МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра комп'ютерних наук

Пояснювальна записка

до кваліфікаційної роботи
другого (магістерського) рівня

на тему: **МЕТОД ПОШУКУ ПАТОЛОГІЙ СЕРЦЯ ЗА ЗОБРАЖЕННЯМ ЕЛЕКТРОКАРДІОГРАМИ З ВИКОРИСТАННЯМ ANDROID STUDIO**

Виконав: студент 2 курсу, групи ІКК 2.1
спеціальності
122 Ком'ютерні науки
Атаулін Олег Андрійович
Керівник Горбачов В.Е.
Рецензент Педяш В.В.

Одеса – 2023 р.

ДОВІДКА

кафедри КН про виконану магістерську роботу
студента 2 курсу ФКПІ та КН групи ІКК 2.1

Атауліна Олега Андрійовича

на тему "Метод пошуку патологій серця за зображенням електрокардіограми з використанням Android Studio"

Висновок нормоконтролера : пояснювальна записка до кваліф. роботи бак. відповідає вимогам ДСТУ. Оформлено згідно вимог по положенню КПІ

Нормоконтролер к.т.н., доцент (науковий ступінь, вчене звання, посада) 15.12.2023 (підпис, дата) Розенвассер Д.М (і. б. прізвище)

Висновок відповідального за наявність академічного плагіату : згідно з сертифікатом ID 1015695769

Відповідальна особа к.т.н., доцент (науковий ступінь, вчене звання, посада) 15.12.2023 (підпис, дата) Розенвассер Д.М (і. б. прізвище)

Попередня експертиза (захист) магістерської роботи
(бакалаврської роботи чи магістерської роботи)

студ. Атаулін О.А (прізвище і.б.) проведена "15" грудня 2023 р.

Кваліфікаційна робота виконана у повному обсязі. В роботі розроблено метод пошуку патологій серця за зображенням електрокардіограми з використанням Android Studio. Кваліфікаційна робота відповідає вимогам до випускних кваліфікаційних робіт зі спеціальності 122 Комп'ютерні науки та рекомендована до захисту.

Члени комісії

(підпис) к.т.н., доц. Соловйова І.М. (науковий ступінь, вчене звання,посада, прізвище і.б.)

(підпис) к.т.н., доц. Русич О.П. (науковий ступінь, вчене звання, посада, прізвище і.б.)

(підпис) к.т.н., доц. Розенвассер Д.М. (науковий ступінь, вчене звання, посада, прізвище і.б.)

МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра комп'ютерних наук
Освітній ступінь магістр
Галузь знань 12 Інформаційні технології
Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ
завідувач кафедрою КН
к.т.н., доц.
_____ І.М.Соловська
"25" 09 2023 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Атауліну Олегу Андрійовичу

1. Тема Метод пошуку патологій серця за зображенням електрокардіограми з використанням Android Studio

керівник роботи к.т.н., доц., Горбачов Віктор Едуардович
затверджені наказом закладу вищої освіти від 25 вересня 2023 р. № 1959
2. Строк подання студентом роботи 11.12.2023 р.

3. Вихідні дані до роботи науково-методичні розробки вітчизняних та зарубіжних авторів, матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки
Чотири розділу – завдання виконуються за допомогою мови JAVA та XML розмітки
Розділ 1: Аналіз систем контролю порушень функціонування серця
Розділ 2: Опис програмного додатку для моніторингу серцевої діяльності
Розділ 3: Розробка технічної складової програмного додатку
Розділ 4: Тестування технічної складової мобільного додатку

5. Перелік графічного матеріалу (з зазначенням обов'язкових креслень)
Слайд 1 – Медичні технології аналізу роботи серця минулого часу
Слайд 2 – Медичні технології аналізу роботи серця, які ми пропонуємо
Слайд 3 – Реалізація домашньої сторінки додатку
Слайд 4 – Реалізація сторінки "Графік зашумленого ЕКГ сигналу"
Слайд 5 – Реалізація сторінки "Графік постійної ЕКГ сигналу"
Слайд 6 – Реалізація сторінки " Графік пошуку зубців QRS "
Слайд 7 – Реалізація сторінки "Графік аналізу даних ЕКГ"
Слайд 8 – Приклад надісланого на пошту повідомлення лікарю

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1			
2			

7. Дата видачі завдання 25.09.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Вступ	30.09.2023 – 07.10.2023	
2	Аналіз систем контролю порушень функціонування серця	08.10.2023 – 15.10.2023	
3	Опис програмного додатку для моніторингу серцевої діяльності	17.10.2023 – 20.10.2023	
4	Розробка технічної складової програмного додатку	21.10.2023 – 24.10.2023	
5	Тестування мобільного додатку "холтер"	26.10.2023 – 16.11.2023	
6	Практичне застосування JUnit в тестуванні Java-додатків	18.11.2023 – 22.11.2023	
7	Висновки та рекомендації	24.11.2023 – 29.11.2023	
8	Перелік джерел посилання	13.12.2023	
9	Додаток А Перелік копій демонстраційного матеріалу	14.12.2023	

Студент _____ Атаулін О.А.
(підпис)

Керівник роботи _____ Горбачов В.Е
(підпис)

ВІДГУК КЕРІВНИКА

на кваліфікаційну роботу другого (магістерського) рівня

студента Атауліна О. А. на тему: Метод пошуку патологій серця за зображенням електрокардіограми з використанням Android Studio

У наш час питання спостереження за здоров'ям дуже гостро стоїть. На даний момент першу позицію у списку десяти провідних причин смертності, як і раніше, займають серцево-судинні захворювання. Відтак питання своєчасного визначення різних серцево-судинних захворювань дуже актуальне і потребує постійного розвитку технологій для полегшення цього завдання

Студент Атаулін О.А. добре виконав завдання до КР. Робота проводилася значною мірою самостійно. Графік консультацій не порушувався. Завдання на КР виконано повністю. Необхідні для цього розрахунки проведені.

При оформленні пояснювальної записки використовувались комп'ютерні технології.

Під час виконання магістерської роботи студент Атаулін О.А. засвоїв методи вимірювань, показав уміння користуватись навчальною та технічною літературою, ставити та розв'язувати дослідницькі задачі.

Магістерська робота відповідає вимогам до випускних кваліфікаційних робіт магістрів та заслуговує оцінки «відмінно».

Студент Атаулін О.А. заслуговує присвоєння кваліфікації магістр з комп'ютерних наук за заявленою спеціальністю 122 Комп'ютерні науки.

Керівник
к.т.н., доцент кафедри ІТ _____ В. Е. Горбачов

РЕЦЕНЗІЯ

на кваліфікаційну роботу другого (магістерського) рівня
студента Атауліна О. А.
на тему: Метод пошуку патологій серця за зображенням електрокардіограми з використанням Android Studio

Магістерська робота виконана на 98 сторінках текстової частини та містить чотири розділи згідно з завданням на магістерську роботу.

У магістерській роботі студента Атауліна О. А. було розроблено мобільний додаток, було виконано аналітичний огляд існуючих медичних технологій, виявлено основні недоліки та напрямки поліпшення. Розроблено інформаційну систему та модуль рекомендацій з урахуванням основних потреб користувачів.

В роботі Атаулін О. А. показав достатню теоретичну підготовку.

Пояснювальна записка й графічні матеріали виконані охайно й відповідно до вимог ЄСКД, оформлення демонстраційних слайдів якісне.

Зауваження:

- не проведене нефункціональне тестування створеного додатку;
- наведено мало тестів, що використовувались у JUnit.
- алгоритми пошуку QRS комплексу працюють повільно та з високою витратою пам'яті, потрібно покращити роботу алгоритмів

Але названі недоліки не знижують цінності виконаної роботи.

Магістерська робота студента Атауліна О. А. відповідає вимогам до випускних кваліфікаційних робіт магістрів та заслуговує оцінки «відмінно».

Студент Атаулін О. А. заслуговує присвоєння кваліфікації магістр з комп'ютерних наук за заявленою спеціальністю 122 Комп'ютерні науки.

Рецензент
к.т.н., доцент каф. КІтаІТ _____ Л. Г. Йона

UNICHECK
by Turnitin

Ім'я користувача:
Анна Серединко

ID перевірки:
1016010078

Дата перевірки:
15.12.2023 22:00:32 MSK

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
17.12.2023 18:57:25 MSK

ID користувача:
100001433

Назва документа: Магістерська робота. Атаулін

Кількість сторінок: 99 Кількість слів: 15900 Кількість символів: 121199 Розмір файлу: 5.12 MB ID файлу: 1015695769

7.79%
Схожість

Найбільша схожість: 2.26% з джерелом з Бібліотеки (ID файлу: 1015695764)

6.66% Джерела з Інтернету	715	Сторінка 101
2.43% Джерела з Бібліотеки	75	Сторінка 104

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	16

РЕФЕРАТ

Текстова частина магістерської роботи: 82 с., 80 рис., 15 джерел.

ІНФОРМАЦІЙНА СИСТЕМА, ОБРОБКА ДАНИХ, МОБІЛЬНИЙ ДОДАТОК, ЕКГ, МОНІТОРИНГ ХОЛТЕРА.

Об'єктом дослідження є медичні технології у сфері кардіології.

Метою даної роботи є розробка мобільного додатка для аналітики аномальної поведінки людського серця, с нагодою оптимізувати та покращити роботу в програмах даного типу.

Метод дослідження – аналітичний з використання комп'ютерних технологій.

У результаті виконання роботи був розроблен мобільний додаток, був виконан аналітичний огляд існуючих медичних технологій, виявлено основні недоліки та напрямки поліпшення. Розроблено інформаційну систему та модуль рекомендацій з урахуванням основних потреб користувачів.Результати розробки можуть бути використані абсолютно різними користувачами, від простої людини до фірми або компанії

ABSTRACT

The text part of the master's thesis: 82 p., 80 fig., 15 sources.

INFORMATION SYSTEM, DATA PROCESSING, MOBILE APPLICATION, ECG, HOLTER MONITORING.

The object of research is medical technologies in the field of cardiology.

The purpose of this work is to develop a mobile application for the analysis of abnormal behavior of the human heart, with the opportunity to optimize and improve the work in programs of this type.

The research method is analytical with the use of computer technologies.

As a result of the work, a mobile application was developed, an analytical review of existing medical technologies was performed, the main shortcomings and areas for improvement were identified.

The information system and recommendations module were developed taking into account the basic needs of users. The results of the development can be used by completely different users, from an ordinary person to a firm or company

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated development environment (Інтегроване середовище розробки);

ЕКГ – Електрокардіограма;

АВРМ – Ambulatory Blood Pressure Monitoring (Моніторинг амбулаторного артеріального тиску);

UI – User Interface (Користувальницький інтерфейс);

XML – eXtensible Markup Language (Розширена мова розмітки);

ОС – Операційна система;

APK – Android Package (Формат файлів додатків для Android);

POM – Project Object Model (Основний конфігураційний xml файл JAVA додатку);

ПЗ – Програмне забеспечення;

FDA – Food and Drug Administration (Агентство Міністерства охорони здоров'я та соціальних служб США);

USB – Universal Serial Bus (Універсальна послідовна шина).

ВСТУП

У наш час питання спостереження за здоров'ям дуже гостро стоїть. На даний момент першу позицію у списку десяти провідних причин смертності, як і раніше, займають серцево-судинні захворювання. Відтак питання своєчасного визначення різних серцево-судинних захворювань дуже актуальне і потребує постійного розвитку технологій для полегшення цього завдання

Саме з цією метою і був розроблений даний проект, який, крім того, що полегшить людське існування, допоможе великій кількості людей впоратися з серйозними захворюваннями, так і ще виведе медичні технології на новий рівень розвитку.

Крім того, що цей проект полегшить життя простим людям, він поліпшить продуктивність медичного персоналу, оскільки лікарі зможуть аналізувати дані в режимі онлайн і відразу ж вжити певні заходи для допомоги людям, яким це потрібно.

1 АНАЛІЗ СИСТЕМ КОНТРОЛЮ ПОРУШЕНЬ ФУНКЦІОНУВАННЯ СЕРЦЯ

1.1 Опис предметної області та актуальність розробки

З кожним роком медицина невпинно розвивається, і ми бачимо нові винаходи для поліпшення людського самопочуття і здоров'я в цілому. Так, буквально ще чотири століття тому, наші предки не знали досконалої будови серця та його повний функціонал, не розуміли як вчасно визначити ту чи іншу хворобу та які заходи щодо оздоровлення людського організму вжити.

Сьогодні кардіологи мають величезну кількість інструментів, методик та можливостей, щоб ретельно розбиратись у найменших деталях та нюансах серцевого м'яза. В наші дні ми маємо сучасні методи аналітики серцевих захворювань по типу моніторування Холтера, які зчитують інформацію та зберігають її у себе в пам'яті. Шляхом вже подальшої аналітики даної інформації кардіологи можуть визначати хвороби, хоч би якими важкими вони були, і виписувати певні рекомендації та лікування.

Але зупинятись на досягнутому людство не збирається, адже коли справа стосується людського здоров'я, немає межі досконалості технологій.

Світова тенденція зараз така, що розробники намагаються відходити від використання масивних медичних пристроїв, замінюючи їх на маленькі портативні технології. Саме ця технологія і буде реалізована у цій дипломній роботі.

1.2 Метод серцевого тестування ЕКГ

Електрокардіограма (ЕКГ) — це простий тест, який можна використовувати для перевірки серцевого ритму та електричної активності.

Датчики, прикріплені до шкіри, використовуються для виявлення електричних сигналів, які виробляються вашим серцем кожен раз, коли воно буде.

Ці сигнали записуються машиною, таа лікар вивчає їх, щоб визначити, чи є вони незвичайними.

ЕКГ може бути запрошений кардіологом або будь-яким лікарем, який вважає, що у вас можуть бути проблеми із серцем, у тому числі вашими лікуючими лікарями.

Тест може бути проведений спеціально навчальним медичним працівником у лікарні, поліклініці.

ЕКГ часто використовується разом з іншими тестами, щоб допомогти діагностувати та контролювати стани, що впливають на серце.

Його можна використовувати для дослідження симптомів можливої проблеми з серцем, таких як біль у грудях, серцебиття (раптово помітне серцебиття), запаморочення та задишка.

ЕКГ може допомогти виявити:

1) аритмії – коли серце б'ється занадто повільно, занадто швидко або нерегулярно;

2) ішемічну хворобу серця – коли кровопостачання серця блокується або переривається через накопичення жирових речовин серцеві;

3) напади – коли кровопостачання серця раптово блокується;

4) кардіоміопатію – коли стінки серця потовщені або збільшені.

Серію ЕКГ також можна зробити з часом, щоб контролювати людину, у якої вже діагностовано захворювання серця, або приймаючи ліки, які, як відомо, потенційно впливають на серце.

Існує кілька різних способів проведення ЕКГ. Як правило, тест включає в себе прикріплення ряду маленьких липких датчиків, які називаються електродами, до ваших рук, ніг і грудей. Вони з’єднані проводами з апаратом для запису ЕКГ.

Для підготовки до тесту не потрібно нічого особливого робити. Попередньо можна їсти і пити як зазвичай.

Перш ніж прикріпити електроди, зазвичай потрібно зняти верхній одяг, а грудну клітку, можливо, потрібно поголити або почистити. Коли електроди будуть на місці, вам можуть запропонувати лікарняний халат, щоб прикритися.

Сам тест зазвичай триває лише кілька хвилин, і незабаром після цього ви зможете повернутися додому або повернутися в палату, якщо вже перебуваєте в лікарні.

1.3 Холтер моніторування серця

Холтерівський монітор є різновидом портативної електрокардіограми (ЕКГ). Він реєструє електричну активність серця безперервно протягом 24 годин або довше, поки ви знаходитесь далеко від кабінету лікаря. Стандартна ЕКГ у стані спокою є одним із найпростіших і найшвидших тестів, які використовуються для оцінки роботи серця. Електроди (маленькі пластикові плями, які прилипають до

шкіри) розміщуються в певних точках на грудях і животі. Електроди з'єднані з апаратом ЕКГ проводами. Потім електричну активність серця можна виміряти, записати та роздрукувати. В організм не надходить електрика.

Природні електричні імпульси координують скорочення різних відділів серця. Завдяки цьому кров тече так, як має. ЕКГ записує ці імпульси, щоб показати, наскільки швидко б'ється серце, ритм серцевих скорочень (постійний або нерегулярний), а також силу та час електричних імпульсів. Зміни на ЕКГ можуть бути ознакою багатьох захворювань серця.

Ваш медичний працівник може попросити провести холтерівську ЕКГ, якщо у вас є такі симптоми, як запаморочення, непритомність, низький кров'яний тиск, постійна втома (втома), серцебиття, а ЕКГ у спокої не показує чіткої причини. Ви носите такі ж пластирі електродів ЕКГ на грудях, а електроди з'єднані проводами з невеликим портативним записуючим пристроєм.

Деякі аритмії (аномальні серцеві ритми) можуть виникати лише час від часу. Або вони можуть виникати лише за певних умов, таких як стрес або активність. Аритмії цього типу важко зафіксувати на ЕКГ, зробленому в офісі. Через це медичний працівник може запросити холтерівський монітор, щоб отримати кращі шанси фіксувати будь-які аномальні серцебиття або ритми, які можуть викликати симптоми. Деякі монітори Холтера також мають функцію моніторингу подій, яку ви активуєте, коли помічаєте симптоми.

Ви отримаєте інструкції щодо того, як довго вам потрібно буде носити монітор (зазвичай від 24 до 48 годин), як вести щоденник своєї діяльності та симптомів під час тесту, а також інструкції з особистого догляду та активності, які включають утримання пристрою сухим під час ти його носиш.

1.4 Добовий моніторинг артеріального тиску

Амбулаторний моніторинг артеріального тиску (АВРМ) є відносно новою методикою для оцінки артеріального тиску людини. АВРМ дозволяє медичним працівникам оцінювати ваш артеріальний тиск під час повсякденного життя, а не тоді, коли ви нервово сидите на оглядовому столі лікаря.

АВРМ є найбільш корисним у вирішенні того, чи дійсно людина страждає на гіпертонію, коли показники артеріального тиску, зроблені в кабінеті медичних працівників, сильно варіюються або викликають незрозумілість. Зокрема, АВРМ використовувався для оцінки людей з «гіпертонією білого халата», викликаною стресом під час медичного прийому.

Амбулаторний моніторинг артеріального тиску здійснюється за допомогою спеціального пристрою, який складається з манжети артеріального тиску, яка надягається на руку та прикріплюється до невеликого записуючого пристрою, який ви носите на поясі.1 Ви носите пристрій АВРМ протягом 24 або 48 годин. , і він періодично реєструє ваш кров'яний тиск (зазвичай з інтервалом в 15 або 30 хвилин) протягом цього періоду, під час вашої повсякденної діяльності та під час сну.

Таким чином, АВРМ надає вашому постачальнику медичних послуг повний запис вашого артеріального тиску протягом одного або двох днів.

Інформація, яку надає АВРМ, суттєво відрізняється від інформації, яку отримує медичний працівник, вимірюючи артеріальний тиск в офісі. Запис артеріального тиску в офісі — це єдине значення, яке має відображати ваш артеріальний тиск під час спокійного відпочинку (це пояснює, чому, враховуючи напружене середовище в офісах більшості медичних працівників сьогодні, показання можуть бути не завжди точними).

АВРМ, навпаки, повідомляє про ваш кров'яний тиск, оскільки вони отримані в результаті різноманітних ситуацій і видів діяльності — від бігу до автобуса до сну. І це нормально, коли кров'яний тиск людини надзвичайно коливається під час багатьох видів діяльності, які людина зазвичай виконує протягом дня. Отже, на відміну від артеріального тиску, який ви отримуєте в кабінеті медичного працівника, АВРМ не повідомляє лише одне значення для систолічного та діастолічного артеріального тиску, яке нібито представляє ваш офіційний «артеріальний тиск». Замість цього він повідомляє про весь діапазон (часто) широко змінних значень протягом дня або довше.

1.5 Ультразвукове дослідження серця Ехокардіографія

Ехокардіограма, або «ехо», — це сканування, яке використовується для огляду серця та найближчих кровоносних судин.

Це тип ультразвукового сканування, що означає, що невеликий зонд використовується для відправки високочастотних звукових хвиль, які створюють відлуння, коли вони відбиваються від різних частин тіла.

Ці відлуння виловлюються зондом і перетворюються на рухоме зображення на моніторі під час сканування.

Ехокардіограму може попросити кардіолог (кардіолог) або будь-який лікар, який вважає, що у вас можуть бути проблеми з серцем, включаючи вашого лікаря загальної практики.

Тест зазвичай проводиться в лікарні або клініці кардіологом, кардіофізіологом або кваліфікованим техніком, який називається сонографістом.

Хоча вона має схожу назву, ехокардіограма не те саме, що електрокардіограма (ЕКГ), яка є тестом, який використовується для перевірки ритму серця та електричної активності.

Ехокардіограма може допомогти діагностувати та контролювати певні серцеві захворювання, перевіряючи структуру серця та навколишніх кровоносних судин, аналізуючи те, як кров тече через них, та оцінюючи насосні камери серця.

Ехокардіограма може допомогти виявити:

1) пошкодження від серцевого нападу, коли кровопостачання серця було раптово заблоковано серцева;

2) недостатність – коли серце не може перекачувати достатню кількість крові по тілу при правильному тиску;

3) вроджені вади серця – вроджені дефекти, які впливають на нормальну роботу серця;

4) проблеми з клапанами серця – проблеми, що впливають на клапани, які контролюють потік крові в серці;

5) кардіоміопатія – коли стінки серця потовщені або збільшені;

6) ундокардит – інфекція в оболонці серця, яка пошкоджує серцеві клапани Ехокардіограма також може допомогти вашим лікарям визначити найкраще лікування для цих станів.

1.6 Постановка мети та задачі

Виходячи з усієї вищезгаданої інформації можна зробити висновок, що більшість методів дослідження добре використовуються на практиці і актуальні досі.

Але більшість з них не зовсім модернізовані під реалії нашої епохи інформаційних технологій, що швидко і стрімко розвивається. У наш час, коли майже кожна сучасна людина має мобільний пристрій, дуже важливо використовувати технології за призначенням, одне з таких призначень – стеження за здоров'ям.

Маючи величезний технологічний потенціал, ми не користуємося їм належним чином. У сучасному світі, щоб пройти обстеження, людині потрібно записатися на прийом до лікаря, як мінімум відвідати кілька разів медичний заклад, пройти безліч оглядів та тестів, що є величезною тратою часу для нас. Найближчий

варіант до зручного це моніторинг Холтер, але у нього, на мій погляд є кілька недоліків, таких як:

1) він записує інформацію, але не передає її лікувальному лікарю в режимі онлайн;

2) пристрій холтер досить громіздкий.

Саме тому головною метою даного проекту є створення мобільного додатка, яке за допомогою портативного пристрою, який через блютуз буде передавати дані безпосередньо в мобільний додаток, програма буде аналізувати дані і безпосередньо відсилати результати медику онлайн, при цьому сам блютуз пристрій буде в рази менше і зручніше, ніж пристрій Холтер.

2 ОПИС ПРОГРАМНОГО ДОДАТКУ ДЛЯ МОНІТОРИНГУ СЕРЦЕВОЇ ДІЯЛЬНОСТІ

2.1 Опис середовища розробки

Для реалізації свого проекту я вибрав інтегроване середовище розробки – **Android Studio**. На даний момент найпопулярніша платформа для створення мобільних додатків для систем Android, і відверто кажучи, що беззастережно перевершує всіх своїх конкурентів

Android Studio — інтегроване середовище розробки виробництва Google, за допомогою якого розробникам стають доступні інструменти для створення програм на платформі Android OS. Android Studio можна встановити на Windows, Mac та Linux. Обліковий запис розробника програм у Google Play App Store коштує $25. Android Studio створювалася з урахуванням IntelliJ IDEA.

IDE можна завантажити та користуватися безкоштовно. У ній є макети для створення UI, з чого зазвичай починається робота над додатком. Studio містить інструменти для розробки рішень для смартфонів і планшетів, а також нові технологічні рішення для Android TV, Android Wear, Android Auto, Glass і додаткові контекстуальні модулі.

Середовище Android Studio призначене як для невеликих команд розробників мобільних додатків (навіть у кількості однієї людини), або великих міжнародних організацій з GIT або іншими подібними системами управління версіями. Досвідчені розробники зможуть вибрати інструменти, які найбільше підходять для масштабних проектів. Рішення для Android розробляються в Android Studio за допомогою Java або C++. В основі робочого процесу Android Studio закладено концепт безперервної інтеграції, що дозволяє відразу виявляти наявні проблеми. Тривала перевірка коду забезпечує можливість ефективного зворотного зв'язку з розробниками. Така опція дозволяє швидше опублікувати версію мобільного додатка в Google Play App Store. Для цього є також підтримка інструментів LINT, Pro-Guard і App Signing.

За допомогою засобів оцінки продуктивності визначається стан файлу з пакетом прикладних програм. Візуалізація графіки дає змогу дізнатися, чи програма відповідає орієнтиру Google в 16 мілісекунд. За допомогою інструмента для візуалізації пам'яті розробник дізнається, коли його додаток буде використовувати занадто багато оперативної пам'яті і коли відбудеться складання

сміття. Інструменти для аналізу батареї показують, яке навантаження посідає пристрій.

Android Studio сумісна з платформою Google App Engine для швидкої інтеграції у хмари нових API та функцій. У середовищі розробки ви знайдете різні API, такі як Google Play, Android Pay та Health. Є підтримка всіх платформ Android, починаючи з Android 1.6. Є варіанти Android, які суттєво відрізняються від версії Google Android. Найпопулярніша з них – це Amazon Fire OS. В Android Studio можна створювати APK для цієї ОС. Підтримка Android Studio обмежується онлайн-форумами.

Як мову програмування я вибрав **JAVA**, тому що вона є однією з найактуальніших мов програмування в цілому, а також найпопулярнішим у світі розробки мобільних додатків на Android. Зв'язка Android Studio та JAVA, на даний момент найпоширеніша, якісна та зручна для розробників усього світу. Створені з його використанням програми здатні працювати на різних програмно-апаратних платформах: від потужних серверів для бізнесу до смартфонів та планшетів.

При розробці Java під Android використовуються не тільки Java-класи, що містять код, але також файли маніфесту на мові XML, що надають системі основну інформацію про програму, і системи автоматичного складання Gradle, Maven або Ant, команди в яких пишуться мовами Groovy, POM та XML відповідно; За замовчуванням у проектах використовується Gradle, а на початкових етапах навчання розробці на Java редагувати файли, написані на Groovy, практично не доведеться. Для верстки UI-частини зазвичай також використовується мова XML.

Для роботи з мовою програмування Java потрібні високі компетенції. Як у будь-якої об'єктно-орієнтованої мови, він має безліч особливостей і підводних каменів, що призводять до помилок у роботі мобільного додатка. Необхідно враховувати ці нюанси та тонкощі при розробці мобільного додатка на Android. Особливості програмного коду на Java – читання та структурність, наявність прийнятих стандартів його оформлення.

2.2 Обґрунтування вибору засобів розробки

Android Studio перевершує конкурентів за багатьма параметрами, до яких можна віднести:

1) гнучкість середовища розробки;

2) більший набір функцій;

3) процес розробки, що підлаштовується під розробника.

Під час створення програм та утиліт для операційної системи Android, користувач програмного забезпечення може спостерігати за змінами у проекті, в режимі реального часу.

З особливостей IDE можна виділити вбудований емулятор, що дозволяє перевірити коректну роботу програми на пристроях з різними екранами, з різними співвідношеннями сторін. Особливо актуальною ця функція стала після входу в тренди смартфонів, де встановлено екрани зі співвідношенням сторін 18:9.

Відмінна риса емулятора – перегляд приблизних показників продуктивності при запуску програми на найпопулярніших пристроях. Середовище розробки для додатків Android Studio останньої версії стало по-справжньому зручним навіть для розробників-початківців. У програмі реалізовані всі сучасні засоби для пакування коду, його маркування. Затребувана багатьма авторами ПЗ функція Drag-n-Drop, що полегшує перенесення компонентів у середу розробки безпосередньо. Локалізація програм стає значно простіше з функцією SDK, яка також входить до списку переваг Android Studio.

Список переваг утиліти:

1) середовище розробки підтримує роботу з кількома мовами програмування, яких ставляться найпопулярніші – C/C++, Java;

2) редактор коду, з яким зручно працювати;

2) дозволяє розробляти програми не тільки для смартфонів/планшетів, а й для портативних ПК, приставок для телевізорів Android TV, пристроїв Android Wear, новомодних мобільних пристроїв із незвичайним співвідношенням сторін екрану;

3) тестування коректності роботи нових ігор, утиліт, їхньої продуктивності на тій чи іншій системі, відбувається безпосередньо в емуляторі;

4) рефакторинг готового коду;

5) досить велика бібліотека з готовими шаблонами та компонентами для розробки ПЗ;

6) розробка програми для Android N – найостаннішої версії операційної системи;

7) попередня перевірка вже створеного додатка щодо помилок у ньому;

8) великий набір засобів інструментів для тестування кожного елемента програми, ігри;

9) для недосвідчених/початківців розробників спеціально створено посібник з використання Android Studio, розміщений на офіційному сайті утиліти.

Недоліки утиліти:

1) незважаючи на наявність вбудованого Android-емулятора в середовищі розробки, з тестуванням розробленої програми можуть виникнути проблеми. Так, для його запуску необхідна досить велика апаратна основа ПК, на якому планується тестування;

2) ще один недолік – це неможливість написати серверні проекти мовою Java для ПК, Android пристроїв.

Висновок:

ПЗ для розробки утиліт та програми на Android дійсно створює приємне перше враження. Android Studio оцінить як досвідчений розробник, так і початківець, який тільки освоює ази. Багатий набір інструментів, гнучкість у розробці, можливості тестування, підтримка кількох мов програмування та вбудований емулятор роблять утиліту однією з найкращих у своїй ніші.

Основні переваги JAVA:

1) основною перевагою Java є підтримка концепції об'єктно-орієнтованого програмування (ООП). Це дозволяє писати розділені і повторно використовувані програмні компоненти, будуючи сувору ієрархію програм;

2) java поставляється з бібліотекою шаблонів проектування з відкритим вихідним кодом, а також дозволяє використовувати передові практики, що адаптуються для розробки серверних, настільних, вбудованих та мобільних додатків;

3) кросплатформеність – Інші мови програмування тією чи іншою мірою прив'язані до функцій програмно-апаратних платформ, але слоган Java говорить: "Напиши один раз, запускай де завгодно". Кросплатформенність сприяє поширенню мови. З 1990 року Java використовується як платформа для розробки мобільних додатків і досі залишається однією з найбільш популярних мов програмування в цій сфері;

4) підтримка спільноти. Спільнота Java допомагає програмістам у вирішенні проблем. Наприклад, форуми для надсилання питань Stack Overflows та інших груп користувачів надають розширену підтримку з різних тем;

5) доступність потужних інструментів. Java підтримується багатьма популярними середовищами розробки (IDE), включаючи Eclipse, NetBeans та Jet Brains. Такі інструменти як Eclipse і NetBeans відіграють вирішальну роль у перетворенні Java на одну з найкращих мов програмування для мобільної розробки.

Надійний набір інструментів не тільки допомагає в кодуванні, але й дозволяє впливати на налагодження, яке необхідно для виключення помилок під час процесу розробки. Інтегроване середовище зробило розробку на Java набагато зручніше та

швидше. При використанні IDE легко шукати та читати код, а також виконувати його рефакторинг.

Висока окупність вкладених у розробку мобільного додатку інвестицій – ще одна важлива причина популярності Java.

Світовий ринок повний мобільних додатків, написаних на JAVA. Яскравими прикладами дуже популярних додатків є: Spotify, Twitter, Opera Mini, Nimbuzz Messenger, CashApp та інші

Недоліки JAVA:

1) це досить складна мова у плані синтаксису, що збільшує ймовірність помилок та багів;

2) java-розробники стикаються з проблемами при розробці програм на базі Android API через певні обмеження в коді;

3) потрібно більше пам'яті порівняно з програмами Kotlin.

Висновок:

Java вважається фундаментальною мовою розробки програм для Android. Він також дозволяє розробникам писати код, що без проблем працює на кількох мобільних платформах. У реальному світі існують програми Java у різних галузях, таких як ігри, обмін миттєвими повідомленнями, потокова передача музики та торгівля.

2.3 Опис новітніх технологій трекінгу ЕКГ

ЕКГ-монітор вимірює та записує частоту та ритм серця, а також відображає цю інформацію у вигляді хвилі. Багато моніторів ЕКГ доступні для домашнього використання.

У даному проекті ми не будемо використовувати моніторинг холтери та інші застарілі технології аналізу роботи серця, у наші дні з'явилися сучасні трекери, які в рази менші та набагато зручніші у використанні

Клінічні або лікарняні ЕКГ-монітори мають відведення або дроти та пристрої, які називаються електродами. Медичний працівник приклеїть електроди до різних частин вашого тіла, щоб виміряти ваш серцевий ритм. ЕКГ для домашнього або особистого користування, які є типом споживчої електроніки, зазвичай мають вбудовані датчики. Ви можете притиснути один або два пальці до датчиків або носити датчики на зап’ясті або тілі. Датчики схожі на електроди, які вловлюють і записують електричну активність вашого серця. Деякі ЕКГ-пристрої персонального користування мають вбудовані екрани, щоб ви могли бачити свій

серцевий ритм на моніторі. Інші пристрої підключаються до мобільного додатку або комп’ютера, де ви можете записувати, переглядати, зберігати та обмінюватися показаннями ЕКГ. Саме такий мобільний додаток ми і будемо розробляти для спільної роботи з сучасними екг моніторами

Розглянемо особливості семи моніторів від надійних брендів і що потрібно знати про ЕКГ–пристрої

EMAY Portable ECG Monitor.

Особливості:

1) цей монітор сумісний з усіма смартфонами;

2) компактний розмір досить маленький, щоб поміститися у вашій кишені;

3) він простий у використанні, без проводів;

4) він записує 30 секунд вашого серцебиття та ритму;

5) за допомогою цього пристрою ви можете зберігати, переглядати та ділитися даними про здоров’я серця на своєму смартфоні або комп’ютері.

Трекер EMAY Portable ECG Monitor відображено на рисунку 2.1.

Рисунок 2.1 – Графічне відображення EMAY Portable ECG Monitor

1byone Portable Wireless ECG Monitor.

Особливості:

1) цей монітор портативний і досить маленький, щоб носити його в кишені;

2) легко записуйте пульс і форму сигналу в будь-якому місці;

3) записуйте пульс і ЕКГ протягом до 30 секунд за раз;

4) надсилайте записи собі електронною поштою або синхронізуйте їх із програмою для смартфона;

5) переглядайте, друкуйте та керуйте своїми даними з програми для смартфона чи комп’ютера;

6) цей монітор перезаряджається.

Трекер 1byone Portable Wireless відображено на рисунку 2.2.

Рисунок 2.2 – Графічне відображення 1byone Portable Wireless ECG Monitor

Omron Complete Wireless Upper Arm Blood Pressure Monitor + ECG.

Особливості:

1) управління з контролю за продуктами і ліками (FDA) підтвердило, що цей пристрій по суті еквівалентний пристроям медичного класу;

2) монітор Omron вимірює артеріальний тиск і частоту серцевих скорочень, одночасно аналізуючи ваш серцевий ритм;

3) він синхронізується з безкоштовною програмою для смартфона Omron Connect;

4) за допомогою монітора ви можете зберігати, відстежувати та ділитися даними про здоров’я серця.

Трекер Omron Complete Wireless Upper Arm Blood Pressure Monitor + ECG відображено на рисунку 2.3.

Рисунок 2.3 – Графічне відображення Omron Complete Wireless Upper Arm Blood Pressure Monitor + ECG

Eko DUO ECG + Digital Stethoscope.

Особливості:

1) це медичний прилад, який можна використовувати вдома;

2) він посилює тони серця і легенів у 60 разів;

3) монітор показує звукові хвилі серця та ЕКГ;

4) ви можете використовувати його без підключення до смартфона чи комп’ютерного додатка;

5) записи ЕКГ можна зберігати та надсилати електронною поштою;

6) він має один провідник і простий у використанні;

7) акумуляторна батарея залишається зарядженою протягом 9 годин;

8) заряджати пристрій можна за допомогою USB-кабелю та адаптера або бездротової зарядної панелі.

Трекер Eko DUO ECG + Digital Stethoscope відображено на рисунку 2.4.

Рисунок 2.4 – Графічне відображення Eko DUO ECG + Digital Stethoscope

Biocare 12-Lead ECG Machine.

Особливості:

1) це медичний пристрій, який може використовуватися вдома медсестрею або іншим медичним працівником;

2) ви можете використовувати його без підключення до смартфона чи комп’ютерного додатка;

3) він компактний, портативний і досить легкий, щоб носити його з собою;

4) ви можете переглянути свою ЕКГ на моніторі або за допомогою роздруківки;

5) монітор має високу чутливість, щоб легко знаходити нерегулярні серцеві ритми або аритмії;

6) акумуляторна батарея має термін служби 3 години.

Трекер Biocare 12-Lead ECG Machine відображено на рисунку 2.5.

Рисунок 2.5 – Графічне відображення Biocare 12-Lead ECG Machine

Omron KardiaMobile ECG.

Особливості:

1) це невелика і непомітна персональна ЕКГ з одним відведенням;

2) він вимірює серцеві ритми всього за 30 секунд;

3) він підключається до будь-якого смартфона;

4) він медичний і портативний.

Трекер Omron KardiaMobile ECG відображено на рисунку 2.6.

Рисунок 2.6 – Графічне відображення Omron KardiaMobile ECG

Wellue Portable ECG Monitor.

Особливості:

1) це портативний монітор, який також можна носити з нагрудним ремінцем;

2) він може вимірювати вашу ЕКГ від 30 секунд до 15 хвилин;

3) використовуйте його зі смартфоном або без нього;

4) він має безкоштовний додаток для смартфона;

5) синхронізуйте його з додатком для смартфона з Bluetooth, щоб переглянути свої дані;

6) за допомогою цього монітора ви можете зберігати, записувати та обмінюватися даними про здоров’я серця.

Трекер Wellue Portable ECG Monitor відображено на рисунку 2.7.

Рисунок 2.7 – Графічне відображення Wellue Portable ECG Monitor

Для даного проекту був обраний трекер Omron KardiaMobile ECG, так як він дуже зручний у користуванні, він є найменшим пристроєм стеження за серцевою активністю, в матеріальному плані він дуже бюджетний, а також розробнику медичних мобільних додатків дуже зручно взаємодіяти з цим пристроєм завдяки дуже якісно написаної документації для розробників.

2.4 Опис функціоналу мобільного додатка

Основний функціонал мобільного додатка "Холтер" полягає у зчитуванні даних з пристрою, графічної побудови електрокардіограм з вхідних даних, коригування даних та графіків, та подальшої аналітики результатів.

Всі побудови графіків і аналітика даних відбувається в режимі Live, щоб користувач бачив актуальний прогрес виконання програмного коду та після досягнення мети повністю.

Користувач має можливість скористатися програмно реалізованим меню навігації, щоб переходити з одного вікна програми до інших.

На графіках реалізовані додаткові функції такі як "наближення/віддалення" та "прокручування" графіка. Також реалізована демо-версія графіка з аналітикою даних, в якому користувач може власноруч додавати або прибирати шум на

вхідний сигнал ЕКГ, щоб протестувати програмну реалізацію аналітики вхідного сигналу.

Графічне відображення зчитування перших 250 значень ЕКГ сигналу на першому графіку зображено на рисунку 2.8.

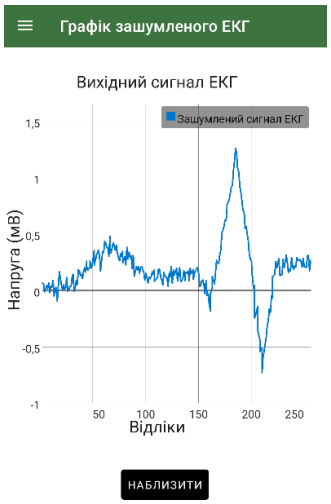


Рисунок 2.8 – Графічне відображення зчитування перших 250 значень ЕКГ сигналу на першому графіку

Графічне відображення початкової побудови постійної складової ЕКГ з перших 250 значень ЕКГ сигналу на другому графіку зображено на рисунку 2.9.

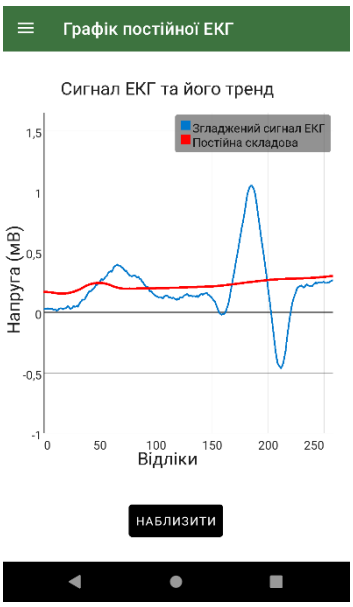


Рисунок 2.9 – Графічне відображення початкової побудови постійної складової ЕКГ з перших 250 значень ЕКГ сигналу на другому графіку

Графічне відображення початкової аналітики першого серцебиття вхідного сигналу ЕКГ на третьому графіку зображено на рисунку 2.10.

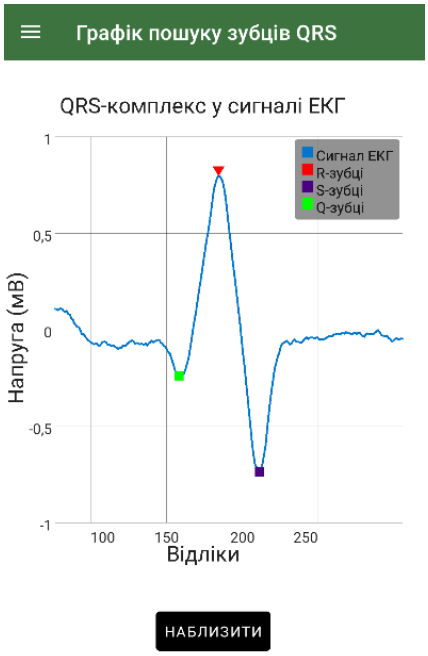


Рисунок 2.10 – Графічне відображення початкової аналітики першого серцебиття вхідного сигналу ЕКГ на третьому графіку

3 РОЗРОБКА ТЕХНІЧНОЇ СКЛАДОВОЇ ПРОГРАМНОГО ДОДАТКУ

3.1 Розробка заставки мобільного додатка

Початок роботи нашого мобільного додатка починається з відображення стартової сторінки. На стартовій сторінці ми бачимо анімований логотип програми та назву самої програми

Контент стартового екрану нашої програми складається з двох елементів: зображення та текстове поле. Відображення зображення в середовищі розробки Android Studio відбувається за допомогою компонента ImageView. А відображення тексту відбувається за допомогою компонента TextView. Моделювати характеристики даних компонентів можна за допомогою коду JAVA, а також за допомогою XML розмітки. Найкращий варіант використання, це міксувати обидва варіанти.

Ініціалізація компонентів у XML розмітці зображена на рисунку 3.1.

```
<ImageView
    android:id="@+id/tv_img"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:scaleType="fitCenter"
    android:src="@drawable/pngtree2" />

<TextView
    android:id="@+id/tv_splash"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:fontFamily="@font/anton"
    android:text="@string/halter_application"
    android:textSize="35sp"
    android:textColor="@color/prime_green" />
```

Рисунок 3.1 – Ініціалізація компонентів ImageView та TextView у XML розмітці

Де “android:id” – це атрибут Android Studio, який відповідає за приєднання компоненту унікального ідентифікатора, та його властивість – "@+id/example_id",

яка за допомогою синтаксису @+id створює у кореневому класі Android Studio сам ідентифікатор компонента з найменуванням “example_id”.У нашому прикладі ми вказуємо ідентифікатори "tv_img" та "tv_splash";

“android:layout_width” – це атрибут Android Studio, який відповідає за ширину нашого компонента на екрані смартфона, та його властивість – "wrap_content", яка робить ширину компонента такою, щоб контент усередині цього компонента був повністю відображений;

“android:layout_height” – це атрибут Android Studio, який відповідає за довжину нашого компонента на екрані смартфона, та його властивість – "wrap_content", яку ми описали вище;

“android:layout_gravity” – це атрибут Android Studio, який відповідає за розташування компонента щодо екрану, та його властивість – "center_horizontal", яка встановлює місце розташування компонента в середині екрана по горизонталі;

“android:scaleType” – це атрибут Android Studio, який відповідає за масштабування зображення, та його властивість – "fitCenter", яка масштабує зображення, зберігаючи пропорції;

“android:src” – це атрибут Android Studio, який відповідає за завантаження зображення, та його властивість – "@drawable/pgntree2", яка за допомогою символу "@" дає зрозуміти нашому додатку, що наступне написане значення – це посилання на зображення, у нашому випадку зображення “pngtree2” лежить у папці “drawable”, в якій знаходяться всі статичні зображення додатка;

“android:fontFamily” – це атрибут Android Studio, який відповідає за встановлення нашому компоненту певного шрифту написання, часто він використовується для текстових фрагментів, та його властивість – “@font/anton”, яка за допомогою символу "@" , як ми вже описали вище, посилається на шрифт "anton" у папці "font", в якій знаходяться всі шрифти додатка;

“android:text” – це атрибут Android Studio, який відповідає за відображення тексту в компоненті, та його властивість – “@string/halter_application”, яка за допомогою символу "@" , як ми вже описали вище, посилається на текстовий рядок “halter_application” у папці "string", в якій знаходяться всі текстові константи додатку;

Усі текстові константи програми потрібно виносити в папку "strings", тому що якщо вам захочеться змінити текст у всіх фрагментах у програмі, а ви писатимете один і той же текст у різних місцях програми і при цьому не винесіть його в константу, то вам доведеться в кожному місці вручну його переписувати, що

є не раціональним рішенням. А також це потрібно робити для подальшої локалізації мобільного додатка;

“android:textSize” – це атрибут Android Studio, який відповідає за встановлення розміру тексту для нашого компоненту, та його властивість – “35sp”, яка позначає розмір тексту, де “sp” – це Scale-independent Pixels, абстрактна одиниця вимірювання, що дозволяє програмам виглядати однаково на різних екранах та дозволах. Одиниця виміру “sp” використовується для розміру шрифтів;

“android:textColor” – це атрибут Android Studio, який відповідає за встановлення кольору для тексту нашого компоненту, та його властивість – “@color/prime_green”, яка за допомогою символу "@" , як ми вже описали вище, посилається на колір “prime_green” у папці "color", в якій знаходяться всі кольори додатка.

Крім ініціалізації компонентів у XML розмітці, так само потрібно проініціалізувати компоненти і у JAVA коді, для зручності роботи з атрибутами компонентів.

Ініціалізація компонентів у JAVA коді зображена на рисунку 3.2.

```
private ImageView imageView;
private TextView textView;
```

Рисунок 3.2 – Ініціалізація компонентів ImageView та TextView у JAVA коді

Де “private” – це модифікатор доступу до змінної в JAVA;

“ImageView” та “TextView” – це JAVA класи, в яких реалізовано функціонал наших компонентів;

“imageView” та “textView” – це назва наших змінних.

Після цієї ініціалізації ми все ще не можемо працювати з даними компонентами в JAVA коді, тому що наша програма не знає з якими конкретними компонентами ми будемо працювати, адже TextView та ImageView численна кількість у нашому проекті, тут і вступає у справу унікальний ідентифікатор кожного з компонентів , за яким ми вкажемо нашому середовищу розробки з якими саме компонентами ми будемо працювати.

Присвоєння унікальних ідентифікаторів компонентам зображено на рисунку 3.3.

```
imageView = findViewById(R.id.tv_img);
textView = findViewById(R.id.tv_splash);
```

Рисунок 3.3 – Присвоєння унікальних ідентифікаторів компонентам ImageView та TextView

Де “imageView” та “textView” це назва наших змінних;

“findViewById(int id)” – це метод, який повертає об'єкт класу View за вказаним нами едентифікатором.

Клас View – це об'єкт якогось компоненту на екрані (наприклад, тексту).

Компоненти TextView, ImageView, Button та інші – це підкласи класу View.

Ідентифікатор ми передаємо не зовсім у звичному нам у вигляді, не у вигляді якогось конкретного рядка, а через звернення до класу “R”.

Клас R – це динамічно генерований клас середовищем розробки, створений у процесі складання для динамічної ідентифікації всіх ресурсів (від рядків до шрифтів і кольорів) для використання в класах JAVA в Android.

Таким чином, спочатку ми звертаємося до класу “R”, далі, оскільки нам потрібен саме ідентифікатор, ми звертаємося до поля "id" у класі “R”, ну і наприкінці ми вказуємо "id", який ми вказали в XML розмітці в атрибуті “android:id”, у нашому випадку – “tv_img” та “tv_splash”

Тепер ми можемо редагувати властивості наших компонентів через JAVA код. Наприклад, даним компонентам я додав певну анімацію виведення на екран, тому що якщо компоненти будуть статичними, то це виглядатиме не зовсім презентабельно.

Створення анімації для компонентів зображено на рисунку 3.4

```
imageView.animate().rotationYBy(360f).setDuration(1000).withEndAction(new Runnable() {
    @Override
    public void run() { imageView.animate().rotationXBy(360f).setDuration(1000).start(); }
});
textView.setAnimation(AnimationUtils.loadAnimation( context: this, R.anim.bottom_animation));
```

Рисунок 3.4 – Створення анімації для компонентів

Де “animate()” – це метод, який безпосередньо додає анімацію до нашого компоненту;

“rotationYBy(float value)” – це метод, який провертає наш компонент по осі Y на певну кількість градусів, у нашому випадку ми вказали на “360” градусів;

“setDuration(long duration)” – це метод, який встановлює тривалість нашої анімації в мілісекундах, у нашому випадку “1000” мілісекунд, що рівноцінно одній секунді;

“withEndAction(Runnable runnable)” – це метод, який після закінчення нашої анімації додає будь-яку нову дію до компонента, на вхід даний метод приймає анонімний клас інтерфейсу Runnable, в якому реалізовано метод “run()” з описаним нами функціоналом. Runnable – це інтерфейс, який має бути реалізований класом, екземпляри якого призначені для виконання потоком. У методі “run()” я додав нашому компоненту ще одну анімацію, тільки тепер із проворотом нашого компонента по осі X. Реалізував я цей функціонал завдяки методу "rotationXBy()";

“start()” – це метод який стартує анімацію на компоненті.

Таким чином, я реалізував анімацію для нашого зображення, яке спочатку протягом секунди буде провертатися по вісі Y на триста шістдесят градусів, а потім по вісі X.

Анімація для нашого тексту реалізована інакше.

"setAnimation(Animation animation)" – це метод, який додає анімацію до нашого компоненту. Він приймає на вхід заздалегідь заготовлену анімацію, на відміну від попереднього методу “animate()”, який дозволяє нам створювати анімацію через код JAVA;

“AnimationUtils” – це JAVA клас, який визначає загальні методи для роботи з анімаціями. За допомогою цього класу ми завантажимо нашу готову анімацію;

“loadAnimation(Context context, ind id)” – це метод, який завантажує нашу анімацію. На вхід він приймає дві змінні, це Context та безпосередньо id нашої анімації;

Контекст (Context) – базовий абстрактний клас, реалізація якого забезпечується системою Android. Цей клас має методи для доступу до специфічних для конкретного застосування ресурсів і класів і служить для виконання операцій на рівні програми.

Таким чином, у метод “loadAnimation(Context context, ind id)” ми передаємо ключове слово this, завдяки якому ми отримуємо посилання на об'єкт Context, і передаємо id нашої анімації, яка знаходиться в папці “anim”, за допомогою динамічного класу ресурсів – “R”.

Реалізація анімації для тексту демонструється на рисунку 3.5.

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:duration="1500"
        android:fromXDelta="0%"
        android:fromYDelta="100" />
    <alpha
        android:duration="1500"
        android:fromAlpha="0.1"
        android:toAlpha="1.0" />
</set>
```

Рисунок 3.5 – Реалізація анімації для тексту

Де “<set>” – це тег, що відповідає за додавання безлічі інших тегів, він є тілом для нашої анімації;

“<translate>” – це тег, що відповідає за переміщення компонента на екрані.

Його атрибути:

1) “android:duration” – це атрибут Android Studio, який відповідає за тривалість відображення анімації, та його властивість – “1500”, яка встановлює тривалість відображення анімації в тисячу п'ятсот мілісекунд;

2) “android:fromXDelta” – це атрибут Android Studio, який відповідає за початкове зміщення компонента по осі X, та його властивість – “0%”, яка встановлює початкове зміщення компонента по осі X в нуль відсотків від ширини батьківського компонента;

3) “android:fromYDelta” – це атрибут Android Studio, який має такий самий функціонал як і атрибут вище, але тільки щодо осі Y, та його властивість – “100”, яка встановлює початкове зміщення компонента по осі Y у сто пікселів щодо початкового розташування компонентаю.

“<alpha>” – це тег, що відповідає за встановлення прозорості компоненту.

Його атрибути:

1) “android:duration=1500” – описав цей атрибут та його властивість вище;
2) “android:fromAlpha” – це атрибут Android Studio, який відповідає за встановлення прозорості компонента при його початковому відображенні, та його властивість – “0.1”, яка встановлює компоненту майже повну прозорість;

3) “android:toAlpha” – це атрибут Android Studio, який відповідає за встановлення прозорості компонента у кінці його відображення, та його властивість – “1.0”, яка встановлює компоненту повну видимість.

Ця анімація буде використана на більшій частині сторінок програми для багатьох візуальних компонентів, оскільки вона додає нашому додатку гарний візуал.

З візуалізацією нашої заставки мобільного додатка ми закінчили, результат візуалізації без анімації можна побачити на на рисунку 3.6, візуалізація з анімацією буде продемонстрована в режимі онлайн.

Рисунок 3.6 – Результат візуалізації без анімації

Крім візуалізації, на стартовій сторінці програми, так само реалізований функціонал перевірки на умову – чи є клієнт новим користувачем програми. Якщо людина перший раз завантажила додаток, то переходить на сторінку реєстрації. В

іншому випадку, якщо користувач вже раніше запускав програму і пройшов реєстрацію, то після стартової сторінки він відразу потрапляє на домашню сторінку програми.

Реалізацію цього функціоналу можна побачити на на на рисунку 3.7.

```
new Handler().postDelayed(new Runnable() {
    @Override
    public void run() {
        try {
            openFileInput(USER_DATA);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            new Handler().post(new Runnable() {
                @Override
                public void run() {
                    startActivity(new Intent(getApplicationContext(), UserDataActivity.class));
                    finish();
                }
            });
        }
        startActivity(new Intent( packageContext: SplashScreen.this, HomeActivity.class));
        overridePendingTransition(R.anim.slide_in_right, R.anim.slide_out_left);
        finish();
    }
}, delayMillis: 4000);
```

Рисунок 3.7 – Реалізація перевірки на умову – чи є клієнт новим користувачем програми

Де “new Handler()” – це ініціалізація нового об'єкту класу Handler;

Handler – це механізм, який дозволяє працювати з чергою повідомлень. Він прив'язаний до конкретного потоку (thread) та працює з його чергою. Handler вміє поміщати повідомлення у чергу. При цьому він ставить себе як одержувач цього повідомлення. І коли настає час, система дістає повідомлення з черги і відправляє його адресату (тобто Handler) на обробку. Крім надсилання повідомлень у чергу потоку, Handler також може відправляти Runnable об'єкти, які ми раніше використовували у проекті.

“postDelayed(Runnable runnable, long delayMillis)” – це метод об'єкту Handler, який додає в чергу потоку анонімний клас інтерфейсу Runnable із затримкою в мілісекундах, у нашому випадку ми передаємо реалізований анонімний клас інтерфейсу Runnable, та затримку в “4000” мілісекунд;

У методі “run()”, за допомогою “try,catch” блоку, який використовується для обробки винятків у JAVA, ми перевіряємо умову викиду винятка.

У світі програмування виникнення помилок та непередбачених ситуацій під час виконання програми називають винятком. У програмі виключення можуть

виникати внаслідок неправильних дій користувача, відсутності ресурсу в пам'яті пристрою, або втрати з'єднання з сервером по мережі. Саме винятки другого типу ми і перевірятимемо.

У блоці “try”, за допомогою методу “openFileInput(String name)”, який приймає аргумент - найменування фіалу з даними, ми намагаємося відкрити файл на нашому мобільному емуляторі. Якщо ж користувач вперше запустив програму, у нас викинеться виняток, який далі ми обробимо в блоці catch, а якщо ж файл відкриється, то жодного виключення не викинеться і програма далі продовжить свою роботу.

У блоці “catch”, який приймає аргумент - виняток, спочатку ми виводимо в консоль всю помилку виключення. Робимо ми це за допомогою рядка коду – “e.printStackTrace()”, де e – це виняток. Потім ми знову створюємо новий об'єкт класу Handler, викликаємо в нього метод “post()”, який виконує той же функціонал, що і “postDelayed()”, але робить це без затримки, і передаємо у метод “post()” анонімний клас інтерфейсу Runnable.

У методі run() я використав декілька нових методів.

“startActivity(Intent intent)” – це метод використовується для запуску нової сторінки додатку, де Intent – це механізм для опису однієї операції – вибрати фотографію, надіслати листа, зробити дзвінок, запустити браузер та перейти за вказаною адресою. В Android-додатках багато операцій працюють через наміри. Найпоширеніший сценарій використання Intent - запуск іншої сторінки додатку.

Таким чином, ми передаємо в метод новий Intent, який при ініціалізації приймає Context і клас сторінки додатку, яку ми хочемо запустити. Запускати ми будемо сторінку з реєстрацією користувача, тому що користувач вперше запустив програму, та файл для зберігання даних про нього ще не створився. Файл створиться тільки після успішної реєстрації в програмі, і більше при запуску програми ніколи не викинеться виняток.

“finish()” – це метод використовується для завершення роботи працюючої сторінки програми;

Все вищезгадане буде виконуватися в коді, якщо метод "openFileInput(String name)" в блоці try викине виняток. В іншому випадку, якщо користувач вже зареєструвався, і файл з його даними був створений на емуляторі, ми викличемо метод “startActivity(Intent intent)”, але запустимо іншу сторінку програми, а саме домашню сторінку.

Крім запуску домашньої сторінки програми, я також додав дві анімації переходу між нашими сторінками за допомогою методу

"overridePendingTransition(int enterAnim, int exitAnim), де enterAnim - анімація появи сторінки, exitAnim - анімація закриття сторінки.

Реалізацію анімації появи сторінки можна подивитися на рисунку 3.8.

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:duration="400"
        android:fromXDelta="100%p"
        android:toXDelta="0" />
</set>
```

Рисунок 3.8 – Реалізація анімації появи сторінки

Детально всі теги, атрибути та їх властивості ми розглянули вище у проекті.

Реалізацію анімації закриття сторінки можна подивитися на рисунку 3.9

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:duration="400"
        android:fromXDelta="0"
        android:toXDelta="-100%p" />
</set>
```

Рисунок 3.9 – Реалізація анімації закриття сторінки

Детально всі теги, атрибути та їх властивості ми розглянули вище у проекті.

На цьому реалізація заставки додатку завершується.

3.2 Розробка сторінки реєстрації

Як ми вже писали в минулому пункті, якщо користувач вперше запускає програму, то після відображення заставки мобільного додатка його одразу кидає на сторінку реєстрації. Це єдиний варіант користувача потрапити на цю сторінку програми, і після успішної реєстрації дана сторінка більше буде недоступна.

Контент сторінки реєстрації складається з чотирнадцяти елементів: шість із них – це текстові поля, п'ять – це поля введення даних, один – це список, та останній елемент – це кнопка. Як реалізовувати текстові поля в Android Studio ми вже

розглядали вище. Відображення полів введення даних в середовищі розробки Android Studio відбувається за допомогою компонента EditText, список за допомогою компонента TextInputLayout. А відображення кнопки відбувається за допомогою компонента Button. Всі компоненти одного типу однакові в ініціалізації, крім ідентифікаторів та тексту, тому наведу приклади ініціалізації по одному з кожного типу елемента

Ініціалізація компонента TextView у XML розмітці зображена на рисунку 3.10.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="20dp"
    android:layout_marginTop="20dp"
    android:fontFamily="@font/dm_sans_medium"
    android:textColor="@color/prime_green"
    android:gravity="start"
    android:text="First Name:"
    android:textStyle="bold"
    android:textSize="17sp"/>
```

Рисунок 3.10 – Ініціалізація компонента TextView у XML розмітці

Де більшість тегів, атрибутів та їх властивостей ми розібрали у пункті 3.1, тому розберемо тільки нові елементи.

“android:layout_marginStart” – це атрибут Android Studio, який вказує початковий відступ компонента від точки старту його розташування, та його властивість – "20dp", яка позначає встановлений відступ, де “dp” – це Density-independent Pixels. Абстрактна одиниця вимірювання, що дозволяє програмам виглядати однаково на різних екранах та дозволах. Використовується для всіх компонентів View (TextView, EditText, Button та інші);

“android:layout_marginTop” – це атрибут Android Studio, який вказує відступ компонента від верхнього кордону, та його властивість – "20dp", яка позначає встановлений відступ;

“android:gravity” – це атрибут Android Studio, який вказує розташування контенту всередині компонента, та його властивість – "start", яка встановлює місце розташування контенту на початку компонента;

“android:textStyle” – це атрибут Android Studio, який вказує стиль для тексту компонента, та його властивість – "bold", яка робить текст жирним.

Ініціалізація компонента EditText у XML розмітці зображена на рисунку 3.11.

```
<EditText
    android:id="@+id/et_age"
    android:layout_width="match_parent"
    android:layout_height="40dp"
    android:layout_marginTop="10dp"
    android:textSize="17sp"
    android:layout_marginStart="18dp"
    android:layout_marginEnd="30dp"
    android:backgroundTint="@color/prime_green"
    android:textCursorDrawable="@drawable/custom_cursor"
    android:hint="Age"
    android:inputType="number"
    />
```

Рисунок 3.11 – Ініціалізація компонента EditText у XML розмітці

Де більшість тегів, атрибутів та їх властивостей ми розібрали у пункті 3.1, тому розберемо тільки нові елементи.

“android:layout_marginEnd” – це атрибут Android Studio, який вказує відступ компонента від кінцевого кордону, та його властивість – "30dp", яка позначає встановлений відступ;

“android:backgroundTint” – це атрибут Android Studio, який вказує відтінок фону для компонента, та його властивість – "@color/prime_green", яка позначає встановлений колір відтінку;

“android:textCursorDrawable” – це атрибут Android Studio, який встановлює кастомний курсор для компонента, та його властивість – "@drawable/custom_ cursor", яка за допомогою символу "@" дає зрозуміти нашому додатку, що наступне написане значення – це посилання на кастомний курсор, у нашому випадку кастомний курсор “custom_ cursor” лежить у папці “drawable”, в якій знаходяться всі статичні зображення додатка;

“android:hint” – це атрибут Android Studio, який встановлює текстову підказку для нашого компонента, та його властивість – "Age", яка позначає встановлений текст;

“android:inputType” – це атрибут Android Studio, який встановлює тип даних, що вводяться у компоненті, та його властивість – “number”, яка встановлює числовий тип даних.

Ініціалізація компонента TextInputLayout у XML розмітці зображено на рисунку 3.12.

```
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:layout_marginStart="19dp"
    android:layout_marginEnd="30dp"
    android:hint="Select Gender"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox.ExposedDropdownMenu"
    app:hintTextColor="@color/prime_green"
    app:boxStrokeColor="@color/prime_green">
    <AutoCompleteTextView
        android:id="@+id/auto_complete_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="none"
        android:textColor="@color/black"/>
</com.google.android.material.textfield.TextInputLayout>
```

Рисунок 3.12 – Ініціалізація компонента TextInputLayout у XML розмітці

Де більшість тегів, атрибутів та їх властивостей ми розібрали у пункті 3.1, тому розберемо тільки нові елементи.

“style” – це атрибут Android Studio, який встановлює стиль для компоненту, та його властивість, яка позначає встановлений певний програмний стиль;

“app:hintTextColor” – це атрибут Android Studio, який встановлює колір для тексту підказки компонента, та його властивість – “@color/prime_green”, яка позначає встановлений колір;

“app:boxStrokeColor” – це атрибут Android Studio, який встановлює колір обведення фрагмента, та його властивість – “@color/prime_green”, яка позначає встановлений колір;

“<AutoCompleteTextView>” – це тег, що відповідає за виведення на екран вибраний користувачем елемент у списку. Обов'язковий тег для TextInputLayout, без якого список не буде працювати коректно.

Ініціалізація компонента Button у XML розмітці зображена на рисунку 3.13.

```
<Button
    android:id="@+id/bt_write_data"
    android:layout_width="200dp"
    android:layout_marginTop="20dp"
    android:layout_height="55dp"
    android:backgroundTint="@color/black"
    android:fontFamily="@font/allerta"
    android:layout_gravity="center"
    android:text="Register"
    android:textColor="@color/white" />
```

Рисунок 3.13 – Ініціалізація компонента Button у XML розмітці

Детально всі теги, атрибути та їх властивості ми розглянули вище у пункті 3.1.

Крім ініціалізації в XML розмітці, так само потрібно ініціалізувати поля введення тексту, список та кнопку в JAVA коді, для подальшої роботи з ними в JAVA.

Ініціалізація компонентів у JAVA коді зображена на рисунку 3.14.

```
private AutoCompleteTextView auto_complete_text;
private Button bt_write_data;
private EditText et_first_name, et_second_name, et_age, et_height, et_weight;
```

Рисунок 3.14 – Ініціалізація компонентів у JAVA коді

Тепер надамо унікальні ідентифікатори кожному з компонентів у JAVA коді.

Присвоєння унікальних ідентифікаторів компонентам зображено на рисунку 3.15.

```
auto_complete_text = findViewById(R.id.auto_complete_text);
et_first_name = findViewById(R.id.et_first_name);
et_second_name = findViewById(R.id.et_second_name);
et_age = findViewById(R.id.et_age);
et_height = findViewById(R.id.et_height);
et_weight = findViewById(R.id.et_weight);
bt_write_data = findViewById(R.id.bt_write_data);
```

Рисунок 3.15 – Присвоєння унікальних ідентифікаторів компонентам

Ми ініціалізували список, але не додали жодного компонента до нього, для додавання нам спочатку потрібно створити шаблон компонента для списку. Шаблон буде називатися “dropdown_menu_item.xml”, і буде створено в папці layout, де зібрані всі макети програми.

Реалізація цього шаблону зображена на рисунку 3.16.

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/dropdown_menu_item"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Test"
    android:textSize="17sp"
    android:padding="5dp"
    android:maxLines="1"
    android:textColor="@color/black"
    />
```

Рисунок 3.16 – Реалізацію шаблону “dropdown_menu_ item.xml”

Більшість тегів, атрибутів та їх властивостей ми розібрали у пункті 3.1, тому розберемо тільки нові елементи.

“android:padding” – це атрибут Android Studio, який встановлює значення полів навколо вмісту елемента, та його властивість – “5dp”, яка позначає значення полів;

“android:maxLines” – це атрибут Android Studio, який встановлює максимальне значення кількості рядків тексту, та його властивість – “1”, яка позначає, що рядок буде один.

Після створення шаблону ми можемо приступити до заповнення списку компонентами, для початку нам потрібно отримати заготовлений текстовий список елементів і покласти ці текстові значення у масив рядків, для цього ми створимо масив рядків “genders”. За допомогою методу “getResources()” ми отримаємо доступ до всіх ресурсів нашого проекту в JAVA коді, а далі викличемо ще один метод, який у всіх ресурсах буде шукати та повертати тільки потрібний масив рядків із зазначеним нами “id” - це метод “getStringArray(int id)”.

Заповнення масиву рядків даними зображено на рисунку 3.17.

```
String[] genders = getResources().getStringArray(R.array.gender_array);
```

Рисунок 3.17 – Заповнення масиву рядків даними

Де “R.array.gender_array” – це “id” заготовленого масиву рядків для нашого списку.

Сам масив рядків для списку вибору статі можна побачити на рисунку 3.18.

```
<string-array name="gender_array">
    <item>Male</item>
    <item>Female</item>
</string-array>
```

Рисунок 3.18 – Масив рядків для списку вибору статі

Де “<string-array>” – це тег, який вказує на те, що дані, розташовані в даному тегу, є масивом рядків;

“<item>” – це тег, який вказує на те, що текст написаний в даному тегу є елементом масиву.

Після того як ми заповнили масив рядків даними, нам потрібно покласти цей масив до нашого списку. Допомагає нам у цьому адаптер “ArrayAdapter<>”. ArrayAdapter є найпростішим адаптером, спеціально призначеним для роботи з елементами списку. Якщо говорити загалом, то адаптери спрощують зв'язування даних з елементом управління, наприклад дані нашого масиву та компонента TextInputLayout. При створенні об'єкта ArrayAdapter конструктор даного об'єкта приймає три обов'язкові параметри - це Context, заготовлений шаблон компонента і безпосередньо масив даних.

Після створення ArrayAdapter, все, що нам залишається це привласнити нашому компоненту TextInputLayout цей адаптер, робиться це за допомогою методу “setAdapter(T adapter)”.

Ініціалізація та присвоєння ArrayAdapter нашому списку зображено на рисунку 3.19.

```
adapterItems = new ArrayAdapter<>( context: this, R.layout.dropdown_menu_item, genders);
auto_complete_text.setAdapter(adapterItems);
```

Рисунок 3.19 – Ініціалізація та присвоєння ArrayAdapter нашому списку

Де “R.layout.dropdown_menu_item” – це шаблон компонента для списку;

“genders”– це масив рядків.

Основну частину візуалізації ми закінчили, залишилося додати функціонал кнопці, натисканням якої будуть зчитуватися дані з компонентів. Ці дані будуть записуватися в створений клас “User” для зберігання, а потім будуть записані до файлу формату “.json” .Так само у функціоналі кнопки потрібно реалізувати валідацію всіх компонентів - якщо користувач залишає порожні поля, то поряд із компонентом з'являється червоний значок помилки з текстом помилки.

Валідація – це перевірка значень, вказаних користувачем, та відображення знайдених помилок.

Спочатку нам треба навчити кнопку реагувати на натискання. Для цього кнопка має метод “setOnClickListener(View.OnClickListener l)”. На вхід подається об'єкт з інтерфейсом “View.OnClickListener”. Саме цьому об'єкту кнопка доручить обробляти натискання.

Присвоєння “View.OnClickListener” кнопці зображено зображено на рисунку 3.20.

```
bt_write_data.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

    }});
```

Рисунок 3.20 – Присвоєння “View.OnClickListener” кнопці

Де у методі “onClick(View v)” буде реалізовано основний функціонал кнопки.

Для початку в методі “onClick()”, ми створюємо об'єкт класу User для зберігання значень, які ми будемо зчитувати з полів. У класі User реалізовані поля:

1) firstName – це поле, в якому зберігатимуться дані про ім’я користувача;

2) secondName – це поле, в якому зберігатимуться дані про прізвище користувача;

3) age – це поле, в якому зберігатимуться дані про вік користувача;

4) weight – це поле, в якому зберігатимуться дані про вагу користувача;

5) height – це поле, в якому зберігатимуться дані про зріст користувача;

6) gender – це поле, в якому зберігатимуться дані про стать користувача.

Також у класі реалізовані гетери та сетери полів. Методи, що повертають значення змінних, називаються гетерами. Методи, що змінюють значення властивостей, називаються сетерами.

Докладніше з класом User можна ознайомитись на рисунку 3.21.

```
public class User {
    private String firstName;
    private String secondName;
    private String age;
    private String weight;
    private String height;
    private String gender;


    public void setFirstName(String firstName) { this.firstName = firstName; }

    public void setSecondName(String secondName) { this.secondName = secondName; }

    public void setAge(String age) { this.age = age; }

    public void setWeight(String weight) { this.weight = weight; }

    public void setHeight(String height) { this.height = height; }

    public void setGender(String gender) { this.gender = gender; }

    public String getFirstName() { return firstName; }

    public String getSecondName() { return secondName; }

    public String getAge() { return age; }

    public String getWeight() { return weight; }

    public String getHeight() {
        return height;
    }

    public String getGender() {
        return gender;
    }
```

Рисунок 3.21 – Опис класу User

Після створення об'єкта User, ми починаємо заповнювати його поля шляхом перевірки певної умови - якщо поле не порожнє, то ми зчитуємо введений текст з поля і привласнюємо його змінної класу User, якщо поле порожнє, то ми встановлюємо на наш компонент з введенням тексту помилку і просимо ввести текст. Реалізуємо цю перевірку за допомогою оператора if. Для кожного поля введення тексту ми використовуємо однотипну перевірку, тому я опишу її один раз на прикладі перевірки поля введення "Ім'я".

Перевірка умови на порожнє поле "Ім'я" введення зображена на рисунку 3.22.

```
if (et_first_name.getText().toString().isEmpty()) {
    et_first_name.setError(getString(R.string.error_first_name));
} else
    user.setFirstName(et_first_name.getText().toString());
```

Рисунок 3.22 – Перевірка умови на порожнє поле "Ім'я"

Де у тілі оператора “if” ми у текстового поля викликаємо метод “getText()”, який повертає нам весь текст написаний у полі введення, далі ми цей текст наводимо до рядка за допомогою методу “toString()” і перевіряємо його на відсутність тексту за допомогою методу “isEmpty()”. Якщо метод “isEmpty()” поверне істину, тоді на даному компоненті ми встановлюємо помилку за допомогою методу “setError(String error)”, де ми як аргумент передали рядок з папки “string”. Якщо ж “isEmpty()” поверне нам брехню, це означає, що поле компонента не порожнє, і ми встановлюємо отриманий текст у поле об'єкта User за допомогою сеттера “setFirstName(String firstName)”. Таким чином реалізовано перевірку на кожному з полів нашої сторінки реєстрації.

Після перевірки кожного поля окремо, ми перевіряємо умову наявності хоча б одного порожнього поля на сторінці реєстрації. Також перевіримо цю умову за допомогою оператора if.

Перевірка наявності хоча б одного порожнього поля на сторінці реєстрації зображена на рисунку 3.23.

```
if (et_first_name.getText().toString().isEmpty() ||
        et_second_name.getText().toString().isEmpty() ||
        et_age.getText().toString().isEmpty() ||
        et_weight.getText().toString().isEmpty() ||
        et_height.getText().toString().isEmpty() ||
        auto_complete_text.getText().toString().isEmpty()) {
    return;
}
```

Рисунок 3.23 – Перевірка наявності хоча б одного порожнього поля на сторінці реєстрації

Де якщо хоч в одному полі метод “isEmpty()” поверне істину, то кнопка припинить подальше виконання коду після блоку коду, припинить за допомогою оператора return.

Таким чином ми реалізували валідацію полів введення тексту.

Повністю реалізацію валідації полів можна побачити на рисунку 3.24.

```
@Override
public void onClick(View v) {
    User user = new User();
    if (et_first_name.getText().toString().isEmpty()) {
        et_first_name.setError(getString(R.string.error_first_name));
    } else
        user.setFirstName(et_first_name.getText().toString());
    if (et_second_name.getText().toString().isEmpty()) {
        et_second_name.setError("Please, enter Second Name");
    } else
        user.setSecondName(et_second_name.getText().toString());
    if (et_age.getText().toString().isEmpty()) {
        et_age.setError("Please, enter Age");
    } else
        user.setAge(et_age.getText().toString());
    if (et_weight.getText().toString().isEmpty()) {
        et_weight.setError("Please, enter Weight");
    } else
        user.setWeight(et_weight.getText().toString());
    if (auto_complete_text.getText().toString().isEmpty()) {
        auto_complete_text.setError("Please, choose Gender");
    } else
        user.setGender(auto_complete_text.getText().toString());
    if (et_height.getText().toString().isEmpty()) {
        et_height.setError("Please, enter Height");
    } else
        user.setHeight(et_height.getText().toString());

    if (et_first_name.getText().toString().isEmpty() ||
            et_second_name.getText().toString().isEmpty() ||
            et_age.getText().toString().isEmpty() ||
            et_weight.getText().toString().isEmpty() ||
            et_height.getText().toString().isEmpty() ||
            auto_complete_text.getText().toString().isEmpty()) {
        return;
    }
```

Рисунок 3.24 – Повна реалізація валідації полів

Фінальна частина функціоналу обробника натискань полягає у записі даних користувача у файл формату “.json”. JSON (англ. JavaScript Object Notation) - текстовий формат обміну даними, що базується на JavaScript. Але при цьому формат незалежний від JS може використовуватися в будь-якій мові програмування.

Для реалізації цього завдання ми скористаємося об'єктом класу JSONObject, який є структурою типу HashMap (ключ – значення). У об'єкт класу JSONObject ми будемо класти значення з полів нашого об'єкта User за допомогою методу put(String name, String value), де name - це ключ, наприклад firstName, value - це значення ключа.

Потім весь цей об'єкт приведемо до рядка за допомогою методу toString(). В результаті ми отримаємо рядок, в якому будуть описані всі ключі та їх значення через двокрапку.

І наприкінці ми цей рядок запишем у файл за допомогою об'єкта класу OutputStream та його методів.

Клас OutputStream – це абстрактний клас, що визначає байтовий потік виведення даних.

Реалізація запису даних у файл зображена на рисунку 3.25.

```
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put( name: "firstName", user.getFirstName());
    jsonObject.put( name: "secondName", user.getSecondName());
    jsonObject.put( name: "age", user.getAge());
    jsonObject.put( name: "gender", user.getGender());
    jsonObject.put( name: "height", user.getHeight());
    jsonObject.put( name: "weight", user.getWeight());
    String filename = "userData.json";
    String jsonSTRING = jsonObject.toString();
    OutputStream fos = openFileOutput(filename, MODE_PRIVATE);
    fos.write(jsonSTRING.getBytes(StandardCharsets.UTF_8));
    fos.close();
} catch (JSONException | IOException e) {
    e.printStackTrace();
}
```

Рисунок 3.25 – Запис даних у файл

Де “openFileOutput(String name, int mode)” – це метод, який відкриває для запису файл, що у пам'яті емулятора. Створює файл у пам'яті емулятора, якщо він ще не існує; “String name” – атрибут методу, який позначає найменування файлу “int mode” – атрибут методу, який позначає режим роботи з файлом;

“MODE_PRIVATE” – режим створення файлу;

“write(byte[] b)” – це метод, який записує масив байтів у файл;

Щоб записати рядок у файл, для початку рядок потрібно перетворити на набір байтів, що ми робимо завдяки методу getBytes(), та встановлюємо в метод стандартне кодування UTF-8 за допомогою рядка коду StandardCharsets.UTF_8.

“close()” – це метод, який закриває потік даних;

Приклад записаних даних у файл можна розглянути на рисунку 3.26.

{"firstName":"Nikolai","secondName":"Baskov","age":"45","gender":"Male","height":"180","weight":"69.5"}

Рисунок 3.26 – Приклад записаних даних у файл

Якщо дані у файл записалися коректно, і ми не зловили жодних винятків, то обробник подій виконує останню ділянку коду - запуск нової сторінки програми.

Реалізація запуску нової сторінки додатку зображена на рисунку 3.27.

В результаті візуалізації сторінки реєстрації ми отримали результат зображений на рисунку 3.28.

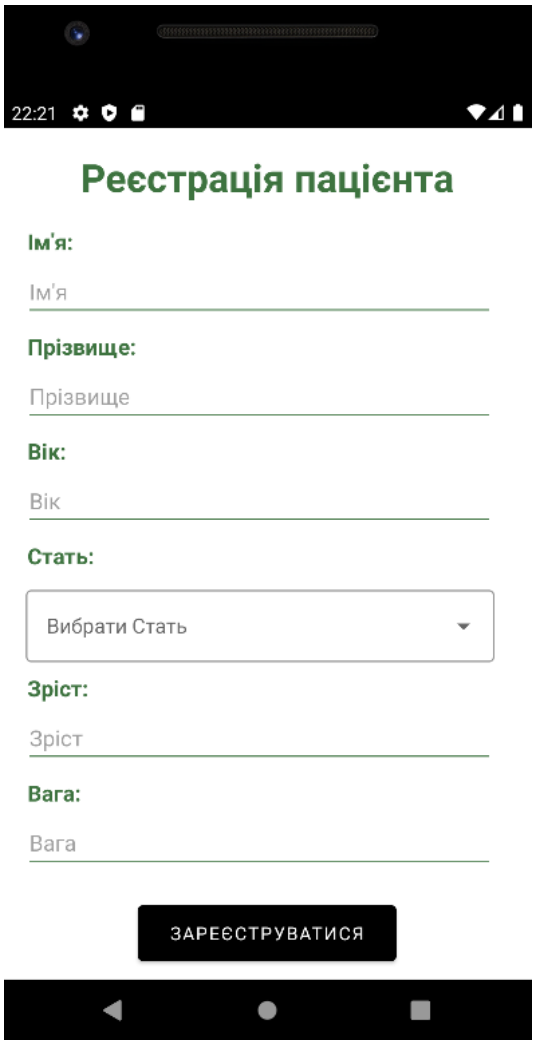


Рисунок 3.28 – Візуалізація сторінки реєстрації

Якщо користувач не введе будь-які дані і натисне на кнопку "Register", то йому не вдасться зареєструватися і програма попросить його ввести відсутні дані.

Візуалізація сторінки реєстрації у разі не заповнення користувачем необхідних полів зображена на рисунку 3.29.

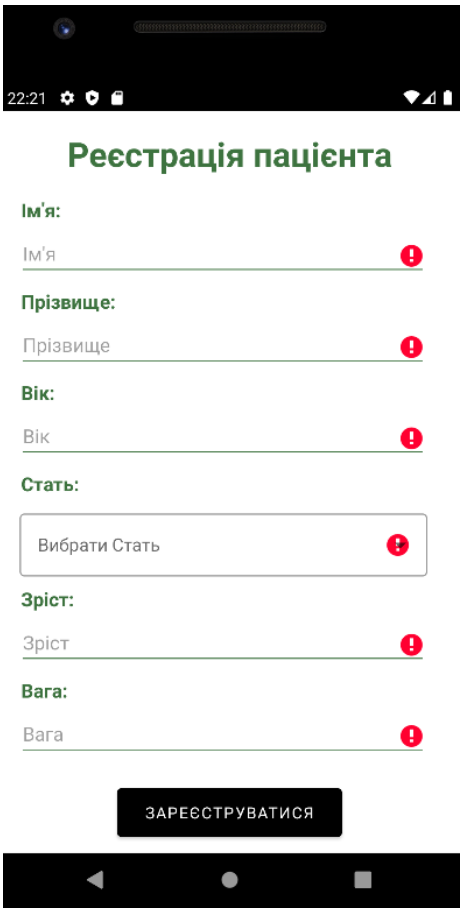


Рисунок 3.29 – Візуалізація сторінки реєстрації у разі не заповнення користувачем необхідних полів

3.3 Розробка меню навігації

Меню навігації одна з найважливіших складових для зручності мобільного додатка. За допомогою меню навігації у користувача є можливість вільно переходити між сторінками програм. Меню навігації буде реалізовано на всіх наступних сторінках нашої програми, які ми опишемо пізніше. Сама реалізація меню навігації буде однакова на всіх сторінках програми, тому ми опишемо її реалізацію всього один раз.

За створення та відображення меню навігації в мобільних додатках відповідає компонент Android Studio – NavigationView.

Для зручної роботи з меню навігації потрібно також додати панель інструментів. Створення та відображення панелі інструментів здійснюється за допомогою компонента Toolbar.

Приклад панелі інструментів у мобільному додатку можна побачити на рисунку 3.30.

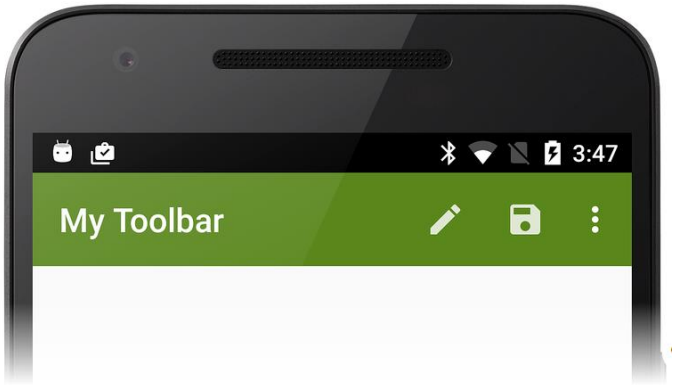


Рисунок 3.30 – Приклад панелі інструментів у мобільному додатку

Панель інструментів Toolbar потрібна нам для того, щоб розташувати на ній кнопку показу меню навігації. Для початку реалізуємо панель інструментів.

Реалізація панелі інструментів Toolbar у XML розмітці зображена на рисунку 3.31.

```
<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="@color/prime_green"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

Рисунок 3.31 – Реалізація панелі інструментів Toolbar у XML розмітці

Більшість тегів, атрибутів та їх властивостей ми розібрали у пунктах вище, тому розберемо тільки нові елементи.

“android:theme” – це атрибут Android Studio, який відповідає за встановлення компоненту певного вбудованого стилю Android Studio, та його властивість, яка позначає яка конкретна тема буде встановлена;

“android:popupTheme” – це атрибут Android Studio, який вказує тему, що використовується при додаванні елементів на панель інструментів, та його властивість, яка позначає яка конкретна тема буде використана;

Реалізація меню навігації NavigationView у XML розмітці зображена на рисунку 3.32.

```
<com.google.android.material.navigation.NavigationView
    android:id="@+id/nav_view"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    app:headerLayout="@layout/nav_header"
    app:menu="@menu/drawer_menu" />
```

Рисунок 3.32 – Реалізація меню навігації NavigationView у XML розмітці

Більшість тегів, атрибутів та їх властивостей ми розібрали у пунктах вище, тому розберемо тільки нові елементи.

Меню навігації складається з двох елементів - це заголовок та саме меню вибору сторінок. Додавання даних елементів у компонент NavigationView відбувається за допомогою двох атрибутів – “app:headerLayout” та “app:menu”.

“app:headerLayout” – це атрибут Android Studio, який додає у наше меню навігації заголовок, та його властивість “@layout/nav_header”, яка вказує який конкретно шаблон ми будемо використовувати як заголовок;

“app:menu” – це атрибут Android Studio, який додає у наше меню навігації меню вибору сторінок, та його властивість “@menu/nav_header”, яка вказує яке саме меню будемо використовувати як меню навігації. Заголовок та меню вибору сторінок ми реалізували так само у XML розмітці.

Реалізація заголовка меню навігації NavigationView у XML розмітці зображена на рисунку 3.33.

```
<ImageView
    android:layout_width="70dp"
    android:layout_height="70dp"
    android:src="@drawable/pngtree2" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:fontFamily="@font/allerta"
    android:text="For business contact"
    android:textColor="@color/white"
    android:textSize="14sp"
    android:textStyle="bold"/>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dp"
    android:fontFamily="@font/allerta"
    android:text="holterapplication@gmail.com"
    android:textColor="@color/white"
    android:textSize="14sp"
    android:textStyle="bold"/>
```

Рисунок 3.33 – Реалізація заголовка меню навігації NavigationView у XML розмітці

Детально всі теги, атрибути та їх властивості ми розглянули вище у проекті. Заголовок у нас складається із зображення та двох текстових полів. Візуальне відображення заголовка можна розглянути на рисунку 3.34.

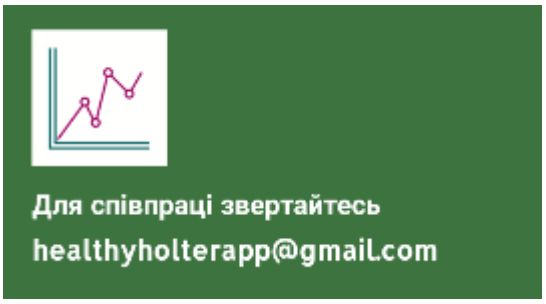


Рисунок 3.34 – Візуальне відображення заголовка

Реалізація меню вибору сторінок для меню навігації NavigationView у XML розмітці зображена на рисунку 3.35.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:showIn="navigation_view">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_home"
            android:icon="@drawable/ic_baseline_home"
            android:title="Home" />
        <item
            android:id="@+id/nav_graph"
            android:icon="@drawable/ic_baseline_graphic"
            android:title="Graph of a noisy ECG signal" />
        <item
            android:id="@+id/nav_graph2"
            android:icon="@drawable/ic_baseline_graphic"
            android:title="Graph of constant ECG signal" />
        <item
            android:id="@+id/nav_graph3"
            android:icon="@drawable/ic_baseline_graphic"
            android:title="RSQ Waveform Search Graph" />
        <item
            android:id="@+id/nav_graph4"
            android:icon="@drawable/ic_baseline_graphic"
            android:title="Graph of ECG data analysis" />
    </group>
    <item android:title="Demo department">
        <menu>
            <group android:checkableBehavior="single">
                <item
                    android:id="@+id/nav_demo"
                    android:icon="@drawable/ic_baseline_bug_report"
                    android:title="@string/demo_testing" />
            </group>
        </menu>
    </item>
</menu>
```

Рисунок 3.35 – Реалізація меню вибору сторінок для меню навігації NavigationView у XML розмітці

Більшість тегів, атрибутів та їх властивостей ми розібрали у пунктах вище, тому розберемо тільки нові елементи.

“<menu> – це тег, який вказує на те, що даний шаблон є шаблоном меню;

“tools:showIn” – це атрибут Android Studio, який вказує в якому компоненті буде відображатися дане меню, та його властивість “navigation_view”, яка вказує, що відображатися це меню буде в компоненті NavigationView;

“<group>” – це тег, який вказує на те, що елементи всередині цього тегу згруповані;

“android:checkableBehavior” – це атрибут Android Studio, який регулює поведінку обраних користувачем елементів, та його властивість “single”, яка говорить нам, що обраний буде тільки той елемент, який вибрав користувач, а не вся група;

“android:icon” – це атрибут Android Studio, який додає до початку елемента іконку, та його властивість “ @drawable/ic_baseline_home”, яка вказує яку саме іконку ми хочемо додати з папки drawable;

“android:title” – це атрибут Android Studio, встановлює елементу заголовок, та його властивість “ Home”, яка вказує конкретний текст заголовка.

Візуальне відображення меню вибору сторінок можна розглянути на рисунку 3.36.

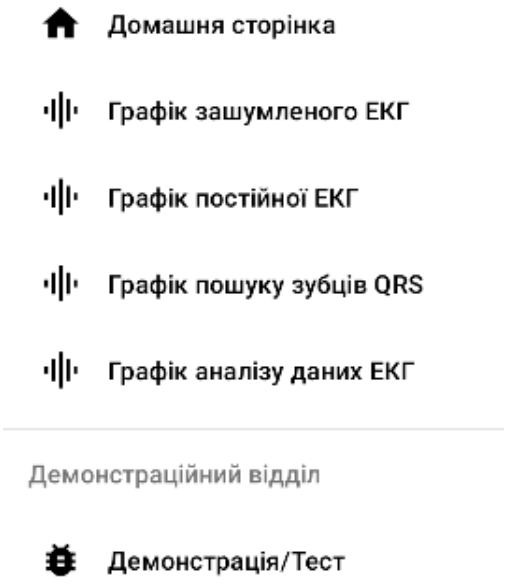


Рисунок 3.36 – Візуальне відображення меню вибору сторінок

З ініціалізацією в XML розмітці ми закінчили, залишилося проініціалізувати меню навігації в коді JAVA.

Ініціалізація панелі інструментів та меню навігації у JAVA коді зображена на рисунку 3.37.

```
NavigationView navigationView = findViewById(R.id.nav_view);
Toolbar toolbar = findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

Рисунок 3.37 – Ініціалізація панелі інструментів та меню навігації у JAVA коді

Де “setSupportActionBar(Toolbar toolbar)” – це метод, який робить наш кастомний “toolbar” головною панеллю інструментів.

Після ініціалізації нам потрібно додати для меню навігації обробник вибору елемента меню, який при виборі елемента меню буде відкривати певну сторінку програми. Для цього меню навігації має метод “setNavigationItemSelectedListener(OnNavigationItemSelectedListener listener)”. На вхід подається об'єкт з інтерфейсом. І саме в методі цього об'єкта – “onNavigationItemSelected(MenuItem item)”, де item - це вибраний користувачем елемент, і буде реалізовано перехід між сторінками залежно від вибраного елемента.

Додавання обробника вибору елемента меню і реалізація його методу “onNavigationItemSelected(MenuItem item)” зображено на рисунку 3.38.

```
navigationView.setNavigationItemSelectedListener(new NavigationView.OnNavigationItemSelectedListener() {
    @Override
    public boolean onNavigationItemSelected(@NonNull MenuItem item) {
        switch (item.getItemId()) {
            case R.id.nav_graph:
                startActivity(new Intent(getApplicationContext(), NoisyECGActivity.class));
                overridePendingTransition(R.anim.slide_in_right, R.anim.slide_out_left);
                finish();
                break;
            case R.id.nav_home:
                break;
            case R.id.nav_graph2:
                startActivity(new Intent(getApplicationContext(), ConstantECGActivity.class));
                overridePendingTransition(R.anim.slide_in_right,R.anim.slide_out_left);
                finish();
                break;
            case R.id.nav_graph3:
                startActivity(new Intent(getApplicationContext(), WaveformRSQActivity.class));
                overridePendingTransition(R.anim.slide_in_right,R.anim.slide_out_left);
                finish();
                break;
            case R.id.nav_graph4:
                startActivity(new Intent(getApplicationContext(), DataAnalysisECGActivity.class));
                overridePendingTransition(R.anim.slide_in_right, R.anim.slide_out_left);
                finish();
                break;
            case R.id.nav_demo:
                startActivity(new Intent(getApplicationContext(), DemoGraphActivity.class));
                overridePendingTransition(R.anim.slide_in_right, R.anim.slide_out_left);
                finish();
                break;
        }
        return true;
    }
```

Рисунок 3.38 – Додавання обробника вибору елемента меню і реалізація його методу “onNavigationItemSelected(MenuItem item)”

Більшість методів і конструкцій у цій реалізації ми описували раніше у проекті, тому зупинимося лише на нових фрагментах коду.

Весь функціонал переходу на інші сторінки програми реалізований в умовному операторі “switch()”, який дозволяє порівнювати змінну зі списком значень. Замість змінної ми передаємо “item.getItemId()” - це id вибраного елемента меню, а порівнювати цей id ми будемо зі списком усіх ідентифікаторів елементів меню , наприклад, один з таких ідентифікаторів - це R.id.nav_home. Якщо id, який ми передаємо в switch, буде еквівалентний хоча б одному id зі списку всіх елементів, то буде виконано перехід на іншу сторінку додатку за допомогою методів после ключевого слова “case”, які ми розглянули в попередніх пунктах проекту.

Таким чином ми завершили алгоритмізацію та візуалізацію меню навігації.

Візуальне відображення меню навігації можна розглянути на рисунку 3.39.

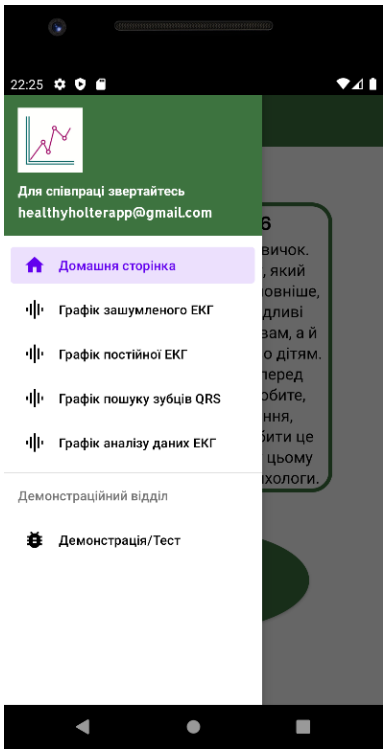


Рисунок 3.39 – Візуальне відображення меню навігації

3.4 Розробка домашньої сторінки

Якщо користувач успішно зареєструвався, то тепер після запуску додатку “Холтер”, він завжди буде потрапляти на домашню сторінку додатку.

Основний контент домашньої сторінки дуже простий і складається з двох текстових полів та однієї кнопки. У попередніх пунктах проекту ми розглянули які саме компоненти в Android Studio відповідають за відображення цих елементів. Кожен із компонентів буде покладено у так званий контейнер LinearLayout, який ми розглянемо детально. Контейнер LinearLayout представляє найпростіший контейнер - об'єкт ViewGroup, який упорядковує всі дочірні елементи в одному напрямку: по горизонталі або по вертикалі. Всі елементи розташовані один за одним. Напрямок розмітки вказується за допомогою атрибута “android:orientation”.

Додаємо ми компоненти в контейнер для того, щоб додати анімацію, таким чином, додати анімацію до цілого контейнера куди раціональніше ніж додавати її до кожного компонента окремо.

Текстові поля нам потрібні для виведення корисних рекомендацій щодо здорового способу життя. Перше текстове поле - це поле, яке буде відображати номер рекомендації. Друге поле - це поле, яке буде відображати сам текст рекомендації.

Кнопка нам потрібна для запуску нової сторінки додатку.

Кнопку та текстові поля ми розташуємо у різних контейнерах LinearLayout у зв'язку з тим, що для текстових полів ми реалізували спеціальну рамку з обведенням.

Ініціалізація текстових полів у XML розмітці зображена на рисунку 3.40.

```
<LinearLayout
    android:id="@+id/fragment_container"
    android:layout_width="300dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="60dp"
    android:background="@drawable/round"
    android:orientation="vertical">

    <TextView
        android:id="@+id/tv_rec_num"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="10dp"
        android:fontFamily="@font/dm_sans_medium"
        android:text=""
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/tv_rec"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginStart="10dp"
        android:layout_marginTop="5dp"
        android:layout_marginEnd="10dp"
        android:layout_marginBottom="5dp"
        android:text=""
        android:textAlignment="center"
        android:textColor="@color/black"
        android:textSize="17sp"
        android:fontFamily="@font/dm_sans_medium"/>
</LinearLayout>
```

Рисунок 3.40 – Ініціалізація текстових полів у XML розмітці

Більшість тегів, атрибутів та їх властивостей ми розібрали у пунктах вище, тому розберемо тільки нові елементи.

“android:orientation” – це атрибут Android Studio, який вказує напрямок розмітки, та його властивість – “vertical”, яка вказує на те, що кожен елемент у контейнері буде відображатися вертикально один за одним;

“android:background” – це атрибут Android Studio, який дозволяє встановлювати фонове зображення у контейнер LinearLayout, та його властивість – “@drawable/round”, яка вказує, яке конкретне зображення буде фоновим зображенням для LinearLayout. Саме за допомогою цього атрибуту ми й встановлюємо так звану рамку для всього контейнера LinearLayout, до якого входять наші текстові поля.

Реалізація самої рамки зображена на рисунку 3.41.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="@android:color/transparent"/>
    <corners android:radius="20dp"/>
    <stroke android:width="4dp"
        android:color="@color/prime_green"/>
</shape>
```

Рисунок 3.41 – Реалізація рамки для LinearLayout

Де “<shape>” – це тег, який вказує форму нашому фоновому зображенню;

“android:shape” – це атрибут Android Studio, який вказує форму нашому фоновому зображенню, та його властивість – “rectangle”, яка вказує, що форма нашого зображення буде прямокутною;

“<solid>” – це тег, який вказує колір заливки нашого зображення;

“@android:color/transparent” – це властивість, яка вказує, що колір заливки буде такого ж кольору, як батьківський елемент. У нашому випадку білого кольору;

“<corners>” – це тег, який відповідає за редагування кутів зображення;

“android:radius” – це атрибут Android Studio, який створює закруглені кути для зображення типу “rectangle”, та його властивість – “20dp”, яка вказує конткретний радіус закруглення кутів;

“<stroke>” – це тег, який додає нашому зображенню лінію обведення;

“anroid:width” – це атрибут Android Studio, який вказує товщину лінії обведення, та його властивість – “4dp” , яка вказує конкретну товщину лінії.

З ініціалізацією текстових полів ми закінчили.

Ініціалізація кнопки у XML розмітці зображена на рисунку 3.42.

```
<LinearLayout
    android:id="@+id/fragment_container_2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal">

    <androidx.appcompat.widget.AppCompatButton
        android:id="@+id/bt_start_work"
        android:layout_width="250dp"
        android:layout_height="110dp"
        android:gravity="center"
        android:layout_marginTop="40dp"
        android:background="@drawable/button_oval"
        android:text="START MEASURING"
        android:textSize="18sp"
        android:textColor="@color/white"
        android:fontFamily="@font/dm_sans_medium"/>
</LinearLayout>
```

Рисунок 3.42 – Ініціалізація кнопки у XML розмітці

Детально всі теги, атрибути та їх властивості ми розглянули вище у пунктах вище.

Як ви вже встигли зауважити, спочатку при ініціалізації текстових полів у XML розмітці ми не вказали на них сам текст. Оскільки у нас всього десять текстових рекомендацій, а компонент лише один, ми через JAVA код будемо привласнювати компоненту один текст із десяти заготовлених. Робити ми це за допомогою методу “chooseRecommendation()”, який ми детально опишемо. Щоб мати можливість встановлювати текст в компонент через код JAVA, для початку нам потрібно проініціалізувати компоненти в коді JAVA. Також потрібно проініціалізувати наші контейнери LinearLayout, щоб додати для них анімацію.

Ініціалізація компонентів у JAVA коді зображена на рисунку 3.43.

```
private TextView tv_rec_num;
private TextView tv_rec;
private LinearLayout linearLayoutMain;
private LinearLayout linearLayoutSecond;
private Button bt_start_work;
```

Рисунок 3.43 – Ініціалізація компонентів у JAVA коді

Присвоєння унікальних ідентифікаторів компонентам зображено на рисунку 3.44.

```
tv_rec_num=findViewById(R.id.tv_rec_num);
tv_rec=findViewById(R.id.tv_rec);
linearLayoutMain=findViewById(R.id.fragment_container);
linearLayoutSecond=findViewById(R.id.fragment_container_2);
bt_start_work=findViewById(R.id.bt_start_work);
```

Рисунок 3.44 – Присвоєння унікальних ідентифікаторів компонентам

Додавання анімації до контейнерів LinearLayout зображено на рисунку 3.45.

```
linearLayoutMain.setAnimation(AnimationUtils.loadAnimation( context: this,R.anim.bottom_animation));
linearLayoutSecond.setAnimation(AnimationUtils.loadAnimation( context: this,R.anim.bottom_animation));
```

Рисунок 3.45 – Додавання анімації до контейнерів LinearLayout

Функціонал додавання анімації до компонентів ми докладно розглянули у пункті 3.1.

Після ініціалізації та додавання анімації, ми викликаємо метод “chooseRecommendation()”, який встановлюватиме текст у наші текстові поля.Оскільки для кожної рекомендації код методу ідентичний, я опишу його один раз.

Реалізація методу “chooseRecommendation()” для першої рекомендації зображена на рисунку 3.46.

```
private void chooseRecommendation(){
    switch((int) (Math.random() * 10)){
        case 0:
            tv_rec_num.setText(R.string.recommendation_1);
            tv_rec.setText(R.string.rec_text_1);
            break;
```

Рисунок 3.46 – Реалізація методу “chooseRecommendation()” для першої рекомендації

У цьому методі за допомогою умовного оператора “switch()”, який ми вже описували вище, порівнюється змінна “(int) (Math.random() * 10)” зі списком зазначених змінних, наприклад “0”. Де “(int)” - це приведення змінної до типу “int”,

“Math.random()” - це метод, який використовується для генерації випадкового числа в діапазоні від 0.0 до 1.0, та “ * 10” - це множення на десять, за допомогою якого ми будемо отримувати значення змінної від 0 до 9.

Якщо наша змінна дорівнюватиме змінній “0”, то у текстові поля буде встановлено текст “Рекомендації №1”, і так далі.

На цьому реалізація домашньої сторінки закінчено.

Візуальне відображення домашньої сторінки можна розглянути на рисунку 3.47.

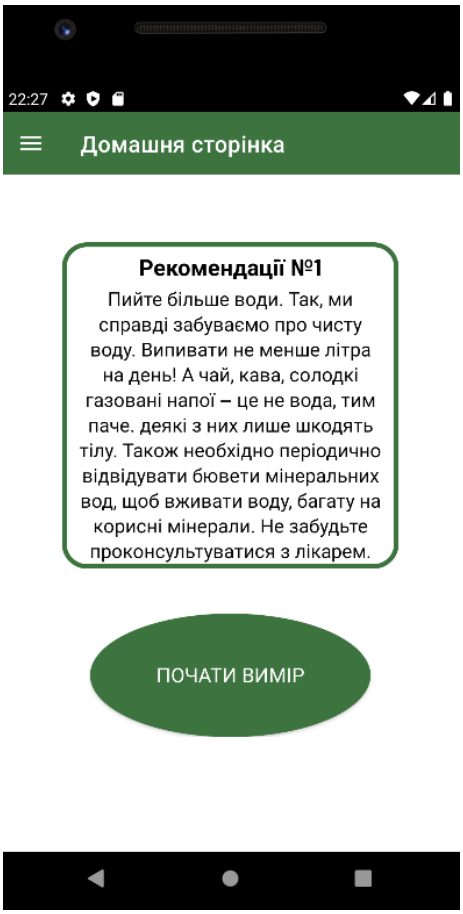


Рисунок 3.47 – Візуальне відображення домашньої сторінки

3.5 Розробка сторінки “Графік зашумленого ЕКГ сигналу”

Сторінка "Графік зашумленого ЕКГ сигналу" є однією з основних сторінок нашого мобільного додатку. Користувач може потрапити на цю сторінку через меню навігації, або ж після натискання кнопки на домашній сторінці.

На цій сторінці сторінці користувач має можливість спостерігати за зчитуванням ЕКГ сигналу та побудовою його функції на графіку в режимі реального часу.

Контент нашої сторінки складається з двох елементів: кнопка та графік. Як ми вже раніше з'ясували, за відображення кнопки у нас відповідає компонент Button. За відображення графіка відповідає компонент GraphView. Реалізація графіка GraphView у XML розмітці не дозволяє нам досконально налаштувати графік, тому основну частину налаштування ми будемо реалізовувати в JAVA коді. На нашому графіку ми будемо відображати вхідний сигнал ЕКГ, який буде анімований, шляхом промальовування кожної точки при її додаванні на графік у режимі реального часу. А кнопка нам потрібна для реалізації наближення та віддалення графіка.

Ініціалізація компонентів у XML розмітці зображена на рисунку 3.48.

```
<com.jjoe64.graphview.GraphView
    android:id="@+id/graph"
    android:layout_width="match_parent"
    android:layout_height="480dp" />

<Button
    android:id="@+id/bt_graph_zoom"
    style="?android:attr/buttonBarButtonStyle"
    android:layout_width="wrap_content"
    android:layout_height="50dp"
    android:backgroundTint="@color/black"
    android:fontFamily="@font/allerta"
    android:text="@string/zoom"
    android:textColor="@color/white" />
```

Рисунок 3.48 – Ініціалізація компонентів у XML розмітці

Детально всі теги, атрибути та їх властивості ми розглянули вище у пунктах вище.

Ініціалізація компонентів у JAVA коді зображена на рисунку 3.49.

```
private Button bt_graph_zoom;
private GraphView graph;
```

Рисунок 3.49 – Ініціалізація компонентів у JAVA коді

Присвоєння унікальних ідентифікаторів компонентам зображено на рисунку 3.50.

```
graph = findViewById(R.id.graph);
bt_graph_zoom = findViewById(R.id.bt_graph_zoom);
```

Рисунок 3.50 – Присвоєння унікальних ідентифікаторів компонентам

Тепер досконально опишемо налаштування самого графіка.

Повне налаштування графіка зображено на рисунку 3.51.

```
graph.setTitle(getString(R.string.original_ecg_signal));
graph.setTitleTextSize(55);
graph.getViewport().setYAxisBoundsManual(true);
graph.getViewport().setXAxisBoundsManual(true);
graph.getViewport().setMinY(-1);
graph.getViewport().setMaxY(1.65);
graph.getViewport().setMinX(0);
graph.getViewport().setMaxX(2000);
graph.getLegendRenderer().setVisible(true);
graph.getLegendRenderer().setAlign(LegendRenderer.LegendAlign.TOP);
graph.getGridLabelRenderer().setLabelVerticalWidth(20);
graph.getGridLabelRenderer().setHorizontalAxisTitle("Countdowns");
graph.getGridLabelRenderer().setVerticalAxisTitle("Voltage (mV)");
graph.getGridLabelRenderer().setHorizontalAxisTitleTextSize(55);
graph.getGridLabelRenderer().setVerticalAxisTitleTextSize(55);
graph.getGridLabelRenderer().setPadding(55);
```

Рисунок 3.51 – Повне налаштування графіка

Де “setTitle(String mTitle)” – це метод, який встановлює графіку заголовок, та його аргумент – “R.string.original_ecg_signal”, який вказує який конкретний заголовок буде встановлений графіку;

“setTitleTextSize(float titleTextSize)” – це метод, який встановлює розмір тексту заголовка, та його аргумент “55”, який вказує конкретний розмір тексту заголовка;

“getViewport()” – це метод, за допомогою якого ми отримуємо доступ до осей графіка;

“setYAxisBoundsManual(boolean mYAxisBoundsManual)” – це метод, за допомогою якого ми отримуємо дозвіл на ручне редагування вісі Y, та його аргумент “true”, який позначає істину і дозволяє нам редагувати вісь Y;

“setXAxisBoundsManual(boolean mXAxisBoundsManual)” – це метод, з таким же функціоналом, як і метод вище, але для вісі X;

“setMinY(double y)” – це метод, який встановлює мінімальне значення вісі Y;

“setMaxY(double y)” – це метод, який встановлює максимальне значення вісі Y;

“setMinX(double x)” – це метод, який встановлює мінімальне значення вісі X.

“setMaxX(double x)” – це метод, який встановлює максимальне значення вісі X;

“getLegendRenderer()” – це метод, за допомогою якого ми отримуємо доступ до заголовка функції графіка;

“setVisible(boolean mIsVisible)” – це метод, який встановлює відображення заголовка функції. Якщо передати в нього істину, то заголовок буде відображатися, якщо брехня, то не буде;

“setAlign(LegendAlign align)” – це метод, який позиціонує відображення заголовка функції, та його аргумент – “LegendRenderer.LegendAlign.TOP”, який встановлює відображення заголовка функції зверху графіка;

“getGridLabelRenderer()” – це метод, за допомогою якого ми отримуємо доступ до заголовків осей;

“setLabelVerticalWidth(Integer width)” – це метод, який вказує відступ між заголовком вісі Y та самою віссю;

“setHorizontalAxisTitle(String mHorizontalAxisTitle)” – це метод, який вказує текст заголовка вісі X;

“setVerticalAxisTitle(String mVerticalAxisTitle)” – це метод, який вказує текст заголовка вісі Y;

“setHorizontalAxisTitleTextSize(float horizontalAxisTitleTextSize)” – це метод, який вказує розмір тексту заголовка вісі X;

“setVerticalAxisTitleTextSize(float verticalAxisTitleTextSize)” – це метод, який вказує розмір тексту заголовка вісі Y;

“setPadding(int padding)” – це метод, який встановлює відступ для заголовків з усіх боків.

З налаштуванням графіка ми закінчили, залишилося відобразити у ньому вихідний сигнал ЕКГ. Робити ми це за допомогою спеціального класу “GraphCreator”, який у свою чергу успадковується від абстрактного класу AsyncTask.

Клас AsyncTask пропонує простий та зручний механізм для переміщення трудомістких операцій у фоновий потік. Завдяки йому ви отримуєте зручність синхронізації обробників подій з графічним потоком, що дозволяє оновлювати елементи інтерфейсу користувача для звіту про хід виконання завдання або для

виведення результатів, коли завдання завершено. Безпосередньо з класом AsyncTask працювати не можна (абстрактний клас), вам потрібно успадковуватися від нього (extends), тому ми створили клас “GraphCreator”, який успадковується від AsyncTask. Основний функціонал даного класу буде реалізований у методі “doInBackground(String... strings)”, який на вхід буде отримувати ім'я файлу, де зберігаються дані ЕКГ сигналу. Цей клас та його метод реалізовані у всіх наступних сторінках нашого проекту.

Реалізація класу “GraphCreator” та його методу “doInBackground(Strings... strings)” зображено на рисунку 3.52.

```
class GraphCreator extends AsyncTask<String, Void, Void> {
    LineGraphSeries<DataPoint> series = new LineGraphSeries<>();
    private double x;
    private double y;

    @Override
    protected Void doInBackground(String... strings) {
        graph.addSeries(series);
        series.setTitle("Noisy ECG Signal");
        series.setAnimated(false);
        Thread k = new Thread(new Runnable() {
            @Override
            public void run() {
                String json = null;
                try {
                    InputStream inputStream = getAssets().open(strings[0]);
                    byte[] data = new byte[inputStream.available()];
                    inputStream.read(data);
                    inputStream.close();
                    json = new String(data, StandardCharsets.UTF_8);
                    JSONObject jsonObject = new JSONObject(json);
                    JSONArray jsonArray = jsonObject.getJSONArray( name: "ecg");
                    for (x = 0; x < jsonArray.length(); x++) {
                        Thread.sleep( millis: 5);
                        JSONObject obj = jsonArray.getJSONObject((int) x);
                        y = obj.getDouble( name: "y");
                        h.post(crGraph);
                    }

                } catch (IOException | JSONException | InterruptedException e) {
                    e.printStackTrace();
                }
            }

        });
        k.start();
        return null;
    }
```

Рисунок 3.52 – Реалізація класу “GraphCreator” та його методу “doInBackground(Strings... strings)”

Де “LineGraphSeries<DataPoint> series” – це функція графіка;

“x” та “y” – це точки функції;

“addSeries(Series series)” – це метод, який додає графіку певну функцію “series” ;

“setTitle(String title)” – це метод, який встановлює нашої функції заголовок “title” ;

“setAnimated(Boolean animated)” – це метод, який робить наш графік анімованим, якщо передати істину, і робить його статичним, якщо передати брехню;

“Thread k” – це зовнішній потік програми в якому відбуватимуться: зчитування даних ЕКГ, обробка цих даних та додавання їх на графік. На вхід даному потоку було передано об'єкт інтерфейсу Runnable.

Почнемо опис методу “run()” у об'єкта Runnable.

Для початку проініцілізуємо рядок “json”, в який ми потім запишемо всі дані про сигнал ЕКГ з файлу. Потім у блоці try, catch ми відкриємо файл з даними ЕКГ сигналу з папки assets, за допомогою методу “getAssets()”, який повертає нам усі файли з вищезгаданої папки. Відкривати сам файл ми за допомогою методу “open(String filename)”. Після відкриття файлу, цей файл запишемо у вхідний потік InputStream. Далі за допомогою методу “read(byte b[])” прочитаємо всі символи з вхідного потоку та запишемо їх у масив байтів data. Після цього привласним рядку “json” перетворений на рядок масив байтів “data” і закриємо вхідний потік InputStream. За цим ми створюємо об'єкт класу JSONObject, який потрібен нам для коректного зчитування даних з рядка, тому що рядок у нас має вигляд даних як у JSON. Так як вид даних є не зовсім JSONObject, а скоріше масив об'єктів JSON, ми скористаємося об'єктом класу JSONArray, щоб отримати з JSONObject масив JSON. Отримувати масив JSON ми за допомогою методу “getJSONArray(String name)”, де “name” - назва масиву об'єктів JSON. Далі в циклі “for”, з масиву “json” ми будемо отримувати об'єкт “json” за допомогою методу “getJSONObject(int index)”, де на місце “index” ми будемо вставляти номер ітерації, таким чином кожну нову ітерацію у нас буде діставатися з масиву новий об'єкт “json”. В об'єкті “json” у нас буде значення y за відлік x. З даного об'єкта ми будемо діставати значення “y” з сигналу ЕКГ за допомогою методу “getDouble(String name)”, де “name” - це назва ключа, і класти це значення в змінну “y”. Після всього через Handler будемо запускати інший об'єкт Runnable “crGraph” котрий і додаватиме отримані точки на наш графік.

Реалізація об'єкту Runnable “crGraph” зображена на рисунку 3.53.

```
Runnable crGraph = new Runnable() {
    @Override
    public void run() {
        series.appendData(new DataPoint(x, y), scrollToEnd: false, maxDataPoints: 2000);
    }
};
```

Рисунок 3.53 – Реалізація об'єкту Runnable “crGraph”

Де додавання точок на графік здійснюється за допомогою методу “appendData(E dataPoint, boolean scrollToEnd, int maxDataPoints)” ;

“dataPoint” – це точка, яка буде додана на графік;

“scrollToEnd” – це булевська змінна, яка вказує чи буде прокручуватися графік до останньої доданої точки чи ні;

“maxDataPoints” – це максимальна кількість точок, що будевідобратися.

Таким чином, ми завершили реалізацію сторінки “Графік зашумленого ЕКГ”.

Візуальне відображення сторінки “Графік зашумленого ЕКГ сигналу” можна розглянути на рисунку 3.54.

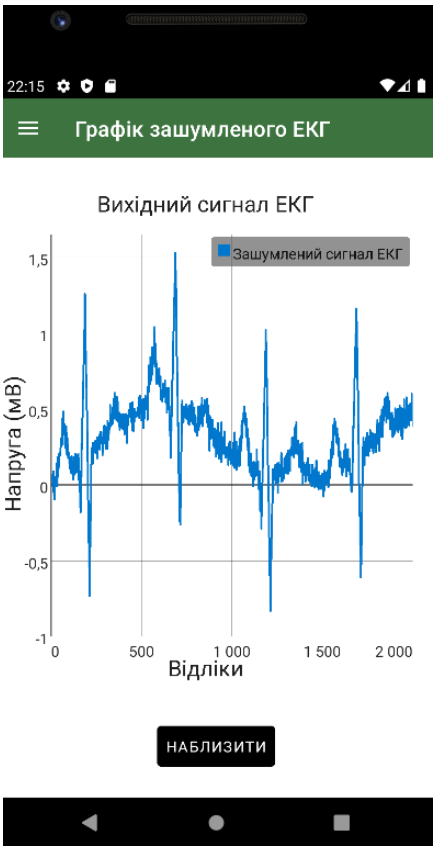


Рисунок 3.54 – Візуальне відображення сторінки “Графік зашумленого ЕКГ сигналу”

3.6 Розробка сторінки “Графік постійної ЕКГ сигналу”

Ця сторінка реалізована за принципом попередньої сторінки та має важливе значення для мобільного додатку.

Контент сторінки ідентичний з контентом попередньої сторінки, єдине, що буде додано - це функція постійної екг сигналу. Ця функція буде безпосередньо додана на наш графік GraphView. Також на графіку попередньої сторінки вхідний сигнал ЕКГ був дуже зашумлений, що не дуже зручно для аналізу цього сигналу, тому цей сигнал ми згладимо за допомогою свого методу “slide”, реалізацію якого ми опишемо нижче. Функція постійної екг також буде розрахована за допомогою методу “slide”.

Постійна ЕКГ сигналу – це функція кривої лінії, яка вираховується із значень ЕКГ сигналу, без урахування зубців QRS комплексу.

На графіку “зашумленого ЕКГ сигналу” ЕКГ сигнал зашумлений, але він ще й не вирівняний щодо вісі X, що теж є незручністю при аналізі ЕКГ сигналу. Саме завдяки постійній екг сигналу ми вирівняємо цей графік шляхом віднімання з початкової точки "y" сигналу ЕКГ точки "y" постійної ЕКГ сигналу. А потім запишемо всі ці нові точки графіка в окремий файл, який будемо використовувати на сторінках програми з аналізом ЕКГ сигналу.

Основний функціонал, ініціалізація та інші моменти ідентичні з минулою сторінкою програми, тому я опишу тільки основний метод “slide”, який дозволяє нам побудувати графік постійної ЕКГ сигналу та згодом вирівняти сигнал ЕКГ щодо вісі X.

“slide” - це метод, написаний на основі функції “ковзного середнього”.

“Ковзне середнє” — загальна назва для сімейства функцій, значення яких у кожній точці визначення дорівнюють деякому середньому значенню вихідної функції за попередній період. Ковзаючі середні зазвичай використовуються для згладжування короткострокових коливань та виділення основних тенденцій або циклів.

Реалізація методу “slide(double[] numS, int k)” зображена на рисунку 3.55.

```
private static double[] slide(double[] numS, int k) {
    double[] res = new double[numS.length];
    int startIndex;
    int endIndex;
    for (int i = 0; i < numS.length; i++) {
        double sum = 0;
        int devNum = 0;
        endIndex = k + (i - k / 2);
        for (startIndex = i - k / 2; startIndex < endIndex; startIndex++) {
            if (startIndex < 0) {
                startIndex = 0;
                devNum = endIndex - startIndex;
            }
            if (startIndex == i - k / 2) {
                devNum = endIndex - startIndex;
            }
            if (endIndex > numS.length) {
                endIndex = numS.length;
                devNum = endIndex - startIndex;
            }
            sum += numS[startIndex];
        }
        res[i] = (sum / devNum);
    }
    return res;
}
```

Рисунок 3.55 – Реалізація методу “slide(double[] numS, int k)”

Де “numS” – це масив точок “y” всього сигналу ЕКГ, з яких ми будемо обчислювати середнє значення на проміжку;

“k” – це встановлений розмір вікна ковзного середнього, кількість змінних з яких ми будемо обчислювати середнє значення для однієї точки;

“startIndex” – це індекс першої точки відліку середнього ковзного;

“endIndex” – індекс останньої точки відліку середнього ковзного.

Далі в циклі “for” для кожної точки масиву проходить основний розрахунок середнього значення в діапазоні точок від “startIndex до endIndex” за допомогою вкладеного циклу for. Оскільки алгоритм “середнього ковзного вікна” є розрахунок значень точок і зліва і справа від нашої основної точки, то для коректного підрахунку нам потрібно реалізувати кілька умов if. Розрахунок середнього значення відбувається за рахунок розподілу локальної змінної “sum”, що позначає суму значень точок “y”, на змінну “devNum”, яка позначає кількість точок “y”. У результаті це середнє значення записуємо в масив точок функції постійної ЕКГ. Наприкінці роботи нашого методу, метод повертає нам масив нових точок у для побудови графіка постійного сигналу “ЕКГ”.

Згладжування основного сигналу ЕКГ ми також робимо за допомогою даного методу, так як завдяки цьому методу, розриви між точками основного графіка зменшаться і таким чином відбудеться згладжування шуму.

Візуальне відображення сторінки “Графік постійної ЕКГ сигналу” можна розглянути на рисунку 3.56.

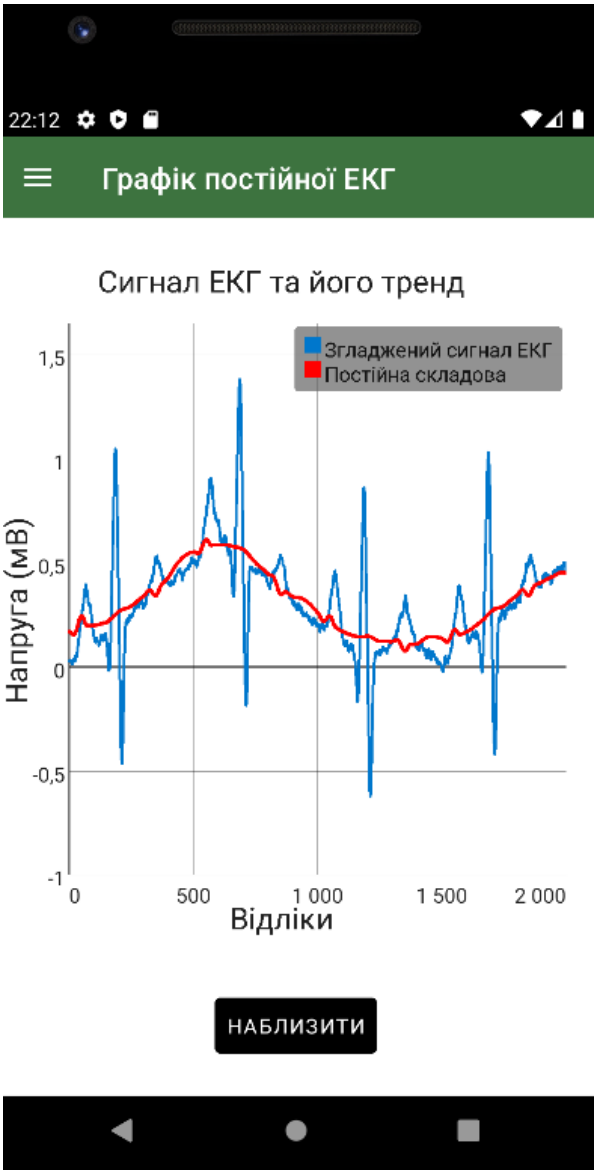


Рисунок 3.56 – Візуальне відображення сторінки “Графік постійної ЕКГ сигналу”

3.7 Розробка методів пошуку QRS зубців

Комплекс QRS є комбінацією трьох графічних відхилень, що спостерігаються на типовій електрокардіограмі ЕКГ. Зазвичай це центральна та найбільш візуально очевидна частина зображення; іншими словами, це основний сплеск, що спостерігається на лінії ЕКГ.

Ознайомитись з QRS комплексом можна на рисунку 3.57.

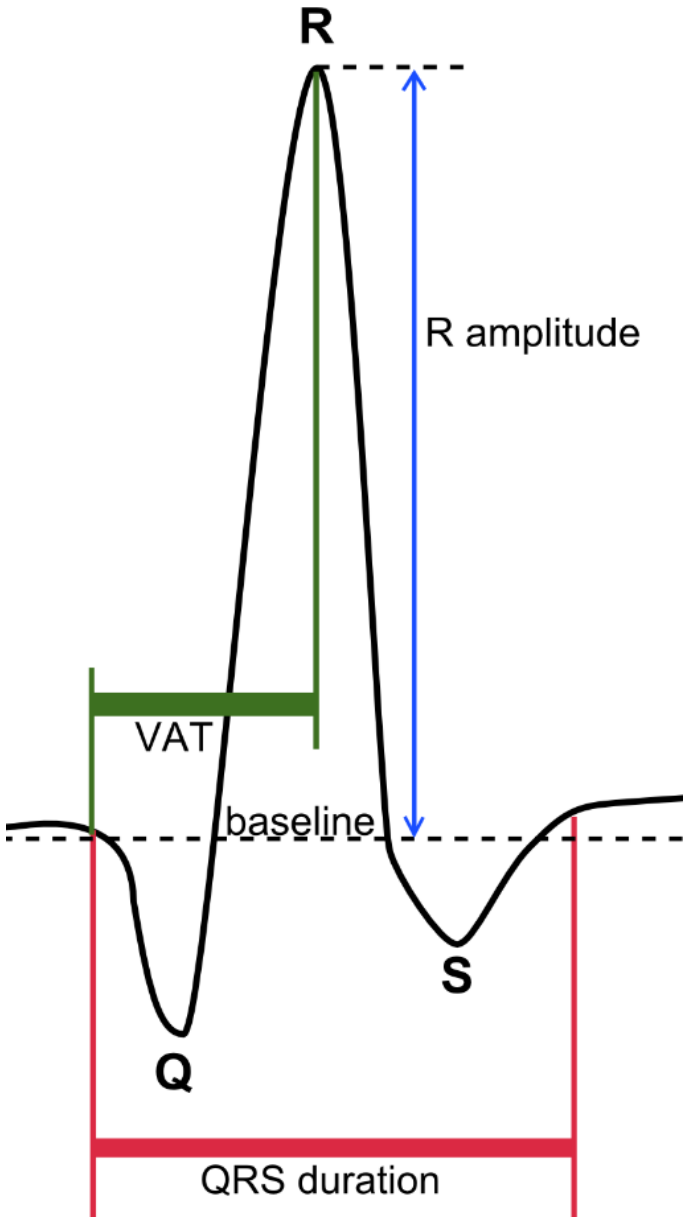


Рисунок 3.57 – QRS комплекс

Пошук зубців QRS є найважливішим завданням даного мобільного додатка, тому що саме за допомогою даних зубців ми будемо аналізувати роботу серця. Пошук зубців QRS реалізовано трьома способами. Перший метод “findR” - це метод, який використовується для пошуку R зубця. Другий метод “findS” - це метод, який використовується для пошуку S зубця. Третій метод “findQ” - це метод, який використовується для пошуку Q зубця. Всі ці методи реалізовані в спеціальному класі FindQRSPoints.

Детально опишемо кожен із методів.

Реалізація методу “findR(double[] yArr)” зображена рисунку 3.58.

```java
public DataPoint[] findR(double[] yArr) {
    int partCount = (int) Math.ceil((double) yArr.length / PART_NUMBERS_COUNT);
    ArrayList<DataPoint> dpList = new ArrayList<>();
    for (int i = 0; i < partCount; i++) {
        double yMax = 0.0;
        double xMax = 0;
        int tillStep = (i + 1) * PART_NUMBERS_COUNT;
        for (int j = i * PART_NUMBERS_COUNT; j < tillStep; j++) {
            if (tillStep > yArr.length) {
                tillStep = yArr.length;
            }
            if (yArr[j] > yMax) {
                yMax = yArr[j];
                xMax = j;
            }
        }
        if (yMax < 0.6) {
            continue;
        }
        dpList.add(new DataPoint(xMax, yMax));
    }
    DataPoint[] resultPoints = new DataPoint[dpList.size()];
    resultPoints = dpList.toArray(resultPoints);
    return resultPoints;
}
```

Рисунок 3.58 – Реалізація методу “findR(double[] yArr)”

Де “yArr” – це масив точок “y” сигналу ЕКГ, на яких ми будемо шукати зубець R;

“partCount” – це кількість частин графіка, на які був поділений весь відрізок сигналу ЕКГ;

“dpList” – це структура даних, в яку ми будемо зберігати знайдені R зубці.

Далі в циклі “for” ми будемо в кожній частині графіка “partCount” шукати R зубець, якщо R зубець буде знайдений, ми додаємо його до нашої структури даних “dpList”. За R зубець ми приймаємо максимальну точку на відрізку, тому створюємо локальну змінну “yMax” в якій буде зберігатися максимальне значення. “yMax” буде перезаписувати в себе значення до тих пір, поки на відрізку не знайдеться найбільше значення, це і буде наш зубець R. Якщо максимальна точка “yMax” дорівнюватиме менше “0.6”, то цю точку ми не вважаємо за R зубець і продовжуємо пошук. Сам пошук “yMax” ми здійснюємо у вкладеному циклі for, де “j” – це стартова точка відрізка, “tillStep” – це кінцева точка відрізка, “PART_NUMBERS_COUNT” – це кількість точок в одному відрізку. Наприкінці роботи нашого методу, метод повертає нам масив R зубців.

Далі в нас метод “findS”, реалізація якого схожа на метод “findQ”, але зі своїми відмінними рисами. Головна відмінність методу полягає в тому, що S зубці шукаються у мінімальних діапазонах точок “y”. Крім цього в циклі “for” методу

“findS” ми не будемо ітеруватися по всім точках вхідного масиву. Так як зубці QRS - це комплекс, нам достатньо просто передати вхідний масив точок у метод “findR”, який поверне нам точки зубців R, і вже від точок зубців R ми будемо шукати зубці S у нижньому діапазоні значень “y” вхідного масиву.

Реалізація методу “findS(double[] yArr)” зображена рисунку 3.59.

```
public DataPoint[] findS(double[] yArr) {
    DataPoint[] dataPointsYMax = findR(yArr);
    ArrayList<DataPoint> dpList = new ArrayList<>();
    if (dataPointsYMax.length == 0) {
        return null;
    }
    for (DataPoint dataPoint : dataPointsYMax) {
        double yMin = 0.0;
        double xMin = 0;
        for (int i = (int) dataPoint.getX(); i < dataPoint.getX() + 60; i++) {
            if (yArr[i] < yMin) {
                yMin = yArr[i];
                xMin = i;
            }
        }
        if (yMin == 0) {
            continue;
        }
        if (yMin > -0.51) {
            continue;
        }
        dpList.add(new DataPoint(xMin, yMin));

    }
    DataPoint[] resultPoints = new DataPoint[dpList.size()];
    resultPoints = dpList.toArray(resultPoints);
    return resultPoints;
}
```

Рисунок 3.59 – Реалізація методу “findS(double[] yArr)”

Де “dataPointsYMax” – це масив точок R зубців;

“dpList” – це структура даних, в яку ми будемо зберігати знайдені S зубці;

Далі відбувається пошук зубців у циклі for, який ми описали вище.

“yMin” – це мінімальне значення у в діапазоні, безпосередньо сам зубець S.

Якщо мінімальна точка "yMin" дорівнюватиме більше "-0.51", то цю точку ми не вважаємо S зубцем та продовжуємо пошук.

Наприкінці роботи нашого методу, метод повертає нам масив S зубців.

Останній метод “findQ” практично нічим не відрізняється від попереднього методу. Єдина відмінність це те, що пошук самого зубця Q буде відбуватися не з

правої частини від R зубця, а з лівої. Також трохи змінено сам діапазон пошуку Q зубця, шукатимемо мінімум будемо у діапазоні значень “y” до "-0.51".

Реалізація методу “findQ(double[] yArr)” зображена рисунку 3.60.

```
public DataPoint[] findQ(double[] yArr) {
    ArrayList<DataPoint> dataPoints = new ArrayList<>();
    DataPoint[] dataPointsYMax = findR(yArr);
    if (dataPointsYMax.length == 0) {
        return null;
    }
    for (DataPoint dataPoint : dataPointsYMax) {
        double yMin = 0;
        int xMin = 0;
        for (int i = (int) dataPoint.getX() - 50; i < dataPoint.getX(); i++) {
            if (yArr[i] < yMin && yArr[i] < -0.2) {
                yMin = yArr[i];
                xMin = i;
            }
        }
        if (yMin < -0.51) {
            xMin = 0;
            yMin = 0;
        }
        if (xMin == 0 || yMin == 0) {
            continue;
        }
        dataPoints.add(new DataPoint(xMin, yMin));
    }
    DataPoint[] dataPointsFinal = new DataPoint[dataPoints.size()];
    dataPointsFinal=dataPoints.toArray(dataPointsFinal);
    return dataPointsFinal;
}
```

Рисунок 3.60 – Реалізація методу “findQ(double[] yArr)”

На цьому розробку методів пошуку QRS зубців ми закінчили, тепер ми можемо приступати до подальшої розробки сторінок мобільного додатка.

3.8 Розробка сторінки “Графік пошуку зубців QRS”

Сторінка "Графік пошуку зубців QRS" - це демонстраційна сторінка проекту, на графіку якої користувач на своєму ЕКГ бачить певні RSQ комплекси. Методи для пошуку даних комплексів ми розробили в минулому пункті проекту. За відображення даних зубців відповідає клас PointsGraphSeries<DataPoint>.

Приклад графічного відображення точок за допомогою класу PointsGraphSeries<DataPoint> зображено рисунку 3.61.

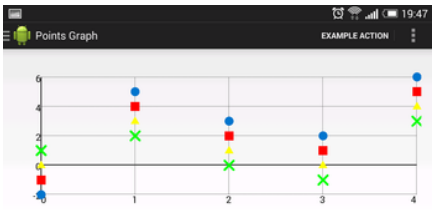


Рисунок 3.61 – Приклад графічного відображення точок за допомогою класу PointsGraphSeries<DataPoint>

Крім додавання на основний графік точок QRS, також ми вирівняли графік сигналу ЕКГ щодо осі X. Функціонал вирівнювання був реалізований на сторінці “Постійної ЕКГ сигналу” та передано на цю сторінку програми.

Візуальне відображення сторінки “Графік пошуку зубців QRS” можна розглянути на рисунку 3.62.

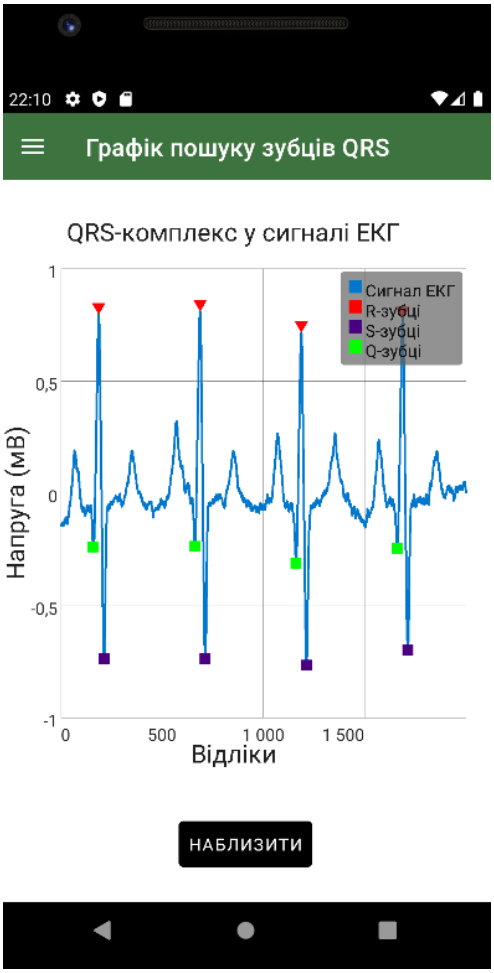


Рисунок 3.62 – Візуальне відображення сторінки “Графік пошуку зубців QRS”

3.9 Розробка сторінки “ Графік аналізу даних ЕКГ”

Сторінка “Графік аналізу даних ЕКГ” - це найголовніша сторінка в мобільному додатку. Вона об'єднує функціонал усіх вищезгаданих сторінок. На цій сторінці в режимі реального часу користувач спостерігатиме за поведінкою свого серця за допомогою відображення сигналу ЕКГ та його аналізу. Графік цієї сторінки безперервно буде відображати сигнал ЕКГ та його аналітику, доки користувач не вийде з програми або зі сторінки.

Так як у мене не було фізичної можливості отримати трекер Omron KardiaMobile ECG, який зчитував би показники серця і передавав нашому додатку, мені довелося реалізувати імітацію ЕКГ сигналу, який який ми будемо аналізувати.

Унікальність ЕКГ сигналу буде досягнуто за допомогою додавання шуму до вихідного сигналу.

Усього я реалізував три варіанти генерації ЕКГ сигналу:

1) ЕКГ сигнал здорового серця – це вихідний сигнал ЕКГ без додавання до нього шуму.

2) ЕКГ сигнал серця з незначними проблемами – це вихідний сигнал ЕКГ з додаванням до нього рандомного шуму

3) ЕКГ сигнал серця з дуже серйозними проблемами - це вихідний сигнал ЕКГ із додаванням негативного шуму у районах зубців R і S.

Залежно від ЕКГ сигналу та його аналізу, мобільний додаток буде видавати нам певний результат по роботі серця: “Проблеми не виявлено” – якщо серце здорове та коректно працює, “Виявлено невеликі проблеми” – якщо не виявлено п'ятдесят і більше відсотків зубців одного типу, “Виявлено серйозні проблеми” – якщо п'ятдесят і більше відсотків всіх зубців не виявлено, “Виявлено дуже серйозні проблеми, зв'яжіться з лікарем” - якщо визначити RSQ зубці практично неможливо.

В останньому та передостанньому випадку, коли у пацієнта виявлено серйозні проблеми в роботі серця, ми надсилаємо повідомлення на пошту його лікарю.

Оскільки функціонал цієї сторінки це весь функціонал попередніх сторінок у сукупності, немає жодного сенсу описувати його повторно.

Варіанти відпрацювання сторінки "Графік аналізу даних ЕКГ" зображені на рисунках нижче.

Відпрацювання сторінки аналізу ЕКГ без додавання шуму відображено на рисунку 3.63.

Рисунок 3.63 Відпрацювання сторінки аналізу ЕКГ без додавання шуму

Відпрацювання сторінки аналізу ЕКГ з рандомним шумом відображено на рисунку 3.64.

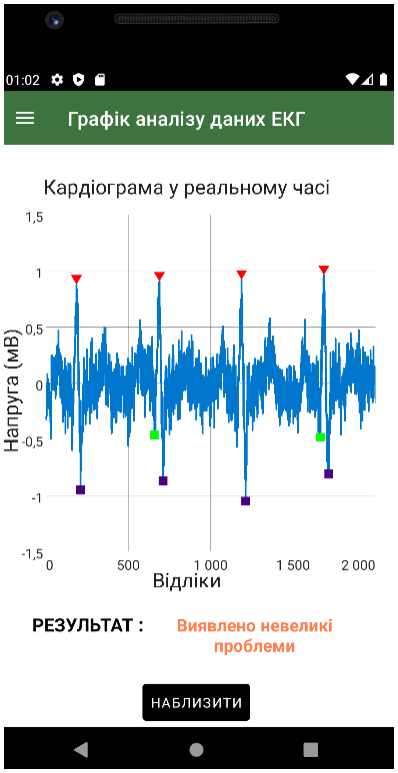


Рисунок 3.64 – Відпрацювання сторінки аналізу ЕКГ з рандомним шумом

Відпрацювання сторінки аналізу ЕКГ з негативним шумом відображено на рисунку 3.65.

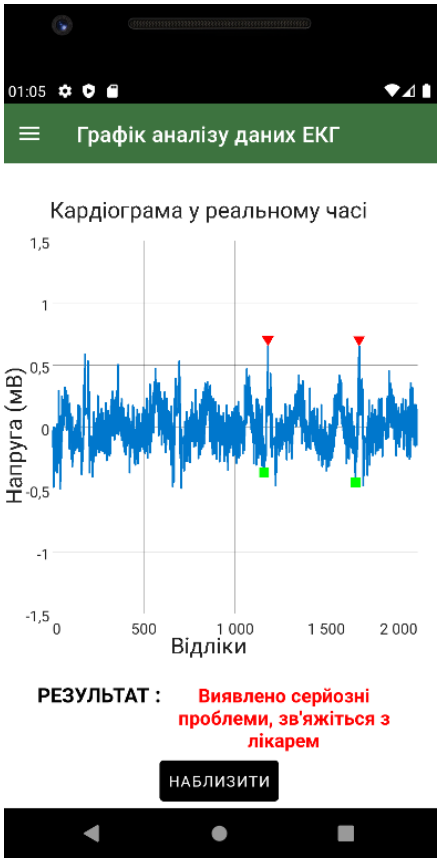


Рисунок 3.65 – Відпрацювання сторінки аналізу ЕКГ з негативним шумом

Приклад надісланого на пошту повідомлення лікарю зображено на рисунку 3.66.

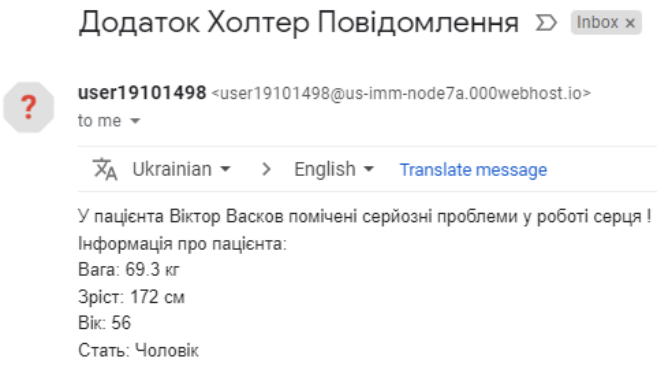


Рисунок 3.66 – Приклад надісланого на пошту повідомлення лікарю

На цьому розробка технічної частини програми закінчена.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ "ХОЛТЕР"

4.1 Основні поняття тестування програм

Тестування – це контроль якості будь-якого продукту розробки: мобільного додатка, сайту чи комп'ютерної програми. Його завдання – зробити кінцеву версію максимально зручною, надійною та безпечною для користувача.

Тестування є важливою частиною розробки програмного забезпечення, оскільки дозволяє виявляти помилки та забезпечує більш надійну роботу програми. Воно забезпечує впевненість у тому, що програмне забезпечення відповідає специфікаціям та працює так, як очікується. Через тестування програми можна гарантувати, що вона працює правильно відповідно до вимог замовника чи заданих специфікацій. Щоб знайти якнайбільше помилок, тестувальник моделює можливі ситуації та сценарії поведінки. Якщо цього не зробити, висока ймовірність замість якісного програмного забезпечення видати абсолютно марний продукт з купою помилок.

Тестування підвищує впевненість у роботі програми та її надійності, що особливо важливо для програм, які можуть бути критичними з погляду безпеки чи безперервної роботи. Програмне забезпечення часто модифікується та розширюється. Тести дозволяють забезпечити, що з внесення змін існуючий функціонал зберігає свою працездатність.

Виявлення та виправлення проблем на ранніх етапах розробки програми заощаджує час та ресурси, які в іншому випадку могли б знадобитися для пошуку та виправлення проблем у пізніші періоди розробки або навіть після випуску продукту. Оскільки тестування допомагає забезпечити більш високу якість програми, це сприяє підвищенню задоволеності та довіри користувачів до програми.

Тестування також дозволяє знизити ризики збитків пов'язаних з неправильною роботою програми або порушенням конфіденційності даних, що зберігаються або оброблюються програмою. Філософія стверджує, що помилки завжди присутні — в будь-якій програмі. Знайти їх усіх неможливо, але якщо не вдалося виявити жодної — робота тестувальника вважається невдачею. Помилки можуть бути виявлені ще на етапі планування системи або навіть під час складання технічного завдання. Щоб їх мінімізувати, код проходить тестування на різних стадіях.

Основні види тестування:

1) функціональне – визначає наскільки програмне забезпечення виконує поставлені завдання, як реагує на дії користувача;

2) нефункціональне – виявляє продуктивність, надійність;

3) статистичне – зазвичай проводять на початку, ще до запуску програми: вивчають документацію і вже існуючий код;

4) динамічне – наступний етап, програму запускають та тестують «у справі».

5) ручне – коли всі випробування виконуються вручну, без автоматизації;

6) автоматичне – із застосуванням програмних засобів;

7) тестування за принципом чорного та білого ящика – у першому варіанті робота ведеться без доступу до коду. Тестувальник перевіряє продуктивність, функції, помилки в інтерфейсі. У другому – код відкритий. Виконується перевірка структури та логіки програми.

Рівень тестів визначається стадією розробки проекту.

Модульне тестування проводиться на початку, коли зібрані лише окремі блоки коду. Під кожну функцію чи метод пишуться тести. Це перший рівень, який можуть проводити і розробники.

Потім виконується інтеграційне тестування. Коли модулі об'єднуються та утворюють цілісний компонент, тести визначають, як він функціонує, перевіряють на сумісність з операційною системою та апаратною частиною.

При системному тестуванні виявляють, наскільки програма відповідає вимогам, чи всі функції, що запитуються, виконуються.

Приймальне тестування — завершальне. Воно здійснюється при передачі кінцевого продукту замовнику. Мета — показати, що програмне забезпечення повністю відповідає вимогам та виконує всі поставлені завдання.

На сьогодні існує чимало фреймворків для тестування Java-програм, які надають різноманітні можливості для автоматизації тестів та полегшення процесу розробки.

Ось декілька з найпопулярніших фреймворків:

1) JUnit. Одна з найпопулярніших бібліотек для написання тестових сценаріїв в Java. JUnit використовує аннотації для визначення тестових методів та надає багато вбудованих перевірок для спрощення тестування.

2) Selenium: Це популярний фреймворк для тестування веб-додатків, який дозволяє автоматизувати тестування веб-інтерфейсу. Selenium надає можливість записувати, редагувати та відтворювати тестові сценарії для веб-додатків.

3) Mockito. Це фреймворк для створення та використання моків (підроблених об'єктів) у тестах. Mockito дозволяє створювати умовні сценарії для тестування класів залежностей.

Ці фреймворки надають широкий спектр можливостей для розробки та виконання тестів в Java, що сприяє підвищенню якості програмного забезпечення та полегшує процес розробки.

Для тестування мобільного додатку було обрано фреймворк JUnit, функціонал та переваги якого ми розглянемо у наступному пункті.

4.2 Фреймворк JUnit: Огляд та основні можливості

JUnit — це фреймворк для мови програмування Java, призначений для автоматичного тестування програм. Його основне призначення — unit-тестування, тобто таке, коли окремо перевіряється функціональність кожного компонента програми.

Юніт-тестування також називають модульним. Благодаря йому програми працюють як треба: можливі помилки та непередбачені ситуації виявляють тестувальники, після чого програмісти усувають недоліки.

Автоматичне тестування допомагає заощадити ресурси: час фахівців, сили та кошти. При цьому воно точніше та швидше, ніж ручне тестування. За допомогою фреймворків, таких як JUnit, навіть складні тести можна створювати легше та швидше.

JUnit «виріс» з серії фреймворків xUnit — вони існують для різних мов програмування та всі орієнтовані на тестування. У них схожа архітектура та функціональність.Знання JUnit корисні як початківцям, щоб ефективніше виконувати завдання, так і професіоналам, щоб розробляти більш обґрунтовані стратегії тестування.

Іноді розробники самі проводять тестування програм, які вони пишуть. У деяких компаніях процес розробки побудований так, що спочатку програмний код перевіряють самі програмісти, а потім його передають тестувальникам. У таких випадках розробникам також потрібні відповідні фреймворки, включаючи JUnit. Їми користуються фахівці, які працюють з мовою програмування Java.

Процес автоматичного тестування виглядає наступним чином: є код, для якого створюють автотести — невеликі програми, які перевіряють певну можливість або компонент. Автотести запускаються і працюють з різними вхідними даними, щоб перевірити роботу основної програми в різних умовах.

Модульне тестування, для якого використовується JUnit, — це вид перевірки, при якому кожен компонент розглядається окремо. Воно допомагає уникнути помилок на "низькому" рівні — на рівні функціонування окремих об'єктів.

Екземпляр тестової програми створюється як спадкоємець від основного класу TestCase. У цього класу є вбудовані методи для ініціалізації, які можна перевизначити, якщо це потрібно, тобто переписати під свої потреби. Після цього екземпляру прописуються методи — функції, конкретні тести, кожен з яких виконує своє завдання. Код всередині методу — це певні дії, а після них — перевірки. Іншими словами, спочатку тест виконує щось з компонентом, який перевіряє, а потім "перевіряє", які результати були отримані. Перевірки показують, чи відповідають результати тим, які передбачалися. Якщо так — тест пройдений. Якщо ні — у роботі компонента є помилка і її слід виправити. Навіть якщо не пройшов лише один із методів або виникло виключення, весь тест вважається неуспішним.

Так як JUnit є фреймворком, він має більш широкі можливості порівняно з вузькими та простими бібліотеками. Тест, написаний за його допомогою, є повноправною окремою програмою, хоч і невеликою. У такому підході є своя перевага: можна відокремити тести від основного коду, тож у фінальній версії програми не буде зайвого. Тестуюча програма не буде включена до збірки.

Юніт-тестування з використанням JUnit також передбачає створення документації. Докладні тести самі по собі пояснюють фахівцям, що робить цей компонент і як він працює. Пояснювати подібні речі потрібно: в комерційній розробці часто зустрічаються ситуації, коли людям доводиться працювати з чужим кодом. Тому краще заздалегідь розповісти їм все за допомогою документації — так буде легше вникнути.

Ось лише кілька технічних можливостей, які надає JUnit, насправді їх набагато більше, але для повного опису знадобиться ціле керівництво:

1) власна точка входу. Це те, про що ми говорили вище: тест на JUnit — окрема програма. Точка входу — це, наприклад, метод main(), з якого починається виконання коду. Всередині цього методу знаходиться вся програма. Якщо у тесту немає власної точки входу, це означає, що він виконується в тілі основної програми, а це не завжди зручно. А свій власний метод для запуску дозволяє йому існувати окремо від продукту, і таке розподіл робить код чистішим та зрозумілішим;

2) налаштування тестів. Щоб ефективно протестувати компонент, потрібно попередньо налаштувати готову тестову програму. Іншими словами, визначити, коли і як вона буде запускатися, ініціалізувати її для приведення в робочий стан,

задати вхідні дані або виконати багато іншого. Для всього переліченого в JUnit є власні функції, які дозволяють виконувати ці дії буквально в дві-три рядки коду. Налаштування в JUnit досить гнучкі. Можна окремо налаштувати, які дії будуть виконуватися перед усіма тестами, які — перед кожним або перед певним конкретним. Є можливість також описати дії після завершення тесту — наприклад, очистку пам'яті та видалення вже непотрібних даних;

3) спільне виконання. JUnit дозволяє паралельно запускати кілька тестів або об'єднувати різні тестові програми в набір. Це надає можливість використовувати групу тестів як один, що допомагає тестувальникам, наприклад, у ситуаціях, коли різні тести пишуть різні люди;

4) відключення тестів. Буває так, що зараз запускати якийсь тест не потрібно. Інші тести мають пройти, але конкретний — ні. JUnit дозволяє відключати такі тести за допомогою спеціальної команди @Ignore. Її можна поставити на окремий метод тестової програми або на неї цілком.

5) тайм-аут. За замовчуванням у тестів немає часових обмежень. Але насправді, якщо компонент працює занадто довго, це неправильно і заважає використанню програми. І швидкість роботи також можна протестувати за допомогою JUnit – в ньому є вбудоване правило, яке дозволяє задати тайм-аут. Це означає, що якщо тест не виконається за визначений час, він вважається проваленим. Такий підхід допомагає відслідковувати, наприклад, замішання або неефективні рішення, через які код починає працювати дуже повільно.

6) динамічні тести. Починаючи з версії JUnit 5, у фреймворку з'явилась можливість створювати та запускати динамічні тести. Їх основна відмінність полягає в тому, що такі тести виконуються для програми не в момент компіляції, а в момент запуску. Це розширює можливості тестування. Наприклад, з'являється можливість запускати тест у циклі з різними параметрами, причому це сприймається як один тест, а не декілька різних.

Основні переваги JUnit:

1) "чистий" Java. Тестувальнику або розробнику, скоріше за все, не знадобиться працювати з іншими мовами програмування або надбудовами. Фреймворк реалізований на чистому Java, повністю підтримує його принципи програмування, узгодження з принципами іменування, синтаксис і інші особливості. Тому тим, хто вже знайомий з Java, освоїти його буде легко;

2) сумісність з Java-інструментами. З цієї ж причини JUnit повністю сумісний з інструментами, якими зазвичай користуються розробники на Java. Наприклад, він чудово працює з Maven та Gradle — це програмне забезпечення, яке відповідає за

збірку Java-проектів. Він також має зворотню сумісність: це означає, що старі програми можуть працювати з новими версіями JUnit. Наприклад, якщо якась програма писалась, коли був актуальний JUnit 4, то вона зрозуміє і JUnit 5, а він, в свою чергу, зрозуміє її;

3) орієнтованість на TDD. У розробці через тестування є кілька переваг, і вона досить популярна. JUnit дозволяє повністю реалізувати її принципи, тому інструмент не потрібно серйозно переделывать під цей підхід. Впровадження TDD стає простіше, а працювати з нею — зручніше.

4) популярність. JUnit — дуже поширений інструмент для модульного тестування, тому завжди є багато матеріалів для тестувальників різних рівнів. Через це з ним досить легко працювати: спільнота широка, завжди можна поставити питання, яке цікавить або проконсультуватися з іншими спеціалістами. Ентузіасти можуть писати для фреймворка свої інструменти або давати поради щодо його використання, а у багатьох задач вже є типові рішення. Це допомагає і в навчанні, і в роботі.

Основні недоліки JUnit:

1) JUnit сильно орієнтований на Java: в ньому ті ж самі особливості іменування та досить багатослівний код. Це плюс для розробників Java і мінус для багатьох інших фахівців. Людині, яка не розбирається в Java, буде важко зрозуміти текст коду. І якщо перегляд тесту знадобиться фахівцю з зовсім іншої галузі, ймовірно, він нічого не зрозуміє. Щ

2) відсутність вбудованих mock-об'єктів. Так називаються сутності-заглушки: вони "імітують" функції справжніх об'єктів, які будуть працювати в коді. У них більш примітивна структура порівняно з реальними об'єктами і жорстко заданий функціонал. Їх часто використовують у тестуванні, але в JUnit відсутня для них механіка: доводиться підключати додаткову бібліотеку Mockito.

Хоча JUnit має деякі обмеження, які можуть не підходити для всіх видів проектів, він все ж є одним із найкращих фреймворків для автоматизованого тестування Java-додатків.

4.3 Практичне застосування JUnit в тестуванні Java-додатків

Для початку нам необхідно встановити JUnit у ваш проект. Якщо ми використовуємо Maven або Gradle, то потрібно додати залежність JUnit до нашого файлу налаштувань.

Приклад додавання залежності для Maven зображено на рисунку 4.1.

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.8.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.8.1</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Рисунок 4.1 – Додавання залежності для Maven

Тестовий клас створюється окремо від основного коду. Він містить один або кілька тестових методів, кожен з яких перевіряє окремий аспект роботи коду. Тестові методи позначаються анотацією @Test. Анотації в Java є своєрідними мітками в коді, що описують метадані для функції/класу/пакету. Наприклад, всім відома анотація @Override, що позначає, що ми збираємося перевизначити метод батьківського класу.

Приклад структури класу для тестування зображено на рисунку 4.2.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculatorTest {

    @Test
    public void testAddition() {
        Calculator calculator = new Calculator();
        int result = calculator.add(2, 3);
        assertEquals(5, result, "2 + 3 should equal 5");
    }
}
```

Рисунок 4.2 – Приклад структури класу для тестування

Утвердження в JUnit дозволяють перевіряти, чи відповідає результат роботи вашого коду очікуваному. Якщо утвердження не виконується, тест вважається невдалим. Для утверджень використовуйте клас org.junit.jupiter.api.Assertions.

Приклад використання утверджень зображено на рисунку 4.3.

```
@Test
public void testSubtraction() {
    Calculator calculator = new Calculator();
    int result = calculator.subtract(5, 3);
    assertEquals(2, result, "5 - 3 should equal 2");
}
```

Рисунок 4.3 – Приклад використання утверджень

Також JUnit дозволяє використовувати методи, які викликаються перед та після кожного тесту або всього тестового класу. Вони корисні для ініціалізації та очищення ресурсів. Для цього використовуйте анотації @BeforeEach, @AfterEach, @BeforeAll та @AfterAll.

Приклад використання анотацій для ініціалізації та звільнення ресурсів зображено на рисунку 4.4.

```
import org.junit.jupiter.api.*;

public class DatabaseTest {

    private DatabaseConnection connection;

    @BeforeAll
    public static void setupClass() {
        // Выполнить действия перед всеми тестами в классе
    }

    @BeforeEach
    public void setup() {
        connection = new DatabaseConnection();
        connection.connect();
        // Выполнить действия перед каждым тестом
    }

    @Test
    public void testDatabaseQuery() {
        // Тестирование запроса к базе данных
    }

    @AfterEach
    public void tearDown() {
        connection.disconnect();
        // Выполнить действия после каждого теста
    }

    @AfterAll
    public static void tearDownClass() {
        // Выполнить действия после всех тестов в классе
    }
}
```

Рисунок 4.4 – Використання анотацій для ініціалізації та звільнення ресурсів

Таким чином ми описали лише малу частину функціоналу фреймворку JUnit, але ця частина є основною і неодноразово використовується для тестування мобільного додатку.

Тестування Java-коду за допомогою JUnit – важливий та корисний процес, який допомагає забезпечити якість та стабільність вашого програмного забезпечення. Використовуючи тестові методи, затвердження, параметризовані тести, життєві цикли, вкладені тести та налаштування, ви можете створювати ретельно протестовані та надійні програми на мові Java. Наведені приклади та поради допоможуть вам швидко розпочати роботу з JUnit та підвищити ефективність вашого процесу розробки.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

В результаті даної роботи був розроблений новий універсальний мобільний додаток, який може замінити велику частину застарілих медичних технологій і набагато полегшити людське життя, заощаджуючи наш час.

На основі проведеної аналітики медичних технологій минулого, можна зробити висновок, що існуючі в наші дні методи аналізу серцево-судинних захворювань трохи застаріли та вимагають оновлення, саме тому запропонований нами мобільний додаток є сучасним та як ніколи актуальним у використанні.

Практична значимість нашої роботи полягає в тому, що завдяки нашому проекту розробники розглянуть по-новому галузь медичних додатків по оздоровленню людства, і за цим піде новий прорив у медичних розробках

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Ian Darwin. Android Cookbook: Problems and Solutions for Android Developers. Publisher: O'Reilly Media, 2018 – 768 с.

2 Paul J Deitel. Android for Programmers: An App-Driven Approach. Publisher: Prentice Hall, 2011 – 481 с.

3 Bill Phillips. Android Programming: The Big Nerd Ranch Guide. Publisher: Big Nerd Ranch Guides, 2019 – 624 с.

4 Thomas H Cormen. Introduction to Algorithms, 3rd Edition. Publisher: MIT Press, 2009 – 1292 с.

5 Google Java Clean Code And Style Guide in action URL: https://google.github.io/styleguide/javaguide.html (дата звернення: 25.02.2023).

6 Herbert Schildt. Java: The Complete Reference, Twelfth Edition. Publisher: McGraw-Hill, 2023 – 1344 с.

7 Sylvain Ratabouil. Android NDK Beginner's Guide. Publisher: Packt Publishing, 2012 – 436 с.

8 Cay Horstmann. Core Java Volume I--Fundamentals (Core Series). Publisher: Pearson, 2018 – 928 с.

9 Brian Goetz. Java Concurrency in Practice 1st Edition. Publisher: 2006. – 432 с.

10 Kathy Sierra. Head First Java, 2nd Edition. Publisher: O'Reilly Media, 2005 – 688 с.

11 Benjamin Evans. The Well-Grounded Java Developer, Second Edition. Publisher: Manning, 2022 – 704 с.

12 Nicolai Parlog. The Java Module System 1st Edition. Publisher: Manning Publications, 2019 – 440 с.

13 Raoul-Gabriel Urma. Modern Java in Action: Lambdas, streams, functional and reactive programming 2nd Edition. Publisher: Manning, 2018 – 592 с.

14 Joshua Bloch. Effective Java 3rd Edition. Publication date: 2017 – 464 с.

ДОДАТОК А

Перелік копій демонстраційного матеріалу

Слайд 1 – Медичні технології аналізу роботи серця теперішнього часу

Слайд 2 – Медичні технології аналізу роботи серця, які ми пропонуємо

Реалізація домашньої сторінки додатку

Слайд 3 – Реалізація домашньої сторінки додатку

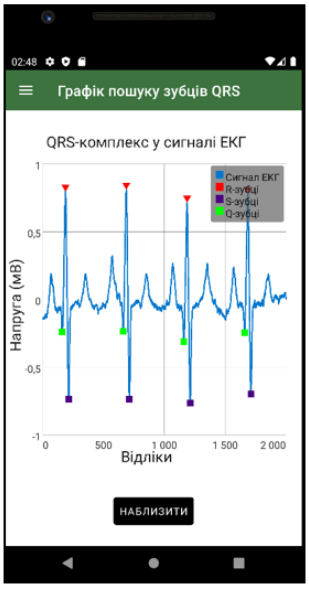
Реалізація сторінки “Графік зашумленого ЕКГ сигналу”

Слайд 4 – Реалізація сторінки “Графік зашумленого ЕКГ сигналу”

Реалізація сторінки “Графік постійної ЕКГ сигналу”

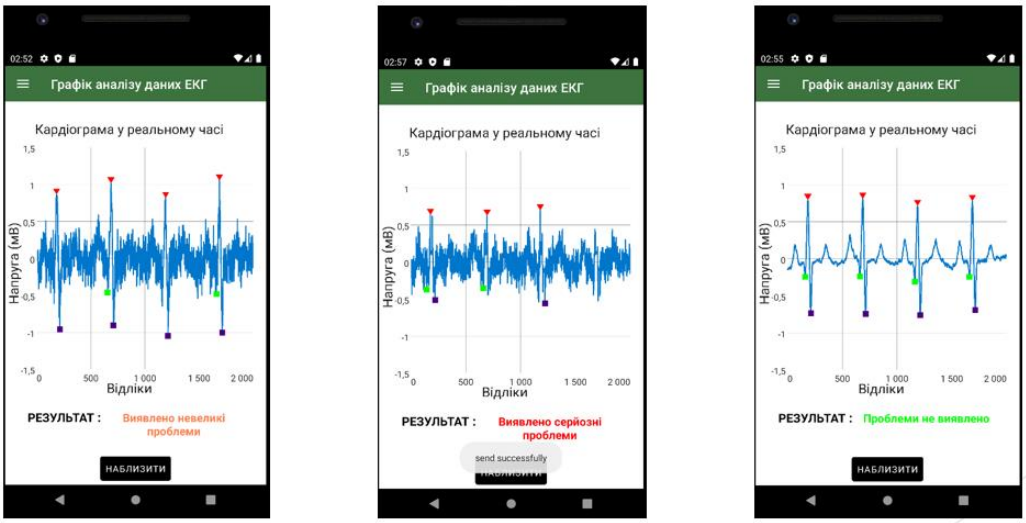
Слайд 5 – Реалізація сторінки “Графік постійної ЕКГ сигналу”

Реалізація сторінки “Графік пошуку зубців QRS”



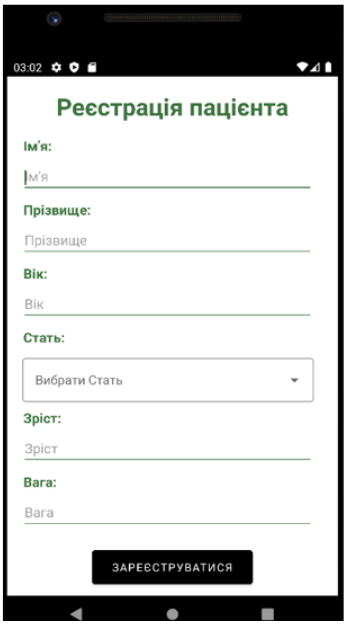
Слайд 6 – Реалізація сторінки “Графік пошуку зубців QRS”

Реалізація сторінки “Графік аналізу даних ЕКГ”



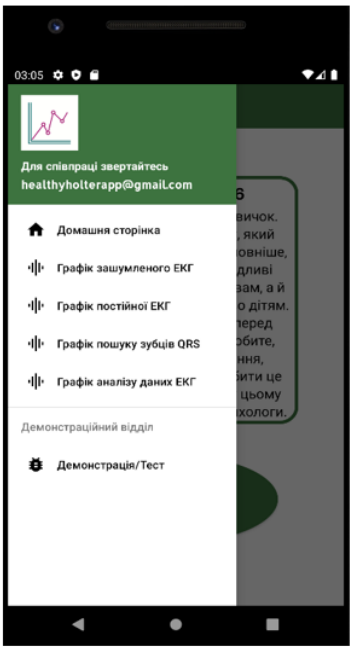
Слайд 7 – Реалізація сторінки “Графік аналізу даних ЕКГ”

Реалізація сторінки реєстрації користувача



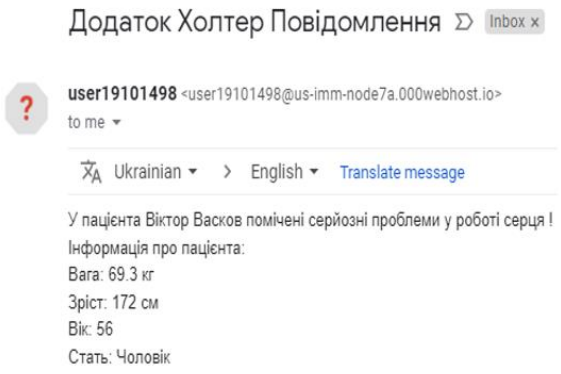
Слайд 8 – Реалізація сторінки реєстрації користувача

Реалізація меню навігації



Слайд 9 – Реалізація меню навігації

Приклад надісланого на пошту повідомлення лікарю



Слайд 10 – Приклад надісланого на пошту повідомлення лікарю