МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЯКУБЕНКО ДМИТРО ОЛЕКСАНДРОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
д. т. н., доцент
_____ Т. В. Нескородєва
«_____» _____ 2022р.

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ З ПІДТРИМКИ ФУНКЦІОНУВАННЯ ПРИВАТНОЇ КЛІНІКИ

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Баркалов О.О. д-р. техн. наук, професор
кафедри інформаційних технологій
_____
(підпис)

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2022

АНОТАЦІЯ

Якубенко Д.О. Розробка мобільного додатку з підтримки функціонування приватної клініки. Спеціальність 122 “Комп’ютерні науки”, Освітня програма “Комп’ютерні технології обробки даних”. Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційній роботі проведено сбір медичної статистики за відвідуваннями пацієнтів та до яких лікарів вони звертались більше за все на протязі певного періоду. Показано приклад роботи додатку та результати аналізу за рік з медичної та фінансової точки зору.

Ключові слова: Android, розробка, статистика, клініки.

65 стр., 29рис., 31 джерел.,

Yakubenko D.O. Development of a mobile application to support the functionality of a private clinic. Specialty 122 "Computer science", Educational program "Computer data processing technologies". Vasyl Stus Donetsk National University, Vinnytsia, 2022.

In the qualifying work, medical statistics were collected based on patient visits and which doctors they consulted the most during a certain period. An example of the application's operation and the results of the analysis for a year from a medical and financial point of view are shown.

Keywords: Android, development, statistics, clinics.

65 pages, 29 pictures, 31 sources,

ЗМІСТ

ВСТУП

У багатьох сферах потребується облік людей за різних причин, так і у медичній сфері відбуваються поновлення та додаються до існуючих фізичних медичних карток їх цифрові версії. Також це стосується інших процесів: запису до лікаря, збереження аналізів, діагнозів та інша інформація, яку до цього зберігали лише на бумазі та функціонал, який дозволяє зберігати час. Замість очікування у черзі з'явилась можливість через сайт чи мобільний додаток записатися на певний день та годину. Для приватної клініки такі можливості спрощують передачу статистик захворювань до загальної статистики захворювань населення. Лікарям такі оновлення дають можливість відслідковувати кількість пацієнтів на кожен день, краще вести статистику та зберігати дані пацієнтів.

Актуальність теми дослідження: Грамотна розробка такого сервісу спрощує роботу цілої клініки, полегшує роботу зі статистикою хворих.
На разі велика частина зроблена у вигляді сайтів, але для зручнішого використання потребуються мобільні версії таких сервісів.

Мета: Аналіз медичної статистики приватної клініки та реалізація мобільного додатку.

Основні задачі: Можливість переглядати історію пацієнта, робити оновлення. Список пацієнтів на день лікаря. Статистика пацієнтів за різними критеріями: За класами (класи беруться із МКХ-10 (Міжнародна класифікація хвороб десятого перегляду), кварталами (1 місяць, 3 місяці, півроку та рік).

Завдання дослідження: Повна реалізація мобільного (Android) додатку з використанням даних інструментів: Kotlin, Android SDK, Git, GitHub, Koin, Coroutine, OkHttp3, Retrofit2, ViewModel, Lifecycle, RecyclerView, Room, Glide, SharedPreferences.

Предмет дослідження: Мобільний додаток з підтримки приватної клініки.

Досягнення поставленої мети потребує вирішення таких задач дослідження:

- Розглянути поняття мобільного додатку;
- Обрати інструмент для створення мобільного додатку;
- Розробити дизайн додатку;

Практична значущість роботи у тому, що її рекомендації можуть бути використані при вивчені мови програмування Kotlin,XML, а розроблений мобільний додаток знайти конкретну практичну реалізацію в галузі медичної статистики для приватної клініки.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ МОБІЛЬНОГО ANDROID ДОДАТКУ ТА ДЕТАЛІЗАЦІЯ ЗАДАЧ ПРОЕКТУВАННЯ

1.1 Аналіз мобільного додатку як предмету дослідження

Мобільний додаток – це програмний продукт, призначений для роботи на смартфонах, планшетах та інших мобільних пристроях. Основні мобільні застосунки встановлені виробників одразу на самому пристрої або можуть бути завантажені на нього з різних онлайн-магазинів мобільних застосунків, таких як AppStore (для смартфонів та планшетів від Apple, GooglePlay (для техніки з OS Android), Windows Phone Store та інших, безкоштовно, за помісячний внесок або за разову плату. Спочатку мобільні застосунки використовувалися для швидкої перевірки електронної пошти, але їх високий попит призвів до розширення їх призначень і в інших областях, таких як ігри для мобільних телефонів, GPS, спілкування, перегляд відео, створення розважального контенту, прослуховування музики та користування Інтернетом.

Велика кількість мобільних пристроїв продаються з уже встановленим набором мобільних застосунків (для андройд пристроїв зазвичай встановлюється пакет програм від Google), такі як веб браузер(Google Chrome,Safari), поштовий клієнт(Gmail), календар (наприклад, Google Calendar), застосунок для придбання та прослуховування музики (Youtube Music, Spotify,Deezer) та інші. Деякі, попередньо встановлені застосунки, можуть бути видалені з пристрою користувачем власноруч, за допомогою звичайного процесу видалення, звільняючи більше місця для зберігання інших (бажаних) застосунків, але додатки встановлені виробником користувач лише може скрити.

Будучи встановленим на пристрій користувача, мобільний додаток є інструментом для прямого контакту з користувачем, і оптимально підходить

для частого і багаторазового використання. За умови грамотної реалізації такий продукт задовольняє конкретну потребу користувача, підвищуючи лояльність аудиторії.

Плюси мобільного додатка:

- Значний рівень інтерактивності,користувач має можливість взаємодіяти з ним різними способами;
- Наявність та доступ до всього або більшої частини функціоналу пристрою, такого як: акселерометр, GPS-навігація, фотокамера, мікрофону, тощо;
- Завдяки можливості зберігати призначений для користувача досвід, швидше завантажуються, і відповідають загальному UI конкретної ОС;
- Їх також можна використовувати в автономному режимі,без наявності Інтернет зв'язку.

Мінуси мобільних додатків:

- Високі вимоги до сумісності(кожного року виходить велика низка різних смартфонів з ОС Android,але з різними надбудовами) та під різні розміри дисплеїв;
- Вартість розробки, підтримки та обслуговування буде набагато вища, ніж у випадку з мобільним сайтом;
- Для розробки мобільного додатку зазвичай потрібна більша кількість людино-годин;
- Щоб розпочати використовувати додаток,користувач повинен встановити його на свій пристрій.

1.2 Класифікація мобільних додатків

Мобільні додатки поділяються на:

- Платні – купуєш та отримуєш повний функціонал;
- Безкоштовні – "добрий розробник" дає можливість використовувати свій додаток безкоштовно;

- Умовно безкоштовно – певний час користувач має можливість спробувати функціонал мобільного додатку,а після завершення пробного періоду вирішити для себе купувати/платити щомісячно за функціонал представлений розробником.

За різновидом мобільні додатки поділяються на:

- Web-додатки, сайти. Найрегулярніший тип додатків. Сучасні мобільні пристрої надають користувачеві, який використовує Інтернет, ті самі можливості, що й персональні комп'ютери. Обумовлено це підтримкою HTML 5. Також, додатки для мобільних пристроїв це чудовий варіант для швидкого старту за невеликі кошти і відмінною функціональністю. Мобільні сайти – це універсальне рішення для всіх платформ.
- Гібридне ПЗ. Це можливість доступу до всього функціоналу мобільних пристроїв, при цьому для їх створення використовуються фреймворки, HTML, Java, CSS. Це нативне ПЗ з інтегрованим мовою розмітки. Такі додатки – чудове рішення для тих, хто бажає використовувати всі переваги нативних додатків в поєднанні з останніми розробками в веб-технологіях. Прикладом гібридного програмного забезпечення може служити Facebook, що завантажується з магазину ПЗ, але потребує підключення до Інтернету.
- Нативне ПЗ. Найвимогливіший до ресурсів вид додатків. У той же самий час, дозволяє скористатися всіма можливостями операційної системи. Це найбільш функціональний і продуктивний вид мобільних додатків.

1.3 Структура мобільного додатку під ОС Android

Мобільний додаток під ОС Android складається з:

1.Android Manifest.xml – це головний файл мобільного додатку.

Його можливості представлені у вигляді xml тегів:

Тег <manifest> – це самий головний тег у якому вкладена вся конфігурація проекту.

За замовчуванням він створюється з файлом AndroidManifest та зразу має певні параметри:

Xmins: android – визначає простір імен Android.

Package - визначає унікальне ім'я пакета програми, яку ви задали під час створення проекту.

Навіщо потрібно вказувати package? Якщо ви захочете завантажити ваш додаток на Google Play, він перевіряє унікальність при прийомі програми, тому рекомендується використовувати своє ім'я для уникнення конфліктів з іншими розробниками.

android:versionCode - по суті, це версія вашої програми. Випускаючи нову версію, ви вказуєте її в цьому полі, воно повинно бути цілим числом.

android:versionName - вказує номер версії користувача. Якщо ви знайшли кілька недоробок у вашому додатку і виправили їх, то в цьому випадку можна вказати для цього поля нову версію, що говоритиме Google Play, що це не нова версія програми, а покращена. Для назви версії можна використовувати рядок або рядковий ресурс.

Тег <uses-permission> - дозволяє вимагати дозволу, які додатку повинні бути надані системою для його нормального функціонування.

Дозволи надаються під час інсталяції програми, а не під час її роботи. <uses-permission> має єдиний атрибут з назвою дозволу android:name. android:name – дозволяє дати дозволи використання ресурсів системи.

Наприклад: android:name='android.permission.CAMERA' або android:name='android.permission.READ_CONTACTS'

Найбільш поширені дозволи –

INTERNET – доступ до Інтернету;

– READ_CONTACTS – читання даних із адресної книги користувача;

– WRITE_CONTACTS – запис даних із адресної книги користувача;

- RECEIVE_SMS - обробка вхідних SMS;

- ACCESS_COARSE_LOCATION - використання приблизного визначення місцезнаходження за допомогою вишок стільникового зв'язку або точок доступу

Wi-Fi;

– ACCESS_FINE_LOCATION – точне визначення місцезнаходження за допомогою GPS.

Тег <permission>

<permission> – дозволяє встановити дозволи на використання ресурсів системи або заборонити використання компонентів програми. android:name – назва дозволу

android:label – ім'я дозволу, яке відображається користувачеві android:description – опис дозволу

android:icon – значок дозволу

android:permissionGroup – визначає приналежність до групи дозволів

android:protectionLevel – рівень захисту

Тег <permission-tree>

<permission-tree> – повідомляє базове ім'я для дерева дозволів. Цей елемент оголошує не саму роздільну здатність, а лише простір імен, в який можуть бути поміщені подальші дозволи.

Тег <permission-group>

<permission-group> – визначає ім'я набору логічно пов'язаних дозволів. Цей елемент не оголошує роздільну здатність безпосередньо, лише категорію, в яку можуть бути розміщені дозволи. Дозвіл можна помістити в групу, призначивши ім'я групи в атрибуті permissionGroup елемента <permission>.

Тег <instrumentation>

<instrumentation> – оголошує об'єкт instrumentation, який дає можливість контролювати взаємодію програми із системою. Зазвичай використовується при налагодженні та тестуванні програми та видаляється з release-версії програми.

Тег <uses-sdk>

<uses-sdk> – дозволяє оголошувати сумісність програми із зазначеною версією (або новішими версіями API) платформи Android.

Рівень API, оголошений програмою, порівнюється з рівнем API системи мобільного пристрою, на який встановлюється ця програма.

Цей тег має такі атрибути:

android:minSdkVersion – визначає мінімальний рівень API, необхідний роботи програми. Система Android перешкоджатиме тому, щоб користувач встановив програму, якщо рівень API системи буде нижчим, ніж значення, визначене в цьому атрибуті. Ви повинні завжди оголошувати цей атрибут, наприклад: android:minSdkVersion="17"

android:maxSDKVersion – дозволяє визначити найпізнішу версію, яку готова підтримувати вашу програму. Ваша програма буде невидимою в Google Play для пристроїв з більш свіжою версією. targetSDKVersion – дозволяє вказати платформу, для якої ви розробляли та тестували програму. Встановлюючи значення цього атрибута, ви повідомляєте системі, що для підтримки цієї конкретної версії не потрібно жодних змін.

Тег <uses-configuration>

<uses-configuration> – вказує потрібну для програми апаратну та програмну конфігурацію мобільного пристрою.

Наприклад, програма для роботи потрібна наявність фронтальної камери або USB порт. Специфікація використовується,

щоб уникнути встановлення програми на пристроях, які не підтримують потрібну конфігурацію.

Якщо програма може працювати з різними конфігураціями пристрою, необхідно включити до маніфесту окремі елементи <uses-configuration> для кожної конфігурації. Ви можете встановити будь-яку комбінацію, що містить наступні пристрої

– reqHardKeyboard – використовуйте значення true, якщо програмі потрібна апаратна клавіатура;

– reqKeyboardType – дозволяє задати тип клавіатури: nokeys, qwerty, twelvekey, undefined;

– reqNavigation – вкажіть одне з значень: nonav, dpad, trackball, wheel або undefined, якщо потрібний пристрій для навігації;

Програма не встановлюватиметься на пристрої, який не відповідає заданій вами конфігурації.

В ідеалі, ви повинні розробити таку програму, яка буде працювати з будь-яким поєднанням пристроїв введення. У цьому випадку <uses-configuration> не потрібно.

Тег <uses-feature>

<uses-feature> оголошує певну функціональність, потрібну для роботи програми.

Таким чином, програма не буде встановлена на пристроях, які не мають необхідної функціональності. Наприклад, програма могла б визначити, що вона вимагає фронтальну камеру з автофокусом. Якщо пристрій не має вбудованої передньої камери з автофокусом, програма не буде інстальовано.

android.hardware.camera.front – потрібна апаратна камера

android.hardware.camera.autofocus – потрібна камера з автоматичним фокусуванням

У оф. документації Android можна переглянути всі параметри, ось посилання.

Можна перевизначити вимогу за умовчанням, додавши атрибут, який вимагає значення false.

Тег <supports-screens>

<supports-screens> – визначає роздільну здатність екрана, необхідну для функціонування пристрою.

Даний тег дозволяє вказати розміри екрана, для якого було створено програму. Система буде масштабувати вашу програму на основі ваших макетів на тих пристроях, які підтримують вказані вами дозволи екран.

Для інших випадків система розтягуватиме макет у міру можливості.

Значення:

smallScreen – QVGA екрани

normalScreen – стандартні екрани HVGA та WQVGA

largeScreen – великі екрани

xlargeScreen – дуже великі екрани, які перевершують розмір планшетів

anyDensity – встановіть значення true, якщо ваша програма здатна масштабуватися для відображення на екрані з будь-якою роздільною здатністю.

За промовчанням для кожного атрибута встановлено значення true. Ви можете вказати, які розміри екранів ваша програма не підтримує.

Починаючи з API 13 – Android 3, у тега з'явилися нові атрибути:

requiresSmallestWidthDp – вказуємо мінімальну підтримувану ширину екрану в апаратно-незалежних пікселях. З його допомогою можна відфільтрувати пристрої під час розміщення програми в Google

Play compatibleWidthLimitDp – задає верхню межу масштабування для вашої програми. Якщо екран пристрою виходить за вказаний кордон, система увімкне режим сумісності.

largestWidthLimitDp – задає абсолютну верхню межу, за межами якої ваша програма точно не може бути змаштабована. У цьому випадку програма запускається в режимі сумісності, яку не можна вимкнути.

Тег <compatible-screens>

<compatible-screens> – вказує конфігурацію для кожного екрана, з якою воно сумісне. Тільки один примірник <compatible-screens> елемента допускається у маніфесті, але він може містити декілька елементів <screen>.

Кожен елемент <screen> вказує конкретний розмір екрану з якою програма сумісна.

Цей елемент є інформаційним і може використовуватися зовнішні послуги (наприклад, Google Play), щоб краще зрозуміти сумісність програми з конкретними конфігураціями екрана і включити фільтрацію для користувачів.

Цей має два атрибути:

android:screenSize – вказує розмір екрану.

Значення:

small – невеликий;

normal – середній;

large – великий;

xlarge – дуже великий;

android:screen – вказуємо dpi

Значення:

ldpi – ~120dpi

mdpi – ~160dpi

hdpi – ~240dpi

xhdpi – ~320dpi

Тег <support-gl-texture>

<supports-gl-texture> – вказує одну текстуру GL стиснення, яку підтримує програма.

Тег <application>

<application> – один із головних тегів файлу маніфеста, що містить опис компонентів програми, доступних у пакеті: стилі, іконки та ін.

Містить дочірні елементи, які оголошують кожен компонент, що входять до складу програми. У маніфесті може бути лише один елемент <application>.

Нижче опис тегів, які вкладені в тег <application>.

Тег <activity>

<activity> – вказує на активність.

Якщо програма містить декілька активностей, вони мають бути оголошені в маніфесті, створюючи для кожного із них елемент <activity>. Якщо активність не буде оголошена в маніфесті,то її не буде видно системі і вона не буде запускатися при виконанні програми або виводитиметься сповіщення про її відсутність.

android:name – ім'я класу.

Ім'я має включати повне позначення пакета, але якщо ім'я пакета вже включено до <manifest>, ім'я класу, що реалізує діяльність, можна записувати у скороченому вигляді, опускаючи ім'я пакета.

android:label - текстова мітка, що відображається користувачеві.

Тег <intent-filter>

Кожен тег <activity> підтримує вкладені вузли <intent-filter>. Елемент <intent-filter> визначає типи намірів, на які може реагувати дія, послуга або приймач намірів. Фільтр намірів дозволяє

клієнтським компонентам отримувати наміри оголошеного типу, відфільтровуючи ті, які не мають відношення до компонента, і містить дочірні елементи <action>, <category>, <data>.

Тег <action>

Елемент <action> додає дії до фільтра намірів. Елемент <intentfilter> має містити один або декілька елементів <action>. Якщо в елемент <intent-filter> не матиме цих елементів, об’єкти намірів не матимуть пройти через фільтр.

Тег <category>

<category> – визначає категорію компонента, яку має опрацювати намір. Це строкові константи, визначені у класі intent

Тег <data>

<data> – додає специфікацію даних до фільтру намірів.

Специфікація може бути лише типом даних (атрибут mimeType), URI або типом даних разом із URI.

Значення URI визначається окремими атрибутами кожної з його частин, т. е. URI ділиться на частини:

android:scheme;

android:host;

android:port;

android:path або android:pathPrefix;

android:pathPattern.

Тег <meta-data>

<meta-data> - визначає пару "ім'я-значення" для елемента додаткових довільних даних, якими можна забезпечити батьківський компонент.

Складовий елемент може містити будь-яку кількість елементів <meta-data>

Тег <service>

<service> – оголошує службу як один із компонентів програми. Служби, які не були оголошені, не будуть виявлені системою та ніколи не будуть запущені.

Тег <receiver>

<receiver> – оголошує приймач широкомовних намірів як із компонентів докладання.

Приймачі широкомовних намірів дають можливість додаткам отримати наміри, передані системою чи іншими додатками, навіть коли інші компоненти програми не працюють.

Тег <provider>

Елемент <provider> повідомляє контент-провайдера (джерело даних) для керування доступом до баз даних.

Елемент <provider> містить свій набір дочірніх елементів для встановлення дозволів доступу до даних:

<grant-uri-permission>

<path-permission>

<meta-data>

Тег <uses-library>

<uses-library> – визначає загальнодоступну бібліотеку, з якою має бути скомпонована програма.

Цей елемент вказує системі необхідність включення коду бібліотеки в завантажувач класів для пакета програми. Кожен проект пов'язаний за замовчуванням з бібліотеками Android, які включають основні пакети для складання додатків.

Однак, деякі пакети знаходяться в окремих бібліотеках, які автоматично не компонуються з програмою. Якщо програма використовує пакети з цих бібліотек або інших, від сторонніх розробників, необхідно зробити явне зв'язування з цими бібліотеками і маніфест обов'язково повинен містити окремий

елемент <uses-library>.

2.**Services** - это компонент, который помогает выполнять длительные процессы, такие как обновление базы данных или сервера, запуск обратного отсчета или воспроизведение звука. По умолчанию Android Services запускаются в том же потоке, что и основные процессы приложения.

3.**Broadcast Receiver -** це механізм для надсилання та отримання повідомлень в Android. Іншими словами – це пошта, через яку ми можемо надіслати листа, а також можемо попросити цю пошту, доставляти нам листи певного змісту (начебто купили передплату на журнал).

4.**Content Provider** - це спосіб розшарити для загального користування дані вашої програми. Зазвичай це дані із БД. І створення класу провайдера схоже створення звичайного класу до роботи з БД. Ми використовуємо SQLiteOpenHelper для управління базою, і реалізуємо методи query, insert, update, delete класу ContentProvider.

1.4 Огляд медичної та фінансової статистики

Одним із головних у соціальній медицині є статистичний метод, який найчастіше і використовується у соціально – медичних та організаційних дослідженнях. Статистичним цей метод називається тому, що користується основними положеннями статистики – теорією ймовірності та законом великих чисел. Слово “статистика” латинського походження, походить від слова “status” – держава, “statista” – особа, що займається державною справою.

Статистика відноситься до суспільних наук. Вона вивчає масові суспільні явища і складені із них кількісні закономірності. Це об’єднує статистику з іншими суспільними науками – історією, політологією, економікою та іншими.

Явищам суспільного життя притаманна кількісна визначеність, яка виражається в тому, що у кожний певний момент часу має певні кількісні

характеристики, які постійно змінюються. Ця кількісна сторона масових суспільних явищ і складає предмет пізнання статистичної науки. Це має величезне явище для вирішення багатьох соціально – економічних та медичних питань, тому що достовірні дані статистики володіють великою доказовою силою.

МКХ-10 та перелік классів хвороб

Міжнаро́дна статисти́чна класифіка́ція хворо́б та пробле́м, пов'я́заних зі здоро́в'ям (англ. International Statistical Classification of Diseases and Related Health Problems) — документ, який використовується як провідна статистична та класифікаційна основа в системі Охорони здоров'я. Періодично переглядається під керівництвом ВООЗ. МКХ є нормативним документом, що забезпечує єдність методичних підходів та міжнародну верифікацію матеріалів.

Метою МКХ є створення умов для систематичної реєстрації, аналізу, інтерпретації та порівняння даних про смертність і захворюваність, отриманих у різних країнах або регіонах і в різний час. МКХ використовується для перетворення словесних формулювань діагнозів захворювань та інших проблем, пов’язаних зі здоров’ям, у коди, які полегшують зберігання, збір та аналіз даних. МКХ стала міжнародною стандартною діагностичною класифікацією для всіх загальних епідеміологічних цілей і багатьох цілей, пов’язаних з управлінням системою охорони здоров’я. Ці цілі включають аналіз загального стану здоров'я груп населення, а також розрахунок частоти та поширеності захворювань та інших проблем, пов'язаних зі здоров'ям, у зв'язку з різними факторами.

Складається МКХ[26,27] з:

1.[A00 B99] Деякі інфекційні та паразитарні хвороби

Рисунок 1.1 – Деякі інфекційні та паразитарні захворювання та їх детальні варіанти

2. [C00 D48] Новоутворення

Рисунок 1.2 – Новоутворення та їх варіанти

3. [D50 D89] Хвороби крові і кровотворних органів та окремі порушення із залученням імунного механізму

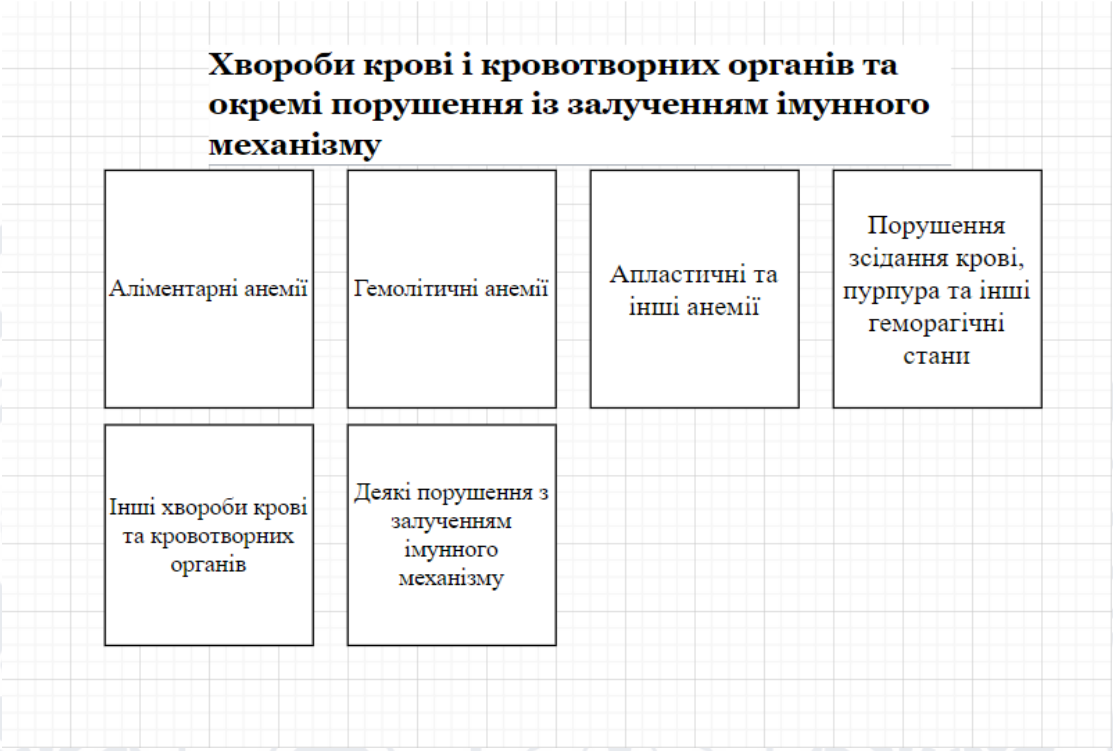

Рисунок 1.3 – Хвороби крові і кровотворних органів та окремі порушення із залученням імунного механізму та їх варіанти

4. [E00 E90] Хвороби ендокринної системи, розладу харчування та порушення обміну речовин

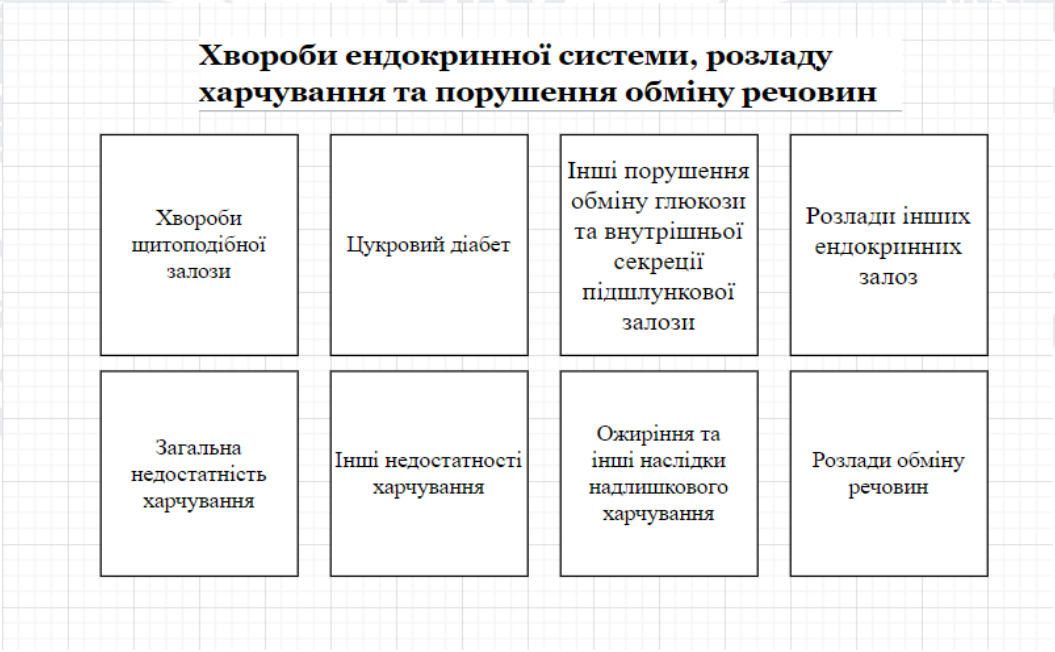

Рисунок 1.4 – Хвороби ендокринної системи

5. [F00 F99] Розлади психіки та поведінки,розладу харчування та порушення обміну речовин та їх варіант

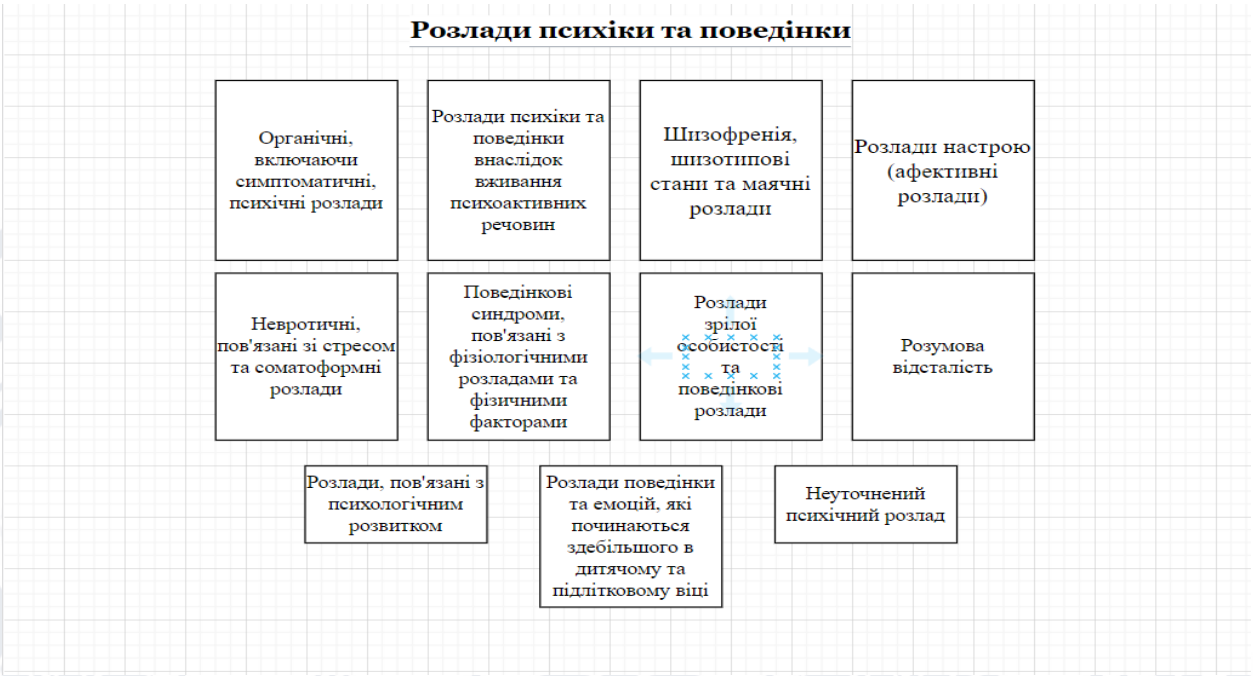

Рисунок 1.5 – Розлади психіки та поведінки і їх варіанти

6. [G00 G99] Хвороби нервової системи

Рисунок 1.6 – Хвороби нервової системи та їх варіанти

7. [H00 H59] Хвороби ока та придаткового апарату

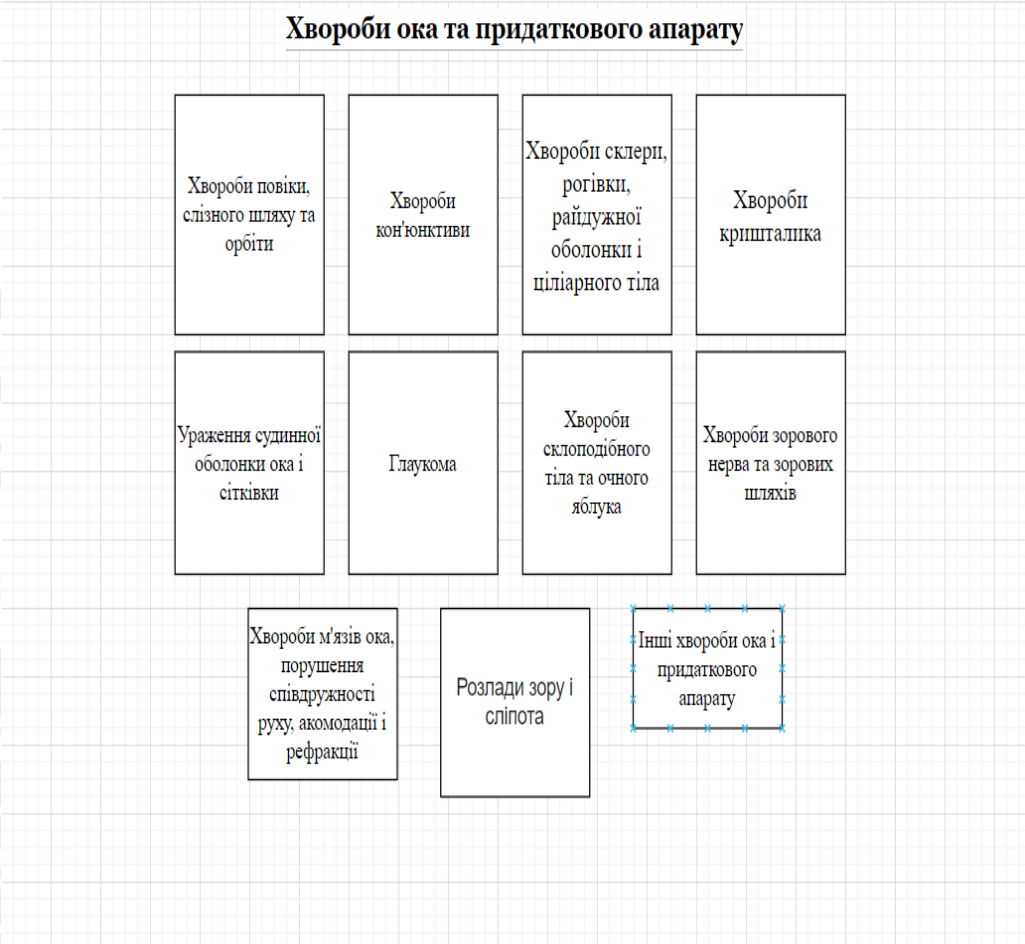

Рисунок 1.7 – Хвороби ока та придаткового апарату і їх варіанти

8. [H60 H95] Хвороби вуха та соскоподібного відростка

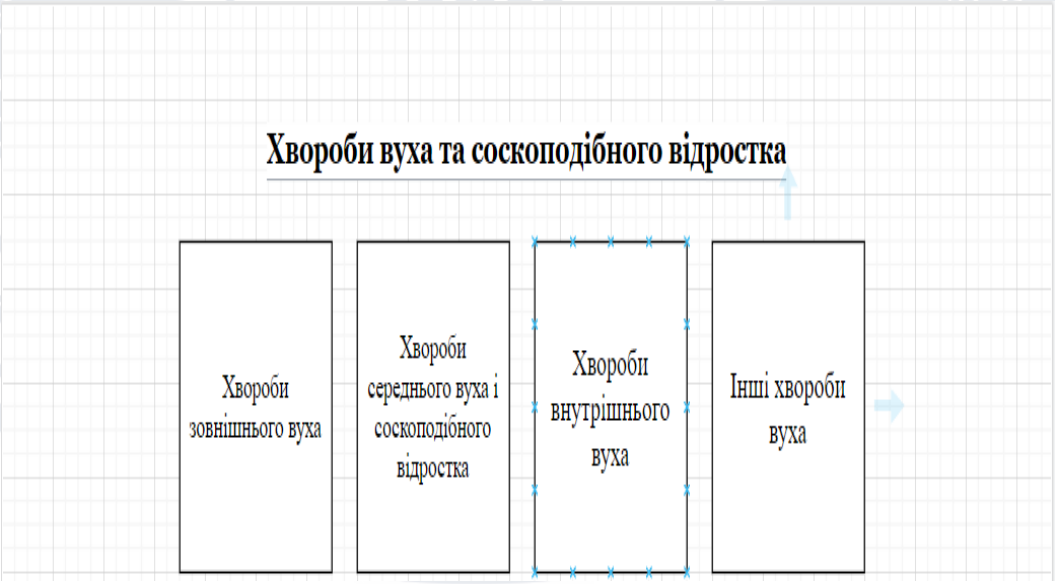

Рисунок 1.8 – Хвороби вуха та соскоподібного відростка та їх варіанти

9. [I00 I99] Хвороби системи кровообігу

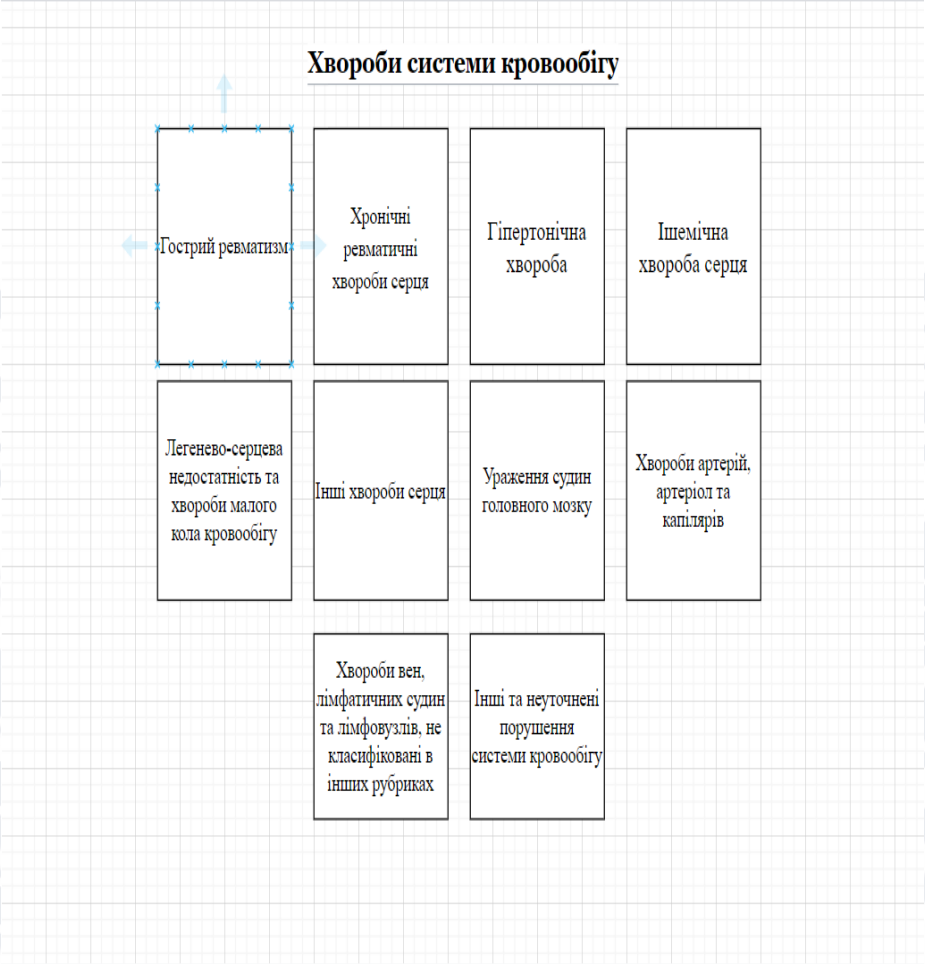

Рисунок 1.9 – Хвороби системи кровообігу та їх варіанти

10. [J00 J99] Хвороби системи дихання

Рисунок 1.10 – Хвороби системи дихання та їх варіанти

11. [K00 K93] Хвороби органів травлення

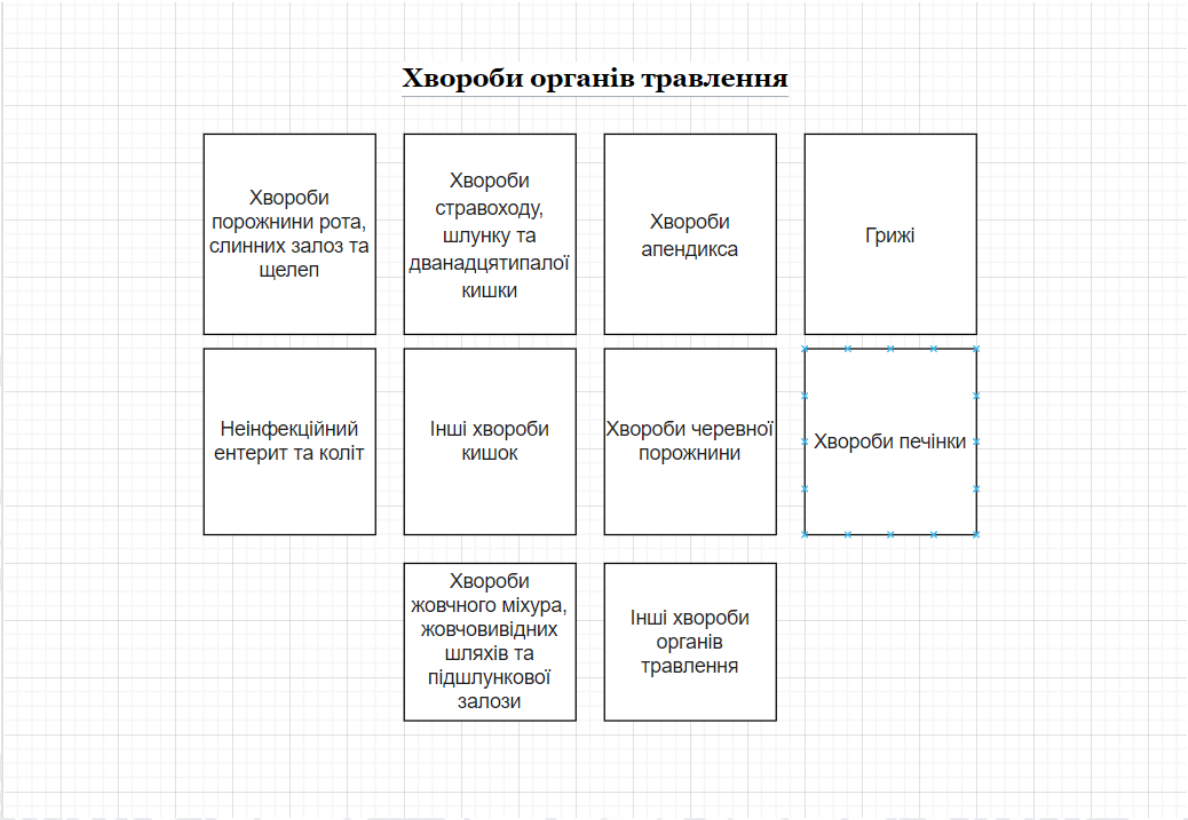

Рисунок 1.11 – Хвороби органів травлення та їх варіанти

12. [L00 L99] Хвороби шкіри та підшкірної клітковини

Рисунок 1.12 – Хвороби шкіри та підшкірної клітковини та їх варіанти

13. [M00 M99] Хвороби кістково-м'язової системи та сполучної тканини

Рисунок 1.13 – Хвороби кістково-мязової системи та сполучної тканини та їх варіанти

14. [N00 N99] Хвороби сечостатевої системи

Рисунок 1.14 – Хвороби сечостатевої системи та їх варіанти

15. [O00 O99] Вагітність, пологи та післяпологовий період

Рисунок 1.15 – Вагітність, пологи та післяпологовий період та їх варіанти

16. [P00 P96] Окремі стани, що виникають в перинатальному періоді

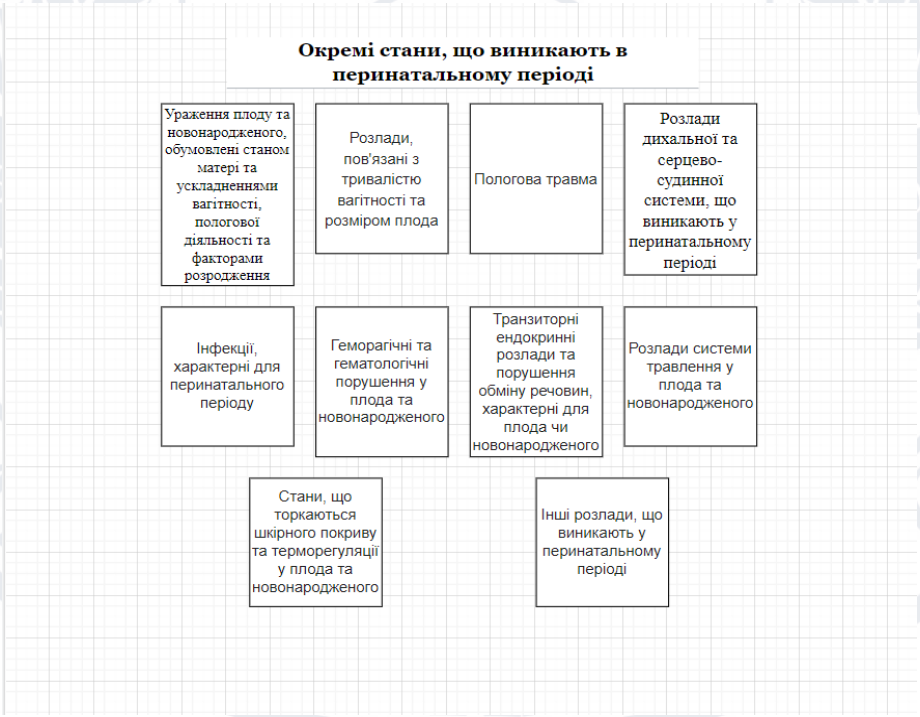

Рисунок 1.16 – Окремі стани, що виникають в перинатальному періоді та їх варіанти

17. [Q00 Q99] Вроджені вади розвитку, деформації та хромосомної аномалії

Рисунок 1.17 – Вроджені вади розвитку, деформації та хромосомної аномалії та їх варіанти

18. [R00 R99] Симптоми, ознаки та відхилення від норми, що виявлені при лабораторних та клінічних дослідженнях, не класифіковані в інших рубриках

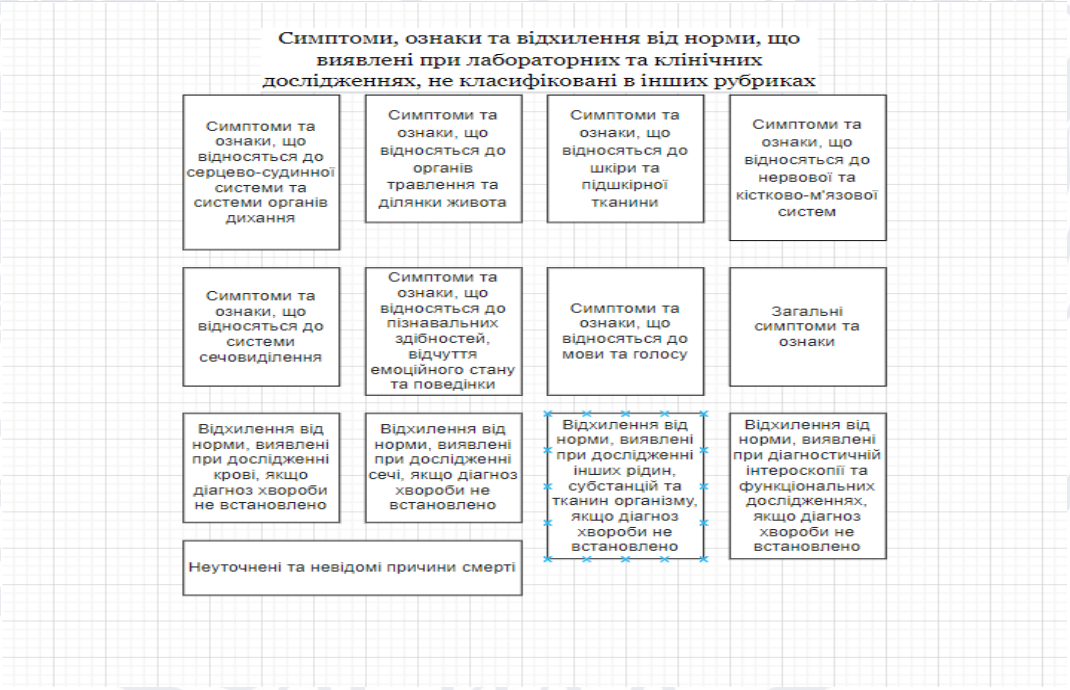

Рисунок 1.18 – Симптоми ознаки та відхилення від норми, що виявлені при лабораторних та клінічних дослідженнях, не класифіковані в інших рубриках та їх варіанти

19. [S00 T98] Травми, отруєння та деякі інші наслідки дії зовнішніх причин

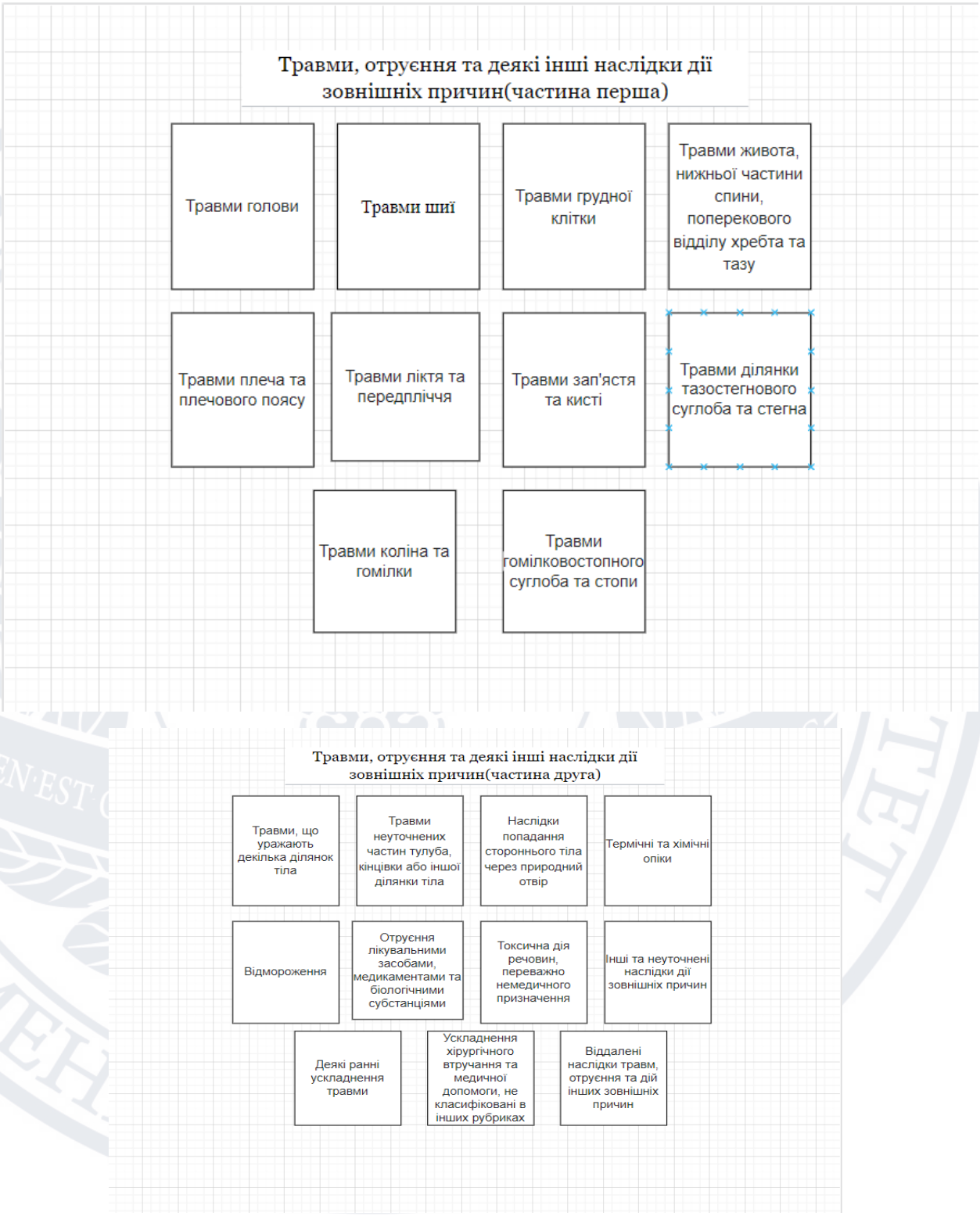

Рисунок 1.19,1.20 – Травми, отруєння та деякі інші наслідки дії зовнішніх причин та їх варіанти

20. Зовнішні причини захворюваності та смертності

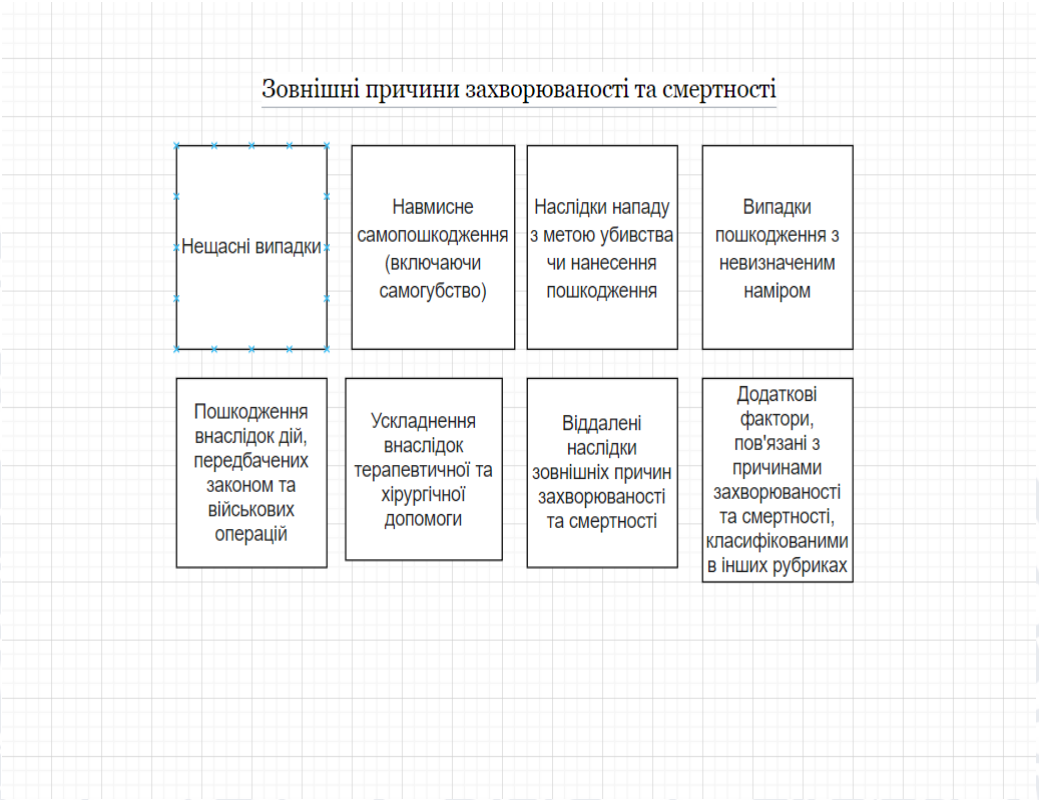

Рисунок 1.21 – Зовнішні захворюваності та смертності та їх варіанти

21. [Z00 Z99] Фактори, що впливають на стан здоров'я населення та звертання до закладів охорони здоров'я

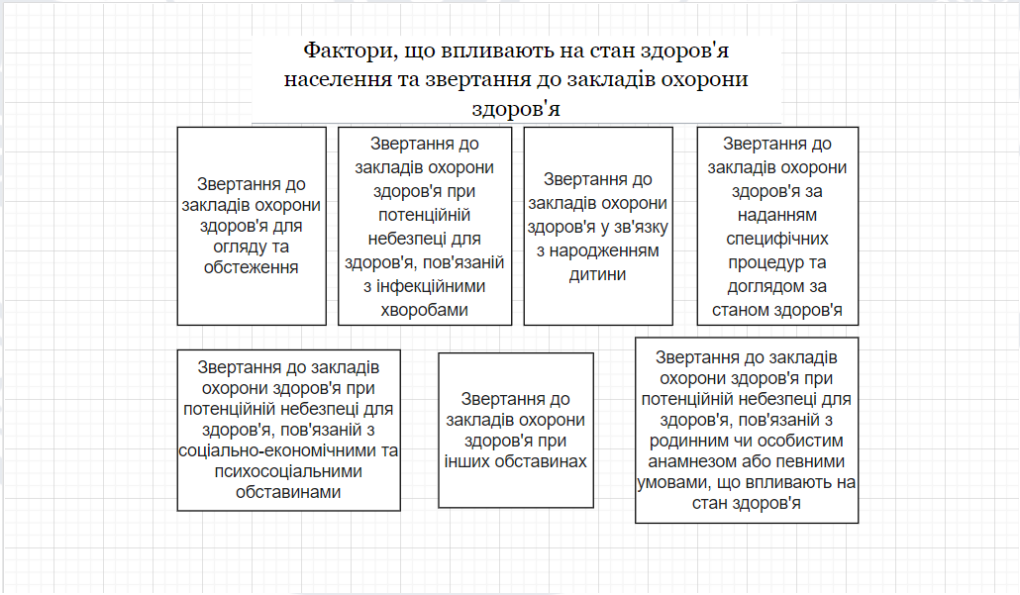

Рисунок 1.22 – Фактори, що впливають на стан здоров’я та їх варіанти

Висновки до розділу

В цьому розділі було розглянуто загальне поняття про мобільний додаток з ОС Android, з яких базових речей він складається та які можливості маємо з самого SDK.

Розглянуто медичну статистику як поняття та що вона із себе представляє. Було оглянуто МКХ-10 (Міжнародну класифікацію хвороб десятого перегляду), за якою хвороби розбиваються на категорії та використовують на разі приватні та державні лікарні.

РОЗДІЛ 2
ДЕТАЛІЗАЦІЯ ЗАВДАННЯ ТА ОБГРУНТУВАННЯ ІНСТРУМЕНТАРІЮ

2.1 **Мови програмування для розробки мобільного додатку під ОС Android**

Ключові мови програмування під ОС Android є:

Java [3] – довготривалий час була основною мовою для написання мобільних додатків. Один із найстаріших і перевірених методів створення якісної програми. Це популярна мова програмування для андроїд, яка у 2019 році увійшла до ТОП 5 найкращих та функціональних мов. Однак у той же час він досить складний, і для новачків точно не підходить. Розібратися у ньому важко, оскільки він працює на кшталт об'єктно-орієнтованого програмування. Це означає, що він працює з класами, винятками, конструкторами, які потребують ретельного моніторингу та логіки. Головною перевагою Java є наявність власного середовища розробки під назвою Android Studio. У 2014 році він був визнаний Google офіційним середовищем програмування Android, що значно полегшило життя розробникам. Процес розробки програми спрощується за рахунок візуального редактора інтерфейсу користувача, функції автозаповнення коду та інших можливостей. З точки зору функціональності Java є однією з найпотужніших мов. Він дозволяє реалізовувати найскладніші завдання та проекти, тому такий популярний. Тут використовуються як класи Java, так і файли XML. Якщо стандарти дизайну коду дотримуються, він нормально читається.

Kotlin [2] – це доволі молода мова програмування, яка була створена у 2017 році. Але за короткий проміжок часу вона змогла завоювати кохання багатьох програмістів, і в 2019 році була визнаною компанією Google найкращою мовою програмування для андроїд, відсунувши Java на друге місце. Kotlin увібрав у себе найкращі якості інших мов. Вона містить масу переваг, серед яких автоматичне визначення типів даних, функції-

розширення та інші. Він простий у розумінні, освоєнні та доступний для кожного. Створено Kotlin на основі Java, тому переходити з Java дуже просто. Мова легко інтегрується з багатьма фреймворками, проста для вивчення, її код відкритий. Завдяки зрозумілому синтаксису підвищується продуктивність при створенні додатків, а послідовність легко дотримуватись. Короткість, легка читання та лаконічність – основні якості Kotlin. Але при цьому він має недоліки. Швидкість компіляції коду нестабільна, може проходити як швидко, і зі значними затримками. Мова нова, тому не так багато навчального матеріалу, співтовариство ще не розвинене ґрунтовно, тому в ході виникнення складнощів чи питань розробникам нерідко доводиться самостійно шукати шляхи вирішення. І все ж, Kotlin по праву вважається одним із найкращих в Android програмуванні.

З моменту виходу першої офіційної версії мови у 2016 році, всього за рік він посів місце у топ-50 у рейтингу TIOBE (індекс, що оцінює популярність мов програмування на основі підрахунку результатів пошукових запитів, що містять назву мови) та не здає позицій.

Kotlin – це статично типізована мова програмування (тип змінної відомий під час компіляції, тобто ще до запуску програми).

На відміну від Java, де програми будуються на класах, основним будівельним блоком програми Kotlin є функція. Однак Kotlin також підтримує об'єктно-орієнтований підхід до програмування.

Синтаксис

На думку авторів Kotlin, Java накопичила багато невдалих рішень за роки існування. Тому в новій мові вони постаралися врахувати всі недоліки Java та інших мов програмування та зробити синтаксис мови лаконічним та зручним. Для розробників це означає, що код Kotlin простіше читається. У деяких випадках він може бути на десятки рядків коротшим, ніж, наприклад, Java.

Переваги:

1.Kotlin повністю сумісний із Java. Це означає, що можна викликати функції, оголошені в коді Java, прямо з коду Kotlin і навпаки, і в тому числі користуватися великою кількістю вже існуючих бібліотек на Java.[6]

У програмі Kotlin можна використовувати будь-які Java-фреймворки. А ще Kotlin можна інтегрувати з системами збирання, такими як Maven та Gradle.

2.Лаконічність. Це один із основних факторів, через які все більше розробників вибирає саме Kotlin.

3.Kotlin null-безпечний. Автори мови зробили все, щоб NullPointerException залишилися в минулому. Це одні з найпоширеніших винятків, і їх майже неможливо відстежити під час компіляції. А це означає, що помилки виникатимуть лише під час роботи програми.

NullPointerException виникає у разі, коли змінна, до якої намагається отримати доступ програма, дорівнює null. Тому в Kotlin змінної за замовчуванням не можна присвоїти null, тільки якщо це не вказано. Наприклад, код нижче не скомпілюється, тому що відсутня «?» після типу змінної:

val name: String = null

Щоб змінна name могла бути null, потрібно вказати це за допомогою знака «?»

4.Функції-розширення. Kotlin дозволяє змінити функціонал існуючих класів без наслідування класів. Це зручний інструмент, який дозволяє підвищити читання коду. Для оголошення extension функції до її імені потрібно додати префікс у вигляді типу, що розширюється, наприклад:

5.Kotlin має відкритий вихідний код. Код Kotlin відкритий для програмістів, і його впровадження у проект теж безплатно. Відкритий вихідний код полегшує пошук проблем. Розробники Kotlin прислухаються до сторонніх розробників і вносять правки, які пропонує співтовариство.

6.Легко вчити. Багато хто відзначає, що Kotlin простий у вивченні та підійде як мова програмування для початківців.

Недоліки:

1.Швидкість компіляції. Часто виникають проблеми зі швидкістю компіляції коду. Це не постійне явище, іноді компіляція відбувається навіть швидше, ніж код Java, але такі моменти засмучують розробників.

2.Невелика кількість туторіалів. Спільнота розробників мовою програмування Kotlin ще молода і статей з розробки не так багато, особливо для «чайників». Ось кілька корисних матеріалів російською для початківців.

Мною була вибрана мова програмування Kotlin оскільки:

- Вона підтримується Google та наразі являється мовою №1 для розробки мобільних додатків під ОС Android
- Зручніша за Java
- Існує багато матеріалів для її використання

2.2 **Розробка візуальної частини та огляд інструментів для його створення (XML або Jetpack Compose)**

XML - в перекладі з англ. eXtensible Markup Language — розширюваний мову розмітки. Служить для зберігання і передачі даних. Так що побачити його можна не тільки в API, але і в коді. Використоується для створення візуальної частини екранів [4].

Compose дозволяє робити більше з меншою кількістю коду порівняно з XML. Compose – інтуітивно зрозумілий. Це означає, що вам просто потрібно сказати Compose, що ви хочете показати користувачеві. Compose сумісний з усім наявним кодом: ви можете викликати код Compose із Views, а Views — із Compose.

Одиницею побудови інтерфейсу є функція, позначена інструкцією @Composable. Таким чином, побудова інтерфейсу полягає у композиції

таких функцій. Створення кастомних хуртовин і їх перевикористання стає набагато простіше і зручніше [5].

AndroidStudio надає зручний тулінг, який дозволяє відразу ж подивитися прев'ю розмітки, що вийшла. Для цього функція повинна бути позначена анотацією @Preview і повинна мати значення за промовчанням для всіх параметрів функції (Також можна задати дані за допомогою @PreviewParameter). @Preview має набір параметрів, які дозволяють налаштувати відображення. Наприклад: widthDp, heightDp ─ задають обмеження в'юпорту, в якому буде відрендерено прев'ю, device ─ задає розмір в'юпорту відповідно до конкретного пристрою, showSystemUi ─ відповідає за рендеринг рамки навколо розмітки з app bar, status bar та bottom bar, locale ─ задає рендерингу. Крім простого рендерингу, прев'ю є також можливість live edit літералів для рядків, Int-ів, розмірностей (Dp), квітів та Boolean. Також є інтерактивний режим, в якому можна проінспектувати роботу анімацій та поведінку UI загалом.

Мною була вибрана мова XML, оскільки наразі вона є більш стабільною та зрозумілою для мене.

2.3 **Вибір патерну проектування (MVVM vs MVP)**

Model-View-ViewModel (MVVM) — це шаблон проектування програмного забезпечення, який структуровано для розділення логіки програми та елементів керування інтерфейсом користувача. MVVM також відомий як model-view-binder і був створений архітекторами Microsoft Кеном Купером і Джоном Госсманом.

Як і багато інших шаблонів проектування, MVVM допомагає організувати код і розбивати програми на модулі, щоб зробити розробку, оновлення та повторне використання коду простішим і швидшим. Шаблон часто використовується в програмному забезпеченні Windows і веб-графічних презентацій.

Шаблон MVVM використовується в Windows Presentation Foundation (WPF), який працює в .NET Microsoft. Silverlight, еквівалентний для Інтернету мультимедійний плагін Microsoft WPF, також використовує MVVM [8].

Поділ коду в MVVM ділиться на View, ViewModel і Model:

Перегляд — це набір видимих елементів, який також отримує дані користувача. Це включає в себе інтерфейс користувача (UI), анімацію та текст. Вміст View не взаємодіє безпосередньо, щоб змінити те, що представлено.

ViewModel знаходиться між шарами View і Model. Тут розміщено елементи керування для взаємодії з View, тоді як зв'язування використовується для підключення елементів інтерфейсу користувача у View до елементів керування у ViewModel.

Модель містить логіку для програми, яку отримує ViewModel після отримання вхідних даних від користувача через View.

MVP розшифровується як Model-View-Presenter (модель-подання-презентер). Якщо розглядати Activity, що відображає якісь дані із сервера, то View - це Activity, а Model - це ваші класи роботи з сервером. Безпосередньо View і Model не взаємодіють. Для цього використається Presenter [10].

Якщо в Activity користувач натиснув кнопку Оновити, Activity повідомляє про це презентера. У цьому Activity не вимагає презентатор завантажити дані. Він просто повідомляє, що користувач натиснув кнопку Оновити. А презентер вже сам вирішує, що після натискання цієї кнопки треба робити. Він запитує дані у моделі та передає їх у Activity, щоб відобразити на екрані.

Якщо екран відображає дані з бази даних, модель - це база даних. Презентер може передплатити повідомлення моделі про оновлення. У разі

коли дані в БД зміняться, модель сповістить про це презентер. Презентер отримає ці зміни та передасть їх до Activity.

Можна сказати, що презентер – це логіка, винесена з Activity до окремого класу. А Activity залишається для відображення даних та взаємодії з користувачем. Якщо ви вирішили зробити інше Activity для відображення даних, то вам не потрібно буде переносити логіку в нове Activity, можна буде використовувати готовий Presenter. А якщо ви вирішили змінити логіку, то вам не потрібно буде лізти в Activity і там, серед коду, який відповідає за відображення даних та взаємодію з користувачем, шукати логіку та змінювати її. Ви змінюєте код у презентері.

Плюси MVP:

Коротко напишу переваги MVP у порівнянні з Activity.

- Легше писати тести

- у невеликих класах шукати щось і вносити зміни легше, ніж в одному великому

- буває так, що одна вистава використовується різними презентерами, або навпаки - один презентер використовується для різних уявлень. Якщо у вас все в одному Activity – ви не зможете так зробити.

Всі плюси випливають із того, що замість одного класу ми використовуємо кілька.

Мною був обраний патерн проектування MVVM,оскільки даний патерн є практичним та найбільш використовуваним для розробки мобільних додатків.

2.4 **Огляд бібліотек для Dependency Injection (використання залежностей)**

Для реалізації цього патерну існує три бібліотеки:

- Koin
- Dagger-Hilt

- Kodein

Koin - Концепція області Koin аналогічна такої в Android. Вона дозволяє, наприклад, обмежити область живучості моделі подання (ViewModel) до певної дії та використовувати цю модель у фрагментах, які заповнюють дію. Як правило, у Koin є три типи тимчасових областей. single (single object) - створюється об'єкт, який зберігається протягом усього періоду існування контейнера (подібно до singleton); factory (фабрика об'єктів) - кожен раз створюється новий об'єкт, без збереження в контейнері (неможливий спільний доступ); scoped (об’єкт області) — створюється об’єкт, який зберігається протягом періоду існування пов’язаної області часу. Поле типу single повертає один і той самий екземпляр на кожен запит, тоді як фабрика повертає новий екземпляр щоразу.

Налаштована область

Стандартні області single та factory у Koin живуть протягом життєвого циклу модулів Koin. Однак у реальних сценаріях використання вимог до впровадження залежностей будуть відрізнятися.

Залежності зазвичай потрібні лише певний період. Наприклад, репозиторій OnBoardRepository в Android-додатку потрібно лише під час реєстрації користувача. Як користувач авторизується, утримання цього репозиторію в пам'яті стане лишньою витратою ресурсів.

Щоб досягти потрібної поведінки в Koin, можна скористатися API для роботи з тимчасовими областями. У модулі Koin можна створити область зі рядковим кваліфікатором і оголошувати залежності в ній за допомогою унікальних кваліфікаторів.

Dagger 2[12] - це повністю статичний фреймворк для впровадження залежностей у Java та Android, що працює під час компіляції. Dagger 2 – це адаптація створеного раніше компанією Square фреймворку Dagger, яку підтримує компанія Google.

Плюси dagger:

- Доводиться писати менше шаблонного коду.
- Допомагає структурувати залежність.
- Сильно спрощує роботу коли багато залежностей
- Код стає простим для читання

Мінуси dagger:

- Відсутність докладної документації
- Dagger намагається зрозуміти наміри розробника за допомогою анотацій. Це ускладнюється, коли він вас неправильно розуміє

dagger генерує код, помилки в яких так само складно визначити

Kodein - Розробники Kodein поділяють два способи отримання залежностей - injection та retrieval. Якщо коротко, то injection це коли клас отримує всі залежності при створенні, тобто в конструктор, а retrieval це коли клас сам відповідає за отримання своїх залежностей [11].

При використанні injection ваш клас нічого не знає про Kodein і код у класі виходить чистішим, але якщо використовувати retrieval, то у вас з'являється можливість гнучкіше керувати залежностями. У випадку з retrieval всі залежності виходять ліниво, тільки в момент першого звернення до залежності.

Методи Kodein, за допомогою яких можна отримати залежність

Примірник класу Kodein має три методи, які повернуть вам залежність, фабрику залежностей або провайдер залежностей — це instance(), factory() і provider() відповідно. Таким чином, якщо ви надаєте залежність за допомогою factory або provider, то й одержувати ви можете не лише результат виконання функції, але й саму функцію. Не забувайте, що завжди можна використовувати теги.

Мною була вибрана бібліотека Koin,оскільки в даному проекті не буде великої кількості екранів.

2.5 **Огляд бібліотек для використання багатопоточності (multithreading)**

Для реалізації багатопоточності існує дві основні бібліотеки:

- RxJava(RxKotlin)
- Coroutine

RxJava - це фреймворк від ReactiveX (RX) для реактивного програмування на Java — підходу, у якому у відповідь зміни одних сутностей автоматично змінюються інші. Він використовується в мобільній та веб-розробці та допомагає реалізувати асинхронність – почергове виконання дій.

RxJava використовують Android-розробники, розробники кросплатформових додатків, веб-розробники на Java та Kotlin:

створення додатків у парадигмі реактивного програмування;

швидкого виконання дій, які мають відбуватися за зміни певних параметрів;

програмування реакції на події, що відбуваються внаслідок дій користувача чи системи;

реалізації відображення відомостей на основі взаємодії із програмою;

обробки множинних потоків даних та програмування відповідей на них;

більшої читальності та керованості коду;

рятування від великої кількості коллбеків - функцій, які запускаються у відповідь на дію.

RxJava реалізує підхід, який називають реактивним програмуванням. Воно передбачає, що програміст працює із потоком даних: весь стан сутностей у коді залежить від цього потоку. Коли змінюється одна змінна або об'єкт, повинні змінитись і пов'язані з ними функції.

Реактивне програмування пояснюють з прикладу Excel-таблиць. Якщо ввести в таблицю формулу, вона виконається на основі даних з

потрібних осередків. А якщо змінити інформацію в цих осередках, формула автоматично перерахується. Також поводяться сутності в коді при реактивному підході. Він вважається компромісом між об'єктно-орієнтованим та функціональним програмуванням.

Принципи роботи фреймворку:

1.Опора на функціональне програмування

RxJava натхненна ідеєю функціонального програмування, де код представлений як набір функцій. Наразі його місцями витісняє ОВП, об'єктно-орієнтований підхід, але функціональні рішення в деяких випадках оптимальніші.

2.Компонування

Оператори RxJava можна компонувати і поєднувати один з одним, в результаті програмування за допомогою фреймворку стає більш гнучким.

Інтуїтивний підхід

Фреймворк написаний таким чином, щоб його поведінка була схожа на інші «навколофункціональні» рішення. Так розробнику простіше розуміти, чого очікувати від тієї чи іншої можливості та як можна вирішити завдання.

3.Розширюваність

У RxJava можна додавати оператори користувача, розширюючи її функціональність.

4.Лаконічність

Фреймворк побудований те щоб складні завдання реакцію події можна було запрограмувати кількома рядками коду.

Ключові компоненти RxJava

Ідея фреймворку — представлення коду у вигляді моделі «Спостережуване та спостерігач». У ній є дві основні сутності, які називаються Observable та Observer. Вони взаємодіють між собою, так обробляються події.

Observable - це "спостерігається". Так називається потік інформації, джерело даних та подій. Коли в Observable змінюються дані, дізнається про це Observer.

Observer - це "спостерігач", обробник подій. Він спостерігає за потоком даних і виконує дії, як ті змінилися. Observable посилає йому сигнал, коли починає чи закінчує видавати інформацію.

Сутність коду, написаного на RxJava, - це спостерігачі і спостерігається. При цьому функція, яка спостерігає за іншою, сама може бути спостерігається для чогось ще.

Дані можна перетворювати, фільтрувати, з'єднувати між собою або розділяти. За допомогою функцій фреймворку можна створювати та видаляти спостережуване, пов'язувати з різними спостерігачами або відкріплювати від них.

Переваги RxJava:

1.Кросплатформеність

Java та її фреймворки за визначенням кросплатформенними, тобто підходять для розробки під будь-які пристрої та операційні системи. Це зручно, тому що для портування проекту на іншу ОС не потрібно радикально переписувати код. Це слушно і для RxJava.

2.Зручний поділ завдань

Щоб код був читаним та зрозумілим, потрібно розділяти різні сутності, не змішувати їх. Реактивне програмування - один із способів такого поділу. Завдання розбиваються більш дрібні, виконання кожної їх є подією. Ці події опрацьовує спостерігач. RxJava допомагає зробити код чистим та гнучким.

3.Відсутність множинних коллбеків

У мобільній розробці альтернатива RxJava – коллбеки. Так називають функції, що запускаються, коли інший блок коду завершує роботу. Перед запуском передається результат виконання цього блоку. Коллбеки - ще один

спосіб реалізувати асинхронність, виконання завдань по черзі один за одним.

Множинні коллбеки в одному блоці коду призводять до явища, яке називається callback hell - "пекло зворотних викликів". Код стає слабко читаним через зайву вкладеність і велику кількість дужок.

Проект ReactiveX і, зокрема, RxJava - спосіб уникнути «пекла зворотних викликів». Завдяки моделі спостерігача та спостерігача користуватися коллбеками не обов'язково. Реалізація асинхронності стає зрозумілішою.

4.Ефективне масштабування

Проекти, написані на RxJava, зручно трансформувати та масштабувати, особливо горизонтально. Для цього не потрібно додавати до коду нові системні потоки даних, які працюють на рівні ОС і помітно витрачають пам'ять. Натомість можна створити більше подій, що «спостерігаються», і обробників для них. Це не так навантажує систему як додавання системних потоків. Код стає ефективнішим [17].

5.Зручне оброблення помилок

Можливості RxJava спрощують обробку помилок та винятків. Не потрібно писати вручну обробники помилок. Побудова програм в RxJava така, що передавати помилки легше, ніж у ряді інших рішень.

Зменшення кількості змінних [17].

Щоб реалізувати ті ж дії в іншій парадигмі, потрібні спеціальні змінні стани і їх у коді може бути дуже багато. RxJava дозволяє передавати дані без великої кількості змінних, тому що події представляються як потік інформації. Код виглядає ясніше і чистіше, і в ньому менше «слабких місць»: велика кількість змінних станів може призводити до помилок.

Наочна реалізація асинхронності

Підхід, реалізований RxJava, наочно показує, як мають працювати асинхронні обчислення. Це дозволяє розробнику краще орієнтуватися у коді.

Недоліки RxJava

Складність у освоєнні

Концепт реактивного програмування складний тим, хто починав вивчати розробку з простіших підходів. Для початку роботи з RxJava потрібні хороша підготовка та теоретична база. Навіть професійним розробникам може знадобитися донавчання.

Неуніверсальність

RxJava може використовувати більше пам'яті, ніж інші методи. Тому його застосування не завжди актуальне. Одні розробники користуються фреймворком контролю асинхронності, інші вважають, що це занадто потужний і складний інструмент таких завдань. RxJava потрібний насамперед для реактивного програмування. У багатьох випадках краще використовувати простіші методи, наприклад Executors.

Велика кількість методів

У RxJava багато сутностей та методів, які розповзаються по всьому коду. Через це підтримувати написану програму може бути складно.

Співпрограми або копрограми — це блоки коду, які працюють асинхронно, тобто чергуються. У відповідний час такий блок припинить виконання та збереже всі свої властивості, щоб дозволити виконувати інший код. Продовжується, коли контроль повертається до першого блоку. У результаті програма виконує кілька функцій одночасно [16]. Для чого потрібні співпрограми? Створюйте асинхронні програми, які можуть виконувати кілька дій одночасно. Для гнучкої та зручної реалізації багатозадачності. Дає вам більше контролю під час перемикання між різними завданнями. Співпрограми контролюються розробниками та

програмами, а не операційною системою. Розвантажити апаратні ресурси пристрою. Як працює спільна програма

Підпрограми — це потоки виконання коду, організовані системними потоками. Потік виконання коду — це послідовність операцій, які виконуються одна за одною. Він працюватиме до часу, зазначеного в коді або визначеного розробником. Потім можна розпочати частину іншого набору операцій. Системний потік містить інструкції процесора, і одне з його ядер чергується з іншим системним потоком. Корутини працюють на більш високому рівні. Кілька спільних програм можуть послідовно виконувати код в одному системному потоці. Для кінцевого користувача співпрограмна робота виглядає як кілька дій, що виконуються паралельно.

Принципи співпрограм:

Ви можете мати кілька точок входу та виходу. Я багато разів зупинявся і починав знову. Вони працюють за стратегією LIFO - перша викликана програма завершується першою. Різниця між співпрограмами та потоками Багатозадачність із співпрограмами часто плутають із багатопоточностю. Це запуск програми в кількох системних потоках. Потік — це компонент процесу, що виконується в операційній системі. Принцип аналогічний: Зупинка та відновлення кількох потоків, очікування один одного, спілкування. Але є різниця. Потоками керує операційна система. Перемикаючи співпрограми - розробники використовують код. Перемикання потоків важко контролювати. Співпрограми керуються більш гнучко. Потоки займають багато ресурсів процесора. Доводиться постійно перемикатися між потоками. Співпрограми не вимагають перемикання контексту, тому код споживає мало ресурсів. Потоки виконуються на апаратному або системному рівні. Корутини — це рішення високого рівня. Це означає, що він далекий від системних і апаратних ресурсів, але ближчий до людського уявлення. Потоки прискорюють складні завдання, але споживають багато ресурсів. Співпрограми не прискорюють роботу, але

вони допомагають оптимізувати використання. Співпрограма виконується в одному потоці або пулі потоків.

Переваги використання

Ефективність

Багатозадачна програма ефективніше витрачає ресурси. Основний код не блокується, щоб виконати допоміжний модуль. Натомість вони працюють асинхронно.

Зручність для користувача

Корутини швидко перемикаються, для користувача це виглядає як одночасне виконання завдань. Йому не доводиться довго чекати на відповідь програми. Він продовжує працювати з нею, доки асинхронні корутини непомітно для нього виконують додаткові дії.

Зниження навантаження на систему

Асинхронність дозволяє виконувати кілька дій у межах одного потоку замість того, щоб множити потоки. Системі простіше виконати один потік ніж кілька. Розробники з JetBrains прояснили це на такому прикладі: запустити 10 тисяч корутин одночасно завдання, з яким пристрій впорається. А 10 тисяч потоків – неможливе явище: комп'ютеру не вистачить пам'яті.

Гнучкість в управлінні

Перемикання між корутинами відбувається вручну. Програміст сам прописує цей момент у коді. Тому керування корутинами можна контролювати, що дає розробнику більше можливостей. Немає ризику, що програма переключиться між блоками коду в невідповідний момент, як це буває з менш гнучкими рішеннями.

Недоліки співпрограм

Складність та високий поріг входження

Розібратися в роботі корутин та навчитися писати асинхронний код може бути складно. Тому співпрограми починають вивчати фахівці, які добре розібралися в базових принципах обраної мови.

Вузька спеціалізація

Корутини – високорівневе рішення. Прискорити складні обчислення за допомогою співпрограм не вдасться. І тут потрібна багатопоточність, а чи не асинхронність. У цьому корутини підходять зниження навантаження на систему.

Відсутність у низці мов

Корутини важко реалізувати у мовах, які їх не підтримують. Для цього потрібно використовувати сторонні рішення або перетворювати код однією мовою на іншу. Це ускладнює розробку.

Мною була вибрана бібліотека Coroutine,оскільки вона є більш сучасним рішенням та наразі у нових проектах використовують саме її.

2.6 Огляд бібліотеки для використання бази даних

Room - це новий спосіб зберегти дані додатків в Android-програмі, представлений цього року на Google I/O. Це частина нової архітектури Android, групи бібліотек Google, які підтримують правильну архітектуру програм. Кімната пропонується як альтернатива Realm, ORMLite, GreenDao тощо. [1]

Room — це інтерфейс високого рівня для вбудованих низькорівневих прив’язок SQLite Android. Подробиці дивіться в документації. Більшість роботи виконується під час компіляції, а API побудовано на основі вбудованого API SQLite, тому вам не доведеться мати справу з Cursors або ContentResolver. [1]

2.7 Огляд системи контролю версії (GIT, GitHub)

Git (читається як "гіт") - це система контролю версій, яка допомагає відстежувати історію змін у файлів. Git використовують програмісти для спільної роботи з проектами.[31]

Git - система контролю версій

У найпростішому вигляді контроль версій – це збереження на комп'ютері серії змінених файлів, наприклад, з різними датами в назві, або режим відстеження виправлень у текстових документах.

Розробникам часто буває потрібно повернутися до попередньої версії коду:

якщо виявляється, що завдання, що вирішується, більше не актуальна;

якщо потрібно внести виправлення до більш ранньої версії програми;

якщо помилка знайшлася під час роботи над новим завданням.

Якщо над проектом працює багато людей, потрібно, щоб вони могли вносити зміни до тих самих файлів без конфліктів і втрат коду. Всі ці завдання зручно вирішуються за допомогою Git.

До базових можливостей Git відносяться:

повернення до будь-якої попередньої версії коду;

перегляд історії змін;

паралельна робота над проектом;

backup коду.

Що таке репозиторій Git?[31]

Репозиторій – це всі файли, що знаходяться під контролем версій, разом з історією їхньої зміни та іншою службовою інформацією.

Репозиторій Git можна створити або вибравши будь-яку папку на комп'ютері, або клонувавши собі вже існуючий репозиторій, наприклад у роботодавця.

Де зберігається репозиторій?

Існують різні способи зберігання та використання репозиторію: виділяють локальні, централізовані та розподілені системи контролю версій.

У локальних системах контролю версій репозиторій зберігається та використовується на одному пристрої, але працювати з такою системою може лише один розробник. У разі централізованої системи репозиторій зберігається одному сервері.

Найкраще для великої кількості розробників підходять розподілені системи контролю версій, до яких і Git. Така система є хмарним сховищем: кожен користувач зберігає на своєму пристрої весь репозиторій повністю, і в міру зміни репозиторії синхронізуються.

Що таке коміт і комітити?

Англійською commit означає «фіксувати». Git-комміт – це операція, яка бере всі підготовлені зміни та відправляє їх до репозиторію як єдине ціле.

Навіщо потрібен коміт, якщо Git і так слідкує за всіма змінами? Коммиты розбивають процес розробки, що з великої кількості правок, окремі кроки. Тобто коміт — це певна логічно завершена зміна всередині проекту та зрозуміла (у тому числі й іншим розробникам) точка, до якої можна повернутися, якщо виникнуть якісь проблеми.

Зміни в рамках одного комміту підпорядковуються певним, встановленим командою розробників правилам та рекомендаціям щодо іменування, опису та змісту коммітів.

Як правило, робочий процес є циклом: комміт — зміна файлів — комміт.

Що таке розгалуження?

Зручна підтримка розгалуження – важлива властивість Git. Використання розгалуження дозволяє вирішувати окремі завдання, не втручаючись у основну лінію розробки.

Гілка в Git – це послідовність коммітів. З технічного погляду гілка — це покажчик чи посилання останній комит у цій гілці. За промовчанням, ім'я основної гілки в Git — master. Щоразу, коли створюється новий коміт, покажчик гілки master автоматично пересувається нею.

Під час створення нової гілки комміту дається новий покажчик, наприклад testing. Якщо перейти на гілку testing і зробити новий коміт, то покажчик на гілку testing переміститься вперед, тоді як покажчик на основну гілку master залишиться на місці. Переключившись назад на гілку master, файли в робочому каталозі повернуться до стану комміта, на який вказує master.

GitHub - це онлайн-сервіс для командної розробки та хостингу проектів. За допомогою GitHub над кодом проекту може працювати необмежену кількість програмістів із будь-яких точок світу. У GitHub є система контролю (управління) версій Git: сервіс дозволяє переглядати та контролювати будь-які зміни коду будь-яким розробником та повертатися до стану до змін.

В цілому GitHub - це соціальна мережа для розробників, в якій можна знайти проекти з відкритим кодом від інших розробників, практикуватися в написанні коду та зберігати своє портфоліо.

Проекти у GitHub

Проект в GitHub зберігається в репозиторії (repository) - колекції всіх змін коду, що створюється. Якщо ви працюватимете над проектом поодинці — вам потрібно створити новий репозиторій. Якщо у вашому проекті кілька розробників — кожен із них клонуватиме репозиторій початкового творця проекту.

Усередині репозиторію зміни коду зберігаються у вигляді гілок та комітів.

Коміт (commit) — основний об'єкт розробки, у якому зберігаються зміни коду за итерацию. По суті, це список з усіма актуальними змінами та

посилання на попередню версію комміту. Кожен комміт має атрибути: ім'я, дату створення, автор та коментарі до поточної версії

Гілка (branch) - указівник на коміт із певними змінами. Наприклад, два розробники взяли коміт, і кожен з них зробив свої зміни в коді, створивши за новим комітом («Створивши сторінку courses.html з особистим кабінетом» та «Створивши сторінку зі вільним доступом на курси»). Так, у проекті з'явилися дві гілки з різним кодом: розробник може вибрати, над яким коммітом йому працювати далі.

Основною гілкою проекту, як правило, вважається гілка main або master – розробники створюють нові гілки на її основі. Також можна створити необмежену кількість гілок, щоб вносити нові зміни, не заважаючи основному проекту.

Злиття гілок

Часто розробники роблять паралельні зміни коду. Наприклад, один розробник працює над зовнішнім виглядом сайту, а інший займається розміщенням контенту на ньому. Після закінчення роботи гілки шкірного з них можна об'єднати в одну, щоб створити комітт зі всіма внесеними розробниками змінами.

Для цього Git використовують функцію pull request (pr). Pull request – це заявка на злиття коду з різних гілок. У процесі злиття Git створить коміт і покаже всі зміни у файлі коду: додані до розгалуження рядки підсвічуються зеленим кольором, видалені червоним. Так кожен із розробників та менеджер проекту побачитимуть, що сталося з кодом після спільної роботи над коммітом. Перед залишковим злиттям (merge) усі розробники повинні переглянути зміни коду (code review) та прийняти їх.

Процес pull request

Тепер подивимося на процес із боку власника проекту, який отримав новий pull request. Власнику потрібно його обробити та об'єднати гілку sme-review із master.

Git зазвичай лише додає дані [31] . Майже все, що ви робите в Git, лише додає дані до бази даних Git. Важко змусити систему зробити щось незворотне або будь-яким способом стерти дані.Як і в будь-якому VCS, ви можете втратити або зіпсувати зміни, які ви ще не внесли, але після того, як ви внесли знімок у Git, це дуже важко втратити, особливо якщо ви регулярно переміщуєте свою базу даних в інше сховище.

Це робить використання Git радістю, оскільки ми знаємо, що можемо експериментувати, не ризикуючи серйозно зіпсувати ситуацію. Щоб отримати докладнішу інформацію про те, як Git зберігає свої дані та як ви можете відновити дані, які здаються втраченими, дивіться Скасування речей.

Висновки до розділу

Розглянувши усі частини створення мобільного додатку з ОС Android був зібраний такий стек:

- Мова програмування – Kotlin
- DI – Koin
- Багатопоточність – Kotlin Coroutine
- Патерн проектування – MVVM(Model – View – ViewModel)
- Візуальна частина – XML
- База даних – Room
- Система контролю версії – GIT,GitHub

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ТА ОБРОБКА ДАНИХ ПАЦІЄНТІВ

3.1. Розробка макету та прототипу мобільного додатку

Проаналізувавши технологію Android-розробки, підібрав сучасний стек технологій можемо приступати до реалізації робочого прототипу.

Даний додаток складається з:

- Нижньої навігації (Пошук,Історія,Статистика)
- Екрану медичної статистики
- Екрану фінансової статистики
- Екрану пошуку картки пацієнта
- Історія відвідування клініки пацієнтом
- Медична картка пацієнта
- Екран зі списком карток пацієнтів

Розпочнемо з нижньої навігації:

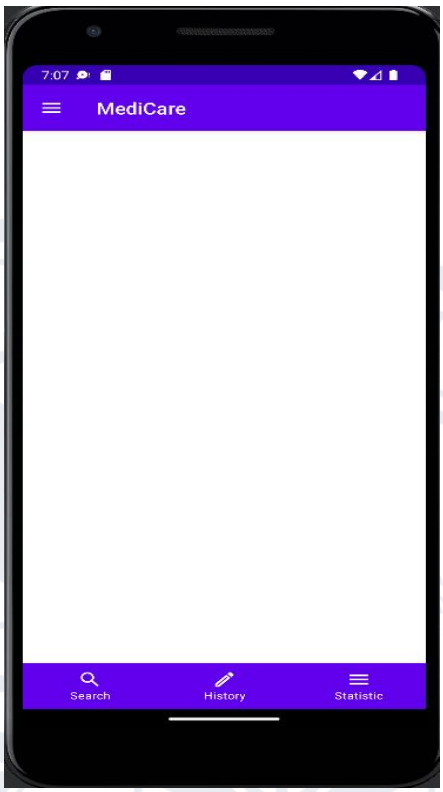

Рисунок 3.1– Екран з нижньою навігацією

Даний екран дає нам можливість переміщатися по додатку.Є три варіанти:

- Перехід на екран пошуку пацієнта
- Історія відвідування
- Огляд статистики

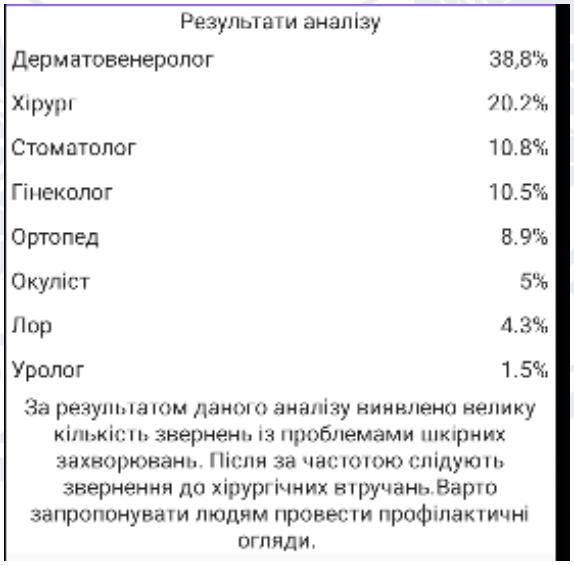

Рисунок 3.2 – Екран медичної статистики

На цьому екрані відображено приклад роботи додатку з загальною інформацією відвідувань за рік до лікарів даних спеціальностей. При зміні відсоткової статистики буде відображуватися різне повідомлення з порадами для проведення профілактичних оглядів у клініці.

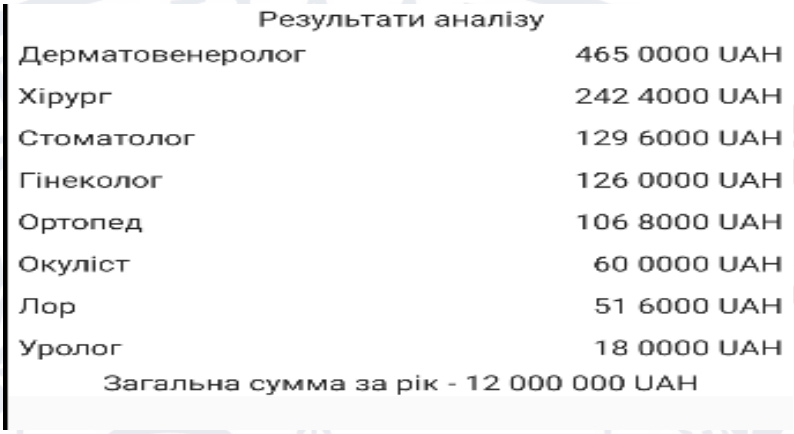

Рисунок 3.3 – Екран фінансової статистики за рік

Тут відображається приклад аналізу фінансової статистика за рік роботи клініки.

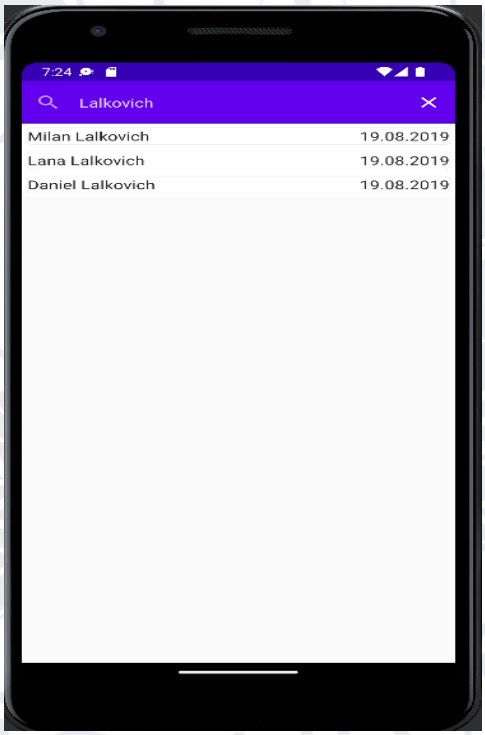

Рисунок 3.4 – Екран пошуку пацієнта

Звичайний пошук пацієнта. Відображає ФІО пацієнта та дату реєстрації. При наявності родичів пошук відобразить і їх.

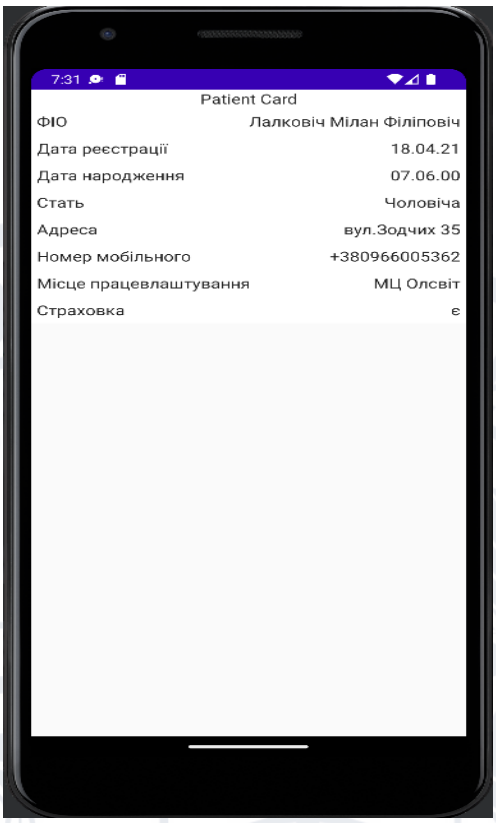

Рисунок 3.5 – Екран картки пацієнта

На цьому екрані ми бачимо повну картку пацієнта, яка складається з:

- ФІО (Фамілія, Ім’я, По-батькові)
- Дата реєстрації
- Дата народження
- Стать
- Адреса проживання
- Номер мобільного
- Місце працевлаштування
- Наявність страховки

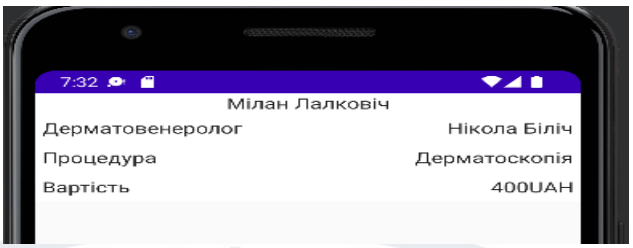

Рисунок 3.6 – Картка відвідування процедури пацієнтом

Рисунок 3.7 – Список відвідувань пацієнтів

На даному екрані відображено список записів пацієнтів, прибув чи ні, та вартість наданої клінікою послуги.

Приклад та порівняння з аналогічним додатком

Для порівняння буде представлений продукт під назвою “Доктор Елекс”.

“Доктор Елекс” - це комплексне рішення, що дозволяє оптимізувати процес лікування, змінюючи уявлення лікарів та пацієнтів про якість медичного обслуговування. [32]

Це сучасне рішення, яке:

- Забезпечить гармонійну співпрацю всього персоналу
- Впровадить контроль за процесом лікування
- Дозволить швидко знаходити потрібну інформацію
- Підвищить якість обслуговування Ваших пацієнтів [32]

Переваги для клініки:

- Новий рівень підготовки до візиту та огляду пацієнтів
- Спрощена процедура розрахунків за надані послуги
- Зменшення понаднормової витрати матеріалів медичної картки пацієнта[32]

Приклад роботи додатку

Даний додаток має можливість демонструвати всі відвідування певного пацієнта, витрати за всі відвідування та за останній сеанс.

На екрані є список усіх процедур пацієнта клініки та інші дані. Також є вибір картки пацієнта та календарний розклад з записами по годинах. У додатку є можливість вибирати пацієнта за пошуком, візитом чи документом. Також є календар, в якому відображаються ФІО пацієнтів та час на який вони записані.

Кольори означають:

- Зелений – пацієнт прибув та сплатив
- Жовтий – пацієнт не прибув
- Червоний – пацієнта приймають або ще не сплатив

Картка прийнятого пацієнта та записи лікаря. Лікар має можливість у додатку заповнювати картку за певним шаблоном.

В шаблоні консультації дерматолога є такі поля як:

- Пацієнт (Його ФІО)
- Стать
- Дата народження
- Дата обстеження
- Скарги

- Об’єктивний статус
- Діагноз

Та інші дані, які стосуються конкретної специфіки лікаря до якого зверталися. Ці картки створюються за шаблоном, які створюють самі лікарі у додатку. Далі буде наведено перелік та приклад шаблонів.

Класифікації лікарів та їх шаблони. Екран відображає перелік із спеціальностей та процедур, для яких були створені шаблони.

Переваги цього додатку у можливості створення шаблонів на будь-який смак, зручна система календарю, наявність статистики за весь час для окремого пацієнта. Мінусу цього додатку у його візуальній частині та відсутності можливості для отримання загальної статистики для лікаря за різні квартали.

Перевагою своєї роботи можу назвати реалізацію аналізу медичної та фінансової статистики.

Мінусами є дуже вузький функціонал у порівняні з “Доктором Елекс”.

Моя робота є прототипом, який можна довести до схожої та навіть кращої системи для приватної клініки.

Висновки до розділу

Було продемонстровано роботу додатку, його дизайн та функціонал. Також продемонстровано аналіз даних пацієнтів приватної клініки, результатом якої стала рекомендація для запуску профілактичних оглядів відштовхуючись від найбільш відвідуваних напрямів лікарів. Продемонстрована і фінансова статистика за один рік та скільки кожна спеціальність заробила грошей.

ВИСНОВКИ

В результаті виконання цієї роботи було проведено аналіз медичної та фінансової статистики приватної клініки, були оглянуті різновиди підходів до розробки мобільного додатку під операційну систему Android.

Була розглянута структура мобільного додатку, з чого він складається, які методи та теги існують та для чого. Розглянута медична класифікація хвороб та їх варіанти для кращого розуміння як лікарі помічають хвороби своїх пацієнтів.

Було оглянуто розроблений мобільний додаток та візуалізація результатів, маючи які та проаналізувавши їх можна:

- Закликати людей для профілактичного походу до найбільш відвідуваних за статистикою або найменш відвідуваних, оскільки людям властиво не звертати уваги на малі зміни, але які з часом можуть призвести до більш фатальних наслідків;
- Відштовхуючись від річної фінансової статистики можна зробити висновки кому варто підняти зарплатню;
- Передача медичної статистики до Міністерства Охорони Здоров’я (МОЗ). Відштовхуючись від наданої інформації з приводу підвищення кількості певних захворювань МОЗ може надавати певні рекомендації з приводу їх лікування та профілактики і реабілітації для лікарів загального профілю;

СПИСОК ЛІТЕРАТУРИ

1. Android Developers,Room,Url: https://developer.android.com/training/data-storage/room (08.10.22)
2. Wikipedia,Kotlin,Url: https://uk.wikipedia.org/wiki/Kotlin (22.09.22)
3. Programiz,About Java,Url: https://www.programiz.com/java-programming (22.09.22)
4. W3Schools,About XML,Url:https://www.w3schools.com/xml/xml_whatis.asp (08.10.22)
5. Android Developers,Kotlin Coroutines,Url:https://developer.android.com/jetpack/compose/documentation (08.10.22)
6. Bruce Eckel,Svetlana Isakova – Atomic Kotlin,2021(22.09.22)
7. Медична класифікація хвороб,Url:http://kod.poltavalk.com.ua/mkkh-10-am - МКХ 10 (18.11.22)
8. Geekstand,about MVVM,Url:https://geekstand.top/development/ponimanie-mvvm-patterna-dlja-android-v-2021-godu/ (12.10.22)
9. Robert Martin – Clean Architecture,2017 (12.10.22)
10. Github, About Kodein library, Url:https://github.com/kosi-libs/Kodein (16.10.22)
11. Coderlessons, Dagger 2, Url:https://coderlessons.com/articles/mobilnaia-razrabotka-articles/vnedrenie-zavisimostei-s-pomoshchiu-dagger-2-na-android (16.10.22)
12. Eric Freeman,Elisabeth Robson – Head First Design Patterns.2ed Ed.2021 O'Reilly Media
13. Erich Gamma,Richard Helm,Ralph Johnson,John Vlissides – Design Patterns, 2009
14. Filip Babic,Luca Kordic,Nishant Srivastava – Kotlin Coroutines by Tutorials, 2022

15. Tomasz Nurkiewicz,Ben Christensen – Reactive Programming with RxJava
16. Massimo Carli – Functional Programming in Kotlin by Tutorials.A Practical Approach To Writing Safer,More Reliable Apps
17. Alexey Sohin – Kotlin Design Patterns and Best Practices
18. Marcin Moskala – Effective Kotlin Best practices
19. GIT, Url:https://git-scm.com/
20. Kinsta,What is a github, Url:https://kinsta.com/knowledgebase/what-is-github/
21. Thomas Künneth-Android UI Development with Jetpack Compose: Bring Declarative and Native UIs to Life Quickly and Easily on Android Using Jetpack Compose.
22. Jason Morris - Hands-On Android UI Development
23. https://esu.com.ua/search_articles?id=66075 – Медична статистика
24. Wikipedia,Список МКХ-10 ,Url : https://uk.wikipedia.org/wiki/ (18.11.22)
25. Wikipedia,МКХ10,Url: https://uk.wikipedia.org/wiki/ (18.11.22)
26. Avada-Media,Json,Url: https://avada-media.ua/ua/json/ (12.10.22)
27. Theastrology Page,Sqlite,Url: https://uk.theastrologypage.com/sqlite (12.10.22)
28. YoungLinux,Sqlite,Url: https://younglinux.info/sqlite/sqlite(12.10.22)
29. Git,Getting Started,Url: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F (20.11.22)

Якубенко Дмитро Олександрович
Прізвище, ім’я по батькові

Інформаційних і прикладних технологій
Факультет

122 «Комп’ютерні науки»
Шифр і назва спеціальності

«Комп’ютерні технології обробки даних (Data Science)»
Освітня програма

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему «Розробка мобільного додатку з підтримки функціонування приватної клініки» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувалась у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

(дата) (підпис здобувача)