ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Мобільний додаток для електронного запису до лікаря на платформі Android із використанням Firebase»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Єлизавета БОНДАРЕНКО
(підпис)

Виконала: здобувачка вищої освіти групи ТЦР-44
Єлизавета БОНДАРЕНКО
Керівник: Вікторія ЖЕБКА
д-р техн. наук, проф.
Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бондаренко Єлизаветі Романівні

1. Тема кваліфікаційної роботи: «Мобільний додаток для електронного запису до лікаря на платформі Android із використанням Firebase»
Керівник кваліфікаційної роботи д-р техн. наук, проф. Вікторія ЖЕБКА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року №81.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості з розробки мобільних додатків на платформі Android Studio, принципи проєктування користувацьких інтерфейсів, архітектурні підходи до побудови Android-додатків, а також технічна документація щодо використання сервісів Firebase, зокрема бази даних.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд та аналіз існуючих мобільних додатків та технологій для електронного запису до лікаря, а також засобів розробки на платформі Android Studio.
2. Проєктування мобільного додатка для електронного запису до лікаря з використанням сучасних підходів до розробки інтерфейсу користувача та архітектури застосунка.

3. Програмна реалізація та опис функціоналу мобільного додатка для електронного запису до лікаря із використанням сервісів Firebase.
4. Тестування мобільного додатку та аналіз його роботи.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів
2. Вимоги до програмного забезпечення
3. Програмні засоби реалізації
4. Діаграма варіантів використання
5. Алгоритм роботи застосунку
6. Діаграма класів
7. Екранні форми застосунку
8. Апробація результатів

6. Дата видачі завдання «_20_» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	20.02. – 20.03.2026	
2	Аналіз та дослідження існуючих аналогів	21.03. – 28.03.2026	
3	Огляд існуючих мобільних додатків та технологій для електронного запису до лікаря.	29.03. – 05.04.2026	
4	Проєктування мобільного додатку вибору лікаря та запису на прийом	06.04. – 13.04.2026	
5	Програмна реалізація застосунку	14.04. - 21.04.2026	
6	Тестування застосунку	22.04. – 29.04. 2026	
7	Оформлення роботи: вступ, висновки, реферат	30.05 – 07.05.2026	
8	Розробка демонстраційних матеріалів	08.05 – 15.05.2026	
9	Попередній захист роботи	18.05.2026	

Здобувачка вищої освіти _____ Єлизавета БОНДАРЕНКО

(підпис)

Керівник

кваліфікаційної роботи _____ Вікторія ЖЕБКА

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 3 табл., 13 рис., 14 джерел.

Мета роботи – підвищення зручності, швидкості та ефективності взаємодії користувачів із медичними установами шляхом розробки сучасного мобільного додатка та реалізації функціоналу онлайн-запису до лікаря, перегляду інформації про спеціалістів, а також персоналізованого профілю користувача.

Об'єкт дослідження – процес організації та автоматизації взаємодії пацієнтів із медичними установами під час запису на прийом до лікаря. Об'єкт дослідження охоплює способи надання медичних послуг у цифровому середовищі, механізми обробки інформації про лікарів, така як спецалізація, досвід роботи та кількість .прийнятих пацієнтів, а також сучасні підходи до використання інформаційних технологій у сфері охорони здоров'я.

Предмет дослідження – методи та засоби розробки мобільного застосунку для вибору лікаря, запису на прийом та комунікації пацієнта з лікарем за допомогою телефонного зв'язку або месенджерів. Предмет дослідження включає проєктування функціоналу застосунку, користувацького інтерфейсу, системи взаємодії користувачів і реалізацію онлайн-запису до лікаря.

Короткий зміст роботи – у роботі проаналізовано існуючі мобільні та вебзастосунки для організації запису на прийом до лікаря, зокрема Helsi, Doc.ua та Medics. Розглянуто сучасні підходи до розробки мобільних застосунків із використанням середовища Android Studio та хмарних сервісів Firebase.

Розроблено алгоритм функціонування застосунку та програмно реалізовано основні функціональні можливості, а саме: перегляд списку лікарів, фільтрацію за категоріями, перегляд детальної інформації про лікаря, запис на прийом, обмін повідомленнями, здійснення телефонного дзвінка, відображення місцезнаходження клініки та взаємодію з базою даних. Проведено налаштування хмарних сервісів Firebase для зберігання та обробки даних

мобільного додатка. Розгортання застосунку здійснено та протестовано на мобільному пристрої Google Pixel 10 Pro.

Сферою використання застосунку є пошук і вибір лікарів для запису на прийом у медичних установах.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, ANDROID, ANDROID STUDIO, FIREBASE, ВИБІР ЛІКАРЯ, ЗАПИС НА ПРИЙОМ, БАЗА ДАНИХ, ХМАРНІ ТЕХНОЛОГІЇ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, МЕДИЧНИЙ СЕРВІС

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

iOS – iPhone Operating System;

XML – Extensible Markup Language;

URL – Uniform Resource Locator;

MVVM – Model-View-ViewModel;

UI – User Interface;

UX – User Experience.

ВСТУП

Актуальність теми – в умовах стрімкого розвитку інформаційних технологій та цифровізації суспільства – зростає потреба у впровадженні сучасних рішень. Зокрема, це актуально для сфери медичних послуг. Однією з актуальних проблем залишається організація запису пацієнтів на прийом до лікаря. Традиційні методи надання медичних послуг, такі як телефонні дзвінки або особисте відвідування спеціалізованих закладів для уточнення вільних годин та дат – малоефективні. Вони займають значну кількість часу та не є зручними для пацієнтів. Також такі підходи можуть призводити до перевантаження реєстратур, помилок у записах, утворення черг та збільшення захворюваності.

Сучасні мобільні технології відкривають нові можливості для вирішення глобальних проблем людства. Використання мобільних додатків дозволяє автоматизувати процес вибору лікаря та запису на прийом, забезпечуючи доступ до необхідної інформації у будь-який час та з будь-якого місця. Це сприяє підвищенню якості обслуговування пацієнтів, оптимізації роботи медичних установ та більш раціональному використанню ресурсів.

Згідно з дослідженнями у сфері цифрової медицини, впровадження мобільних сервісів дозволяє підвищити доступність медичних послуг та зменшити навантаження на медичний персонал. Крім того, використання електронних систем запису сприяє оптимізації процесів обслуговування пацієнтів та підвищенню ефективності роботи медичних установ.

Особливо актуальним використання мобільного додатка є для окремих категорій населення, зокрема для батьків з малолітніми дітьми та людей похилого віку. Для цих груп населення важливо мати швидкий і зручний доступ до медичних послуг, без необхідності тривалого очікування або особистого відвідування медичних установ. Використання мобільного додатка дозволяє значно спростити процес запису на прийом та підвищити рівень доступності медичної допомоги.

Застосування хмарних технологій, в даному випадку Firebase, дозволяє забезпечити надійне зберігання даних, їх синхронізацію між пристроями та

швидкий доступ до інформації в режимі реального часу. Використання даної платформи спрощує процес розробки, оскільки надає готові рішення для роботи з базами даних, автентифікації користувачів та обробки запитів без необхідності створення власного серверного середовища. Це, у свою чергу, підвищує ефективність розробки та зменшує витрати часу на реалізацію функціоналу додатка.

Розробка мобільного додатка з використанням середовища Android Studio є доцільною, оскільки воно надає широкий набір інструментів для створення, тестування та налагодження мобільних застосунків. Ця платформа є однією з найпоширеніших серед користувачів, що забезпечує широке охоплення аудиторії. Крім того, використання сучасних підходів до розробки дозволяє створювати зручні, функціональні та адаптивні інтерфейси, що відповідають потребам користувачів. Підсумовуючи, треба сказати, що створення мобільного додатка для вибору лікаря та запису на прийом є актуальним завданням, що відповідає сучасним вимогам цифрового суспільства та сприяє підвищенню доступності медичних послуг.

Мета роботи – підвищення зручності, швидкості та ефективності взаємодії користувачів із медичними установами шляхом розробки сучасного мобільного додатка та реалізації функціоналу онлайн-запису до лікаря, перегляду інформації про спеціалістів, а також персоналізованого профілю користувача.

Об'єкт дослідження – процес організації та автоматизації взаємодії пацієнтів із медичними установами під час запису на прийом до лікаря. Об'єкт дослідження охоплює способи надання медичних послуг у цифровому середовищі, механізми обробки інформації про лікарів, така як спецалізація, досвід роботи та кількість .прийнятих пацієнтів, а також сучасні підходи до використання інформаційних технологій у сфері охорони здоров'я.

Предмет дослідження – методи та засоби розробки мобільного застосунку для вибору лікаря, запису на прийом та комунікації пацієнта з лікарем за допомогою телефонного зв'язку або месенджерів. Предмет дослідження включає проєктування

функціоналу застосунку, користувацького інтерфейсу, системи взаємодії користувачів і реалізацію онлайн-запису до лікаря.

Короткий зміст роботи – у роботі проаналізовано існуючі мобільні та вебзастосунки для організації запису на прийом до лікаря, зокрема Helsi, Doc.ua та Medics. Розглянуто сучасні підходи до розробки мобільних застосунків із використанням середовища Android Studio та хмарних сервісів Firebase.

Розроблено алгоритм функціонування застосунку та програмно реалізовано основні функціональні можливості, а саме: перегляд списку лікарів, фільтрацію за категоріями, перегляд детальної інформації про лікаря, запис на прийом, обмін повідомленнями, здійснення телефонного дзвінка, відображення місцезнаходження клініки та взаємодію з базою даних. Проведено налаштування хмарних сервісів Firebase для зберігання та обробки даних мобільного додатка. Розгортання застосунку здійснено та протестовано на мобільному пристрої Google Pixel 10 Pro.

Сферою використання застосунку є пошук і вибір лікарів для запису на прийом у медичних установах.

1 АНАЛІЗ ІСНУЮЧИХ МОБІЛЬНИХ ЗАСТОСУНКІВ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ ЗАПИСУ ПАЦІЄНТІВ НА ПРИЙОМ ДО ЛІКАРЯ

1.1 Проблематика організації запису пацієнтів на прийом до лікаря

Організація запису пацієнтів на прийом до лікаря є важливою складовою сучасної системи охорони здоров'я. Від ефективності цього процесу залежить не лише зручність отримання медичних послуг, а й загальний рівень обслуговування клієнтів і навантаження на медичний персонал. Незважаючи на активний розвиток інформаційних технологій, у багатьох медичних установах досі використовуються традиційні підходи до організації запису, які мають суттєві недоліки.

До основних традиційних способів запису на прийом належать телефонні дзвінки та особисте відвідування реєстратури медичного закладу. Такі підходи є малоефективними, бо потребують значних витрат часу як з боку пацієнтів, так і з боку працівників реєстратури. Люди змушені витрачати час на очікування в чергах або здійснювати численні телефонні звернення, які не завжди забезпечують успішний запис. Як наслідок, перевантаження реєстратур може призводити до помилок у записах, втрати даних або дублювання записів.

Ще одна проблема – обмежений доступ до актуальної інформації про лікарів. У більшості випадків пацієнти не мають можливості заздалегідь ознайомитися з кваліфікацією лікаря, його досвідом роботи, спеціалізацією або відгуками інших людей. Це ускладнює процес вибору спеціаліста та може негативно впливати на якість отриманої медичної допомоги.

Важливим недоліком традиційних підходів є відсутність гнучкості у процесі запису. Пацієнти часто не можуть швидко змінити або скасувати запис, що створює додаткові труднощі як для них, так і для медичних установ. Це, у свою чергу, призводить до неефективного використання робочого часу лікарів у разі неявки пацієнта без попередження. Крім того, у деяких випадках відсутній безпосередній зв'язок з медичними фахівцями, що знижує зручність використання таких систем.

З розвитком цифрових технологій почали з’являтися мобільні та вебзастосунки, що дозволяють автоматизувати процес запису на прийом до лікаря. Такі системи надають можливість переглядати доступні часові слоти, обирати лікаря за спеціалізацією, а також отримувати додаткову інформацію про медичні послуги. Однак, навіть сучасні рішення не позбавлені певних недоліків.

Також варто відзначити проблему інтеграції різних інформаційних систем. Не всі медичні установи використовують єдині стандарти зберігання та обробки даних, що ускладнює обмін інформацією між системами. Це може призводити до фрагментації даних та зниження ефективності цифрових рішень у сфері охорони здоров’я.

Важливим аспектом є також доступність – частина користувачів не має достатнього рівня цифрової грамотності або доступу до сучасних мобільних пристроїв. З огляду на це, під час розробки подібних систем необхідно враховувати потреби різних категорій населення та забезпечувати простоту й зручність використання.

Аналіз існуючих підходів до організації запису пацієнтів на прийом до лікаря свідчить про наявність низки проблем, пов’язаних із неефективністю традиційних методів, обмеженою функціональністю сучасних рішень та недостатнім рівнем зручності для користувачів. Це зумовлює необхідність розробки нових мобільних інформаційних систем, які враховуватимуть виявлені недоліки та забезпечуватимуть більш ефективну, зручну й доступну взаємодію між пацієнтами та медичними установами.

1.2 Особливості організації запису пацієнтів на прийом до лікаря

На відміну від виявлених проблем, сучасні системи запису повинні враховувати низку ключових особливостей функціонування. Специфіка організації запису пацієнтів на прийом до лікаря полягає у поєднанні технічних, організаційних та користувацьких аспектів. Така система повинна одночасно забезпечувати зручність для пацієнтів, ефективність для медичних установ і

надійність обробки даних. Це вимагає врахування особливостей взаємодії між різними категоріями користувачів, а також забезпечення швидкого доступу до актуальної інформації.

В першу чергу функціонування систем запису базується на обробці великого обсягу даних. До них належать відомості про лікарів, а саме: їхня спеціалізація, графіки прийому, доступні часові інтервали та інформація про пацієнтів. Забезпечення актуальності цих даних є критично важливим, оскільки будь-які неточності можуть призводити до помилок у записі та зниження якості обслуговування. Також система повинна забезпечувати швидкий доступ до інформації та її синхронізацію в режимі реального часу.

Ще важливим аспектом інтеграції з іншими інформаційними системами. У сучасних умовах медичні установи використовують різні електронні сервіси, тому важливо забезпечити можливість обміну даними між ними. Це дозволяє створити єдине інформаційне середовище, підвищити ефективність роботи та уникнути дублювання даних.

Окрему увагу слід приділити питанням безпеки та конфіденційності. Медична інформація є чутливою, тому система повинна забезпечувати захист персональних даних користувачів. Це передбачає використання сучасних методів автентифікації, шифрування даних та обмеження доступу до інформації.

Ще важливою особливістю є використання мобільних технологій. Мобільні застосунки забезпечують доступ до системи в будь-який час та з будь-якого місця, що значно підвищує зручність користування. Це особливо актуально в умовах сучасного ритму життя, коли швидкість отримання послуг відіграє ключову роль.

Системи запису пацієнтів на прийом до лікаря характеризуються поєднанням вимог до обробки даних, зручності використання, гнучкості функціоналу, інтеграції з іншими системами та забезпечення безпеки. Врахування цих особливостей є необхідною умовою для створення ефективного та сучасного програмного рішення, що відповідає потребам користувачів та вимогам медичної сфери.

1.3 Дослідження існуючих аналогів

Враховуючи виявлені проблеми та визначені особливості функціонування систем запису пацієнтів на прийом до лікаря, доцільно провести аналіз наявних рішень у цій сфері. На сьогоднішній день існує низка інформаційних сервісів, що реалізують функції онлайн-запису, зокрема Helsi, Doc.ua та Medics.

Helsi – одна з провідних медичних інформаційних систем в Україні, яка забезпечує електронний запис пацієнтів на прийом до лікаря та інтеграцію з електронною системою охорони здоров'я. Сервіс призначений для спрощення взаємодії між пацієнтами та медичними установами, а також для автоматизації процесів обліку медичних послуг. Користувачі можуть обирати лікаря, переглядати графік його прийому, записуватися на консультацію та отримувати доступ до електронних медичних документів, зокрема направлень і записів. Приклад екранних форм вебзастосунку Helsi наведено на рисунку 1.1.

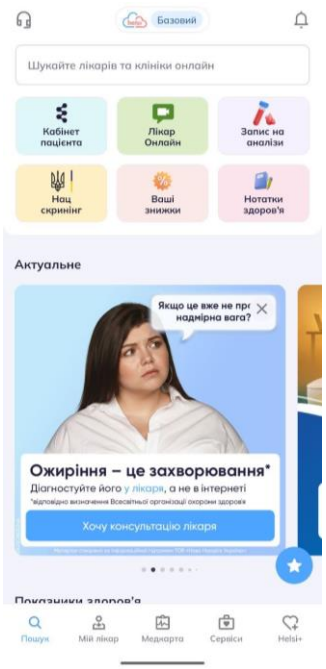


Рис. 1.1 Приклад екранних форм вебзастосунку Helsi

Переваги:

– інтеграція з електронною системою охорони здоров'я;

– доступ до електронних медичних документів;

– широкий функціонал для взаємодії з медичними установами;

– можливість онлайн-запису до лікаря.

Недоліки:

– орієнтація переважно на державні медичні установи;

– обмежений вибір лікарів у приватному секторі;

– складний інтерфейс для окремих категорій користувачів;

– необхідність авторизації та наявності електронних даних для повноцінного використання.

Сервіс Doc.ua є одним із найбільш поширених в Україні та надає можливість пошуку лікаря за спеціалізацією, місцем розташування, рейтингом і відгуками пацієнтів. Система орієнтована на спрощення вибору спеціаліста та процесу запису на прийом. Користувачі можуть переглядати інформацію про лікаря, зокрема досвід роботи, освіту та перелік послуг, а також здійснювати онлайн-запис на прийом. Приклад екранних форм вебзастосунку Doc.ua наведено на рисунку 1.2.

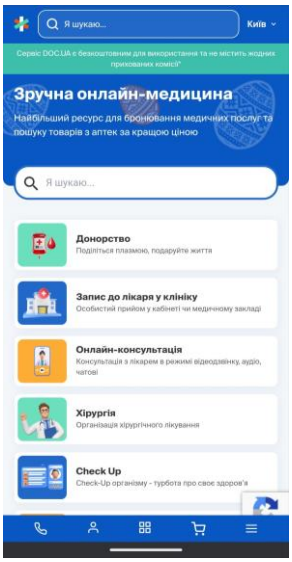


Рис. 1.2 Приклад екранних форм вебзастосунку Doc.ua.

Переваги:

– широкий вибір лікарів та медичних закладів;

– орієнтація на приватні медичні установи;

– наявність детальної інформації про лікаря;

– можливість пошуку за різними критеріями (спеціалізація, місце розташування, рейтинг);

– зручний онлайн-запис на прийом.

Недоліки:

– переважно платний характер медичних послуг;

– обмежена доступність для окремих категорій пацієнтів;

– можливе перевантаження інтерфейсу великою кількістю інформації;

– відсутність інтеграції з державною електронною системою охорони здоров’я.

Сервіс Medics орієнтований на спрощення процесу запису до лікаря та характеризується лаконічним і зрозумілим інтерфейсом. Система спрямована на швидке виконання основних дій без перевантаження користувача зайвими функціями. Застосунок надає базові функціональні можливості, зокрема перегляд списку лікарів, вибір спеціаліста та запис на прийом у зручний час. Приклад екранних форм вебзастосунку Medics наведено на рисунку 1.3.

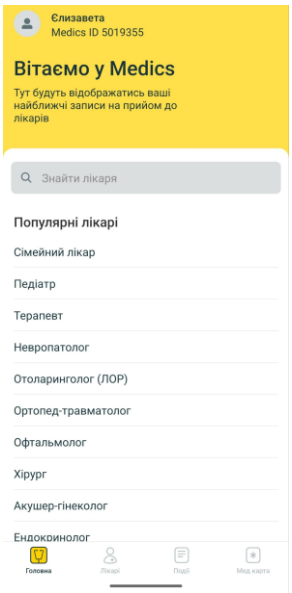


Рис. 1.3 Приклад екранних форм вебзастосунку Medics

Переваги:

– простий і зрозумілий інтерфейс;

– швидкий доступ до основних функцій;

– відсутність перевантаження додатковими можливостями;

– зручність використання для користувачів із різним рівнем цифрової грамотності.

Недоліки:

– обмежена функціональність порівняно з іншими сервісами;

– відсутність детальної інформації про лікаря (досвід, кваліфікація, відгуки);

– недостатньо розвинені засоби взаємодії з лікарем;

– обмежені можливості вибору серед великої кількості спеціалістів.

Аналіз розглянутих застосунків показує, що вони забезпечують необхідний базовий функціонал для запису на прийом до лікаря та є зручними у використанні. Також вони мають певні обмеження щодо рівня персоналізації, інформативності та інтеграції. Це зумовлює доцільність розробки мобільної інформаційної системи, яка враховуватиме виявлені особливості існуючих рішень та забезпечуватиме більш ефективну й зручну взаємодію між пацієнтами та медичними установами.

Таблиця 1.1

Зведена таблиця аналізу існуючих застосунків для вивчення методів шифрування та їх порівняння

Показник	Helsi	Doc.ua	Medics
Тип медичних установ	Переважно державні	Переважно приватні	Приватні та окремі клініки
Функціональність	Широкий функціонал (запис, медичні дані, направлення)	Розширений функціонал (пошук, запис, інформація)	Базовий функціонал (запис, список лікарів)
Інформація про лікаря	Обмежена, залежить від системи	Детальна (досвід, освіта, відгуки)	Обмежена
Зручність інтерфейсу	Середня (може бути складним)	Середня (інколи перевантажений)	Висока (простий і зрозумілий)
Комунікація з лікарем	Обмежена	Частково доступна	Відсутня або обмежена

Продовження таблиці 1.1

Показник	Helsi	Doc.ua	Medics
Інтеграція з eHealth	Є	Відсутня	Відсутня
Рівень доступності	Високий (державний сектор)	Середній (переважно платні послуги)	Середній
Персоналізація	Обмежена	Частково реалізована	Обмежена
Платформи	Web, Android, iOS	Web	Web / мобільна версія
Інтерфейс	Функціональний, може бути складним	Інтуїтивний, але інформаційно насичений	Простий, інтуїтивний
Доступ	Доступ після реєстрації та авторизації	Частково відкритий, запис після реєстрації	Доступ після реєстрації

1.4 Засоби реалізації

На сучасному етапі розвитку інформаційних технологій особливу роль відіграє правильний вибір засобів реалізації програмного забезпечення. Від обраних інструментів і технологій залежить не лише швидкість розробки застосунку, а й надійність, продуктивність, зручність використання та можливість подальшого масштабування.

Розробка мобільних застосунків у сфері охорони здоров’я висуває підвищені вимоги до якості програмного продукту, зокрема щодо стабільності роботи, швидкого доступу до даних, захисту інформації користувачів та зручності інтерфейсу. Тому вибір технологічного стеку повинен враховувати як технічні характеристики інструментів, так і специфіку предметної області.

У рамках даної роботи для розробки мобільного додатка для електронного запису до лікаря було використано сучасні інструменти та технології, такі як Android Studio – середовище розробки, мову програмування Kotlin для реалізації логіки застосунку, а також хмарну платформу Firebase для зберігання та обробки даних.

Застосування зазначених технологій дозволило забезпечити ефективну реалізацію функціональних можливостей системи, підвищити швидкість обробки даних та створити зручний і зрозумілий інтерфейс користувача.

1.4.1 Android Studio

Android Studio – офіційне інтегроване середовище розробки (IDE) для створення мобільних застосунків на платформі Android, розроблене компанією Google. Воно базується на IntelliJ IDEA та надає широкий набір інструментів для повного циклу розробки програмного забезпечення – від створення інтерфейсу до тестування та налагодження застосунку.

У процесі розробки мобільного застосунку Android Studio використовувалося як основне середовище програмування. За допомогою нього здійснювалися створення структури проєкту, розробка інтерфейсу користувача за допомогою XML-розмітки, написання програмного коду, а також тестування та налагодження функціональних можливостей системи.

Однією з ключових переваг Android Studio – наявність вбудованого візуального редактора, що дозволяє проєктувати екрани застосунку у зручному графічному режимі. Це значно спрощує процес створення користувацького інтерфейсу та дозволяє швидко вносити зміни.

Android Studio підтримує пряме встановлення та запуск застосунку на мобільному пристрої як через USB-з’єднання, так і за допомогою бездротового підключення (Wi-Fi), що значно спрощує процес тестування та налагодження.

Середовище розробки підтримує інтеграцію з системами контролю версій, зокрема Git, що дозволяє ефективно керувати змінами в проєкті. Крім того, середовище має інструменти для аналізу продуктивності, пошуку помилок та оптимізації коду.

Отже, використання Android Studio забезпечує зручність розробки, підвищує якість програмного продукту та дозволяє ефективно організувати процес створення мобільного застосунку.

1.4.2 Kotlin

Kotlin – сучасна мова програмування, яка офіційно підтримується компанією Google як основна мова для розробки Android-застосунків. Вона поєднує у собі переваги об’єктно-орієнтованого та функціонального програмування, що робить її зручною для створення складних програмних систем.

У даному проєкті Kotlin використовувався як основна мова програмування для реалізації логіки мобільного застосунку. За допомогою цієї мови було реалізовано обробку даних, взаємодію з базою даних, обробку подій користувача, а також навігацію між екранами застосунку.

Однією з важливих переваг мови програмування є її лаконічний синтаксис, що дозволяє значно зменшити обсяг коду порівняно з іншими мовами програмування. Це підвищує читабельність програмного коду та спрощує його підтримку.

Крім того, Kotlin має вбудовані механізми забезпечення безпеки, зокрема захист від помилок, пов’язаних із null-значеннями (NullPointerException), що сприяє зменшенню кількості збоїв під час виконання програми та підвищенню її стабільності.

Мова також підтримує асинхронне програмування за допомогою корутин, що є важливим для роботи з мережею та обробки даних у реальному часі. Це дозволяє уникнути блокування основного потоку виконання та забезпечити плавну роботу застосунку.

Використання Kotlin сприяє підвищенню якості програмного коду, спрощує процес розробки та забезпечує високу продуктивність мобільного застосунку.

1.4.3 Firebase

У проєкті використовувалася база даних Firebase (Firestore / Realtime Database), яка забезпечує зберігання та синхронізацію даних у режимі реального часу. Firebase – це хмарна платформа, розроблена компанією Google, що надає широкий набір інструментів для створення та підтримки мобільних застосунків.

Платформа дозволяє реалізувати серверну частину без необхідності розробки власного бекенду, що значно спрощує процес створення застосунку та

скорочує час розробки.

У рамках даного проєкту Firebase використовувався для організації зберігання та обробки даних, а також для забезпечення взаємодії між користувачем і системою. Зокрема, база даних застосовувалася для зберігання інформації про лікарів, користувачів та записів на прийом.

Однією з ключових переваг бази даних є підтримка роботи з даними в режимі реального часу. Це означає, що зміни в базі даних миттєво відображаються у застосунку, що забезпечує актуальність інформації та покращує взаємодію користувача з системою.

Firebase також надає можливості для підтримки масштабування, що є важливим у разі збільшення кількості користувачів.

Використання Firebase дозволяє значно спростити процес розробки, зменшити витрати часу на створення серверної частини та забезпечити стабільну роботу застосунку.

1.4.4 XML

Для створення інтерфейсу користувача у мобільному застосунку використовувалася мова розмітки XML. Вона дозволяє описувати структуру екранів, розташування елементів та їхній зовнішній вигляд.

У проєкті XML застосовувався для реалізації основних екранів застосунку, а також для створення окремих елементів списків (viewholder). Такий підхід дозволяє чітко розділити логіку застосунку та його візуальне представлення.

Використання XML-розмітки забезпечує зручність підтримки інтерфейсу та спрощує внесення змін у структуру застосунку.

1.4.5 RecyclerView

Для відображення списків даних у застосунку використовувався компонент RecyclerView. Він дозволяє ефективно відображати великі обсяги інформації, зокрема списки лікарів та категорій.

RecyclerView забезпечує повторне використання елементів інтерфейсу (ViewHolder), що дозволяє зменшити навантаження на систему та підвищити продуктивність застосунку. У проєкті реалізовано власні адаптери та шаблони

елементів списку для відображення інформації про лікарів.

Використання даного компонента дозволяє створити динамічний та зручний інтерфейс користувача.

1.4.6 Gradle

Для керування залежностями та автоматизації процесу збірки застосунку використовувався інструмент Gradle. Він дозволяє підключати необхідні бібліотеки, налаштовувати конфігурацію проєкту та керувати процесом компіляції.

Завдяки використанню Gradle забезпечується гнучкість налаштувань та спрощується інтеграція сторонніх бібліотек.

1.4.7 Figma

На етапі проєктування мобільного застосунку для розробки інтерфейсу користувача було використано інструмент Figma. Даний сервіс є середовищем для створення прототипів та дизайну інтерфейсів, що дозволяє візуалізувати структуру застосунку ще до початку програмної реалізації. Figma широко використовується у процесі розробки програмного забезпечення, оскільки забезпечує можливість створення інтерактивних прототипів, спільної роботи над проєктом та швидкого внесення змін до дизайну.

За допомогою Figma було розроблено макети основних екранів застосунку, зокрема головного екрану, списку лікарів, сторінки детальної інформації про лікаря, а також екранів запису на прийом. Це дозволило визначити розташування елементів інтерфейсу, логіку навігації між екранами та забезпечити зручність користування застосунком.

Дуже багато часу приділялося створенню інтуїтивно зрозумілого та мінімалістичного інтерфейсу, який забезпечує швидкий доступ до основних функцій. Використання Figma дозволило заздалегідь оцінити зручність взаємодії користувача із системою та внести необхідні зміни до структури інтерфейсу ще на етапі проєктування.

Використання Figma допомогло розробити якісний та інтуїтивно зрозумілий інтерфейс.

2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ДОДАТКА ДЛЯ ЕЛЕКТРОННОГО ЗАПИСУ ДО ЛІКАРЯ

Система реалізована у вигляді мобільного застосунку для платформи Android із використанням хмарного середовища для зберігання даних та обробки запитів користувачів. Клієнтська частина застосунку розроблена за допомогою мови програмування Kotlin у середовищі Android Studio, тоді як серверна логіка реалізована на базі платформи Firebase. Мобільний застосунок забезпечує взаємодію користувача з системою, зокрема перегляд інформації про лікарів, вибір спеціаліста, запис на прийом та отримання актуальних даних у режимі реального часу.

2.1 Проектування загальної архітектури системи

Система реалізована за клієнт-серверною схемою з використанням хмарних сервісів Firebase (автентифікації та бази даних) на боці сервера. Клієнтська частина представлена мобільним застосунком, розробленим для платформи Android із використанням мови програмування Kotlin, тоді як серверні функції забезпечуються сервісами Firebase.

Додаток має централізовану архітектуру, де основна логіка обробки даних і їх зберігання реалізується за допомогою хмарних сервісів, а мобільний застосунок виконує роль клієнта, який взаємодіє з користувачем. Для організації структури коду застосовано поділ на окремі компоненти відповідно до їх функціонального призначення.

Систему можна розбити на такі функціональні блоки:

– модуль навігації: перегляд головного меню, сортування лікарів по категоріям

– модуль перегляду лікарів: відповідає за отримання та відображення списку лікарів із бази даних, включаючи основну інформацію, таку як ім’я, спеціалізація, рейтинг та досвід роботи;

– модуль запису на прийом: забезпечує можливість вибору лікаря;

– модуль профілю користувача: дозволяє переглядати та редагувати особисті дані;

– модуль пошуку та фільтрації: реалізує пошук лікарів за спеціалізацією та іншими параметрами, що підвищує зручність використання застосунку;

У клієнтському застосунку обробка подій здійснюється за допомогою механізмів Android, зокрема обробників подій і життєвого циклу Activity. Взаємодія з Firebase забезпечує отримання та оновлення даних у режимі реального часу.ф

Компоненти застосунку реалізовані таким чином, щоб логіка прикладного рівня була відокремлена від інтерфейсу користувача: Kotlin відповідає за обробку даних і бізнес-логіку, XML – за структуру інтерфейсу, що дозволяє підвищити гнучкість і зручність підтримки застосунку.

2.2 Проєктування сценаріїв використання системи

Головним користувачем системи є пацієнт. Для ілюстрації сценаріїв роботи мобільного застосунку застосовано діаграму варіантів використання (use-case діаграму), наведену на рисунку 2.1, на якій користувач пов’язаний із такими функціональними можливостями системи:

– пошук лікарів – для того, щоб користувач мав можливість переглядати список доступних спеціалістів та здійснювати пошук за спеціалізацією;

– перегляд детальної інформації про лікарів –дає можливість користувачу отримати інформацію про лікаря, а саме: спеціалізацію, рейтинг, досвід роботи та інші відомості;

– запис на прийом реалізовано для забезпечення можливості вибору лікаря та створення запису на консультацію у зручний для користувача час;

– перегляд та редагування профілю зумовлені необхідністю керування персональними даними користувача, зокрема ім’ям, електронною поштою та іншою контактною інформацією;

– зв’язок із лікарем забезпечує можливість взаємодії користувача з медичним спеціалістом для уточнення інформації або отримання консультації;

– перегляд місцезнаходження клініки дозволяє користувачу отримати інформацію про розташування медичного закладу для зручного планування візиту;

– надсилання повідомлень реалізоване для забезпечення обміну інформацією між користувачем і медичним персоналом.

Усі перелічені функціональні можливості забезпечують зручність використання мобільного застосунку та підвищують ефективність взаємодії користувача з медичними послугами.

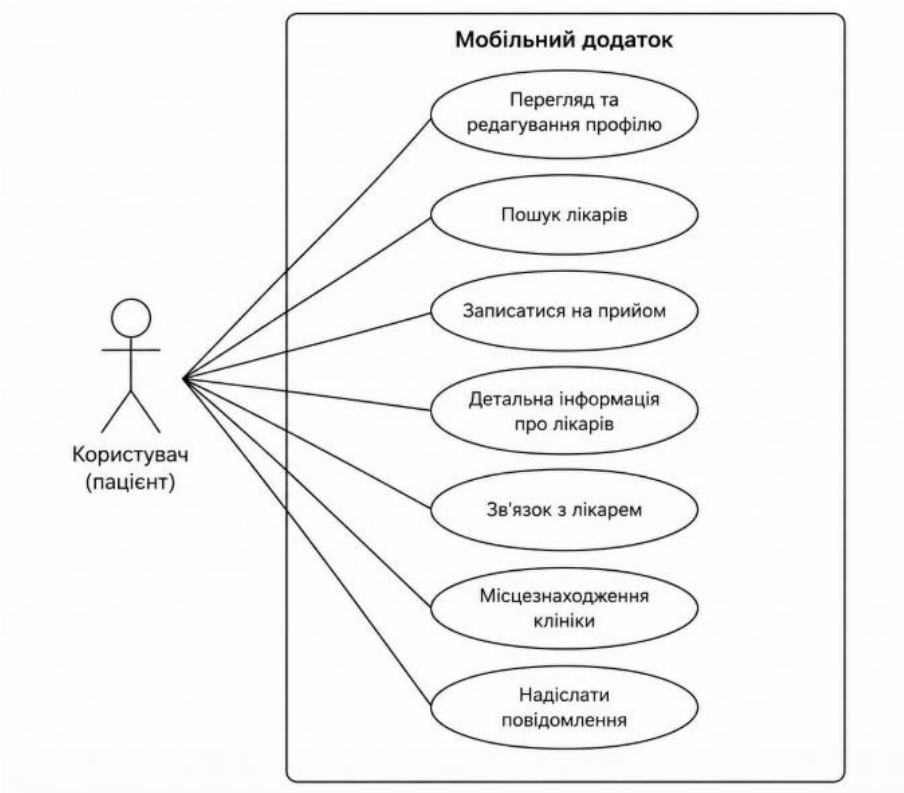


Рис. 2.1 Діаграма випадків використання

2.3 Функціональні та не функціональні вимоги

Процес проєктування будь-якого мобільного додатка розпочинається з визначення основних вимог до системи. Саме функціональні та нефункціональні вимоги дозволяють окреслити можливості застосунку, особливості його роботи та ключові характеристики якості. Для більш наочного представлення вимог нижче

наведено таблицю функціональних вимог (таблиця 2.1) та таблицю нефункціональних вимог (таблиця 2.2) для мобільного додатка для електронного запису до лікаря.

Таблиця 2.1

Зведена таблиця функціональних вимог для мобільного додатка для електронного запису до лікаря

№	Вимога	Опис
1	Перегляд списку лікарів	Система повинна надавати користувачу можливість переглядати список доступних лікарів із короткою інформацією про них.
2	Пошук лікарів	Користувач повинен мати можливість здійснювати пошук лікарів за спеціалізацією або іншими параметрами.
3	Перегляд детальної інформації	Система повинна відображати детальну інформацію про лікаря, зокрема спеціалізацію, рейтинг та досвід роботи.
4	Запис на прийом	Користувач повинен мати можливість записатися на прийом до обраного лікаря.
5	Робота з профілем	Система повинна дозволяти перегляд та редагування персональної інформації користувача.
6	Збереження даних	Дані про лікарів та записи на прийом повинні зберігатися у хмарній базі даних Firebase Firestore.

Таблиця 2.2

Зведена таблиця нефункціональних вимог для мобільного застосунку запису до лікаря

№	Вимога	Опис
1	Інтуїтивно зрозумілий інтерфейс	Інтерфейс застосунку має бути простим, сучасним і зручним для користувача.
2	Швидкодія та відгук	Застосунок повинен швидко реагувати на дії користувача та забезпечувати оперативне завантаження даних.
3	Масштабованість	Використання Firebase дозволяє забезпечити автоматичне масштабування системи без необхідності налаштування власного сервера.
4	Доступність	Мобільний застосунок повинен коректно працювати на більшості Android-пристроїв.

Наведені у таблиці 2.1 функціональні вимоги визначають основні сценарії взаємодії користувача з мобільним застосунком.

Однією з ключових функціональних можливостей системи є перегляд списку лікарів і пошук необхідного спеціаліста за певними параметрами, що значно спрощує процес вибору лікаря.

Також не менш важливо, що користувач має можливість переглядати детальну інформацію про лікаря, зокрема спеціалізацію, рейтинг та досвід роботи. Це дозволяє отримати більш повне уявлення про медичного фахівця перед записом на прийом.

Наступною важливою функціональною можливістю є запис на прийом, який забезпечує швидке та зручне бронювання консультації без необхідності особистого звернення до медичного закладу.

Окрему увагу приділено модулю профілю користувача, який дозволяє переглядати та редагувати персональні дані. Збереження інформації у хмарній базі Firebase Firestore забезпечує швидкий доступ до даних та їх синхронізацію у режимі реального часу.

Нефункціональні вимоги (таблиця 2.2) визначають основні стандарти якості мобільного застосунку.

Інтуїтивно зрозумілий інтерфейс підвищує зручність використання застосунку та забезпечує комфортну взаємодію користувача із системою. Висока швидкодія дозволяє оперативно отримувати інформацію та мінімізує затримки під час роботи із застосунком.

Важливою складовою є забезпечення безпеки персональних даних користувачів, оскільки система працює з конфіденційною інформацією. Використання хмарної платформи Firebase також забезпечує масштабованість системи та дозволяє підтримувати стабільну роботу застосунку при збільшенні кількості користувачів.

У підсумку, наведені вимоги формують зручний, надійний та ефективний мобільний застосунок для запису до лікаря.

2.4 Проектування структури бази даних

Система зберігання даних мобільного застосунку реалізована за допомогою хмарної бази даних Firebase Firestore, яка належить до класу документно-орієнтованих NoSQL-систем. Такий підхід дозволяє забезпечити швидке збереження, отримання та синхронізацію інформації між клієнтською частиною застосунку та хмарним середовищем. Для ілюстрації взаємодії мобільного додатка з базою даних наведена діаграма класів на рисунку 2.2.

Firebase Firestore організовує дані у вигляді колекцій та документів. Кожен документ містить набір полів із відповідними значеннями, що дозволяє гнучко структурувати інформацію та легко масштабувати систему. У межах мобільного застосунку база даних використовується для зберігання інформації про лікарів, записів на прийом, а також інших даних, необхідних для функціонування системи.

Взаємодія застосунку з Firestore здійснюється через Firebase SDK для Android, що дозволяє виконувати операції читання, запису, оновлення та видалення даних у режимі реального часу. Завдяки цьому користувач отримує актуальну інформацію без необхідності повторного завантаження сторінок або оновлення застосунку.

Однією з переваг використання Firebase Firestore є автоматична синхронізація даних між пристроями та підтримка роботи у хмарному середовищі без потреби у створенні власного серверного обладнання. Також документно-орієнтована структура бази даних дозволяє легко розширювати функціональні можливості системи шляхом додавання нових полів або колекцій без зміни вже існуючої структури.

Такий підхід забезпечує надійність зберігання інформації, високу швидкодію та зручність підтримки програмного коду, що є важливим для сучасних мобільних застосунків у сфері медичних послуг.

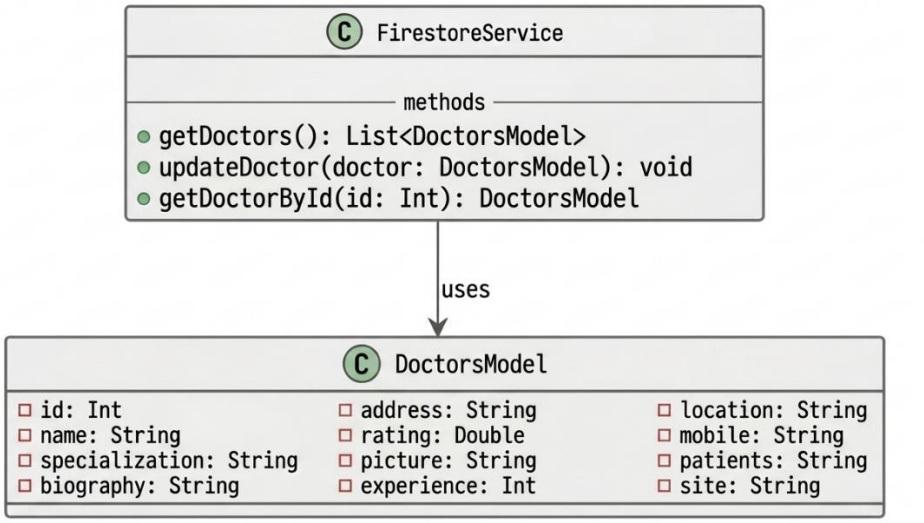


Рис. 2.2 Діаграма класів

Діаграма класів, наведена на рисунку 2.2, демонструє основні компоненти системи та взаємозв'язки між ними. Центральним елементом архітектури є сервіс FirestoreService, який забезпечує взаємодію мобільного застосунку з хмарною базою даних Firebase Firestore. Саме через цей сервіс здійснюються операції отримання, оновлення та обробки інформації про лікарів.

Для представлення даних використовується модель DoctorsModel, яка містить основні характеристики лікаря, зокрема ім'я, спеціалізацію, біографію, адресу, рейтинг, контактні дані та стаж роботи. Така структура дозволяє зручно організувати дані та забезпечити швидкий доступ до необхідної інформації у межах мобільного застосунку.

Методи getDoctors(), updateDoctor() та getDoctorById() реалізують основний функціонал взаємодії з базою даних. Зокрема, вони забезпечують отримання списку лікарів, оновлення інформації про конкретного спеціаліста та пошук лікаря за ідентифікатором. Використання окремого сервісного класу для роботи з Firebase дозволяє централізувати логіку обробки даних і спрощує підтримку програмного коду.

Запропонована архітектура забезпечує гнучкість, масштабованість та зручність подальшого розширення функціональних можливостей мобільного застосунку. Також використання Firebase Firestore у поєднанні з Android SDK

дозволяє забезпечити стабільну синхронізацію даних у режимі реального часу та високу швидкодію системи.

2.5 Проектування інтерфейсу користувача

На ранніх етапах розробки мобільного застосунку для проєктування користувацького інтерфейсу було використане середовище Figma. Даний інструмент дозволив створити попередні макети основних екранів застосунку, продумати структуру інтерфейсу та організувати розташування основних елементів керування ще до початку програмної реалізації. Дуже важливо було розробити гармонійний дизайн, щоб екрани мали спільний стиль оформлення та єдину кольорову гаму.

Також для розробки користувацького інтерфейсу застосовувалися інструменти Android Studio, XML-макети для побудови екранів та мова програмування Kotlin для реалізації логіки взаємодії користувача із системою.

Основні сценарії взаємодії користувача із застосунком реалізовані через головний екран, а саме: список лікарів, сторінку профілю та детальну інформацію про лікаря. Головний екран містить привітальний блок, пошуковий рядок, категорії медичних спеціалізацій та список популярних лікарів. Для реалізації навігації між розділами використано нижнє меню навігації (Bottom Navigation View), яке забезпечує швидкий доступ до основних функцій застосунку.

Екран списку лікарів відображає картки спеціалістів із короткою інформацією про лікаря, зокрема ім’ям, спеціалізацією, рейтингом та стажем роботи. Для створення списків використано компонент RecyclerView, що дозволяє ефективно відображати велику кількість елементів та забезпечує плавну прокрутку інтерфейсу. Кожна картка лікаря містить кнопку швидкого запису на прийом. Для побудови інтерфейсу застосовувалися компоненти Material Design, які забезпечують сучасний зовнішній вигляд застосунку та зручність взаємодії користувача із системою. У дизайні використано світлу кольорову гаму з фіолетовими акцентами, округленими елементами інтерфейсу та мінімалістичним

стилем оформлення. Іконки медичних спеціалізацій були інтегровані для покращення візуального сприйняття та швидкої навігації між категоріями лікарів.

Інтерфейс мобільного застосунку адаптовано під різні розміри екранів мобільних пристроїв. Для цього використовувалися ConstraintLayout та адаптивні XML-макети, що дозволяють коректно масштабувати елементи інтерфейсу на смартфонах із різними характеристиками дисплея. Ще важливо уточнити, що при розробці даного мобільного додатка були враховані потреби всіх вікових груп населення для максимального комфорту при використанні.

Важливою вимогою до UI/UX було забезпечення швидкого відгуку системи та простоти використання. Завдяки інтеграції Firebase Firestore оновлення інформації про лікарів і взаємодія з даними здійснюються у режимі реального часу, що забезпечує актуальність інформації без необхідності ручного оновлення екрана. Варто зазначити, що інтерфейс побудований таким чином, щоб користувач міг швидко виконати основні дії – знайти лікаря, переглянути інформацію про нього та перейти до запису на прийом.

На рисунках нижче представлено основні екрани мобільного додатка, розроблені в середовищі Figma та реалізовані під час розробки інтерфейсу користувача.

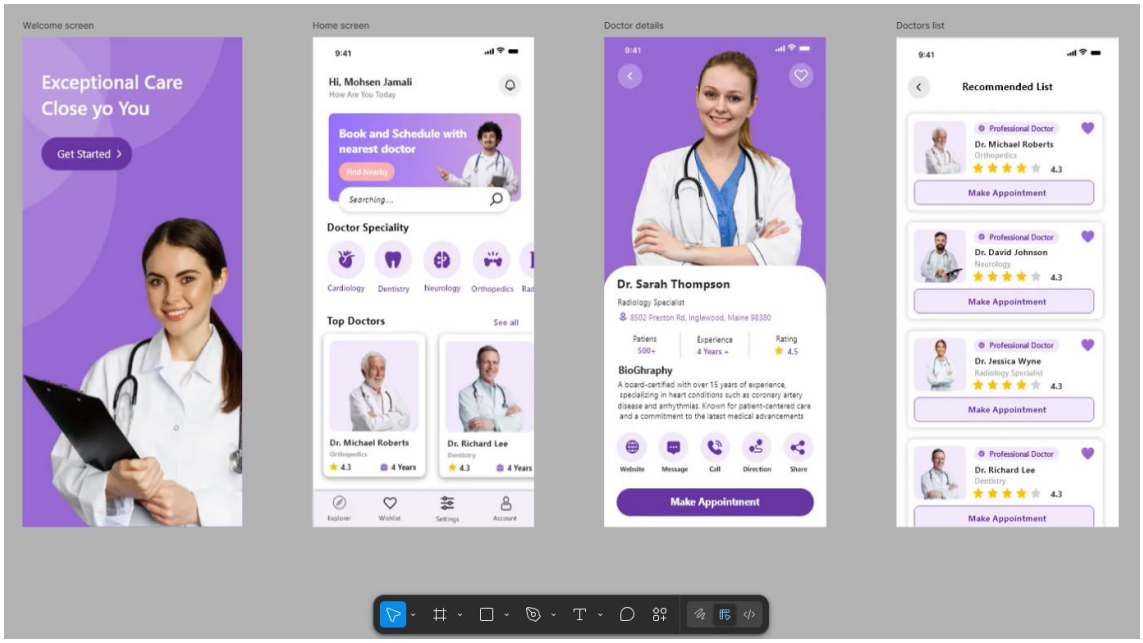


Рис. 2.3 Основні екрани мобільного додатка

3 РЕАЛІЗАЦІЯ МОДУЛЯ ВЗАЄМОДІЇ З КОРИСТУВАЧЕМ ТА УПРАВЛІННЯ ПРИЙОМАМИ У МЕДИЧНОМУ ДОДАТКУ

3.1 Опис реалізації програмного забезпечення

Мобільний додаток реалізовано у вигляді клієнтського застосунку з інтеграцією хмарних Firebase-сервісів. Користувацька частина застосунку відповідає за відображення інтерфейсу, взаємодію користувача із системою та обробку введених даних, тоді як хмарна база даних забезпечує зберігання інформації, синхронізацію даних і підтримку роботи в режимі реального часу.

Програмний модуль складається з кількох основних екранів мобільного застосунку, зокрема головного екрана, сторінки списку лікарів, екрана детальної інформації про лікаря та сторінки профілю користувача.

Архітектура застосунку побудована за багаторівневим принципом. Рівень представлення реалізовано за допомогою XML-макетів Android Studio, які відповідають за відображення елементів інтерфейсу користувача. До них належать текстові поля, кнопки, картки лікарів, списки спеціалізацій та навігаційні елементи. Для створення адаптивного інтерфейсу використовувалися ConstraintLayout та компоненти Material Design, що забезпечують коректне відображення застосунку на мобільних пристроях із різними характеристиками дисплеїв.

Логіка реалізована мовою програмування Kotlin. Саме вона відповідає за обробку дій користувача, навігацію між екранами, завантаження інформації про лікарів, роботу пошукових елементів та взаємодію із базою даних Firebase Firestore. Для ефективного відображення списків лікарів використовувався компонент RecyclerView, який забезпечує плавну прокрутку інтерфейсу та оптимізоване відображення великої кількості елементів.

Рівень даних реалізовано за допомогою Firebase Firestore, що забезпечує хмарне зберігання інформації про лікарів, їхні спеціалізації, рейтинги та інші дані застосунку. Інтеграція Firebase дозволяє синхронізувати інформацію у режимі реального часу без необхідності ручного оновлення сторінок. Крім того,

використання хмарних технологій підвищує надійність збереження даних та забезпечує можливість масштабування системи.

Обрана архітектура мобільного застосунку забезпечує гнучкість розробки, адаптивність інтерфейсу та високу швидкість взаємодії користувача із системою. Перевагами такого підходу є сучасний дизайн, можливість роботи з хмарними сервісами, швидке оновлення інформації та зручна навігація між екранами застосунку. Серед можливих недоліків можна виділити залежність від підключення до мережі Інтернет під час отримання даних із Firebase. Незважаючи на це, обрана архітектура є ефективною для медичного мобільного застосунку, оскільки дозволяє користувачам швидко знаходити необхідну інформацію про лікарів та взаємодіяти із системою у зручному форматі.

3.2 Програмна реалізація мобільного застосунку

У даному підрозділі розглянуто програмну реалізацію основних функціональних можливостей мобільного застосунку для пошуку лікарів та запису на прийом. Основна логіка роботи системи реалізована мовою програмування Kotlin у середовищі Android Studio. Для створення інтерфейсу користувача використовувалися XML-макети та компоненти Material Design, а для зберігання та синхронізації даних – хмарна база даних Firebase Firestore.

3.2.1 Реалізація головного екрана

Головний екран застосунку реалізований як центральний елемент взаємодії користувача із системою. На екрані розташовано привітальний блок, пошуковий рядок, категорії медичних спеціалізацій та список популярних лікарів. Для реалізації списків використовувався компонент RecyclerView, який забезпечує ефективне відображення великої кількості елементів та плавну прокрутку інтерфейсу.

Інформація про лікарів завантажується із Firebase Firestore у режимі реального часу. Кожна картка лікаря містить ім’я спеціаліста, спеціалізацію,

рейтинг та стаж роботи. Для швидкої навігації між розділами застосунку було використано нижнє меню навігації Bottom Navigation View.

3.2.2 Реалізація екрана інформації про лікаря

Екран детальної інформації про лікаря містить фотографію спеціаліста, його рейтинг, спеціалізацію, адресу медичного закладу, стаж роботи та короткий опис професійної діяльності. Також реалізована кнопка швидкого запису на прийом.

Передача інформації між екранами здійснюється за допомогою Intent, що дозволяє відкривати сторінку конкретного лікаря після вибору картки зі списку. Дані про лікаря динамічно підтягуються з Firebase Firestore, що забезпечує актуальність інформації без необхідності оновлювати застосунок.

3.2.3 Реалізація списку лікарів

Сторінка списку лікарів реалізована у вигляді вертикального списку карток спеціалістів. Для кожного елемента відображаються фотографія лікаря, його спеціалізація, рейтинг та кнопка запису на прийом. Для створення даного інтерфейсу використовувався RecyclerView Adapter, який відповідає за зв’язок між даними та елементами інтерфейсу.

Користувач може швидко переглядати список спеціалістів та переходити до сторінки детальної інформації про лікаря. Для покращення UX/UI у дизайні використано округлені елементи інтерфейсу, світлу кольорову гаму та фіолетові акценти.

3.2.4 Реалізація запису до лікаря

У програмному модулі реалізовано функціонал онлайн-запису до лікаря, який забезпечує взаємодію користувача з медичним сервісом через спеціальну форму введення даних. Даний функціонал створено у вигляді окремої Activity (AppointmentActivity), яка відкривається після натискання кнопки Make Appointment на сторінці вибраного лікаря.

Інтерфейс запису реалізовано за допомогою XML-розмітки Android та містить поля для введення імені користувача, номера телефону, електронної пошти, вибору дати та часу прийому, вибору лікаря зі списку, а також текстове поле для короткого опису проблеми або скарг пацієнта. Для прокручування контенту

використовується компонент ScrollView, що забезпечує коректне відображення форми на різних розмірах екранів мобільних пристроїв.

Вибір дати реалізовано за допомогою діалогового вікна DatePickerDialog, яке дозволяє користувачу обрати необхідний день прийому через інтерактивний календар. Аналогічно, вибір часу здійснюється за допомогою TimePickerDialog, що забезпечує зручний вибір години та хвилин прийому. Після вибору значення автоматично відображаються у відповідних текстових полях форми.

Для вибору лікаря використовується компонент Spinner, який містить список доступних спеціалістів. Дані для списку формуються програмно через адаптер ArrayAdapter, що забезпечує зв’язок між масивом даних та елементом інтерфейсу.

У програмному модулі також реалізовано механізм валідації введених даних. Перед підтвердженням запису система перевіряє заповнення обов’язкових полів: імені, номера телефону, електронної пошти, дати, часу та опису проблеми. Для перевірки коректності email-адреси використовується клас Patterns.EMAIL_ADDRESS. У разі помилки користувачу відображається відповідне повідомлення без виконання подальших дій.

Після успішного проходження перевірки формується текст підтвердження запису, який містить інформацію про обраного лікаря, дату та час прийому, а також короткий опис проблеми пацієнта. Дане повідомлення відображається на екрані додатка та додатково надсилається на електронну пошту користувача.

Надсилання повідомлення реалізовано за допомогою механізму Android Intent (Intent.ACTION_SENDTO) із використанням URI-схеми mailto:. Після формування листа система автоматично відкриває встановлений поштовий клієнт користувача із заповненими полями теми листа та тексту повідомлення. Це дозволяє забезпечити підтвердження запису та зручне збереження інформації про прийом.

Реалізований функціонал підвищує зручність взаємодії користувача з додатком, автоматизує процес запису до лікаря та забезпечує базову перевірку коректності введених даних.

3.2.5 Реалізація взаємодії з Firebase Firestore

Firebase Firestore використовується для зберігання інформації про лікарів, медичні спеціалізації та інші дані застосунку. Взаємодія із базою даних реалізована через Firebase SDK для Android, що дозволяє виконувати операції читання та отримання даних у режимі реального часу.

Використання Firebase забезпечує автоматичну синхронізацію інформації між мобільним застосунком та хмарним середовищем. Завдяки цьому користувач отримує актуальні дані про лікарів без необхідності ручного оновлення сторінок. Крім того, хмарна архітектура спрощує масштабування системи та підвищує надійність зберігання інформації.

3.3 Діаграма класів

У основі реалізації модуля взаємодії з користувачем та відображення даних лежить модульна архітектура, побудована на базі шаблону проєктування MVVM (Model-View-ViewModel) та компонентів Jetpack. Це дозволяє розділити бізнес-логіку програми, представлення даних та безпосередній графічний інтерфейс користувача, забезпечуючи високий рівень гнучкості, тестованості та масштабованості системи.

Центральним архітектурним компонентом, що пов’язує логіку збереження даних із графічним інтерфейсом, є клас MainViewModel. Він виступає в ролі посередника, акумулюючи поточний стан екрану у вигляді реактивних контейнерів даних LiveData. Зокрема, властивості category та doctors містять динамічні списки об’єктів CategoryModel та DoctorsModel відповідно. Через методи loadCategory() та loadDoctors() забезпечується асинхронне завантаження інформації (наприклад, з віддаленої бази даних чи репозиторію). Використання патерну Спостерігач (Observer) дозволяє автоматично оновлювати графічні елементи в момент зміни стану даних.

Головним координатором графічного інтерфейсу виступає клас MainActivity, який наслідується від базового компонента BaseActivity. Він ініціалізує первинну розмітку, налаштовує віджети відображення прогресу (ProgressBar) та організує навігацію. За допомогою методів initCategory() та initTopDoctors() здійснюється підписка на відповідні потоки даних від MainViewModel. При отриманні актуальних списків MainActivity динамічно конфігурує компоненти RecyclerView, передаючи дані в спеціалізовані адаптери. Окрім цього, через обробники натиснень реалізовано механізм навігації (через об'єкти Intent) до інших екранів системи, зокрема до ProfileActivity та TopDoctorsActivity.

Для оптимізованого відображення динамічних списків застосовано модульні класи-адаптери, які наслідуються від базового RecyclerView.Adapter.

1. Клас CategoryAdapter оперує масивом об’єктів CategoryModel, трансформуючи їх у візуальні елементи категорій медичних послуг. Навігаційне керування всередині адаптера дозволяє переходити до списку лікарів конкретної спеціалізації (TopDoctorsActivity) із передачею контексту через параметри Intent.

2. Класи TopDoctorAdapter та TopDoctorAdapter2 орієнтовані на роботу із сутностями медичного персоналу DoctorsModel. Зокрема, TopDoctorAdapter2 містить логіку детального представлення картки лікаря (включаючи відображення рейтингу, спеціалізації та фотографії) та ініціює перехід до екрану розгорнутої інформації – DetailActivity, передаючи туди серіалізований об’єкт моделі за допомогою ключа "Object".

3. Кожен із зазначених адаптерів містить внутрішні службові класи Viewholder (наприклад, CategoryAdapter.Viewholder та TopDoctorAdapter2.Viewholder), які використовують технологію ViewBinding для швидкого й безпечного доступу до XML-компонентів розмітки, що запобігає зайвому дублюванню операцій пошуку віджетів у пам'яті.

Сутності предметної області зосереджені в пакеті Domain та представлені класами моделей:

1. CategoryModel містить властивості Name (назва медичного напрямку) та Picture (посилання на графічний ресурс).

2. DoctorsModel є складною структурою даних, яка акумулює вичерпну інформацію про лікаря: анкетні дані (Name, Biography), професійні характеристики (Special, Experience, Rating), контактну інформацію (Mobile, Site) та координати (Address, Location).

Клас DetailActivity, що також є спадкоємцем BaseActivity, виконує роль екрана детальної інформації. Метод getBundl() забезпечує десеріалізацію отриманого об'єкта DoctorsModel та наповнює інтерфейс даними. Окрім суто інформаційної функції, DetailActivity інтегрує додаток із системними сервісами операційної системи Android через неявні інтенцій (Implicit Intents), дозволяючи користувачеві здійснювати телефонні дзвінки (ACTION_DIAL), надсилати SMS-повідомлення (ACTION_SENDTO), переглядати вебсайти лікарів чи їхнє розташування на картах (ACTION_VIEW). Кінцева точка взаємодії на цьому екрані реалізована через кнопку запису, яка перенаправляє користувача до AppointmentActivity для формування заявки на прийом.

Екран профілю користувача винесено в окремий ізольований компонент ProfileActivity, що відповідає за відображення особистих даних та базові навігаційні дії, такі як повернення на попередній екран за допомогою методу finish().

Такий модульний підхід, заснований на чіткому розподілі обов'язків (Separation of Concerns), гарантує високу стабільність додатка. Логіка збереження та обробки даних повністю відокремлена від логіки відображення. Це дозволяє безболісно масштабувати додаток (наприклад, змінювати джерело даних у MainViewModel чи додавати нові типи адаптерів), не порушуючи роботу вже існуючих екранів, а також спрощує процес проведення модульного та інструментального тестування компонентів системи.

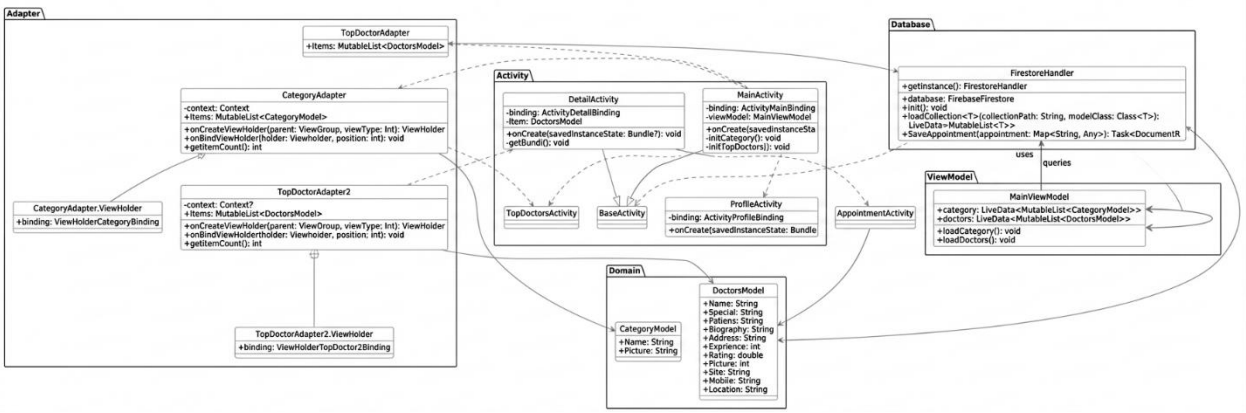


Рис 3.1 Діаграма класів

3.4 Інтерфейс користувача

Процес взаємодії користувача з мобільним додатком починається з головного екрана MainActivity, розробленого на основі архітектурного шаблону MVVM. Цей екран є стартовою точкою взаємодії користувача із системою й виконує роль головного навігаційного хабу. Поведінка, сценарії дій користувача та програмний опис елементів інтерфейсу мають наступний вигляд:

1. Панель вітання та автентифікації користувача – під час відкриття додатка користувач бачить у верхній лівій частині персоналізоване вітання із власним іменем («Hi, Liza Bondarenko»), що підтверджує успішний вхід у систему. У верхньому правому куті розміщено графічний елемент сповіщень із лічильником нечитаних повідомлень та іконку швидкого переходу до профілю. При натисканні на кнопку профілю (accountBtn) архітектурний сценарій ініціює явний Intent, який відкриває екран особистих даних пацієнта (ProfileActivity).

2. Банер швидкої дії та пошукова діяльність – центральну частину верхньої зони займає яскравий інформаційний банер із закликом до дії («Book and Schedule with nearest doctor»). Під ним інтегровано текстове поле швидкого пошуку (Searching...) з іконкою лупи. У разі потреби миттєвого пошуку конкретного лікаря користувач взаємодіє з цим інтерактивним полем. Введення перших літер імені або спеціалізації дозволяє в реальному часі відфільтрувати інформацію, мінімізуючи час на пошук потрібного спеціаліста.

3. Інфікатори асинхронного завантаження даних (ProgressBar) – візуальні компоненти progressBarCategory та progressBarTopDoctor. Оскільки додаток використовує реактивне програмування, завантаження списків із бази даних Firebase Firestore відбувається асинхронно через MainViewModel. На початку запиту індикатори переходять у стан View.VISIBLE. Після того як дані успішно отримані та передані в адаптери, їхній стан змінюється на View.GONE. Користувач не відчуває «зависання» інтерфейсу, а бачить плавний перехід від стану завантаження до відображення готових елементів.

4. Вибір медичного напрямку («Doctor Speciality») – динамічний список на основі компонента RecyclerView (viewCategory), що використовує горизонтальний менеджер розмітки LinearLayoutManager.HORIZONTAL. За відображення карток відповідає CategoryAdapter. Гортаючи цей список за допомогою жестів свайпу (скролінгу), користувач візуально ідентифікує потрібну галузь за допомогою фіолетових графічних іконок (Cardiology, Dentistry, Neurology, Orthopedics, Radiology) та текстових підписів (titleTxt). Текстова кнопка «See all» відкриває повний список категорій. При виборі конкретного елемента адаптер ініціює перехід до TopDoctorsActivity, передаючи назву обраного напрямку як фільтр.

5. Стрічка популярних спеціалістів («Top Doctors») – нижній функціональний блок recyclerViewTopDoctor, керований адаптером TopDoctorAdapter. Він виводить картки лікарів (наприклад, Dr. David Johnson) у вигляді прямокутних блоків із закругленими кутами. Користувач має можливість ознайомитися з картками рекомендованих лікарів, де візуально оцінює фотографію лікаря (яка завантажується асинхронно через бібліотеку Glide), його медичну категорію, а також ключові критерії вибору: рейтинг (зафіксований за допомогою жовтої зірки та числової оцінки 4.2) та професійний стаж у роках з іконкою медичної валізи. Поруч із заголовком секції розміщено кнопку doctorListTxt («See all»), натискання на яку перенаправляє користувача на повний перелік спеціалістів у TopDoctorsActivity.

6. Нижня панель навігації (Bottom Navigation Bar) – забезпечує глобальне переміщення між розділами додатка. За допомогою нижньої панелі користувач може в один клік перемикатися між головними розділами завдяки чотирьом інтерактивним вкладкам з відповідними піктограмами: «Explorer» (огляд головного екрана), «Wishlist» (обрані лікарі), «Settings» (налаштування системи) та «Account» (швидкий доступ).

Переходячи до розділу особистого кабінету (шляхом натискання на аватар у верхній панелі або через вкладку «Account» на нижній панелі навігації головного екрана), користувач опиняється на екрані ProfileActivity. Цей функціональний модуль спроєктовано за принципами Material Design для забезпечення максимальної читабельності персональних даних та оперативного моніторингу статусу поточних записів.

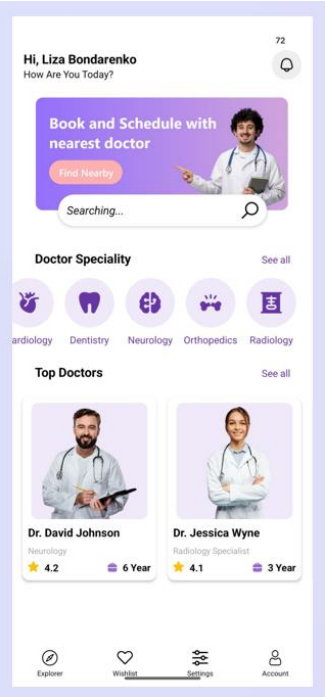


Рис. 3.2 Приклад екранної форми головного екрану

Процес взаємодії користувача з елементами цього інтерфейсу побудовано за лінійним сценарієм і розділено на кілька логічних зон:

1. Первинна навігація (Кнопка «Назад»): у верхньому лівому куті екрана користувач бачить круглу кнопку зі стрілкою повернення. При натисканні на неї ініціюється метод finish(), що дозволяє в один клік закрити екран профілю та

безперешкодно повернутися до попереднього вікна (MainActivity) без втрати контексту сесії.

2. Блок візуальної та текстової ідентифікації: центральну частину верхньої зони займає графічний аватар користувача у вигляді світло-фіолетового кола. Безпосередньо під ним великим фіолетовим шрифтом відображається повне ім'я («Liza Bondarenko»), а нижче, шрифтом меншого розміру, – зареєстрована в системі електронна адреса (l.bondarenko2504@gmail.com) та контактний номер телефону (+380972289241). Це дозволяє користувачеві миттєво верифікувати актуальність своїх анкетних даних.

3. Керування обліковим записом: під контактною інформацією розташована велика кнопка дій із фіолетовим контуром – «EDIT PROFILE». Натискання на неї активує сценарій переходу до форми редагування, де користувач може оновити анкетні дані або змінити фотографію профілю.

Нижня частина екрана відведена під інформаційну стрічку, яка динамічно наповнюється актуальними даними про поточні, заплановані або завершені візити пацієнта до лікарів. З точки зору користувача, цей компонент виконує роль інтерактивного електронного талона, який позбавляє від необхідності запам'ятовувати чи записувати деталі візиту.

Програмно цей блок реалізовано як динамічний текстовий контейнер (або елемент списку RecyclerView), який зчитує масив даних із бази даних Firebase Firestore, прив'язаний до унікального ідентифікатора користувача (userId).

Користувач бачить офіційне підтвердження успішного бронювання прийому, структуроване у вигляді послідовного текстового блоку. Для покращення сприйняття та швидкого візуального сканування інформації інтерфейс містить тематичні графічні акценти (емодзі), які чітко розмежовують смислові блоки:

1. Персоналізоване офіційне звернення (👩): стрічка відкривається рядком «Dear Бондаренко Єлизавета, Your appointment has been successfully booked». Це візуально підтверджує користувачеві, що транзакція запису в базі даних пройшла успішно, а сесія бронювання завершена.

2. Ідентифікація медичного спеціаліста (🧑⚕️): наступний рядок містить чітку вказівку на призначеного лікаря: «Doctor: Dr. Michael Roberts». Користувач може миттєво ідентифікувати особу лікаря, до якого він записався, що мінімізує ризик плутанини при одночасному оформленні кількох прийомів.

3. Часово-координаційний блок (📅, 🕘): інформація про час візиту виділена окремими підпунктами: «Date: 18/5/2026, Time: 09:00». Дані значення динамічно передаються з екрана бронювання (AppointmentActivity). Для користувача це найважливіший координаційний елемент, який дозволяє точно спланувати свій візит до клініки, а структурований вигляд запобігає випадковому пропуску часового слота.

4. Анамнестичний опис скарги пацієнта (📝): завершальний рядок відображає симптоматику, яку користувач власноруч вносив у текстове поле під час реєстрації заявки: «Problem description: Сильні головні болі». Відображення цього поля дозволяє користувачеві переконатися, що його скарга правильно збережена в системі й буде доступна лікарю під час первинного огляду для формування анамнезу.

5. Елемент клієнтської лояльності: блок завершується фінальним рядком ввічливості: «Thank you for choosing our clinic!», що завершує формування позитивного користувацького досвіду (UX) та сигналізує про високу клієнтоорієнтованість медичного закладу.

Після перегляду інформації про запланований візит, користувач використовує кнопку «Назад» для повернення до головного меню. Архітектурне зв'язування графічних елементів через технологію ViewBinding (об'єкт ActivityProfileBinding) гарантує стабільну роботу інтерфейсу, відсутність затримок та миттєвий відгук системи на дії користувача.

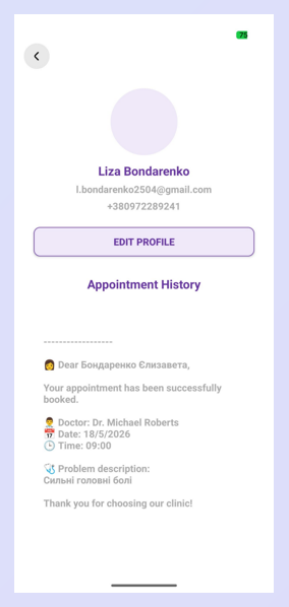


Рис. 3.3 Приклад екранної форми особистого кабінету

Для забезпечення швидкої та точної фільтрації медичного персоналу в системі реалізовано ізольований блок вибору медичних категорій «Doctor Speciality», розташований у центральній частині головного екрана. Цей модуль є одним із первинних інструментів взаємодії користувача з додатком у процесі пошуку спеціаліста.

З технічної сторони логіка роботи цього блоку базується на взаємодії кількох паттернів та компонентів архітектури додатка:

1. Джерело даних (Firebase Firestore): дані про кожну категорію зберігаються у вигляді окремих документів. Кожен документ ініціалізується через модель даних CategoryModel, яка містить два базові атрибути: назву напрямку (Name: String) та посилання на графічну піктограму (Picture: String).

2. Асинхронне отримання через MVVM: при створенні екрана метод initCategory() викликає функцію loadCategory() всередині класу MainViewModel. Запит до бази даних виконується асинхронно, а хід його виконання візуалізується для користувача за допомогою компонента progressBarCategory. Коли об'єкт LiveData типу MutableList<CategoryModel> оновлюється актуальним списком із Firestore, індикатор завантаження приховується (View.GONE).

3. Компонент відображення (RecyclerView): візуалізація списку здійснюється за допомогою віджета viewCategory. Для забезпечення компактності та збереження корисної площі екрана, менеджер розмітки налаштовано у горизонтальному режимі (LinearLayoutManager.HORIZONTAL), що дозволяє користувачеві взаємодіяти зі списком за допомогою стандартного жесту горизонтального прокручування (свайпу).

Під час роботи з блоком «Doctor Speciality» користувач проходить такий покроковий сценарій:

1. Візуальне сканування: після приховування індикатора завантаження користувач бачить лінійний ряд округлих іконок фіолетового кольору, під кожною з яких розміщено текстовий підпис медичного напрямку. Користувач легко ідентифікує потрібну галузь медицини завдяки зрозумілим піктограмам:
    1) Cardiology (Кардіологія) – іконка із зображенням серця.
    2) Dentistry (Стоматологія) – іконка із зображенням зуба.
    3) Neurology (Неврологія) – іконка із зображенням головного мозку.
    4) Orthopedics (Ортопедія) – іконка із зображенням суглоба/кістки.
    5) Radiology (Радіологія/Рентгенологія) – іконка з медичним знімком.

2. Ініціалізація кліку (Вибір категорії): Визначившись із необхідним напрямком лікування, користувач натискає на відповідну іконку або підпис. В цей момент у класі CategoryAdapter всередині методу onBindViewViewHolder() спрацьовує слухач натискань (setOnClickListener).

3. Програмна реакція та перехід: програма миттєво реагує на дію користувача, створюючи явний об'єкт Intent для відкриття екрана TopDoctorsActivity. При цьому адаптер автоматично додає до інтенту обрану текстову назву спеціалізації (наприклад, "Cardiology" або "Neurology") як стрінговий параметр-фільтр.

4. Альтернативна дія («See all»): якщо користувачеві необхідно переглянути повний, розгорнутий список усіх доступних у клініці медичних

напрямків та підкатегорій, він натискає на текстове посилання «See all», розташоване з правого боку від заголовка секції.

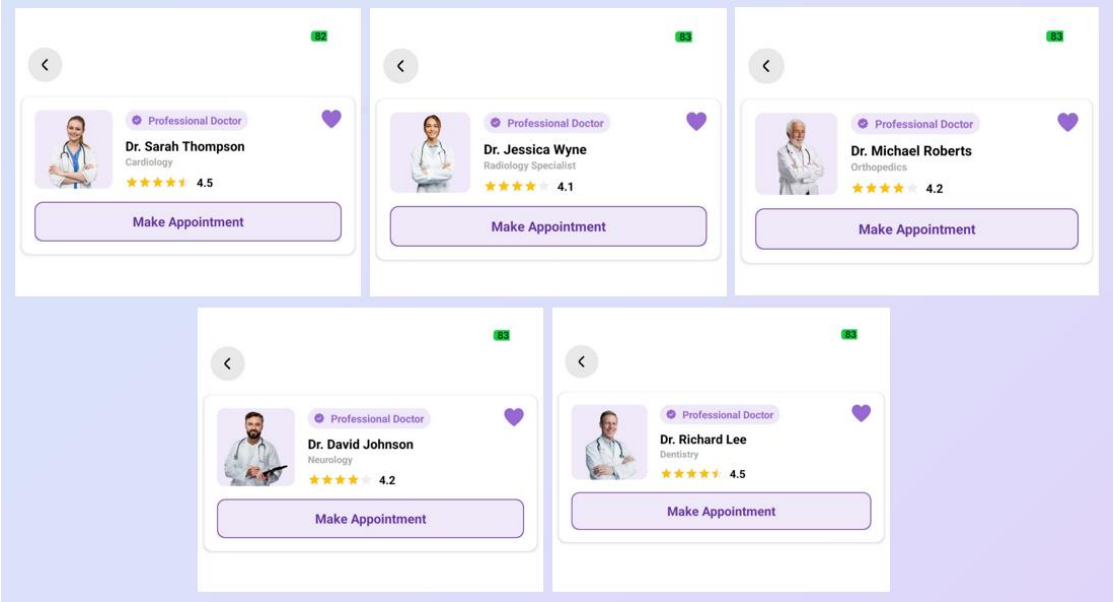


Рис. 3.4 Приклад екранної форми фільтрації лікарів по спеціальностям

Для ознайомлення пацієнта з доступним медичним персоналом клініки та швидкого переходу до бронювання послуг, у нижній частині головного екрана розроблено динамічний інформаційний блок «Top Doctors». Цей розділ агрегує ключові дані про кожного лікаря, дозволяючи користувачеві провести первинну оцінку кваліфікації спеціаліста без відкриття додаткових вікон.

Відображення інформації та зв'язок із серверною частиною додатка базується на взаємодії таких компонентів:

1. Структура об'єкта (Domain Model): сутність кожного медичного працівника описується за допомогою класу даних DoctorsModel. Модель містить повний набір необхідних для виведення та фільтрації атрибутів: Name (ім'я), Special (категорія), Patients (кількість пацієнтів), Biography (біографія), Address (фізична адреса клініки), Experience (стаж роботи), Rating (числовий рейтинг), Picture (посилання на фото в репозиторії Storage), а також контактні дані Site, Mobile та координати Location.

2. Реактивне оновлення інтерфейсу (MVVM): під час ініціалізації головного вікна метод initTopDoctors() звертається до MainViewModel. За допомогою методу loadDoctors() ініціюється асинхронний запит до колекції в Firebase Firestore. Отримані дані публікуються в об'єкт LiveData, за станом якого стежить (observe) головна Activity. На час виконання транзакції користувачеві виводиться progressBarTopDoctor, який автоматично приховується після наповнення контейнера даних.

3. Управління списками (RecyclerView & Adapters): для побудови стрічки карток використовується компонент recyclerViewTopDoctor. Процес зв'язування даних (Data Binding) покладено на клас TopDoctorAdapter (або його оптимізовану версію TopDoctorAdapter2), який використовує вкладений клас ViewHolder та технологію ViewBinding (ViewHolderTopDoctor2Binding) для безпосереднього доступу до графічних віджетів.

4. Візуальна оцінка та критерії вибору: кожна картка спроєктована у вигляді компактного прямокутного блока із закругленими кутами для кращого фокусування уваги. Користувач послідовно аналізує такі елементи:

1) Фотографія лікаря завантажується асинхронно з мережі за допомогою бібліотеки Glide та виводиться в округлений контейнер ImageView.
2) Ім'я та спеціалізація – текстові поля виводять чіткі назви (наприклад, Neurology чи Radiology Specialist), що підтверджує профіль лікаря.
3) Рейтинговий показник – маркується жовтою зіркою та числовим значенням (наприклад, 4.2 або 4.1), що відображає рівень задоволеності інших пацієнтів.

5. Досвід роботи – іконка медичної валізи та підпис (наприклад, 6 Year, 3 Year) вказують на професійний стаж спеціаліста.

6. Перегляд усіх спеціалістів: якщо користувачеві недостатньо виведених на головній сторінці лікарів, він натискає на посилання «See all» у заголовку розділу, що перенаправляє його на повний перелік медичного персоналу в TopDoctorsActivity.

7. Ініціалізація переходу до деталей (клік на картку): натискання на будь-яку область картки лікаря перехоплюється слухачем setOnClickListener в адаптері. Система ініціює явний Intent, прикріплюючи поточний об'єкт DoctorsModel через Bundle ("Object"), та відкриває екран детальної інформації про лікаря (DetailActivity).

Опинившись на екрані деталей (DetailActivity), користувач бачить розгорнуту інформацію, яка повністю відповідає об'єкту обраного лікаря (наприклад, картці Dr. Sarah Thompson або Dr. Michael Roberts). На цьому етапі користувач отримує доступ до розширеного аналізу та взаємодії з елементами інтерфейсу:

8. Аналіз досвіду та адреси: користувач перевіряє точну фізичну адресу розташування кабінету лікаря (8502 Preston Rd, Inglewood, Maine 98380), аналізує загальну кількість пацієнтів, які вже скористалися послугами (500+ / 1200+), детальний стаж (4 Years / 20 Years) та підтверджений рейтинг (4.5 / 4.2).

9. Вивчення біографії: окремий блок Biography надає розгорнутий текстовий опис кваліфікації медичного спеціаліста, його сертифікації, основних напрямків лікування та підходів до взаємодії з пацієнтами.

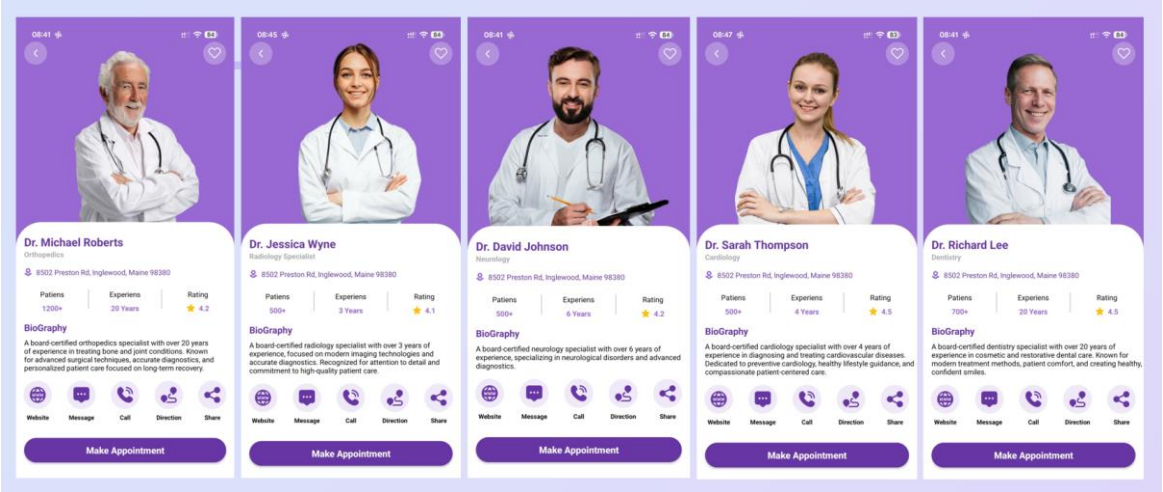


Рис. 3.5 Приклад екранної форми топ спеціалістів

Для забезпечення швидкого зв'язку з клінікою, інтеграції із зовнішніми картами та ініціалізації цільової дії у нижній частині екрана деталей (DetailActivity)

реалізовано горизонтальну панель швидких дій. Процес взаємодії користувача з п'ятьма круглими кнопками комунікації та головною кнопкою запису має таку програмну та інтерфейсну логіку:

1. Взаємодія з кнопкою «Website» (інтеграція з веб): при натисканні користувачем на кнопку websiteBtn система зчитує URL-адресу з атрибута item.Site моделі DoctorsModel. Програма створює неявний інтент Intent(Intent.ACTION_VIEW, Uri.parse(item.Site)). Для користувача це виглядає як миттєве згортання додатка та відкриття вбудованого системного браузера (наприклад, Google Chrome), який завантажує офіційну веб-сторінку лікаря або клініки для ознайомлення з ліцензіями та додатковими сертифікатами спеціаліста.

2. Взаємодія з кнопкою «Message» (текстова комунікація за протоколом SMS): кнопка messageBtn відповідає за швидку відправку текстових повідомлень спеціалісту або в реєстратуру клініки. Натискання на неї активує неявний інтент Intent(Intent.ACTION_SENDTO) із передачею схеми smsto: та номера мобільного телефону лікаря item.Mobile.

Як результат, операційна система Android перенаправляє користувача у свій додаток за замовчуванням «Повідомлення». При цьому інтерфейс автоматично підставляє зчитаний номер (наприклад, 00123456789) у поле одержувача, а в текстове поле введення за допомогою ключів Intent.putExtra заздалегідь імпортується шаблон вітання («Hello»). Це дозволяє пацієнтові розпочати діалог в один клік без необхідності вручну копіювати контакти.

3. Взаємодія з кнопкою «Call» (прямий телефонний виклик через діалер): якщо пацієнту необхідно терміново зв'язатися з лікарем або реєстратурою в голосовому режимі, він взаємодіє з кнопкою callBtn. Програма викликає неявний системний інтент Intent(Intent.ACTION_DIAL, Uri.parse("tel:" + item.Mobile.trim())).

Безпечний виклик через прапорець ACTION_DIAL (на відміну від ACTION_CALL, який вимагає додаткових дозволів READ_PHONE_STATE у маніфесті та здійснює дзвінок автоматично) відкриває штатний системний додаток «Телефон». У цифрове поле миттєво підставляється номер телефону лікаря, відформатований відповідно до міжнародних стандартів (наприклад, 00 1 234-567-

89). Користувачеві залишається лише натиснути інтерактивну зелену кнопку «Виклик» для старту розмови. Такий підхід захищає користувача від випадкових фінансових витрат та забезпечує повний контроль над процесом виклику.

4. Взаємодія з кнопкою «Direction» (геонавігація та карти): кнопка directionBtn призначена для побудови оптимального маршруту до кабінету лікаря чи корпусу клініки. Вона використовує географічні координати або текстову адресу з атрибута item.Location. Система формує інтент для картографічних сервісів (наприклад, Google Maps), дозволяючи користувачеві в один клік побачити точне розташування лікарні на мапі міста та розрахувати час у дорозі від свого поточного місця перебування.

5. Взаємодія з кнопкою «Share» (поширення інформації): При взаємодії з кнопкою shareBtn ініціюється інтент Intent(Intent.ACTION_SEND) з типом даних text/plain. Програма агрегує ім'я лікаря, його спеціалізацію, адресу та номер телефону в один структурований текстовий блок і викликає рідне системне вікно Android «Поділитися». Користувач може миттєво надіслати контакти перевіреного лікаря своїм близьким через будь-який зручний месенджер (Telegram, Viber) або соціальну мережу.

6. Ініціалізація цільової дії – кнопка «Make Appointment»: фінальним етапом взаємодії з екраном деталей є велика фіолетова кнопка makeBtn, яка займає всю ширину нижньої частини екрана. Переконавшись у виборі спеціаліста, користувач здійснює клік, що перехоплюється системою та викликає явний Intent для відкриття AppointmentActivity (екрана безпосереднього вибору дати й часу прийому). Разом з інтентом через putExtra() передаються ідентифікаційні дані лікаря, що забезпечує безперервність процесу оформлення медичної заявки.

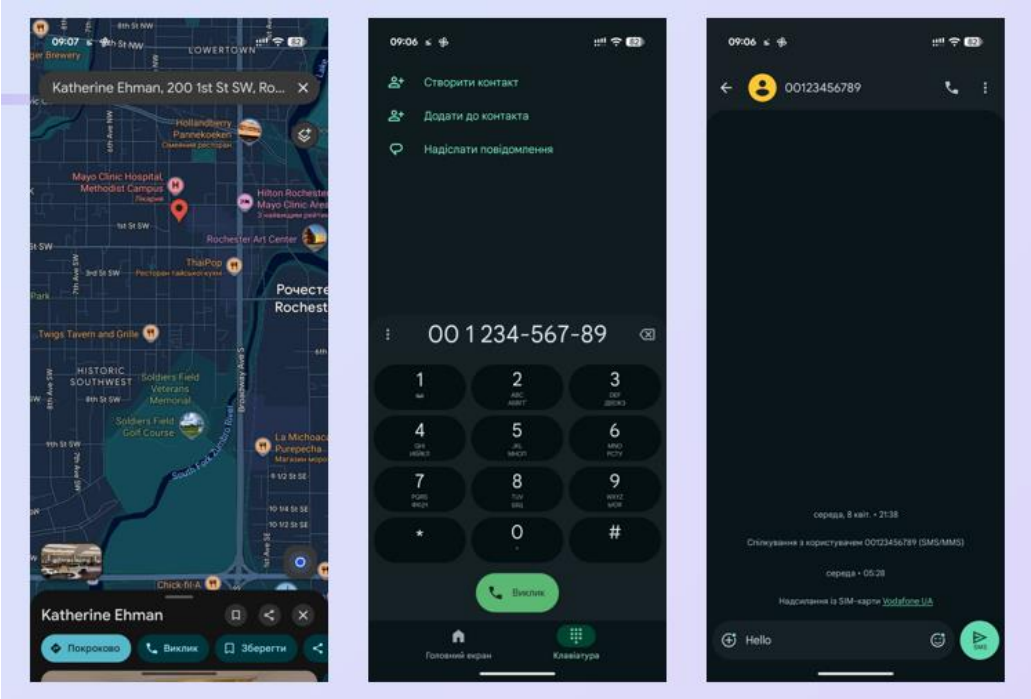


Рис. 3.6 Приклад екранної форми швидкого зв’язку з лікарем та геолокацією

Фінальним етапом цільового сценарію використання додатка є процес формування заявки на медичну консультацію, який реалізовано в межах класу AppointmentActivity. Цей екран акумулює персональні дані пацієнта, параметри розкладу та медичні скарги для подальшого збереження в базі даних.

Функціонування даного модуля базується на наступних програмних рішеннях:

1. Отримання даних через Intent: при відкритті екрана додаток зчитує ім'я обраного спеціаліста (наприклад, Dr. David Johnson), яке було передане з DetailActivity. Це забезпечує безперервність контексту: заголовок екрана («Appointment with...») динамічно оновлюється, підтверджуючи користувачеві, до якого саме лікаря ініційовано запис.

2. Зв’язування даних (ViewBinding): використання об'єкта ActivityAppointmentBinding дозволяє безпечно взаємодіяти з текстовими полями введення (EditText) та забезпечує високу швидкість обробки подій натискання.

3. Модель даних запису: усі введені користувачем дані (прізвище, телефон, час, опис проблеми) консолідуються в окремий об'єкт або Map-структуру для подальшої транзакції в колекцію Appointments у Firebase Firestore.

Інтерфейс екрана спроєктований як лінійна форма, яку користувач заповнює згори донизу:

4. Блок ідентифікації та контактів: користувач вводить або перевіряє свої персональні дані у відповідних полях:

1) Прізвище та ім'я («Бондаренко Єлизавета»).
2) Контактний номер телефону («0972289241»).
3) Електронна адреса («l.bondarenko2504@gmail.com»). Ці дані є критично важливими для зворотного зв'язку реєстратури клініки з пацієнтом.

5. Вибір часових параметрів візиту: за допомогою інтерактивних полів користувач встановлює бажану дату («18/5/2026») та точний час («09:00») прийому. Програмно ці поля можуть бути інтегровані з системними компонентами DatePicker та TimePicker, що мінімізує помилки ручного введення.

6. Верифікація обраного спеціаліста: у секції «Choose doctor» користувач бачить підтвердження обраного раніше лікаря, що виключає ризик помилкового запису до іншого фахівця.

7. Формування первинної скарги: у нижньому текстовому полі користувач вводить опис своєї проблеми (наприклад, «Головні болі»). Це поле дозволяє лікарю заздалегідь ознайомитися з причиною візиту та підготуватися до консультації.

Завершальним кроком є взаємодія з головною кнопкою дії – «Confirm Appointment». При натисканні на цей елемент програмний модуль виконує наступні операції:

1. Валідація полів: перевірка, чи не є поля порожніми та чи відповідають вони заданим форматам (наприклад, коректність e-mail).

2. Синхронізація з БД: відправка сформованого пакета даних до хмарного сховища.

3. Зворотний зв'язок: після успішного збереження даних система виводить повідомлення про успіх та перенаправляє користувача до головного меню або екрана профілю для перегляду оновленої історії записів.

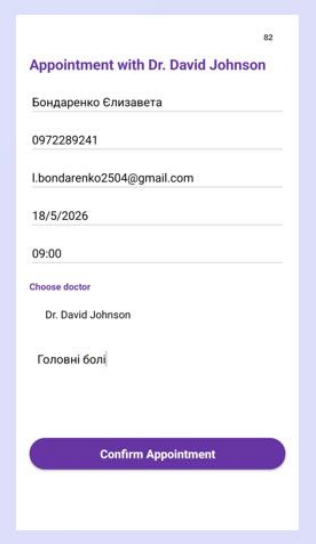


Рис. 3.7 Приклад екранної форми для запису до лікаря

3.8 Тестування вебсайту

Тестування мобільного застосунку на реальних Android-пристроях Google Pixel, а саме: Google Pixel 10 Pro, Google Pixel 8 та Google Pixel 7.

Метою тестування була перевірка коректності роботи інтерфейсу, стабільності застосунку, швидкодії та адаптивності на різних версіях Android і пристроях із різними технічними характеристиками.

Під час тестування перевірялася коректність відображення основних екранів застосунку, зокрема головного меню, списку лікарів, сторінки детальної інформації про лікаря, профілю користувача та форми запису на прийом. Було підтверджено, що XML-розмітка та компоненти ConstraintLayout коректно адаптуються під різні розміри дисплеїв без зміщення елементів інтерфейсу або втрати функціональності.

На пристрої Google Pixel 10 Pro було протестовано роботу застосунку в умовах високого навантаження, швидкість синхронізації даних із Firebase

Firestore та плавність роботи RecyclerView при завантаженні списків лікарів. Застосунок працював стабільно, без затримок і зависань, а оновлення даних у режимі реального часу відбувалося коректно.

Тестування на Google Pixel 8 та Google Pixel 7 дозволило перевірити стабільність роботи застосунку при зміні орієнтації екрана, повторному відкритті

Activity та переходах між екранами через Bottom Navigation View. Було встановлено, що навігація між екранами здійснюється коректно, а стан інтерфейсу зберігається без втрати даних користувача.

За результатами тестування встановлено, що мобільний застосунок коректно працює на різних моделях смартфонів Google Pixel, забезпечує адаптивність інтерфейсу, стабільну взаємодію з Firebase та високу швидкодію основних функціональних модулів системи.

Також під час тестування на пристроях Google окремо перевірявся процес відображення запису на прийом до лікаря. Було протестовано коректність роботи форми запису, введення персональних даних користувача, вибору дати та часу прийому через компоненти DatePickerDialog і TimePickerDialog, а також відображення інформації про обраного лікаря.

Під час тестування перевірялося, чи правильно відображаються введені користувачем дані після підтвердження запису, а саме: ім’я пацієнта, електронна пошта, номер телефону, дата та час прийому, а також короткий опис проблеми. Додатково було перевірено коректність формування повідомлення підтвердження запису та передачу даних через Android Intent до поштового клієнта користувача.

Також тестувалася синхронізація інформації із Firebase Firestore у режимі реального часу. Було підтверджено, що після створення запису інформація коректно зберігається у хмарній базі даних та відображається без необхідності повторного запуску застосунку. Під час перевірки не було виявлено помилок відображення інтерфейсу, дублювання записів або втрати введених даних користувача.

ВИСНОВКИ

1. Проведено аналіз предметної галузі організації електронного запису пацієнтів на прийом до лікаря. Визначено основні проблеми традиційних способів запису, зокрема перевантаження реєстратур, складність отримання актуальної інформації про лікарів та недостатній рівень зручності для користувачів.

2. Проведено аналіз існуючих мобільних та вебзастосунків для запису до лікаря, зокрема Helsi, Doc.ua та Medics. Визначено їхні основні функціональні можливості, переваги та недоліки, які були враховані під час розробки власного мобільного застосунку.

3. Досліджено сучасні підходи до розробки мобільних застосунків для платформи Android. Проаналізовано можливості використання Android Studio, Kotlin, Firebase Firestore, XML та інших технологій для створення сучасного та зручного програмного забезпечення.

4. Визначено функціональні та нефункціональні вимоги до мобільного застосунку для електронного запису до лікаря. Проведено проєктування системи за допомогою діаграми варіантів використання та діаграми класів.

5. Проведено аналіз засобів програмної реалізації та обрано технологічний стек для розробки застосунку. Для реалізації клієнтської частини використано Android Studio та мову програмування Kotlin, а для зберігання та синхронізації даних – хмарну платформу Firebase Firestore.

6. Розроблено архітектуру мобільного застосунку та реалізовано основні функціональні модулі системи, зокрема перегляд списку лікарів, пошук за спеціалізаціями, перегляд детальної інформації про лікаря, запис на прийом, роботу з профілем користувача та інтеграцію з Firebase.

7. Реалізовано сучасний користувацький інтерфейс із використанням XML-макетів, RecyclerView, Bottom Navigation View та компонентів Material Design. Інтерфейс адаптовано під різні розміри екранів мобільних пристроїв та забезпечено зручну навігацію між екранами застосунку.

8. Проведено тестування мобільного застосунку на пристроях Google Pixel. У результаті тестування підтверджено стабільність роботи системи, коректність відображення інтерфейсу, працездатність функціоналу запису на прийом та правильність синхронізації даних із Firebase Firestore у режимі реального часу.

9. Результати виконаної роботи підтверджують доцільність використання мобільних технологій та хмарних сервісів для автоматизації процесу запису пацієнтів на прийом до лікаря. Розроблений застосунок забезпечує зручну взаємодію користувачів із медичними установами та відповідає сучасним вимогам до мобільних інформаційних систем.

10. Робота пройшла апробацію:

1) Бондаренко Є.Р., Жебка В.В. Проєктування та реалізація мобільної інформаційної системи для запису пацієнтів на прийом до лікаря // Всеукраїнська науково-технічну конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1–3 травня 2026 року, м. Київ. Збірник тез. (Подано до друку).

2) Бондаренко Є.Р., Жебка В.В. Використання хмарних обчислювальних технологій Firebase у мобільному застосунку // Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15 травня 2026 року, м. Київ. Збірник тез. (Подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Firebase Documentation. Firebase documentation. Google, 2026. URL: https://firebase.google.com/docs (дата звернення: 19.05.2026).

2. Android Developers. Android Developers documentation. Google, 2026. URL: https://developer.android.com (дата звернення: 19.05.2026).

3. Kotlin Documentation. Kotlin programming language documentation. JetBrains, 2026. URL: https://kotlinlang.org/docs/home.html (дата звернення: 19.05.2026).

4. Material Design for Android. Material Design Guidelines. Google, 2026. URL: https://m3.material.io/develop/android (дата звернення: 19.05.2026).

5. RecyclerView Overview. Android Developers documentation. Google, 2026. URL: https://developer.android.com/develop/ui/views/layout/recyclerview (дата звернення: 19.05.2026).

6. Firebase Firestore Documentation. Cloud Firestore documentation. Google, 2026. URL: https://firebase.google.com/docs/firestore (дата звернення: 19.05.2026).

7. Figma. Figma official website. Figma Inc., 2026. URL: https://www.figma.com (дата звернення: 19.05.2026).

8. Android Jetpack Guide. Android Jetpack documentation. Google, 2026. URL: https://developer.android.com/jetpack (дата звернення: 19.05.2026).

9. ConstraintLayout Documentation. Android Developers documentation. Google, 2026. URL: https://developer.android.com/develop/ui/views/layout/constraint-layout (дата звернення: 19.05.2026).

10. Helsi. Медична інформаційна система Helsi. URL: https://helsi.me (дата звернення: 19.05.2026).

11. Doc.ua. Сервіс онлайн-запису до лікаря. URL: https://doc.ua/ua/doctors/kiev/all (дата звернення: 19.05.2026).

12. Medics. Онлайн-сервіс запису до лікаря. URL: https://medics.ua (дата звернення: 19.05.2026).

13. Android Developers. Guide to app architecture. Google, 2026. URL: https://developer.android.com/topic/architecture (дата звернення: 19.05.2026).

14. Google Firebase. Cloud Firestore overview. Google, 2026. URL: https://firebase.google.com/products/firestore (дата звернення: 19.05.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Технологій цифрового розвитку

Мобільний додаток для електронного запису до лікаря на платформі Android із використанням Firebase

Виконала студентка 4 курсу
Групи ТЦР-44
Бондаренко Єлизавета Романівна
Керівник роботи Д-р
техн. наук, професор, зав. кафедри ТЦР Жебка Вікторія Вікторівна

Київ-2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підвищення зручності, швидкості та ефективності взаємодії користувачів із медичними установами шляхом розробки сучасного мобільного додатка та реалізації функціоналу онлайн-запису до лікаря, перегляду інформації про спеціалістів, а також персоналізованого профілю користувача.

Об'єкт дослідження – процес організації запису пацієнтів на прийом до лікаря.

Предмет дослідження – підходи, технології та засоби створення зручного користувацького інтерфейсу мобільного медичного додатка, орієнтованого на швидкий доступ до інформації про лікарів і функціоналу онлайн-запису на прийом, враховуючи різні категорії населення.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.Провести аналіз сучасних мобільних медичних додатків та сервісів онлайн-запису до лікаря.

2.Визначити основні проблеми UI/UX інтерфейсів мобільних медичних застосунків.

3.Сформувати вимоги до функціоналу та інтерфейсу мобільного медичного додатка.

4.Розробити структуру навігації та основні екрани застосунку.

5.Створити зручний та адаптивний UI/UX інтерфейс для взаємодії користувача із системою.

6.Реалізувати функціонал перегляду інформації про лікарів.

7.Можливість онлайн-запису на прийом до обраного фахівця.

8.Провести тестування роботи мобільного додатка та оцінити ефективність реалізованого інтерфейсу.

3

АНАЛІЗ АНАЛОГІВ

Показник	Helsi	Doc.ua	Medics
Тип медичних установ	Переважно державні	Переважно приватні	Приватні та окремі клініки
Функціональність	Широкий функціонал (запис, медичні дані, направлення)	Розширений функціонал (пошук, запис, інформація)	Базовий функціонал (запис, список лікарів)
Інформація про лікаря	Обмежена, залежить від системи	Детальна (досвід, освіта, відгуки)	Обмежена
Зручність інтерфейсу	Середня (може бути складним)	Середня (інколи перевантажений)	Висока (простий і зрозумілий)
Комунікація з лікарем	Обмежена	Частково доступна	Відсутня або обмежена
Інтеграція з eHealth	Є	Відсутня	Відсутня
Рівень доступності	Високий (державний сектор)	Середній (переважно платні послуги)	Середній
Персоналізація	Обмежена	Частково реалізована	Обмежена
Платформи	Web, Android, iOS	Web	Web / мобільна версія
Інтерфейс	Функціональний, може бути складним	Інтуїтивний, але інформаційно насичений	Простий, інтуїтивний
Доступ	Доступ після реєстрації та авторизації	Частково відкритий, запис після реєстрації	Доступ після реєстрації

4

Проблеми існуючих рішень

Метод розробки та оптимізації UI/UX інтерфейсу мобільного додатка передбачає поетапне покращення взаємодії користувача із системою та забезпечення швидкого доступу до основних функцій застосунку.

Основні етапи методу:

- аналіз потреб користувачів мобільного медичного додатка;
- визначення основних сценаріїв використання застосунку;
- побудова логічної структури екранів мобільного додатка;
- оптимізація навігації між основними розділами застосунку;
- створення зручних форм для пошуку лікарів і запису на прийом;
- розробка сучасного та візуально узгодженого UI-дизайну;
- тестування зручності мобільного додатка та внесення покращень у користувацький інтерфейс.

Запропонований метод дозволяє зробити мобільний додаток більш зрозумілим, послідовним та зручним для користувачів.

5

Запропонований метод розробки мобільного додатка

Метод розробки та оптимізації інтерфейсу мобільного додатка передбачає поетапне покращення взаємодії користувача із системою та забезпечення швидкого доступу до основних функцій застосунку.

Основні етапи методу:

1. Аналіз потреб користувачів мобільного медичного додатка.
2. Визначення основних сценаріїв використання застосунку.
3. Побудова логічної структури екранів мобільного додатка.
4. Оптимізація навігації між основними розділами застосунку.
5. Створення зручних форм для пошуку лікарів і запису на прийом.
6. Розробка сучасного та візуально узгодженого ui-дизайну.
7. Тестування зручності мобільного додатка та внесення покращень у користувацький інтерфейс.

Запропонований метод дозволяє зробити мобільний додаток більш зрозумілим, послідовним та зручним для користувачів.

6

Вимоги до мобільного медичного додатка

Функціональні вимоги:

- Система повинна надавати можливість переглядати список лікарів із короткою інформацією про них.
- Користувач повинен мати можливість шукати лікарів за спеціалізацією.
- Користувач повинен мати можливість записатися на прийом до обраного лікаря.
- Користувач повинен мати доступ до перегляду та редагування персональної інформації профілю.
- Дані про лікарів і записи на прийом повинні зберігатися у Firebase Firestore.

Нефункціональні вимоги:

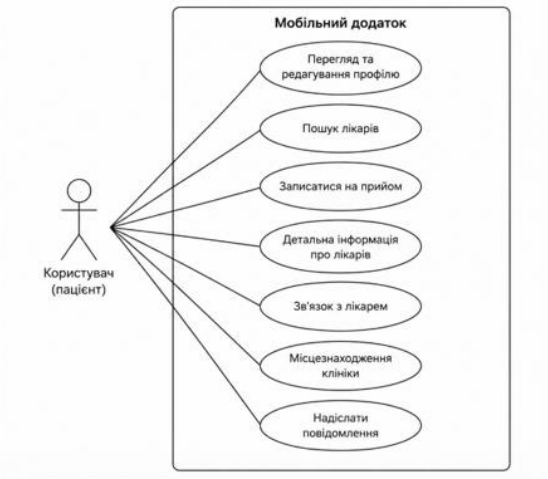
- Інтерфейс застосунку має бути простим, сучасним і зручним для користувача.!
- Застосунок повинен швидко реагувати на дії користувача та забезпечувати оперативне завантаження даних.
- Використання Firebase дозволяє забезпечити автоматичне масштабування системи без необхідності налаштування власного сервера.
- Мобільний застосунок повинен коректно працювати на більшості Android-пристроїв.

Програмні засоби реалізації

1. Android Studio — у процесі розробки мобільного застосунку Android Studio використовувалося як основне середовище програмування. За допомогою нього здійснювалися створення структури проєкту, розробка інтерфейсу користувача за допомогою XML-розмітки, написання програмного коду, а також тестування та налагодження функціональних можливостей системи.
2. Kotlin — у даному проєкті Kotlin використовувався як основна мова програмування для реалізації логіки мобільного застосунку. За допомогою цієї мови було реалізовано обробку даних, взаємодію з базою даних, обробку подій користувача, а також навігацію між екранами застосунку.
3. Firebase — у рамках даного проєкту Firebase використовувався для організації зберігання та обробки даних, а також для забезпечення взаємодії між користувачем і системою. Зокрема, база даних застосовувалася для зберігання інформації про лікарів, користувачів та записів на прийом.
4. Figma — за допомогою Figma було розроблено макети основних екранів застосунку, зокрема головного екрану, списку лікарів, сторінки детальної інформації про лікаря, а також екранів запису на прийом. Це дозволило визначити розташування елементів інтерфейсу, логіку навігації між екранами та забезпечити зручність користування застосунком.

Архітектура системи та навігація мобільного додатка

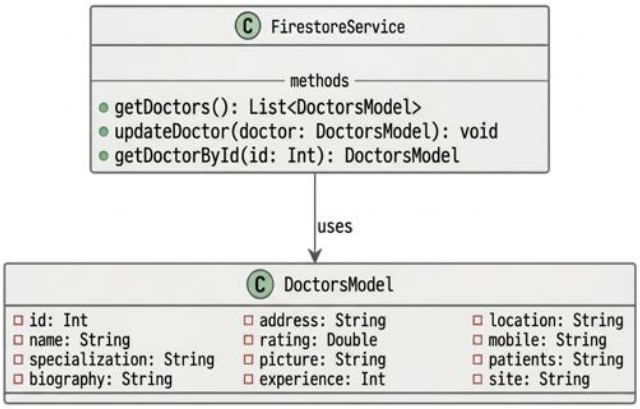
Діаграма варіантів використання



9

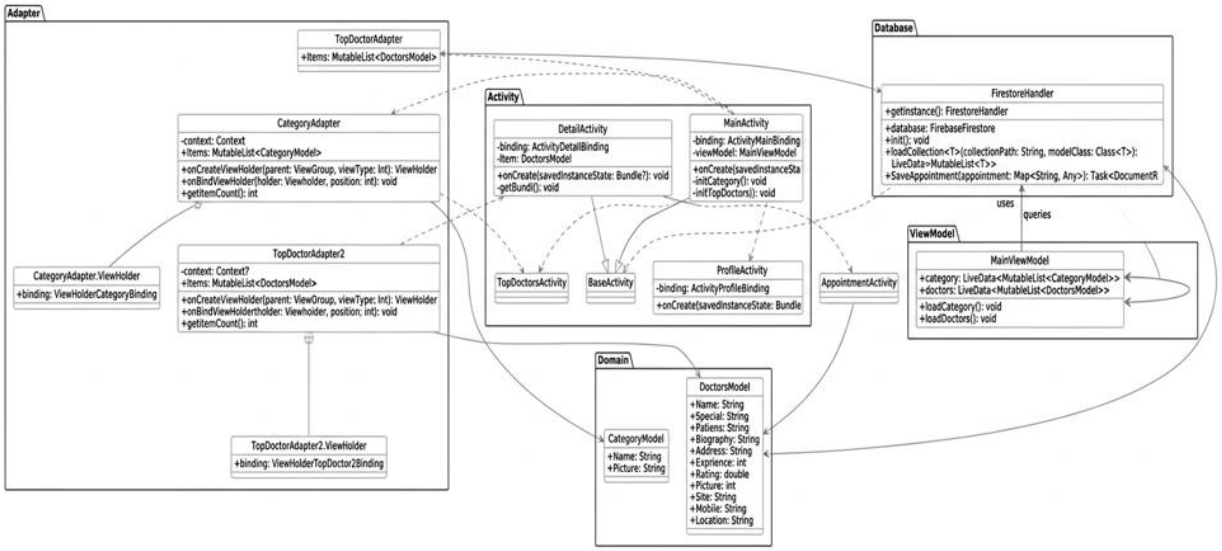
Архітектура системи та навігація мобільного додатка

Діаграма класів



10

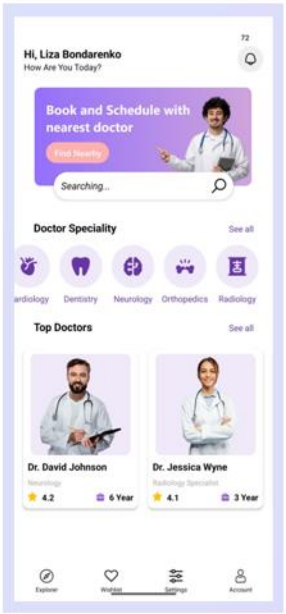
Архітектура системи та навігація мобільного додатка



11

Екранні форми інтерфейсу

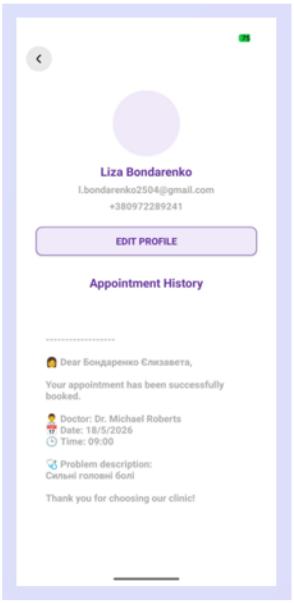
Початкова то головна сторінки



12

Екранні форми інтерфейсу

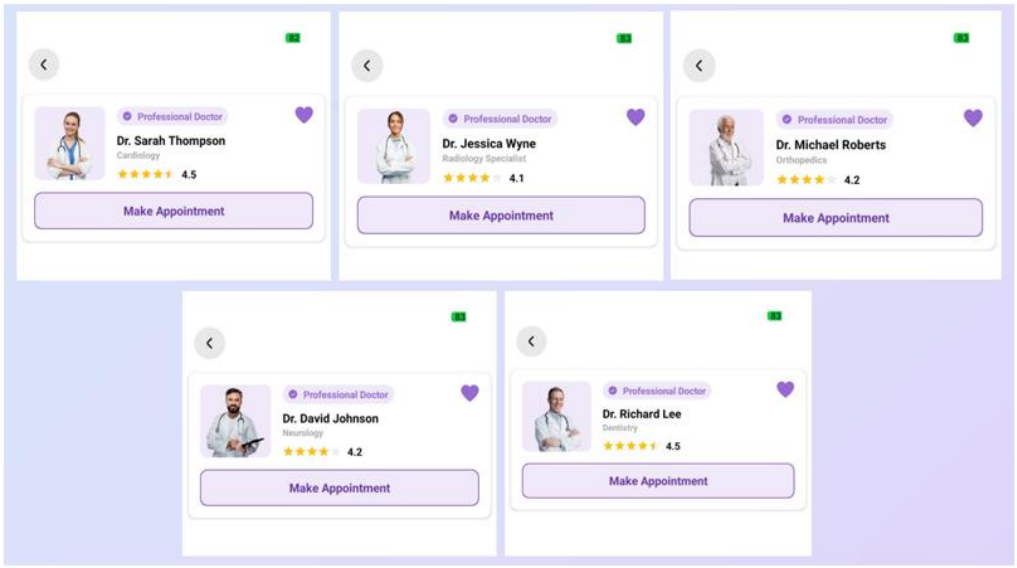
Профіль користувача



13

Екранні форми інтерфейсу

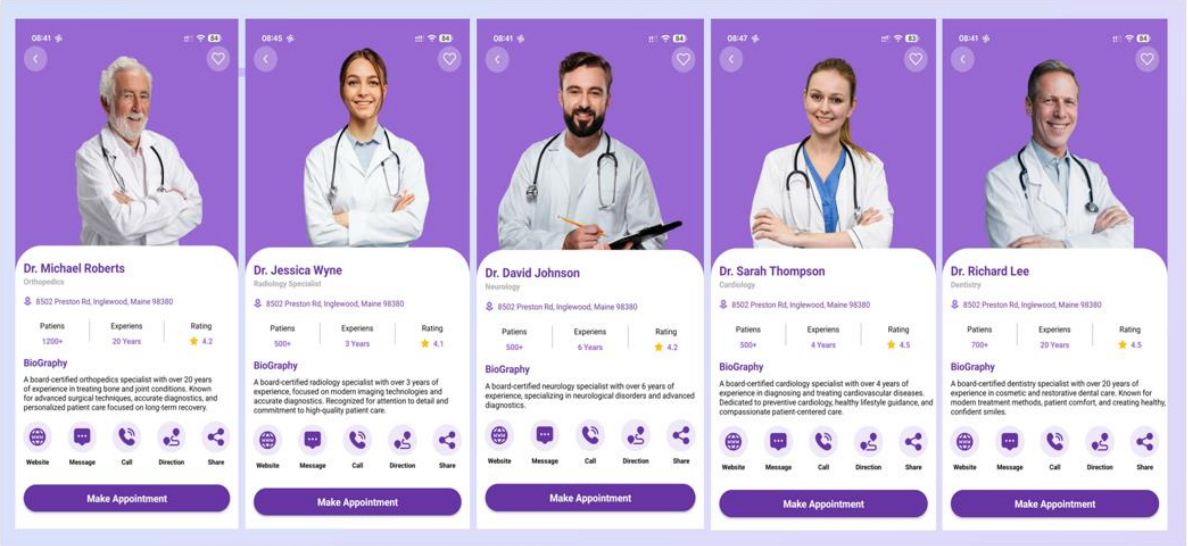
фільтрація лікарів по спеціальностям



14

Екранні форми інтерфейсу

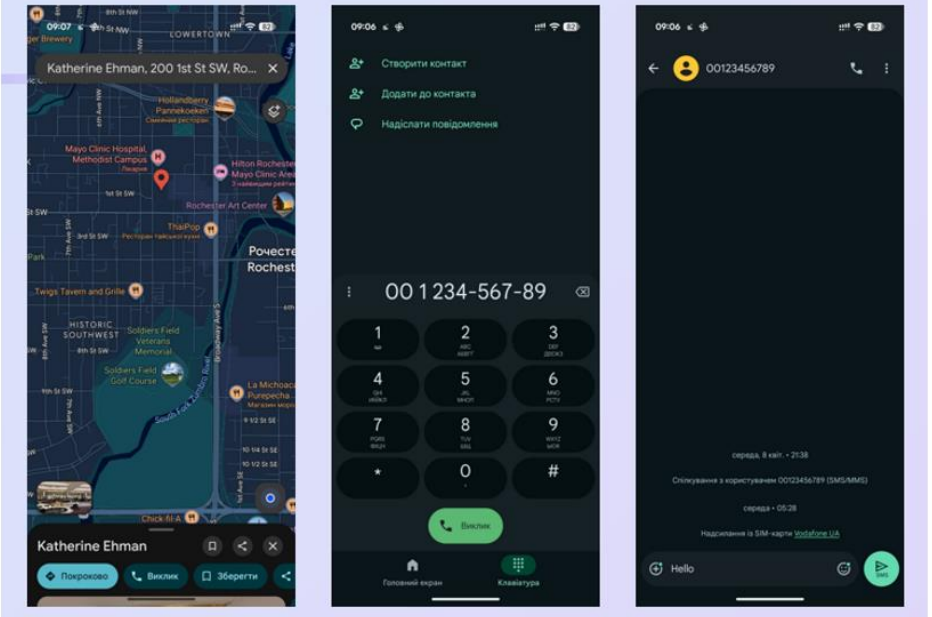
екранна форма топ спеціалістів



15

Екранні форми інтерфейсу

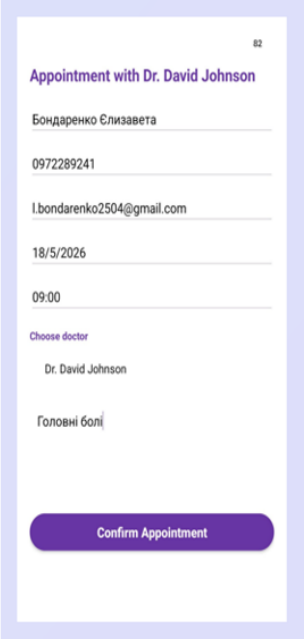
екранної форми швидкого зв’язку з лікарем та геолокацією



16

Екранні форми інтерфейсу

Користувач



17

АПРОБАЦІЯ

1. Бондаренко Є.Р., Жебка В.В. Проєктування та реалізація мобільної інформаційної системи для запису пацієнтів на прийом до лікаря // Всеукраїнська науково-технічну конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1–3 травня 2026 року, м. Київ. Збірник тез. (Подано до друку).

1. Бондаренко Є.Р., Жебка В.В. Використання хмарних обчислювальних технологій Firebase у мобільному застосунку // Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15 травня 2026 року, м. Київ. Збірник тез. (Подано до друку).

18

ВИСНОВКИ

Розроблено мобільний медичний додаток із сучасним UI/UX інтерфейсом, який забезпечує зручну навігацію, швидкий доступ до інформації про лікарів та можливість онлайн-запису на прийом.

Основні результати:

- проаналізовано аналоги мобільних медичних застосунків та визначено їхні UI/UX недоліки;
- сформовано вимоги до інтерфейсу мобільного додатка;
- розроблено структуру екранів та систему навігації;
- реалізовано функціонал перегляду лікарів, запису на прийом і профілю користувача;
- створено адаптивний та візуально цілісний дизайн інтерфейсу.

Результатом роботи є мобільний медичний додаток із логічною структурою, персоналізованим профілем користувача та зручним механізмом взаємодії із системою.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
// Файл: MainViewActivity.kt
// Завантаження даних із Firebase
package com.example.appointmentapp_10.ViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.example.appointmentapp_10.Domain.CategoryModel
import com.example.appointmentapp_10.Domain.DoctorsModel
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ValueEventListener
class MainViewModel(): ViewModel() {
   private  val firebaseDatabase = FirebaseDatabase.getInstance()
   private val _category =
MutableLiveData<MutableList<CategoryModel>>()
   private val _doctors = MutableLiveData<MutableList<
DoctorsModel>>()
   val category: LiveData<MutableList<CategoryModel>> = _category
   val doctors: LiveData<MutableList<DoctorsModel>> = _doctors
   fun loadCategory() {
      val Ref = firebaseDatabase.getReference("Category")
      Ref.addValueEventListener(object : ValueEventListener {
         override fun onDataChange(snapshot: DataSnapshot) {
            val lists = mutableListOf<CategoryModel>()
            for (childSnapshot in snapshot.children) {
               val list =
childSnapshot.getValue(CategoryModel::class.java)
               if (list != null) {
                  lists.add(list)
               }
            }
            _category.value = lists
         }
         override fun onCancelled(error: DatabaseError) {}
      })
   }
      fun loadDoctors(){
         val Ref=firebaseDatabase.getReference("Doctors")
         Ref.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
               val lists= mutableListOf<DoctorsModel>()
               for (childSnapshot in snapshot.children){
                  val
list=childSnapshot.getValue(DoctorsModel::class.java)
                  if(list!=null){
                     lists.add(list)
                  }
               }
               _doctors.value = lists
            }
            override fun onCancelled(error: DatabaseError) {
               TODO("Not yet implemented")
            }
         })
      }
   }
//Файл: DoctorsModel.kt
// Модель даних лікарів
package com.example.appointmentapp_10.Domain
import android.os.Parcel
import android.os.Parcelable
data class DoctorsModel(
   val Address: String = "",
   val Biography: String = "",
   val Id: Int = 0,
   val Name: String = "",
   val Picture: String = "",
   val Special: String = "",
   val Expriense: Int = 0,
   val Location: String = "",
   val Mobile: String = "",
   val Patiens: String = "",
   val Rating: Double = 0.0,
   val Site: String = ""
) : Parcelable {
   constructor(parcel: Parcel) : this(
      parcel.readString().toString(),
      parcel.readString().toString(),
      parcel.readInt(),
      parcel.readString().toString(),
```

```
        parcel.readString().toString(),
        parcel.readString().toString(),
        parcel.readInt(),
        parcel.readString().toString(),
        parcel.readString().toString(),
        parcel.readString().toString(),
        parcel.readDouble(),
        parcel.readString().toString()
    )
    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(Address)
        parcel.writeString(Biography)
        parcel.writeInt(Id)
        parcel.writeString(Name)
        parcel.writeString(Picture)
        parcel.writeString(Special)
        parcel.writeInt(Expriense)
        parcel.writeString(Location)
        parcel.writeString(Mobile)
        parcel.writeString(Patiens)
        parcel.writeDouble(Rating)
        parcel.writeString(Site)
    }
    override fun describeContents(): Int {
        return 0
    }
    companion object CREATOR : Parcelable.Creator<DoctorsModel> {
        override fun createFromParcel(parcel: Parcel): DoctorsModel {
            return DoctorsModel(parcel)
        }
        override fun newArray(size: Int): Array<DoctorsModel?> {
            return arrayOfNulls(size)
        }
    }
}
// Файл: TopDoctorAdapter.kt
// Відображення списку лікарів
package com.example.appointmentapp_10.Adapter
import android.content.Context
import android.content.Intent
import android.util.Log
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.bumptech.glide.request.RequestOptions
import com.bumptech.glide.load.resource.bitmap.CenterCrop
import com.example.appointmentapp_10.Activity.DetailActivity
import com.example.appointmentapp_10.Domain.DoctorsModel
import
com.example.appointmentapp_10.databinding.ViewholderTopDoctorBi
nding
class TopDoctorAdapter(val items: MutableList<DoctorsModel>):
RecyclerView.Adapter<TopDoctorAdapter.Viewholder>() {
    private var context: Context? = null
    class Viewholder(val binding: ViewholderTopDoctorBinding) :
        RecyclerView.ViewHolder(binding.root) {
    }
    override fun onCreateViewHolder(
        parent: ViewGroup,
        viewType: Int
    ): TopDoctorAdapter.Viewholder {
        context = parent.context
        val binding =
            ViewholderTopDoctorBinding.inflate(
                LayoutInflater.from(context),
                parent,
                false
            )
        return Viewholder(binding)
    }
    override fun onBindViewHolder(
        holder: TopDoctorAdapter.Viewholder,
        position: Int
    ) {
        holder.binding.nameTxt.text = items[position].Name
        holder.binding.specialTxt.text = items[position].Special
        holder.binding.scoreTxt.text = items[position].Rating.toString()
        holder.binding.yearTxt.text =
            items[position].Expriense.toString() + "Year"
        Glide.with(holder.itemView.context)
            .load(items[position].Picture)
            .apply(RequestOptions().transform(CenterCrop()))
            .into(holder.binding.img)
        holder.itemView.setOnClickListener {
```

```
            val intent = Intent(context, DetailActivity::class.java)
            intent.putExtra("Object", items[position])
            context?.startActivity(intent)
        }
    }
    override fun getItemCount(): Int = items.size
}
 // AppointmetActivity.kt
// Запис пацієнтів на прийом
package com.example.appointmentapp_10
import android.app.DatePickerDialog
import android.app.TimePickerDialog
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.util.Patterns
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import
com.example.appointmentapp_10.databinding.ActivityAppointmentBin
ding
import com.google.firebase.database.FirebaseDatabase
import java.util.Calendar
class AppointmentActivity : AppCompatActivity() {
    private lateinit var binding: ActivityAppointmentBinding
    private val database =
        FirebaseDatabase.getInstance().reference
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding =
            ActivityAppointmentBinding.inflate(
                layoutInflater
            )
        setContentView(binding.root)
        val doctorName =
            intent.getStringExtra("doctorName")
        binding.doctorNameTxt.text =
            "Appointment with $doctorName"
        // Список лікарів
        val doctors = listOf(
            "Dr. Michael Roberts",
            "Dr. Sarah Thompson",
            "Dr. David Johnson",
            "Dr. Jessica Wyne",
            "Dr. Richard Lee"
        )
        val adapter = ArrayAdapter(
            this,
            R.layout.spinner_item,
            doctors
        )
        adapter.setDropDownViewResource(
            android.R.layout.simple_spinner_dropdown_item
        )
        binding.doctorSpinner.adapter = adapter
        doctorName?.let {
            val position = doctors.indexOf(it)
            if (position >= 0) {
                binding.doctorSpinner
                    .setSelection(position)
            }
        }
        val calendar = Calendar.getInstance()
        // Вибір дати
        binding.dateEdt.setOnClickListener {
            DatePickerDialog(
                this,
                { _, year, month, day ->
                    binding.dateEdt.setText(
                        "$day/${month + 1}/$year"
                    )
                },
                calendar.get(Calendar.YEAR),
                calendar.get(Calendar.MONTH),
                calendar.get(Calendar.DAY_OF_MONTH)
            ).show()
        }
        // Вибір часу
        binding.timeEdt.setOnClickListener {
            TimePickerDialog(
                this,
                { _, hour, minute ->
                    binding.timeEdt.setText(
                        String.format(
```

```
                "%02d:%02d",
                hour,
                minute
            )
        )
    },
    calendar.get(Calendar.HOUR_OF_DAY),
    calendar.get(Calendar.MINUTE),
    true
).show()
}
// Кнопка запису
binding.makeBtn.setOnClickListener {
    val name =
        binding.nameEdt.text
            .toString()
            .trim()
    val phone =
        binding.phoneEdt.text
            .toString()
            .trim()
    val email =
        binding.emailEdt.text
            .toString()
            .trim()
    val date =
        binding.dateEdt.text
            .toString()
            .trim()
    val time =
        binding.timeEdt.text
            .toString()
            .trim()
    val problem =
        binding.problemEdt.text
            .toString()
            .trim()
    val selectedDoctor =
        binding.doctorSpinner
            .selectedItem
            .toString()
    when {
        name.isEmpty() -> {
            binding.nameEdt.error =
                "Enter your name"
            binding.nameEdt.requestFocus()
        }
         phone.isEmpty() -> {
            binding.phoneEdt.error =
                "Enter phone number"
            binding.phoneEdt.requestFocus()
        }
        phone.length < 6 -> {
            binding.phoneEdt.error =
                "Invalid phone number"
            binding.phoneEdt.requestFocus()
        }
        email.isEmpty() -> {
            binding.emailEdt.error =
                "Enter email"
            binding.emailEdt.requestFocus()
        }
        !Patterns.EMAIL_ADDRESS
            .matcher(email)
            .matches() -> {
            binding.emailEdt.error =
                "Invalid email"
            binding.emailEdt.requestFocus()
        }
        date.isEmpty() -> {
            binding.dateEdt.error =
                "Select date"
            binding.dateEdt.requestFocus()
        }
        time.isEmpty() -> {

            binding.timeEdt.error =
                "Select time"
            binding.timeEdt.requestFocus()
        }
        problem.isEmpty() -> {
            binding.problemEdt.error =
                "Describe your problem"
            binding.problemEdt.requestFocus()
```

```
            }
            else -> {
                val message =
                    "👩 Dear $name,\n\n" +
                        "Your appointment has been successfully
booked.\n\n" +
                        "👨‍⚕️ Doctor: $selectedDoctor\n" +
                        "📅 Date: $date\n" +
                        "🕒 Time: $time\n\n" +
                        "📝 Problem description:\n$problem\n\n" +
                        "Thank you for choosing our clinic!"
                // Показ повідомлення
                binding.statusTxt.text = message
                // Дані для Firebase
                val appointment = hashMapOf(
                    "name" to name,
                    "phone" to phone,
                    "email" to email,
                    "doctor" to selectedDoctor,
                    "date" to date,
                    "time" to time,
                    "problem" to problem
                )
                // Збереження у Firebase
                database.child("appointments")
                    .push()
                    .setValue(appointment)
                    .addOnSuccessListener {
                        Toast.makeText(
                            this,
                            "Appointment saved to database!",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                    .addOnFailureListener {
                        Toast.makeText(
                            this,
                            "Database error!",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                // Email
                val emailIntent =
                    Intent(Intent.ACTION_SENDTO)
                emailIntent.data =
                    Uri.parse("mailto:")
                emailIntent.putExtra(
                    Intent.EXTRA_EMAIL,
                    arrayOf(email)
                )
                emailIntent.putExtra(
                    Intent.EXTRA_SUBJECT,
                    "Medical Appointment"
                )
                emailIntent.putExtra(
                    Intent.EXTRA_TEXT,
                    message
                )
startActivity(
                    Intent.createChooser(
                        emailIntent,
                        "Send Email"
                    )
                )
            }
        }
    }
}
// Файл: BaseActivity.kt
// Налаштування інтерфейсів
package com.example.appointmentapp_10.Activity
import android.os.Bundle
import android.view.WindowManager
import androidx.appcompat.app.AppCompatActivity
open class BaseActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        window.setFlags(
        WindowManager.LayoutParams.FLAG_LAYOUT_NO_LIMITS,
WindowManager.LayoutParams.FLAG_LAYOUT_NO_LIMITS
        )
    }
```

```
}
// Файл: DetailActivity.kt
// Укран детальної інформації про лікарів
Відображення детальної інформації про лікарів
package com.example.appointmentapp_10.Activity
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import com.example.appointmentapp_10.AppointmentActivity
import com.example.appointmentapp_10.Domain.DoctorsModel
import
com.example.appointmentapp_10.databinding.ActivityDetailBinding
class DetailActivity : BaseActivity() {
    private lateinit var binding: ActivityDetailBinding
    private lateinit var item: DoctorsModel
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityDetailBinding.inflate(layoutInflater)
        setContentView(binding.root)
        getBundl()
    }
    private fun getBundl() {
        item = intent.getParcelableExtra("Object") ?: return
        binding.apply {
            titleTxt.text = item.Name
            specialTxt.text = item.Special
            patiensTxt.text = item.Patiens
            bioTxt.text = item.Biography
            addressTxt.text = item.Address
            experienceTxt.text =
                item.Expriense.toString() + " Years
            ratingTxt.text = "${item.Rating}"
            img.setImageResource(item.Picture)
            backBtn.setOnClickListener {
                finish()
            }
            websiteBtn.setOnClickListener {
                val intent = Intent(
                    Intent.ACTION_VIEW,
                    Uri.parse(item.Site)
                )
                startActivity(intent)
            }
            messageBtn.setOnClickListener {
                val uri = Uri.parse("sms:${item.Mobile}")
                val intent = Intent(
                    Intent.ACTION_SENDTO,
                    uri
                )
                intent.putExtra(
                    "sms_body",
                    "Hello"
                )
                startActivity(intent)
            }
            callBtn.setOnClickListener {
                val uri = "tel:${item.Mobile.trim()}"
                val intent = Intent(
                    Intent.ACTION_DIAL,
                    Uri.parse(uri)
                )
                startActivity(intent)
            }
            directionBtn.setOnClickListener {

                val intent = Intent(
                    Intent.ACTION_VIEW,
                    Uri.parse(item.Location)
                )
                startActivity(intent)
            shareBtn.setOnClickListener {
                val intent = Intent(Intent.ACTION_SEND)
                intent.type = "text/plain"
                intent.putExtra(
                    Intent.EXTRA_SUBJECT,
                    item.Name
                )
                intent.putExtra(
                    Intent.EXTRA_TEXT,
                    "${item.Name} ${item.Address} ${item.Mobile}"
                )
                startActivity(
                    Intent.createChooser(
```

```
                intent,
                "Choose one"
            )
        )
        makeBtn.setOnClickListener {
            val intent = Intent(
                this@DetailActivity,
                AppointmentActivity::class.java
            )
            intent.putExtra(
                "doctorName",
                item.Name
            )
            startActivity(intent)
        }
    }
  }
}
// Файл: IntroActivity.kt
// Вступний екран застосунку
package com.example.appointmentapp_10.Activity
import android.os.Bundle
import
com.example.appointmentapp_10.databinding.ActivityIntroBinding
import android.content.Intent
class IntroActivity : BaseActivity() {
  private lateinit var binding: ActivityIntroBinding
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
  binding = ActivityIntroBinding.inflate(layoutInflater)
    setContentView(binding.root)
  binding.apply{
    startBtn.setOnClickListener{
      startActivity(Intent(this@IntroActivity,
MainActivity::class.java))
    }
  }
  }
}
// Файл: MainActivity.kt
//  Головний екран медичногоо додатку
package com.example.appointmentapp_10.Activity
import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.lifecycle.Observer
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.appointmentapp_10.Adapter.CategoryAdapter
import com.example.appointmentapp_10.Adapter.TopDoctorAdapter
import com.example.appointmentapp_10.R
import com.example.appointmentapp_10.ViewModel.MainViewModel
import
com.example.appointmentapp_10.databinding.ActivityMainBinding
import androidx.activity.viewModels
class MainActivity : BaseActivity() {
  private lateinit var binding: ActivityMainBinding
  private  val viewModel = MainViewModel()
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding= ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
    initCategory()
    initTopDoctors()
    binding.accountBtn.setOnClickListener {
      val intent = Intent(this@MainActivity,
ProfileActivity::class.java)
      startActivity(intent)
    }
  }
  private fun initTopDoctors() {
    binding.apply {
      progressBarTopDoctor.visibility= View.VISIBLE
      viewModel.doctors.observe(this@MainActivity, Observer{
        recyclerViewTopDoctor.layoutManager=
LinearLayoutManager(this@MainActivity,
LinearLayoutManager.HORIZONTAL,false)
        recyclerViewTopDoctor.adapter= TopDoctorAdapter(it)
        progressBarTopDoctor.visibility= View.GONE
      })
      viewModel.loadDoctors()
```

```
        doctorListTxt.setOnClickListener {
            startActivity(Intent(this@MainActivity,
TopDoctorsActivity::class.java))
        }
    }
  }
  private fun initCategory() {
   binding.progressBarCategory.visibility=View.VISIBLE
    viewModel.category.observe(this, Observer{
        binding.viewCategory.layoutManager=
LinearLayoutManager(this@MainActivity,
            LinearLayoutManager.HORIZONTAL,false)
        binding.viewCategory.adapter= CategoryAdapter(it)
        binding.progressBarCategory.visibility= View.GONE
    })
    viewModel.loadCategory()
  }
}
// Файл: ProfileActivity.kt
// Екран профілю користувача
package com.example.appointmentapp_10.Activity
import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.lifecycle.Observer
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.appointmentapp_10.Adapter.CategoryAdapter
import com.example.appointmentapp_10.Adapter.TopDoctorAdapter
import com.example.appointmentapp_10.R
import com.example.appointmentapp_10.ViewModel.MainViewModel
import
com.example.appointmentapp_10.databinding.ActivityMainBinding
import androidx.activity.viewModels
class MainActivity : BaseActivity() {
  private lateinit var binding: ActivityMainBinding
  private  val viewModel = MainViewModel()
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding= ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
    initCategory()
    initTopDoctors()
    binding.accountBtn.setOnClickListener {
        val intent = Intent(this@MainActivity,
ProfileActivity::class.java)
        startActivity(intent)
    }
  }
  private fun initTopDoctors() {
    binding.apply {
        progressBarTopDoctor.visibility= View.VISIBLE
        viewModel.doctors.observe(this@MainActivity, Observer{
            recyclerViewTopDoctor.layoutManager=
LinearLayoutManager(this@MainActivity,
LinearLayoutManager.HORIZONTAL,false)
            recyclerViewTopDoctor.adapter= TopDoctorAdapter(it)
            progressBarTopDoctor.visibility= View.GONE
        })
        viewModel.loadDoctors()
        doctorListTxt.setOnClickListener {
            startActivity(Intent(this@MainActivity,
TopDoctorsActivity::class.java))
        }
    }
  }
  private fun initCategory() {
   binding.progressBarCategory.visibility=View.VISIBLE
    viewModel.category.observe(this, Observer{
        binding.viewCategory.layoutManager=
LinearLayoutManager(this@MainActivity,
            LinearLayoutManager.HORIZONTAL,false)
        binding.viewCategory.adapter= CategoryAdapter(it)
        binding.progressBarCategory.visibility= View.GONE
    })
    viewModel.loadCategory()
  }
}
package com.example.appointmentapp_10.Activity
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import
com.example.appointmentapp_10.databinding.ActivityProfileBinding
class ProfileActivity : AppCompatActivity() {
```

```
    private lateinit var binding: ActivityProfileBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityProfileBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.backBtn.setOnClickListener {
            finish()
        }
        // Отримання історії записів
        val sharedPreferences =
            getSharedPreferences(
                "appointments",
                MODE_PRIVAT
            )
        val history =
            sharedPreferences.getString(
                "history",
                "No appointments yet
            )
        // Відображення історії
        binding.historyTxt.text = history
    }
}
 // Файл: TopDoctorsActivity.kt
// Список найкращих лфкарів
package com.example.appointmentapp_10.Activity
import android.os.Bundle
import android.view.View
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.lifecycle.Observer
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.appointmentapp_10.Adapter.CategoryAdapter
import com.example.appointmentapp_10.Adapter.TopDoctorAdapter2
import com.example.appointmentapp_10.R
import com.example.appointmentapp_10.ViewModel.MainViewModel
import
com.example.appointmentapp_10.databinding.ActivityTopDoctorsBind
ing
class TopDoctorsActivity : BaseActivity() {
    private lateinit var binding: ActivityTopDoctorsBinding
    private val viewModel= MainViewModel()
    private var selectedCategory: String? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding= ActivityTopDoctorsBinding.inflate(layoutInflater)
        setContentView(binding.root)
        selectedCategory = intent.getStringExtra("category")
        initTopDoctors()
    }
    private fun initTopDoctors() {
       binding.apply {
        binding.progressBarTopDoctor.visibility=View.VISIBLE
          viewModel.doctors.observe(this@TopDoctorsActivity, Observer
{ list ->
              val filteredList = if (selectedCategory.isNullOrEmpty()) {
                  list
              } else {
                  list.filter {
                      it.Special.contains(selectedCategory!!, ignoreCase =
true)
                  }.toMutableList()
              }
              binding.viewTopDoctorList.layoutManager =
LinearLayoutManager(
                  this@TopDoctorsActivity,
                  LinearLayoutManager.VERTICAL,
                  false
              )
              binding.viewTopDoctorList.adapter =
TopDoctorAdapter2(filteredList)
              binding.progressBarTopDoctor.visibility = View.GONE
          })
        viewModel.loadDoctors()
        backBtn.setOnClickListener { finish() }
        }
    }
}
```