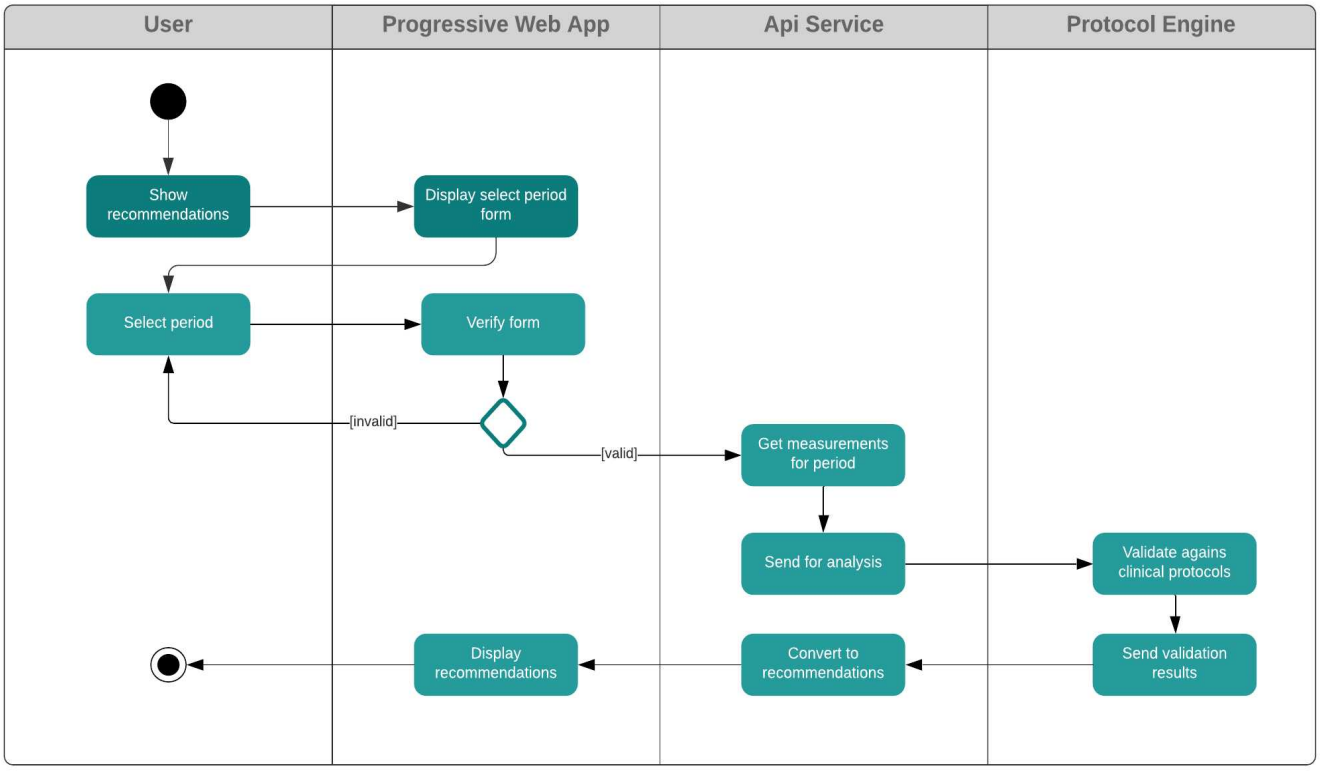


Рис. 2.4 Діаграма діяльності отримання рекомендацій

Розглянемо схему даних, що буде використовуватись в роботі системи (рис. 2.5). Для зберігання даних про користувачів використовується таблиця “users”. У кожного користувача є ім’я, тип автентифікації (базова чи за допомогою Google), пароль при використанні базової автентифікації, ідентифікатор календаря при використанні функції нагадувань, посилання на зображення, адреса електронної пошти та статус, який задає чи може користувач здійснити вхід в систему. Для розмежування прав доступу в систему у кожного користувача є ролі. Оскільки ролі представлені окремою таблицею «roles» й можуть бути призначені будь-якому користувачу, то виникає необхідність створення зв’язку багато-до-багатьох через

допоміжну таблицю «users_roles», де первинним ключем виступає композитний ключ, що складається з двох зовнішніх ключів, що вказують на первинні сурогатні ключі таблиць «users» та «roles». Для зберігання даних вимірювань МБП служить таблиця «measurements», яка має зв’язок з таблицею «users» багато до одного. Кожне вимірювання має певну одиницю вимірювання та значення. Поле для значення вимірювання відрізняється в залежності від типу вимірювання. Так для вимірювання пульсу необхідно заповнювати поле «pulse», а для артеріального тиску – «systolic» та «diastolic». Такий підхід дозволяє оптимізувати запити до бази даних в тих випадках, коли необхідно вибрати дані всіх вимірювань чи деяких їх видів. У альтернативної схеми даних, що приведена в додатку Ж, застосовано підхід у створенні окремої таблиці на кожен вид вимірювання. Такий підхід дозволяє задавати індивідуальні обмеження на вхідні дані для кожного вимірювання, проте виявляється менш продуктивним при виконанні запитів на отримання більше одного виду вимірювань. Таблиця «flyway_schema_history» передбачена для використання інструментом міграції бази даних Flyway, де записується історія виконання sql-скриптів міграції, на основі якої Flyway приймає рішення про необхідність виконання міграції.

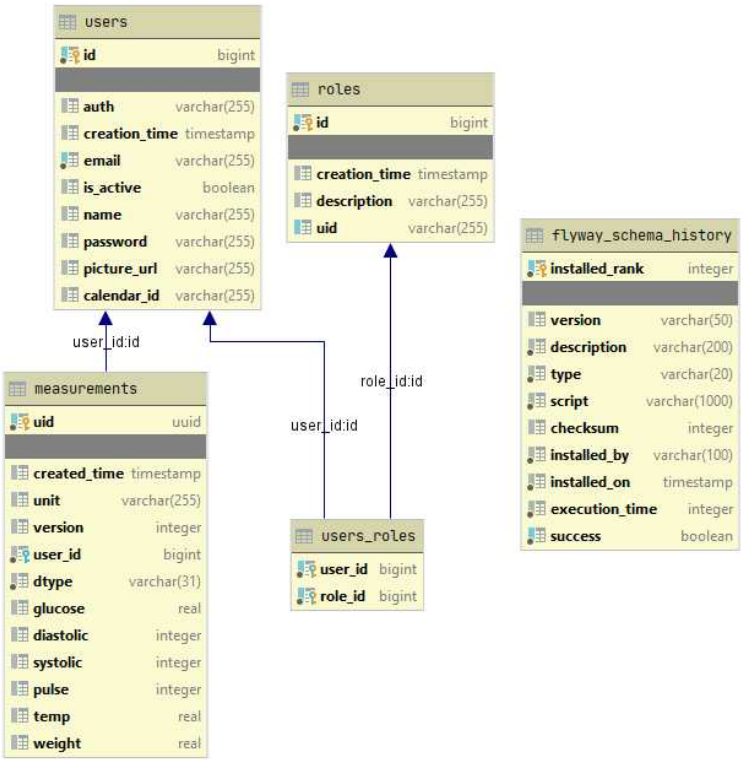


Рис. 2.5 Схема даних ПЗ

Розгортання системи бути проводитися з використанням провайдера хмарних сервісів Microsoft Azure, що зображено відповідною діаграмою в додатку Е. Так кожна підсистема розгортається окремо на віртуальних машинах Microsoft Azure. Розгортання в середовищі для розробки чи тестування допускає використання однієї віртуальної машини. Для PWA в якості середовища виконання використовується веб-сервер Nginx, а для Admin, Protocol Engine, Api Service – сервлет-контейнер Apache Tomcat. В якості сервера бази даних використовується Azure SQL Server, а в якості сховища для зображень профілів користувачів – Azure Blob Storage. Користувачі зможуть використовувати систему через веб-браузер на ПК або за допомогою встановлення PWA на смартфон.

Для спрощення розуміння програмної системи на даній діаграмі було створено таблицю відповідності між компонентом та артефактом, у складі якого він перебуває, а також підсистемою в рамках якої розгортається артефакт.

Таблиця 2.1

Відповідність між підсистемою, артефактом та компонентом

Підсистема	Компонент	Артефакт
Збору й зберігання МБП	Api Service	api-service.jar
Аналізу МБП	Protocol Engine	protocol-engine.jar
Моніторингу	Admin	admin.jar
Користувацького інтерфейсу	PWA	pwa/index.js

2.3 Вибір середовища та засобів розробки

Для реалізації Api Service, Admin та Protocol Engine було обрано мову програмування Java 11, яка на даний момент є однією з найбільш популярних. Для Java є багато фреймворків та бібліотек, з допомогою яких можна вирішувати будь-які завдання – від нейронних мереж до вбудованих систем. Завдяки кросплатформеності

додатки, розроблені цією мовою програмування, можна виконувати на будь-якій операційній системі, де встановлено JRE, а код написаний з використанням минулих версій Java можна запустити на новіших версіях, що забезпечує обернену сумісність, на відміну від інших мов програмування. Наявність збирача сміття, що звільняє розробника від ручної очистки пам’яті, та вичерпної документації робить вибір Java виправданим.

З метою імплементації підсистем на мові програмування Java буде використовуватись фреймворк Spring Boot 2. Він дозволяє створювати мікросервісні додатки з мінімально необхідною конфігурацією для їх швидкого розвитку та подальшого розгортання. Для додавання нових функцій, наприклад використання технології ORM для бази даних, достатньо додати лише 1 бібліотеку в залежності проекту, а роботу по конфігурації виконає фреймворк за рахунок заздалегідь підготовлених автоконфігурацій. Такий підхід дозволяє командам швидше реалізовувати бізнес-функції за рахунок зниження зусиль на конфігурацію, проте потребує певних зусиль для освоєння інженерами ПЗ.

Для реалізації користувацького інтерфейсу було обрано мову програмування Typescript. Це мова програмування зі статичною типізацією та підтримкою концепцій ООП, яка компілюється в мову програмування Javascript, що вже інтерпретується веб-браузером. Синтаксис Typescript за рахунок явного задавання типів є більш звичним й зрозумілим для інженерів ПЗ, які до цього працювали з мовами строгої типізації. Загалом Typescript вдосконалює Javascript з метою зменшення кількості помилок та пришвидшення реалізації нових компонентів системи. Користувацький інтерфейс буде реалізовуватись за допомогою фреймворка Angular, який дозволяє використовувати Typescript та створювати SPA – односторінкові додатки. Також це дозволяє відділити користувацький інтерфейс від основної логіки ПЗ, що забезпечує більшу гнучкість та незалежність в еволюції кожного компонента ПЗ.

В якості інтегрованого середовища для мови програмування Java було обрано JetBrains Intellij IDEA, що є найкращим рішенням в даний момент. Окрім підтримки

багатьох фреймворків та допоміжних засобів розробки ПЗ, дане середовище також підтримує інші мови програмування такі як Groovy та SQL, що в рази спрощує написання коду для збірки проекту та запитів для бази даних. Окрім того, для навчальних цілей JetBrains надає безкоштовну ліцензію на Ultimate версію, однією з найголовніших переваг якої є універсальний клієнт до будь-якої бази даних, підтримка мови програмування Typescript, фреймворків Angular та Spring Boot.

Незважаючи на той факт, що Intellij IDEA Ultimate підтримує створення Javascript проектів, було прийнято рішення використовувати для мови програмування Typescript окреме інтегроване середовище JetBrains WebStorm, яке було спеціально створене для програмування користувацьких інтерфейсів та включене в студентську ліцензію.

Для реалізації предметно-орієнтованої мови програмування буде застосовуватись система метапрограмування JetBrains MPS. Це ПЗ вільно розповсюджується, тому його використання не супроводжується необхідністю купівлі ліцензії, якщо порівнювати його зі схожими системами, такими як MetaEdit+. Перевагу над Eclipse Xtext забезпечує підтримка проекційного редактора, що дає можливість використовувати декілька нотацій, наприклад табличну, текстову, графічну та ін.

Інструментом збірки проектів на мові програмування Java виступає Gradle, що заснований на принципах Apache Ant та Apache Maven, але відрізняється використанням спеціальної мови програмування для створення конфігурації збірки проекту, наявністю кешу, що пришвидшує наступні збірки, та великою здатністю до кастомізації. За кількістю наявних плагінів Gradle майже не поступається Apache Maven, а вибір Gradle компанією Google як офіційного інструмента збірки для додатків під управлінням ОС Android говорить про міжнародне визнання. Для збірки PWA буде використовуватись Angular CLI, що є доступним при використанні Angular.

Системою контролю версій у всіх проектах виступатиме Git. Це найбільш поширена система контролю версій, що підтримується більшістю провайдерів

віддалених репозиторіїв ПЗ, таких як Github, Bitbucket, Gitlab та ін. Також в багатьох системах автоматизації процесів безперервної інтеграції та розгортання Git є вбудованим рішенням для завантаження вихідного коду з репозиторіїв. В якості хостингу Git-репозиторія буде використовуватись Github – найбільш поширене рішення, з яким у автора роботи найбільше досвіду.

Для зменшення кількості помилок в вихідному коді та уникнення загальновідомих проблемних рішень буде використовуватись статичний аналізатор вихідного коду Sonar Cloud. Він дозволяє проаналізувати код на виявлення вразливостей чи помилок, а також перевірити відповідність заданим значенням метрик ПЗ, таким як цикломатична складність, когнітивна складність та ін. Sonar Cloud не потребує купівлі ліцензії для проектів з відкритим вихідним кодом. У порівнянні зі схожим аналізатором Codacy, Sonar Cloud є більш функціональним й гнучким в налаштуванні правил перевірки.

В якості бази даних буде виступати Azure SQL Server – це хмарна Microsoft SQL Server-сумісна база даних, яку можна легко масштабувати. Підтримує геореплікацію, шифрування даних, маскування даних, автоналаштування продуктивності та надає потужні інструменти аналітики sql-запитів. Важливим фактором вибору бази даних слід вважати й питання ціни. Microsoft Azure надає для нових користувачів 12 місяців безкоштовного використання Azure SQL Server сервісного плану S0 DTU 10 та до 250GB для зберігання даних. Цього цілком достатньо для тестового розгортання програмної системи. Для зберігання зображень з профілів користувачів буде застосовуватись сховище бінарних файлів Azure Blob Storage, де 1ГБ даних для зберігання буде коштувати 2 центи, кожні 10000 операцій запису – 5 центів, а кожні 10000 операцій читання – 0.4 цента.

Для розгортання Java-підсистем буде використовуватись Azure Application Service, що легко інтегрується з GitHub, адже обидва сервіси належать компанії Microsoft. Окрім того для тестового розгортання в рамках одного Azure Application Service Plan, тобто використовуючи спільні апаратні ресурси, можна розгорнути всі 3 Java-підсистеми. Розгортання PWA було вирішено здійснювати у вигляді docker-

контейнера з використанням Azure Container Instances, через необхідність встановлення власних правил перенаправлення запитів для підтримки віртуальної маршрутизації фреймворка Angular.

Відповідно до піраміди тестування – модульні тести складають більшу частину всіх тестів й здатні виявляти помилки, які неможливо виявити будь-яким іншим способом, адже такі тести знаходяться найближче до вихідного коду за рівнем абстракції. Для тестування Java-підсистем буде використовуватись бібліотека JUnit 5. Для створення заглушок буде використовуватись бібліотека Mockito. Модульне тестування PWA буде здійснюватись з використанням бібліотеки karma, що одразу доступна для використання в Angular.

РОЗДІЛ 3
РОЗРОБКА ГЕНЕРАТОРА ТА МІКРОСЕРВІСНОГО ПЗ

3.1 Створення проекту мікросервісного ПЗ

3.1.1 Створення проекту Progressive Web Application

Підсистема PWA буде являти собою односторінковий веб-додаток, створений за допомогою фреймворка Angular та мови програмування Typescript. Особливістю цього додатку буде те, що він реалізований у вигляді прогресивного веб-додатку, що означає що його можна встановлювати на смартфони як нативний додаток. Це дозволяє отримати доступ до апаратних ресурсів (обмежений), отримувати push-повідомлення, працювати в офлайн режимі та запускати додаток за допомогою значка на робочому столі. Серйозною перепоною в масовому поширенні прогресивних веб-додатків на смартфонах з ОС Android до недавна була відсутність можливості завантаження з Google Play, де публікуються нативні додатки для цієї ОС. Проте в 2019 році компанія Google представила технологію Trusted Web Activity, що дозволяє публікувати нативні додатки в Google Play, які будуть використовувати PWA. Хоча даний проект не буде використовувати цю технологію, але це хороша можливість для подальшого розвитку.

В даному проекті використовується компонентний підхід до реалізації елементів користувацького інтерфейсу. Це означає, що елементи інтерфейсу є незалежними один від одного та їх можна повторно застосовувати. Проект складається з трьох модулів: «shared», «dashboard» та «login». Перший містить директиви та класи, що безпосередньо не відносяться до жодного елемента користувацького інтерфейсу, але використовуються ними. Другий містить всі компоненти інтерфейсу, що необхідні починаючи з головного екрану додатку. Третій включає в себе компоненти, сервіси та шаблони, які забезпечують автентифікацію та авторизацію користувача.

Для пришвидшення процесу розробки згідно з вимогами було задіяно бібліотеку готових компонентів «@angular/material». З метою зменшення об’єму

роботи, пов’язаної зі стилізацією розмітки HTML було використано бібліотеку «@flex-layout», що спрощує використання CSS Flex Layout за допомогою директив. Робота з формами виконувалася з допомогою бібліотеки «@angular/forms», а запити до Api Service – з допомогою вбудованого HTTP-клієнта з використанням бібліотеки «rxjs». Для відображення даних вимірювань МБП застосовувалась бібліотека графіків «chart.js».

Оскільки PWA є повноцінним додатком й розгортається окремо, то виникає необхідність виконувати власну навігацію по URL. Для цього в проекті в кожному модулі застосовувались підмодулі маршрутизації.

3.1.2 Створення проекту Api Service

Підсистема Api Service реалізована у вигляді Spring Boot додатку, застосовуючи принципи багаторівневої архітектури. Так ця частина програмної системи являє собою серверне ПЗ, яке складається з трьох рівнів:

- рівень представлення;
- рівень бізнес-логіки;
- рівень доступу до даних.

Рівень представлення реалізовано у вигляді REST інтерфейсу й документовано за допомогою використання специфікації OpenApi. Згенеровану документацію приведено на рисунку 3.1. Таким чином підсистема PWA отримує доступ до реалізованих функцій, а документація, яка генерується одразу з коду, завжди є актуальною та пришвидшує розробку інших систем, що потребують в своїй роботі звернення до даного REST інтерфейсу.

В наведеній документації відображається назва проекту та його ліцензія. Ліцензія в даному випадку MIT. MIT – це ліцензія відкритого програмного забезпечення, яка надає право на безоплатне використання, копіювання, модифікацію, розповсюдження й продаж третіми суб’єктами зі збереженням авторського права. Згенерована документація дозволяє не тільки переглянути, які REST ресурси доступні для використання, а також дозволяє одразу провести їх апробацію, тобто виконує функції HTTP-клієнта. Оскільки REST ресурси є

захищеними, тобто потребують попередньої авторизації перед отриманням доступу до них, то в згенерованій документації передбачений механізм авторизації за допомогою JWT-токена. Для того, щоб ним скористатися, необхідно натиснути на кнопку «Authorize» та ввести в модальному вікні рядок наступного вигляду «Bearer <token>», де:

- Bearer – це вид схеми авторизації протокола HTTP з застосуванням токена доступу;
- <token> – це текстове представлення JWT-токена, закодоване за допомогою алгоритма Base64.

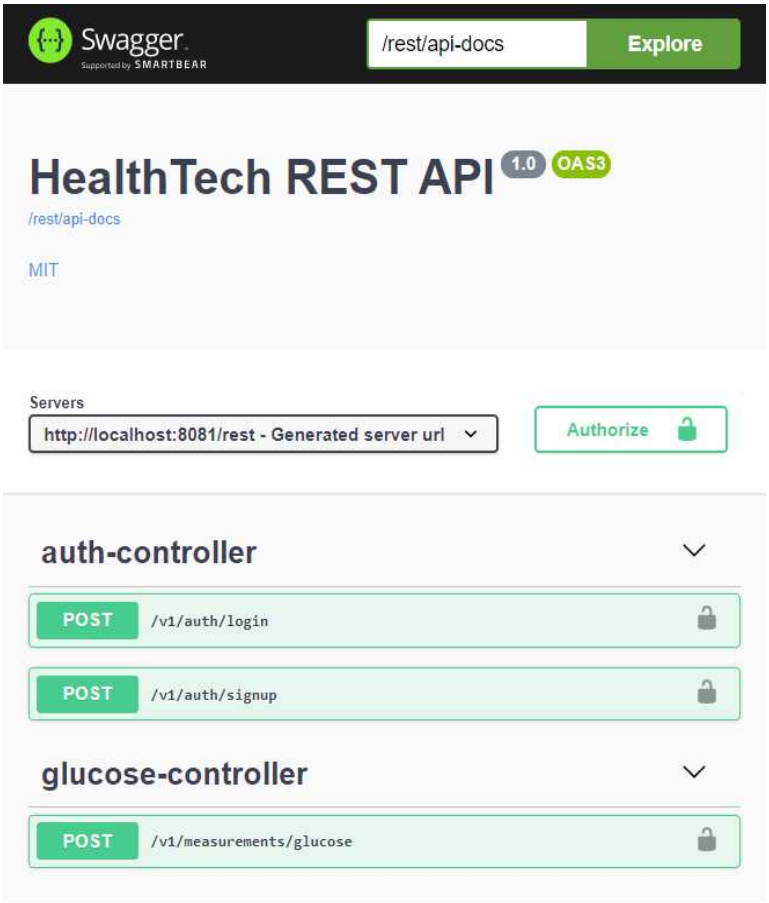


Рис. 3.1 Згенерована OpenApi документація для REST інтерфейсу

Реалізований REST інтерфейс приймає запити в форматі JSON, що є більш короткою та виразною альтернативою XML. Відповіді REST інтерфейсу також здійснюються в форматі JSON, що дозволяє без додаткових зусиль, пов’язаних з

парсингом відповіді, реалізовувати клієнтський інтерфейс, адже для мов програмування Typescript та Javascript JSON є нативним форматом.

Для підтримки реалізації підсистеми моніторингу Admin необхідно було реалізувати програмний інтерфейс, використовуючи який можна було б отримати доступ до метрик підсистеми, що включають в себе але не обмежені наступним переліком:

- відсоток використання процесорного часу;
- відсоток використання оперативної пам’яті;
- структура Java купи;
- кількість активних потоків;
- статус бази даних.

Для організації такого інтерфейсу було вирішено використовувати бібліотеку з екосистеми Spring Boot, що називається Spring Boot Actuator. Для налаштування інтерфейсу метрик необхідно лише додати цю бібліотеку в залежності проекту та сконфігурувати, які метрики будуть доступні. Spring Boot Actuator підтримує реалізацію інтерфейсу метрик з застосуванням протоколів JMX та HTTP. Для спрощення реалізації було обрано протокол HTTP.

Для реалізації рівня доступу до бази даних буде застосовуватись технологія ORM, що реалізована згідно з специфікацією JPA фреймворком Hibernate. Це дозволяє уникати однотипної роботи пов’язаної з написанням запитів до бази даних, розбору результатів, обробки виключень, управління підключеннями до БД та ін. Також Hibernate забезпечує автоматичну генерацію схеми даних на основі створених класів-сутностей, а також багаторівневе кешування, що дозволяє зменшити навантаження на БД.

Важливою перевагою використання Hibernate є той факт, що програмна реалізація буде не залежати від конкретної реалізації БД, що полегшує перехід на інші реалізації, що й було реалізовано в рамках даного проекту, коли було здійснено перехід з PostgreSql на Azure SQL Server. В даному проекті Hibernate генерує оновлення схеми даних в спеціальний sql файл, який з деякими правками можна

перемістити в директорію міграції схеми даних. Міграція схеми даних забезпечується використанням бібліотеки Flyway DB, яка при старті додатку перевіряє за даними власної службової таблиці версію схеми даних та виконує додаткові sql-скрипти у випадку, якщо поточна версія схеми даних не актульна.

З метою спрощення реалізації шаблона об’єкта доступу до даних було застосовано бібліотеку Spring Data JPA, що дозволяє описувати запити до бази даних у вигляді анотацій над методами інтерфейсу доступу до даних або у імені метода. Також є підтримка пагінації, сортування та предикатів. В даному проекті такий підхід себе виправдав, адже не довелося реалізовувати методи доступу до даних: реалізація генерується Spring Data JPA в момент запуску додатку.

3.1.3 Створення проекту Protocol Engine

Підсистема Protocol Engine – це серверне ПЗ, розроблене за допомогою фреймворка Spring Boot. Даний проект побудований за схожими принципами з проектом Api Service, проте в даній реалізації є більш простим. Protocol Engine також реалізована за принципами багаторівневої архітектури, де застосовувалось 2 рівні:

- рівень представлення;
- рівень бізнес-логіки.

Рівень доступу до даних не реалізовувався через відсутність необхідності зберігати історичні дані про аналіз вимірювань користувачів.

Рівень представлення реалізовано у вигляді REST інтерфейсу та документовано аналогічно Api Service. Єдиною відмінністю тут є відсутність обов’язкової авторизації за допомогою токена. Така реалізація обумовлена тим, що ця підсистема буде розгортатися в локальній мережі, що унеможливлює доступ з інших мереж, таких як інтернет.

Рівень бізнес-логіки забезпечує перевірку даних вимірювань МБП згідно з положеннями клінічних протоколів, які надаються у вигляді бібліотеки класів. Кожен протокол, що представлений окремим класом, створюється як компонент іос-

контейнера Spring, що забезпечує можливість їх використання з застосуванням принципу ін’єкції залежностей (англ. Dependency injection).

Програмний інтерфейс надання метрик для підсистеми моніторингу Admin було розроблено по аналогії з Api Service.

3.1.4 Створення проекту Admin

Підсистема Admin також була реалізована з використанням фреймворка Spring Boot і є серверним ПЗ. На відміну від інших підсистем, Admin не реалізовує функцій, що є важливими для кінцевого користувача, а слугує для використання системними адміністраторами та інженерами ПЗ у роботі з підтримки програмної системи.

Для реалізації збору й відображення метрик, що надаються підсистемами Api Service та Protocol Engine, використовувалась бібліотека Spring Boot Admin. Це дозволяє слідкувати за статусом підсистем, читати їх метрики та повідомляти відповідальних осіб при помилках в їх роботі. Також це зручний спосіб здійснювати моніторинг підсистем під час тестування продуктивності або під час відтворення дефектів, оскільки Admin дозволяє отримати доступ до журналу та налаштовувати рівні журналювання. На головній сторінці Admin відображаються підсистеми, що моніторяться в даний момент, а також кількість екземплярів кожної з них. Для перегляду більш детальної інформації слід натиснути на значок необхідної підсистеми. Сторінка, що відкриється бути мати вигляд як на рисунку 3.2. Тут відображається статус підсистеми, дата запуску, час роботи, ідентифікатор процесу, кількість використовуваних ядер центрального процесора, використання процесорного часу, структура купи Java в даний момент, кількість потоків та структура використання пам’яті, що не відноситься до купи.

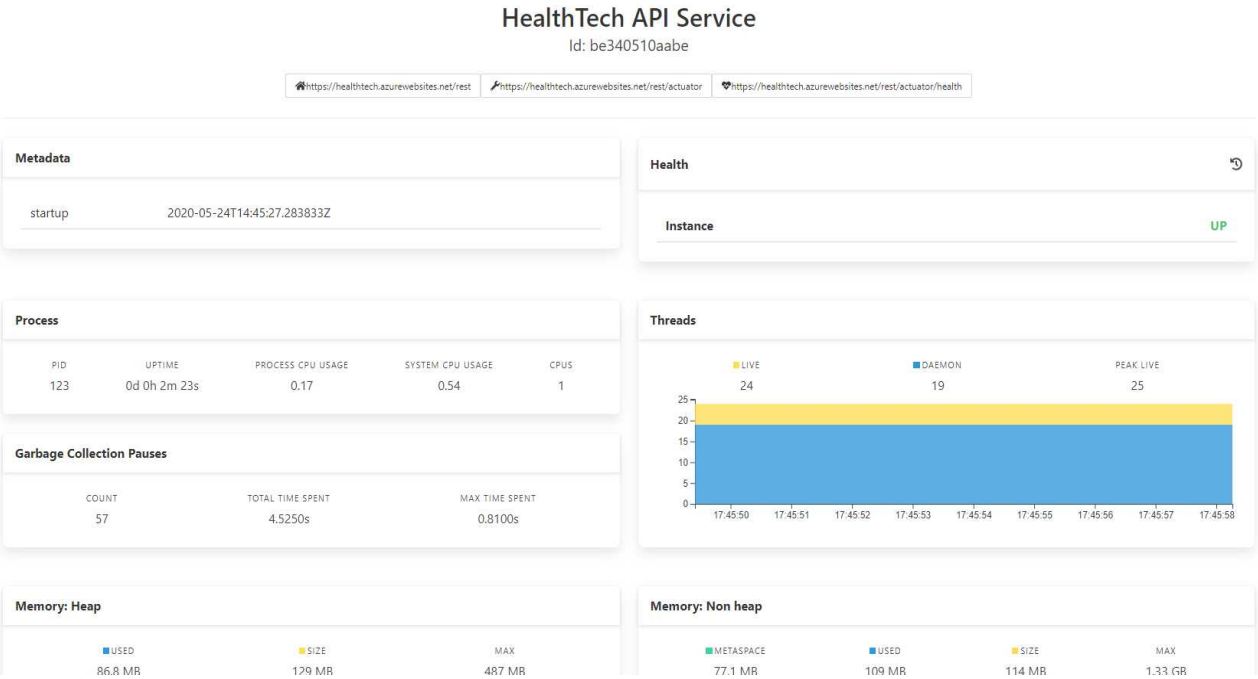


Рис. 3.2 Детальна інформація про статус підсистеми Api Service

Для отримання доступу до журналу підсистеми необхідно відкрити вкладку «Logging», де вибрати «Logfile». На сторінці буде відображатися файл журналу, який можна завантажити, як показано на рисунку 3.3.

Рис. 3.3 Сторінка доступу до журналу підсистеми Api Service

3.1.5 Створення проекту Protocol Language

Створення проекту Protocol Language включало в себе наступні завдання:

- реалізація синтаксису предметно орієнтованої мови;
- реалізація автодоповнення та меню трансформації;
- реалізація перевірки на помилки;
- створення генератора з розробленої HDSL в Java;
- збірка згенерованого коду в jar бібліотеку.

HDSL повинна забезпечувати експертів предметної області засобами для опису профілактичних положень клінічних протоколів у розрізі даних вимірювань МБП, таких як пульс, артеріальний тиск, температура, вага та ін. Звичайно, що для опису будь-яких положень профілактичних протоколів цього буде не достатньо, але розроблена HDSL дозволяє проводити подальше розширення для додавання нових видів МБП та інших даних, що можна отримати від користувача з застосуванням доступних приладів вимірювання.

Для створення синтаксису HDSL було створено 2 кореневих концепти: «Protocol» та «MeasurementUnitConfig». Перший використовується для створення моделі клінічного протоколу, а другий – для створення конфігурації одиниць вимірювання, які відповідають заданим видам вимірювань. В системі метапрограмування JetBrains MPS створений протокол буде являти собою модель, яка буде представлена у текстовому вигляді. Проте через застосування проекційного редактора – модель протокола насправді є абстрактним синтаксичним деревом (англ. AST). Це дозволяє не використовувати парсер, а це означає, що розроблювана HDSL не буде обмежена лише текстовою нотацією.

Будь-який концепт складається з властивостей, дочірніх концептів та посилань на інші концепти. Дочірні концепти відрізняються від посилань тим, що вони є частиною батьківського концепта, тобто є його дітьми по AST, а посилання це інші концепти, які не є безпосередніми дітьми цього концепта. В парадигмі ООП концепт можна сприймати як клас, а створену модель на основі концепта – як об’єкт.

Кожен протокол має свій унікальний ідентифікатор, опис та посилання повний текст. Для обмеження мінімальної кількості вимірювань, що будуть перевірятися та часового інтервалу, за який вони надаються, було реалізовано концепт «InputSpec».

Далі в протоколі можна використовувати тільки МБП, що були вказані в концепті «InputSpec». Для опису профілактичних положень на основі значень МБП використовується концепт «EvaluationEntry», який може містити декілька діапазонів МБП з одиницями їх вимірювання, а також результат у випадку, якщо значення МБП користувача потрапили в цей діапазон.

Реалізація автодоповнення здійснюється автоматично при реалізації концептом інтерфейсу INamedConcept або IValidIdentifier. Було розроблено меню трансформації та заміни з метою забезпечення коректності при створенні моделей «EvaluationEntry», тобто недопущення використання МБП, які не зазначалися раніше в моделі «InpuSpec», а також при створенні діапазонів МБП – заповнювати одиниці вимірювання відповідно до моделі «MeasurementUnitConfig».

Також було реалізовано перевірку діапазонів МБП де значення зліва повинно бути меншим за значення справа, а одиниця вимірювання не може відрізнятися від заданої в моделі «MeasurementUnitConfig».

Процес генерації в JetBrains MPS являє собою серію трансформацій типу «модель-модель». Це означає, що виконується створення з моделей однієї предметно-орієнтованої мови моделей іншої з застосуванням генератора. Для розробленої HDSL було створено генератор, який здійснює перетворення в модель на базовій мові програмування (англ. Base Language), що являє собою описану концептами мову програмування Java. Таким чином не довелося створювати текстовий генератор, а використовувались концепції Java, такі як класи, інтерфейси, методи, цикли та ін.

Кожен протокол було вирішено генерувати у вигляді класу на Java, а в його конструкторі виконувати ініціалізацію об’єктів, що представляють концепти «InputSpec» та «EvaluationEntry». Оскільки в даному випадку тільки ініціалізація об’єкта протоколу є динамічною, то виникла необхідність створення рішення часу виконання (англ. Runtime solution), що являє собою код мовою програмування Java, який було імпортовано в систему метапрограмування JetBrains MPS, для подальшого

використання в інших рішеннях або генераторах. Для HDSL це були класи, які не мають динамічної природи, тобто не змінюються в процесі генерації.

Згенерований код на мові програмування Java було вирішено компілювати та включити в jar архів з використанням інструмента збірки ant, для якого в JetBrains MPS було створено окрему мову програмування зі зручними плагінами.

Отримана Java бібліотека використовується підсистемою Protocol Engine для надання рекомендацій.

3.2 Демонстрація можливостей розробленого ПЗ

У цьому підрозділі буде здійснено демонстрацію деяких функцій розробленого ПЗ з точки зору кінцевого користувача. Розгортання здійснювалося на локальній машині розробника. Клієнтом виступає смартфон під управлінням ОС Android 10.

Для початку використання необхідно здійснити реєстрацію. Для цього на екрані реєстрації необхідно заповнити ім’я, адресу електронної пошти та пароль, як зображено на рисунку 3.4.

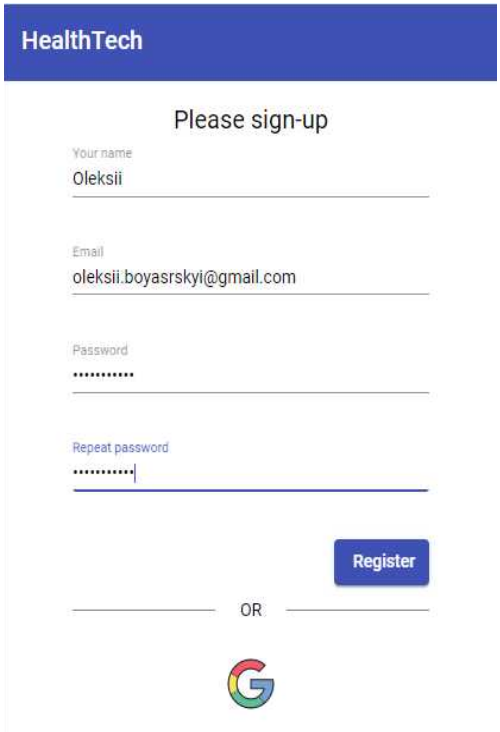


Рис. 3.4 Екран реєстрації

Після виконання реєстрації буде здійснено автоматичний перехід на екран входу, де потрібно ввести адресу електронної пошти та пароль як зображено на рисунку 3.5.

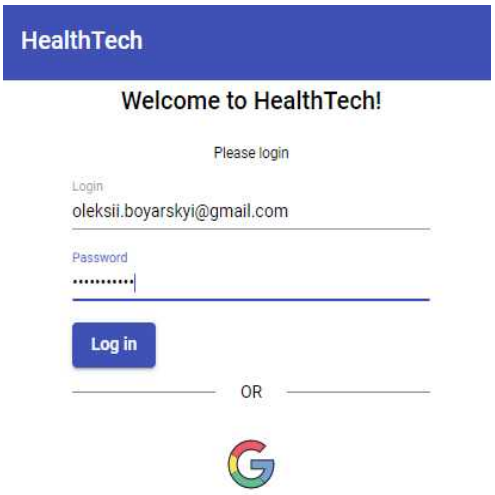


Рис. 3.5 Екран входу

Після виконання входу виконається перехід на головний екран, що зображено на рисунку 3.6а. На основному екрані відображається зображення профіля користувача, його адреса електронної пошти та ім’я. В нижній частині екрану розташовані 4 кнопки, що надають доступ до таких функцій:

- «Measurement» – перегляд даних вимірювань МБП у вигляді списку або графіка, додавання нових вимірювань МБП;
- «Reminders» – перегляд нагадувань, додавання нових нагадувань та видалення існуючих;
- «Recommendations» – отримання рекомендацій за певний період вимірювань;
- «Settings» – виконання операцій виходу, видалення облікового запису, експорту та імпорту даних вимірювань МБП.

Переглянути наявні вимірювання МБП можна натиснувши на кнопку «Measurement», тоді відкриється екран, що приведено на рисунку 3.6б.

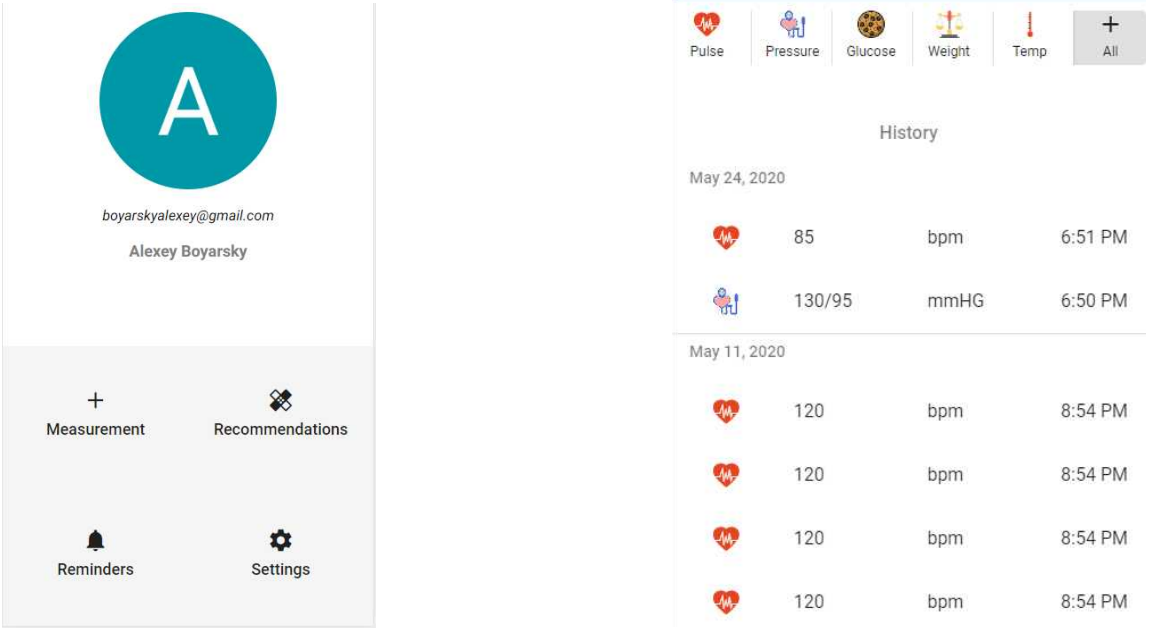


Рис. 3.6 а) Головний екран. б) Екран відображення вимірювань

Для запису вимірювання необхідно натиснути кнопку зі знаком «+» в нижній частині екрану, тоді буде виконано перехід на екран запису вимірювання, де необхідно заповнити форму. Приклад заповнення форми для вимірювання пульсу та артеріального тиску приведено на рисунках 3.7а та 3.7б відповідно.

Для отримання рекомендацій на основі вимірювань МБП необхідно з головного екрану натиснути на кнопку «Recommendations». Далі необхідно натиснути на кнопку «Analyze my measurements». Рекомендації базуються на основі даних вимірювань МБП за заданий період, значенням за замовчуванням є 1 місяць.

Після виконання аналізу МБП в нижній частині екрану будуть відображені результати. Приклад результатів аналізу за протоколом захворювання «Гіпертензія» приведено на рисунку 3.8.

З результату видно, що артеріальний тиск знаходиться на межі нормального. Згідно з тестовою моделлю клінічного протоколу передбачається встановлення нагадування на вимірювання артеріального тиску кожного тижня для подальшого

спостереження. Для відображення деталей перевірки необхідно натиснути на необхідний результат. Деталі будуть відображені в вспливаючому вікні.

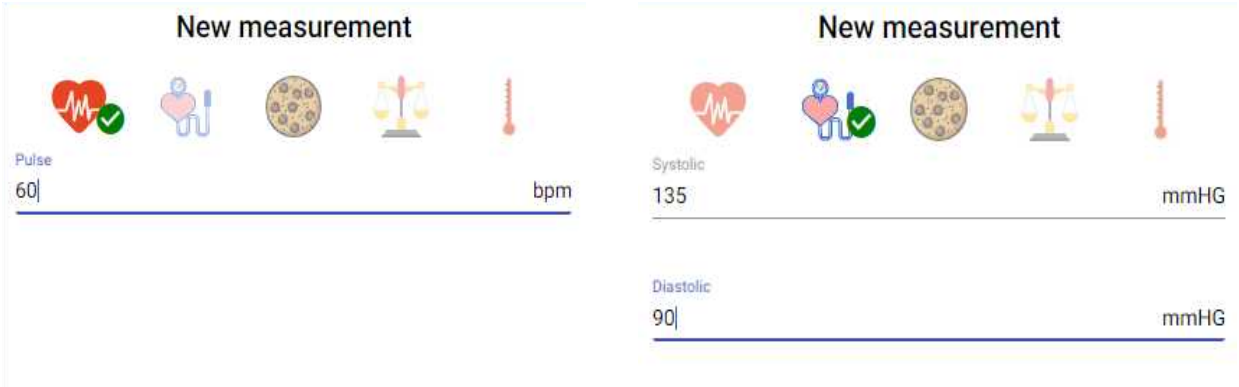


Рис. 3.7 а) Заповнення значення пульсу. б) Заповнення значення артеріального тиску

Для створення нагадування необхідно натиснути на кнопку «Reminders» на головному екрані. Далі в нижній частині екрану натиснути кнопку «+». Нагадування про вимірювання можна створювати як такі що повторюються. За замовчуванням нагадування не буде повторюватися. Нагадування може включати декілька видів вимірювань, як показано на рисунку 3.9, де створюється нагадування для вимірювання пульсу та артеріального тиску з повторенням кожного тижня. Повторення буде здійснюватися в час заданий в першому полі форми, а днем повтору за замовчуванням вибирається понеділок. Нагадування в момент спрацювання будуть відображатися на всіх пристроях користувача, де виконано вхід в обліковий запис Google. Це забезпечує незалежність нагадувань від пристрою й операційної системи, на яких встановлено додаток.

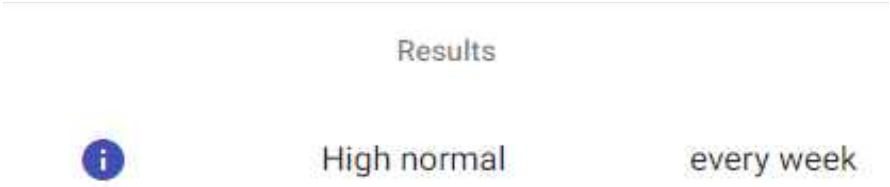


Рис. 3.8 Приклад результатів за протоколом захворювання «Гіпертензія»

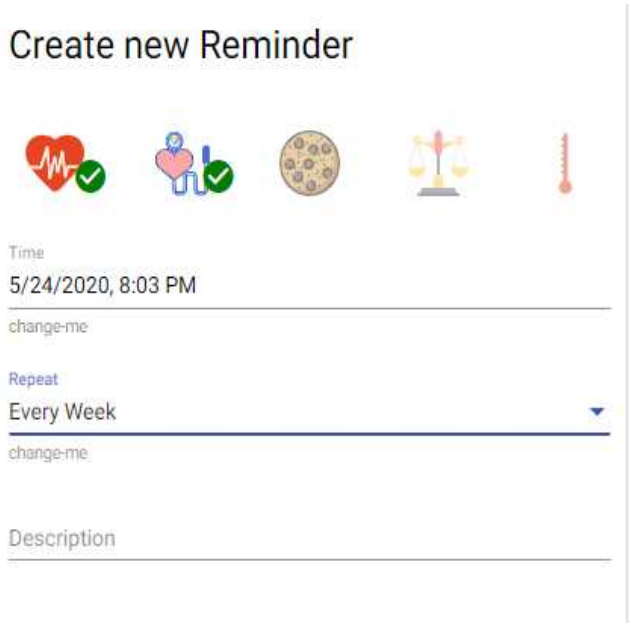


Рис. 3.9 Створення повторюваного нагадування про вимірювання артеріального тиску та пульсу

РОЗДІЛ 4
ТЕСТУВАННЯ Й РОЗГОРТАННЯ МІКРОСЕРВІСНОГО ПЗ

4.1 Методологія тестування

Тестування проводилося відповідно до піраміди автоматизації тестування. Так, основна логіка підсистем перевірялася за допомогою модульних тестів, а інтеграційні тести, що пов’язані з використанням інших підсистем або доступом до бази даних, створювались за потреби, наприклад при тестуванні рівня доступу до даних. В свою чергу тестування користувацького інтерфейсу, а саме підсистеми PWA, було вирішено здійснювати вручну, адже розгортання середовища для автоматизованих тестів, їх створення та підтримка виявилися б затратними не лише з точки зору фінансових ресурсів, необхідних для функціонування середовища виконання таких тестів в Microsoft Azure, а й з точки зору затраченого часу, що може складати не менше, ніж час затрачений на розробку ПЗ.

Всього в ході роботи було створено 55 автоматизованих тестів, з яких:

- 42 модульних та 8 інтеграційних тестів для Api Service;
- 4 модульних тести для Protocol Engine;
- 1 модульний тест для Admin.

Запуск тестів здійснюється при збірці проектів за допомогою інструмента Gradle та є обов’язковим перед відправкою вихідного коду до віддаленого репозиторію Github. Також запуск тестів здійснюється при оновленні гілки розробки «develop» за допомогою реалізованого процесу неперервної інтеграції (англ. Continious Integration).

Процес неперервної інтеграції було реалізовано за допомогою хмарного сервісу Circle CI, що надає безкоштовний доступ до побудови ланцюгів неперервної інтеграції (англ. CI pipeline) для проектів з відкритим вихідним кодом в рамках певної квоти на використання апаратних ресурсів. Основні кроки, що виконує Circle CI, приведені на рисунку 4.1. Після оновлення гілки розробки «develop» Github повідомляє Circle CI, що починає запуск ланцюга.

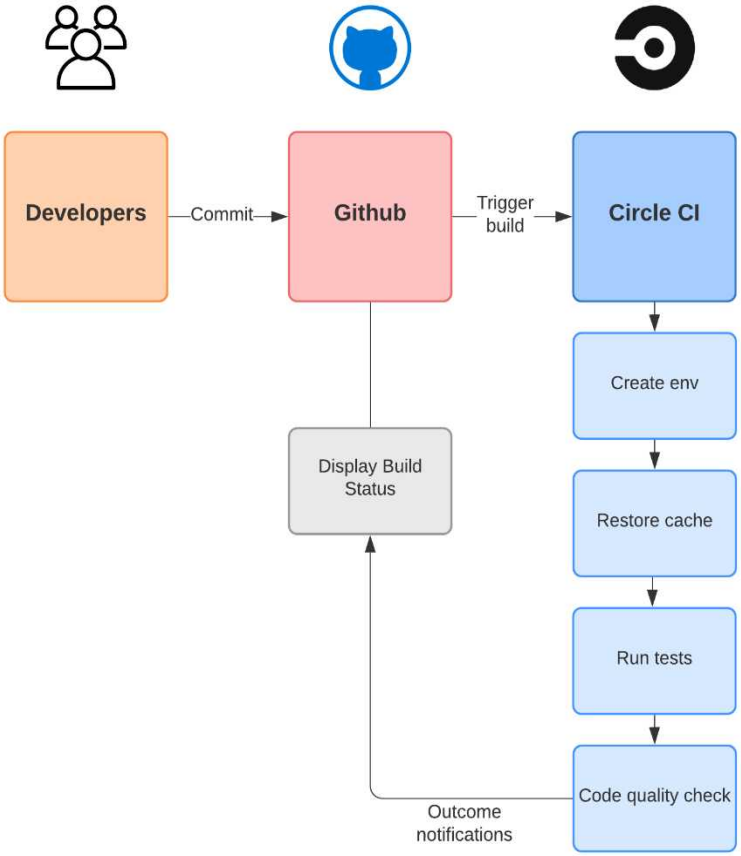


Рис. 4.1 Схема процесу неперервної інтеграції

Сам ланцюг неперервної інтеграції складається з наступних кроків: створення середовища виконання, де задаються необхідні системні змінні, далі виконується відновлення Gradle кешу, що було збережено попереднім запуском ланцюга. Наступним кроком виступає запуск модульних та інтеграційних тестів, якщо результат їх виконання виявився позитивним, то Circle CI повідомляє сервіс аналізу якості вихідного коду Sonar Cloud про необхідність здійснення перевірки. Після завершення аналізу коду здійснюється повідомлення Github про статус операції. В даному ланцюзі невдале виконання будь-якого з кроків приводить до дострокового завершення виконання всього ланцюга з повідомленням Github про помилки при виконанні.

Як згадувалося вище для статичного аналізу якості вихідного коду буде використовуватись хмарний сервіс Sonar Cloud, що дозволяє не розгортати

власноруч сервер SonarQube. Згідно з результатами аналізу не було виявлено серйозних дефектів в жодній підсистемі, що підтверджується загальним статусом кожної підсистеми на рисунку 4.2.

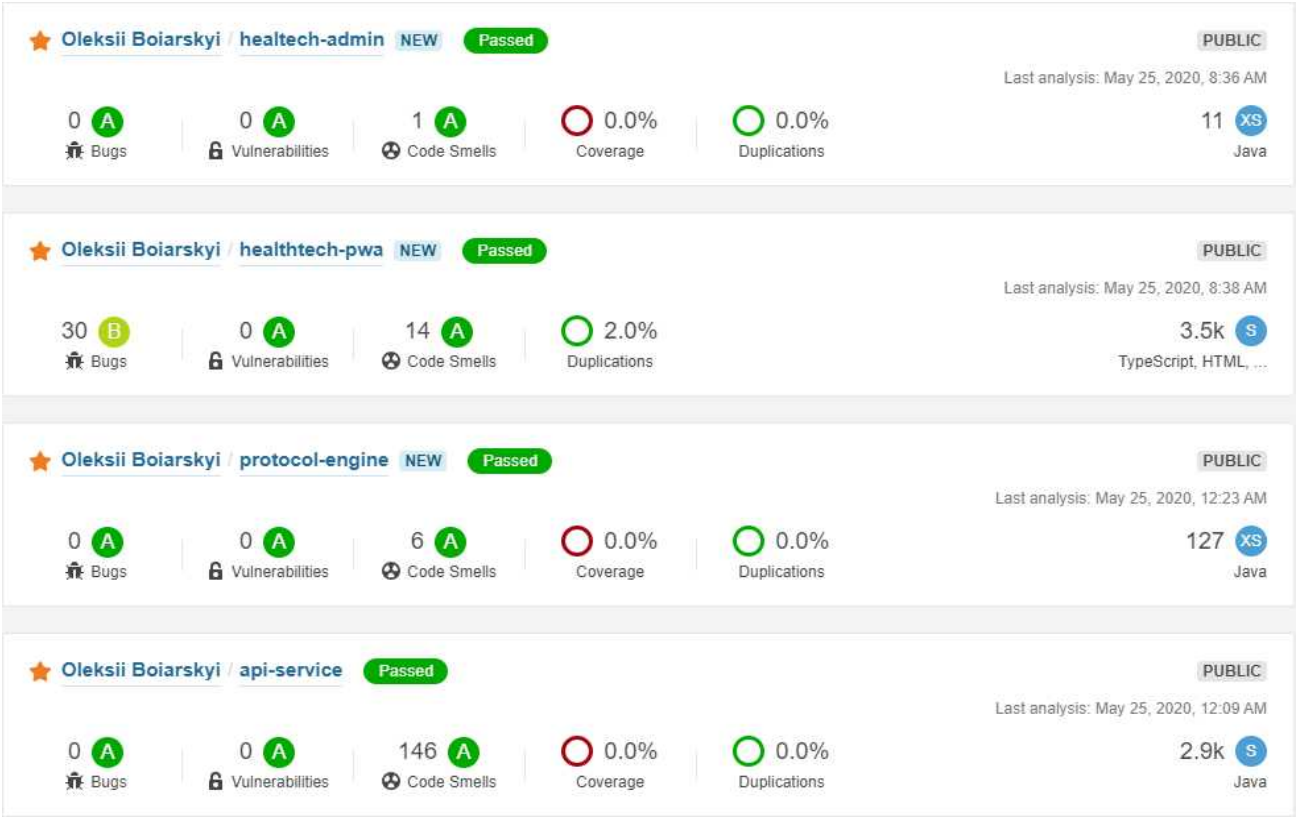


Рис. 4.2 Звіт Sonar Cloud про якість вихідного коду АС

Виявлені 30 дефектів в вихідному коді підсистеми PWA відносяться до семантичного значення деяких тегів HTML, що впливає лише на інтонацію голосу при використанні зчитувачів. Деякі рекомендації для покращення якості вихідного коду, що надав аналіз Sonar Cloud, будуть взяті до уваги при подальшому розвитку розробленого ПЗ.

4.2 Процес розгортання мікросервісного ПЗ

З метою реалізації вимог до розгортання розробленого ПЗ на віртуальних машинах Microsoft Azure було вирішено провести пробне розгортання використовуючи Azure Application Service.

Організацію швидкої доставки останньої версії вихідного коду було здійснено залучення сервісів Github Actions та DockerHub. Github Actions виконує розгортання таких підсистем як Api Service, Admin та Protocol Engine в момент оновлення гілки розробки «develop». Після закінчення завантаження zip-архіва з вихідним кодом в файлову систему Application Service, останній здійснює зупинку працюючої підсистеми та запускає її знову використовуючи новий вихідний код.

DockerHub використовувався для створення docker-образу з вихідного коду підсистеми PWA, що обслуговується веб-сервером nginx. Таке рішення було прийнято через неможливість налаштування віртуальної маршрутизації в Azure Application Service.

Для швидкого розгортання контейнеризованого ПЗ Microsoft Azure пропонується використовувати платформу Container Instances, що й було зроблено в рамках даної роботи. Платформа Container Instances виконує підключення до репозиторія docker-образів DockerHub, здійснює завантаження та запуск. У такого рішення є перевага над Application Service, оскільки є можливість гнучкого вибору апаратної конфігурації, проте є також й недоліки, наприклад відсутність вбудованої підтрики TLS шифрування, вибору власного домену, горизонтального масштабування та ін.

4.3 Шляхи вдосконалення та подальшого розвитку

Заходи щодо вдосконалення розробленого ПЗ необхідно розглядати за декількома групами:

1. Процес безперервної інтеграції та розгортання;
2. Процес тестування й аналізу якості коду;
3. Впровадження мікросервісних шаблонів.

До першої групи входить вироблення узагальненого підходу, й зведення до виконання операцій безперервної інтеграції та розгортання за допомогою одного сервісу, на відміну від теперішньої реалізації з залученням трьох сервісів, таких як Github Actions, Circle CI й DockerHub.

До другої групи входить розширення автоматизованого тестування, особливо з використанням модульних тестів для користувацького інтерфейсу. Також необхідно взяти до уваги існуючі результати аналізу якості вихідного коду та впровадити відслідковування коефіцієнта покриття тестами.

До третьої групи відносяться заходи з проектування та впровадження мікросервісних шаблонів, таких як Discovery Service, Configuration Service, Gateway та Circuit Breaker, що дозволять значно підвищити надійність та зменшити кількість однотипних конфігурацій в рамках підсистем розробленого ПЗ.

Розвиток розробленого ПЗ необхідно розглядати з двох сторін: розширення функціональних можливостей розробленого ПЗ та створеної HDSL. Впровадження нових функцій для кінцевих користувачів ПЗ повинно відбуватися незалежно від еволюції предметно-орієнтованої мови програмування.

Напрямками подальшого розвитку ПЗ є:

- інтеграція з приладами вимірювань МБП;
- публікація в Google Play у вигляді нативного додатку для ОС Android;
- локалізація користувацького інтерфейсу;
- додавання нових видів вимірювань МБП.

Для подальшого розвитку HDSL необхідно розглядати наступні напрямки:

- розширення синтаксису й меню автодоповнень;
- розширення кількості аналізованих захворювань й вдосконалення точності надання профілактичних рекомендацій;
- створення окремого середовища програмування для експертів предметної області;
- вдосконалення механізму фільтрації вимірювань МБП перед перевіркою;
- вдосконалення алгоритму аналізу МБП на великих масивах даних.

ВИСНОВКИ

У роботі описано результати реалізації мікросервісного ПЗ, що призначене для збору й обробки даних медичного призначення з наданням рекомендацій щодо покращення стану здоров’я, враховуючи профілактичні положення клінічних протоколів.

1. Проаналізовано існуючі програмні рішення для збору даних вимірювань МБП, що працюють на рівні клієнта згідно класифікації ВООЗ; виявлені їх переваги та недоліки.

2. Розроблено специфікацію вимог до ПЗ відповідно до міжнародних стандартів, де було окреслено його межі, функціональні та нефункціональні вимоги.

4. Описано мікросервісну клієнт-серверну архітектуру ПЗ, а також взаємодію компонентів.

5. Проведено проектування архітектури згідно шаблону SAD

6. Розроблено 4 окремі підситеми згідно специфікації вимог, в рамках Protocol Engine було розроблено генератор мікросервісного ПЗ з предметно-орієнтованих моделей клінічних протоколів.

7. Розроблено предметно-орієнтовану мову програмування для надання рекомендацій на основі клінічних протоколів;

8. Проведено тестування ПЗ за допомогою автоматизованих модульних тестів, перевірено якість вихідного коду як частини реалізованого процесу неперервної інтеграції.

9. Здійснено тестове розгортання ПЗ на віртуальних машинах провайдера хмарних технологій Microsoft Azure.

10. Визначено подальші напрямки вдосконалення та розвитку як мікросервісного ПЗ, так і предметно-орієнтованої мови програмування.

Окрім безпосереднього використання за прямим призначенням, розроблене ПЗ може застосовуватись як платформа для незалежних експериментів, що фокусуються на вдосконалення та розширення профілактичних рекомендацій за допомогою

розробленої предметно-орієнтованої мови програмування без необхідності розробки інших компонентів для можливості швидкої демонстрації чи застосування отриманих результатів.

Використання генератора з предметно-орієнтованої моделі дозволяє отримати готове до інтеграції мікросервісне ПЗ, що вирішує питання масштабування при збільшенні кількості даних для аналізу як результату збільшення користувацької бази. Мікросервісна архітектура забезпечує незалежність компонентів розробленої програмної системи, що в майбутньому дозволить їх незалежну еволюцію та розгортання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Прасол І. В., Єрошенко О. А. Розробка медико-технічного комплексу збору даних індивідуального призначення. Стратегії розвитку сучасної освіти і науки : матеріали І Міжнар. наук.-практ. інт. конф., м. Бердянськ, 28 лют. 2020 р. / БДПУ. Бердянськ, 2020. - 92 с.

2. Гойко О. В., Мохначов С.І., Аналіз сучасного програмного забезпечення для статистичного оброблення й аналізу біомедичних досліджень. - Київ // Медична інформатика та інженерія, 2012. - № 4. - 49 с.

3. Сітнікова О. О., Почебут М.В., Розробка системи підтримки прийняття рішень для багатопрофільної медичної допомоги. - Харків // Вісник НТУ "ХПІ", 2015. - №58(1167). - 86 с.

4. Патент US 20180137177 A1 / USA. DOMAIN SPECIFIC LANGUAGE TO QUERY MEDICAL DATA / Thomas Belcher , Simon McKenna. Опубл. 17.05.2018, Бюл. №15/816,533.

5. A Domain Specific Modeling Language Framework (DSL) for Representative Medical Prescription by using Generic Modeling Environment (GME) - Ben Dalla, Llahm Omar - [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/publication/344327532_A_Domain_Specific_Modeling_Language_Framework_DSL_for_Representative_Medical_Prescription_by_using_Generic_Modeling_Environment_GME.

6. Generalizing the Arden Syntax to a Common Clinical Application Language. Studies in health technology and informatics - S. Kraus - [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/publication/235376701_Integrating_Arden-Syntax-based_clinical_decision_support_with_extended_presentation_formats_into_a_commercial_patient_data_management_system

7. Automated System and Domain-Specific Language for Medical Data Collection and Processing - O. Boiarskyi, S. Popereshnyak - [Електронний ресурс]. - Режим доступу: https://link.springer.com/chapter/10.1007/978-3-030-82014-5_25.

8. Боярський О.В., Поперешняк С.В. Автоматизована система для збору й обробки даних медичного призначення. Застосування програмного забезпечення в інфокомунікаційних технологіях : збірник тез Всеукр. наук.-тех. конф., м. Київ, 2021 р. / ДУТ. Київ, 2021. - 16 с.

9. Боярський О.В., Поперешняк С.В. Тестування автоматизованої системи для збору й обробки даних медичного призначення. Математичні та програмні технології Internet of Everything : збірник матер. 7 Сх.-Євр. конф., м. Київ, 22 груд. 2020 р. / КНУ ім. Т. Шевченка. Київ, 2021. - 55 с.

10. Classification of digital health interventions v1.0 - World Health Organization - [Електронний ресурс]. - Режим доступу: https://www.who.int/reproductivehealth/publications/mhealth/classification-digital-health-interventions/en/.

11. Noncommunicable diseases country profiles 2018 - World Health Organization - [Електронний ресурс]. - Режим доступу: https://www.who.int/nmh/publications/ncd-profiles-2018/en.

12. Настанова 00069. Гіпертензія: обстеження та стартове лікування - Nikkilä Matti - [Електронний ресурс]. - Режим доступу: https://guidelines.moz.gov.ua/documents/2988.

13. Настанова 00057. Синусова тахікардія - Pekka Raatikainen - [Електронний ресурс]. - Режим доступу: https://guidelines.moz.gov.ua/documents/2980.

14. Настанова 00065. Шлуночкова тахікардія - Pekka Raatikainen - [Електронний ресурс]. - Режим доступу: https://guidelines.moz.gov.ua/documents/2986.

15. Настанова 00499. Оцінка пацієнта з ожирінням - Kirsi Pietiläinen - [Електронний ресурс]. - Режим доступу: https://guidelines.moz.gov.ua/documents/3335.

16. Настанова 00486. Діабет: визначення, диференційна діагностика і класифікація - H. Yki-Järvinen, T. Tuomi - [Електронний ресурс]. - Режим доступу: https://guidelines.moz.gov.ua/documents/3323.

17. Systems and software engineering - Life cycle processes - Requirements engineering - ISO - [Електронний ресурс]. - Режим доступу: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29148:ed-1:v1:en.

18. Material Design Guidelines - Material - [Електронний ресурс]. - Режим доступу: https://material.io/design/guidelines-overview.

19. General Data Protection Regulation - Official Journal of the European Union - [Електронний ресурс]. - Режим доступу: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN.

20. Voelter M. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages / Markus Voelter., 2013. – 558 c.

ДОДАТКИ

Додаток А

Диплом І ступеня за перемогу у 2 турі Всеукраїнського конкурсу студентських нукових робіт

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ДИПЛОМ

І ступеня

нагороджується

студент Київського національного університету
імені Тараса Шевченка

Боярський Олексій Володимирович

за перемогу у ІІ турі Всеукраїнського конкурсу
студентських наукових робіт зі спеціальності
163 Біомедична інженерія у 2020/2021 н.р.

Голова галузевої конкурсної комісії
Проректор з наукової роботи — Віталій ПАСІЧНИК

20-21 квітня 2021 року

Додаток Б

Акти впровадження результатів роботи

ЗАТВЕРДЖУЮ

Директор приватного підприємства

«Уніфільтр»

Троян Д.О.

"21" грудня 2020 р.

АКТ

про впровадження результатів наукових досліджень Боярського Олексія Володимировича, одержаних під час виконання наукової роботи «Автоматизована система для збору й обробки даних медичного призначення»

Комісія у складі:

голови комісії – директора приватного підприємства «Уніфільтр» Троян Д.О.,

та членів комісії: головного інженера Троян О.Д., завідувача виробництвом Ніколаєнко Р.М.

встановила, що наукові положення, розроблені особисто Боярським О.В. реалізовані на підприємстві наступним чином:

Реалізація автоматизованої системи була проведена у відповідності з принципами клієнт-сервісної мікросервісної архітектури. Це означає, що кожна підсистема являє собою самостійний компонент, який виконуватиме одне або групу споріднених завдань. Підсистеми будуть розгортатися як самостійні сутності, що забезпечить незалежну еволюцію від інших компонентів. Такий підхід дозволяє розробляти кожну підсистему окремими командами розробників та знижує кількість необхідних знань про автоматизовану систему для початку розробки інженером ПЗ.

Впровадження принципів які були виділені на етапі проектування дано системи допомогли в забезпеченні незалежності конфігурації підсистем моніторингу від підсистем, що підлягають такому моніторингу інформаційній технології управління виробництвом та реалізувати механізм самостійної реєстрації підсистем на моніторинг. Таким чином у раз горизонтального масштабування підсистем управління виробництвом, що підлягають моніторингу, нові екземпляри здійснять самостійну реєстрацію без необхідності зміни конфігурації підсистеми моніторингу та перезапуску.

Голова комісії: директор ПП «Уніфільтр» Троян Д.О.

Члени комісії: головний інженер Троян О.Д.

завідувач виробництва Ніколаєнко Р.М.

ЗАТВЕРДЖУЮ

Директор ТОВ "Долинське"

Стеценко Н.І.

"21" грудня 2020 р.

АКТ

про впровадження результатів наукових досліджень Боярського Олексія Володимировича, одержаних під час виконання наукової роботи «Автоматизована система для збору й обробки даних медичного призначення»

Комісія у складі:

голови комісії – директора ТОВ «Долинське» Стеценко Н.І.,

та членів комісії: головного інженера Соколовського І.Г., завідувача виробництвом Ігнатьєва В.С.

встановила, що наукові положення, розроблені особисто Боярським О.В. реалізовані на підприємстві наступним чином:

Користувачами даної автоматизованої системи став кожен працівник підприємства, особлива увага приділялася працівникам з сіл та селищ міського типу, де пункти надання медичної допомоги знаходилися досить віддалено. Дана автоматизована система інформувала про стан здоров'я користувачів, нагадувала про необхідність повторного проходження обстеження, динамічного порівняння медико-біологічних параметрів, їх аналізу та попередження захворювання на ранніх стадіях захворювання. Завдяки впровадженню системи був введений автоматичний контроль, щодо обов'язкового проходження медичного обліку, а у віддалених регіонах організована допомога в централізованому проходженні медичної комісії.

Впровадження даної системи допомогли зменшити ризики, щодо невиконання планових робіт у весняно-посівній та осінній періоди, за рахунок зменшення захворюваності працівників на сезоні захворювання та попередження загострення хронічних хвороб, і як наслідок виконання сільськогосподарських робіт згідно графіку та економія бюджету за рахунок незалучення допоміжних працівників на 6 % в порівнянні з минулим роком.

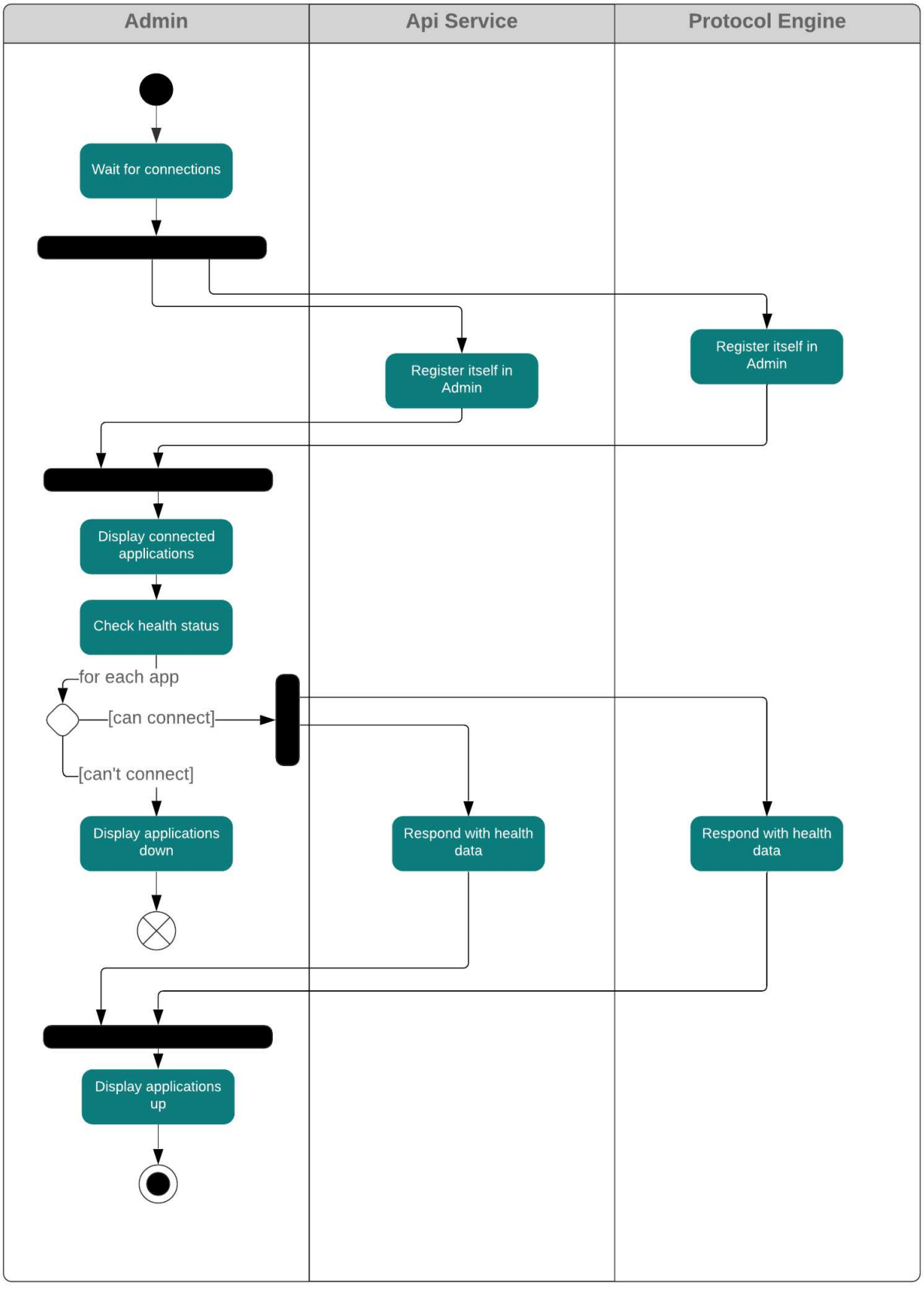
Голова комісії: директор ТОВ «Долинське» Стеценко Н.І.

Члени комісії: головний інженер Соколовський І.Г.

завідувач виробництва Ігнатьєв В.С.

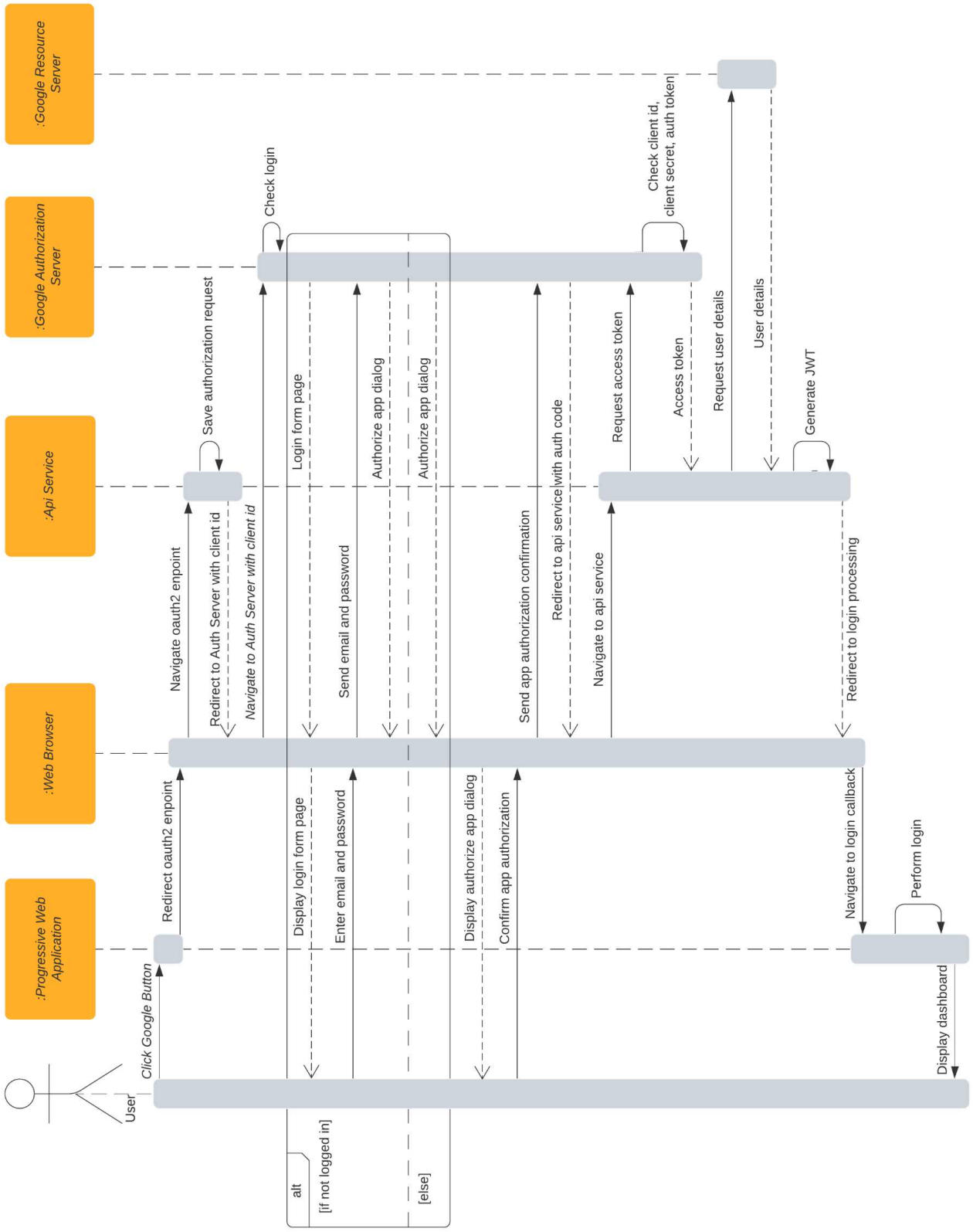
Додаток В

Діаграма діяльності реєстрації й моніторингу статусу компонентів



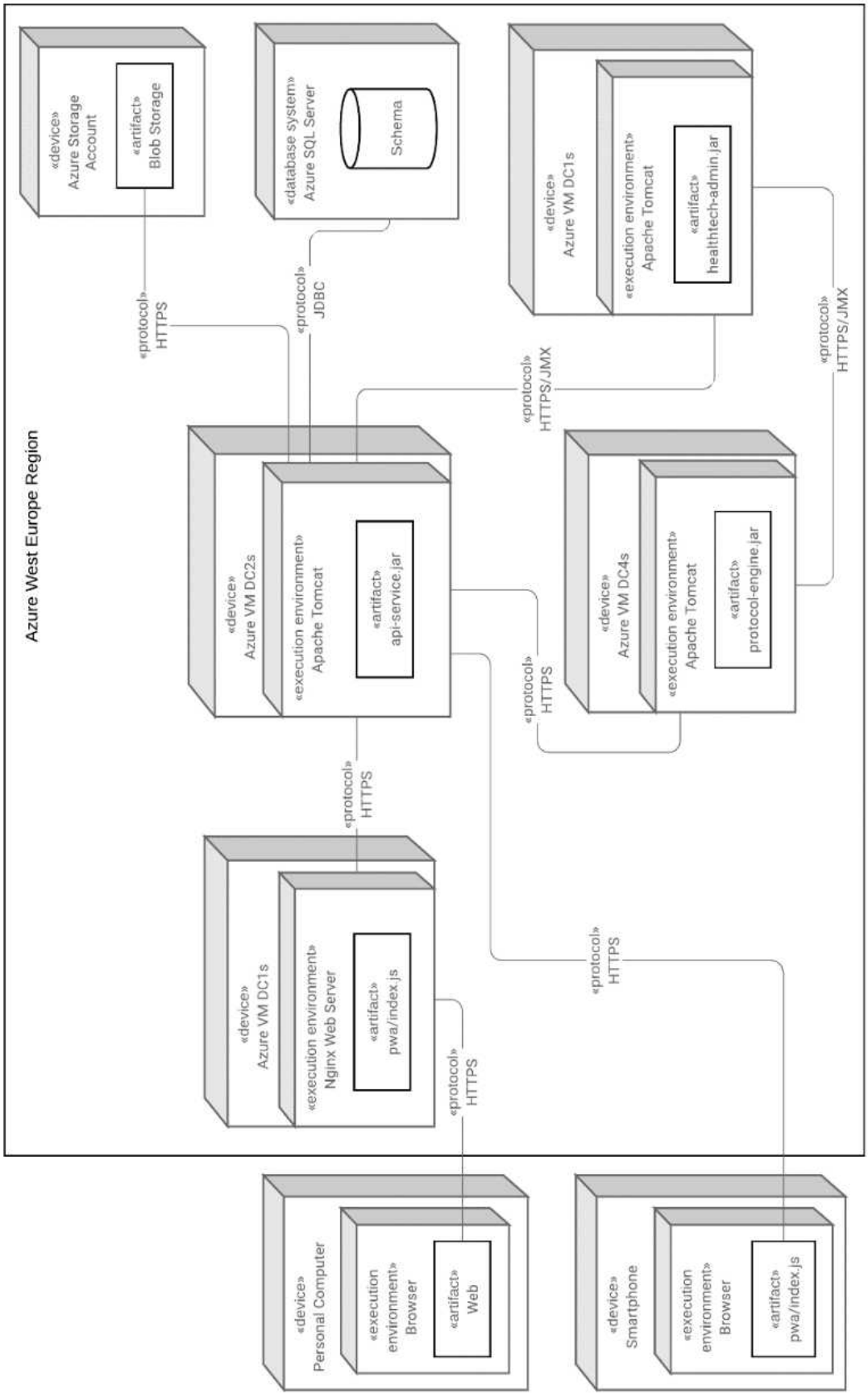
Додаток Д

Діаграма послідовності виконання реєстрації та входу за допомогою облікового запису Google



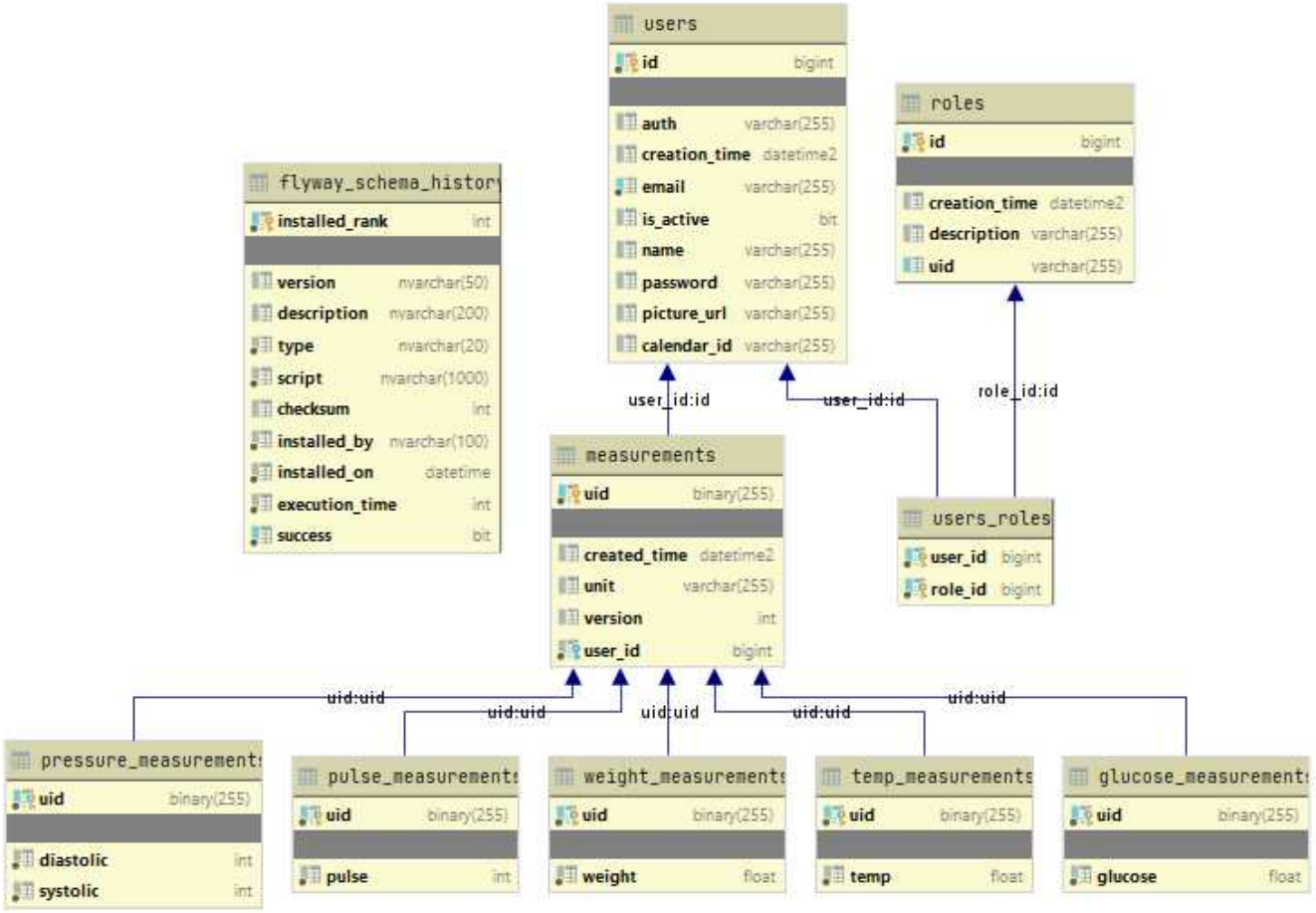
Додаток Е

Діаграма розгортання мікросервісного ПЗ



Додаток Ж

Альтернативна схема даних



Додаток 3

Документ опису архітектури ПЗ

SOFTWARE ARCHITECTURE DOCUMENT

Author: Oleksii Boiarskyi

Document Number: 1 Release: v0.1 Release Date: 04-24-2022

TABLE OF CONTENTS

INTRODUCTION

In recent years, many wearable electronics devices have been developed to measure blood pressure and heart rate. Also devices for measuring the level of glucose, hemoglobin, etc. Blood counts have become more affordable not only in price but also in the ability to be purchased by ordinary people rather than medical institutions. In general, people's awareness of their health has increased, and a healthy lifestyle in an friendly environment with a healthy diet is a growing concern. Therefore, the development of software that will allow the collection of medical and biological parameters and analysis with recommendations for improving health, is relevant.

Purpose

Problem statement

The use of information technology in the field of health care has increased significantly in recent years in Ukraine and abroad. There are many mobile applications and web applications in the field of healthcare on the market today, so there is a need to clearly classify such applications, as well as to formulate the main problems they solve.

The healthcare sector has traditionally been considered rather conservative and slow in adapting to new technologies, as any mistake can have irreversible consequences for the lives and health of patients. However, in the digital age - the penetration of IT in any sphere of human life is inevitable, because only in this way can achieve further development of the information society. The Classification of Digital Health Interventions (DHI) was created by the World Health Organization (WHO) to address this issue.

The problem arises when we look closely at client level of DHI. There we can see the Personal Health Tracking item but nowadays all application on this level suffer from the lack of connection between medical personnel and actual a potential patient that uses the app on daily basis. Developers of such applications cannot guarantee that an automated medical data analysis is precise and valid because they don’t have a medical education. Of course, we can connect a patient and a doctor directly but it will not be a Personal Health Tracking application but rather a Telemedicine one. So, the developed system must solve this problem without going beyond the selected DHI scope.

Objectives

Our system must provide doctors with DSL and IDE to create recommendation system.

System must contain monitoring component, recommendation engine and API component. API component must provide interfaces for client web application.

Through this document, the architecture of the system will be described, as a way that compliments the code and describes what the code itself wouldn't do. This document is intended to describe the architectural decisions which have been made on the system.

Scope

There are 2 subscopes of the project: client and medical. Client scope includes capabilities for the user(patient) while using the web application. Medical scope is related to medical personnel which uses DSL and IDE.

Web application must give an ability to collect, analyze, remove user's medical data. Also, it must have a capability to set reminders of a need to make a measurement. Automatic data collection through Bluetooth protocol along with mobile application for the operating systems Android and IOS are out of scope.

DSL gives the ability to the domain experts to create models of clinical protocols that are used for analysis of users' measurements. DSL can't be and shouldn't be allmighty. In current project it should support base recommendation cases, IDE can be not customized to save time.

Glossary

Term list:

Stakeholder: any person involved or affected, directly or indirectly, by this product.

Java: A high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.

Typescript: A programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language.

Acronym list:

DSL: Domain specific language

IDE: Integrated development environment

REST: Representational State Transfer, web API featuring a state-less client-server infrastructure.

API: Application Programming Interface, a protocol used as an interface to allow communication between different components.

CSS: Cascading-Style Sheets, document that describes the appearance of web pages.

JSON: JavaScript Object Notation, a text-based standard for human-readable data exchange.

MVC: Model-View-Controller, a software architecture pattern that separates the physical way to store data, the business logic and the appearance to the user.

Stakeholders

Each stakeholder is concerned with different characteristics of the system. Here is a list of the

stakeholder roles considered in the development of the architecture described by this SAD.

Name
Domain experts
Description
They are users of DSL and they create rules of users’ measurements analysis
Concerns
DSL accuracy, DSL compliance to the medical protocols, DSL syntax legibility

Name
Developers
Description
They are coders of the system
Concerns

Modern programming tools and libs, frameworks, cloud deployment, accuracy and completeness of the requirements

Name
End users
Description
They are users of the system namely web application
Concerns
Convenient and fast user interface, low installation and configuration efforts, data security, validity of data analysis

Non-functional requirements

The system must ensure the simultaneous operation of 1000 users with the time of request processing:

- no more than 1 second for requests that include a graphical interface subsystem and a subsystem for collecting and storing medical and biological parameters;
- no more than 10 seconds for requests that involve the subsystem for processing medical and biological parameters;
- no more than 2 seconds for requests involving third-party systems (Google Calendar, Google OAuth2).

ARCHITECTURE OVERVIEW

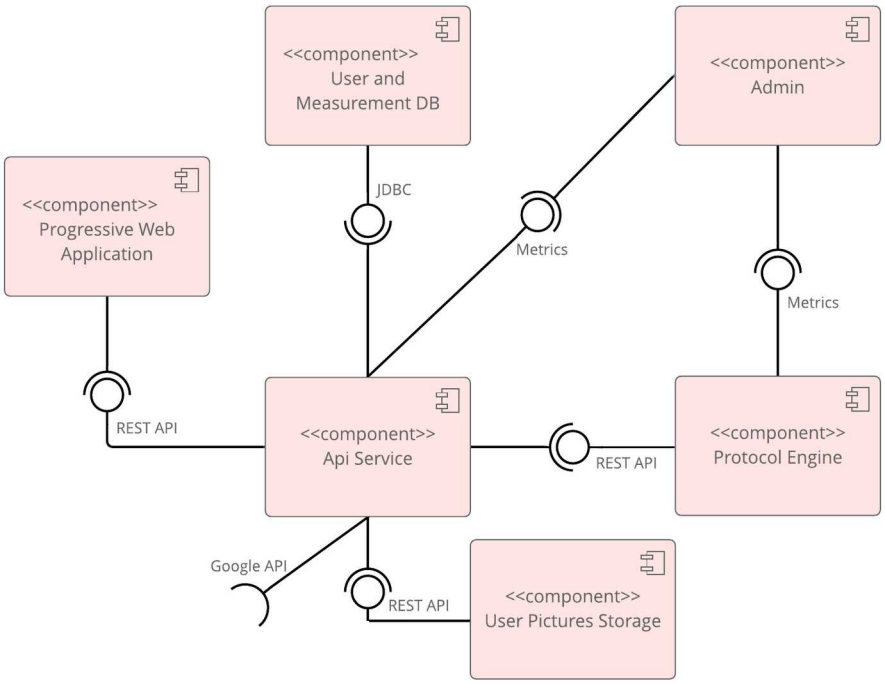
Architecture summary

The implementation of the system will be carried out in accordance with the principles of client-service microservice architecture. This means that each subsystem will be a separate component that will perform one or a group of related tasks. Subsystems will be deployed as independent entities, which will ensure independent evolution from other components. This approach allows each subsystem to be developed by individual

development teams and reduces the amount of knowledge required to start a software engineer. Since the developed system includes a few subsystems, it is necessary to consider their interaction in the form of a diagram in figure below.

There are 6 components:

- Progressive Web Application (PWA) – application used by end user, can be installed on mobile or desktop platforms.
- Api Service – provides PWA with REST interface, utilizes MVC pattern and multi-layered architecture. Uses Google API to implement OAuth2-compatible sign in and sign up, calendar access for reminders.



- Protocol Engine – provides REST interface for Api Service, runs user's measurements analysis.
- Admin – monitors load and health of other components, show logs, db migration status, memory usage. Gathers metrics through the corresponding interfaces of Api Service and Protocol Engine.
- User and measurement DB – SQL database, used by Api Service to store user database.
- User Pictures Storage – storage of unstructured data, keeps user profile images.

To better understand the whole system architecture we will consider the deployment diagram in figure down below. It shows communication protocols selected and networks

involved. We decided to choose Microsoft Azure as a provider of deployment infrastructure.

Here we see two end platforms – PC and mobile, both support PWA installation. When user makes any request, it goes into Microsoft Azure Network where is routed to the West Europe Region. In that region all server-side components are deployed. They are located in the same private virtual network, where only Api Service has an access to the Internet to have a capability to work with Google API. Neither of the databases is accessible from the Internet.

Concerns and decisions

For the implementation of Api Service, Admin and Protocol Engine, the programming language Java 11 was chosen, which is currently one of the most popular. For Java, there are many frameworks and libraries that can be used to solve any task - from neural networks to embedded systems. Due to the cross-platform nature of applications developed in this programming language, you can run on any operating system where JRE is installed, and code written using previous versions of Java can run on newer versions, which provides backward compatibility, unlike other programming languages. The presence of a garbage collector, which frees the developer from manual memory cleaning, and comprehensive documentation makes the choice of Java justified.

In order to implement subsystems in the Java programming language, the Spring Boot 2 framework will be used. It allows you to create microservice applications with the minimum necessary configuration for their rapid development and further deployment. To add new features, such as the use of ORM technology for the database, it is enough to add only 1 library depending on the project, and the configuration work will be performed by the framework due to pre-prepared autoconfigurations. This approach allows teams to implement business functions faster by reducing configuration efforts, but requires some effort for software engineers to master.

JetBrains Intellij IDEA was chosen as the integrated environment for the Java programming language, which is the best solution at the moment. In addition to supporting many software development frameworks and utilities, this environment also supports other programming languages such as Groovy and SQL, which greatly simplifies the writing of project build code and database queries. In addition, for educational purposes, JetBrains provides a free license for the Ultimate version, one of the main advantages of which is a universal client for any database, support for the Typescript programming language, Angular and Spring Boot frameworks.

Despite the fact that Intellij IDEA Ultimate supports the creation of Javascript projects, it was decided to use a separate JetBrains WebStorm integrated environment for the Typescript programming language, which was specially created for programming user interfaces and included in the student license.

The JetBrains MPS metaprogramming system will be used to implement the subject-oriented programming language. This software is freely distributed, so its use is not accompanied by the need to purchase a license compared to similar systems such as MetaEdit +. The advantage over Eclipse Xtext is provided by the support of the projection editor, which allows you to use several notations, such as tabular, text, graphic, etc.

Gradle is a tool for building projects in the Java programming language, based on the principles of Apache Ant and Apache Maven, but distinguished by the use of a special programming language to configure the project collection, cache, which speeds up subsequent builds, and great customization. The number of available Gradle plugins is almost second to that of Apache Maven, and Google's choice of Gradle as the official build tool for Android applications speaks volumes about international recognition. The Angular CLI, which is available when using Angular, will be used to build the PWA.

The issue of price should also be considered an important factor in choosing a database. Microsoft Azure provides 12 months of free use of the Azure SQL Server S0 DTU 10 service plan and up to 250GB of storage for new users. This is enough for a test deployment of the system. Azure Blob Storage binary storage will be used to store images from user profiles, where 1GB of storage data will cost 2 cents, every 10,000 write operations 5 cents, and every 10,000 read operations 0.4 cents. The Azure Application Service, which integrates easily with GitHub, will be used to deploy Java subsystems, as both services are owned by Microsoft. In addition, all 3 Java subsystems can be deployed for a single Azure Application Service Plan test deployment, ie using shared hardware resources. It was decided to deploy PWA as a docker container using Azure Container Instances, due to the need to establish its own request redirection rules to support virtual routing of the Angular framework. According to the testing pyramid, modular tests make up the majority of all tests and are able to detect errors that cannot be detected in any other way, as such tests are closest to the source code in terms of abstraction. The JUnit 5 library will be used to test Java subsystems. The Mockito library will be used to create stubs. Modular PWA testing will be performed using the karma library, which is immediately available for use in Angular.

COMPONENTS

The PWA subsystem will be a one-page web application created using the Angular framework and the Typescript programming language. The peculiarity of this application will be that it is implemented as a progressive web application, which means that it can be installed on smartphones as a native application. This allows you to access hardware resources (limited), receive push notifications, work offline and launch the application using the icon on the desktop. Until recently, a major obstacle to the proliferation of progressive web applications on Android smartphones was the lack of the ability to download from Google Play, which publishes native applications for this OS. However, in 2019, Google introduced Trusted Web Activity technology, which allows you to publish native applications on Google Play that will use PWA. Although this project will not use this technology, but it is a good opportunity for further development.

This project uses a component approach to the implementation of user interface elements. This means that the interface elements are independent of each other and can be reused. The project consists of three modules: "shared", "dashboard" and "login". The first contains directives and classes that do not directly apply to any UI element, but are used by them. The second contains all the components of the interface that are needed from the main screen of the application. The third includes components, services and templates that provide user authentication and authorization.

The Api Service subsystem is implemented as a Spring Boot application, applying the principles of multilevel architecture. So this part of the system is a server software, which consists of three levels:

- level of representation;
- level of business logic;
- level of data access.

The presentation level is implemented in the form of a REST interface and is documented using the OpenApi specification. The generated documentation is shown in Figure 3.1. In this way, the PWA subsystem gets access to the implemented functions, and the documentation generated immediately from the code is always relevant and speeds up the development of other systems that require access to this REST interface.

The documentation shows the name of the project and its license. License in this case MIT. MIT is an open source software license that entitles you to free use, copying, modification, distribution and sale by third parties subject to copyright. The generated documentation allows not only to see which REST resources are available for use, but also allows you to immediately test them, ie performs the functions of HTTP-client. Because

REST resources are protected, i.e. require prior authorization before accessing them, the generated documentation must provide an authorization mechanism using a JWT token.

The Protocol Engine subsystem is a server software developed using the Spring Boot framework. This project is built on similar principles to the Api Service project, but in this implementation is simpler. Protocol Engine is also implemented on the principles of multilevel architecture, which used 2 levels: level of representation; level of business logic.

The level of access to data was not realized due to the lack of need to store historical data on the analysis of user measurements.

The presentation level is implemented in the form of a REST interface and is documented similarly to Api Service. The only difference here is the lack of mandatory token authorization. This implementation is due to the fact that this subsystem will be deployed on a local network, which prevents access from other networks, such as the Internet.

The level of business logic provides verification of measurement data in accordance with the provisions of clinical protocols, which are provided in the form of a class library. Each protocol presented in a separate class is created as a component of the Spring ioc container, which allows their use using the principle of dependency injection.

TESTING

Testing should be performed in accordance with the pyramid of test automation. Thus, the basic logic of subsystems should be tested using modular tests, and integration tests related to the use of other subsystems or access to the database gonna be created as needed, such as testing the level of access to data. It was decided to test the user interface, namely the PWA subsystem, manually with no automation, because deploying, creating and maintaining an environment for automated tests would be costly not only in terms of financial resources required to operate such tests in Microsoft Azure, but also in terms of time spent, which may be not less than the time spent on the development of the system. Total number of tests isn’t limited, but they should at least reach target coverage and be maintainable. Tests are run when building projects using the Gradle tool and are required before submitting the source code to the remote Github repository. Also, tests are launched when updating the "development" branch using the implemented process of continuous integration. The process of continuous integration is implemented using the Circle CI cloud service, which provides free access to the construction of continuous integration chains (CI pipeline) for open source projects under a certain quota for the use of hardware resources. The main steps performed by Circle CI are shown in figure below. After updating the "develop" branch, Github informs Circle CI that it is starting the workflow.