Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмне забезпечення мобільного iOS-клієнта удосконаленої муніципальної інформаційної системи медичних послуг”

Виконала здобувачка вищої освіти
ІІ курсу, групи КІ-21М1,4
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Дробко О.С.
«____» _____ 20_ р.

Керівник проекту
кандидат технічних наук, доцент
_____ Доренський О.П.
«____» _____ 20_ р.

Рецензент
доктор технічних наук, професор
_____Мелешко Є.В.
«____» _____ 20_ р.

м. Кропивницький

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *магістр*
Галузь знань *12 "Інформаційні технології"*
Спеціальність *123* "Комп'ютерна інженерія"
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А. Смірнов
"____" _____20___року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Дробко Олені Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмне забезпечення мобільного iOS-клієнта удосконаленої муніципальної інформаційної системи медичних послуг.*
2. Керівник роботи *Доренський Олександр Павлович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти №19-13 від 17.08.2022 року
3. Строк подання роботи до захисту *19.12.2022*
4. Мета та завдання випускної кваліфікаційної роботи *Метою роботи є дослідження та удосконалення програмного забезпечення муніципальної інформаційної системи медичних послуг міста Кропивницький.*
5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*4 аркуші*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	25.10.2022	12.11.2022
Охорона праці	Оришака О.В., к.т.н., доцент	04.11.2022	20.11.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2022 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2022 р.	
3.	Розробка моделі компонента	20.10.2022 р.	
4.	Розробка структур даних	25.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2022 р.	
6.	Програмування алгоритмів	10.11.2022 р.	
7.	Розрахунок економічної ефективності	13.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2022 р.	
9.	Оформлення ПЗ	17.11.2022 р.	
10.	Попередній захист роботи	10.12.2022 р.	

Дата видачі завдання
«__»_____20__ р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20__ р.

Підпис здобувача

_____ _____

АНОТАЦІЯ

Дробко О.С. Дослідження та програмне забезпечення мобільного iOS-клієнта удосконаленої муніципальної інформаційної системи медичних послуг. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В цій випускній кваліфікацій роботі за другим (магістерським) рівнем вищої освіти удосконалено програмне забезпечення, яке призначено для того, щоб бути частиною муніципальної інформаційної системи медичних послуг, а саме клієнтом.

Метою є дослідження та удосконалення програмного забезпечення муніципальної інформаційної системи медичних послуг міста Кропивницький.

Об'єктом дослідження є процес діджиталізації надання інформації з медичних послуг на муніципальному рівні.

Предметом дослідження є методи побудови інформаційних систем та мобільних додатків для медичних послуг.

Наукова новизна полягає у вдосконаленні моделі муніципальної інформаційної систем медичних послуг за рахунок реалізації процесу офлайнового режиму функціонування системи, що на відміну від існуючих моделей муніципальних систем забезпечує доступ до даних ІС у період відсутності зв'язку з Інтернетом.

В процесі роботи над програмною моделлю виконано аналіз існуючих електронних сервісів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача, інструкції по використанню програмного продукту.

Програма може використовуватися на всіх моделях iPhone, що підтримують iOS 11+.

Програму розроблено в середовищі Xcode.

Ключові слова: комп'ютерна інженерія, додаток, клієнт, інформаційна система, iOS.

ABSTRACT

Drobko O.S. Research and software development of a mobile iOS client of an improved municipal information system for medical services.
123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2022

In this final qualification work for the second (master's) level of higher education, the software is improved, which is intended to be part of the municipal information system of medical services, namely the client.

The goal is research and improvement of the software of the municipal information system of medical services in the city of Kropyvnytskyi.

The object of the research is the process of digitalization of providing information on medical services at the municipal level.

The subject of the research is methods of developing information systems and mobile applications for medical services.

The scientific novelty consists in the improvement of the model of the municipal information systems of medical services due to the implementation of the process of the offline mode of the system functioning, which, unlike the existing models of municipal systems, provides access to IS data during the period of no Internet connection.

In the process of working on a software model an analysis of existing municipal services. All components of the software developed are fully described.

User-friendly user interface is developed. These are instructions for working with software.

The program can be used on any iPhone with iOS 11+.

The program was developed in the Xcode IDE.

Keywords: computer engineering, app, maze, client, information system, iOS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

API – Application Programming Interface – прикладний програмний інтерфейс.

Flutter – це безкоштовний фреймворк для мобільного інтерфейсу з відкритим кодом, створений Google.

IDE – Integrated development environment – інтегроване середовище розробки.

iOS – це операційна система розроблена Apple для мобільних пристроїв, планшетів та деяких інших.

React Native – це фреймворк для розробки кросплатформого програмного забезпечення для iOS та Android.

Swift – це нова мова програмування, створена Apple для розробки програмного забезпечення під девайси компанії.

UI – User interface – інтерфейс користувача.

Xcode – інтегроване середовище розробки (IDE) виробництва Apple.

XNU – ядро операційної системи, розроблене компанією Apple для використання у macOS і анонсоване як вільне та відкрите, є частиною операційної системи Darwin.

БД – база даних.

ІС – Інформаційна система.

МІС – медична інформаційна система.

МП – медичні послуги.

ПЗ – Програмне забезпечення.

ПК – персональний комп’ютер.

ТЗ – технічне завдання.

ВСТУП

Актуальність теми. Розвиток сучасних технологій зробив можливим розміщення великих потужностей на невеликих девайсах, що стало причиною популяризації смартфонів. Ці компактні девайси заполонили ринок і стати синонімом мобільності та ефективності. Зараз дуже багато компаній конкурує на поприщі розробки смартфонів, однією з найбільш відомих є компанія Apple. Програмне забезпечення смартфонів цієї фірми значно відрізняється від інших і розробка для нього потребує знання особливих програмних засобів. Проте не дивлячись на ці складнощі саме розробка під iOS вважається найбільш прибутковою.

Мобільні додатки стали повноцінними аналогами ПК застосунків та веб-сервісів. Зважаючи на це, не видається дивним, бажання органів влади використовувати цей інструмент для поліпшення механізму роботи держави. Проект “Країна в смартфоні”, намагання місцевого самоврядування покращити співпрацю з населенням стали передумовою ініціативи по створенню інформаційних систем, які прискорюють і спрощують вирішення проблем громадян. Саме мобільні застосунки дуже часто стають частинами таких інформаційних систем.

Органи управління міста Кропивницького не стали виключенням і на основі суспільної потреби на муніципальні сервіси ініціювали створення кількох інформаційних систем, що необхідні для розвитку різних сфер діяльності. Наприклад, успішно реалізований проект “Зелений Кропивницький” – інструмент, що допомагає інформувати громадськість про рішення прийняті у сфері озеленення та контролювати ці рішення [37].

Ще однією ініціативою було створення інформаційної системи медичних послуг. Муніципальна влада зацікавлена в сервісі, який буде забезпечувати доступ до актуальних даних про медичне обладнання всіх лікарень міста через

мобільний застосунок.

Зацікавленість органів управління міста Кропивницького та потреба в діджиталізації сфери медичних послуг забезпечують актуальність теми випускної кваліфікаційної роботи.

Мета й завдання дослідження. Метою роботи є дослідження та удосконалення програмного забезпечення муніципальної інформаційної системи медичних послуг міста Кропивницький.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих інформаційних систем медичних послуг.

– Дослідження муніципальної інформаційної системи медичних послуг

– Програмна реалізація муніципальної інформаційної системи медичних послуг.

Об'єктом дослідження є процес діджиталізації надання інформації з медичних послуг на муніципальному рівні.

Предметом дослідження є методи побудови інформаційних систем та мобільних додатків для медичних послуг.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії алгоритмів, методах розробки програмного забезпечення, а також методах тестування програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

Вдосконалено модель муніципальної інформаційної систем медичних послуг за рахунок реалізації процесу офлайнового режиму функціонування системи, що на відміну від існуючих моделей муніципальних систем забезпечує доступ до даних ІС у період відсутності зв'язку з Інтернетом.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі програмної реалізації мобільного iOS-клієнта муніципальної інформаційної системи

медичних послуг.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Частина результатів цієї роботи апробовані на IV міжнародній науково-практичній конференції «Інформаційна безпека та комп'ютерні технології» (Кропивницький, 2021) та на II всеукраїнській науково-практичній конференції здобувачів вищої освіти й молодих учених «Комп'ютерна інженерія і кібербезпека: досягнення та інновації» (Кропивницький, 2020), на LV науково-технічній конференції здобувачів вищої освіти «Наука в ЦНТУ: основні досягнення та перспективи розвитку» за підсумками проведення «Дня науки – 2021» (Кропивницький, 2021), а також опубліковані у збірниках матеріалів означених науково-практичних конференцій. Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація мобільного iOS-клієнта муніципальної інформаційної системи медичних послуг, є актуальною задачею, яка потребує вирішення у даній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Програмне забезпечення, що є предметом дослідження та удосконалення, виступає частиною муніципальної інформаційної системи медичних послуг. Призначення ІС полягає в описі об’єкта або групи об’єктів, станів, взаємодій, що виражається через певні показники. До головних завдань належать [27]:

– виявлення джерел інформації;

– збирання, реєстрація, обробка та видача інформації;

– розподіл інформації між користувачами.

Муніципальна означає, що ІС створюється за ініціативи органів місцевого самоврядування задля поліпшення комунікації з громадою.

А те що це ІС медичних послуг визначає, що головне її призначення це автоматизація будь-яких процесів пов’язаних з інформуванням про роботу медичних установ.

Муніципальна ІС медичних послуг призначена для:

– автоматизації процесу збору та оновлення даних про медичне обладнання всіх лікарень міста;

– забезпечення комфортного доступу до інформації для кінцевого користувача;

– збору даних про активність інтересу громади до даного питання.

Таким чином ця ІС надасть жителям міста інформацію, що раніше ніколи не групувалася в один пакет даних.

1.2 Область застосування

Область застосування муніципальної ІС медичних послуг випливає з призначення:

– муніципальна – для всіх жителів міста Кропивницький;

– медична – для всіх задіяних в медичній сфері: лікарів та хворих.

За ініціативою органів місцевого самоврядування всі медичні заклади міста надають повну інформацію про медичні послуги та стан медичного обладнання, за допомогою ІС ця інформація обробляється та стає доступною всім жителям міста. Робимо висновок, що областю застосування даного програмного забезпечення є сфера медичних послуг конкретного міста.

Висновки до розділу 1

Під час роботи над першим розділом було визначено призначення та область застосування муніципальної інформаційної системи медичних послуг. Призначенням є інформування громади про медичні послуги та роботу медичних закладів міста. Виходячи з призначення областю застосування є сфера медичних послуг міста Кропивницький.

Визначена область використання та призначення ІС стане основою для дослідження методів удосконалення системи та вибору мови програмування і засобів роботи з системою.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи

Міська рада Кропивницького вже деякий час займається розвитком електронних сервісів для громади, на офіційному сайті міста представлений перелік різних інформаційних сервісів (рисунок 2.1).

Рисунок 2.1 – Електронні сервіси міста Кропивницький [21]

Зелений Кропивницький

Зелений Кропивницький – це інструмент, що допомагає міській раді міста Кропивницького інформувати громадськість про рішення прийняті у сфері озеленення, а громадськості – контролювати ці рішення (рисунок 2.2).

Сфера озеленення у більшості українських міст є закритою та непрозорою. Несподівані вирубки дерев, неправильна обрізка крон призводить до обурення в соціальних мережах, звернень громадськості та протестів. Інколи посадовці навмисне приймають такі рішення для того, щоб “розчистити” територію під будівництво або новий магазин.

Міська рада міста Кропивницького однією з перших в Україні відкриває детальні та актуальні дані щодо управління зеленими насадженнями. Відтепер будь-хто може завчасно перевірити, які роботи передбачені, хто є їх замовником, а хто – виконавцем. За декілька кліків можна подивитися, чи мають право зрізати дерева комунальні служби, та чи достовірні їх документи.

Це забезпечуватиме прозорість та обґрунтованість рішень міської влади, допоможе ефективніше використовувати кошти громади, зменшить корупційні ризики [23].

Проект представлений у вигляді веб-сторінки з актуальними даними.

Рисунок 2.2 – Сервіс «Зелений Кропивницький» [23]

Освітня карта Кропивницького

Управління освіти Кіровоградської міської ради запустило онлайн-карту освітніх закладів міста (рисунок 2.3). Освітня карта інформує про фінансування з міського бюджету ремонтів, харчування, передбачені на той чи інший навчальний заклад. За допомогою карти можна переглянути напрямки класів, витрати батьківського комітету, обсяг фінансування з міського бюджету [40].

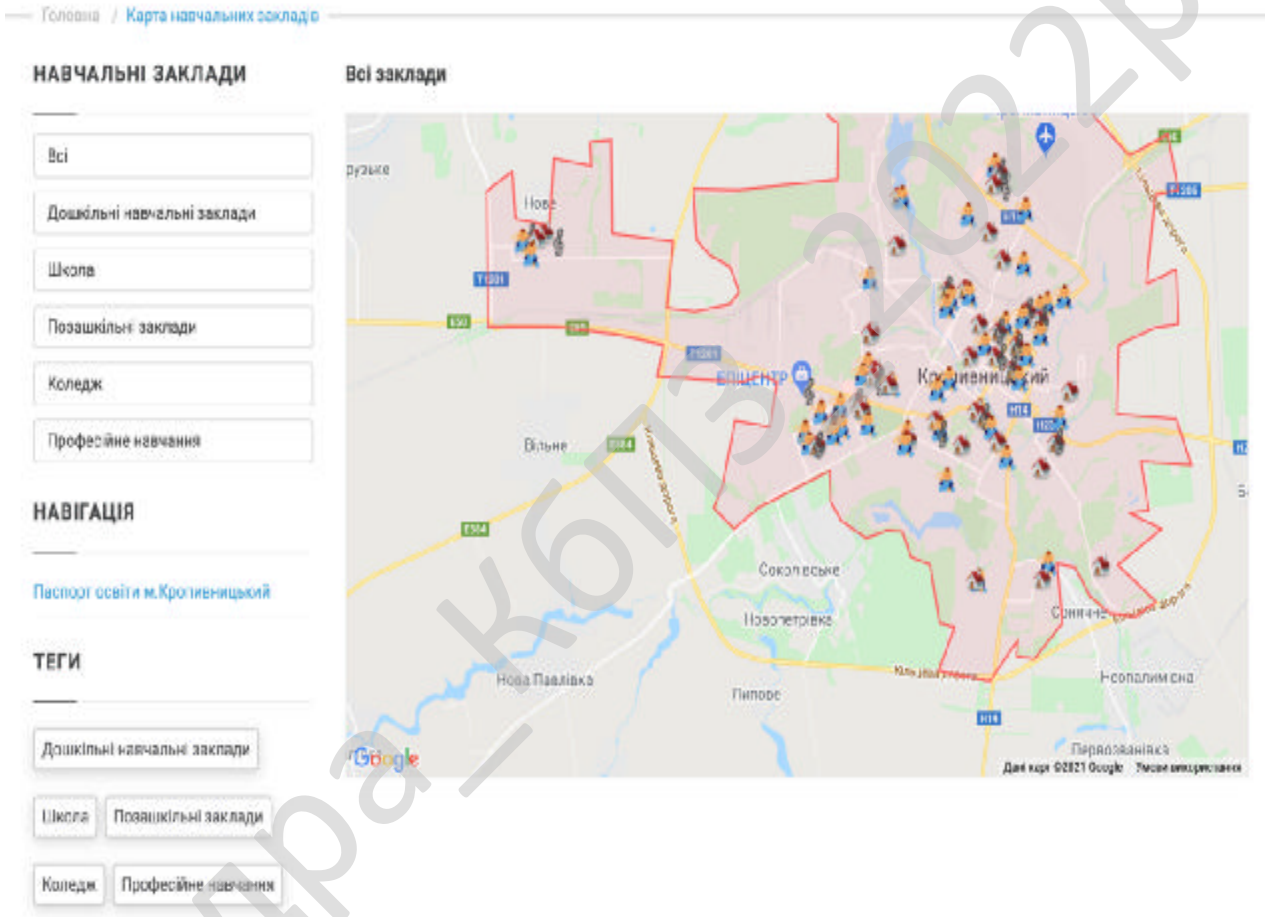

Рисунок 2.3 – Сервіс «Освітня карта Кропивницького» [40]

Громадський бюджет

Громадський бюджет – це е-сервіс, що запечує прозорість розподілу спеціальної частини бюджету, з якої здійснюється фінансування заходів, виконання робіт та надання послуг, визначених безпосередньо членами територіальної громади та оформлених відповідно проектних пропозицій, які

стали переможцями конкурсу (рисунок 2.4) [16].

Тут зібрані документи, новини, адреси пунктів супроводу, інформація про проекти та їх динаміку. Користувачі задля голосування можуть використати усі наявні в країні варіанти ідентифікації. На сервісі можна побачити детальну звітність про використання коштів і реалізацію проектів. На ньому зберігаються уся історія кожного етапу та за всі роки реалізації проектів тощо [59].

Інструкція по онлайн голосуванню на платформі для користувачів: https://youtu.be/BoOEGgfkIMM

Інструкція для подання проекту в рамках конкурсу Громадський бюджет-2021

В межах громадського бюджету міста Кропивницький можуть бути реалізовані ідеї, втілення яких належить до компетенції міської влади та які спрямовані на покращення інфраструктури міста. Реалізація проекту повинна бути можлива протягом одного року

Питання та відповіді

1. Що таке Громадський бюджет або бюджет участі?

Громадський бюджет – це частина бюджету міста Кропивницького, з якого здійснюється фінансування заходів, виконання робіт та надання послуг, визначених безпосередньо членами територіальної громади та оформлених відповідно проєктних пропозицій, які стали переможцями конкурсу.

Рисунок 2.4 – Сервіс «Громадський бюджет» [16]

Обґрунтування необхідності удосконалення системи

Міська рада Кропивницького зацікавлена в розвитку електронних сервісів для жителів міста і саме тому вони оголосили список сервісів для яких вони готові забезпечити пакет оновлюваної інформації. Одним з таких сервісів є муніципальна інформаційна система медичних послуг.

З основного призначення МІС, а саме забезпечення доступу до інформації,

випливає напрям покращення системи. В ПЗ мобільного додатку можна додати офлайн режим. Офлайн режим допоможе забезпечити доступ до інформації незалежно від підключення до мережі та зможе позитивно вплинути на ефективність роботи муніципальної інформаційної системи медичних послуг.

Основні принципи побудови та класифікація медичних ІС

Зазвичай медична інформаційна система складається із блоків:

– електронні медичні записи пацієнтів;

– ресурси установи та їх розподіл;

– дані медичних досліджень;

– фінансова та адміністративна інформація;

– засоби зв'язку між працівниками закладу.

Представлені на ринку України медичні інформаційні системи прийнято ділити на системи документообігу (Docflow) та системи процесів (Workflow).

Іноді складно розрізнити ці два види систем, проте спрощену відмінність між ними можна виразити так:

– якщо об'єктом дій користувача медичної інформаційної системи є документ (історія хвороби, замовлення пацієнта, журнал запису тощо), то перед нами типова система документообігу;

– якщо ж користувач інформаційної системи керує лікувальним процесом, ресурсами, статусами процесів тощо, то ця медична інформаційна система є процесною.

Від виду медичних інформаційних систем залежить результати їх застосування.

Системи документообігу дозволяють:

– зменшити кількість паперових документів, використовуючи їх наслідуваність;

– зменшити кількість помилок під час оформлення;

– досягти типізації документів та обов'язковості їх створення тощо.

На відміну від систем документообігу процесні системи призначені для

оптимізації використання ресурсів, поліпшення медичного сервісу, підвищення ефективності праці медичних працівників.

Інша класифікація медичних інформаційних систем пов'язана зі сферою, яку вони дають можливість організувати:

– Electronic Medical Records (EMR) – системи медичних записів (історії хвороби, амбулаторні карти , медична статистика і т.д.).

– Radiology Information System (RIS) – збереження медичних зображень та доступ до них.

– Enterprise Resource Planning (ERP) – системи для управління фінансовими, трудовими та матеріальними ресурсами закладу.

В основному системи, розроблені для державних закладів охорони здоров'я, належать до систем медичних записів (EMR), а системи, що застосовуються в приватних закладах, – до управлінських систем (ERP) різного ступеня складності. Це визначається різницею підходів до організації робочого процесу в установах різних форм власності. Адже якщо приватна медицина зосереджена на оптимізації та ефективності, то державна – переважно на обліку та медичних записах.

Завдання медичних ІС

Першим і найважливішим завданням, з яким здатні впоратися медичні інформаційні системи, є централізація даних. Вона передбачає, що всі дані, внесені до інформаційної системи закладу охорони здоров'я, доступні у будь-якій точці входу до системи. Це дозволяє централізовано забезпечувати інформацією весь персонал закладу.

Другим важливим завданням медичних інформаційних систем є типізація даних як щодо інформації, так і щодо процесів. Тобто підхід до всіх пацієнтів уніфікується, як і їхня медична документація, яку оформлюють за єдиним зразком.

Третє завдання інформаційних систем у медицині – забезпечення доступності інформації для її обробки. Типізована та централізована інформація

доступна для будь-якої обробки та аналізу. Це основна перевага медичних інформаційних систем, яку, проте, часто недооцінюють.

Інші переваги медичних інформаційних систем – покращення сервісу для пацієнтів, підвищення якості медичного обслуговування тощо є результатом успішної реалізації згаданих завдань.

Етапи впровадження медичної ІС

Впровадження медичної інформаційної системи починається ще до її обрання на етапі планування.

Насправді, першим етапом впровадження медичної інформаційної системи є не планування, а прийняття рішення щодо необхідності її впровадження в конкретній установі. На цьому етапі необхідно знайти відповіді на такі питання:

- навіщо закладу потрібна медична інформаційна система;
- які завдання вона може вирішити у цьому закладі;
- які для цього є ресурси.

На етапі планування слід приділити достатньо уваги запитам працівників закладу, оскільки саме вони надалі будуть користувачами системи. Від того, наскільки результат співпадатиме з їхніми очікуваннями, залежить ефективність роботи системи в цілому.

Наступним етапом є вибір системи, зустрічі з її розробниками, тестове використання. На цьому етапі складають плани безпосереднього впровадження системи та здійснюють навчання персоналу.

Особливу увагу слід приділити вибору відповідального за впровадження (представника закладу охорони здоров'я). Цей співробітник стане експертом та впроваджуватиме зміни в середу закладу. Крім того, слід продумати, що відбуватиметься з інформацією, що вже є (електронною – у разі переходу з іншої інформаційної системи або паперової – при першому впровадженні).

Третій етап – розгортання платформи, тобто підготовка інфраструктури для навчання персоналу. Саме на цьому етапі розгортаються мережі, сервери та програмне забезпечення. Тут керівнику закладу охорони здоров'я варто звернути

увагу, що тестування медичної інформаційної системи доцільно здійснювати безпосередньо у закладі, де планується її використання, а не користуватися тестовими серверами розробників. Адже, на жаль, досить часто при тестуванні системи виявляють недоліки інфраструктури, проте завчасне виявлення дозволяє виправити їх.

Найгірша ситуація складається, якщо старт системи запланований на завтра, а в останній момент виявиться, що сервер не підходить або, банально, не прокладено мережу до конкретного кабінету.

Четвертий етап – прописування алгоритмів процесів. Цей етап передує навчанню персоналу. Однак часто помилковий підхід до навчання персоналу полягає в тому, що його вчать натискати на клавіші, тобто працювати з інформаційною системою.

Набагато найкращим рішенням є використання кейсів. Суть кейсів полягає у вирішенні типових завдань, з якими надалі персонал зіштовхуватиметься щодня. За такого підходу замість «ця кнопка створить нову історію хвороби», а «ця кнопка роздрукує епікриз», навчання проходить по кейсах «Стаціонарне лікування пацієнта від прийому до виписки», розділеним по структурним підрозділам установи (наприклад, від приймального відділення до операційної). Цей етап також дозволяє виявити деякі недоліки медичної інформаційної системи, які можна виправити до її впровадження.

П’ятий етап – навчання персоналу. Він є найскладнішим, оскільки є нескінченним. Необхідно ретельно спланувати графік навчання. Його краще проводити у невеликих групах, об’єднаних за типовими завданнями. Перед початком навчання слід витратити деякий час на роз’яснення персоналу важливості всього проекту, пояснити, скільки це триватиме і до кого слід звертатися на вирішення окремих питань.

Шостий етап – "невдалий" старт. Часто після закінчення навчання робиться спроба першого старту, під час якого виявляють ще декілька істотних недоліків навчання чи самої системи. Тому перший старт краще виконати на

тестовому полігоні: всі люди на своїх місцях, всі, як насправді, тільки пацієнти несправжні. Крім того, для запобігання неприємним сюрпризам навчальний етап слід завершити іспитом для персоналу на знання системи.

Фінальний етап – старт. Краще розділити його на кілька підетапів, поступово залучаючи до роботи з медичною інформаційною системою дедалі більше користувачів.

Постфінальний етап – доопрацювання. Ще не було жодного проекту, де вдалося б передбачити всі потреби закладу охорони здоров'я. Саме тому важка робота починається після впровадження медичної інформаційної системи. На початку цього етапу необхідно налагодити систему зворотного зв'язку між персоналом та розробниками інформаційної системи.

Переваги використання медичних ІС

Переваги використання медичних інформаційних систем у лікарні залежать від виду МІС та успішності її впровадження у конкретній установі.

При вдалому виборі медичної інформаційної системи її впровадження має сприяти таким позитивним змінам у організації роботи установи:

– спрощення доступу до необхідної інформації;

– відсутність необхідності у подвійному внесенні даних;

– можливості використання шаблонів документів;

– полегшення пошуку ресурсів;

– можливості доступу до довідкової інформації.

У той же час при впровадженні інформаційної системи персоналу закладу охорони здоров'я потрібно докласти певних зусиль – вносити інформацію до системи за наявними шаблонами та формами, послідовно вести електронну медичну документацію та обов'язково дотримуватися алгоритму роботи з тією чи іншою інформаційною системою.

Переваги, які матиме керівник закладу охорони здоров'я від використання медичної інформаційної системи, залежать від завдань, які він ставив під час виборів та впровадження тієї чи іншої системи. Зазвичай переваги зводяться до:

– прозорості системи обслуговування пацієнтів;

– прозорості фінансової діяльності та матеріальних ресурсів;

– широкі можливості щодо звітів, зокрема звітність у режимі реального часу.

Пацієнти закладу охорони здоров'я мають переваги при використанні медичної інформаційної системи. Ці переваги залежать від можливостей самої системи, а також від його функцій, до яких заклад надав доступ пацієнтам. Серед простих переваг для пацієнтів можна назвати такі:

– доступ до власної медичної інформації;

– спілкування з лікарем поза медичним закладом;

– відсутність черг (планування часу відвідування лікаря);

– покращення медичного результату (управління якістю).

Безумовно, цей перелік не є вичерпним, лише дозволяє окреслити деякі переваги впровадження медичних інформаційних систем. Кожен конкретний випадок застосування системи має свої результати.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Мобільне програмне забезпечення, що буде вдосконалюватися в результаті цієї роботи розраховане для використання на основі операційної системи iOS.

iOS – це операційна система розроблена Apple для мобільних пристроїв, планшетів та деяких інших. Основною особливістю даної ОС є те, що вона може використовуватися лише на пристроях виготовлених компанією Apple. В цій ОС використовується ядро XNU(мікроядро Match), що зберігає закритий початковий код [1].

Створювати такі програми можна використовуючи ноутбуки або ПК різних фірм, але вони будуть мати деякі відмінності.

До недавнього часу розробка під iOS вважалася доступною лише ПК та

ноутбуках Apple, але в 2015 та 2017 роках з'явилися нові технології Flutter та React Native, відповідно.

Flutter – це безкоштовний фреймворк для мобільного інтерфейсу з відкритим кодом, створений Google. Він дозволяє створити власний мобільний додаток лише з однією кодовою базою, використовувати одну мову програмування та одну базу коду для створення двох різних програм (для iOS та Android) [60].

React Native – це фреймворк для розробки кросплатформого ПЗ для iOS та Android. Він створений на основі React [12].

Обидва фреймворки можна встановити на кросплатформові IDE, але для створення версії для публікації все ще залишається необхідність використання нативного IDE від Apple – Xcode.

Xcode IDE розроблене з урахуванням усіх потреб при розробці ПЗ для macOS, iOS, watchOS і tvOS. Apple додали в Xcode велику кількість нового функціоналу для полегшення процесу розробки:

– систему управління проектом для визначення програмних продуктів;

– середовище для редагування коду, що включає можливості підсвічування синтаксису, автозаповнення коду та індексацію символів;

– просунуту програму перегляду і пошуку документації Apple;

– контекстно-залежний інспектор для перегляду інформації про виділений в коді символі;

– просунуту систему збирання проекту з перевіркою залежностей і перевіркою правил складання;

– компілятори GCC, що підтримують мови C, C ++, Objective-C, Objective-C ++ та інші;

– підтримка LLVM і Clang для мов C, C ++ і Objective-C;

– вбудоване налагодження вихідного коду з використанням GDB;

– розподілені обчислення, що дозволяють поширювати великі проекти через кілька мережевих пристроїв;

– інтелектуальна компіляція, яка прискорює час компіляції одного файлу;

– просунуті можливості по налагодженню коду;

– просунуті засоби налагодження, що дозволяють робити глобальні зміни в коді без зміни його поведінки;

– підтримка знімків проекту, які надають легше управління вихідним кодом;

– підтримка запуску засобів продуктивності для аналізу програми;

– підтримка вбудованої системи управління вихідним кодом;

– підтримка AppleScript для автоматизації процесу складання;

– підтримка налагоджувальної інформації в форматах DWARF і Stabs (налагоджувальна інформація для всіх проектів за замовчуванням генерується в форматі DWARF).

– Interface Builder (для розробки інтерфейсу);

– Simulator (доступ до віртуальний девайсів для компіляції проекту).

Зважаючи на всі плюси та мінуси, а також той факт, що мобільний додаток буде створюватися лише для iOS девайсів, найпростішим для використання буде Xcode і цей вибір також впливає вибір мови програмування. Xcode підтримує дві мови програмування: Swift та Objective-C.

Objective-C – це первинна мова розробки для iOS, синтаксис походить від C та SmallTalk. Хоча нещодавно уведена мова Swift – це майбутнє розвитку iOS, багато просунутих проектів все ще покладаються на Objective-C [31].

Swift – це нова мова програмування (2014), створена Apple спеціально для розробки ПЗ під девайси компанії. Вона інтуїтивно зрозуміла, легка до читання, суворо типізована, проте доволі стійка до помилок програмістів.

Окремо треба зауважити, що Apple реалізували механізми інтеграції, як Swift в Objective-C, так і навпаки.

Поточна кодова база була створена на Swift та Xcode, але новий код може інтегруватися, використовуючи будь-яке із вище запропонованих рішень. За рекомендацією Apple для роботи над даним проектом буде використовуватися

Swift та Xcode.

2.3 Розгорнута постановка завдання

Відповідно до технічного завдання на магістерську дипломну роботу в процесі дослідження та удосконалення програмного забезпечення муніципальної інформаційної системи медичних послуг необхідно виконати наступні завдання:

1) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей, результати аналізу врахувати при подальшій роботі;

2) провести наукове дослідження методів вдосконалення муніципальної інформаційної системи медичних послуг та визначити структуру та методологію покращення інформаційної системи, розробити функціональну, структурну схеми та узагальнену діаграму процесів ПЗ;

3) внести всі необхідні правки в програмне забезпечення для мобільного додатку, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

4) описати результати наукової діяльності, виділити наукову новизну;

5) розробити рекомендації для організаційних та методичних заходів, які забезпечують впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

6) сформувати висновки про виконаний обсяг робіт та одержані результати.

Висновки до розділу 2

Під час роботи над другим розділом було виконано аналіз аналогічних систем, доступних програмних засобів та виконано розгорнуту постановку завдання. В результаті аналізу аналогічних систем відмічено схожість за типом

клієнту всіх МІС, окрім муніципальної інформаційної системи медичних послуг. Враховуючи унікальність клієнту та призначення ІС, був обраний напрям та метод удосконалення через додавання офлайн режиму в роботу мобільного додатку. Результатом огляду всіх доступних програмних засобів став вибір середовища та мови розробки, а саме Xcode IDE та Swift відповідно.

Розгорнута постановка задачі виконана на основі ТЗ, та розробки покроково описані дії вході дослідження та вдосконалення програмного забезпечення муніципальної інформаційної системи медичних послуг.

Тож отриманий результат є базою для проектних рішень щодо удосконалення муніципальної інформаційної системи медичних послуг.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Муніципальна інформаційна система медичних послуг функціонує на основі клієнт-серверної архітектури. Клієнт підтримує зв’язок з сервером через мережу інтернет. Клієнтом виступає мобільний додаток для iPhone, а сервером – група із декількох сервісних елементів. Варто зауважити, що кожен з сервісних елементів має незалежний канал для обміну даними з клієнтом.

Сервісні елементи відповідають за побудову маршрутів до лікарень з урахуванням геопозиції користувача, аналіз аналітичних даних та побудову статистичних графіків та зв’язок з БД медичних послуг. Найголовніша роль у сервера, який займається взаємодією з БД та надсиланням на клієнт інформацію про лікарні та медичні послуги нашого міста.

Мобільний додаток використовує графічний інтерфейс для взаємодії з користувачем. За потребою користувача він відправляє або отримує інформацію від серверної частини, а також працює з мапами, будує маршрути та виконує збір даних для статистики. Окрім того, буде доопрацьовано модуль роботи в офлайн режимі, що забезпечує стабільний доступ до інформації незалежно від підключення до мережі інтернет.

Медичні інформаційні системи зазвичай складаються з модулів. Це дозволяє зібрати та налаштувати МІС у потрібній конфігурації для установ різного типу та забезпечити необхідний функціонал з можливістю подальшого додавання/видалення модулів. Структура медичної інформаційної системи – окремі компоненти, які можна поєднати у кілька великих груп:

1. *Аналітичні та управлінські компоненти*. Модулі та засоби ведення управлінського обліку, інструменти аналізу якості та ефективності медичних послуг. Ці складові МІС дозволяють проаналізувати стан медичної організації,

виявити проблемні місця та оптимізувати бізнес-процеси. На рівні користувача – пошук медичних записів за будь-якими критеріями з урахуванням обмежень за рівнем доступу. Результати аналізу можна вивести на екран у вигляді графіків, таблиць або друку.

2. *Медичні компоненти*. Усі модулі, пов'язані з реєстрацією пацієнтів, ведення реєстру електронних медичних карток, облік лікарняних листів, ведення протоколів лікування, інформаційний супровід лікування пацієнтів у різних типах установ (амбулаторія, поліклініка, стаціонар), медична статистика та аналітика, історія хвороби та багато іншого.

3. *Фінансово-економічні складові*. До них відносяться інструменти ведення обліку медикаментів, управління запасами, розрахунок собівартості лікування та тарифів на надання медичних послуг, розрахунок надбавок лікарям, інструменти проведення економічного аналізу діяльності організації тощо.

4. *Компоненти обміну даними*. Ведення уніфікованих реєстрів, каталогів та довідників, обмін даними у системі закладів охорони здоров'я, обробка отриманих даних.

5. *Загальнотехнічні компоненти.* Контроль доступу користувачів та захист бази даних, а також підтримка можливостей інтеграції з іншими системами та програмами.

3.2 Розробка структурної схеми

В основі структури муніципальної інформаційної системи лежить клієнт-серверна архітектура. Одразу можна виділити два основних компоненти: клієнт та сервер. В класичній моделі клієнт-серверної архітектури сервер відповідає за обробку і зберігання даних, виконує функції по запитам клієнтів та надає доступ до інших ресурсів. В даному випадку за допомогою мережі інтернет отримано доступ до трьох сервісних елементів.

По-перше, це сервер, що взаємодіє з базою даних медичних послуг

лікарень міста. Він відповідає за роботу з даними та комунікацію з клієнтом за допомогою затвердженого API інтерфейсу. По-друге, це Firebase – сервіс Google створений для полегшення процесу розробки. Firebase надає багато різних послуг, але нас цікавить функція збору аналітичних даних – Google Analytics. Google Analytics – це безкоштовне рішення, яке надає уявлення про використання додатків та залучення користувачів. Звіти Analytics допомагають чітко зрозуміти, як поводяться користувачі, що дозволяє приймати обґрунтовані рішення щодо маркетингу додатків та оптимізації продуктивності. По-третє, це Google Maps – це сервіс розроблений Google, основним його призначення є робота з мапами, крім того він має доступ супутникових знімків та умов руху в режимі реального часу. В даному ПЗ Google Maps використовується для побудови маршрутів з урахуванням поточної геолокації користувача та координатами необхідної лікарні.

Клієнтом виступає мобільний застосунок основними його функціями можна назвати відображення даних та отримання інформації від користувача через графічний інтерфейс, а також робота з серверною частини інформаційної системи.

Основні структурні елементи мобільного ПЗ:

– графічний інтерфейс для взаємодії з користувачем;

– бізнес-логіка – забезпечує роботу додатку, а саме контролює різні процеси;

– навігація – забезпечує взаємодію між екранами, а саме перехід на наступні екрани та повернення до попередніх;

– нетворк – це сервіс, що забезпечує взаємодію додатку (що виступає клієнтом) з сервером, через мережу інтернет. Він формує запити та отримує відповіді відповідно до узгодженої документації (API);

– аналітика – це сервіс, що займається аналізом стану додатку та дій користувача, дані отримані на основі цього аналізу передаються через мережу інтернет до спеціального хмарного сервісу;

– робота з мапою – цей елемент відповідає за роботу з нативною мапою та GoogleMaps (через мережу інтернет).

– кешування – цей модуль забезпечити роботу додатку навіть при відсутності підключення до мережі інтернет.

На основі структурної схеми клієнт-серверної архітектури [28] та структурних елементів муніципальної інформаційної системи медичних послуг була створена створена структурна схема (рисунок 3.1).

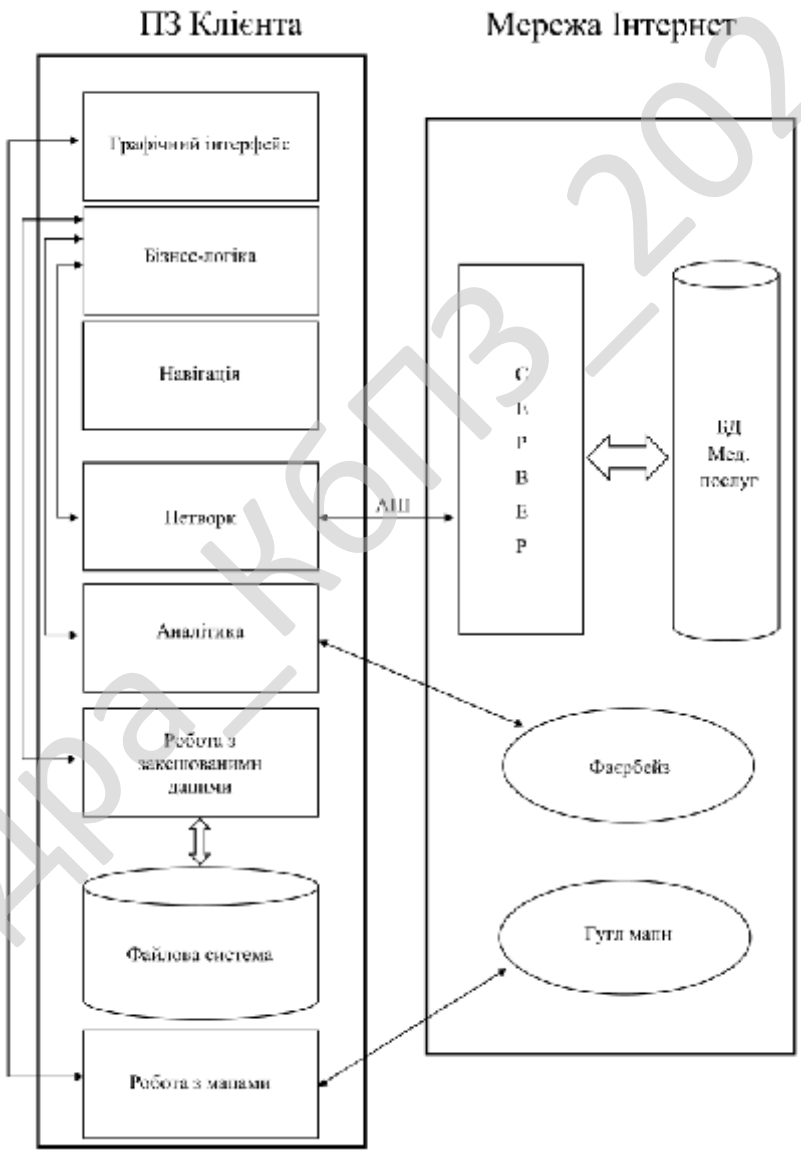

Рисунок 3.1 – Структурна схема удосконаленої муніципальної інформаційної системи медичних послуг

В результаті розробки структурної схеми муніципальної інформаційної системи медичних послуг виділений ряд основних компонентів та зв’язків між ними, як системи в цілому, так і окремо ПЗ клієнта та ПЗ серверної частини.

3.3 Розробка функціональної схеми

Розробка функціональної схеми інформаційної системи базується на використанні структурної схеми. За допомогою структурної схеми легко виділяються основні компоненти системи, а далі залишається визначити лише їх головні функції та типи зв’язків між ними.

Перший великий блок на функціональній схемі розділений на три частини: ViewController, View Model та Coordinator. Таке об’єднання трьох елементів в один блок обґрунтовується архітектурою додатку в якій кожен екран представлений трьома даними елементами.

Елемент View Controller відповідає за побудову екранів та взаємодію з UI елементами. Екран можно побудувати лише за допомогою коду, або можна використати файли для побудови графічних елементів – *.xib або *.storyboard. Всі екрани в проекті створені в *.storyboard, їх графічні представлення побудовані за допомогою вбудованих або спеціальних графічних елементів, а потім прописана відповідність між ними та контролерами. Якщо View Controller має графічне представлення в іншому файлі з’являється необхідність створити зв’язки між елементами та кодом, що відповідає за їх конфігурацію. Після всіх налаштувань ViewController може заповнювати та оновлювати дані, що відображаються на екрані. Дані для екрану отримуються від View Model.

View Model відповідає за всю бізнес-логіку екрану. View Model створює пакети даних для екрану, отримує всю необхідну інформацію та відформатує її до необхідного вигляду. Для отримання або передачі інформації View Model зв’язується з сервісами, таким чином вся робота з сервісами відбувається лише через View Model. Важливо підкреслити, що саме View Model реалізує передачу

даних між View Controller та Coordinator. Найпоширенішим прикладом такої передачі є навігація між екранами, View Controller отримує інформацію, що користувач нажав на кнопку, передає це в View Model, а та в свою чергу в Coordinator, що виконує процес переходу на інший екран.

Як вже було описано вище, Coordinator – це елемент, що відповідає за навігацію між екранами, проте ще однією важливою функцією даного класу є створення View Model та View Controller. При ініціалізації зв'язок з Coordinator зберігає лише View Model, тому зв'язок з View Controller зображено одностороннім.

Окрім, першого великого блоку на діаграмі зображені, ще чотири блоки. Три з них мають зв'язок лише з View Model – це сервіси, а один з View Controller – це просто узагальнення процесу роботи з мапою.

Першим з двох сервісів є Network або робота з серверною частиною. Він містить деякі доповнення, що дозволяють йому працювати з мережею інтернет. Інтерфейс для взаємодії з даним блоком має чотири функцій:

– getServiceType – ініціює запит на отримання всіх видів медичних послуг;

– getSearch – ініціює запит на пошук;

– getHospitals – ініціює запит на отримання всіх лікарень, що надають певну медичну послугу;

– getHospital – ініціює запит на отримання детальної інформації про певну лікарню;

Після створення запит надсилається до серверу і відповідно до API, приходить відповідь. Отримані дані перетворюються з формату json в спеціальні моделі, які потім повертаються в View Model. Варто зауважити, що існує декілька типів відповідей, деякі з них можна вважати валідними, а деякі помилками. Проте єдиною залишається необхідність розпарсити відповідь та передати її.

Другим сервісом є центр збору аналітичних даних. Його інтерфейс досить простий, він очікує стан додатку від View Model. Він аналізує цей стан та формує

спеціальне повідомлення, що відправляється в Firebase.

Налаштування файлової системи девайсу та робота з закешованими даними, що забезпечують роботу офлайн режиму – це задачі третього сервісу, зображеного у вигляді окремого блоку. Після отримання відповіді від серверу, разом із запитом її буде в записано в файли, що будуть зберігатися на девайсі, і потім при втраті зв'язку з мережею інтернет відповіді за запити будуть зчитуватися з цих файлів. Вся інформація буде передаватися і повертатися в View Model, а вся робота з даними буде інкапсульована.

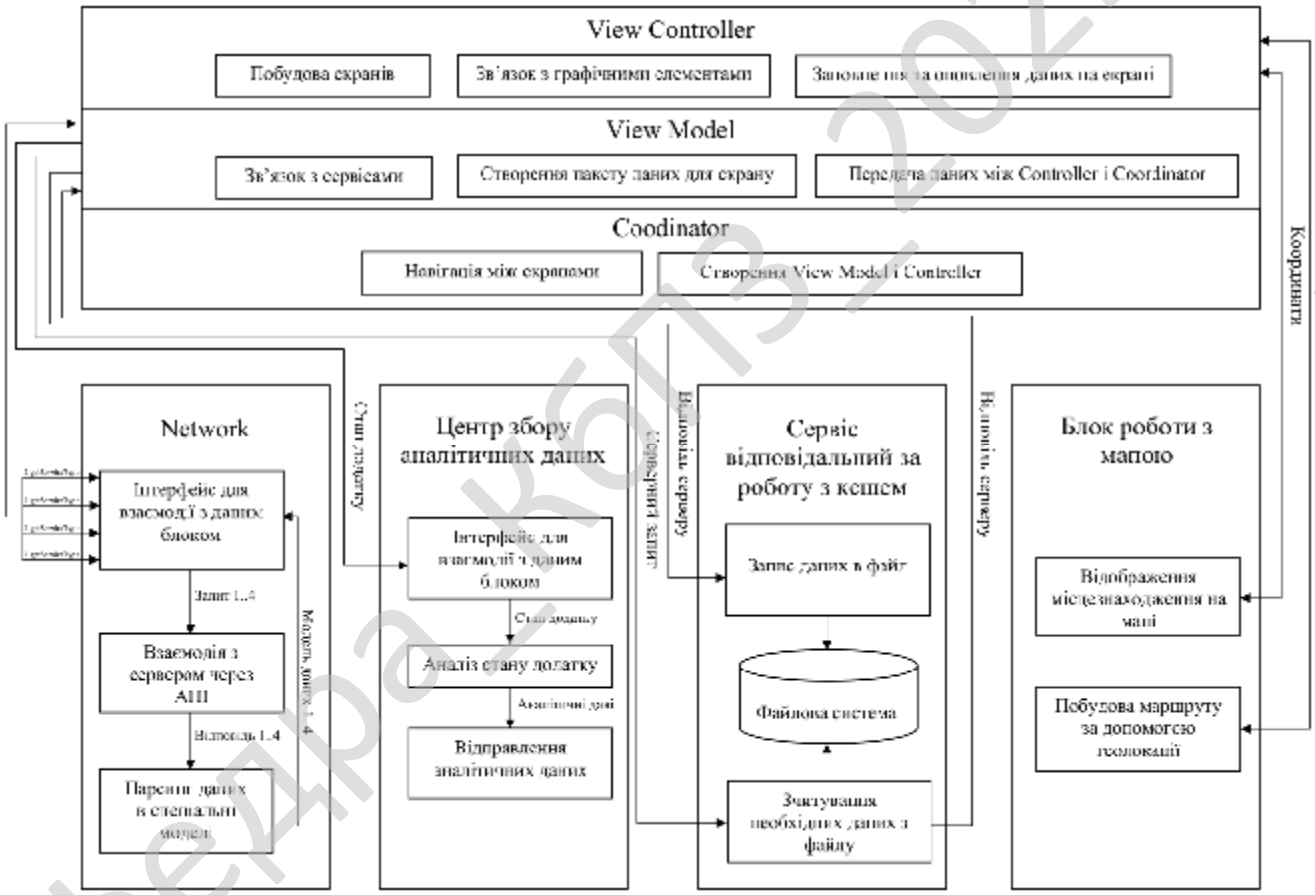

Рисунок 3.2 – Функціональна схема удосконаленої муніципальної ІС медичних послуг

Блок роботи з мапою, як було описано вище, виступає узагальненням для двух основних завдань, відображенням локації та побудови маршруту. Обидва

завдання виконуються незалежно, проте розпочинає їх роботу View Controller, передаючи координати. Відображення місцезнаходження певної локації на мапі виконується за допомогою нативного UI елементу, що містить самостійну логіку, тому View Controller лише передає координати та змінює деякі його конфігурації. Побудова маршруту забезпечується стороннім сервісом GoogleMaps. View Model формує посилання та передає його View Controller, що в відкриває посилання на Google Maps за допомогою браузеру.

В результаті розробки функціональної схеми на основі структурної схеми (рисунок 3.1), визначені функції та зв'язки для усіх компонентів програмного забезпечення клієнту удосконаленої муніципальної інформаційної системи медичних послуг. За допомогою цих даних побудована функціональна схема (рисунок 3.2).

3.4 Розробка діаграми процесів

Розробка діаграми процесів вимагає чіткого та покрокового розуміння функціонування системи. В конкретному випадку, розглянуто процеси роботи ПЗ клієнту удосконаленої муніципальної інформаційної системи медичних послуг.

На початку роботи ініціалізується головний Coordinator, котрий в свою чергу створює головний View Controller та головну View Model, залишаючи між ними двосторонній зв'язок. Важливо, що якщо зв'язок односторонній, як у випадку з View Controller та Coordinator, то лише Coordinator може впливати на роботу View Controller. В свою чергу View Controller не має ніякої інформації про роботу чи існування Coordinator. Етап на якому всі три елементи головний View Controller, головний Coordinator та головна View Model запущені, можна вважати завершенням запуску головного екрану.

Під час початку роботи головна View Model починає взаємодіяти з сервісами, а саме передає стан додатку, надсилає запит на отримання інформації про медичні послуги та після отримання відповідь кешується або у випадку

втрати зв’язку з мережею Інтернет відповідь зчитується з раніше записаних в файлову систему даних. Ці запити розпочинають процес роботи з сервісами – зв’язуються з необхідним сервісами та передають:

– запит на інформацію про медичні послуги, який розпочинає роботу з серверною частиною, відправляється на сервер, отримана на цей запит відповідь обробляється та повертається у головну View Model;

– (у випадку отримання відповіді від серверу) відповідь на запит інформації про медичні послуги, яка передається в сервіс відповідальний за кешування даних, де вона записується в файл для використання у випадку втрати зв’язку з мережею інтернет;

– (у випадку втрати зв’язку з Інтернетом) запит на інформацію про медичні послуги, який передається в сервіс відповідальний за кешування даних, де відповідь зчитується з даних записаних в файлову систему та повертається у головну View Mode;

– стан додатку, який передається в центр збору аналітики, аналізують, після чого дані відправляються через мережу інтернет.

З відповіді від сервісів формується пакет даних, який головна View Model передає в головний View Controller. Головний View Controller заповнює екран даними.

Якщо користувач обирає таб пошуку і вводить туди текст, то головний View Controller передає цей текст в головну View Model, що запускає запит на пошук, а відповідь на нього(список лікарень) передає в головний View Controller для оновлення даних на екрані. Варто зауважити, що якщо відповідь була отримана від серверу, то вона кешується. А у випадку втрати зв’язку з мережею то раніше записана відповідь на запит зчитується з файлової системи.

Після того як користувач, обирає певну певну лікарню головний View Controller передає її в головну View Model. Головна View Model надсилає запит на отримання детальної інформації про певну лікарню, який кешується або, за потреби, зчитується з кешу. Після чого головна View Model передає в Coordinator

команду запуску наступного екрану і той ініціалізує та запускає Coordinator екрану про лікарню.

Якщо користувач обирає певний тип медичних послуг, головний View Controller передає цей тип в головну View Model, що в свою чергу передає в головний Coordinator і той ініціалізує і запускає Coordinator екрану зі списком медичних послуг.

Coordinator екрану зі списком медичних послуг створює View Model екрану зі списком медичних послуг та View Controller екрану зі списком медичних послуг.

Після того як користувач, обирає певну медичну послугу View Controller передає її в View Model.View Model надсилає запит на отримання всіх лікарень(який кешується або, за потреби, зчитується з кешу), що надають певну медичну послугу та передає стан додатку. Після чого View Model передає в Coordinatorкоманду запуску наступного екрану.

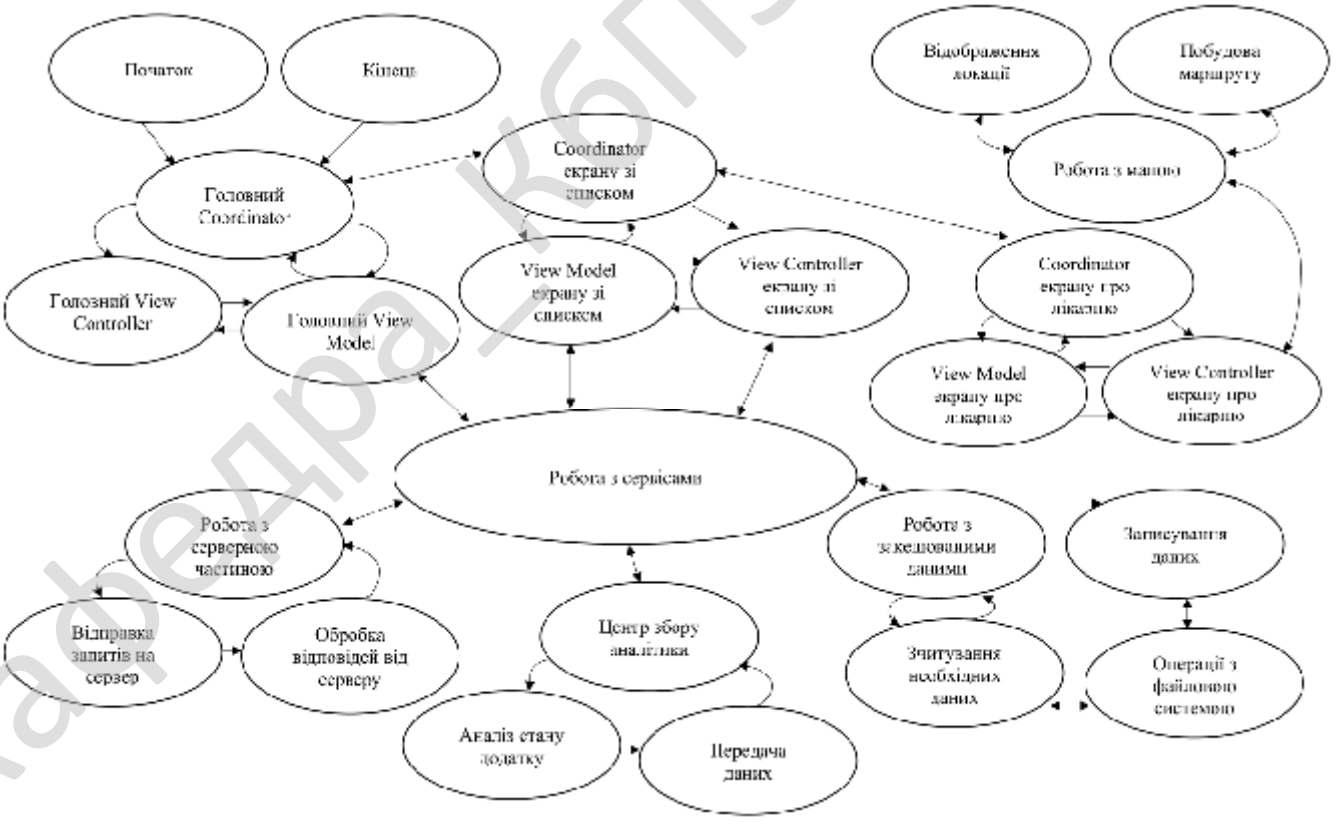

Рисунок 3.3 – Діаграма узагальнених процесів удосконаленої муніципальної ІС медичних послуг

Coordinator екрану зі списком медичних послуг ініціалізує та запускає Coordinator екрану зі списком лікарень. Всі процесів екрану зі списком лікарень аналогічні до екрану зі списком медичних послуг, за винятком запиту який надсилає View Model після вибору користувача. View Model екрану зі списком лікарень надсилає запит на отримання детальної інформації про певну лікарню, який кешується або, за потреби, зчитується з кешу. Coordinator екрану зі списком лікарень ініціалізує та запускає Coordinator екрану про лікарню.

Coordinator екрану про лікарню ініціалізує та запускає View Model та View Controller. На початку роботи View Controller екрану про лікарню розпочинає процес роботи з мапою, а саме відображення локації лікарні.

Якщо користувач вибере опцію побудувати маршрут, то View Controller отримає від View Model екрану про лікарню посилання і розпочне другий процес роботи з мапою, побудує маршрут.

Якщо користувач захоче повернутися до попереднього екрану, то View Controller будь-якого екрану передає це в View Model, та в Coordinator, котрий завершує роботу екрану та повертає керування класу, яким був запущений.

Діаграма, зображена на рисунку 3.3, розроблена з деяким рівнем абстракції, проте повністю відображає всі основні процеси життєвого циклу ПЗ клієнту удосконаленої муніципальної інформаційної системи медичних послуг.

Висновки до розділу 3

Під час планування та опису проектних рішень був визначений функціонал удосконаленої муніципальної інформаційної системи медичних послуг, на основі якого була підібрана клієнт-серверна архітектура для реалізації проекту та визначені головні компоненти – клієнт та група серверних елементів.

В ході розробки структурної схеми були визначені всі структурні елементи системи (для клієнта: UI, бізнес-логіка, навігація, аналітика, Network, робота з кешованими даними (при використанні файлової системи девайсу) та з мапами»;

для сервера: Google Maps, Firebase та сервер, що звертається до БД медичних послуг) та зв’язки між ними, після чого при розробці функціональної схеми на основі структурних елементів клієнта були розписані їх основні функції та була проведена деталізація їх взаємозв’язків. Завершенням стала робота над побудовою узагальненої діаграми процесів, що є логічного обгрунтованою за допомогою функціональної схеми та надає представлення про весь робочий цикл ПЗ клієнта.

В результаті роботи над проектними рішеннями розроблені структурна і функціональна схеми та узагальнена діаграма процесів для удосконаленої муніципальної інформаційної системи медичних послуг, а також повністю описані всі майбутні елементи, зв’язки та функціонал ІС, які стануть основою для проектних рішень та алгоритмів.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Після того, як були прийняті основні проектні рішення, визначені структурні елементи та майбутній функціонал ПЗ, логічним наступним кроком є проведення розрахунків та їх перевірка. В ході цих розрахунків необхідно, використовуючи діаграму процесів та функціональну схему, розробити детальні алгоритми роботи як усієї системи так і окремих її сервісів з урахуванням усіх потреб захисту ПЗ.

Перед початком роботи над алгоритмами для реалізації ПЗ потрібно виконати всі розрахунки та експериментально підтвердити вірність обраних проектних рішень. Відповідно до специфіки розроблюваного ПЗ можливо виконати розрахунок правил для генерації посилань в GoogleMaps та перевірку роботи апі, що з'єднує додаток-клієнт та сервер з БД медичних послуг.

По-перше, згідно з документацією GoogleMaps посилання на побудову маршруту має генеруватися за такими правилами:

https://maps.google.com/?saddr= x&daddr=y&directionsmode=z,

де: *x* – початкова точка маршруту (якщо вона не задана то використовуватиметься поточна геолокація), може бути задана використовуючи координати або відформатовану адресу;

y – кінцева точка маршруту, тип даних такий самий як і *x*;

z – тип транспорту: *driving, transit, bicycling or walking*.

Таким чином отримано формулу, що буде основою для алгоритму генерації посилань в додатку. В якості експерименту створено посилання на побудову маршруту та перевірено його в браузері, для кінцевої точки

використано координати лікарні з попереднього експерименту. Було побудовано посилання: https://maps.google.com/?saddr=&daddr=48.542429,32.260350&directionsmode=driving. Результат перевірки підтвердив правильність проектного рішення та зображений на рисунку 4.1.

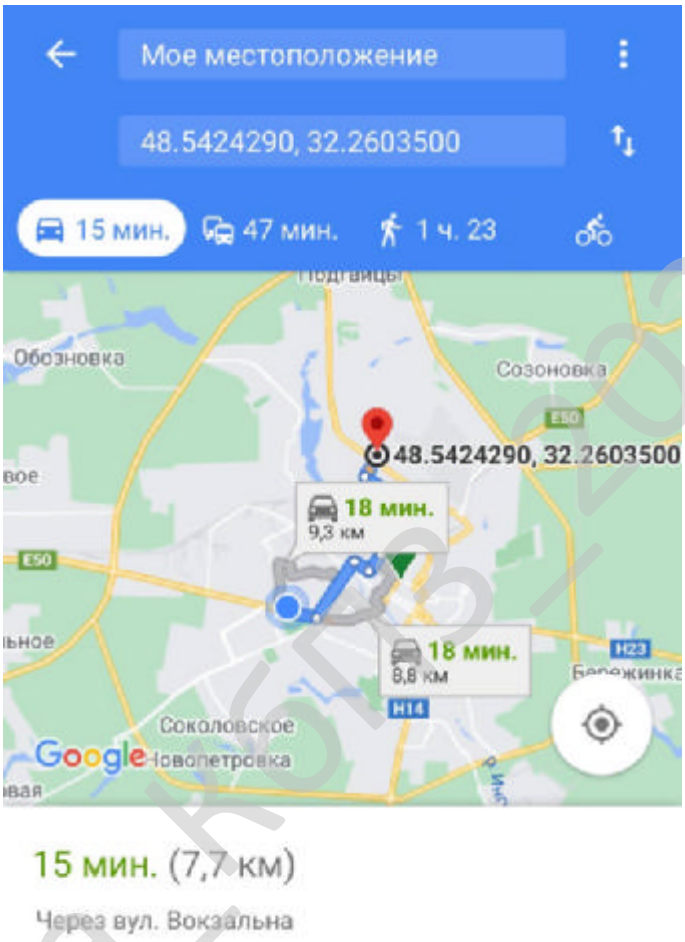

Рисунок 4.1 – Скріншот Google Maps

По-друге, відповідно до заплановано інтерфейсу роботи з серверною частиною за допомогою спеціальної програми Postman, що дозволяє відслідковувати відповіді на відправлені запити, проведено експеримент, що дозволив перевірити стабільність та відлагодженість клієнт-серверного зв'язку. Для експеремента створено запит на отримання повної інформації про лікарню: https://medical.kr-rada.gov.ua/api/hospital/472cf05208fcf4c9f1cf473de68904db. Результат перевірки підтвердив правильність проектного рішення та зображений на рисунку 4.2.

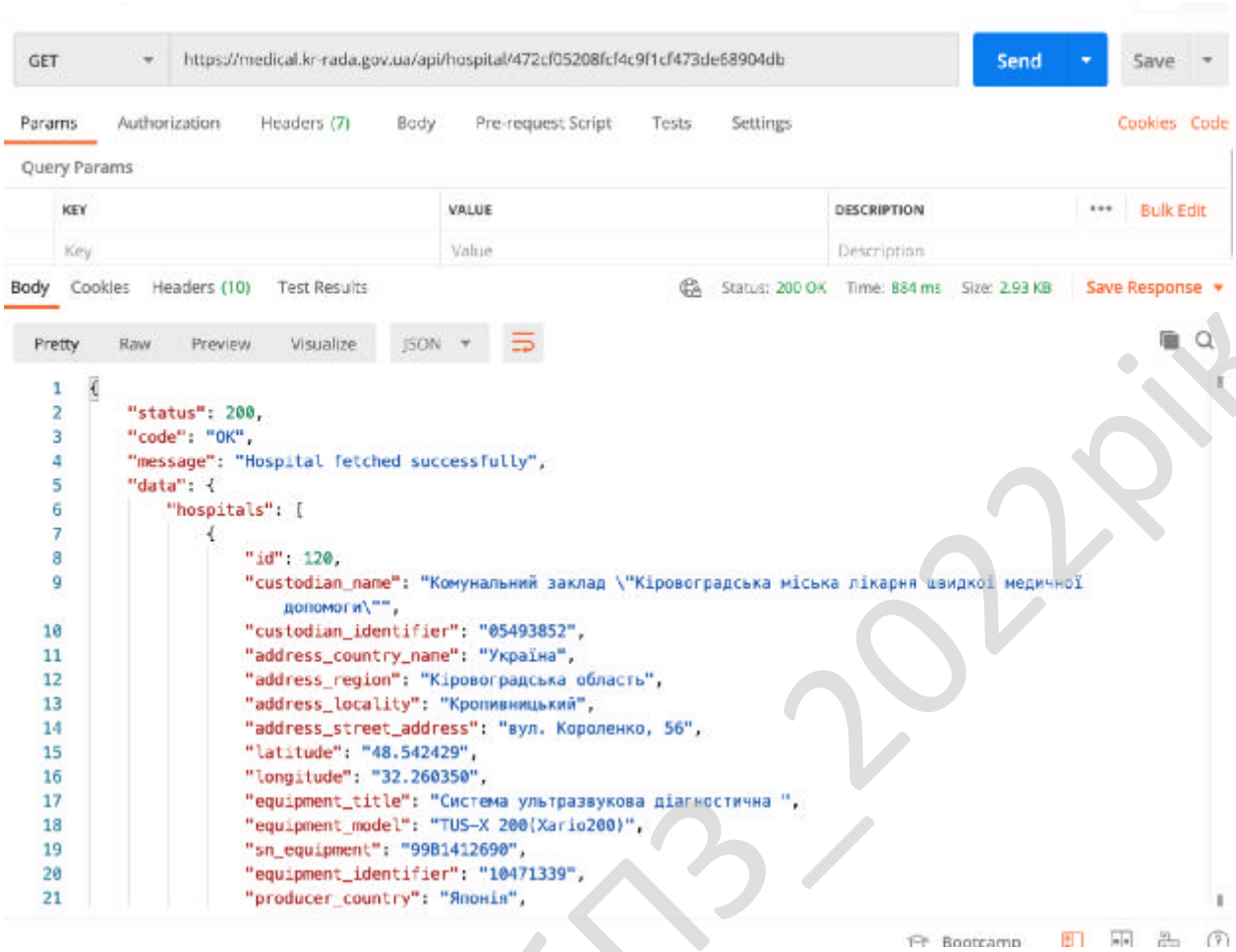

Рисунок 4.2 – Скріншот Postman

В результаті всіх розрахунків та експериментів, отримано актуальну формулу для подальшої розробки алгоритмів для роботи з мапою та підтверджено стабільність та ефективність клієнт-серверної архітектури муніципальної інформаційної системи медичних послуг.

Узагальнена діаграма процесів та функціональна схема з розділу 3 зображує роботу всіх екранів (їх View Model) з різними сервісами. Саме використання сервісів є головною частиною роботи ПЗ і темою основного алгоритму ПЗ клієнта муніципальної інформаційної системи медичних послуг.

Початок алгоритму – запуск додатку, після якого має відбуватися запит отримання списку медичних послуг, на основі відповіді повинен оновлюватися зміст екрану. На цьому етапі подальше функціонування додатку залежить від того, яку інформацію користувач буду вводити. Якщо користувач обере певний

тип медичний послуг, то мають бути відправлені аналітичні дані та другий запит, а саме запит на список всіх медичних послуг даної категорії. На основі відповіді має оновлюватися зміст екрану і додаток повинен очікувати наступних дій користувача. Якщо ж користувач обере швидкий пошук лікарні, то відбудеться відправка аналітичних даних. Проте, незалежно від того, який вибір зробить користувач, наступним кроком алгоритму має бути виклик запиту на отримання список лікарень. Після цього по-черзі будуть виконуватися наступні дії: виведення даних на екран, введення користувачем даних (а саме вибір конкретної лікарні), відправка аналітичних даних, запит на отримання детальної інформації про лікарню, виведення даних на екран, відображення місцезнаходження лікарні на мапі та введення даних. Залежно від того, чи обере користувач при останньому введенні даних побудову маршруту, повинен бути запущеним процес побудови маршруту. На цьому виконання узагальненого алгоритму ПЗ клієнта муніципальної інформаційної системи медичних послуг має завершитися.

В ході роботи над удосконаленням роботи даної програми було додано підпрограму, що при кожному запиті інформації виконує перевірку на наявність підключення до мережі Інтернет, за залежно від цього запускає або підпрограму запиту на сервер або в файлову систему.

Графічно узагальненого алгоритму ПЗ зображений за допомогою блок-схеми на рисунку 4.3 (варто зауважити, що розроблену загальну блок-схему розроблено без урахування процесів навігації та зміни керуючого контролеру), а алгоритм обробки інформаційного запиту на рисунку 4.4.

Усі наступні алгоритми будуть описувати роботу функціональних елементів ПЗ. По-перше, це робота з серверною частиною – відправка запитів та отримання відповідей. Якщо описувати алгоритм, то він починається після запуску Network сервісу, який очікує на введення даних про медичну послугу. На основі отриманих даних повинен створюватися запит, який потім має відправлятися на сервер. Якщо сервер не відповідатиме, то це стане завершенням роботи даного алгоритму. Якщо ж сервіс відповість, то отримана відповідь

повинна перевірятися на наявність помилок та можливість розпарсити її в очікувані моделі даних. При знаходженні чи виникненні критичних помилок в процесі парсингу, має відбуватися виведення помилки, а при успішному завершенні парсингу даних – виведення отриманої моделі. На цьому алгоритм повинен завершувати свою роботу.

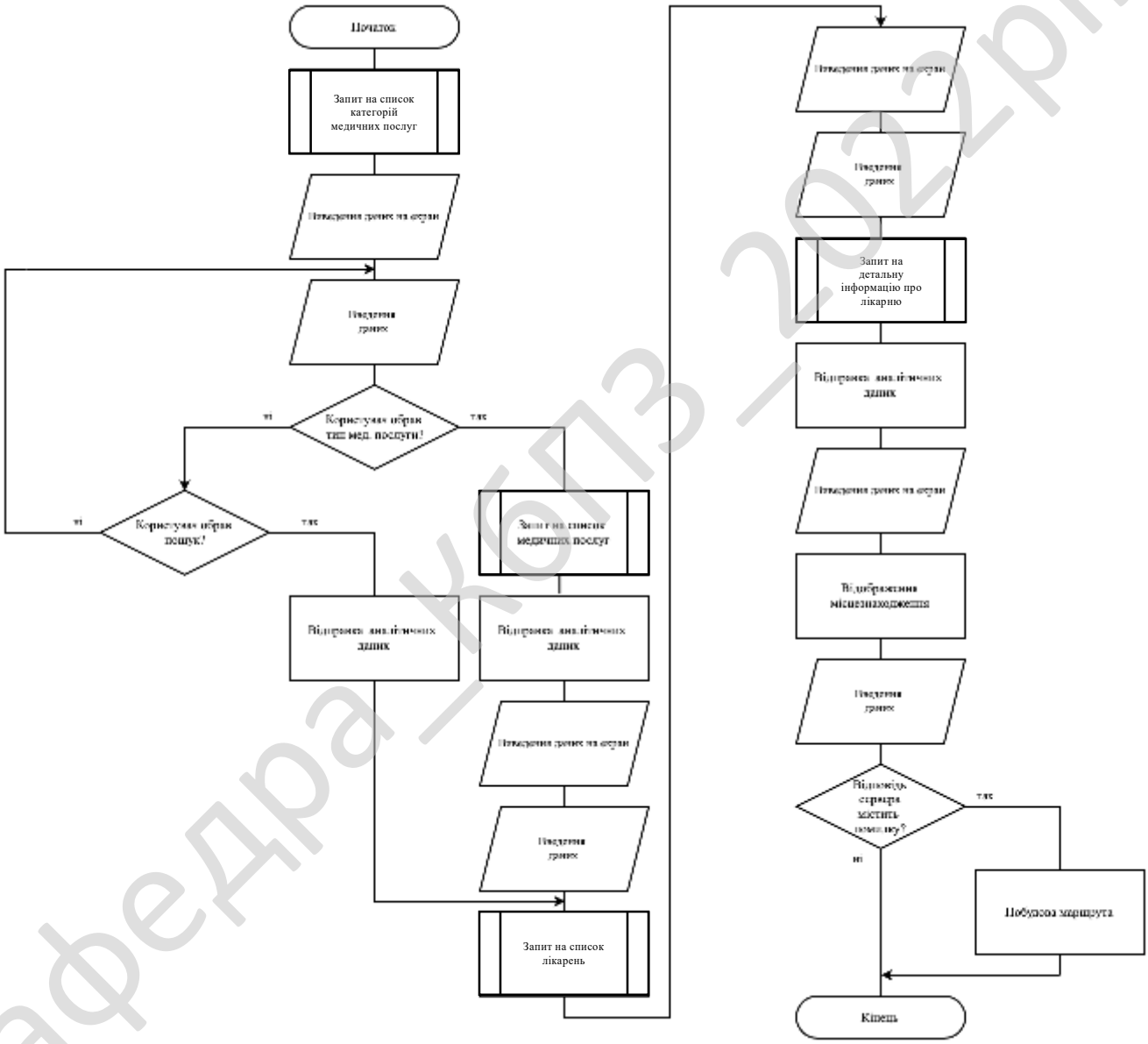

Рисунок 4.3 – Узагальнена блок-схема алгоритму функціонування ПЗ клієнта удосконаленої муніципальної інформаційної системи медичних послуг

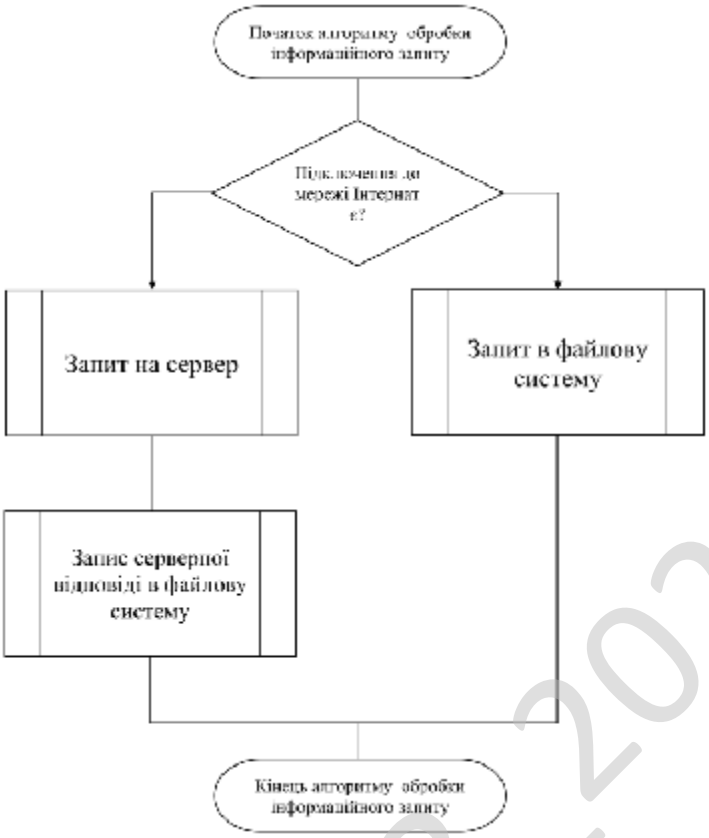

Рисунок 4.4 – Блок-схема алгоритму обробки інформаційного запиту

Блок-схема на рисунку 4.5 графічно представляє алгоритм роботи сервісу, що відповідає за роботу з серверною частиною. Ця блок-схема описує відправлення запиту на отримання списку лікарень, що надають певну медичну послугу, та обробку відповіді від серверу. Проте, єдине чим цей алгоритм відрізняється від алгоритмів для інших запитів – це даними, що вводяться, та типом очікуваної відповіді, загальний процес взаємодії з сервером є однаковим для всіх запитів.

По-друге, це робота з мапою. Цей сервіс використовує одразу два алгоритми, тому що виконує два різних види задач з використанням різних інструментів. На наведених блок-схемах зображені функції алгоритмів роботи з мапою. На рисунку 4.6 – відображення місцезнаходження на мапі та на рисунку 4.7 – побудова маршруту за допомогою геолокації. Необхідно відмітити, що перший алгоритм описує роботу з вбудованим компонентом, а другий генерацію посилання для роботи з стороннім сервісом за допомогою мережі Інтернет та при

використанні браузера.

По-третє, це центр збору аналітичних даних. Алгоритм роботи даного сервісу, що базується на аналізі поточного стану додатку та відправленні результатів аналізу до стороннього хмарного сервісу за допомогою мережі Інтернет. Графічно цей алгоритм зображений за допомогою блок-схеми на рисунку 4.8.

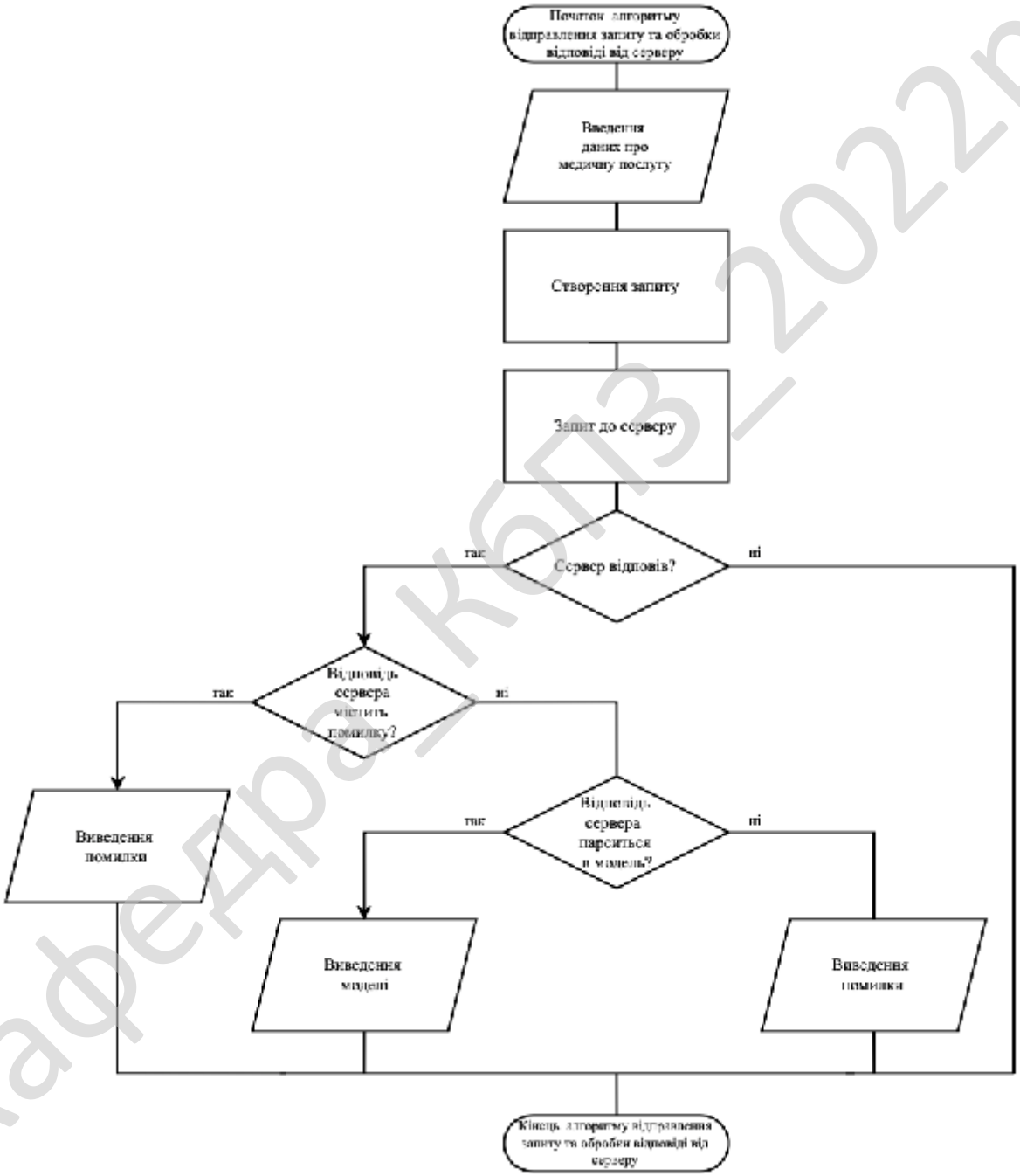

Рисунок 4.5 – Блок-схема алгоритму відправлення запиту та обробки відповіді від серверу

Рисунок 4.6 – Блок-схема алгоритму відображення місцезнаходження на мапі

Рисунок 4.7 – Блок-схема алгоритму побудови маршруту за допомогою геолокації

Рисунок 4.8 – Блок-схема алгоритму роботи центру збору аналітичних даних

І останній – це сервіс відповідальний за роботу з кешем, його алгоритми відображають роботу з файловою системою мобільного пристрою. Цей сервіс представлений у вигляді двох алгоритмів тому що виконує два види операцій – запис серверної відповіді в файл та зчитування відповіді на запит з файлу у випадку втрати зв'язку інтернетом. Графічно алгоритми зображений за допомогою блок-схеми на рисунку 4.9 і 4.10.

В результаті розробки блок-схем та опису алгоритмів функціонування системи визначено алгоритми основних сервісів додатку – Network (рисунок 4.2),

блоку роботи з мапою (рисунок 4.5 – 4.6), центру збору аналітичних даних (рисунок 4.8) та сервісу відповідального за роботу з кешем (рисунок 4.9 – 4.10). А головне описаний узагальнений алгоритм, який вказує на те як додаток повинен взаємодіяти з усіма сервісами.

Рисунок 4.9 – Блок-схема алгоритму зчитування відповіді на запит з файлу у випадку втрати зв'язку інтернетом

Рисунок 4.10 – Блок-схема алгоритму запису серверної відповіді в файл

4.2 Захист розробленого програмного забезпечення

Захист програмного забезпечення, що виступає клієнтом, в більшості випадків пов'язаний з роботою з серверною частиною – це може бути або захист зв'язку або особливі кодування та декодування даних. У випадку, коли є три окремих серверних елементи, роботу з кожним треба розглянути окремо.

По-перше, сервер, що отримує дані від БД медичних послуг, має зв'язок захищений за допомогою спеціального протоколу зв'язку (HTTPS), а інформація передається у вигляді JSON файлу без додаткових кодувань. Проте, варто зауважити, що в багатьох інших клієнт-серверних системах (а можливо і в цій системі в майбутньому) використовується кодування усієї відповіді, або певної її частини. Найпопулярнішою бібліотекою для роботи з кодуванням та декодуванням даних при розробці в XCode є CommonCrypto, що підтримує усі найвідоміші типи кодувань – MD5, SHA-256 та інші.

Приклад програмного коду використання бібліотеки CommonCrypto:

```
import CommonCrypto

func md5(str: String) -> String {
    if let strData = str.data(using: String.Encoding.utf8) {
        var digest = [UInt8](repeating: 0, count:Int(CC_MD5_DIGEST_LENGTH))
        strData.withUnsafeBytes {
            CC_MD5($0.baseAddress, UInt32(strData.count), &digest)
        }
        var md5String = ""
        for byte in digest {
            md5String += String(format:"%02x", UInt8(byte))
        }
        return md5String
    }
    return ""
}

func sha256(str: String) -> String {
```

```
            if let strData = str.data(using: String.Encoding.utf8) {
                var digest = [UInt8](repeating: 0,
count:Int(CC_SHA256_DIGEST_LENGTH))
                strData.withUnsafeBytes {
                    CC_SHA256($0.baseAddress, UInt32(strData.count), &digest)
                }
                var sha256String = ""
                for byte in digest {
                    sha256String += String(format:"%02x", UInt8(byte))
                }
                return sha256String
            }
            return ""
        }
```

По-друге, центр збору аналітичних – зв'язок з ним захищений за допомогою GoogleService-Info.plist, що містить API_KEY. Цей файл створено за допомогою сервісу Firebase та інтегровано в ПЗ, скріншот цього файлу представлений на рисунку 4.11.

Рисунок 4.11 – Скріншот GoogleService-Info.plist

По-третє, Google Maps, що надає доступ до інтерактивних мап через браузер, не має ніяких захистних кодувань та протоколів, тому генерація посилання до нього не потребує додаткових дій.

Висновки до розділу 4

Першим етапом четвертого розділу була робота над усіма необхідними розрахунками та експериментальна перевірка правильності отриманих результатів. Таким чином отримано актуальну формулу роботи з GoogleMaps та перевірена робота клієнт-серверної архітектури муніципальної інформаційної системи медичних послуг.

Перед початком робіт по реалізації проекту були перевірені, розроблені та описані алгоритми, а саме – узагальнений алгоритм всього додатку та алгоритми для роботи сервісів. Окрім, алгоритмів також були зплановані механізми захисту ПЗ, що були використані при роботі з серверною чатиною, а саме – захист зв'язку через мережу Інтернет.

Після завершення підготовчого етапу, була виконана робота по реалізації проекту. Відповідно до визначеної архітектури додатку розроблено класи View Controller, Coordinator та ViewModel, та налаштовано взаємодію компонентів системи. Згідно з усіма описаними алгоритмами були створені класи сервісів. А також, спираючись на структурну та функціональні схеми реалізовано зв'язні, системні та програмні інтерфейси, в результаті яких вдалося зкомпонувати всі частин ІС.

Цей програмний продукт повністю готовий до подальшого впровадження в промислову експлуатацію.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для впровадження офлайн режиму мобільного додатку муніципальної інформаційної системи медичних послуг необхідно зробити його загальнодоступним для використання серед жителів міста. Єдиний спосіб опублікувати нову версію додатку для всіх користувачів — завантажити її в App Store. App Store — це інформаційна платформа, що розроблена і підтримувана Apple Inc. для поширення комп'ютерних програм та мобільних застосунків на своїх операційних системах.

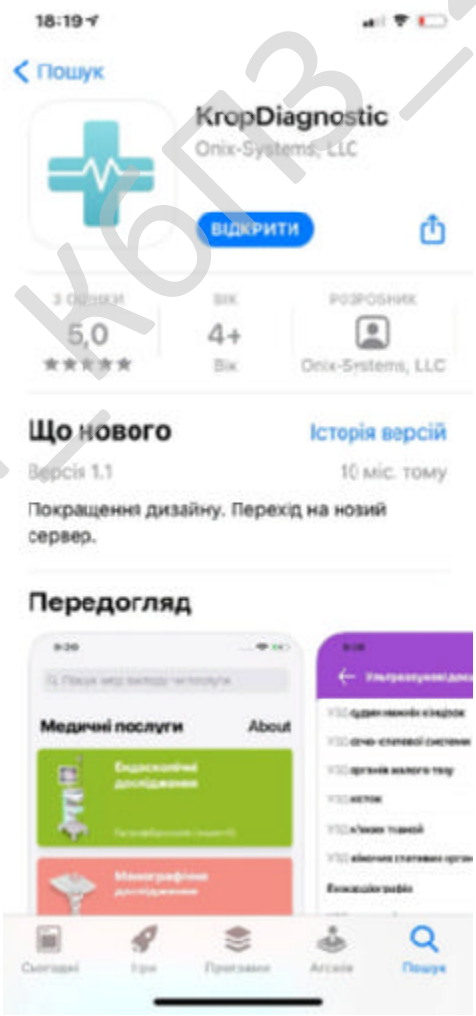

Рисунок 5.1 – Екран Krop Diagnost в App Store

Щоб опублікувати програму в App Store необхідно:

1) створити профіль розробника та створити модель майбутнього додатку;

2) за допомогою Xcode IDE зібрати архів та відправити його через в профіль в AppStore;

3) описати майбутню публікацію та надіслати додаток на перевірку в App Store;

4) після успішної перевірки опублікувати програму, у випадку якщо додаток не пройде перевірку необхідно внести всі необхідні корективи (вони будуть надіслані App Store в профіль) та повернутися на крок 2.

5.1 Інструкція користувача розробленого програмного забезпечення

Муніципальна інформаційна система медичних послуг функціонує на основі інформації про медичні послуги всіх лікарень міста, що постійно оновлюється. Саме ця інформація відображається в мобільному додатку.

На головному екрані додатку представлений список категорій медичних послуг (рисунок 5.2), доступ до даних про виробника ПЗ (рисунок 5.3) та пошук мед. закладу по назві (рисунок 5.4). В результаті пошуку відбувається перехід на екран зі списом лікарень, що їх надають.

При виборі певної категорії відбувається перехід на екран зі списком медичних послуг (рисунок 5.5). Потім виборі певної послуги відбувається перехід на екран зі списком лікарень (рисунок 5.6), що їх надають. Після вибору певної лікарні відбувається перехід на екран з інформацією про лікарню (рисунок 5.7) та медичну послугу, а також мапою з позначенням місцезнаходження лікарні. На цьому екрані доступна побудова маршруту до лікарні.

Додаток має одну основну функцію — це відповідно до потреб користувача надати повну інформацію про лікарню, де є необхідна медична послуга.

Користувач може скористатися двома сценаріями. Перший це пошук через категорії:

- на головному екрані (рисунок 5.2) обрати категорію;
- на екрані зі списком медичних послуг (рисунок 5.5) вибрати медичну послугу або якщо категорія була обрана не правильно натиснути кнопку в верхньому лівому кутку та повернутися на головний екран та знову виконати крок 1;

Рисунок. 5.2 – Головний екран додатку

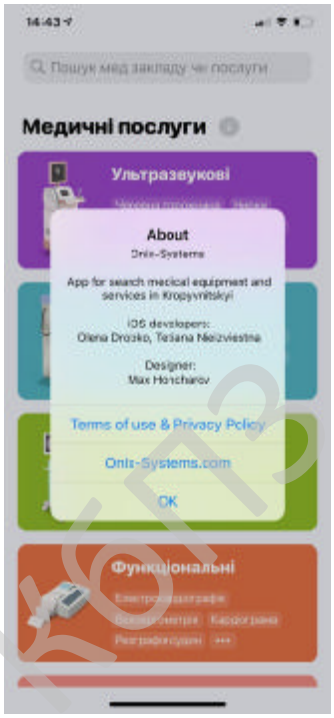

Рисунок. 5.3 – Дані про виробника ПЗ

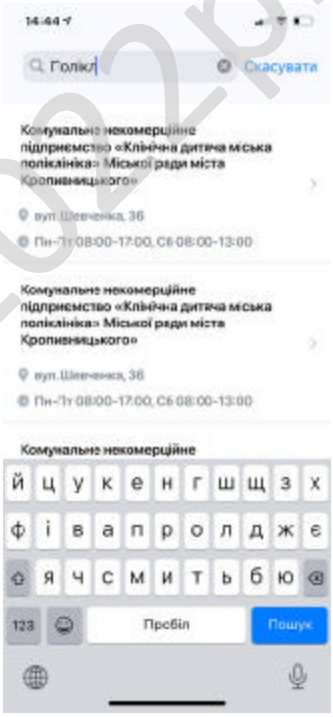

Рисунок. 5.4 – Пошук мед. закладу по назві

- на екрані зі списком лікарень (рисунок 5.6) обрати лікарню або якщо медична послуга була обрана не правильно натиснути кнопку в верхньому лівому кутку та повернутися на головний екран та знову виконати крок 2;
- на екрані з детальною інформацією про лікарню (рисунок 5.7) за необхідності можна побудувати маршрут, для цього треба натиснути на кнопку «Подивитися на мапі».

Другий сценарій — це швидкий пошук за назвою медичної послуги чи лікарні:

1) на головному екрані (рисунок 5.2) обрати пошук (зверху над списком категорій;

2) на екрані пошуку (рисунок 5.4) ввести назву лікарні або медичної послуги або якщо необхідно завершити пошук натиснути кнопку «Скасувати» та повернутися на головний екран;

3) на екрані пошуку (рисунок 5.4) обрати лікарню;

4) на екрані пошуку (рисунок 5.4) обрати лікарню;

5) на екрані з детальною інформацією про лікарню (рисунок 5.7) за необхідності можна побудувати маршрут, для цього треба натиснути на кнопку «Подивитися на мапі».

Окремо слід зауважити, що всі дії користувача записуються та використовуються для аналітичного аналізу потреб містян.

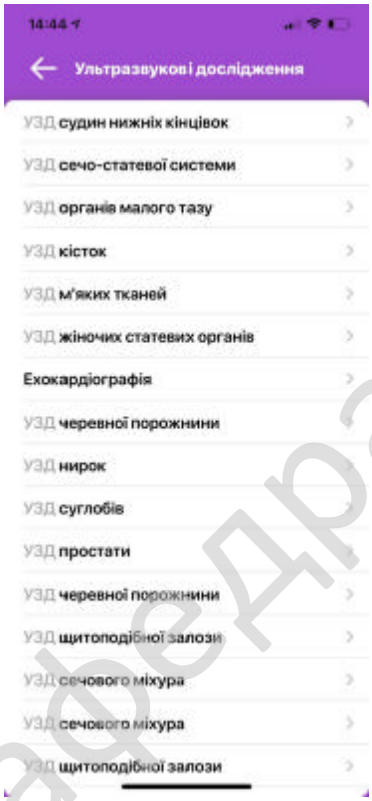

Рисунок. 5.5 – Екран зі списком медичних послуг

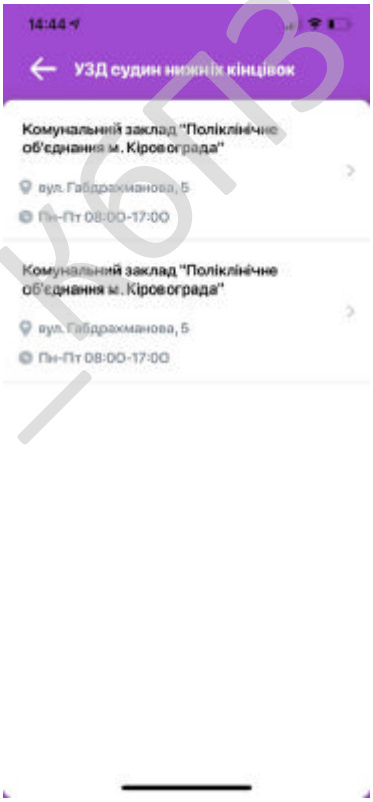

Рисунок. 5.6 – Екран зі списком лікарень

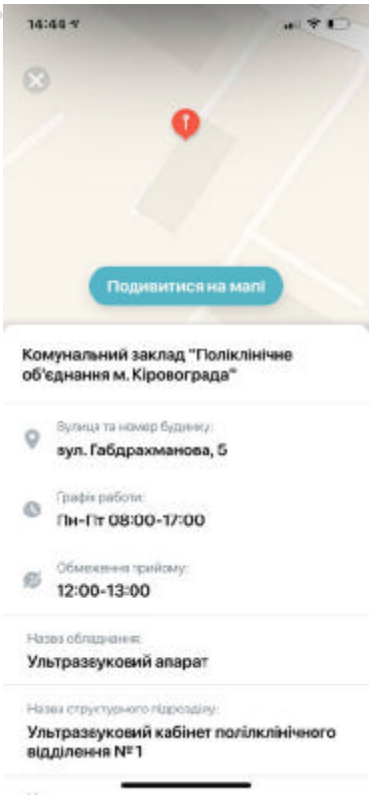

Рисунок. 5.7 – Екран з інформацією про лікарню

Висновки до розділу 5

В ході впровадження покращень системи в експлуатацію були проаналізовані доступні для виконання цієї задачі інструменти і виявлено, що для публікації додатків під iOS пристроїв існує лише одна платформа – це App Store.

Окрім того, було розроблено інструкцію для користувача, що описує функціонал додатку, механіку використання додатку та містить всі необхідні ілюстрації та пояснення для двох сценаріїв використання додатку: пошук через категорії та швидкий пошук за назвою медичної послуги чи лікарні.

6 НАУКОВА НОВИЗНА

Науково-дослідницька робота є важливою частиною магістерської дипломної роботи та демонструє рівень знань та розвитку студента. Проте, найважливішим є виділення отриманого результату в нове знання. Загалом наукову новизну можна поділити на теоретичну та практичну за значенням результатів дослідження та на три типи за рівнем новизни:

1) перетворення відомих даних, докорінна їх зміна;

2) розширення, доповнення відомих даних;

3) уточнення, конкретизація відомих даних, поширення відомих результатів на новий клас об’єктів, систем.

В даній дипломній роботі науково-дослідницька робота спрямована на отримання практичного результату, другого та третього рівня новизни. Метою дослідження було покращення роботи муніципальної інформаційної системи медичних послуг. Джерелами інформації виступали публікації з питань покращення роботи ІС та iOS додатків, аналіз роботи інших муніципальних інформаційних систем [див. розділ 2] та статистика використання додатку “KropDiagnost” зібрана за допомогою за допомогою платформи Firebase та App Store.

Дослідження було розпочато з аналізу аналітики роботи додатку, в результаті якого було виявлено, що популярність використання муніципальної інформаційної системи медичних послуг набагато нижча ніж очікувалось, враховуючи той факт, що дана система не має аналогів та є загально необхідною – можна зробити висновок, що причина в роботі МІС. І тоді пошук рішення вимагав ознайомитися з іншими науковими роботами, що досліджували це питання. Вивчення публікацій та статей про покращення роботи iOS додатків допомогли виявити невикористаний потенціал клієнту муніципальної інформаційної системи медичних послуг.

Наукова новизна роботи полягає у вдосконаленні моделі муніципальної інформаційної систем медичних послуг за рахунок реалізації процесу офлайнового режиму функціонування системи, що на відміну від існуючих моделей муніципальних систем забезпечує доступ до даних ІС у період відсутності зв'язку з Інтернетом.

Задля реалізації офлайн режиму муніципальної інформаційної систем медичних послуг, клієнт даної ІС, що представлений у вигляді мобільного додатку, має бути доповнений відповідним сервісом та базою даних. Новий сервіс буде використовувати базу даних для зберігання відповідей від серверу, та, у випадку втрати відключення до мережі інтернет, буде забезпечувати додаток інформацією для роботи.

Висновки до розділу 6

В ході роботи над магістерською дипломною роботою було проведене наукове дослідження задля покращення роботи муніципальної інформаційної системи медичних послуг. Джерелами інформації виступили статистичні дані, аналіз аналогів та публікації на пов'язані теми. В результаті отримано наукову новизну, що удосконалює роботу МІС медичних послуг м. Кропивницький

Результати роботи апробовані й схвалені на IV міжнародній науково-практичній конференції «Інформаційна безпека та комп'ютерні технології» (Кропивницький, 2021), на II всеукраїнській науково-практичній конференції здобувачів вищої освіти й молодих учених «Комп'ютерна інженерія і кібербезпека: досягнення та інновації» (Кропивницький, 2020), на LV науково-технічній конференції здобувачів вищої освіти «Наука в ЦНТУ: основні досягнення та перспективи розвитку» за підсумками проведення «Дня науки – 2021» (Кропивницький, 2021), а також опубліковані у збірниках матеріалів означених науково-практичних конференцій.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми дипломного проекту

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 80 днів (чотири місяці).

В магістерській роботі були проведені дослідження та виконана програмна реалізація мобільного iOS-клієнта удосконаленої муніципальної інформаційної системи медичних послуг.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп’ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	необмежено
3. Запланований термін розробки, днів	Fpq	80 (4 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Г
6. Складність алгоритму (1, 2, 3)	–	3

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	2
8. Кількість форм вихідної інформації.	–	2
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	6
11. Гнучкість проекту ПП (1-6)	–	6
12. Детальність проекту ПП (1-6)	–	6
13. Рівень спрацьованості колективу (1-6)	–	6
14. Ступінь вимірності процесів (1-6)	–	6
15. Необхідна надійність програмного забезпечення (1-6)	–	1
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	3
17. Складність кінцевого програмного продукту (1-6)	–	1
18. Необхідний рівень забезпечення повторного використання (1-6)	–	1
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	1
20. Вимоги до швидкодії ПП (1-6)	–	1
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	1
23. Професійний рівень аналітиків (1-6)	–	6
24. Професійний рівень програмістів (1-6)	–	6
25. Постійність складу команди розробників (1-6)	–	6
26. Досвід розробки додатків (1-6)	–	6
27. Досвід роботи з обчислювальною платформою (1-6)	–	6

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	6
29. Досвід роботи з програмними інструментами розробки (1-6)	–	6
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	6
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	3
32. Вартість ПЗ у розробника (НМА), грн.	–	-
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП як одна з найбільш тривалих і трудомістких робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B, \qquad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \qquad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(0+0+0+0+0) = 1{,}01.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}01} = 6{,}68 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \qquad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}68 \cdot (0{,}75 \cdot 0{,}93 \cdot 0{,}75 \cdot 0{,}91 \cdot 0{,}89 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 0{,}67 \cdot 0{,}74 \cdot 0{,}84 \cdot 0{,}81 \cdot$$
$$\cdot 0{,}81 \cdot 0{,}84 \cdot 0{,}72 \cdot 0{,}78 \cdot 1) = 0{,}32 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \qquad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 3{,}23 \cdot 0{,}32^{0{,}33+0{,}2(1{,}01-1{,}01)} \cdot 25 = 17 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	4	МВ Додаток Е
Ескізний проект	5	МВ Додаток Ж
Технічний проект	6	МВ Додаток З
Робочий проект	17	Ф 7.1-7.4
Впровадження	10	МВ Додаток О
Всього	42	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \qquad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{42 \cdot 1}{80\text{-}5} = 0,56 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	*1*	*90*	*1,5*
Монітор	60	1	*60*	*1*
Клавіатура	30	1	*30*	*0,5*
Маніпулятор «мишка»	30	1	*30*	*0,5*
Принтер матричний	60	*0*	*0*	*0,0*
Принтер лазерний	120	*1*	*120*	*2*
Принтер струминний	60	*0*	*0*	*0,0*
Сканер	20	*0*	*0*	*0,0*
Концентратор–маршрутизатор	30	*1*	*30*	*0,5*
Кабельні господарства ЛВС на 1 м. п.	2,5	*50*	*125*	*2,08*
Копіювальний апарат	140	*0*	*0*	*0,0*
Усього за рік:			$З_ч$	8,08

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{8,08 \cdot 4}{1,2} = 29 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = 29/(80 \cdot 8) = 0{,}05 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків. Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2012 R2, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	0,1	0,05
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,05	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,05	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	0,2	
Всього		0,4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	0,3	0,1
	Створення графічних і стилістичних елементів сайту	0,2	
	Оформлення банерів і промо-сторінок	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,1	
Всього		0,8	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	0,24	18000	17280
Інженер-програміст	0,56	18000	40320
Інженер-електронщик	0,05	17000	3400
Адміністратор мережі	0,05	17000	3400
Дизайнер WEB	0,1	17000	6800
Всього за період розробки	$R_{сп} = 1$	-	$Ф_{роб} = 71200$

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рq}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{71200}{1 \cdot 80} = 890 \text{ грн.}$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 13 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 1 \cdot 8 \cdot 20000 = 160000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 16000 грн.

Балансова вартість інвентарю розраховується за нормою 25000 грн. на одне робоче місце. Тобто:

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 1 \cdot 25000 = 25000 \text{ грн.}$$

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Мобильная рабочая станция ThinkPad P70 / 17.3" (1920x1080) IPS / Intel Core i7-6700HQ (4 (8) ядра по 2.6 – 3.5 GHz) / 16 GB DDR4 / 128 GB SSD + 1000 GB HDD / nVidia Quadro M600M, 2 GB GDDR5, 128-bit / WebCam / DVD-ROM		31181
Принтер лазерний	HP LaserJet Enterprise P3015DN	5351

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	1	31181	3118,1	34299,14
Принтер лаз.	1	5351	535,1	5886,1
Всього	–	–	–	40185,24

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет-магазину Компбест за 21.11.22 – джерело https://compbest.com.ua.

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	160000	-	-
2. Передавальні пристрої	16000	-	-
Всього по групі	176000	5	8800
Група 4			
3. Обчислювальна техніка	40185	-	-
Всього по групі	40185	50	20092,5
Група 5, 6			
4. Вимірювальні пристрої	не передбачено	25	0,0
5. Транспортні засоби	не передбачено	20	0,0
6. Господарський інвентар	25000	25	6250
Всього по групі 5,6	25000	-	6250
Нематеріальні активи			
7. Нематеріальні активи	не передбачено	10	0,0
Разом	$К_р$ = 241185		$А_р$ = 35143

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \qquad (7.11)$$

де: N_e – кількість екземплярів програм, шт.

$$З_о = 890 \cdot 42/1 = 37380 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 37380 \cdot 10 \cdot 0{,}01 = 3738 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(37380+3738) = 9046 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 37380 \cdot 15 \cdot 0{,}01 = 5607 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.; $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.; $З_{М3}$ – вартість фарби, картріджей, тонеру, грн.; $N_е$ – кількість екземплярів програм, шт.

Кількість паперу на період розробки визначаємо по середній його витраті за попередній період (одна пачка на місяць). Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=210 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 210 \cdot 4 = 840 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих

пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (програма опублікована в додатках «App Store» тому вартість CD/DVD не враховуємо):

$$З_{М2} = \sum Ц_д\,, \tag{7.17}$$

де: $Ц_д$ – вартість дисків CD/DVD.

Вартість розхідних матеріалів на заправку друкуючих пристроїв не враховуємо:

$$З_{М3} = \sum Ц_з.\,, \tag{7.18}$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу.

$$З_М = 840 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \tag{7.19}$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 37380 \cdot 15 \cdot 0{,}01 = 5607 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 1 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \tag{7.20}$$

де: $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 35143 \cdot 4/(1 \cdot 12) = 11714 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м\,. \tag{7.21}$$

$$С_п = 37380+3738+9046+5607+840+5607+11714 = 73932 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($Р_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 30%.

$$П_р = 0{,}01 \cdot P_п \cdot C_п, \quad (7.22)$$

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 30 \cdot 73932 = 22180 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	37380
2. Додаткова зарплата виконавців	$З_д$	3738
3. Відрахування на соціальні потреби	$С_{оц}$	9046
4. Загальногосподарські витрати	$Г_{осп}$	5607
5. Витрати на матеріали	$З_М$	840
6. Освоєння нових операційних систем, мов програмування	$О_п$	5607
7. Амортизація основних фондів	$А_м$	11714
8. Повна собівартість програмного забезпечення	$С_п$	73932
9. Плановий прибуток	$П_р$	22180
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	96112
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	19222,4
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	115334,4

Собівартість та ціна програмної продукції розрахована згідно методичних рекомендацій до виконання розділу "Економічна ефективність розробленої

програми" випускної кваліфікаційної роботи.

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах.

Враховуючи, що програмний продукт розроблений на волонтерській основі по замовленню міської ради міста Кропивницький, вартість капіталовкладень у споживача приймаємо рівним нулю.

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року.

Програмний продукт представляє собою мобільний додаток, який вільно встановлюється з “App Store”, тому експлуатаційні витрати не обраховуються.

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \qquad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий

коефіцієнт капіталовкладень.

Період окупності додаткових капітальних вкладень у виробника програмної продукції визначається:

$$T_в = \frac{K_р}{(Ц_п - C_п) \cdot N_е} \cdot \frac{n_{міс}}{12}, \quad (7.26)$$

де: $K_р$ – балансова вартість основних фондів розробника.

Величину економічного ефекту у користувача програмної продукції за формулою визначаємо:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно; $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

Період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.28)$$

Враховуючи те, що програмний продукт розроблений як волонтерська діяльність показники економічної ефективності програмної продукції не обраховуються.

Висновки до розділу 7

Розроблена програма економічно не вигідна, але має значний соціальний ефект. Програмний продукт є важливою частиною удосконаленої цифрової муніципальної інформаційної системи медичних послуг закладів охорони здоров'я міста Кропивницький За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1. Вступ

Впровадження комп'ютерних технологій принципово змінило характер праці різних категорій фахівців. Працівники, використовують комп'ютерну техніку, на своєму досвіді оцінили її величезні можливості. Одночасно виникла певна безтурботність при її експлуатації.

Характерною ознакою сучасного науково-технічного прогресу практично у всіх сферах діяльності людини є широке застосування комп'ютерних технологій, заснованих на використанні електронно-обчислювальних машин (ЕОМ). Сьогодення, а тим більше, майбутнє, вже важко уявити без комп'ютерів та іншої електронної техніки. Адже саме завдяки їм стала можливою швидка переробка величезних обсягів інформації, проведення необхідних розрахунків, виконання різних видів робіт, пов'язаних обробкою текстових та ілюстраційних зображень, організація оперативного отримання та передачі інформації, збереження її значних обсягів електронним способом.

Недотримання вимог безпеки призводить до того, що й через кілька днів роботи за комп'ютером співробітник починає відчувати певний дискомфорт: в нього виникає головний біль і різь у власних очах, з'являються почуття виснаження й дратівливість. В окремих людей порушується сон, погіршується зір, занедужують руки, шия, поперек тощо.

До недоліків умов праці користувачів комп'ютерної техніки можна віднести:

– недостатню площу і обсяг виробничого приміщення;

– недотримання вимог, мікроклімату на робочих місцях;

– низький рівень освітленості у приміщеннях і на робочих поверхнях апаратури;

– підвищений рівень низькочастотних магнітних полів від моніторів;

– порушення вимог організації робочих місць;

– недотримання вимог до режимам праці та відпочинку;

– надмірне виробничу навантаження працівників;

– відсутність навичок зниження впливу психоемоційного напруги.

Відповідно до ст.14 Закону «Про охорони праці» [22] на роботодавця покладено обов'язок забезпечити: безпеку працівників при експлуатації устаткування; застосування коштів індивідуальної захисту працівників; відповідні вимоги охорони праці, умови праці в кожному робоче місце; дотримання режиму праці та відпочинку працівників; навчання безпечним методам і прийомам виконання; інструктаж з охорони праці; організацію контролю над станом умов праці в робочих місць; проведення атестації робочих місць в умовах праці.

Максимально зменшити кількість шкідливих впливів на людину при високій продуктивності праці, створити комфортні умови для роботи людей – ось одна з головних задач охорони праці.

8.2. Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальни машини (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98 [19].

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

– ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.

– ризик виникнення пожежі;

– негативний вплив на органи зору людини;

– ризики ураження електричним струмом;

– недостатня, або надмірна освітленість робочого місця;

– електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);

– несприятливі мікрокліматичні умови;

– нервово-емоційна напруженість праці;

– інтелектуальні навантаження;

– монотонність праці;

– невідповідність ергономічних показників робочого місця діючим вимогам;

– шум;

– статичні навантаження на кістково-м'язовий апарат.

8.3. Аналіз умов праці на робочому місці програміста

Робота програміста пов'язана з постійною роботою на ЕОМ, яка відбувається у кімнаті розмірами 4,8 м×7,2 м×2,8 м. Одна з її більших стін має шість двостулкових вікон, розмірами 2,1 м×1,9 м, які виходять на північний схід. Вікна розташовані рівномірно по всій довжині стіни. Підлога в кімнаті покрита лінолеумом, всі стіни пофарбовані світло оранжевого кольору до висоти 2,8м, а далі підвісна стеля. Уздовж стін розташовані комп'ютерні столи. На них розташовуються 2 персональні комп'ютери й інша оргтехніка (сканер принтери, телефони й ксерокс). Столи мають пластикове покриття. Габарити їхньої робочої поверхні 1245 мм×840 мм. Висота столів 750 мм. Висота стільців від рівня підлоги становить 425 мм.

Згідно НПАОП 0.00 – 1.28 – 10 «Правила охорони праці під час електронно-обчислювальних машин» площа повинна задовольняти умові – не менш 6 $м^2$ на одне робоче місце. Кратність повітрообміну в приміщенні вузла

також регламентується ДСанПіН 3.3.2.007-98 [19], вона повинна становити 20 м^3/годину на одне місце. Виконання даних вимог забезпечить підтримку в приміщенні вузла оптимального значення вологості й складу повітря.

Відповідно ДБН В.2.5 – 28 – 2006 [18] роботу програміста можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення вузла можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при сполученому висвітленні), повинен становити 0,5%, освітленість при штучному висвітленні повинна становити 300 лк.

Таблиця 8.1 – Допустимі спектри рівнів звукового тиску

Робоче місце	Рівень звукового тиску, дБ, в октавних смугах із середньогеометричними частотами, Гц								Рівень звуку і еквівалентний рівень звуку, дБА
	63	125	250	500	1000	2000	4000	8000	
Приміщення конструкторських бюро, програмістів обчислювальних машин, лабораторій для теоретичних робіт і опрацювання експериментальних даних, прийому хворих в медпунктах	71	61	54	49	45	42	40	38	50

За результатами виміру освітленості відділом охорони праці величина освітленості від системи загального штучного висвітлення лежить у межах 200-250 лк, що не відповідає вимогам, які пред'являються до приміщення.

Відповідно ДСанПіН 3.3.2.007-98 [19] рівні звукового тиску в робочому приміщенні не повинні перевищувати в октавних смугах із середньо геометричними частотами наступних значень, наведених у таблиці 8.1.

У приміщенні перебувають наступні джерела шуму: електродвигуни внутрішнього вентилятора ЕОМ; працюючі принтери; працюючі дисководи. Шум, вироблений вентилятором можна класифікувати як постійний, всі інші джерела шуму, як імпульсні. Відповідно паспорта на приміщення рівень звуку, Дб(А), обмірюваний за шкалою (А) шумоміра досяг величини 28,3 Дб(А) при роботі всього встаткування вузла, включаючи й ксерокс. Це дозволяє зробити висновок про відповідність рівня звуку в приміщенні вимогам нормативних актів.

Ергономічні вимоги до робочого місця працюючого з ВДТ ЕОМ і ПЕОМ нормуються НПАОП 0.00 – 1.28 – 10**.** Оптимальне положення тіла того, що працює забезпечується відповідною конструкцією робочого місця, а також регуляцією висоти робочої поверхні, сидіння, простори й підставки для ніг. Даного місця програміста не мають регульованих параметрів. Відмінності реальних параметрів робочого місця від параметрів відповідні вимоги нормативного акту дані в таблиці 8.2.

Таблиця 8.2 – Відмінності реальних параметрів робочого місця від параметрів відповідні вимоги нормативного акту

Ріст людини, см	Висота робочої поверхні мм,	Висота простору для ніг, мм	Висота робочого сидіння, мм
175	765(740)	655(600)	450(440)

У дужках зазначені реальні значення параметрів робочого місця; всі вони не відповідають параметрам, зазначеним у стандарті.

Параметри мікроклімату можуть мінятися в широких межах, тоді як

необхідною умовою життєдіяльності людини є підтримка сталості температури тіла завдяки властивості терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище.

У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У санітарних нормах ДСН 3.3.6.042 – 99 встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються в залежності від пори року, характеру трудового процесу і характеру виробничого приміщення (див. табл. 8.3).

Таблиця 8.3 – Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	22 – 24°С 40 – 60% до 0,1 м/с
Теплий	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	23 – 25°С 40 ... 60% 0,1 ... 0,2 м / с

8.4. Розрахункова частина

Для захисного штучного заземлення застосовуються ветикальні електроди: метелевий куток *63·63·6 мм*., (згідно з ДСТУ 2251-93 «Кутики сталеві гарячекатані рівнополичні. Сортамент») довжиною *L=1,7 м.,* та горизонтальний електрод — метелева полоса з перетином *60·5 мм*. Напруга — *220/380 В*. Розрахункова схема розташування заземлюючих електродів — по контуру (прямокутником).

Розрахунок проведем за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта – чорнозем, нижнього шару грунта — глина (питомий опір $\rho_2 = 40$ *Ом·м)*. Умовна товщина верхнього шару грунта: $H=0,5$ *м*. Відстань між вертикальними заземлювачами (електродами) $A=3$ *м*. Глибина закладення горизонтального контура заземлення $t=0,6$ *м*. Опір заземлювача, який нормується: $R_{ЗН} = 4$ *Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та довжину полоси (горизонтального заземлювача).

Розрахунок

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,6+1,7/2=1,45 \text{ м.}$$

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$$\rho = \psi\rho = 1,36 \cdot 40 = 54,5 \text{ Ом·м.}$$

де $\psi = 1,36$ – табличне значення коефіцієнта сезонності для відповідної климатичной зони у багатошаровому грунті [58];

$\rho_2 = 40$ *Ом·м*. – табличне значення питомого опору нижнього шару грунта (глина) [12].

Еквівалентний діаметр ветикальньного електрода (кутка) [58]:

$$Dв=0,95 \cdot K = 0,95 \cdot 63 = 59,85 \text{ мм.} = 0,0598 \text{ м.}$$

де $K = 63$ *мм.* – розмір метелевого кутка (задан).

Відношення $A/L=3/1,7=1,76$

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [58]:

$$R_O=0,366(\rho/L)[\lg(2L/Dв)+(1/2)\lg((4T+L)/(4T-L))] =$$

$$=0,366(54,5/1,7)[(\lg(2 \cdot 1,7/0,0598)+(1/2)\lg((4 \cdot 1,45+1,7)/(4 \cdot 1,45-1,7))] =$$

$$=22 \text{ Ом.}$$

Визначаємо коефіцієнт екранування ветикальньних електродів $Кев=0,62$ при орієтовній кількості ветикальньних електродів, яке дорівнює 5 [58].

Визначаємо необхідну кількість вертикальних електродів заземлювача

(без вхарування горизонтального заземлювача), при $R_{ЗН} = 4$ *Ом*:

$$N=R_О /(Кев\ R_{ЗН})= 22 / (0,62\cdot 4)= 8,9 \approx 9\ шт.$$

Визначаємо довжину з'єднуючої полоси:

$$L_П =1,05\cdot A\cdot N= 1,05\cdot 3\cdot 9=28,35\approx 28\ м.$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору грунта $K_П$ [58]:

$$R_П =0,366(\rho\cdot K_П/L_П)lg(2\cdot L_П{}^2/(B\cdot t))=$$

$$=0,366(40\cdot 5/28)\cdot lg((2\cdot 28^2)/(0,06\cdot 0,6))=12,1\ Ом.$$

де $K_П=5$ – табличне значення кліматичного коефіцієнта питомого опору грунта для відповідної кліматичної зони для з'єднуючої полоси [58]:

$B = 60\ мм. = 0,06\ м.$- ширина з'єднуючої полоси (задана).

Загальньний опір розтіканню електричного струму заземлювача [58]:

$$R =(R_О\cdot R_П)/(R_О\cdot\eta_П+ N\cdot R_П\cdot Кев)=$$

$$=(22\cdot 12,1)/(22\cdot 0,6+ 9\cdot 12,1\cdot 0,62)=3,35\ Ом.$$

де $\eta_П = 0,6$ – табличне значення коефіцієнта екранування з'єднуючої полоси [58].

Умова $R \le R_{ЗН}$ виконується ($3,35 \le 4$).

Так як при *9* вертикальних електродах *R* суттево менше $R_{ЗН}$, зменшимо кількість вертикальних електродів *N* до *8* і виконаємо перерахунок. У результаті остаточно отримали: $R =3,7$ *Ом. при* кількості вертикальних електродів $N=8$.

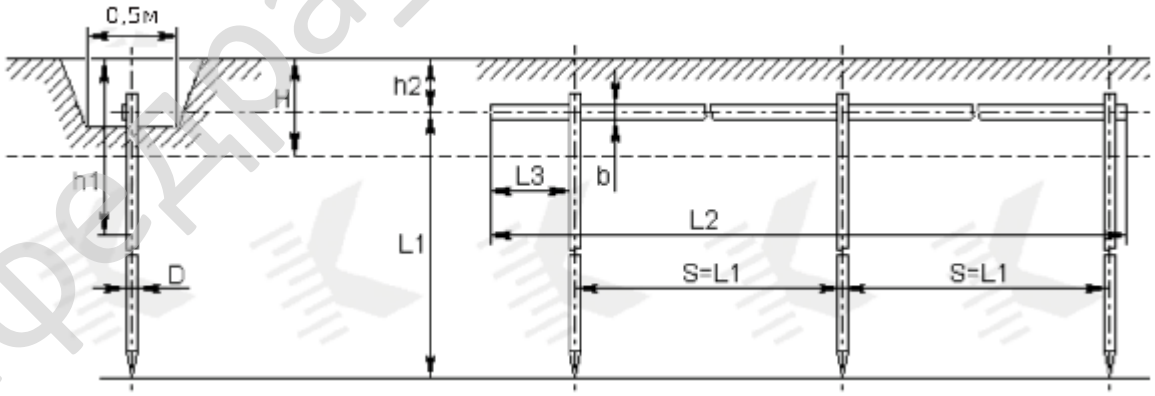

Рисунок 8.1 – Схема штучного заземлення

Висновки до розділу 8

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз умов праці на робочому місці програміста, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи.

Тільки повна усвідомленість працівника про можливу шкоду та небезпеку, що можуть підстерігати його на робочому місці, та дотримання вимог нормативних актів з питань охорони праці та відповідних рекомендацій фахівців, дозволять значною мірою знизити негативний вплив шкідливих та небезпечних факторів при роботі з комп'ютером на організм людини.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

9 ОСНОВНІ ВИСНОВКИ

Ініціатива органів самоврядування міста Кропивницького та потреба в діджиталізації сфери медичних послуг зробили муніципальну інформаційну систему медичних послуг актуальною темою для дипломної роботи. Робота над дослідженням та удосконаленням є частиною програми по створенню інформаційних ресурсів для громади міста. Загалом це унікальна інформаційна система, її призначенням є інформування громади про медичні послуги та роботу медичних закладів міста. Виходячи з призначення областю застосування є сфера медичних послуг міста Кропивницький.

Програма реалізована використовуючи мову високого рівня Swift та програмне середовище Xcode, використання саме цих інструментів при створенні додатку дозволяє мінімізувати строк розробки програмного забезпечення, і, як наслідок, зменшити витрати на його розробку.

В процесі планування та опису проектних рішень було визначено функціонал муніципальної інформаційної системи медичних послуг, на основі якого була підібрана клієнт-серверна архітектура для реалізації проекту та визначені головні компоненти – клієнт та група серверних елементів. В ході розробки структурної схеми були визначені всі структурні елементи системи та зв’язки між ними, після чого було розроблено функціональну схему на основі структурних елементів клієнта були розписані їх основні функції та була проведена деталізація їх взаємозв’язків. Наступним кроком було побудовано узагальнену діаграму процесів, що надає представлення про весь робочий цикл ПЗ клієнта.

Перед початком робіт по реалізації проекту було проведено роботу над усіма необхідними розрахунками (Google Maps) та експериментальну перевірку роботи клієнт-серверної архітектури. Після підтвердження правильності отриманих результатів були перевірені, розроблені та описані усі необхідні

алгоритми, а саме – узагальнений алгоритм всього додатку та алгоритми для роботи сервісів. Окрім, алгоритмів також були зплановані механізми захисту ПЗ, що були використані при роботі з серверною чатиною, а саме – захист зв'язку через мережу Інтернет.

Після підготовчого етапу було виконано роботу по реалізації проекту. Відповідно до визначеної архітектури додатку розроблено класи View Controller, Coordinator та ViewModel, та налаштовано взаємодію компонентів системи. Згідно з усіма описаними алгоритмами були створені класи сервісів. А також, спираючись на структурну та функціональні схеми реалізовано зв'язні, системні та програмні інтерфейси, в результаті яких вдалося зкомпонувати всі частин ІС.

Були проаналізовані доступні для впровадження в експлуатацію інструменти і виявлено, що для публікації додатків під iOS пристроїв існує лише одна платформа – це App Store. Також, було розроблено інструкцію для користувача, що описує доступний функціонал, механіку використання додатку та містить всі необхідні ілюстрації та пояснення.

Завершальним етапом в роботі над дипломним проектом було викладення результатів досліджень у вигляді пояснення наукової новизни, що полягає у вдосконаленні моделі муніципальної інформаційної систем медичних послуг за рахунок реалізації процесу офлайнового режиму функціонування системи, що на відміну від існуючих моделей муніципальних систем забезпечує доступ до даних ІС у період відсутності зв'язку з Інтернетом.

Економічно не вигідна розроблена програма, але має значний соціальний ефект. Програмний продукт є важливою частиною удосконаленої цифрової муніципальної інформаційної системи медичних послуг закладів охорони здоров'я міста Кропивницький

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. ПЗ має простий, дружній та зручний інтерфейс користувача та забезпечує легкий доступ до інформації, що є його основним призначенням.

Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення та використанні даного програмного коду при створенні аналогів.

Частина результатів цієї роботи апробовані на IV міжнародній науково-практичній конференції «Інформаційна безпека та комп'ютерні технології» (Кропивницький, 2021) та на II всеукраїнській науково-практичні конференції здобувачів вищої освіти й молодих учених «Комп'ютерна інженерія і кібербезпека: досягнення та інновації» (Кропивницький, 2020), на LV науково-технічній конференції здобувачів вищої освіти «Наука в ЦНТУ: основні досягнення та перспективи розвитку» за підсумками проведення «Дня науки – 2021» (Кропивницький, 2021), а також опубліковані у збірниках матеріалів означених науково-практичних конференцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Apple iOS [Електронний ресурс]. – Режим доступу: https://ru.bmstu.wiki/ Apple_iOS

2. CommonCrypto in Swift [Електронний ресурс]. – Режим доступу: https://academy.realm.io/posts/danny-keogan-swift-cryptography/

3. Getting Started With RxSwift and RxCocoa [Електронний ресурс]. – Режим доступу: https://www.raywenderlich.com/1228891-getting-started-with-rxswift-and-rxco coa

4. Google Maps Platform. Google Maps URL Scheme for iOS [Електронний ресурс]. – Режим доступу: https://developers.google.com/maps/documentation/urls/ios-urlscheme

5. How Collaboration Works in Postman [Електронний ресурс]. – Режим доступу: https://www.postman.com/how-api-collaboration-works/

6. How to use SwiftLint with Xcode to enforce Swift style and conventions? [Електронний ресурс]. – Режим доступу: https://medium.com/developerinsider/how-to-use-swiftlint-with-xcode-to-enforce-swift-style-and-conventions-368e49e910

7. Jonathon Manning, Paris Buttfield-Addison iOS Swift Game Development Cookbook. – Boston 2018. — 330 p.

8. MBProgressHUD [Електронний ресурс]. – Режим доступу: https://github.com/jdg/MBProgressHUD

9. MVVM with Coordinators and RxSwift [Електронний ресурс]. – Режим доступу: https://academy.realm.io/posts/mobilization-lukasz-mroz-mvvm-coordinato-rsr xswift/

10. TagListView [Електронний ресурс]. – Режим доступу: https://github.com/ElaWorkshop/TagListView

11. The Swift Programming Language Documentation [Електронний ресурс]. – Режим доступу: https://swift.org/documentation/

12. React Native [Електронний ресурс]. – Режим доступу: https://reactnative.dev

13. What is COCOAPODS [Електронний ресурс]. – Режим доступу: https://cocoapods.org

14. What is a Thin Client [Електронний ресурс]. – Режим доступу: https://www.clearcube.com/posts/what-is-a-thin

15. Грамотна клієнт-серверна архітектура: як правильно проектувати і розробляти web API [Електронний ресурс]. – Режим доступу: https://echo.lviv.ua/dev/6455

16. Громадський бюджет [Електронний ресурс]. – Режим доступу: https:// www.kr-rada.gov.ua/gromadskiy-byudzhet/

17. Держава у смартфоні [Електронний ресурс]. – Режим доступу: https://www.ukrinform.ua/tag-derzava-u-smartfoni

18. Державні будівельні норми України: ДБН В.2.5-28:2018 [Електронний ресурс]. – Режим доступу до ресурсу: https://goo.su/9AkQ

19. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98 [Електронний ресурс]. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98 (дата звернення 19.10.22).

20. Бази даних та інформаційні системи [Електронний ресурс]. – Режим доступу: https://wiki.cuspu.edu.ua/index.php/Бази_даних_та_інформаційні_системи

21. Електроні сервіси [Електронний ресурс]. – Режим доступу: https://kr-rada. gov.ua/elektronni-servisi

22. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII [Електронний ресурс]. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення 19.10.22).

23. Зелений Кропивницький [Електронний ресурс]. – Режим доступу: https:// texty.org.ua/d/2018/trees/

24. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

25. Інформаційна відкритість органів місцевого самоврядування [Посібник для муніципальних посадовців]. – Івано-Франківськ: ГО «Агенство з розвитку приватної ініціативи» [Електронний ресурс]. – Режим доступу: https://arpi.org.ua/doc2/posibnyk.pdf

26. Інформаційна система [Електронний ресурс]. – Режим доступу: https://stud.com.ua/34527/informatika/informatsiyna_sistema

27. Інформаційні системи та їх роль в управлінні економікою. [Електронний ресурс]. – Режим доступу: https://www.uzhnu.edu.ua/uk/infocentre/get/6742

28. Клієнт-серверна архітектура та ролі серверів. [Електронний ресурс]. – Режим доступу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229

29. Медичні інформаційні системи: огляд можливостей і приклади використання [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/articles/medical-information-systems.html

30. Методичні рекомендації до виконання розділу "Заходи з охорони праці та техніки безпеки" випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти для здобувачів вищої освіти спеціальностей 123 "Комп'ютерна інженерія" та 122 "Комп'ютерні науки" / М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програм. забезпечення; [укл. О.В. Оришака, К.М. Марченко]. – Кропивницький: ЦНТУ, 2022. — 19 с.

31. Мови програмування для мобільної розробки [Електронний ресурс]. – Режим доступу: https://code.tutsplus.com/uk/articles/mobile-development-languages-cms-29138

32. Моделі життєвого циклу, принципи і методології розробки програмного забезпечення (ПЗ) [Електронний ресурс]. – https://evergreens.com.ua/articles/ software-development-metodologies.html

33. Моделі і методи проектування інформаційних систем [Електронний ресурс]. – https://elearning.sumdu.edu.ua/free_content/lectured:de1c9452f2a161439391120eef364dd8ce4d8e5e/20151203140326/165292/index.html

34. Наказ Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 31.10.2016 «Про затвердження ДБН В.1.1-702016» [Електронний ресурс]. – Режим доступу до ресурсу: ttps://ips.ligazakon.net/document/fn025551 (дата звернення 19.09.22).

35. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [Електронний ресурс]. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508 (дата звернення 19.10.22).

36. О.П. Доренский та О.С. Дробко Модель муніципальної інформаційної системи медичних послуг: матеріали IV Міжнародної наук.-практ. конф. «Інформаційна безпека та комп'ютерні технології», Кропивницький, 2021 р., 39 с.

37. О.П. Доренский та О.С. Дробко Структура і методологічні засади реалізації мобільного застосунку муніципальної інформаційної системи медичних послуг: матеріали II Всеукраїнської наук.-практ. конф. здобувачів вищої освіти й молодих учених «Комп'ютерна інженерія і кібербезпека: досягнення та інновації», Кропивницький, 2020 р., 23 с.

38. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький : ЦНТУ, 2022. – 175 с.

39. Оришака О.В. Охорона праці в галузі та цивільний захист / О.В Оришака, Г.П. Горбачова, О.М. Мезенцева, К.М. Марченко, К.О. Буравченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький: Видавець Лисенко В.Ф., 2019. − 226 с.

40. Освітня карта Кропивницького [Електронний ресурс]. – Режим доступу: https://rk.kr.ua/osvitnja-karta-kropivnitskogo

41. Офіційний сайт Apple About Objective-C [Електронний ресурс]. – Режим доступу: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

42. Офіційний сайт Apple Developer Documentation [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation

43. Офіційний сайт Apple Framework Core Location [Електронний ресурс]. –Режим доступу: https://developer.apple.com/documentation/corelocation

44. Офіційний сайт Apple Framework Foundation [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/foundation

45. Офіційний сайт Apple Framework MapKit [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/mapkit/

46. Офіційний сайт Apple iOS 14 [Електронний ресурс]. – Режим доступу: https://developer.apple.com/ios/

47. Офіційний сайт Apple iOS Design Themes [Електронний ресурс]. – Режим доступу: https://developer.apple.com/design/human-interface-guidelines/

48. Офіційний сайт Apple Human Interface Guidelines [Електронний ресурс]. – Режим доступу: https://developer.apple.com/design/human-interface-guidelines/

49. Офіційний сайт Apple Swift. [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/swift

50. Офіційний сайт Apple Framework UIKit [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/uikit

51. Офіційний сайт Apple Xcode. [Електронний ресурс]. – Режим доступу: https://developer.apple.com/xcode

52. Офіційний сайт App Store [Електронний ресурс]. – Режим доступу: https://www.apple.com/ua/app-store/

53. Офіційний сайт Firebase [Електронний ресурс]. – Режим доступу: https://firebase.google.com/

54. Охорона праці. Ч. 1. Захисне заземлення : метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд. ; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. – Кіровоград : КІСМ, 1997. – 20 с.

55. Охорона праці. Ч. 2. Занулення : метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ−сумісного типу / [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака, А. Е. Солових, С. Е. Катеринич] ; Мін-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – 2−ге вид., перероб. та доп. – Кропивницький : ЦНТУ, 2019. – 27 с.

56. Петрішенко С. Розробка мобільних застосувань для операційної системи iOS за допомогою мови програмування Swift, Київ, 2016, 17с.

57. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [Електронний ресурс]. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення 19.09.22).

58. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : “Колос”, 1973. – 238 с.

59. У Кропивницькому конкурс «Громадський бюджет» отримав новий е-сервіс [Електронний ресурс]. – Режим доступу: https://cbn.com.ua/2020/03/13/u-kropyvnytskomu-konkurs-gromadskyj-byudzhet-otrymav-novyj-e-servis/

60. Що таке Флаттер і чому ви повинні навчитися цьому в 2020 році [Електронний ресурс]. – Режим доступу: https://hackit-ukraine.com/445-what-is-flutter-and-why-you-should-learn-it-in-2020

61. Що це таке тестування програмного забезпечення та яке його призначення? [Електронний ресурс]. – Режим доступу: https://www.quality-

assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/

62. Усов В. Swift. Основы разработки приложений под iOS и macOS 3-е издание, дополненное и переработанное. – Питер, 2017. — 368 с.

63. Ханг В. Оптимизация производительности приложений для iOS. Для профессионалов. – Питер, 2013. — 320 с.

64. Медичні інформаційні системи: переваги та покрокове впровадження [Електронний ресурс]. – Режим доступу: https://medplatforma.com.ua/article/1084-meditsinskie-informatsionnye-sistemy-rus

65. Медичні інформаційні системи: огляд можливостей та приклади використання [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/ru/articles/medical-information-systems.html

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-123.22.0009.00.00.ТЗ		
Вим	*Арк.*	*№ докум.*	*Підп.*	*Дата*			
Розроб.		*Дробко О.С.*			*Дослідження та програмне забезпечення мобільного iOS-клієнта удосконаленої муніципальної інформаційної системи медичних послуг*	Літера	Аркуш	Аркушів
Перев.		*Доренський О.П*				М	1	7
Н.контр.		*Гермак В.С.*			*ЦНТУ КІ-21М1,4*			
Затв.		*Смірнов О.А.*						

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення удосконаленої муніципальної інформаційної системи медичних послуг міста Кропивницького.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (наказ №19-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є досліждення та розробка програмного забезпечення мобільного клієнта удосконаленої муніципальної інформаційної системи медичних послу на основі впровадження нових інформаційних технологій і застосування сучасних засобів програмування.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є ініціатива органів місцевого самоврядування та дані про медичні послуги та заклади міста Кропивницького.

5 Технічні вимоги

5.1 Вміст проекту

Складовими розробки є:

– аналіз існуючих електроних сервісів міста Кропивницький;

– вибір і обґрунтування методики побудови додатку і засобів його реалізації;

– розробка структур даних і механізму їхньої взаємодії, робочих форм і засобів;

– техніко-економічне обґрунтування доцільності прийнятого до розробки компоненту автоматизованої системи бухгалтерського обліку;

– аналіз умов праці програміста в лабораторії К505;

– розробка програми, яка реалізує алгоритми роботи клієнту удосконаленої муніципальної інформаційної системи.

5.2 Показники призначення

Система повинна забезпечувати:

– доступ до інформаціїї про медичні послуги;

– простий, інтуїтивно зрозумілий інтерфейс для користувача;

– збір аналітичних даних.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення повинно забезпечувати повний доступ до інформаціїї про медичні послуги міста Кропивницький.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати клієнт-серверну архітертуру, Google Maps, Firebase Analitics та файлову систему мобільного девайсу.

5.5 Вимоги до надійності

Компонент повинен використати існуючі угоди по стандартним викликам процедур, функцій, засобів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Розроблене програмне забезпечення може використовуватися лише на мобільних пристроях iPhone.

5.7 Вимоги до складу і параметрів технічних засобів

Компонент повинен бути реалізований на ПК або ноутбуці фірми Apple в операційному середовищі MacOS і орієнтований на сумісні з платформою iOS зовнішні пристрої, мережне обладнання і прикладне програмне забезпечення.

5.8 Вимоги до інформаційної та програмної сумісності

Сумісність програмного забезпечення повинна бути забезпечена за рахунок використання бібліотеки UnityEnigen, System.Collections, Generic, яка сумісна з усіма останніми версіями операційної системи Microsoft Windows.

5.8.1 Обладнання

MacBook Pro (15-inch, 2019) / 2,3 GHz 8-Core Intel Core i9 / 16 GB 2400 MHz DDR4 / 516 Gb / Radeon Pro 560X 4 GB та будь-який iPhone.

5.8.2 Мова програмування

Swift з використанням IDE Xcode.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена в вигляді опису схем і алгоритмів, інструкції користувача, а також текстів вхідних модулів програмного забезпечення в відповідності з ЄСПД.

7 Економічні вимоги

7.1 Для програми ядра виконавчої системи моделі необхідно виробити функціонально-вартісний аналіз варіантів розробки

7.2 Виконати розрахунок кошторису витрат на реалізацію прийнятого варіанту з урахуванням розцінок на 1 жовтня 2022 р.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути наведений аналіз умов праці програміста в лабораторії К505 ЦНТУ і вироблений розрахунок категорії

важкості праці програміста.

9 Перелік документів, які необхідно розробити

– наукова новизна;
– структурна схема системи;
– функціональна схема системи;
– блок-схеми алгоритму роботи системи;
– діаграма процесів;
– пояснювальна записка.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок-схем алгоритмів роботи програмного забезпечення компоненту.

10.4 Побудова схем взаємодії структур даних.

10.5 Створення прототипу компоненту. Створення програмного продукту.

10.6 Відлагодження компоненту, аналіз отриманих результатів.

10.7 Робота над питаннями охорони праці і техніки безпеки.

10.8 Розрахунки по техніко-економічному обґрунтуванню (остаточні).

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю і приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2022 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 19.12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за другим (магістерським) рівнем вищої освіти

_____ О.П. Доренський

Дослідження та програмне забезпечення мобільного iOS-клієнта удосконаленої муніципальної інформаційної системи медичних послуг

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 32

Літера: РП

Кропивницький – 2022 року

```
//Copyright (c) 2021 Onix-Systems
//
// Permission is hereby granted, free of charge, to any person obtaining
// a copy of this software and associated documentation files (the
// "Software"), to deal in the Software without restriction, including
// without limitation the rights to use, copy, modify, merge, publish,
// distribute, sublicense, and/or sell copies of the Software, and to
// permit persons to whom the Software is furnished to do so, subject to
// the following conditions:
//
// The above copyright notice and this permission notice shall be included
// in all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
// EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
// MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
// IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
// CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
// TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
// SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

AppDelegate.swift

```
import UIKit
import Firebase

@UIApplicationMain
 class AppDelegate: UIResponder, UIApplicationDelegate {

    private var appCoordinator: AppCoordinator?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        let window = UIWindow(frame: UIScreen.main.bounds)

        if #available(iOS 13.0, *) {
            window.overrideUserInterfaceStyle = .light
        }

        appCoordinator = AppCoordinator(window: window)
        appCoordinator?.start()

        registerServices()

        return true
    }

    private func registerServices() {

        FirebaseApp.configure()
        Analytics.setAnalyticsCollectionEnabled(true)
    }

}
```

AppCoordinator.swift

```
import UIKit

final class AppCoordinator {
    private var navigationController: UINavigationController?
    private let window: UIWindow
```

```
    private var networkService: NetworkServiceType!
    private var analyticsService: AnalyticsServiceType!
    private var serviceHolder: ServiceHolder!
    private var mainCoordinator: MainCoordinatorType?

    init(window: UIWindow) {
        self.window = window

        startServices()
    }

    private func startServices() {
        serviceHolder = ServiceHolder()

        networkService = NetworkService()
        serviceHolder.add(NetworkServiceType.self, for: networkService)

        analyticsService = AnalyticsService()
        serviceHolder.add(AnalyticsServiceType.self, for: analyticsService)
    }

    func start() {
        navigationController = UINavigationController()

        window.rootViewController = navigationController
        window.makeKeyAndVisible()

        mainCoordinator = MainCoordinator(navigationController:
navigationController, serviceHolder: serviceHolder)
        mainCoordinator?.start()
    }
}
```

NetworkManager.swift

```
import Foundation

///Variable Result(generic): nessary type or String with error-message
enum Result<T> {
    case success(T)
    case failure(String)
}

///Singlton /(shared)
final class NetworkManager {

    static let shared = NetworkManager()
    fileprivate let path = "https://medical.kr-rada.gov.ua/api/"

    func getHospital(id: String, complition: @escaping (Result<Data>) -> Void) {
        let endpoint = (path + "hospital/\(id)")
        loadByEndpoint(by: endpoint, completion: complition)
    }

    func getHospitals(type: String, completion: @escaping (Result<Data>) ->
Void) {
        let endpoint = (path + "get-equipment/category-by-type?type=" + type)
        loadByEndpoint(by: endpoint, completion: completion)
    }

    func getAllCategories(completion: @escaping (Result<Data>) -> Void) {
        let endpoint = (path + "get-equipment/category")
        loadByEndpoint(by: endpoint, completion: completion)
    }
```

```
    func getSearch(text: String, complition: @escaping (Result<Data>) -> Void) {
        if text.count >= 3 {
        let endpoint = (path + "search?q=" + text)
            loadByEndpoint(by: endpoint, completion: complition)
        }
    }

    fileprivate func loadByEndpoint(by endpoint: String, completion: @escaping
(Result<Data>) -> Void) {
        guard let point = endpoint.addingPercentEncoding(withAllowedCharacters:
.urlQueryAllowed),
            let url = URL(string: point) else { assertionFailure("URL is nil");
return }

        loadData(by: url) { data, _, error in
            if let error = error {
completion(.failure(error.localizedDescription)) }
            if let data = data { completion(.success(data)) }
        }
    }

    fileprivate func loadData(by url: URL, completion: @escaping (Data?,
URLResponse?, Error?) -> Void) {
        URLSession.shared.dataTask(with: url, completionHandler:
completion).resume()
    }
}
```

ParsingHelper.swift

```
import Foundation

///Singlton /(parsingByType)
final class ParsingHelper {

    static func parsingByType<T: Codable>(_ result: Result<Data>, _ complition:
(Result<T>) -> Void) {

        switch result {
        case .success(let data):
            do {
                let object: T = try
                    JSONDecoder().decode(T.self, from: data)
                complition(.success(object))
            } catch {
//              assertionFailure("parse error")
                complition(.failure(error.localizedDescription))
            }

        case .failure(let error):
//            assertionFailure("data error")
            complition(.failure(error))
        }
    }
}
```

Hospital.swift

```
import Foundation

final class Hospital: Codable {

    var id: String
```

```
    var name: String?
    var adress: String?
    var lan: String?
    var lat: String?
    var workTime: String?
    var unworkTime: String?
    var roomNumber: Int?
    var floorNumber: Int?
    var equipName: String?
    var structureName: String?
    var equipCountry: String?
    var eqipYear: String?
    var equipCondition: String?


    enum CodingKeys: String, CodingKey {
        case id = "id_u"
        case name = "custodian_name"
        case adress = "address_street_address"
        case lan = "longitude"
        case lat = "latitude"
        case workTime = "work_shedule"
        case unworkTime = "availability_restriction"
        case roomNumber = "room_number"
        case floorNumber = "floor_number"
        case equipName = "equipment_title"
        case structureName = "structure_name"
        case equipCountry = "producer_country"
        case eqipYear = "equipment_year"
        case equipCondition = "equip_condition"
    }
}

final class HospitalRequest: Codable {
    var status: Int
    var message: String
    var data: [String: [Hospital]]?

    enum CodingKeys: String, CodingKey {
        case status
        case message
        case data
    }
}

final class HospitalsRequest: Codable {
    var status: Int
    var message: String
    var data: HospitalsData?

    enum CodingKeys: String, CodingKey {
        case status
        case message
        case data
    }
}

final class HospitalsData: Codable {
    var qty: Int
    var hospitals: [Hospital]

    enum CodingKeys: String, CodingKey {
        case qty
        case hospitals
```

```
    }
}

final class SearchRequest: Codable {
    var status: Int
    var message: String
    var data: SearchData?

    enum CodingKeys: String, CodingKey {
        case status
        case message
        case data
    }
}

final class SearchData: Codable {
    var qty: Int
    var hospitals: [Hospital]

    enum CodingKeys: String, CodingKey {
        case qty
        case hospitals = "search_result"
    }
}
```

Categories.swift

```
import Foundation

protocol Response {
    var status: Int { get set }
    var code: String { get set }
    var message: String { get set }
}

struct CategoriesResponse: Codable, Equatable, Response {
    var status: Int
    var code: String
    var message: String

    let data: Categories
}

struct Categories: Codable, Equatable {
    let qty: Int
    let categories: [Category]

}

struct Category: Codable, Equatable {
    let name: String // name of categoty, i.e. "Ультразвукові дослідження"
    let list: [String] // names of subCategories, i.e. "Ультразвукові
дослідження нирок"

}
```

ServiceTypeModel.swift

```
import UIKit

final class ServiceTypeModel {

    var name: String
    var image: String
```

```
    var tagImage: String
    var backColor: UIColor
    var services: [String]

    init(name: String, image: String, tagImage: String, backColor: UIColor,
services: [String]) {
        self.name = name
        self.image = image
        self.backColor = backColor
        self.services = services
        self.tagImage = tagImage
    }

    func copy() -> ServiceTypeModel{
        return ServiceTypeModel.init(name: name, image: image, tagImage:
tagImage, backColor: backColor, services: services)
    }

}
```

HospitalInfoDetailModel.swift

```
import Foundation

final class HospitalInfoDetailModel {
    let infoType: DetailsInfoType
    let infoTypeStr: String
    let info: String

    init(infoType: DetailsInfoType, infoTypeStr: String, info: String) {
        self.infoType = infoType
        self.infoTypeStr = infoTypeStr
        self.info = info
    }
}
```

ServiceHolder.swift

```
import Foundation

protocol Service {}
protocol InitializableService: Service {
    init()
}

final class ServiceHolder {
    private var servicesDictionary: [String: Service] = [:]

    func add<T>(_ protocolType: T.Type, for concreteType:
InitializableService.Type, with name: String? = nil) {
        self.add(protocolType, for: concreteType.init(), with: name)
    }

    func add<T>(_ type: T.Type, with name: String? = nil, constructor: () ->
Service) {
        self.add(type, for: constructor(), with: name)
    }

    func add<T>(_ protocolType: T.Type, for instance: Service, with name:
String? = nil) {
        let name = name ?? String(reflecting: protocolType)
        servicesDictionary[name] = instance
    }
```

```
    func get<T>(by type: T.Type = T.self) -> T {
        return get(by: String(reflecting: type))
    }

    func get<T>(by name: String) -> T {
        guard let service = servicesDictionary[name] as? T else {
            fatalError("Firstly you have to add the service.")
        }

        return service
    }
}
```

AnalyticsService.swift

```
import Foundation
import FirebaseAnalytics

protocol AnalyticsServiceType: Service {
    func log(event: AnalyticsEvent, parameters: [String: Any]?)
}

class AnalyticsService: AnalyticsServiceType {

    func log(event: AnalyticsEvent, parameters: [String: Any]?) {
        Analytics.logEvent(event.rawValue, parameters: parameters)
    }
}

enum AnalyticsParameter: String, Codable {
    case TYPE
    case TEXT
}

enum AnalyticsEvent: String, Codable {
    case APP_OPEN
    case CATEGORY_OPEN // TYPE
    case ABOUT_OPEN
    case GO_TO_ONIX
    case HOSPITAL_OPEN
    case GOOGLEMAP_OPEN
    case SEARCH //TEXT
}
```

NetworkService.swift

```
import RxSwift
import Foundation

protocol NetworkServiceType: Service {

    var stypesObserver: ReplaySubject<Result<Categories>> { get }
    var hospitalsObserver: ReplaySubject<Result<[Hospital]>> { get }
    var searchObserver: ReplaySubject<Result<[Hospital]>> { get }
    var hospitalObserver: ReplaySubject<Result<Hospital>> { get }

    func getServiceTypes()
    func getSearch(text: String)
    func getHospitals(type: String)
    func getHospital(id: String)
}

final class NetworkService: NetworkServiceType {
```

```
    private let networkManager = NetworkManager.shared

    init() {
        getServiceTypes()
    }

    var stypesObserver = ReplaySubject<Result<Categories>>.create(bufferSize: 1)
    var hospitalObserver = ReplaySubject<Result<Hospital>>.create(bufferSize: 1)
    var hospitalsObserver = ReplaySubject<Result<[Hospital]>>.create(bufferSize:
1)
    var searchObserver = ReplaySubject<Result<[Hospital]>>.create(bufferSize: 1)

    func getServiceTypes() {
        let completion: ((Result<CategoriesResponse>) -> Void) = { result in

            switch result {
            case .success(let categoriesResonse):
                if categoriesResonse.status == 200 {
                    self.stypesObserver.onNext(.success(categoriesResonse.data))
                } else {

     self.stypesObserver.onNext(.failure(categoriesResonse.message))
                }
            case .failure(let error):
                self.stypesObserver.onNext(.failure(error))
            }
        }

        networkManager.getAllCategories(completion: { data in
            ParsingHelper.parsingByType(data, completion)
        })

    }

    func getSearch(text: String) {
        let completion: ((Result<SearchRequest>) -> Void) = { result in

            switch result {
            case .success(let hospitals):
                if hospitals.status == 200 {
                    if let hospitals = hospitals.data?.hospitals {
                        self.searchObserver.onNext(.success(hospitals))
                    } else {
                        self.searchObserver.onNext(.failure(hospitals.message))
                    }
                } else {
                    self.searchObserver.onNext(.failure(hospitals.message))
                }
            case .failure(let error):
                self.searchObserver.onNext(.failure(error))
            }
        }

        networkManager.getSearch(text: text, complition: { data in
            ParsingHelper.parsingByType(data, completion)
        })
    }

    func getHospitals(type: String) {

        let completion: ((Result<HospitalsRequest>) -> Void) = { result in

            switch result {
            case .success(let hospitals):
```

```
                if hospitals.status == 200 {
                    if let hospitals = hospitals.data?.hospitals {
                        self.hospitalsObserver.onNext(.success(hospitals))
                    } else {

self.hospitalObserver.onNext(.failure(hospitals.message))
                    }
                } else {
                    self.hospitalObserver.onNext(.failure(hospitals.message))
                }
            case .failure(let error):
                self.hospitalObserver.onNext(.failure(error))
            }
        }

        networkManager.getHospitals(type: type, completion: { data in
            ParsingHelper.parsingByType(data, completion)
        })

    }

    func getHospital(id: String) {

        let completion: ((Result<HospitalRequest>) -> Void) = { result in

            switch result {
            case .success(let hospital):
                if hospital.status == 200 {
                    if let firstHospital = hospital.data?["hospitals"]?.first {
                        self.hospitalObserver.onNext(.success(firstHospital))
                    } else {
                        self.hospitalObserver.onNext(.failure(hospital.message))
                    }
                } else {
                    self.hospitalObserver.onNext(.failure(hospital.message))
                }
            case .failure(let error):
                self.hospitalObserver.onNext(.failure(error))
            }
        }

        networkManager.getHospital(id: id, complition: { data in
            ParsingHelper.parsingByType(data, completion)
        })
    }
}
```

HospitalDetailesController.swift

```
import UIKit

final class HospitalDetailsController: UIViewController {
    var viewModel: HospitalDetailsModelType!

    @IBOutlet weak var tableViewShadowView: UIView!
    @IBOutlet private weak var gradientView: UIView!
    @IBOutlet private weak var lookOnMapButton: UIButton!
    @IBOutlet private weak var tableView: UITableView!
    @IBOutlet private weak var map: MapView!
    @IBOutlet private weak var closeButton: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()
        configure()
```

```
        setUpClosure()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        self.viewModel.sendEventOpen()
        navigationController?.isNavigationBarHidden = true
    }

    @IBAction func pushToMap(_ sender: Any) {
        if let point = viewModel.point {
            self.viewModel.sendEventMap()
            UrlOpenHelper.openDirections(to: point)
        }
    }

    private func configure() {
        if viewModel.isHUD {
            showHud()
        }
        configureMap()
        configureOpenMapBtn()
        configureTableView()
    }

    private func setUpClosure() {
        viewModel.didLoadData = {
            DispatchQueue.main.async {
                self.hideHUD()
                self.tableView.reloadData()
                self.configureMap()
            }
        }

        viewModel.didLoadFailed = { [weak self] error in
            DispatchQueue.main.async {
                self?.hideHUD()
                let alert = UIAlertController(title: "Error", message: error,
preferredStyle: .alert)
                alert.addAction(UIAlertAction(title: "OK", style: .default,
handler: nil))
                self?.present(alert, animated: true, completion: nil)
            }

        }
    }

    private func configureMap() {
        if let point = viewModel.point {
            map.customSetup(with: point)
            map.addMarker(coordinate: point)
            map.moveToLocation(coordinate: point)
        }

        gradientView.layer.shadowColor = UIColor.black.cgColor
        gradientView.layer.shadowOpacity = 1
        gradientView.layer.shadowOffset = CGSize(width: 0, height: 3)
        gradientView.layer.shadowRadius = 70
        gradientView.layer.shouldRasterize = true
        gradientView.layer.rasterizationScale = UIScreen.main.scale

    }

    private func configureOpenMapBtn() {
```

```
        lookOnMapButton.backgroundColor = Style.Color.mapButton

        lookOnMapButton.layer.cornerRadius = 20.0
        //lookOnMapButton.layer.borderWidth = 0.5
        //lookOnMapButton.layer.borderColor = Style.Color.borderColor.cgColor
        lookOnMapButton.layer.applySketchShadow(color: Style.Color.shadowColor,
alpha: 0.14, xxx: 0, yyy: 4, blur: 12, spread: 0)
    }

    private func configureTableView() {

        tableViewShadowView.layer.shadowColor = UIColor.lightGray.cgColor
        tableViewShadowView.layer.shadowOpacity = 0.7
        tableViewShadowView.layer.shadowOffset = .zero
        tableViewShadowView.layer.shadowRadius = 4
        tableViewShadowView.layer.shadowPath = UIBezierPath(rect:
tableViewShadowView.bounds).cgPath
        tableViewShadowView.layer.shouldRasterize = true
        tableViewShadowView.layer.rasterizationScale = UIScreen.main.scale

        tableView.register([HospitalDetailsTableViewCell.className,
TitleTableViewCell.className])
        tableView.setDataSource(self)
        tableView.layer.cornerRadius = 10.0
        self.tableView.reloadData()
    }

    @IBAction func closeBtnClicked(_ sender: UIButton) {
        viewModel.goBack()
    }
}

extension HospitalDetailsController: UITableViewDataSource {

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
-> Int {
        return viewModel.hospitalInfoDetailModels.count + 1
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

        if indexPath.row == 0 {
            guard let cell: TitleTableViewCell = tableView.dequeCell(for:
indexPath) else {
                print("can't find cell")
                return UITableViewCell()
            }
            cell.configure(title: viewModel.title)
            return cell

        } else {
            guard let cell: HospitalDetailsTableViewCell =
tableView.dequeCell(for: indexPath) else {
                print("can't find cell")
                return UITableViewCell()
            }

            cell.configure(model:
viewModel.hospitalInfoDetailModels[indexPath.row - 1])
            return cell
        }
    }
}
```

HospitalDetailsCoordinator.swift

```
import Foundation
import UIKit

protocol HospitalDetailsCoordinatorType {
    func start()
    func goBack()
}

final class HospitalDetailsCoordinator: HospitalDetailsCoordinatorType {

    private weak var controller: HospitalDetailsController? =
Storyboard.hospital.instantiateViewController()
    private let navigationController: UINavigationController?
    private var serviceHolder: ServiceHolder!

    init(navigationController: UINavigationController?, serviceHolder:
ServiceHolder) {
        self.navigationController = navigationController
        self.serviceHolder = serviceHolder

        controller?.viewModel = HospitalDetailsModel(self, serviceHolder:
self.serviceHolder)
    }

    func start() {
        if let controller = controller {
            navigationController?.pushViewController(controller, animated: true)
        }
    }

    func goBack() {
        _ = navigationController?.popViewController(animated: true)
    }

    deinit {
        print("ListCoordinator - deinit")
    }
}
```

HospitalDetailesModel.swift

```
import Foundation
import CoreLocation
import RxSwift

protocol HospitalDetailsModelType {
    var didLoadData: (() -> Void)? { get set }
    var didLoadFailed: ((String) -> Void)? { get set }

    var point: CLLocationCoordinate2D? { get }
    var hospitalInfoDetailModels: [HospitalDetailsCellViewModel] { get }
    var title: String { get set }
    var isHUD: Bool { get }

    func sendEventOpen()
    func sendEventMap()
    func goBack()
}

final class HospitalDetailsModel: HospitalDetailsModelType {

    var hospitalInfoDetailModels: [HospitalDetailsCellViewModel] = []
```

```
    var title: String = ""
    var point: CLLocationCoordinate2D?
    var isHUD: Bool = true

    private var separator = 0

    private let coordinator: HospitalDetailsCoordinatorType
    private let networkService: NetworkServiceType
    private let analiticsService: AnalyticsServiceType
    private let disposeBag = DisposeBag()

    var didLoadData: (() -> Void)?
    var didLoadFailed: ((String) -> Void)?

    init(_ coordinator: HospitalDetailsCoordinatorType, serviceHolder:
ServiceHolder) {
        self.coordinator = coordinator

        networkService = serviceHolder.get(by: NetworkServiceType.self)
        analiticsService = serviceHolder.get(by: AnalyticsServiceType.self)
        networkService.hospitalObserver.subscribe(onNext: { [weak self] result
in
            switch result {
            case .success(let model):
                self?.update(model: model)
                print("Отримано відповідь про лікарню з id = ", model.id)
                self?.isHUD = false
                self?.didLoadData?()
            case .failure(error: let error):
                self?.didLoadFailed?(error)
                print("Отримано помилку", error.description)
            }
        }).disposed(by: disposeBag)
    }

    func goBack() {
        coordinator.goBack()
    }

    private func update(model: Hospital) {
        if let name = model.name {
            self.title = name
        }
        if let latit = model.lat, !latit.contains("null"), let lat =
Double(latit),
            let lanit = model.lan, !lanit.contains("null"), let lan =
Double(lanit) {
            self.point = CLLocationCoordinate2D.init(latitude: lat, longitude:
lan)
        }

        self.hospitalInfoDetailModels = []
        if let addressa = model.adress, !addressa.contains("null") {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .address, infoTypeStr: "Вулиця та номер
будинку:", info: addressa))
        }

        if let workTime = model.workTime, !workTime.contains("null") {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .schedule, infoTypeStr: "Графік работи:",
info: workTime))
        }
```

```
        if let unworkTime = model.unworkTime, !unworkTime.contains("null") {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .closedTime, infoTypeStr: "Обмеження
прийому:", info: unworkTime))
        }

        if let equipName = model.equipName, !equipName.contains("null") {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .none, infoTypeStr: "Назва обладнання:",
info: equipName))
        }

        if let stuctName = model.structureName, !stuctName.contains("null") {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .none, infoTypeStr: "Назва структурного
підрозділу:", info: stuctName))
        }

        if let floorNumber = model.floorNumber, floorNumber != 0 {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .none, infoTypeStr: "Номер поверху:",
info: "\(floorNumber)"))
        }

        if let roomNumber = model.roomNumber, roomNumber != 0 {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .none, infoTypeStr: "Номер кабінету:",
info: "\(roomNumber)"))
        }

        if let equipCountry = model.equipCountry, !equipCountry.contains("null")
{
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .none, infoTypeStr: "Країна виробник:",
info: equipCountry))
        }
        if let eqipYear = model.eqipYear, !eqipYear.contains("null") {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .none, infoTypeStr: "Рік випуску
обладнання:", info: eqipYear))
        }

        if let equipCondition = model.equipCondition,
!equipCondition.contains("null") {
            self.hospitalInfoDetailModels.append(
HospitalDetailsCellViewModel(infoType: .none, infoTypeStr: "Експлуатаційний стан
обладнання:", info: equipCondition))
        }

        setCellSeparator(self.hospitalInfoDetailModels)
    }

    func sendEventOpen() {
        analiticsService.log(event: .HOSPITAL_OPEN, parameters: nil)
    }

    func sendEventMap() {
        analiticsService.log(event: .GOOGLEMAP_OPEN, parameters: nil)
    }

    func setCellSeparator(_ hospitals: [HospitalDetailsCellViewModel]) {

        hospitals.forEach { hospital in
            if hospital.infoType == .none {
```

```
                separator += 1
            } else {
                separator = 0
            }
            hospital.separator = (separator >= 2)
        }
    }
}
```

ListViewController.swift

```
import UIKit

final class ListViewController: UIViewController {
    var viewModel: ListViewModelType!

    @IBOutlet weak var tableViewShadowView: UIView!
    @IBOutlet private weak var tableView: UITableView!

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        configureNavigationBar()
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        configure()
        setUpClosure()
    }

    private func setUpClosure() {

        viewModel.didLoadData = {
            DispatchQueue.main.async { [weak self] in
                self?.hideHUD()
                self?.tableView.reloadData()
            }
        }

        viewModel.didLoadFailed = { [weak self] error in
            DispatchQueue.main.async {
                self?.hideHUD()
                let alert = UIAlertController(title: "Error", message: error,
preferredStyle: .alert)
                alert.addAction(UIAlertAction(title: "OK", style: .default,
handler: nil))
                self?.present(alert, animated: true, completion: nil)
            }
        }
    }

    private func configure() {

        tableViewShadowView.layer.shadowColor = UIColor.darkGray.cgColor
        tableViewShadowView.layer.shadowOpacity = 0.5
        tableViewShadowView.layer.shadowOffset = .zero
        tableViewShadowView.layer.shadowRadius = 5
        tableViewShadowView.layer.shadowPath = UIBezierPath(rect:
tableViewShadowView.bounds).cgPath
        tableViewShadowView.layer.shouldRasterize = true
        tableViewShadowView.layer.rasterizationScale = UIScreen.main.scale
```

```
        self.view.backgroundColor =
navigationController?.navigationBar.barTintColor

        tableView.layer.cornerRadius = 10.0
        tableView.register([ServiceTableViewCell.className,
HospitalCell.className])
        tableView.setDataSource(self, delegate: self)

        tableView.reloadData()

        if viewModel.isHUD {
            showHud()
        }
    }

    private func configureNavigationBar() {
        navigationController?.isNavigationBarHidden = false
        navigationController?.navigationBar.isTranslucent = false
        navigationController?.navigationBar.shadowImage = UIImage()

        let backBarButton = UIBarButtonItem(title: "", style: .plain, target:
self, action: nil)
        navigationItem.backBarButtonItem = backBarButton

        let btn = UIButton(type: .custom)
        btn.frame = Style.Size.backBtnFrame
        btn.addTarget(self, action: #selector(backBtnClicked), for:
.touchUpInside)
        btn.setImage(Style.Images.backIcon, for: .normal)
        navigationItem.leftBarButtonItem = UIBarButtonItem(customView: btn)

        let titleLabel = UILabel(frame: CGRect(x: 0, y: 0, width:
view.frame.width - 50, height: 50))
        titleLabel.font = Style.Font.navTitleFont
        titleLabel.textAlignment = .justified
        titleLabel.textColor = .white
        titleLabel.text = viewModel.screenTitle
        navigationItem.titleView = titleLabel
    }

    @objc
    func backBtnClicked() {
        viewModel.goBack()
    }
}

extension ListViewController: UITableViewDelegate {
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        switch viewModel.screenType {
        case .serviceDetails:
            viewModel.openHospitals(row: indexPath.row)
        case . hospitals:
            viewModel.openDetails(row: indexPath.row)
        }
    }
}

extension ListViewController: UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
-> Int {
        switch viewModel.screenType {
        case .serviceDetails:
```

```
            return viewModel.serviceDetailsModels.count
        case . hospitals:
            return viewModel.hospitalModels.count
        }
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
        switch viewModel.screenType {
        case .serviceDetails:
            guard let cell: ServiceTableViewCell = tableView.dequeCell(for:
indexPath) else {
                print("can't find cell")
                return UITableViewCell()
            }
            cell.configure( viewModel.serviceDetailsModels[indexPath.row])
            return cell
        case .hospitals:
            guard let cell: HospitalCell = tableView.dequeCell(for: indexPath)
else {
                print("can't find cell")
                return UITableViewCell()
            }
            cell.configure(model: viewModel.hospitalModels[indexPath.row])
            return cell
        }
    }
}
```

ListCoordinator.swift

```
import UIKit

protocol ListCoordinatorType {
    func start()

    func openHospitals(model: ServiceTypeModel, _ serviceIndex: Int)
    func openDetails()
    func goBack()
}

final class ListCoordinator: ListCoordinatorType {

    private weak var controller: ListViewController? =
Storyboard.list.instantiateViewController()
    private let navigationController: UINavigationController?
    private var serviceHolder: ServiceHolder!

    init(navigationController: UINavigationController?, serviceHolder:
ServiceHolder, _ screenType: ListScreenType = .serviceDetails, serviceTypeModel:
ServiceTypeModel, _ serviceIndex: Int = 0) {

        self.navigationController = navigationController
        self.serviceHolder = serviceHolder

        controller?.viewModel = ListViewModel(self, serviceHolder:
self.serviceHolder, screenType: screenType, serviceTypeModel: serviceTypeModel,
serviceIndex: serviceIndex)
    }

    func start() {
        if let controller = controller {
            navigationController?.pushViewController(controller, animated: true)
        }
```

```
    }

    func openHospitals(model: ServiceTypeModel, _ serviceIndex: Int) {
        let coordinator = ListCoordinator(navigationController:
navigationController, serviceHolder: serviceHolder, .hospitals,
serviceTypeModel: model, serviceIndex)

        coordinator.start()
    }

    func openDetails() {
        let coordinator = HospitalDetailsCoordinator(navigationController:
navigationController, serviceHolder: serviceHolder)
        coordinator.start()
    }

    func goBack() {
        _ = navigationController?.popViewController(animated: true)
    }

    deinit {
        print("ListCoordinator - deinit")
    }

}
```

ListViewModel.swift

```
import Foundation
import RxSwift

enum ListScreenType {
    case serviceDetails
    case hospitals
}

protocol ListViewModelType {
    var didLoadData: (() -> Void)? { get set }
    var didLoadFailed: ((String) -> Void)? { get set }

    var screenType: ListScreenType { get set }
    var screenTitle: String { get set }

    var serviceModel: ServiceTypeModel { get set }
    var serviceDetailsModels: [String] { get }
    var hospitalModels: [HospitalModel] { get }
    var isHUD: Bool { get }

    func openHospitals(row: Int)
    func openDetails(row: Int)
    func goBack()
}

final class ListViewModel: ListViewModelType {

    private let coordinator: ListCoordinatorType
    private let networkService: NetworkServiceType
    private let disposeBag = DisposeBag()

    var didLoadData: (() -> Void)?
    var didLoadFailed: ((String) -> Void)?

    var screenType: ListScreenType
    var screenTitle: String
```

```
    var serviceModel: ServiceTypeModel
    var isHUD: Bool

    var serviceDetailsModels: [String] = []
    var hospitalModels: [HospitalModel] = []

    init(_ coordinator: ListCoordinatorType, serviceHolder: ServiceHolder,
screenType: ListScreenType = .serviceDetails, serviceTypeModel:
ServiceTypeModel, serviceIndex: Int) {
        self.coordinator = coordinator
        networkService = serviceHolder.get(by: NetworkServiceType.self)

        self.screenType = screenType
        serviceModel = serviceTypeModel


        switch screenType {
        case .serviceDetails:
            self.screenTitle = serviceTypeModel.name
            serviceDetailsModels = serviceModel.services
            self.isHUD = false
        case .hospitals:
                self.isHUD = true
                self.screenTitle = serviceTypeModel.services[serviceIndex]
               networkService.hospitalsObserver.subscribe(onNext: { [weak self]
result in
                switch result {
                case .success(let model):
                    self?.hospitalModels = []
                    model.forEach { element in
                        self?.hospitalModels.append(HospitalModel.init(model:
element))
                    }
                    self?.didLoadData?()
                case .failure(error: let error):
                    self?.didLoadFailed?(error)
                }
            }).disposed(by: disposeBag)
        }
    }

    func openHospitals(row: Int) {
        networkService.getHospitals(type: serviceDetailsModels[row])
        coordinator.openHospitals(model: serviceModel, row)
    }

    func openDetails(row: Int) {
        networkService.getHospital(id: hospitalModels[row].id)
        coordinator.openDetails()
    }

    func goBack() {
        coordinator.goBack()
    }
}
```

MainViewController.swift

```
import UIKit

final class MainViewController: UIViewController {

    var viewModel: MainViewModelType!
```

```
    @IBOutlet private weak var titleLabel: UILabel!
    @IBOutlet private weak var searchBar: UISearchBar!
    @IBOutlet private weak var tableView: UITableView!
    @IBOutlet private weak var searchResult: SearchResultView!
    @IBOutlet private weak var cancelButton: UIButton!

    @IBOutlet private weak var aboutButton: UIButton!

    @IBAction func cancelTapped(_ sender: Any) {
        configureSearchMode(false)
        searchBar.endEditing(true)
        searchBar.text = ""
    }

    @IBAction func aboutBtnClicked(_ sender: UIButton) {
        self.viewModel.sendEventOpenAbout()
        let text = "Onix-Systems \n\n App for search medical equipment and
services in Kropyvnitskyi \n\n iOS developers: \n Olena Drobko, Tetiana
Nieizviestna \n\n  Designer: \n Max Honcharov"
        let alert = UIAlertController(title: "About", message: text,
preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "Terms of use & Privacy Policy",
style: .default, handler: { _ in
            if let url = URL(string: "https://onix-systems-krop-hakaton-
2019.staging.onix.ua/terms-of-use-and-privacy-policy") {
                self.viewModel.sendEventOnix()
                UIApplication.shared.open(url)
            }
        }))
        alert.addAction(UIAlertAction(title: "Onix-Systems.com", style:
.default, handler: { _ in
            if let url = URL(string: "https://onix-systems.com") {
                self.viewModel.sendEventOnix()
                UIApplication.shared.open(url)
            }
        }))
        alert.addAction(UIAlertAction(title: "OK", style: .default, handler:
nil))

        self.present(alert, animated: true, completion: nil)
    }

    @IBOutlet private weak var cancelLeading: NSLayoutConstraint!
    @IBOutlet private weak var searchBarTraling: NSLayoutConstraint!
    @IBOutlet private weak var cancelTraling: NSLayoutConstraint!

    override func viewDidLoad() {
        super.viewDidLoad()
        showHud()
        configure()
        setupClosure()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        configureNavigationBar()
        self.viewModel.sendEventApp()
    }

    private func configureNavigationBar() {
        navigationController?.isNavigationBarHidden = true
        navigationController?.navigationBar.isTranslucent = true
    }
```

```
    private func configure() {
        tableView.register([ServiceCell.identifier])
        tableView.setDataSource(self, delegate: self)
        tableView.reloadData()

        searchBar.delegate = self
        searchBar.isUserInteractionEnabled = true
        searchResult.configure()
    }

    private func setupClosure() {
        searchResult.didSelected = { [weak self] row in
            DispatchQueue.main.async {
                self?.hideHUD()
                self?.viewModel.openDetails(row, text: self?.searchBar.text ??
"")
            }
        }

        viewModel.didLoadData = {
            DispatchQueue.main.async { [weak self] in
                self?.hideHUD()
                self?.tableView.reloadData()
                self?.searchResult.update(self?.viewModel.searchModels ?? [])
            }
        }

        viewModel.didLoadFailed = { [weak self] error in
            DispatchQueue.main.async {
                self?.hideHUD()
                let alert = UIAlertController(title: "Error", message: error,
preferredStyle: .alert)
                alert.addAction(UIAlertAction(title: "OK", style: .default,
handler: nil))
                self?.present(alert, animated: true, completion: nil)
            }

        }
    }

    private func configureSearchMode(_ state: Bool) {
        showCancel(state)
        searchResult.configure()
        searchResult.isHidden = !state
        tableView.isHidden = state
        titleLabel.isHidden = state
        if state {
            searchResult.clean()
        }
    }

    private func showCancel(_ state: Bool) {
        cancelButton.isHidden = !state
        cancelTraling.isActive = state
        cancelTraling.isActive = state
        searchBarTraling.isActive = !state
    }
}

extension MainViewController: UITableViewDelegate {
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        viewModel.openList(row: indexPath.row)
```

```
    }
}

extension MainViewController: UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
-> Int {
        return viewModel.serviceModels.count
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {
        return .init(signOf: tableView.bounds.size.width, magnitudeOf:
tableView.bounds.size.width / 2.3)
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

        guard let cell: ServiceCell = tableView.dequeCell(for: indexPath) else {
return UITableViewCell() }
        cell.configure(viewModel.serviceModels[indexPath.row])

        return cell
    }
}

extension MainViewController: UISearchBarDelegate {

    func searchBarShouldBeginEditing(_ searchBar: UISearchBar) -> Bool {
        configureSearchMode(true)
        return true
    }

    func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
        viewModel.search(text: searchBar.text)
    }

    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
        searchBar.endEditing(true)
    }
}
```

MainCoordinator.swift

```
import UIKit

protocol MainCoordinatorType {
    func start()
    func openList(model: ServiceTypeModel)
    func openDetails()
}

final class MainCoordinator: MainCoordinatorType {

    private weak var controller: MainViewController? =
Storyboard.main.instantiateViewController()
    private let navigationController: UINavigationController?
    private var serviceHolder: ServiceHolder!

    init(navigationController: UINavigationController?, serviceHolder:
ServiceHolder) {
        self.navigationController = navigationController
        self.serviceHolder = serviceHolder
```

```
        controller?.viewModel = MainViewModel(self, serviceHolder:
self.serviceHolder)
    }

    func start() {
        if let controller = controller {
            navigationController?.pushViewController(controller, animated: true)
        }
    }

    func openDetails() {
        let coordinator = HospitalDetailsCoordinator(navigationController:
navigationController, serviceHolder: serviceHolder)
        coordinator.start()
    }

    func openList(model: ServiceTypeModel) {
        let coordinator = ListCoordinator(navigationController:
navigationController, serviceHolder: serviceHolder, .serviceDetails,
serviceTypeModel: model)
        navigationController?.navigationBar.barTintColor = model.backColor
        coordinator.start()
    }

    deinit {
        print("MainCoordinator - deinit")
    }

}
```

MainViewModel.swift

```
import Foundation
import RxSwift

protocol MainViewModelType {
    var didLoadData: (() -> Void)? { get set }
    var didLoadFailed: ((String) -> Void)? { get set }

    var serviceModels: [ServiceTypeModel] { get }
    var searchModels: [HospitalModel] { get }

    func openList(row: Int)
    func openDetails(_ index: Int, text: String)
    func search(text: String?)

    func sendEventApp()
    func sendEventCategory(type: String)
    func sendEventOpenAbout()
    func sendEventOnix()
    func sendEventSearch(text: String)
}

final class MainViewModel: MainViewModelType {
    var serviceModels: [ServiceTypeModel] = []
    var searchModels: [HospitalModel] = []

    private let coordinator: MainCoordinatorType
    private let networkService: NetworkServiceType
    private let analiticsService: AnalyticsServiceType
    private let disposeBag = DisposeBag()

    var didLoadData: (() -> Void)?
    var didLoadFailed: ((String) -> Void)?
```

```
    init(_ coordinator: MainCoordinatorType, serviceHolder: ServiceHolder) {
        self.coordinator = coordinator

        analiticsService = serviceHolder.get(by: AnalyticsServiceType.self)
        networkService = serviceHolder.get(by: NetworkServiceType.self)

        networkService.stypesObserver.subscribe(onNext: { [weak self] result in
            switch result {
            case .success(let categories):
                self?.createServiceTypesArray(categories: categories)
                self?.didLoadData?()
            case .failure(error: let error):
                self?.didLoadFailed?(error)
            }
        }).disposed(by: disposeBag)

        networkService.searchObserver.subscribe(onNext: { [weak self] result in
            switch result {
            case .success(let searchModels):
                self?.searchModels = []
                searchModels.forEach { element in
                    self?.searchModels.append(HospitalModel.init(model:
element))
                }
                self?.didLoadData?()
            case .failure(error: let error):
                self?.didLoadFailed?(error)
            }
        }).disposed(by: disposeBag)
    }

    private func createServiceTypesArray(categories: Categories) {
        serviceModels = []
        categories.categories.enumerated().forEach { category in
            let style = Style.setCellStyle(category: category.element)
            let serviceType = ServiceTypeModel(name: category.element.name,
image: style.0, tagImage: style.1, backColor: style.2, services:
category.element.list)
            serviceModels.append(serviceType)
        }

        serviceModels = filterServices(serviceModels: serviceModels)
    }

    private func filterServices(serviceModels: [ServiceTypeModel]) ->
[ServiceTypeModel] {
        var startModels = serviceModels
        var filteredModels: [ServiceTypeModel] = []
        let parametrs = ["Ультразвук", "Рентген", "Ендоскоп", "Функціонал"]
        for filter in parametrs {
            for i in 0...startModels.count - 1 {
                if startModels[i].name.contains(filter) {
                    filteredModels.append(startModels[i])
                    startModels.remove(at: i)
                    break
                }
            }
        }
        return filteredModels + startModels
    }

    func search(text: String?) {
        networkService.getSearch(text: text ?? "")
```

```
    }

    func openDetails(_ index: Int, text: String) {
        sendEventSearch(text: text)
        networkService.getHospital(id: searchModels[index].id)
        coordinator.openDetails()
    }

    func openList(row: Int) {
        sendEventCategory(type: serviceModels[row].name)
        coordinator.openList(model: serviceModels[row])
    }

    func sendEventSearch(text: String) {
        analiticsService.log(event: .SEARCH, parameters:
[AnalyticsParameter.TEXT.rawValue: text])
    }

    func sendEventApp() {
        analiticsService.log(event: .APP_OPEN, parameters: nil)
    }

    func sendEventCategory(type: String) {
        analiticsService.log(event: .CATEGORY_OPEN, parameters:
[AnalyticsParameter.TYPE.rawValue: type])
    }

    func sendEventOpenAbout() {
        analiticsService.log(event: .ABOUT_OPEN, parameters: nil)
    }

    func sendEventOnix() {
        analiticsService.log(event: .GO_TO_ONIX, parameters: nil)
    }
}
```

MapPin.swift

```
import MapKit

class Pin: NSObject, MKAnnotation {
  let title: String? = ""
  let locationName: String? = ""
  let discipline: String? = ""
  let coordinate: CLLocationCoordinate2D

  init(coordinate: CLLocationCoordinate2D) {
    self.coordinate = coordinate
    super.init()
  }
}
```

MapView.swift

```
import UIKit
import MapKit

final class MapView: UIView {

    @IBOutlet private weak var ivMap: MKMapView!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        self.nibSetup()
```

```
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        self.nibSetup()
    }

    func customSetup(with coordinate: CLLocationCoordinate2D) {

        let regionRadius: CLLocationDistance = 100
        let coordinateRegion = MKCoordinateRegion(center: coordinate,
                                                  latitudinalMeters:
regionRadius, longitudinalMeters: regionRadius)
        ivMap.setRegion(coordinateRegion, animated: true)
        ivMap.isZoomEnabled = true
        ivMap.isScrollEnabled = true
    }

    func moveToLocation(coordinate: CLLocationCoordinate2D) {
        ivMap.isZoomEnabled = true

        let regionRadius: CLLocationDistance = 100
        let coordinateRegion = MKCoordinateRegion(center: coordinate,
                                                  latitudinalMeters:
regionRadius, longitudinalMeters: regionRadius)
        ivMap.setRegion(coordinateRegion, animated: true)
    }

    func addMarker(coordinate: CLLocationCoordinate2D) {
        let artwork = Pin(coordinate: coordinate)
        ivMap.addAnnotation(artwork)
    }
}
```

SearchResultView.swift

```
import UIKit

final class SearchResultView: UIView {
    @IBOutlet private weak var tableView: UITableView!

    var models: [HospitalModel] = []
    var didSelected: ((Int) -> Void)?

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        self.nibSetup()

    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        self.nibSetup()
    }

    func clean() {
        models = []
    }

    func configure() {
        tableView.register([HospitalCell.identifier])
        tableView.setDataSource(self, delegate: self)
        self.tableView.reloadData()
    }
```

```
    func update(_ models: [HospitalModel]) {
        self.models = models
        self.tableView.reloadData()
    }
}

extension SearchResultView: UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
-> Int {
        return models.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
        guard let cell: HospitalCell = tableView.dequeCell(for: indexPath) else
{
            print("can't find cell")
            return UITableViewCell()
        }
        if indexPath.row < models.count {
        cell.configure(model: models[indexPath.row])
        }
        return cell
    }
}

extension SearchResultView: UITableViewDelegate {
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath){
        self.didSelected?(indexPath.row)
    }
}
```

TitleTableViewCell.swift

```
import UIKit

final class TitleTableViewCell: UITableViewCell {

    @IBOutlet private weak var titleLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()

    }

    func configure(title: String) {
        titleLabel.text = title
    }

}
```

HospitalDetailsTableViewCell.swift

```
import UIKit

enum DetailsInfoType: String {
    case address
    case schedule
    case closedTime
    case none
}
```

```
protocol HospitalDetailsCellViewModelType: AnyObject {
    var infoType: DetailsInfoType { get set }
    var infoTypeStr: String { get set }
    var info: String { get set }
    var separator: Bool { get set }
}

final class HospitalDetailsCellViewModel: HospitalDetailsCellViewModelType {
    var infoType: DetailsInfoType
    var infoTypeStr: String
    var info: String
    var separator = false

    init(infoType: DetailsInfoType, infoTypeStr: String, info: String) {
        self.infoType = infoType
        self.infoTypeStr = infoTypeStr
        self.info = info
    }
}

final class HospitalDetailsTableViewCell: UITableViewCell {

    @IBOutlet private weak var infoTypeIcon: UIImageView!
    @IBOutlet private weak var infoTypeLabel: UILabel!
    @IBOutlet private weak var infoLabel: UILabel!
    @IBOutlet weak var leadingSpaceDetailsView: NSLayoutConstraint!

    @IBOutlet private weak var imageWidthConstraint: NSLayoutConstraint!

    @IBOutlet private weak var topSeparatorView: UIView!
    @IBOutlet private weak var bottomSeparatorView: UIView!

    override func awakeFromNib() {
        super.awakeFromNib()

    }

    func configure(model: HospitalDetailsCellViewModelType) {
        imageWidthConstraint.constant = model.infoType == .none ? 0 : 20
        leadingSpaceDetailsView.constant = model.infoType == .none ? 0 : 10

        switch model.infoType {
        case .address:
            infoTypeIcon.image = Style.Images.locationIcon
        case .schedule:
            infoTypeIcon.image = Style.Images.scheduleIcon
        case .closedTime:
            infoTypeIcon.image = Style.Images.closedTimeIcon
        case .none:
            break
        }

        infoTypeLabel.text = model.infoTypeStr
        infoLabel.text = model.info

        if model.separator {
            topSeparatorView.isHidden = true
            bottomSeparatorView.isHidden = false
        } else {
            topSeparatorView.isHidden = model.infoType != .none
            bottomSeparatorView.isHidden = model.infoType != .none
        }
    }
}
```

HospitalCell.swift

```
import UIKit

final class HospitalCell: UITableViewCell {

    @IBOutlet weak var border: UIView!
    @IBOutlet private weak var hospitalNameLabel: UILabel!
    @IBOutlet private weak var wrapperView: UIView!
    @IBOutlet private weak var locationLabel: UILabel!
    @IBOutlet private weak var workTimeLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    func configure(model: HospitalModel) {
        border.backgroundColor = UIColor(red: 0.8, green: 0.82, blue: 0.84,
alpha: 0.5)
        hospitalNameLabel.text = model.title

        if model.address.contains("null") {
            locationLabel.text = "інформація відсутня"
        } else {
            locationLabel.text = model.address
        }
        if model.schedule.contains("null") {
            workTimeLabel.text = "інформація відсутня"
        } else {
            workTimeLabel.text = model.schedule
        }
    }

}
```

HospitalServiceTableViewCell.swift

```
import UIKit

final class ServiceTableViewCell: UITableViewCell {

    @IBOutlet private weak var nameLabel: UILabel!
    @IBOutlet weak var border: UIView!

    override func awakeFromNib() {
        super.awakeFromNib()

    }

    func configure(_ serviceName: String) {
        border.backgroundColor = UIColor(red: 0.8, green: 0.82, blue: 0.84,
alpha: 0.5)

        let uzd = "УЗД"
        let isUzd = hasStrUZD(string: serviceName)
        let typeStr = isUzd ? uzd : ""
        var name = serviceName

        if serviceName.hasPrefix(uzd) {
            name = serviceName.removing(charactersOf: uzd)
        }

        let font = UIFont.sfRoundedSemibold(17)
```

```
        var attributes: [NSAttributedString.Key: Any]? =
[NSAttributedString.Key.font: font,
                                                              .foregroundColor:
UIColor.black]

        let attributedName = NSAttributedString(string: name, attributes:
attributes)

        attributes = [NSAttributedString.Key.font: font,
                      .foregroundColor: UIColor.lightGray]


        let attributedSpeck = NSAttributedString(string: typeStr, attributes:
attributes)

        let combination = NSMutableAttributedString()
        if isUzd { combination.append(attributedSpeck) }
        combination.append(attributedName)
        nameLabel.attributedText = combination

    }

    private func hasStrUZD(string: String) -> Bool {
        return string.hasPrefix("УЗД ")
    }

}
```

ServiceCell.swift

```
import UIKit
import TagListView

final class ServiceCell: UITableViewCell {

    @IBOutlet weak var tagsImageView: UIImageView!
    @IBOutlet weak var tagsListView: TagListView!
    @IBOutlet private weak var servicesLabel: UILabel!
    @IBOutlet private weak var wrapperView: UIView!
    @IBOutlet private weak var serviceTypeLabel: UILabel!
    @IBOutlet private weak var serviceImageView: UIImageView!

    var model: ServiceTypeModel!

    override func awakeFromNib() {
        super.awakeFromNib()

        serviceImageView.layer.cornerRadius = Style.Radius.defaultRadius
        wrapperView.layer.cornerRadius = 11.0

        tagsListView.delegate = self
    }

    func configure(_ model: ServiceTypeModel) {
        self.model = model.copy()
        self.wrapperView.backgroundColor = self.model.backColor

        self.textChanger()
        self.serviceTypeLabel.text = self.model.name
        self.serviceImageView.image = UIImage(named: self.model.image) ??
UIImage(named: "redCross")
        self.tagsImageView.image = UIImage(named: self.model.tagImage) ??
UIImage()
    }
```

```
    private func textChanger() {
        let wordToRemove = "дослідження"
        if let range = self.model.name.range(of: wordToRemove) {
            self.model.name.removeSubrange(range)
        }
        if self.model.name.contains("Мамографічне") {
            self.model.name = "Мамографічні"
        }
    }

    func getSize() -> Int {
        switch UIScreen.main.nativeBounds.height {
        case 1136:
            return 30
        case 1334:
            return 60
        case 1920, 2208:
            return 95
        case 2436:
            return 60
        case 2688:
            return 95
        case 1792:
            return 95
        default:
            return 50
        }
    }
}

extension ServiceCell: TagListViewDelegate {}
```