

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій**

ПОГОДЖЕНО
Декан факультету

_____ **Ігор БОЛБОТ**
(підпис)

«___» _____ 2026 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних наук

_____ **Белла ГОЛУБ**
(підпис)

«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему
Мобільний програмний застосунок для споживачів послуг страхової
компанії
Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми
д.е.н., професор _____

Руденський Р.А.

Керівник бакалаврської кваліфікаційної роботи

ст викладач
(науковий ступінь та вчене звання)

(підпис)

Панкратьєв В.О.
(ПІБ)

Виконав

(підпис)

Голота В.С
(ПІБ студента)

КИЇВ – 2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій**

ЗАТВЕРДЖУЮ

Завідувач кафедри

Комп'ютерних наук

_____ Голуб Б.Л.

“06” січня 2026 р.

**З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту**

_____ Голоті Владиславу Сергійовичу

(прізвище, ім'я, по батькові)

Спеціальність 122 – «Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи **Мобільний програмний застосунок для споживачів послуг страхової компанії** затверджена наказом від «6» січня 2026 р. № 46 «С»

Термін подання завершеної роботи на кафедру 22.05.2026

(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:

опис програмного забезпечення

Перелік питань, які потрібно розробити:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування програмного застосунок.
4. Висновки.

Дата видачі завдання “06” січня 2026 р.

Керівник бакалаврської кваліфікаційної роботи

ст. викладач

(науковий ступінь та вчене звання)

Завдання прийняв до виконання

(підпис)

(підпис)

Панкратьєв В.О.

(ПІБ)

Голота В.С.

(ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	4
ВСТУП	5
1 АНАЛІЗ РИНКУ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ СТРАХОВИХ КОМПАНІЙ	7
1.1 Особливості ринку мобільних додатків для страхових компаній та аналіз основних функціональних вимог до додатків такого типу	7
1.2 Аналіз сучасних технологій, які можуть бути використані для розробки мобільних додатків	11
1.3 Постановка задачі	14
2 ПРОЄКТУВАННЯ ДОДАТКУ	16
2.1 Розробка дизайну інтерфейсу додатку	16
2.2 Опис функціональних вимог до інтерфейсу додатку	20
2.3 Характеристика технологій, які будуть використані в додатку	22
3 ЗАСОБИ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ	34
3.1 Аутентифікація та авторизація користувачів	34
3.2 Налаштування мобільного додатку	41
3.4. Створення бази даних та структури даних застосунку	44
4. ОПИС РЕАЛІЗАЦІЇ	
4.1 Реалізація страхових продуктів	47
4.2 Демонстраційна версія застосунку та порядок відкриття проєкту в Android Studio	60
ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	67
ДОДАТОК А	70
ДОДАТОК Б	102
ДОДАТОК В	104
ДОДАТОК Д	105

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API - Application Programming Interface, програмний інтерфейс для взаємодії між програмними компонентами.

CSS - Cascading Style Sheets, мова опису стилів інтерфейсу.

GDPR - General Data Protection Regulation, загальний регламент захисту даних Європейського Союзу.

HTML - HyperText Markup Language, мова гіпертекстової розмітки.

JavaScript - мова програмування, що використовується для реалізації інтерактивної логіки інтерфейсу.

UI - User Interface, користувацький інтерфейс.

UX - User Experience, досвід користувача під час взаємодії із застосунком.

КАСКО - добровільне страхування транспортного засобу.

МТСБУ - Моторне (транспортне) страхове бюро України.

ОСЦПВ - обов'язкове страхування цивільно-правової відповідальності власників наземних транспортних засобів.

ВСТУП

Актуальність теми зумовлена активною цифровізацією страхового ринку та зростанням потреби клієнтів у швидкому доступі до страхових послуг без обов'язкового звернення до офісу компанії. Мобільний застосунок дає змогу автоматизувати частину взаємодії між клієнтом і страховою компанією, спростити оформлення страхових продуктів, організувати комунікацію через чат і забезпечити доступ до інформації про послуги в зручному цифровому форматі.

Мета роботи - розробити мобільний програмний застосунок для страхової компанії, який забезпечує клієнтам доступ до основних страхових продуктів, підтримує аутентифікацію користувачів, роботу з профілем, вибір страхових послуг та комунікацію з працівниками компанії.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати предметну область і сучасний стан використання мобільних застосунків у страхових компаніях.
- визначити функціональні вимоги до мобільного застосунку страхової компанії.
- обґрунтувати вибір технологій і засобів розробки програмного рішення.
- спроектувати структуру, інтерфейс і основні функціональні модулі застосунку.
- реалізувати основні сторінки та функції застосунку.
- провести тестування розробленого прототипу та визначити напрями його подальшого вдосконалення.

Об'єкт дослідження - процес розробки мобільного програмного застосунку для страхової компанії.

Предмет дослідження - технології, методи та інструменти проєктування і реалізації мобільного застосунку для надання страхових послуг.

У роботі використано методи аналізу предметної області, порівняльного аналізу існуючих цифрових рішень, моделювання функціональних вимог, прототипування інтерфейсу, програмної реалізації та функціонального тестування. Для реалізації демонстраційного прототипу застосовано Android Studio, Java, Android SDK та Gradle.

Апробацію програмного застосунку здійснено шляхом перевірки основних сценаріїв використання: авторизації користувача, роботи з особистим кабінетом, вибору страхових продуктів, заповнення форм і використання чату. Під час навчання було зроблено доповідь на конференції в НаУКМА «Менеджмент та маркетинг, як засоби розвитку бізнесу» на секції «Цифрові інструменти ефективного функціонування бізнесу» (<https://surl.li/rxygip>). Сертифікат доданов Додаток В.

Пояснювальна записка (70 сторінок основного тексту) складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проаналізовано предметну область і технологічні підходи до розробки страхових мобільних застосунків. У другому розділі розглянуто проєктування інтерфейсу та функціональні вимоги. У третьому розділі опис засобів розробки реалізацію програмного застосунку та порядок відкриття Android-проєкту. У четвертому розділі наведено тестування, аналіз результатів і можливі напрями вдосконалення.

1 АНАЛІЗ РИНКУ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ СТРАХОВИХ КОМПАНІЙ

1.1 Особливості ринку мобільних додатків для страхових компаній та аналіз основних функціональних вимог до додатків такого типу

Ринок мобільних додатків в Україні розвивається під впливом глобальних цифрових тенденцій. Для страхового сектору характерними є зростання ролі електронних полісів, дистанційного оформлення договорів, онлайн-комунікації з клієнтами та інтеграції з платіжними й державними сервісами. Мобільні додатки фокусуються на управлінні полісами, оформленні страхових продуктів, обробці заяв, комунікації через чат та персоналізації послуг [1-7].

Ключовими гравцями та лідерами ринку є: GEICO, State Farm, UnitedHealthcare (США), Lemonade, Salesforce. У Європі сильні позиції у традиційних insurers та insurtech-стартапів; в Україні — hotline.finance та EWA для порівняння полісів. На глобальному ринку мобільних додатків для страхових компаній домінують великі гравці з США та insurtech-стартапи. У Європі та Україні лідерство поділяють традиційні insurers та локальні агрегатори. Глобальні лідери GEICO, State Farm та Progressive контролюють понад 50 % завантажень у США завдяки швидкій обробці претензій та телематиці. Lemonade та Hippo вирізняються AI-персоналізацією, досягаючи 2-3 млн активних користувачів кожна. (табл. 1.1)

В ЄС AXA, Allianz та Aviva лідирують з додатками для багатомовного управління полісами, з ринковою часткою 30-40 % у Західній Європі. Insurtech як BIMA фокусуються на мікрострахуванні в Східній Європі.

Ринок мобільних додатків для страхових компаній в Україні активно зростає завдяки цифровізації ОСЦПВ та переходу на е-поліси у 2026 році.

Сегмент займає 2-3 % від загального ринку додатків (1,5 млрд завантажень щорічно), але демонструє +40 % зростання YoY через обов'язкову електроніку.

Лідери генерують 1,5+ млн інсталяцій: Hotline.finance — №1 (1,9К завантажень/тиждень), Мій ТАС — 2,4К/тиждень (Q4 2025). Ринок стимулюють МТСБУ API, Дія-інтеграція та воєнні реалії (швидкі претензії).

Агрегатори домінують на 70 % ринку: Hotline.finance (20+ СК), Polis.ua, EWA — фокус на порівнянні ОСЦПВ/Зелена Картка. Традиційні СК: ТАС, ARX (MY ARX), UNIQA — власні кабінети для лояльних клієнтів з офлайн-полісами та трекінгом виплат.

Повний перехід на е-поліси (папір — виняток), AI-чатботи, фотофіксація збитків, оплата штрафів ПДР. Android — 85 % ринку; нативна розробка (Kotlin/Swift) для безпеки. Прогноз 2026: super-apps з медстрахом та UBI-тарифами.

Таблиця 1.1

Тренди та особливості використання мобільних додатків

Гравець	Частка (%)	Завантаження/тиждень	Фішки
Hotline.finance	40-50	1,9К	20+ СК, штрафи ПДР
Мій ТАС	20-25	2,4К	Трекінг претензій
MY ARX	10-15	1К+	Офлайн-поліси
UNIQA/Polis.ua	10-15	800+	Е-QR, МТСБУ

Лідери українського страхового ринку активно розвивають власні мобільні додатки для оформлення полісів, обробки претензій та управління контрактами.

Топ-лідери в Україні мають наступні додатки:

- ARX: MY ARX — показує договори (КАСКО, ОСЦПВ, медстрах), чат з компанією, офлайн-доступ;
- ТАС: Мій ТАС — лідер за завантаженнями (2,4К на тиждень у Q4 2025), для полісів та виплат;
- UNIQA: UNIQA Страхівка — порівняння, покупка ОСЦПВ/КАСКО, зберігання е-полісів;
- PZU: PZU Ukraine — онлайн-оформлення, трекінг претензій;
- Оранта: Оранта Онлайн — базовий кабінет для клієнтів.

Агрегатори домінують у завантаженнях. Так Hotline.finance є Страховка — №1 за популярністю (1,9К завантажень/тиждень), порівнює 20+ компаній (ARX, UNIQA, PZU). EWA — аналог для ОСЦПВ.

Hotline.finance вирізняється як агрегатор страхових продуктів від багатьох компаній, а не власник полісів, що робить його платформою для порівняння та вибору.

Hotline.finance пропонує прямі ціни без націнок від 20+ страховиків (ARX, UNIQA, TAC), бонуси, розстрочку та акції, тоді як додатки страховиків обмежені їхніми продуктами. Агрегатор автоматизує оформлення через API, забезпечуючи швидкість без агентів, на відміну від фокусу страхових додатків на внутрішніх претензіях та кабінетах клієнтів.

Додаток Hotline зберігає всі поліси в одному місці, нагадує про пролонгацію, дозволяє платити штрафи ПДР — функцій, яких немає в типових додатках типу MY ARX чи Мій ТАС.

Hotline.finance інтегрується з низкою українських страхових компаній і пропонує користувачам інструменти для порівняння та онлайн-оформлення страхових продуктів [8].

Таблиця 1.2

Порівняльна таблиця страхових продуктів

Аспект	Hotline.finance	Додатки страховиків (ARX, TAC)
Продукти	20+ компаній, порівняння	Тільки власні
Ціни/знижки	Без націнок, бонуси до 40%	Власні тарифи
Додаткові сервіси	Штрафи ПДР, розстрочка	Претензії, кабінет
UX	Єдиний кабінет для всіх	Компанія-специфічний

Ключові страхові компанії включають ARX, ЕКТА, GRAWE Україна, INGO, Salamandra, TAS Life та UNIQA. Серед найпопулярніших за продажами — Українська страхова група (USG), PZU Україна, VUSO та UNIVERSALNA, що охоплюють ОСЦПВ, КАСКО, Зелена Картка та інші продукти. Список динамічно оновлюється: hotline.finance додає нових партнерів і може

призупиняти співпрацю, фокусуючись на надійних гравцях з державними ліцензіями.

Мобільні додатки для страхових компаній в Україні характеризуються фокусом на зручності, автоматизації процесів та інтеграцією з державними сервісами. Вони спрощують оформлення полісів ОСЦПВ/КАСКО, обробку претензій та управління контрактами, часто з офлайн-доступом та чатами.

Типовий функціонал включає особистий кабінет з деталями договорів, онлайн-замовлення полісів, каталог послуг, чат з консультантами, геолокацію офісів та інструменти фотофіксації для претензій. Додатки як MY ARX чи Мій ТАС підтримують офлайн-перегляд полісів, нагадування про пролонгацію, оплату штрафів ПДР та інтеграцію з месенджерами.

Багато додатків використовують API для швидкої обробки, з акцентом на безпеку даних (GDPR-сумісність у Європі) та AI для персоналізації рекомендацій.

Такі додатки підвищують лояльність клієнтів завдяки мобільності — доступ 24/7 без візиту до офісу. В Україні вони адаптовані до воєнних реалій: швидка фотофіксація збитків, сімейні профілі для кількох полісів та пошук клінік для медстраху (табл. 1.3).

Таблиця 1.3

Порівняння типових характеристик

Характеристика	Опис	Приклади (ARX, ТАС)
UX/UI	Інтуїтивний, офлайн-доступ	MY ARX: родинні профілі
Безпека	Шифрування, біометрія	UNIQA: е-поліси
Інтеграції	API банків, МТСБУ, ПДР	Hotline: 20+ СК
Інновації	AI-чат, фото-огляд авто	ТАС: статус претензій у реальному часі

Мобільні додатки для страхових компаній використовують широкий спектр технологій для забезпечення безпеки, швидкості та персоналізації.

1.2 Аналіз сучасних технологій, які можуть бути використані для розробки мобільних додатків

Для страхових мобільних додатків технологічний стек обирається з урахуванням безпеки чутливих даних (поліси, платежі), масштабованості та кросплатформенності, щоб мінімізувати витрати та прискорити розробку. Для страхових мобільних додатків рекомендуються технології, що поєднують високу безпеку, кросплатформенність та інтеграцію з fintech-сервісами, на основі аналізу сучасних практик (табл. 1.4).

Таблиця 1.4

Узагальнення технологій, які можуть бути використані для мобільних додатків [9-12]

<i>Шар</i>	<i>Технології</i>	<i>Застосування</i>
Фронтенд	Flutter, React Native	Поліси, фотофіксація
Бекенд	Node.js, PostgreSQL	Платежі, аналітика
Безпека	OAuth, Stripe	KYC, виплати

Перша технологія - фронтенд: кросплатформенні фреймворки. Кросплатформенні фреймворки є важливими для ефективної розробки мобільних додатків, оскільки дозволяють використовувати спільну кодову базу для різних платформ, скорочують час розробки та спрощують підтримку інтерфейсу [3]. Flutter та React Native належать до найпоширеніших рішень у цій групі технологій.

React Native (JavaScript/TypeScript) використовує нативні компоненти платформ через JavaScript-мост (JSI у нових версіях покращує швидкість), підходить для команд з веб-досвідом; Expo спрощує прототипи, але потребує нативних модулів для біометрії чи камери. Велика екосистема (React Navigation, Redux), але нижча стабільність на старих пристроях та вищий апетит до пам'яті/батареї Hot Reload миттєво оновлює зміни без перезапуску, виджети

Material/Cupertino забезпечують нативний дизайн; ідеально для кастомних інтерфейсів страхових форм з фотофіксацією.

Результати порівняння технологій розробки мобільних додатків для страхування наведено в табл. 1.5.

Таблиця 1.5

Порівняння технологій розробки мобільних додатків для страхових компаній [13-14]

Аспект	Flutter	React Native
Продуктивність	60 FPS, нативний рендер	58 FPS, JS-мост
Швидкість розробки	Hot Reload, виджети	Ехро, JS-екосистема
UI-консистентність	Висока (Skia)	Залежить від платформи
Екосистема	Зростаюча, Google	Величезна, Meta

Можна зробити висовок, що технологія Flutter краща для UI-інтенсивних додатків (форми полісів, дашборди ризиків) завдяки консистентності та швидкості; React Native — для швидкого MVP з інтеграціями (Firebase, Stripe).

Наступна технологія Бекенд — це серверна частина програмного забезпечення або веб-додатку, яка відповідає за обробку логіки, зберігання даних, аутентифікацію користувачів, взаємодію з базами даних та іншими сервісами. Він працює "за сценою" і не є безпосередньо видимим для користувача, але забезпечує функціональність, яку користувачі бачать на фронтенді (інтерфейсі).

API (Application Programming Interface) - це набір правил і протоколів, які дозволяють різним програмним компонентам взаємодіяти між собою. У контексті бекенду API часто використовується для зв'язку між сервером і клієнтською частиною, а також для інтеграції з іншими сервісами. API визначає, як запити мають бути сформовані, які дані можна отримати або відправити, і у якому форматі. Таким чином можна зробити висновок, що API дозволяє створювати модульні, масштабовані системи. Приклади: REST API, GraphQL.

Хмарні технології (Cloud computing) - це модель надання обчислювальних ресурсів (сервери, сховища, бази даних, мережі, програмне забезпечення) через інтернет. Хмари дозволяють розгортати бекенд-сервіси без необхідності утримувати власну фізичну інфраструктуру. До переваг хмарних технологій відносять: масштабованість, гнучкість, зниження витрат на інфраструктуру. Популярні провайдери: Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure. Хмарні сервіси часто використовують для розміщення API, баз даних, серверів додатків. На практиці все реалізується наступним чином:

- бекенд часто реалізується як набір API, які працюють на сервері або в хмарі;
- хмарні платформи надають інструменти для розробки, розгортання та масштабування бекенд-сервісів і API;
- використання хмарних сервісів дозволяє швидко адаптуватися до змін навантаження, забезпечувати високу доступність і безпеку.

Вимоги безпеки в страхових мобільних додатках є критичними через обробку чутливих даних (персональні дані, платежі, медична історія), з обов'язковим дотриманням GDPR, PCI DSS та ISO/IEC 27001.

Багатофакторна аутентифікація (MFA: OTP, біометрія Face ID/Touch ID), JWT/OAuth 2.0 токени з коротким терміном дії; заборона слабких паролів та примусова перевірка KYC для верифікації клієнтів. Шифрування даних -

AES-256 для зберігання (Keychain на iOS, Keystore на Android), TLS 1.3 для передачі; токенизація карток платежів; заборона незашифрованих локальних баз (Realm з шифруванням).

Захист від атак- перевірка на SQL-ін'єкції, XSS, CSRF; root/jailbreak-детекція; б- пезпечне кодування (ProGuard/R8 для обфускації); обмеження мережевих запитів (CORS, certificate pinning).

Тестування та моніторинг - регулярний pentest (OWASP Mobile Top 10), сканування вразливостей (MobSF); моніторинг з Crashlytics/Sentry; аудит compliance перед релізом.

Тестування на проникнення (pentest) для мобільних додатків є ключовим етапом перевірки безпеки, особливо для страхових, де обробляються чутливі дані; воно моделює реальні атаки хакерів за стандартами OWASP Mobile Top 10 та MSTG.

Аналіз технологій для мобільних додатків страхових компаній показує, що оптимальний стек поєднує кросплатформенність, безпеку та масштабованість для обробки полісів, платежів і заяв.

1.3 Постановка задачі

Після проведення аналізу ринку мобільних додатків для страхових компаній та виявлення основних функціональних вимог до таких додатків, була поставлена задача розробити мобільний додаток для страхової компанії, який буде надійним та зручним інструментом для її клієнтів. Головною метою додатку є забезпечення максимальної зручності для споживача страхових послуг, можливості щодо швидкого отримання пакету страхових послуг, а також забезпечення максимальної безпеки та захисту персональних даних користувачів.

Основні завдання, які стоять перед розробниками додатку, включають:

- розробка інтуїтивно зрозумілого та простого інтерфейсу додатку, який дозволяє користувачам швидко та легко здійснювати операції зі страхування;
- забезпечення максимальної безпеки даних користувачів та конфіденційності;
- інтеграція додатку з системами страхової компанії та забезпечення автоматичної обробки заявок на страхування;
- підтримка різноманітних видів страхування та можливість швидкої оплати страхових послуг через додаток;
- забезпечення роботи додатку на різних платформах та мобільних пристроях.

Висновки до розділу

У ході дослідження було проаналізовано діяльність провідних міжнародних та українських страхових компаній і агрегаторів страхових послуг, зокрема ARX, ТАС, UNIQA, PZU, Hotline.finance та інших. Встановлено, що український ринок страхових мобільних додатків активно розвивається завдяки впровадженню електронного страхування, інтеграції з МТСБУ та державними цифровими сервісами. Особливу роль на ринку відіграють агрегатори страхових продуктів, які забезпечують можливість порівняння послуг різних страхових компаній в межах єдиної платформи.

Також було визначено основні функціональні вимоги до мобільних додатків для страхових компаній, серед яких: наявність особистого кабінету користувача, підтримка електронних полісів, онлайн-оформлення страхових послуг, чат-комунікація, інтеграція з платіжними сервісами, фотофіксація страхових випадків, геолокаційні сервіси та засоби захисту персональних даних. Встановлено, що важливими характеристиками сучасних страхових застосунків є інтуїтивний інтерфейс, висока швидкодія, мобільність, безпечність та можливість роботи в режимі реального часу.

У межах дослідження сучасних технологій розробки мобільних додатків було проаналізовано кросплатформенні фреймворки Flutter та React Native, серверні технології, API, хмарні сервіси та засоби забезпечення інформаційної безпеки. Визначено, що використання сучасних технологічних рішень дозволяє забезпечити масштабованість системи, зручність підтримки програмного продукту та високий рівень захисту даних користувачів.

За результатами проведеного аналізу було сформульовано постановку задачі, яка полягає у розробці мобільного додатку для страхової компанії, що забезпечуватиме зручне та безпечне отримання страхових послуг, підтримку взаємодії користувачів із системою, автоматизацію основних страхових процесів та адаптацію до сучасних вимог цифрового страхового середовища.

2 ПРОЄКТУВАННЯ ДОДАТКУ

2.1 Розробка дизайну інтерфейсу додатку

Проектування додатка є однією з ключових стадій розробки мобільного додатка для страхової компанії. На цій стадії визначається архітектура додатку, розроблюються дизайн та функціональність, створюється прототип додатку та встановлюються технічні вимоги до розробки.

Одним із перших кроків у проектуванні додатка є створення прототипу, який можна протестувати з потенційними користувачами та отримати відгуки щодо його функціоналу та інтерфейсу. Важливо також визначити технічні вимоги до додатку, які повинні враховувати особливості платформи, на якій буде запуснений додаток, а також вимоги до безпеки та захисту персональних даних.

Наступним кроком у проектуванні додатка є розробка його архітектури, яка визначає, які компоненти та модулі будуть входити до додатку та як вони будуть взаємодіяти між собою. Наприклад, до компонентів додатку можуть входити модулі для авторизації користувачів, обробки платежів, оформлення заявок на страхування тощо.

Окрему увагу слід приділити дизайну додатку. Необхідно розробити інтерфейс, який буде зручним для користувача та надихатиме довіру. Дизайн має бути відповідним бренду страхової компанії та відповідати найсучаснішим тенденціям у дизайні мобільних додатків.

Нарешті, після розробки прототипу, визначення технічних вимог та розробки архітектури додатку, можна приступати до розробки його функціональності.

Розробка дизайну інтерфейсу додатку є важливим етапом проектування, оскільки відповідно до статистики, перші 15 секунд використання додатку визначають, чи залишиться користувач на сайті або залишить його. Отже,

забезпечення привабливого, зручного та інтуїтивно зрозумілого інтерфейсу - це ключовий фактор для забезпечення успіху додатку.

Перед початком розробки дизайну інтерфейсу необхідно визначити ключові функції, що будуть включені в додаток, та створити скетчі інтерфейсу. Після цього можна перейти до створення прототипу, який дозволяє розробникам отримати повне уявлення про те, як буде виглядати кінцевий продукт.

Під час проєктування дизайну інтерфейсу додатка необхідно дотримуватися основних принципів дизайну, таких як простота, наочність, зручність користування та логічність. Крім того, варто враховувати технічні особливості різних платформ (iOS, Android), щоб забезпечити оптимальний дизайн для кожної з них.

Для розробки дизайну інтерфейсу додатку можна використовувати спеціальні програмні засоби, такі як Adobe Photoshop, Sketch або Figma. У процесі розробки дизайну потрібно враховувати вимог до інтерфейсу, які визначені на етапі постановки задачі, а також забезпечувати зручне розташування елементів та привабливий зовнішній вигляд (рис.2.1.).

Після розробки дизайну інтерфейсу, потрібно здійснити його аналіз, перевірити на зручність користування та відповідність вимогам. Після визначення функціональної складової додатку та його архітектури, наступним кроком було розроблення дизайну інтерфейсу додатку. Дизайн було розроблено з урахуванням вимог користувачів, щоб зробити його максимально зручним та зрозумілим.

Для мобільних додатків страхових компаній доцільно використовувати кольори, які асоціюються з надійністю, безпекою, стабільністю та довірою. У сфері страхування дизайн має викликати у користувача відчуття впевненості та професійності. Основним фоновим кольором було вибрано блакитний, який є одним з найбільш популярних кольорів серед страхових компаній, оскільки він символізує спокій та стабільність. Також було використано білий колір для основного тексту та шрифти з хорошою читабельністю.

ГОЛОВНИЙ ЕКРАН (Dashboard)

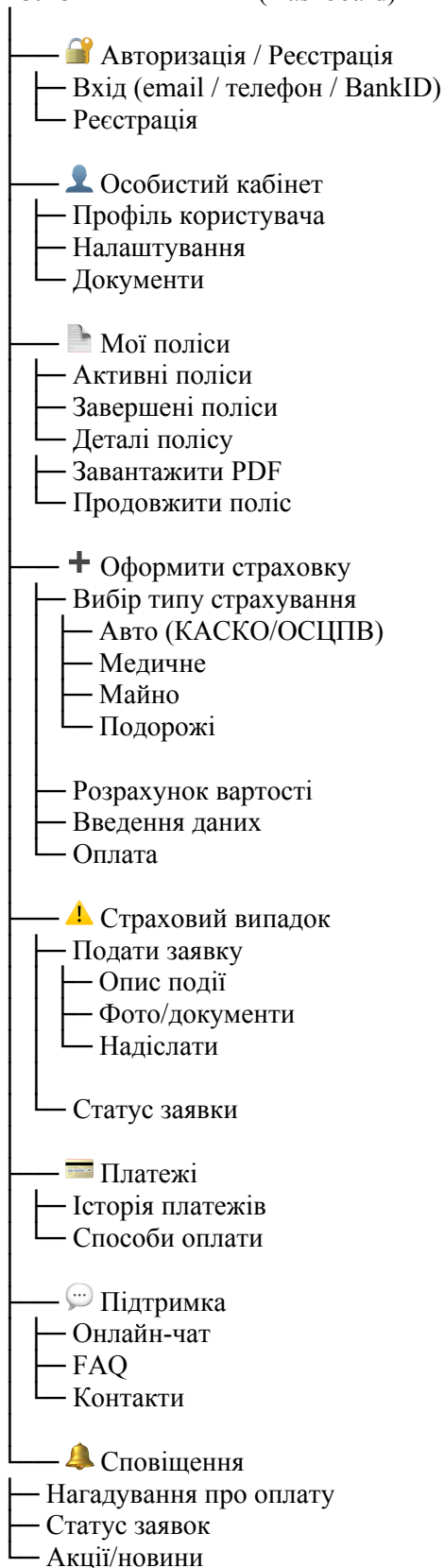


Рис.2.1 UX-схема структури мобільного додатку для споживачів страхових послуг

Головний екран додатку було спроектовано таким чином, щоб користувач з першого погляду міг знайти потрібну інформацію та перейти до необхідної функції. Він складався з трьох основних блоків: меню, персональної інформації та швидкого доступу до послуг. Меню містило основні пункти, такі як "Страхування", "Заявки", "Профіль", "Налаштування", "Допомога", "Вихід" (рис.2.2.)



Рис.2.2 UX-схема структури мобільного додатку для споживачів страхових послуг з елементами Меню

У розділі "Профіль" користувач міг переглянути свою персональну інформацію та історію взаємодії з додатком. Розділ "Швидкий доступ до послуг" містив кнопки швидкого доступу до найбільш популярних послуг страхової компанії, таких як оформлення договору страхування, подання заявки на відшкодування збитків та ін.

2.2 Опис функціональних вимог до інтерфейсу додатку

Функціональні вимоги до інтерфейсу мобільного додатку визначають сукупність можливостей системи, що забезпечують ефективну взаємодію користувача зі страховими послугами в цифровому середовищі. Вони формують основу розробки UX/UI-дизайну та визначають логіку роботи додатку.

Інтерфейс має відповідати принципам зручності використання (usability), доступності (accessibility) та орієнтації на користувача (user-centered design), забезпечуючи швидкий доступ до основних функцій. Узагальнення вимог узагальнено в табл. 2.1.

При аналізі функціональних вимог до інтерфейсу мобільного додатку для страхової компанії було враховано наступні принципи:

- орієнтація на користувача (user-centered design) – інтерфейс проєктується з урахуванням потреб, очікувань і поведінкових моделей користувачів;
- зручність використання (usability) – забезпечення простоти навігації, зрозумілості елементів інтерфейсу та мінімізації кількості дій для досягнення результату;
- функціональна достатність – наявність повного набору функцій для отримання страхових послуг в онлайн-форматі;
- консистентність інтерфейсу – уніфікованість дизайну, повторюваність елементів і єдина логіка взаємодії;
- доступність та інклюзивність (accessibility) – адаптація інтерфейсу для різних категорій користувачів, включаючи осіб з особливими потребами;

- безпека та конфіденційність – захист персональних даних і забезпечення надійної автентифікації;
- адаптивність – коректне відображення інтерфейсу на різних мобільних пристроях;
- швидкодія та ефективність – мінімізація часу виконання операцій і швидке завантаження екранів;
- персоналізація – можливість налаштування інтерфейсу відповідно до індивідуальних потреб користувача.

Таблиця 2.1

Відповідність функціональних вимог, функцій та UI-елементів інтерфейсу мобільного додатку

№	Функціональна вимога	Функція системи	UI-елемент
1	Забезпечення доступу користувача	Реєстрація та авторизація	Екран входу, поля вводу, кнопка «Увійти», біометрія
2	Простота навігації	Перехід між розділами	Нижнє меню (bottom navigation), іконки меню
3	Доступ до страхових продуктів	Перегляд видів страхування	Каталог продуктів, картки страхових послуг
4	Розрахунок вартості	Онлайн-калькуляція полісу	Форма введення даних, кнопка «Розрахувати»
5	Оформлення страхового полісу	Створення договору	Покрокова форма (wizard), кнопка «Оформити»
6	Управління полісами	Перегляд і продовження	Список полісів, кнопки «Деталі», «Продовжити»
7	Подання страхового випадку	Створення заявки	Форма заявки, кнопка «Додати файл»
8	Завантаження документів	Додавання фото/файлів	Кнопка «Upload», галерея/камера
9	Відстеження статусу	Перегляд стану заявки	Статус-бар, timeline, повідомлення
10	Проведення оплат	Онлайн-оплата	Кнопка «Оплатити», платіжна форма
11	Управління профілем	Редагування даних	Екран профілю, поля редагування
12	Налаштування системи	Персоналізація	Toggle-перемикачі, меню налаштувань
13	Отримання підтримки	Зв'язок із службою	Онлайн-чат, кнопка «Допомога»
14	Інформування користувача	Push-сповіщення	Pop-up повідомлення, notification center
15	Забезпечення безпеки	Захист даних	PIN/Face ID, підтвердження дій

2.3 Характеристика технологій, які будуть використані в додатку

Розробка мобільних додатків для Android неможлива без використання сучасних мов програмування та ефективних технологій. Тривалий час основною мовою для створення Android-додатків була Java, яка зарекомендувала себе як надійна, стабільна та універсальна технологія. Kotlin також широко використовується в Android-розробці та офіційно підтримується Google. У межах демонстраційного проєкту обрано Java, оскільки вона є зрозумілою для навчальної реалізації, не потребує додаткового Kotlin-плагіна та дозволяє створити простий Android Studio-проєкт з базовими екранами застосунку.

- сучасна об'єктно-орієнтована та функціональна мова — підтримує як ООП, так і функціональне програмування.
- повна сумісність з Java — можна використовувати Java-бібліотеки без додаткових налаштувань.
- менше коду — значно зменшує кількість шаблонного коду.
- безпечна робота з null — знижує кількість помилок типу NullPointerException.
- підтримка корутин (Coroutines) — спрощує асинхронне програмування.

Наступний інструмент Android Studio — це офіційне інтегроване середовище розробки (IDE) для створення мобільних додатків на платформі Android. Воно розроблене компанією Google на основі IntelliJ IDEA та забезпечує повний набір інструментів для проєктування, програмування, тестування й оптимізації мобільних застосунків.

У процесі розробки мобільного додатку для страхових послуг Android Studio є одним із найбільш ефективних рішень, оскільки дозволяє створювати функціональні, безпечні та зручні у використанні застосунки для клієнтів страхових компаній. Такий додаток може включати можливості оформлення страхових полісів, перегляду договорів, розрахунку вартості страхування, подання заяв на страхові випадки, онлайн-консультацій та оплати послуг. Основні характеристики Android Studio. При вивченні Android Studio виходили із наступних характеристик:

1) Android Studio надає сучасний редактор коду з підсвічуванням синтаксису, автоматичним доповненням коду, перевіркою помилок у реальному

часі та рефакторингом. Це значно спрощує написання програмного коду мовами Java та Kotlin;

2) середовище повністю підтримує обидві основні мови Android-розробки — Java та Kotlin. Практиками-розробниками страхових мобільних додатків перевага часто надається Kotlin завдяки більшій безпеці коду та зручності підтримки;

3) вбудований Layout Editor дозволяє візуально створювати інтерфейс користувача за допомогою XML або Jetpack Compose. Для страхового додатку це важливо при розробці зручних форм реєстрації, калькуляторів страхових внесків, кабінету клієнта та платіжних модулів;

4) Android Studio має вбудований емулятор, який дозволяє тестувати додаток на різних моделях смартфонів без використання фізичних пристроїв. Це особливо корисно для перевірки коректної роботи страхового застосунку на різних версіях Android;

5) для страхових додатків важливо забезпечити обмін даними з сервером, CRM-системами та базами даних. Android Studio підтримує роботу з REST API, Firebase, Room Database, SQLite та іншими технологіями для збереження й обробки інформації.

6) важливим є реалізація механізмів захисту, оскільки страхові додатки працюють із персональними даними клієнтів: авторизацію, шифрування даних, біометричний вхід, захист платіжної інформації та безпечну передачу даних через HTTPS;

7) IDE містить профайлер продуктивності, логування, систему тестування та аналізу помилок, що дозволяє забезпечити стабільну роботу страхового додатку та зменшити ризик збоїв.

Таким чином, Android Studio є потужним і надійним інструментом для створення мобільного додатку у сфері страхових послуг. Завдяки широкому функціоналу, підтримці сучасних мов програмування, засобам безпеки та інтеграції з серверними системами це середовище дозволяє розробляти якісні, безпечні та зручні застосунки для страхових компаній і їхніх клієнтів.

Використання Android Studio сприяє підвищенню ефективності розробки та створенню конкурентоспроможного програмного продукту.

Наступним інструментом, який планується використати, є Firebase — це хмарна платформа для розробки мобільних і веб-додатків, створена компанією Google. Вона надає широкий набір інструментів для зберігання даних, автентифікації користувачів, аналітики, тестування, надсилання повідомлень та інтеграції серверної логіки без необхідності створення власної складної серверної інфраструктури.

Firebase є потужним інструментом для розробки мобільних додатків у сфері страхових послуг. Платформа дозволяє швидко створювати функціональні, безпечні та масштабовані застосунки без значних витрат на серверну інфраструктуру. Для страхових компаній це особливо важливо, оскільки забезпечується оперативна робота з клієнтськими даними, автоматизація процесів та підвищення якості обслуговування користувачів. Використання Firebase значно спрощує розробку сучасного мобільного страхового додатку та підвищує його ефективність.

В якості ефективного інструменту для локального зберігання даних у мобільних додатках пропонується SQLite. SQLite — це легка вбудована реляційна система керування базами даних (СУБД), яка широко використовується у мобільній розробці для локального зберігання інформації без необхідності окремого серверного підключення. Вона є стандартним компонентом платформи Android і часто застосовується для створення мобільних додатків, що потребують швидкого, надійного та автономного доступу до даних.

SQLite особливо корисна у мобільних застосунках, де необхідно зберігати дані користувача без постійного доступу до Інтернету. Наприклад, у страхових мобільних додатках вона може використовуватися для збереження профілю клієнта, історії операцій, даних страхових полісів, кешування документів та тимчасового зберігання інформації для подальшої синхронізації з сервером.

Для роботи з мережевими запитами в мобільних додатках на платформі Android пропонується Retrofit. Вона розроблена компанією Square та використовується для взаємодії мобільного застосунку з сервером через REST API. Retrofit значно спрощує обмін даними між клієнтською частиною програми та серверною системою.

У мобільних додатках для страхових компаній Retrofit відіграє важливу роль, оскільки дозволяє реалізувати стабільну передачу даних між користувачем і сервером. За допомогою цієї бібліотеки можна отримувати інформацію про страхові поліси, передавати заявки на страхові випадки, виконувати розрахунок вартості страхування, проводити авторизацію користувачів та синхронізувати дані з центральною базою страхової компанії. Важливим є врахування переваги Retrofit для страхових мобільних додатків та недрліків. Серед переваг фахівці виділяють наступні:

- спрощує роботу з сервером;
- зменшує кількість шаблонного коду;
- забезпечує швидку інтеграцію з REST API;
- підтримує безпечну передачу даних;
- зручний для масштабування великих проєктів;
- добре інтегрується з Android Studio;
- підтримує сучасні архітектури Android-додатків.

В той же час не можна нехтувати рядом недоліків Retrofit:

- потребує правильно налаштованого серверного API;
- додаткова залежність від сторонніх бібліотек;
- складніше налаштування для початківців;
- потребує контролю помилок мережі та винятків;
- не підходить для повністю офлайн-додатків без серверної взаємодії.

Наступним інструментом є Glide — це популярна бібліотека для завантаження, обробки та кешування зображень у мобільних додатках на платформі Android. Вона розроблена для ефективної роботи з графічним

контентом, оптимізації використання пам'яті та забезпечення швидкого відображення зображень у застосунку.

У мобільних додатках для страхових компаній Glide відіграє важливу роль, оскільки дозволяє працювати з фотографіями документів, зображеннями страхових випадків, фото пошкоджень автомобілів, профільними фото клієнтів, скан-копіями договорів та іншими візуальними матеріалами, які часто використовуються у страхових сервісах.

Для розробки мобільного додатку для iOS доцільно використовувати наступні технології та інструменти.

Swift та Objective-C — це основні мови програмування для розробки мобільних додатків на платформі iOS, яка використовується на пристроях компанії Apple, таких як iPhone та iPad. Обидві мови застосовуються для створення нативних мобільних застосунків, проте Swift є сучаснішою мовою, а Objective-C — класичною технологією, яка тривалий час була основною для iOS-розробки.

Objective-C залишається важливою мовою для підтримки старих iOS-проектів, однак Swift є сучасним стандартом для створення нових мобільних додатків на платформі Apple. Завдяки простоті, безпеці та високій продуктивності Swift значно спрощує процес розробки та дозволяє створювати якісні конкурентоспроможні мобільні застосунки. Саме тому більшість нових iOS-проектів сьогодні реалізуються саме на Swift.

Xcode — це інтегроване середовище розробки (IDE) від Apple для розробки додатків для iOS. Воно надає широкі можливості для програмістів, зокрема, можливість швидко створювати і тестувати додатки з допомогою інструментів розробки, таких як візуальний редактор інтерфейсу, дебаггер і інші.

Для створення графічного інтерфейсу користувача (UI) в мобільних додатках на платформі iOS UIKit — це основний фреймворк. Він розроблений компанією Apple та використовується для побудови нативних застосунків для iPhone, iPad та інших пристроїв Apple.

UIKit є базовим інструментом для створення вікон, кнопок, текстових полів, таблиць, навігації, анімацій та інших елементів інтерфейсу. Саме за допомогою цього фреймворку розробники реалізують зручний, зрозумілий та функціональний дизайн мобільних додатків.

У мобільній розробці для збереження даних користувача, кешування інформації, роботи в офлайн-режимі та синхронізації з сервером широко застосовується Core Data. Core Data є потужним інструментом для локального зберігання та управління даними в iOS-додатках. Він дозволяє ефективно працювати з великими обсягами інформації, забезпечує офлайн-функціональність та високу продуктивність. У мобільних застосунках Core Data особливо корисний для збереження даних користувачів, історії операцій, налаштувань та кешування інформації, що робить його важливим компонентом сучасної iOS-розробки.

Firebase є потужною хмарною платформою для розробки мобільних додатків, яка значно спрощує створення серверної частини, авторизації, роботи з даними та аналітики. Завдяки широкому набору сервісів Firebase дозволяє швидко розробляти сучасні, масштабовані та безпечні мобільні додатки з мінімальними витратами на інфраструктуру.

MapKit — це фреймворк компанії Apple, призначений для інтеграції картографічних сервісів у мобільні додатки на платформі iOS. Він дозволяє розробникам додавати карти, маркери, маршрути, геолокацію та інші функції, пов'язані з відображенням і обробкою просторових даних.

MapKit широко використовується у мобільних додатках, де необхідна робота з місцезнаходженням користувача, наприклад у сервісах доставки, навігації, туристичних застосунках, а також у страхових мобільних додатках для фіксації місця страхового випадку або пошуку найближчих офісів компанії.

Push-сповіщення — це технологія доставки повідомлень від серверу мобільного застосунку до пристрою користувача, навіть якщо додаток не запущений у даний момент. Вони є важливим інструментом комунікації між сервісом і користувачем у сучасних мобільних додатках.

У мобільних системах push-сповіщення використовуються для інформування користувачів про події, оновлення, нагадування та важливі зміни. Наприклад, у страхових додатках вони можуть повідомляти про статус страхового випадку, нагадування про оплату полісу або завершення терміну дії договору.

Зроблені висновки під час дослідження та проєктування додатку дають підстави стверджувати, що для демонстраційної реалізації доцільно використати Android Studio, Java, Android SDK та Gradle. Такий підхід дозволяє створити нативний Android-проєкт, який можна відкрити, запустити та продемонструвати безпосередньо в Android Studio.

Для забезпечення простоти відкриття й перевірки прототипу Java була обрана як основна мова реалізації клієнтської частини. У подальшій промисловій версії застосунок може бути розширений за допомогою Kotlin, Firebase, REST API, SQLite/Room Database, платіжного модуля та серверної частини.

Firebase був обраний як основна технологія для розробки бекенду, оскільки ця платформа надає необхідні інструменти для зберігання та обробки даних, а також автентифікації користувачів та керування рівнями доступу. Firebase також має готові інструменти для розробки інтерактивних додатків, що значно спрощує процес розробки та зменшує час на написання коду.

Android Studio та Java є поширеними інструментами в навчальній і промисловій Android-розробці, тому використання такого стеку спрощує демонстрацію результату, подальше розширення функціональності та підтримку застосунку.

Отже, використання Android Studio, Java, Android SDK та Gradle дозволило підготувати демонстраційний мобільний застосунок страхової компанії з базовими екранами, навігацією та сценаріями взаємодії користувача.

Для візуального представлення логіки проєктування було підготовлено структурну діаграму застосунку, ER-діаграму інформаційної моделі та блок-схему основного алгоритму роботи. Ці матеріали деталізують взаємозв'язок

функціональних модулів, структуру даних і послідовність дій користувача під час використання мобільного застосунку (рис. 2.3-2.5).

Структурна діаграма мобільного застосунку страхової компанії відображає логічну побудову системи, її основні модулі та взаємозв'язки між ними. Така діаграма дозволяє зрозуміти архітектуру додатку, розподіл функціональних блоків та принцип взаємодії користувача з системою.

Мобільний застосунок страхової компанії має клієнтську частину, яка працює на Android-пристрої, а у промисловій версії також повинен мати серверну частину, базу даних і захищений API для обміну даними між застосунком та інформаційними системами страхової компанії.

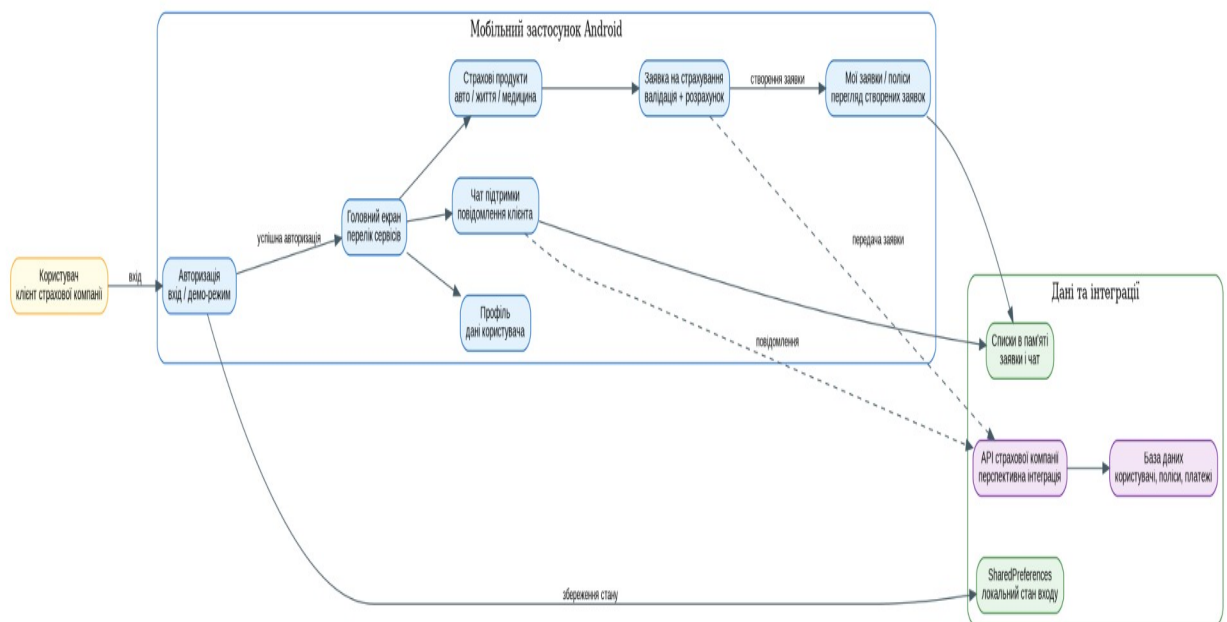


Рис. 2.3. Структурна діаграма мобільного застосунку страхової компанії

ER-діаграма (Entity-Relationship діаграма) інформаційної моделі мобільного застосунку страхової компанії відображає структуру бази даних, основні сутності системи, їхні атрибути та зв'язки між ними. Вона використовується для логічного проєктування даних і дозволяє чітко визначити, як інформація зберігається, обробляється та взаємодіє у межах системи. У контексті страхового мобільного застосунку ER-діаграма відображає ключові

об'єкти, пов'язані з клієнтами, страховими полісами, платежами та страховими випадками (рис. 2.4)

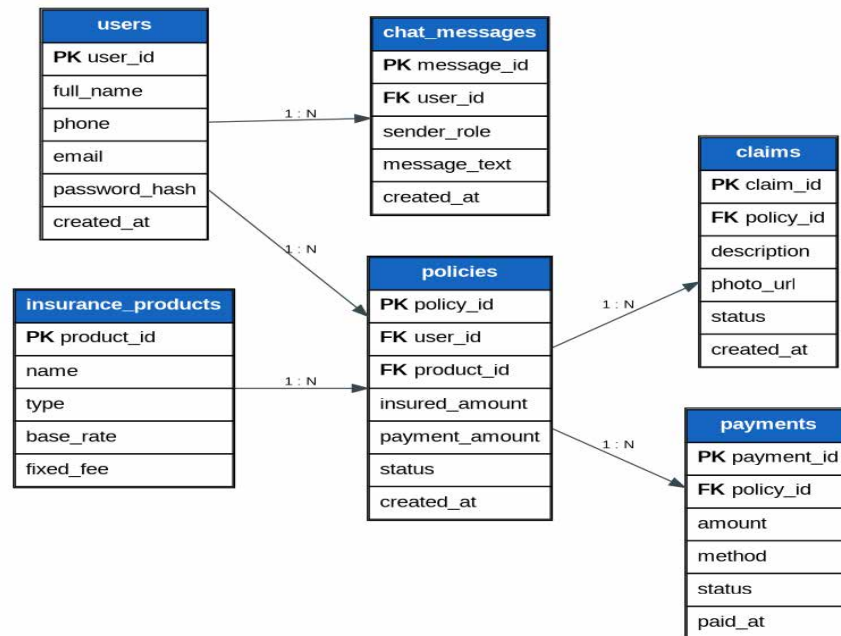


Рис. 2.4 ER-діаграма інформаційної моделі застосунку

ER-діаграма інформаційної моделі мобільного застосунку страхової компанії дозволяє чітко структурувати дані системи та визначити взаємозв'язки між основними сутностями. Така модель забезпечує правильну організацію бази даних, зручність обробки інформації, цілісність даних і ефективну роботу мобільного застосунку в цілому.

Блок-схема алгоритму роботи мобільного застосунку страхової компанії відображає послідовність виконання основних процесів у системі, починаючи з запуску додатку і завершуючи виконанням користувацьких операцій. Вона демонструє логіку роботи програмного забезпечення, взаємодію користувача з інтерфейсом, а також обмін даними з серверною частиною та базою даних. На першому етапі – початок роботи застосунку. Після відкриття додатку система ініціалізує основні компоненти, перевіряє підключення до Інтернету та завантажує стартовий екран.

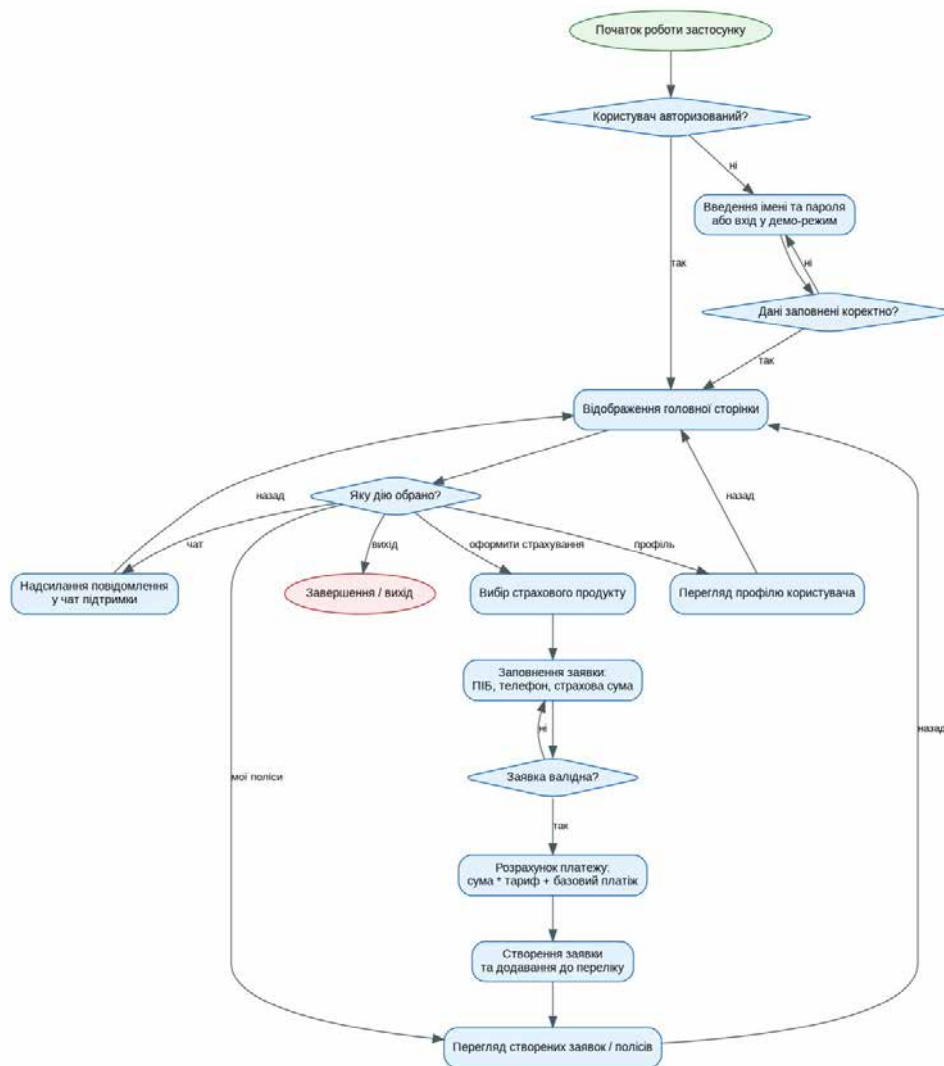


Рис. 2.5 Блок-схема алгоритму роботи мобільного застосунку

Другий етап – авторизація користувача. На цьому етапі користувач проходить процес входу в систему: введення логіну та пароля; перевірка даних у базі; у разі успішної авторизації – перехід до головного меню; у разі помилки – повторний ввід або реєстрація. Наступний етап – відображення головної сторінки.

Після входу користувач отримує доступ до основних функцій:

- перегляд профілю;
- вибір страхового продукту;
- завершення;
- надсилання повідомлень в чат.

У випадку вибору страхового продукту є можливість заповнити заявку та додати необхідну інформацію. Користувач може переглядати, оновлювати або перевіряти статус свого страхового договору. Система дозволяє здійснювати оплату та переглядати історію транзакцій.

Усі запити з мобільного додатку передаються на сервер через API (наприклад, з використанням Retrofit). Сервер обробляє запити та взаємодіє з базою даних.

Інформація зберігається у локальній або хмарній базі даних SQLite. Вона містить дані про користувачів, поліси, платежі та страхові випадки. У разі важливих подій система надсилає push-повідомлення через Firebase Cloud Messaging:

- нагадування про оплату;
- статус розгляду заяви;
- оновлення інформації.

Користувач може завершити сесію, після чого система виконує очищення тимчасових даних та повертається до екрану входу.

Таким чином, блок-схема алгоритму роботи мобільного застосунку страхової компанії демонструє послідовну взаємодію користувача з системою, обробку даних на сервері та збереження інформації в базі даних. Вона дозволяє зрозуміти загальний принцип функціонування застосунку, логіку обробки запитів і структуру програмного процесу, що забезпечує стабільну та ефективну роботу системи.

Висновок до розділу

У другому розділі було здійснено проєктування мобільного додатку для страхової компанії, визначено основні принципи побудови інтерфейсу, функціональні вимоги та технологічний стек розробки. У процесі роботи було розроблено UX-структуру застосунку, спроектовано основні екрани та логіку взаємодії користувача із системою.

Проведений аналіз дозволив визначити ключові вимоги до мобільного додатку, серед яких зручність використання, безпека персональних даних,

адаптивність, швидкодія та підтримка основних страхових сервісів. Також було обґрунтовано вибір технологій Android Studio, Java, Firebase, SQLite та Retrofit для реалізації функціональних можливостей системи.

У результаті проєктування було сформовано структурну діаграму застосунку, ER-діаграму інформаційної моделі та блок-схему алгоритму роботи системи, що дозволило визначити архітектуру додатку, взаємозв'язки між компонентами та послідовність виконання основних процесів. Отримані результати створюють основу для подальшої програмної реалізації мобільного застосунку страхової компанії.

3 ЗАСОБИ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

3.1 Аутентифікація та авторизація користувачів

Аутентифікація та авторизація є ключовими елементами забезпечення безпеки мобільних додатків, зокрема у сфері страхових послуг, де обробляються персональні та фінансові дані користувачів. Вони забезпечують контроль доступу до системи та захист інформації від несанкціонованого використання. Аутентифікація — це процес підтвердження особи користувача шляхом перевірки його облікових даних. Аутентифікація є базовим механізмом інформаційної безпеки мобільних додатків, що забезпечує підтвердження особи користувача перед наданням доступу до системи. У контексті мобільних застосунків, зокрема у сфері страхування, аутентифікація відіграє критичну роль, оскільки обробляються конфіденційні дані (персональні, фінансові, медичні).

Основною метою аутентифікації є:

- ідентифікація користувача;
- запобігання несанкціонованому доступу;
- забезпечення цілісності та конфіденційності даних.

Сучасні системи аутентифікації базуються на використанні одного або кількох факторів:

- фактор знання (Something you know) (пароль, PIN-код, відповіді на секретні питання);
- фактор володіння (Something you have) (мобільний пристрій, одноразовий код (OTP), токен або смарт-карта);
- біометричний фактор (Something you are) (відбиток пальця, розпізнавання обличчя, голосова ідентифікація).

Процес аутентифікації включає такі етапи:

- 1) введення користувачем облікових даних;
- 2) передача даних на сервер;

- 3) перевірка (валідація);
- 4) генерація токена доступу;
- 5) надання доступу до системи.

Авторизація користувачів у мобільних додатках страхових компаній є процесом визначення та контролю прав доступу до функціональних можливостей системи після успішної аутентифікації. У зв'язку з обробкою персональних, фінансових та медичних даних клієнтів, вимоги до авторизації в страхових сервісах є підвищеними.

Основною метою авторизації є:

- забезпечення диференційованого доступу до функцій додатку;
- захист конфіденційної інформації користувачів;
- запобігання несанкціонованим операціям;
- контроль виконання дій у системі.

На відміну від загальних мобільних сервісів, страхові додатки характеризуються:

- високим рівнем регуляторних вимог до захисту даних;
- необхідністю обробки фінансових транзакцій;
- наявністю юридично значущих документів (страхових полісів);
- потребою у відстеженні історії дій користувача.

Це обумовлює використання більш складних моделей авторизації та багаторівневого контролю доступу.

На рис. 3.1, 3.2 зображено процес авторизації в мобільному додатку. Цей процес складається з п'яти кроків.

Перший етап - користувач запускає мобільний додаток і потрапляє на екран авторизації.

Другий етап - користувач вводить свої ідентифікаційні дані, наприклад, електронну пошту та пароль, у відповідні поля на екрані.

Третій етап - користувач натискає кнопку "Увійти" або подібну, щоб ініціювати процес авторизації.

Четвертий етап - мобільний додаток перевіряє введені ідентифікаційні дані зі збереженими в базі даних.

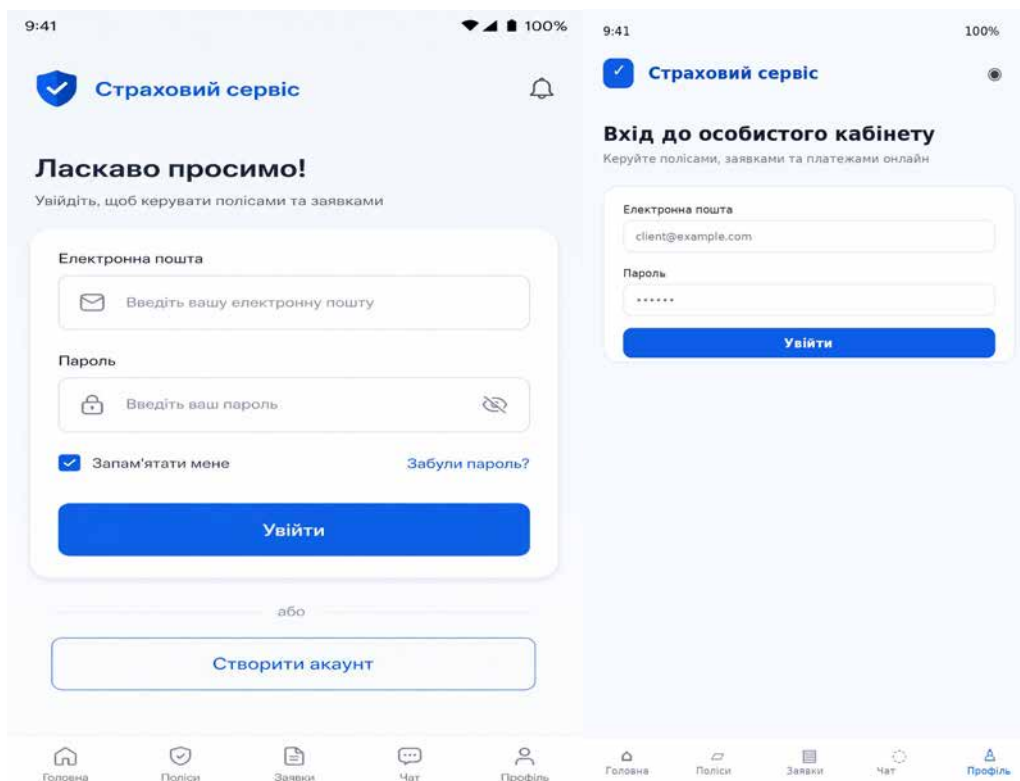


Рис. 3.1 Екран входу до демонстраційної версії мобільного застосунку (власна розробка автора)

```
public void showLogin() {
    base(); // створює чистий екран і задає білий/світлий фон
    LinearLayout c = contentBase(); // головний контейнер екрана
    c.setGravity(Gravity.CENTER_HORIZONTAL); // вирівнює вміст по центру
    c.addView(appHeader("Страховий сервіс", false)); // верхня частина з логотипом і назвою
    c.addView(tv("Вхід до особистого кабінету", 26, DARK, Typeface.BOLD));
    // великий заголовок екрана входу
    c.addView(tv("Керуйте полісами, заявками та платежами онлайн", 15, MUTED, Typeface.NORMAL));
    // короткий опис під заголовком
    EditText email = input("Електронна пошта", "client@example.com");
    // поле введення email
    EditText password = input("Пароль", "123456");
    // поле введення пароля
    password.setInputType(InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD);
    // робить пароль прихованим
    Button login = button("Увійти", BLUE, Color.WHITE);
    // синя кнопка входу
    login.setOnClickListener(v -> {
        long userId = dbHelper.authenticate(email.getText().toString().trim(),
password.getText().toString().trim());
        // перевіряє email і пароль у таблиці users
        if (userId > 0) {
            currentUserId = userId; // зберігає ID користувача
            showDashboard(); // відкриває головний екран
        } else {
            Toast.makeText(this, "Невірний email або пароль", Toast.LENGTH_LONG).show();
            // показує помилку входу
        }
    });
}
```

Рис 3.2. Фрагмент коду з поясненням

Таблиця 3.1

Загальна структура коду

MainActivity.java	головний файл інтерфейсу: створює екрани, картки, кнопки, нижнє меню та переходи між екранами.
InsuranceDbHelper.java	файл бази даних: створює SQLite-таблиці, зберігає користувача, поліси, заявки та чат.
AndroidManifest.xml	описує назву програми, тему та стартовий екран.
schema.sql	структура бази даних у SQL-форматі.
Що бачить користувач	Користувач вводить email і пароль. Після натискання «Увійти» програма перевіряє дані в SQLite-базі.
Де це в коді	showLogin()
Головна логіка	метод створює елементи екрана, додає їх у контейнер і призначає дії кнопкам.

П'ятий етап — якщо ідентифікація пройшла успішно, додаток надає користувачеві доступ до особистого облікового запису та переходить до відповідного екрану або функціоналу. Опис функціоналу наведено в табл. 3.2. Приклад наведений на рис. 3.3

При формуванні головного екрану відповідальний метод у коді: showDashboard(), summaryCard(), addQuickAction(), addServiceCard()

Таблиця 3.2

**Опис функціоналу головного екрана мобільного застосунку
страхової компанії**

Що бачить користувач	Показує назву сервісу, привітання, активні поліси, швидкі дії та послуги страхової компанії.
Де це в коді	showDashboard(), summaryCard(), addQuickAction(), addServiceCard()
Головна логіка	метод створює елементи екрана, додає їх у контейнер і призначає дії

	кНОПКАМ.
--	----------

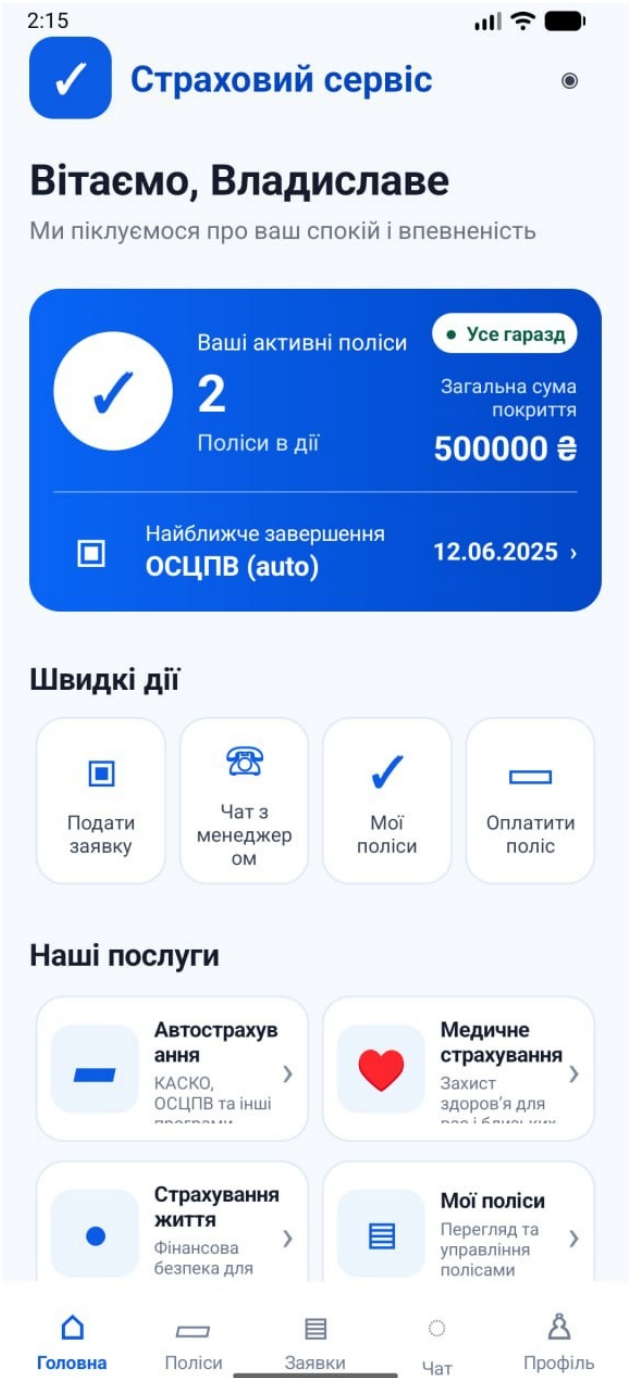


Рис. 3.3 Головна сторінка демонстраційної версії мобільного застосунку страхової компанії (власна розробка автора)

```
public void showDashboard() {
    base(); // очищує екран

    LinearLayout c = contentBase(); // створює контейнер для головної сторінки
    c.addView(appHeader("Страховий сервіс", true)); // додає верхнє меню з назвою і сповіщеннями

    int activePolicies = dbHelper.getActivePoliciesCount(currentUserId);
    // бере з бази кількість активних полісів користувача

    double coverage = dbHelper.getTotalCoverage(currentUserId);
    // рахує загальну суму покриття за активними полісами

    c.addView(tv("Вітаємо, Владиславе", 28, DARK, Typeface.BOLD));
    // показує персональне привітання

    c.addView(summaryCard(activePolicies, coverage));
    // додає велику синю картку з активними полісами

    addQuickAction(quick, "■", "Подати\пзаявку", v -> showApplicationForm());
    // кнопка швидкої дії відкриває форму заявки

    addQuickAction(quick, "💬", "Чат з\нменеджером", v -> showChat());
    // кнопка відкриває чат із менеджером

    addServiceCard(services, "■", "Автострахування", "КАСКО, ОСЦПВ та інші програми", v ->
showProducts());
    // додає картку послуги автострахування

    setScreen(c, "Головна");
    // показує головний екран і підсвічує пункт меню «Головна»
}
```

Рис.3.4 Фрагмент коду з поясненням до головної сторінки застосунку (власна розробка)

На рис. 3.3 представлено приклад екрана демонстраційного застосунку, який відображає вибір страхових продуктів та перехід до основних сценаріїв роботи користувача.

По-перше, мінімальна довжина: повинна містити принаймні 4 символи для забезпечення базового рівня безпеки, хоча є рекомендації використовувати більш складні паролі.

По-друге, дотримуватись використовувати складні символи: це містити різні типи символів (великі літери, маленькі літери, цифри та спеціальні символи), що допомагає ускладнити процес перебору паролів та підвищує його безпеку.

По-третє, заборона використання загальовживаних паролів, наприклад, не повинен бути таким як "password", "123456" тощо, це допомагає уникнути очевидних паролів, які легко вгадати.

По-четверте, хешування паролів: при збереженні паролів в базі даних, рекомендується використовувати хеш-функцію для збереження хеш-значення паролю замість збереження самого пароля, що забезпечує додатковий рівень безпеки, оскільки навіть у разі отримання доступу до бази даних з хеш-значеннями паролів, злоумисник не зможе одержати оригінальні паролі.

Якщо ідентифікація не вдалася, користувачу відображається повідомлення про помилку, і він може спробувати авторизуватися знову або скористатися опцією відновлення пароля або створення нового облікового запису (рис. 3.5)

9:41 100%

Страховий сервіс

Ласкаво просимо!
Увійдіть, щоб керувати полісами та заявками

Електронна пошта
Введіть вашу електронну пошту

Пароль
Введіть ваш пароль

☒ Запам'ятати мене [Забули пароль?](#)

Увійти

або

Створити акаунт

Головна Поліси Заявки Чат Профіль

Рис. 3.5 Екран повторного входу до застосунку після неуспішної спроби авторизації (власна розробка автора)

В даному випадку проілюстровано базовий процес авторизації в мобільному додатку. В реальному додатку можуть бути додаткові функції, такі як використання двофакторної аутентифікації, використання сторонніх служб для авторизації (наприклад, Google або Facebook), а також забезпечення безпеки за допомогою шифрування та інших методів.

3.2 Налаштування мобільного додатку

Налаштування мобільного додатку є важливим функціональним компонентом інтерфейсу, що забезпечує адаптацію системи до індивідуальних потреб користувача. У мобільних додатках страхових компаній цей модуль відіграє особливу роль, оскільки дозволяє керувати параметрами безпеки, персоналізації та взаємодії з сервісом.

Основною метою налаштувань є:

- підвищення зручності користування додатком;
- забезпечення контролю над персональними даними;
- адаптація функціоналу до потреб користувача;
- підвищення рівня безпеки

При проєктуванні враховувалася можливість редагування особистої інформації користувача, такої як прізвище, ім'я, номер телефону та адреса проживання. Це дозволяє користувачам оновлювати свої дані та забезпечує актуальність інформації в системі.

Налаштування додатку також надають можливість доступу до особистого кабінету користувача. Цей особистий кабінет зазвичай містить додаткові функції та інформацію, які стосуються страхових полісів, заявок, історії платежів та інших важливих деталей.

Особистий кабінет зображено на рис. 3.6.

Сторінка клієнта є одним із основних елементів мобільного додатку страхової компанії, оскільки саме через неї користувач отримує доступ до персональної інформації, страхових послуг та основних функцій системи. Вона виконує роль особистого кабінету, де зберігаються дані про клієнта, його страхові поліси, платежі, історія звернень та інші важливі відомості.

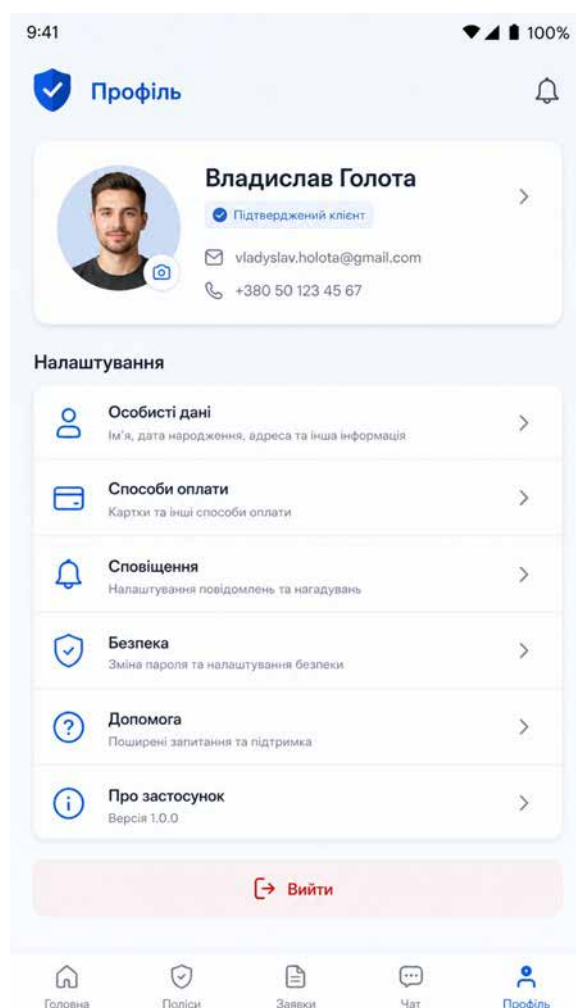


Рис. 3.5 Екран профілю користувача в демонстраційній версії мобільного застосунку (власна розробка автора)

Створення сторінки клієнта передбачає проєктування інтерфейсу, налаштування взаємодії з базою даних, реалізацію авторизації та забезпечення безпеки персональних даних.

Після натискання на кнопку «Профіль» користувача перенаправляє на сторінку з особистими даними. На даній сторінці відображається інформація, яка була введена при реєстрації в мобільному застосунку (рис. 3.6). Фрагмент коду з поясненням представлено на рис. 3.7.

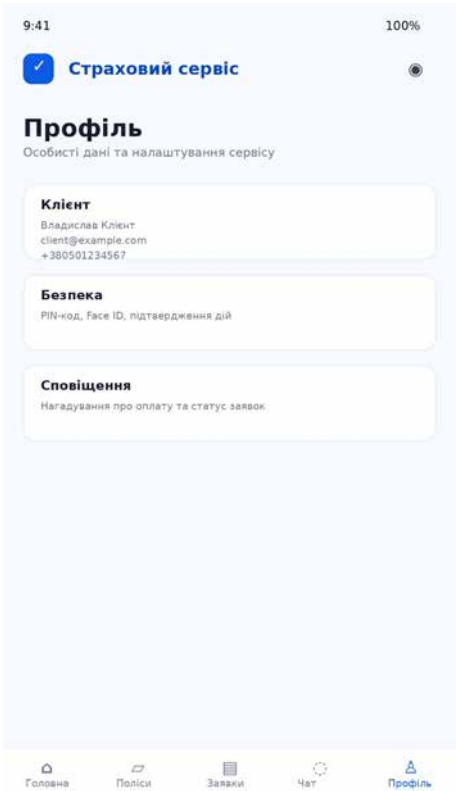


Рис. 3.6 Екран профілю у застосунку

Таблиця 3.3

Опис функціоналу «Екран профілю» - відповідальний метод у коді:
showProfile()

Що бачить користувач	Показує дані клієнта, налаштування безпеки, сповіщення і кнопку виходу.
Де це в коді	showProfile()
Головна логіка	метод створює елементи екрана, додає їх у контейнер і призначає дії кнопкам.

```
public void showProfile() {
    base(); // очищує екран
    LinearLayout c = contentBase(); // контейнер профілю
    c.addView(appHeader("Страховий сервіс", true)); // верхня панель
    c.addView(tv("Профіль", 26, DARK, Typeface.BOLD));
    // заголовок екрана
    c.addView(cardText("Клієнт", dbHelper.getUserFullName(currentUserId) +
"\nclient@example.com\n+380501234567"));
    // картка з персональними даними клієнта
    c.addView(cardText("Безпека", "PIN-код, Face ID, підтвердження важливих дій"));
    // картка налаштувань безпеки
```

```

c.addView(cardText("Сповіщення", "Нагадування про оплату, статус заявок, оновлення полісів"));
// картка налаштувань сповіщень
Button logout = button("Вийти", Color.rgb(239, 68, 68), Color.WHITE);
// червона кнопка виходу з акаунта
logout.setOnClickListener(v -> showLogin());
// після натискання повертає користувача на екран входу
setScreen(c, "Профіль");
// показує екран і підсвічує пункт меню «Профіль»
}

```

Рис. 3.7 Фрагмент коду з поясненнями до вікна «Екран профілю»

Ілюстрація створення та реалізації описи відповідних функціоналів «поліси», «заявки», «чат» з менеджером відображено в розділі 4.

3.3 Створення бази даних та структури даних застосунку

У демонстраційній версії застосунку використано спрощену локальну структуру даних, яка відображає основні сутності страхової системи: користувачів, страхові продукти, поліси, заявки, платежі та повідомлення чату. У промисловій версії ці дані мають зберігатися у серверній базі даних або у хмарному сховищі з використанням захищеного API.

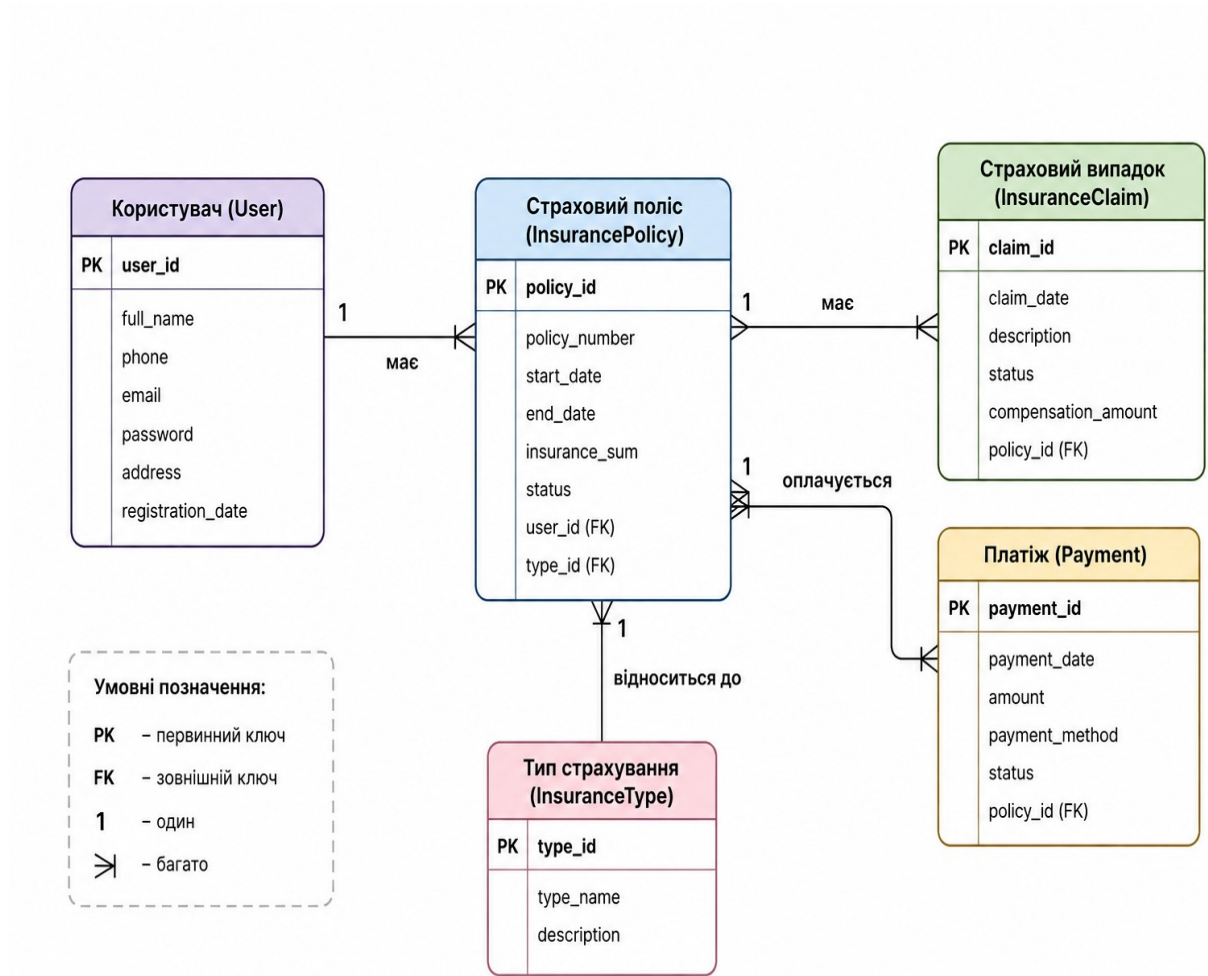


Рис. 3.8 Концептуальна схема бази даних мобільного застосунку для страхової компанії

У табл. 3.4 наведено основні таблиці, які можуть використовуватися в базі даних мобільного застосунку, та їх призначення.

Таблиця 3.4
Основні таблиці бази даних та їх призначення

Назва таблиці	Призначення
users	Зберігання облікових даних користувачів, ролей та контактної інформації
insurance_products	Довідник страхових продуктів: авто, медичне страхування, страхування життя
policies	Дані про оформлені страхові поліси, терміни дії, статуси та суми покриття
applications	Заявки користувачів на оформлення страхового

	продукту
claims	Звернення щодо страхових випадків і їх поточні статуси
payments	Історія платежів, суми, дати та статуси оплат
chat_messages	Повідомлення між користувачем і працівником страхової компанії
status_dictionary	Довідник статусів договорів, заявок і платежів
documents	Дані про прикріплені файли: фото, PDF-поліси, скани документів
notifications	Push-сповіщення та системні повідомлення для користувача

Таблиця 3.5

Довідник статусів страхового договору

StatusID	StatusName
1	Чернетка
2	Очікує оплати
3	Активний
4	Завершений
5	Скасований
6	На розгляді
7	Потребує уточнення
8	Закритий

Висновок до розділу

У третьому розділі було розглянуто основні засоби розробки мобільного додатку для страхової компанії, а також реалізовано базові механізми аутентифікації, авторизації, налаштування користувацького профілю та структури зберігання даних.

У процесі роботи було досліджено принципи забезпечення інформаційної безпеки мобільних застосунків, визначено особливості реалізації авторизації та аутентифікації користувачів, а також розроблено демонстраційні екрани входу, головної сторінки та профілю користувача. Реалізований функціонал забезпечує базовий контроль доступу до системи, взаємодію користувача з основними сервісами застосунку та можливість керування персональними даними.

Також у межах розділу було сформовано структуру бази даних мобільного застосунку, визначено основні сутності системи та їх взаємозв'язки. Запропонована модель даних включає таблиці користувачів, страхових продуктів, полісів, заявок, платежів, повідомлень та системних сповіщень, що забезпечує логічну організацію інформації та підтримку основних бізнес-процесів страхової компанії.

Отримані результати підтверджують доцільність використання Android Studio, Java та SQLite для створення демонстраційної версії мобільного застосунку та формують основу для подальшого розширення функціональних можливостей системи, інтеграції із серверною інфраструктурою та впровадження додаткових механізмів захисту даних.

4 ОПИС РЕАЛІЗАЦІЇ

4.1 Реалізація страхових продуктів

4.1.1 Створення сторінки споживача з вибором страхового продукту.

Реалізація страхових продуктів у мобільному додатку є одним із ключових етапів проєктування цифрового сервісу страхової компанії. Вона передбачає створення функціоналу, який дозволяє користувачам ознайомлюватися з видами страхування, розраховувати вартість полісів, оформлювати договори, здійснювати оплату та керувати власними страховими послугами без відвідування офісу компанії.

Правильне налаштування цього процесу забезпечує зручність для клієнта, автоматизацію внутрішніх бізнес-процесів та підвищення ефективності роботи страхової компанії.

На першому етапі формується перелік послуг, які будуть доступні в мобільному застосунку:

- автострахування (ОСЦПВ, КАСКО);
- медичне страхування;
- страхування майна;
- туристичне страхування;
- страхування життя;
- корпоративне страхування;
- страхування відповідальності.

Для кожного продукту визначаються умови, тарифи, строки дії та особливості оформлення.

У додатку створюється зручний каталог, де користувач може:

- переглядати доступні страхові продукти;
- ознайомлюватися з умовами страхування;
- порівнювати варіанти;
- отримувати рекомендації;
- переходити до оформлення полісу.

Інтерфейс реалізується через Android Studio (рис. 4.1)

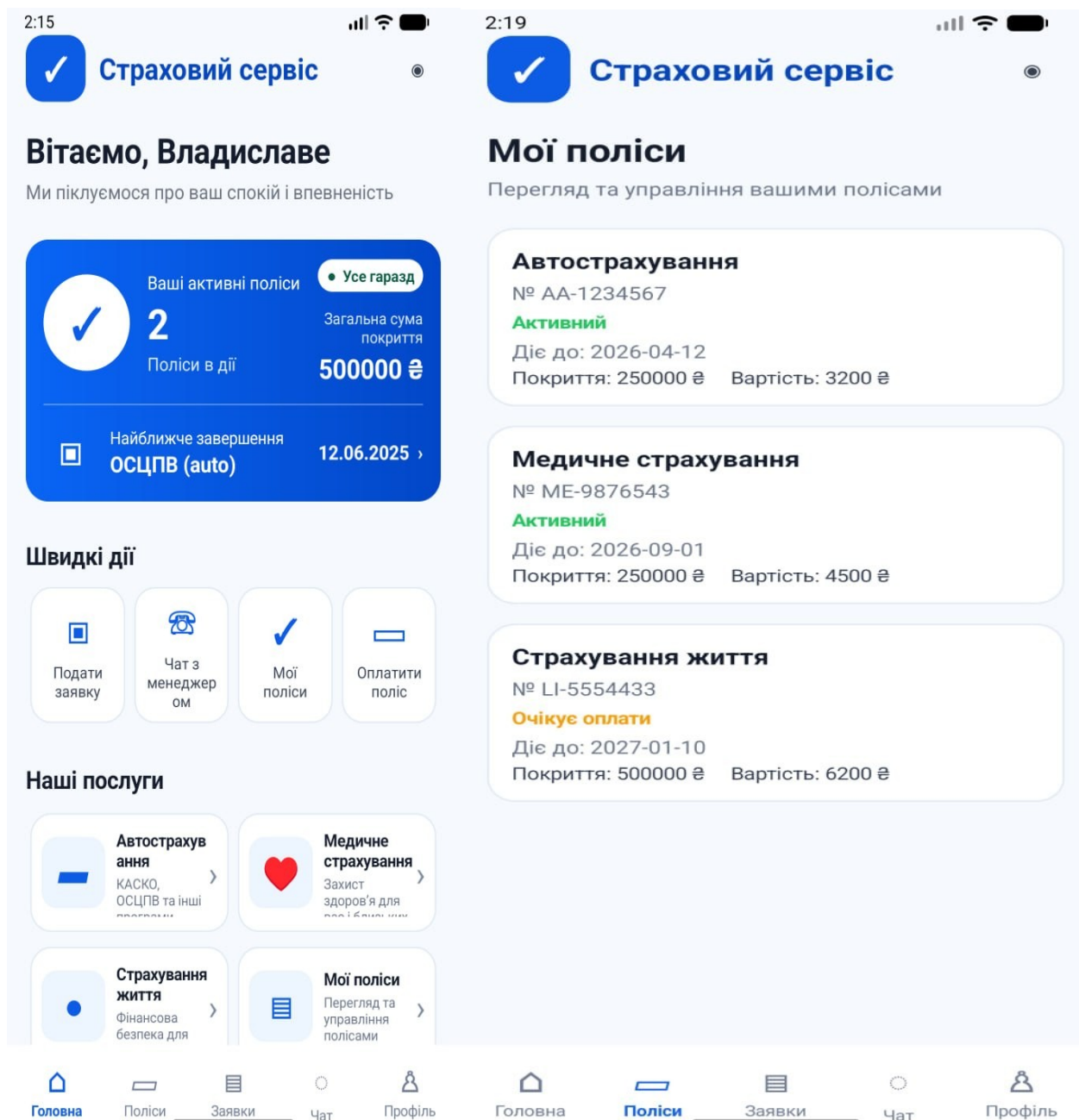


Рис. 4.1 Екран переліку заявок і полісів користувача у застосунку (власна розробка автора)

Для зручності користувача створюється калькулятор, який дозволяє:

- вводити параметри страхування;
- автоматично розраховувати вартість полісу;
- враховувати ризики, вік, стаж, тип майна тощо;
- формувати попередню пропозицію.

Таблиця 4.1

Опис функціоналу екран «поліси» - відповідальний метод у коді: showPolicies()

Що бачить користувач	Показує поліси користувача. Дані беруться з таблиці policies і страхових продуктів.
Де це в коді	showPolicies()
Головна логіка	метод створює елементи екрана, додає їх у контейнер і призначає дії кнопкам.

```
public void showPolicies() {
    base(); // створює чистий екран
    LinearLayout c = contentBase(); // контейнер для списку полісів
    c.addView(appHeader("Страховий сервіс", true)); // верхня панель
    c.addView(tv("Мої поліси", 26, DARK, Typeface.BOLD));
    // заголовок екрана
    List<String[]> policies = dbHelper.getPolicies(currentUserId);
    // отримує з бази всі поліси поточного користувача
    for (String[] p : policies) {
        LinearLayout card = cardLayout(Color.WHITE, BORDER, 18);
        // створює білу картку для одного поліса
        card.addView(tv(p[1], 18, DARK, Typeface.BOLD));
        // назва страхового продукту
        card.addView(padBottom("№ " + p[0], 4));
        // номер поліса
        card.addView(tv(p[2], 14, p[2].contains("Активний") ? GREEN : Color.rgb(245, 158, 11),
            Typeface.BOLD));
        // статус поліса: зелений для активного, жовтий для очікування
        card.addView(padBottom("Діє до: " + p[3], 2));
        // дата завершення дії поліса

        c.addView(card); // додає картку поліса на екран
    }
    setScreen(c, "Поліси");
    // відкриває екран і підсвічує пункт нижнього меню «Поліси»
}
```

Рис. 4.2. Фрагмент коду з поясненнями до вікна «поліси» (власна розробка автора)

Це підвищує швидкість прийняття рішення клієнтом.

До системи пред'являються певні вимоги, а саме вона повинна дозволяти:

- заповнення анкети;
- завантаження документів;
- перевірку даних;
- електронне підписання;
- автоматичне формування договору;

- отримання електронного поліса.

Для збереження даних використовуються Firebase або SQLite.

4.1.2 Приклади реалізації сторінки «Автострахування». Страхування автомобіля з точки зору користувацького сценарію включає вибір типу страхування, введення даних транспортного засобу, попередній розрахунок вартості та подання заявки. На рис.4.2 зображено екран оформлення автострахування в демонстраційній версії застосунку.

- заголовок: на верхній частині сторінки розташований заголовок "Оформлення автострахування", що інформує користувача про ціль цієї сторінки.
- тип страхування: вид страховки.
- введення даних про автомобіль: модель авто, рік випуску, номерний знак тощо;
- вибір страхової суми: користувач може обрати страхову суму в залежності від терміну;
- вибір тарифу: користувач може вибрати один з доступних тарифів страхування.

Опція "Подати заявку" (рис. 4.4): після заповнення всіх необхідних полів, користувач може натиснути на цю кнопку для оформлення страхування авто.

Ця сторінка страхування авто надає зручний і простий інтерфейс для користувачів, де вони можуть легко вибрати та налаштувати своє страхування авто, ознайомитися з необхідною інформацією та оформити поліс.

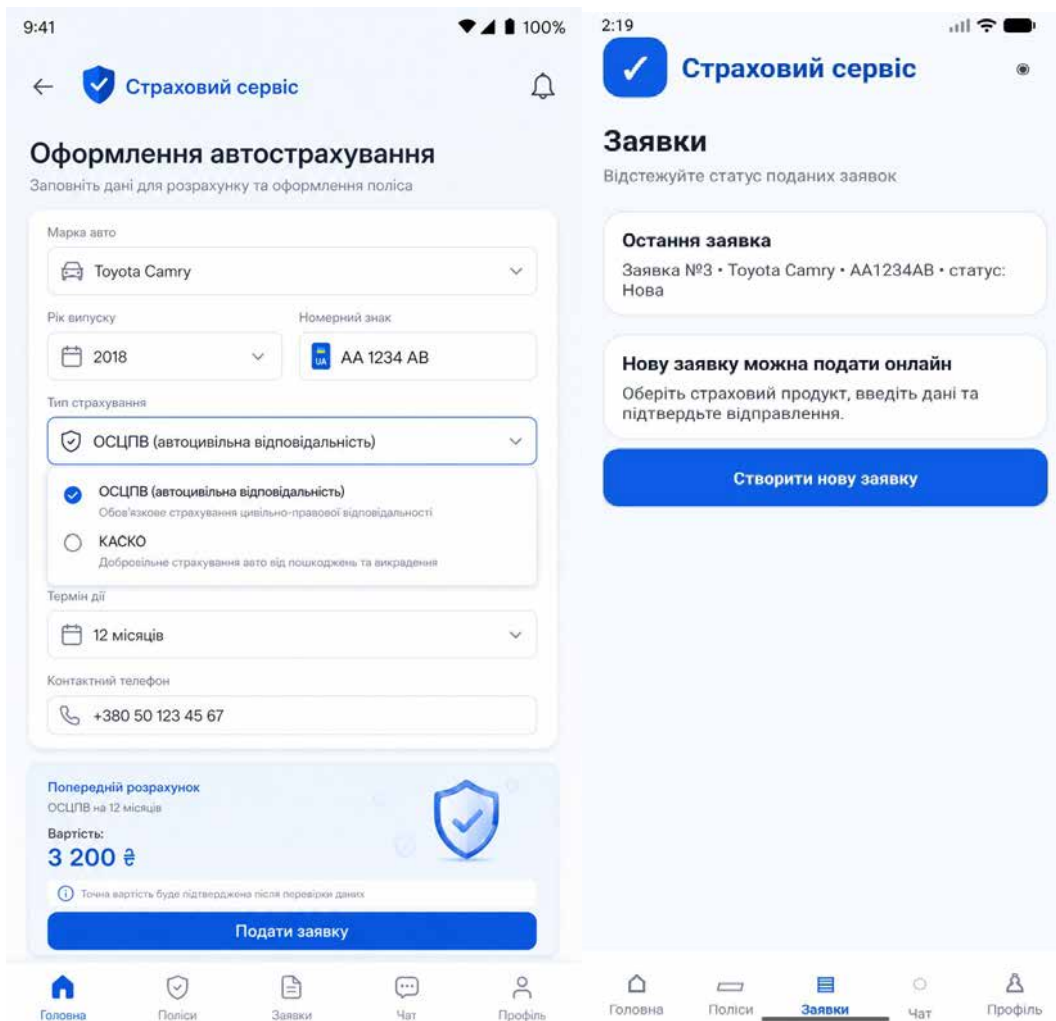


Рис. 4.3 – Екран оформлення автострахування в демонстраційній версії Android-застосунку (власна розробка автора)

На рис. 4.3 зображено екран із параметрами дії полісу страхування авто. Цей етап відбувається після вибору страхового продукту та заповнення базових даних.

Таблиця 4.2

Опис функціоналу екран «заявки» - відповідальний метод у коді:

`showApplicationForm()`

Що бачить користувач	Користувач вводить дані автострахування. Після натискання кнопки заявка записується в таблицю applications.
Де це в коді	<code>showApplicationForm()</code>
Головна логіка	метод створює елементи екрана, додає їх у контейнер і призначає дії кнопкам.

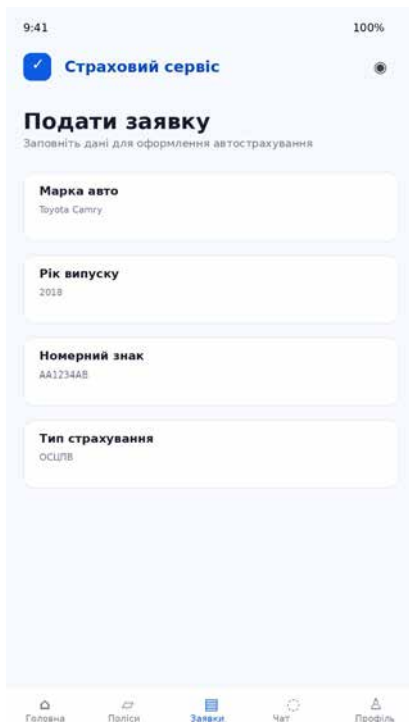


Рис. 4.4 Екран «заявка» у мобільному застосунку

```
public void showApplicationForm() {
    base(); // очищує екран
    LinearLayout c = contentBase(); // контейнер форми заявки
    c.addView(appHeader("Страховий сервіс", true)); // верхня панель
    EditText brand = input("Марка авто", "Toyota Camry");
    // поле введення марки автомобіля
    EditText year = input("Рік випуску", "2018");
    year.setInputType(InputType.TYPE_CLASS_NUMBER);
    // поле року випуску, дозволяє вводити тільки числа
    EditText plate = input("Номерний знак", "AA1234AB");
    // поле номерного знака
    EditText type = input("Тип страхування", "ОСЦПВ");
    // поле типу страхування
    Button send = button("Подати заявку", BLUE, Color.WHITE);
    // кнопка надсилення заявки
    send.setOnClickListener(v -> {
        long id = dbHelper.insertApplication(
            currentUserId, 1,
            brand.getText().toString().trim(),
            parsedYear,
            plate.getText().toString().trim(),
            type.getText().toString().trim(),
            3200
        );
        // записує заявку у таблицю applications

        Toast.makeText(this, "Заявку №" + id + " подано", Toast.LENGTH_LONG).show();
        // повідомляє користувачу, що заявку створено

        showApplications(); // переходить на екран заявок
    });
}
```

Рис. 4.5 Фрагмент коду з поясненнями до вікна «заявки»

1. Початок дії полісу — дата, з якої страховий поліс набуває чинності. У демонстраційній версії вона відображається як інформаційне поле.

2. Кінець дії полісу — дата завершення дії договору. Для автострахування типовим є період дії один рік, але у реальній системі термін може залежати від умов продукту.

3. Умови дії полісу визначають територію покриття, тип транспортного засобу, страхові ризики та інші параметри, які впливають на обсяг захисту.

4. Обов'язки страхувальника включають своєчасну оплату полісу, надання достовірної інформації, дотримання умов договору та повідомлення страхової компанії про настання страхового випадку.

5. Випадки, не покриті полісом, мають бути окремо зазначені в умовах договору. Це дає користувачу можливість зрозуміти обмеження страхового захисту.

4.1.3 Реалізація сторінки «Страхування життя». Страхування життя є одним із видів страхування, яке має свої особливості. При проєктуванні додатку варто враховувати особливості такого виду страхування. По-перше, тариф страхування життя визначається на основі ризиків, пов'язаних зі здоров'ям та віком страховика. Цей тариф може бути фіксованим або залежати від індивідуальних характеристик страхувальника. Страхова сума визначає суму, яку страхова компанія виплачує бенефіціару у разі смерті страхувальника. Ця сума може бути фіксованою або залежати від потреб та можливостей страхувальника. Деякі поліси страхування життя можуть включати додаткові витрати, такі як оплата лікарських послуг, оплата похоронних витрат або надання фінансової підтримки в разі тяжкого захворювання. Ці витрати можуть бути додатковими опціями, за які страхувальник сплачує додаткові премії. Страхування життя може мати певні особливості. Наприклад, існують два основних типи полісів: тимчасове страхування життя, яке надає покриття на

обмежений період часу, і постійне страхування життя, яке надає покриття протягом усього життя страхувальника. Крім того, страхувальник може мати можливість внести зміни до полісу, наприклад, збільшити страхову суму або додати додаткові опції.

На рис. 4.6 зображено екран страхування життя в демонстраційній версії мобільного застосунку страхової компанії.

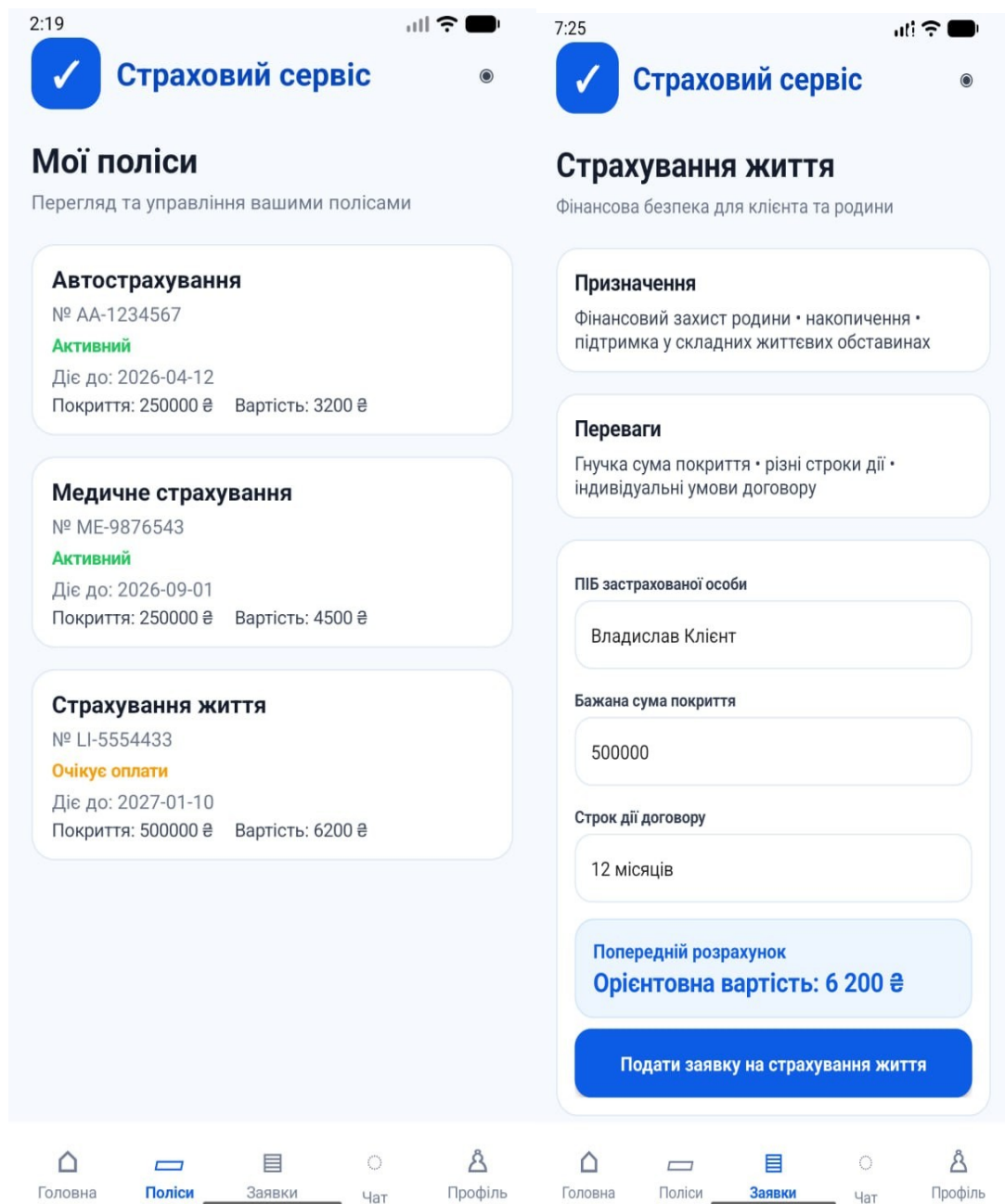


Рис. 4.6 – Екран вибору страхування життя у демонстраційній версії застосунку (власна розробка автора)

1. Назва: На верхній частині сторінки розміщена назва "Страхування життя", що чітко ідентифікує функціональність сторінки.
2. Особливості страхування життя: сторінка містить короткий опис продукту, його призначення, переваги та умови отримання страхового захисту.
3. Форма заповнення даних: користувач вводить базову інформацію, необхідну для попереднього розрахунку вартості страхового продукту.
4. Відображення страхової суми та тарифу: після заповнення форми система може показувати орієнтовні параметри страхового покриття.
5. Додаткові витрати: у промисловій версії сторінка може відображати комісії, додаткові послуги або умови оплати, якщо вони передбачені страховим продуктом.

Екран страхування життя може бути використаний як основа для подальшої адаптації під конкретний бренд, правила андеррайтингу та умови страхової компанії.

4.1.4 Реалізація сторінки «Медичне страхування». Медичне страхування є важливою складовою страхової сфери, а його реалізація в мобільному застосунку повинна забезпечувати користувачу зрозумілий опис продукту, умови покриття, розрахунок вартості та можливість оформлення заявки.

1. Медичне страхування забезпечує фінансовий захист користувача у разі хвороби, травми або необхідності отримання медичних послуг. У застосунку цей продукт подається через окрему інформаційну сторінку.
2. Тариф та страхова сума: тариф визначає вартість медичного страхування, а страхова сума — максимальний обсяг покриття, який може бути наданий за умовами договору.
3. Додаткові витрати: окремі медичні послуги можуть не входити до базового пакета, тому у промисловій системі потрібно окремо відображати франшизу, винятки та додаткові умови.
4. Особливості: медичне страхування може мати обмеження щодо віку,

стану здоров'я, переліку медичних закладів, видів послуг і терміну дії договору.

У мобільному застосунку страхової компанії можна реалізувати сторінку медичного страхування, де користувач ознайомлюється з умовами продукту, обирає тарифний план, вводить необхідні дані та подає заявку.

На рис. 4.7 зображено екран медичного страхування.

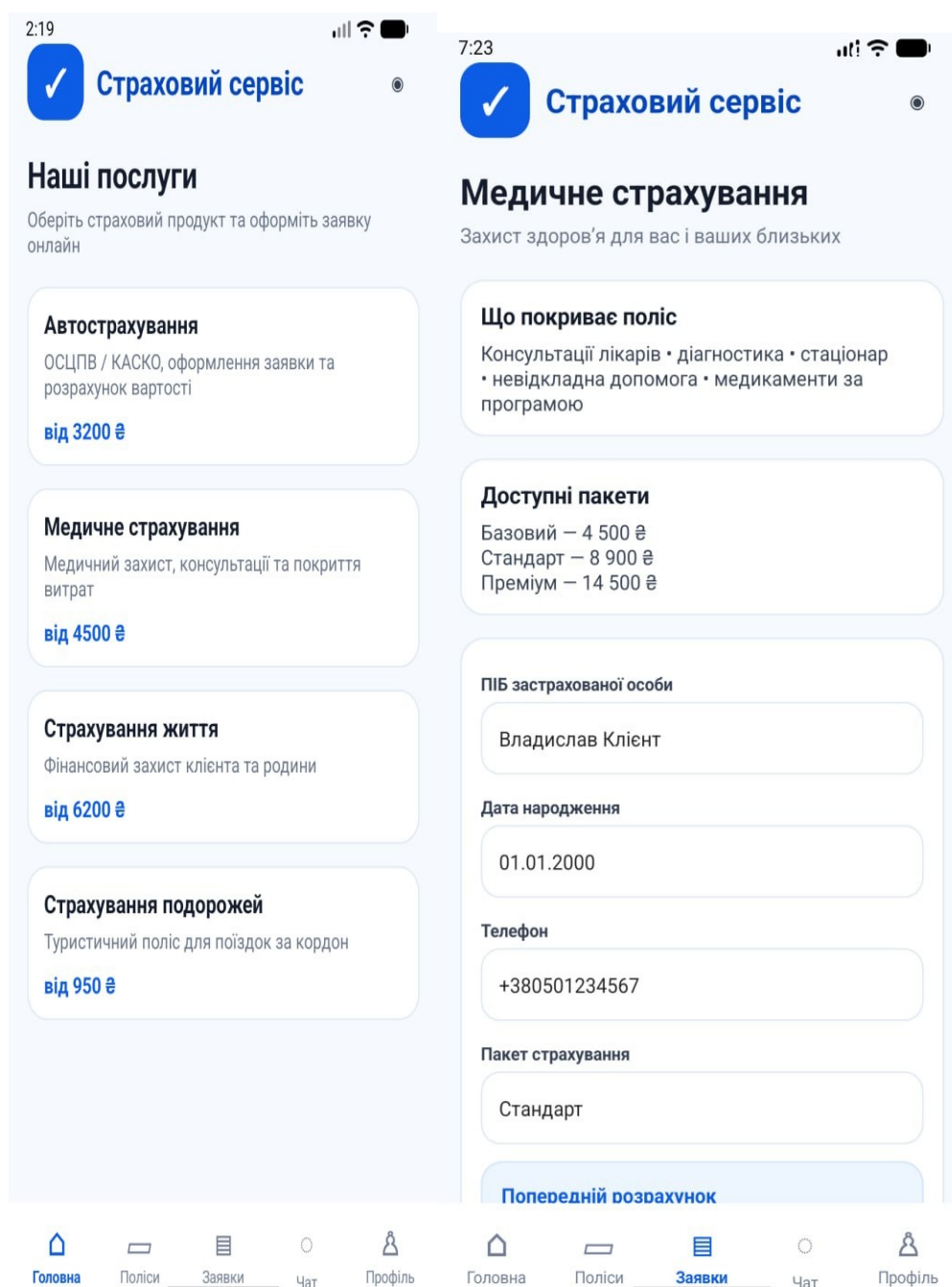


Рис. 4.7 Екран вибору медичного страхування у демонстраційній версії Android-застосунку (власна розробка автора)

Основні елементи цієї сторінки подано нижче.

1. Назва: у верхній частині сторінки розміщено назву «Медичне страхування», що чітко вказує на тип страхового продукту.
2. Особливості: сторінка містить інформацію про покриття медичних послуг, доступ до медичної мережі, консультації з лікарем, швидку допомогу або спеціалізований страховий захист.
3. Страхова сума: на сторінці зазначається максимальна сума покриття, яку страхова компанія може надати у разі настання страхового випадку.
4. Тариф: відображається орієнтовна вартість медичного страхування за обраний період дії договору.
5. Додаткові витрати: сторінка може містити інформацію про франшизу, винятки, додаткові медичні послуги або обмеження покриття.

Ці елементи допомагають користувачу зрозуміти умови медичного страхування, порівняти параметри продукту та прийняти рішення щодо оформлення заявки.

4.1.5 Створення сторінки чату працівників компанії з користувачами мобільного застосунку. Для забезпечення зручного спілкування працівників компанії з користувачами мобільного застосунку, створюється спеціальна сторінка чату. Ця сторінка дозволяє працівникам вести діалог з користувачами безпосередньо в межах мобільного застосунку, що забезпечує зручність і ефективність комунікації.

Створення сторінки чату передбачає наступні етапи:

1. Розробка інтерфейсу: Створення інтуїтивно зрозумілого і привабливого інтерфейсу, що дозволить працівникам легко взаємодіяти з користувачами. Інтерфейс може містити список контактів, повідомлення та додаткові функціональні елементи, такі як кнопки відправлення повідомлень або емодзі.

2. Функціональні можливості: Реалізація функцій чату, включаючи можливість відправлення та отримання повідомлень, можливість додавати вкладення, такі як фотографії або документи, а також можливість перегляду історії повідомлень.

3. Авторизація та аутентифікація: Забезпечення безпеки шляхом вимог авторизації та аутентифікації користувачів перед початком спілкування. Це може включати введення облікових даних, таких як ім'я користувача та пароль, або використання системи одноразових паролів або біометричного впізнавання.

4. Оновлення чату: Забезпечення автоматичного оновлення чату, щоб працівники могли бачити нові повідомлення в режимі реального часу. Це можна досягти за допомогою технологій, таких як сокети або пуш-сповіщення.

5. Система сповіщень: Інтеграція з системою сповіщень, щоб працівники отримували повідомлення навіть поза додатком. Це може включати сповіщення відразу на екрані пристрою, додаткові сповіщення на електронну пошту або SMS-повідомлення.

6. Можливість розширення: Розробка сторінки чату з урахуванням можливості майбутнього розширення функціоналу. Наприклад, можливість відправки файлів, проведення аудіо- та відеодзвінків або інтеграція з іншими системами.

7. Візуальний дизайн: Розробка привабливого та зручного використання візуального дизайну сторінки чату, включаючи вибір кольорів, шрифтів та іконок, що відповідають корпоративному стилю компанії.

Для початку чату з клієнтом, працівник повинен перейти на відповідну сторінку, натиснувши на кнопку користувача в мобільному застосунку. На цій сторінці працівник може спілкуватися з клієнтом безпосередньо, використовуючи чат, що дозволяє уникнути телефонного зв'язку. Чат автоматично оновлюється, тому коли надходять нові повідомлення, вони відображаються миттєво, і працівнику не потрібно оновлювати сторінку вручну. Також на сторінці показано час надсилання кожного повідомлення, що

допомагає контролювати процес спілкування з клієнтом. Зразок сторінки чату наведено на рис. 4.8.

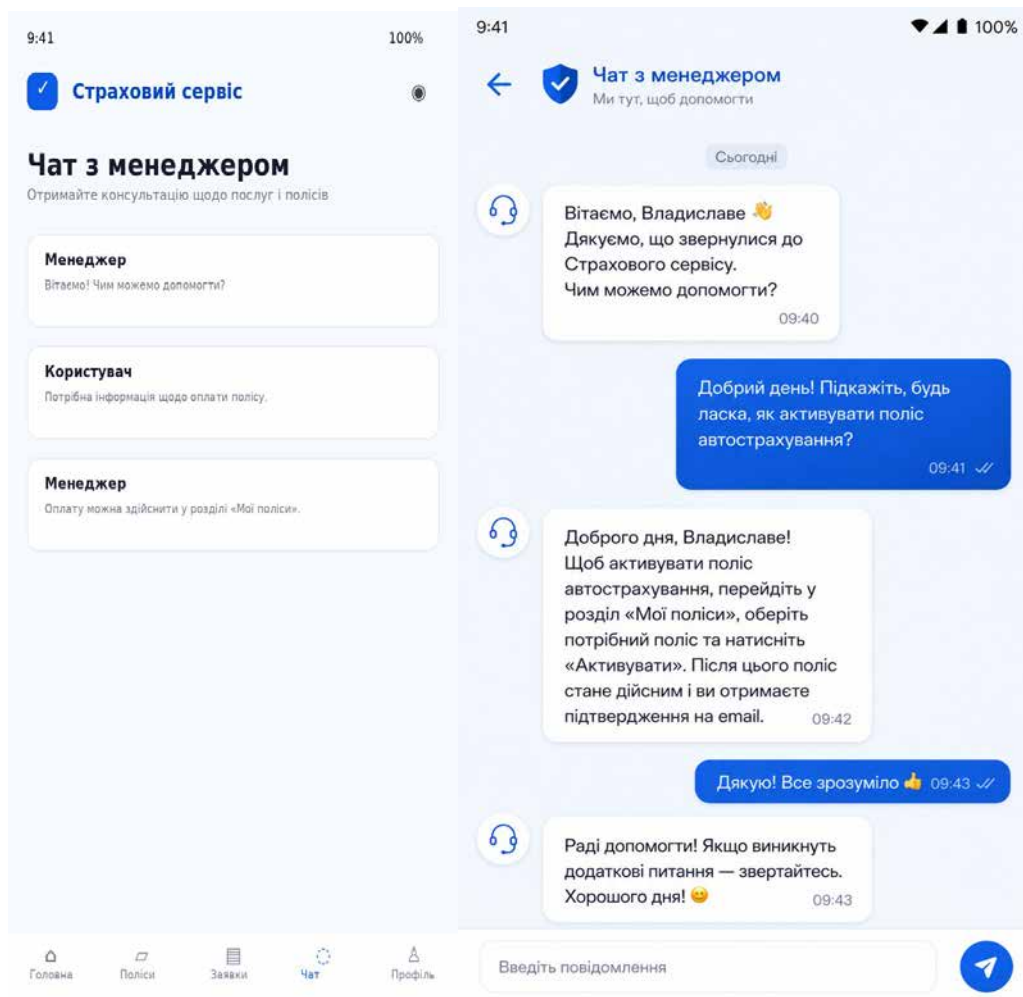


Рис. 4.8 Екран чату користувача з працівником страхової компанії (власна розробка автора)

Таблиця 4.3

Опис функціоналу екран «чат» - відповідальний метод у коді: `showChat()`

Що бачить користувач	Показує листування з менеджером. Нові повідомлення записуються в таблицю <code>chat_messages</code> .
Де це в коді	<code>showChat()</code>
Головна логіка	метод створює елементи екрана,

	додає їх у контейнер і призначає дії кнопкам.
--	---

```

public void showChat() {
    base(); // очищує екран
    LinearLayout c = contentBase(); // контейнер чату
    c.addView(appHeader("Страховий сервіс", true)); // верхня панель
    List<String[]> messages = dbHelper.getChatMessages(currentUserId);
    // отримує з бази повідомлення користувача
    for (String[] msg : messages) {
        boolean manager = msg[0].contains("Менеджер");
        // визначає, хто написав повідомлення: менеджер чи користувач
        LinearLayout bubble = cardLayout(manager ? Color.WHITE : LIGHT_BLUE, BORDER, 18);
        // створює «бульбашку» повідомлення
        bubble.addView(tv(msg[0], 13, manager ? BLUE : DARK, Typeface.BOLD));
        // показує автора повідомлення

        bubble.addView(tv(msg[1], 15, TEXT, Typeface.NORMAL));
        // показує текст повідомлення
    }

    EditText input = input("Введіть повідомлення", "");
    // поле для нового повідомлення

    Button send = button("Надіслати", BLUE, Color.WHITE);
    // кнопка відправки повідомлення

    send.setOnClickListener(v -> {
        dbHelper.addUserMessage(currentUserId, input.getText().toString().trim());
        // зберігає повідомлення у таблицю chat_messages

        showChat(); // оновлює чат після відправлення
    });
}

```

Рис. 4.9 Фрагмент коду з поясненнями до вікна «чат»

4.2 Демонстраційна версія застосунку та порядок відкриття проєкту в Android Studio

Розроблений програмний продукт є демонстраційною версією для захисту та опису у бакалаврській кваліфікаційній роботі. Його основне призначення полягає у демонстрації структури, логіки інтерфейсу та базових сценаріїв взаємодії користувача із мобільним застосунком страхової компанії.

У межах демонстраційної версії реалізовано основні екрани та сценарії: вхід користувача, перегляд головного екрана, вибір страхового продукту, оформлення заявки на автострахування, перегляд полісів, використання чату з менеджером та перегляд профілю користувача. Дані, що використовуються у прототипі, мають навчально-демонстраційний характер і не пов'язані з реальною інформаційною системою страхової компанії.

Для реального промислового впровадження такого застосунку необхідно додати серверну частину, повноцінну базу даних, механізми шифрування, захищену авторизацію, API страхової компанії, платіжний модуль, журналювання дій користувачів, систему резервного копіювання та повноцінне тестування безпеки. Також потрібно забезпечити відповідність вимогам щодо захисту персональних даних і правилам обробки платіжної інформації.

Порядок відкриття демонстраційного проєкту в Android Studio є таким:

1. Встановити Android Studio з офіційного сайту розробника та під час першого запуску погодитися із завантаженням необхідних компонентів Android SDK.

2. Розпакувати архів з проєктом InsuranceApp_AndroidStudio.zip у зручну папку на комп'ютері. Рекомендовано використовувати шлях без кириличних символів і пробілів, наприклад C:\Projects\InsuranceApp_AndroidStudio.

3. Відкрити Android Studio та обрати пункт меню File → Open. У вікні вибору потрібно вказати саме кореневу папку InsuranceApp_AndroidStudio, а не лише вкладену папку app.

4. Дочекатися завершення Gradle Sync. Під час першого відкриття Android Studio може додатково завантажити Gradle, Android Gradle Plugin або компоненти SDK.

5. Якщо середовище попросить обрати SDK, потрібно встановити актуальну версію Android SDK і підтвердити використання рекомендованої версії.

6. Для запуску можна створити вбудований емулятор через Device Manager або підключити фізичний Android-пристрій із увімкненим режимом розробника та USB debugging.

7. Після вибору емулятора або пристрою потрібно натиснути кнопку Run. Android Studio збере проєкт, встановить застосунок на вибраний пристрій і відкриє стартовий екран.

8. Під час демонстрації рекомендується послідовно перевірити основні сценарії: вхід у застосунок, перехід на головний екран, вибір страхового

продукту, оформлення автострахування, перегляд полісів, відкриття чату та профілю користувача.

9. У разі помилки збірки слід виконати File → Sync Project with Gradle Files, потім Build → Clean Project і Build → Rebuild Project. Також потрібно перевірити, чи встановлений Android SDK та чи правильно обрана версія Gradle JDK у налаштуваннях Android Studio.

Таким чином, підготовлений Android-проект може бути використаний як навчальний прототип для демонстрації результатів бакалаврської кваліфікаційної роботи, а також як основа для подальшого розширення функціоналу до рівня повноцінного промислового мобільного застосунку.

Висновки до розділу

У четвертому розділі дипломної роботи було реалізовано основні функціональні можливості мобільного застосунку для страхової компанії, пов'язані з наданням страхових послуг, оформленням полісів та взаємодією користувача із системою. У процесі розробки створено сторінки вибору страхових продуктів, оформлення автострахування, страхування життя, медичного страхування, а також реалізовано функціонал перегляду полісів і подання заявок.

Під час розробки застосунку використано середовище Android Studio, мову програмування Java, Android SDK та систему збірки Gradle. Для збереження та обробки інформації передбачено використання SQLite та Firebase, що забезпечує зручне керування даними користувачів, страховими полісами та заявками.

У межах реалізації страхових продуктів було створено інтуїтивно зрозумілий інтерфейс користувача, який дозволяє переглядати доступні страхові послуги, ознайомлюватися з умовами страхування, розраховувати вартість полісів та оформлювати заявки без необхідності відвідування офісу страхової компанії. Реалізовані екрани автострахування, страхування життя та медичного страхування демонструють можливість адаптації застосунку до різних типів страхових продуктів.

Також у розділі реалізовано функціонал чату між користувачем та працівником страхової компанії, що забезпечує оперативну комунікацію, підвищує рівень підтримки клієнтів та покращує взаємодію між сторонами. Описані фрагменти програмного коду підтверджують практичну реалізацію основних елементів системи та демонструють логіку роботи застосунку.

Отримані результати свідчать про те, що розроблений мобільний застосунок може бути використаний як основа для створення сучасного цифрового сервісу страхової компанії, спрямованого на автоматизацію бізнес-процесів, підвищення якості обслуговування клієнтів та розвиток електронного страхування.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

1. У процесі виконання бакалаврської кваліфікаційної роботи було проведено комплексне дослідження особливостей проєктування та реалізації мобільних програмних застосунків для споживачів послуг страхової компанії. Актуальність обраної тематики обумовлена активними процесами цифрової трансформації страхового ринку, зростанням попиту на дистанційне обслуговування клієнтів та необхідністю забезпечення швидкого доступу до страхових послуг у мобільному середовищі.

2. У ході дослідження було проаналізовано сучасний стан розвитку мобільних страхових сервісів в Україні та світі, визначено основні тенденції розвитку цифрових страхових платформ, а також досліджено функціональні можливості сучасних мобільних застосунків страхових компаній. Встановлено, що ключовими вимогами до мобільних страхових систем є забезпечення інформаційної безпеки, підтримка дистанційної взаємодії з клієнтами, зручність користувацького інтерфейсу, інтеграція з електронними сервісами та автоматизація процесів оформлення страхових послуг.

3. У результаті виконання роботи було досягнуто поставленої мети, яка полягала у розробці мобільного програмного застосунку для споживачів послуг страхової компанії, що забезпечує доступ до основних страхових сервісів, підтримує взаємодію користувача із системою та реалізує базові механізми цифрового обслуговування клієнтів.

4. У межах програмної реалізації було створено демонстраційний прототип мобільного застосунку, у якому реалізовано механізми авторизації та аутентифікації користувачів, особистий кабінет клієнта, модулі перегляду страхових продуктів, сторінки оформлення страхування автомобіля, життя та медичного страхування, а також функціонал комунікації користувачів із представниками страхової компанії через чат. Розроблений застосунок забезпечує базову навігацію між функціональними модулями та підтримує основні сценарії взаємодії користувача із системою.

5. У процесі тестування було перевірено працездатність основних функціональних модулів застосунку, стабільність його роботи на різних мобільних пристроях та коректність виконання користувацьких сценаріїв. Результати тестування підтвердили можливість використання розробленого програмного продукту як демонстраційної моделі мобільної системи для страхових компаній. Аналіз отриманих результатів засвідчив, що реалізований застосунок забезпечує зручний доступ користувачів до страхових послуг та сприяє підвищенню ефективності цифрової взаємодії між клієнтом і страховою компанією.

6. Практичне значення одержаних результатів полягає у створенні демонстраційного програмного рішення, яке може бути використане як основа для подальшого розширення функціональних можливостей мобільної страхової системи та інтеграції із серверною інфраструктурою страхової компанії. Запропоновані підходи до організації інтерфейсу, структури програмного застосунку та реалізації функціональних модулів можуть бути використані під час створення сучасних цифрових сервісів у сфері страхування.

7. Перспективними напрямками подальшого розвитку програмного застосунку є:

- інтеграція із серверною частиною та зовнішніми інформаційними системами;
- реалізація повноцінної системи керування базою даних користувачів і страхових полісів;
- підключення платіжних сервісів та електронних платіжних систем;
- впровадження push-сповіщень для інформування користувачів;
- реалізація біометричної автентифікації;
- інтеграція з державними електронними сервісами;
- використання технологій штучного інтелекту для автоматизації підтримки клієнтів;
- розширення функціоналу для автоматизованої обробки страхових випадків та фотофіксації пошкоджень.

Таким чином, у бакалаврській кваліфікаційній роботі виконано теоретичне обґрунтування та практичну реалізацію мобільного програмного застосунку для споживачів послуг страхової компанії. Отримані результати підтверджують доцільність використання мобільних технологій у сфері страхування та створюють основу для подальшого розвитку цифрових страхових сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ічанська Н., Улько С. Основні аспекти створення мобільних додатків та вибір інструментів їх розробки // Системи управління, навігації та зв'язку. 2020. № 1. С. 74–78. DOI: 10.26906/SUNZ.2020.1.074.
2. Бондаренко В. Технологічний та соціокомунікаційний аспекти становлення та розвитку мобільних сервісів в Україні // Український журнал з бібліотекознавства та інформаційних наук. 2019. № 4. С. 45–53. DOI: 10.31866/2616-7654.4.2019.187824.
3. Біганич В., Шуста В., Євич М., Сусла А. Розвиток мобільних додатків та їх роль в сучасних електронних комунікаціях // Herald of Khmelnytskyi National University. Technical Sciences. 2025. № 347(3).
4. Аронов А. О., Миронюк М. Ю., Бажан Т. О. Розвиток мобільних та web-додатків в контексті цифрової трансформації України на прикладі мобільного додатку «Дія» // Телекомунікаційні та інформаційні технології. 2022. DOI: 10.31673/2412-4338.2022.046373.
5. Цимбалістий В. В., Сілагін О. В. Особливості використання React Native для розробки мобільних додатків // Матеріали конференції ВНТУ. 2020.
6. Каневський М., Захарченко С. Особливості моніторингу клієнтської частини мобільних додатків // Інформаційні технології та комп'ютерна інженерія. 2021. № 3. С. 22–26. DOI: 10.31649/1999-9941-2021-52-3-22-26.
7. Єрешко Ю. О., Черненко А. Ю. Пошук оптимальної моделі монетизації мобільних додатків // Економічний вісник НТУУ «КПІ». 2017. № 14.
8. Hotline.finance — сервіс онлайн-страхування. Офіційний сайт. URL: <https://hotline.finance/ua/> (дата звернення: 11.05.2026).
9. Android Developers. Meet Android Studio. URL: <https://developer.android.com> (дата звернення: 10.04.2026).
10. Burrows J. Android Gradle Java App Template // GitHub. URL: <https://github.com/jaredsburrows/android-gradle-java-app-template> (дата звернення: 11.04.2026).

- 11.CodePath. Using Android Studio // Android Cliffnotes. URL: <https://guides.codepath.com/android/Using-Android-Studio> (дата звернення: 11.04.2026).
- 12.Java Code Geeks. Build your First Android App using Android Studio. URL: <https://examples.javacodegeeks.com/android/first-android-app-using-android-studio/> (дата звернення: 11.03.2026).
- 13.Android Developers. Android Studio Overview. URL: <https://developer.android.com/studio> (дата звернення: 11.05.2026).
- 14.Google. Android SDK Platform Tools. URL: <https://developer.android.com/studio/releases/platform-tools> (дата звернення: 11.05.2026).
- 15.Flutter Documentation. Flutter for Mobile Development. URL: <https://docs.flutter.dev> (дата звернення: 11.05.2026).
- 16.Берко А. Ю., Верес О. М. Організація баз даних та знань : навч. посіб. Львів : Магнолія 2006, 2021. 440 с.
- 17.Білушак Т. М., Пасічник В. В. Проєктування інформаційних систем : навч. посіб. Львів : Новий Світ–2000, 2022. 380 с.
- 18.Бондаренко М. Ф., Лисенко С. М. Програмування мобільних пристроїв на платформі Android : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 256 с.
- 19.Гужва В. М. Інформаційні системи і технології на підприємствах : навч. посіб. Київ : КНЕУ, 2020. 400 с.
- 20.Івахненко С. В. Інформаційні технології в організації бізнесу : навч. посіб. Київ : КНЕУ, 2021. 343 с.
- 21.Коваленко В. В., Кузнєцова О. В. Страхові послуги : навч. посіб. Київ : Центр учбової літератури, 2020. 512 с.
- 22.Кучерявий А. О. Android-програмування для мобільних пристроїв : практикум. Львів : Видавництво Львівської політехніки, 2022. 280 с.
- 23.Морзе Н. В., Вембер В. П. Основи розробки мобільних застосунків : навч. посіб. Київ : Університет «Україна», 2021. 310 с.

- 24.Олійник А. В., Субботін С. О. Інформаційні системи та технології у фінансових установах : навч. посіб. Запоріжжя : ЗНТУ, 2020. 365 с.
- 25.Плескач В. Л., Затонацька Т. Г. Інформаційні системи і технології на підприємствах : підручник. Київ : Знання, 2021. 718 с.
- 26.Про страхування : Закон України від 18.11.2021 № 1909-IX. URL: <https://zakon.rada.gov.ua> (дата звернення: 11.05.2026).
- 27.Руденко В. Д. Проєктування мобільних інформаційних систем : навч. посіб. Київ : Каравела, 2022. 296 с.
- 28.Ситник В. Ф. Основи інформаційних систем : навч. посіб. Київ : КНЕУ, 2020. 420 с.
- 29.Федорченко А. В. Цифровізація страхового ринку України в умовах розвитку інформаційних технологій // Економіка та держава. 2023. № 5. С. 87–92.

ДОДАТОК А

Реалізація програмного рішення мобільного застосунку для реалізації страхових продуктів

Текст програми

Текст файлу сторінки Polisapplication

```
package com.example.insuranceapp;

import android.app.Activity;
import android.graphics.Color;
import android.graphics.Typeface;
import android.graphics.drawable.GradientDrawable;
import android.os.Bundle;
import android.text.InputType;
import android.view.Gravity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.GridLayout;
import android.widget.LinearLayout;
import android.widget.ScrollView;
import android.widget.TextView;
import android.widget.Toast;

import java.util.List;
import java.util.Locale;

public class MainActivity extends Activity {
```

```

private LinearLayout root;
private InsuranceDbHelper dbHelper;
private long currentUserId = 1L;
private String currentScreen = "Головна";

```

```

private final int BLUE = Color.rgb(13, 92, 230);
private final int BLUE_DARK = Color.rgb(4, 70, 190);
private final int LIGHT_BLUE = Color.rgb(237, 246, 255);
private final int BG = Color.rgb(246, 250, 255);
private final int DARK = Color.rgb(20, 28, 45);
private final int TEXT = Color.rgb(55, 65, 86);
private final int MUTED = Color.rgb(107, 118, 140);
private final int BORDER = Color.rgb(224, 234, 247);
private final int GREEN = Color.rgb(34, 197, 94);
private final int ORANGE = Color.rgb(245, 158, 11);
private final int RED = Color.rgb(239, 68, 68);

```

```

@Override

```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    dbHelper = new InsuranceDbHelper(this);
    dbHelper.getReadableDatabase();
    showLogin();
}

```

```

private int dp(int value) {
    return (int) (value * getResources().getDisplayMetrics().density + 0.5f);
}

```

```

private GradientDrawable bg(int color, int radiusDp) {

```

```

        GradientDrawable d = new GradientDrawable();
        d.setColor(color);
        d.setCornerRadius(dp(radiusDp));
        return d;
    }

```

```

    private GradientDrawable borderBg(int color, int borderColor, int radiusDp,
int strokeDp) {
        GradientDrawable d = bg(color, radiusDp);
        d.setStroke(dp(strokeDp), borderColor);
        return d;
    }

```

```

    private GradientDrawable blueGradient() {
        GradientDrawable d = new GradientDrawable(
            GradientDrawable.Orientation.LEFT_RIGHT,
            new int[] {Color.rgb(10, 101, 245), Color.rgb(2, 71, 200)}
        );
        d.setCornerRadius(dp(20));
        return d;
    }

```

```

    private TextView tv(String text, float sp, int color, int style) {
        TextView v = new TextView(this);
        v.setText(text);
        v.setTextSize(sp);
        v.setTextColor(color);
        v.setTypeface(Typeface.DEFAULT, style);
        v.setIncludeFontPadding(true);
        return v;
    }

```

```
}
```

```
private LinearLayout vbox() {
    LinearLayout l = new LinearLayout(this);
    l.setOrientation(LinearLayout.VERTICAL);
    return l;
}
```

```
private LinearLayout hbox() {
    LinearLayout l = new LinearLayout(this);
    l.setOrientation(LinearLayout.HORIZONTAL);
    l.setGravity(Gravity.CENTER_VERTICAL);
    return l;
}
```

```
private void base() {
    root = new LinearLayout(this);
    root.setOrientation(LinearLayout.VERTICAL);
    root.setBackgroundColor(BG);
    setContentView(root);
}
```

```
private Button button(String text, int bgColor, int textColor) {
    Button b = new Button(this);
    b.setText(text);
    b.setTextSize(15);
    b.setTextColor(textColor);
    b.setTypeface(Typeface.DEFAULT_BOLD);
    b.setAllCaps(false);
    b.setPadding(dp(8), dp(10), dp(8), dp(10));
}
```

```

        b.setBackground(borderBg(bgColor, bgColor, 14, 0));
        return b;
    }

```

```

private TextView inputLabel(String text) {
    TextView v = tv(text, 13, TEXT, Typeface.BOLD);
    v.setPadding(0, dp(10), 0, dp(4));
    return v;
}

```

```

private EditText input(String hint, String value) {
    EditText e = new EditText(this);
    e.setHint(hint);
    e.setText(value);
    e.setTextSize(15);
    e.setSingleLine(true);
    e.setPadding(dp(14), 0, dp(14), 0);
    e.setBackground(borderBg(Color.WHITE, BORDER, 14, 1));
    LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT, dp(52)
    );
    lp.setMargins(0, 0, 0, dp(4));
    e.setLayoutParams(lp);
    return e;
}

```

```

private TextView padBottom(String text, int bottom) {
    TextView v = tv(text, 15, MUTED, Typeface.NORMAL);
    v.setPadding(0, dp(4), 0, dp(bottom));
    return v;
}

```

```
}
```

```
private LinearLayout contentBase() {
    LinearLayout c = vbox();
    c.setPadding(dp(18), dp(16), dp(18), dp(18));
    return c;
}
```

```
private void setScreen(LinearLayout content, String screenName) {
    currentScreen = screenName;
    root.removeAllViews();

    ScrollView scroll = new ScrollView(this);
    scroll.setFillViewport(false);
    scroll.addView(content);
    root.addView(scroll, new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT, 0, 1
    ));

    addBottomNav();
}
```

```
private LinearLayout appHeader(String title, boolean showBell) {
    LinearLayout header = hbox();
    header.setPadding(0, dp(6), 0, dp(20));

    TextView shield = tv("✓", 26, Color.WHITE, Typeface.BOLD);
    shield.setGravity(Gravity.CENTER);
    shield.setBackground(borderBg(BLUE, BLUE, 16, 0));
}
```

```

        LinearLayout.LayoutParams shieldLp = new
LinearLayout.LayoutParams(dp(54), dp(54));
        shieldLp.setMargins(0, 0, dp(12), 0);
        header.addView(shield, shieldLp);

        TextView name = tv(title, 24, BLUE_DARK, Typeface.BOLD);
        header.addView(name, new LinearLayout.LayoutParams(0,
LinearLayout.LayoutParams.WRAP_CONTENT, 1));

        if (showBell) {
            TextView bell = tv("🔔", 24, Color.rgb(55, 65, 81),
Typeface.NORMAL);
            bell.setGravity(Gravity.CENTER);
            header.addView(bell, new LinearLayout.LayoutParams(dp(42),
dp(42)));
        }

        return header;
    }

    private void addBottomNav() {
        LinearLayout nav = new LinearLayout(this);
        nav.setOrientation(LinearLayout.HORIZONTAL);
        nav.setGravity(Gravity.CENTER);
        nav.setPadding(dp(6), dp(6), dp(6), dp(6));
        nav.setBackgroundColor(Color.WHITE);

        addNavItem(nav, "🏠", "Головна");
        addNavItem(nav, "👮", "Поліси");
        addNavItem(nav, "📄", "Заявки");
    }

```

```
addNavItem(nav, "○", "Чат");
addNavItem(nav, "△", "Профіль");
```

```
root.addView(nav, new LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.MATCH_PARENT, dp(74)
));
}
```

```
private void addNavItem(LinearLayout nav, String icon, String label) {
    boolean active = currentScreen.equals(label);
    LinearLayout item = vbox();
    item.setGravity(Gravity.CENTER);
    item.setPadding(0, dp(4), 0, dp(2));
```

```
    TextView i = tv(icon, 22, active ? BLUE : Color.rgb(114, 126, 151),
Typeface.BOLD);
    i.setGravity(Gravity.CENTER);
```

```
    TextView t = tv(label, 12, active ? BLUE : Color.rgb(114, 126, 151),
active ? Typeface.BOLD : Typeface.NORMAL);
    t.setGravity(Gravity.CENTER);
```

```
    item.addView(i);
    item.addView(t);
```

```
    item.setOnClickListener(v -> {
        switch (label) {
            case "Головна": showDashboard(); break;
            case "Поліси": showPolicies(); break;
            case "Заявки": showApplications(); break;
```

```

        case "Чат": showChat(); break;
        case "Профіль": showProfile(); break;
    }
});

```

```

        nav.addView(item, new LinearLayout.LayoutParams(0,
LinearLayout.LayoutParams.MATCH_PARENT, 1));
    }

```

```

private LinearLayout cardLayout(int color, int border, int radius) {
    LinearLayout card = vbox();
    card.setPadding(dp(16), dp(14), dp(16), dp(14));
    card.setBackground(borderBg(color, border, radius, 1));
    LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.WRAP_CONTENT
    );
    lp.setMargins(0, dp(8), 0, dp(8));
    card.setLayoutParams(lp);
    return card;
}

```

```

private LinearLayout cardText(String title, String text) {
    LinearLayout card = cardLayout(Color.WHITE, BORDER, 18);
    card.addView(tv(title, 17, DARK, Typeface.BOLD));
    TextView value = tv(text, 15, TEXT, Typeface.NORMAL);
    value.setPadding(0, dp(5), 0, 0);
    card.addView(value);
    return card;
}

```

```

public void showLogin() {
    base();
    LinearLayout c = contentBase();
    c.setGravity(Gravity.CENTER_HORIZONTAL);

    c.addView(appHeader("Страховий сервіс", false));
    c.addView(tv("Вхід до особистого кабінету", 26, DARK,
Typeface.BOLD));
    c.addView(padBottom("Керуйте полісами, заявками та платежами
онлайн", 18));

    LinearLayout panel = cardLayout(Color.WHITE, BORDER, 22);

    panel.addView(inputLabel("Електронна пошта"));
    EditText email = input("Електронна пошта", "client@example.com");
    panel.addView(email);

    panel.addView(inputLabel("Пароль"));
    EditText password = input("Пароль", "123456");
    password.setInputType(InputType.TYPE_CLASS_TEXT |
InputType.TYPE_TEXT_VARIATION_PASSWORD);
    panel.addView(password);

    Button login = button("Увійти", BLUE, Color.WHITE);
    LinearLayout.LayoutParams loginLp = new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT, dp(52)
    );
    loginLp.setMargins(0, dp(14), 0, dp(6));
    panel.addView(login, loginLp);

```

```

login.setOnClickListener(v -> {
    long userId = dbHelper.authenticate(
        email.getText().toString().trim(),
        password.getText().toString().trim()
    );

    if (userId > 0) {
        currentUserId = userId;
        showDashboard();
    } else {
        Toast.makeText(this, "Невірний email або пароль",
Toast.LENGTH_LONG).show();
    }
});

c.addView(panel);

TextView hint = tv("Тестовий вхід: client@example.com / 123456", 13,
MUTED, Typeface.NORMAL);
hint.setGravity(Gravity.CENTER);
hint.setPadding(0, dp(12), 0, 0);
c.addView(hint);

root.addView(c, new LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.MATCH_PARENT,
    LinearLayout.LayoutParams.MATCH_PARENT
));
}

```

```

public void showDashboard() {
    base();
    LinearLayout c = contentBase();

    c.addView(appHeader("Страховий сервіс", true));

    int activePolicies = dbHelper.getActivePoliciesCount(currentUserId);
    double coverage = dbHelper.getTotalCoverage(currentUserId);

    c.addView(tv("Вітаємо, Владиславе", 28, DARK, Typeface.BOLD));
    c.addView(padBottom("Ми піклуємося про ваш спокій і впевненість",
16));
    c.addView(summaryCard(activePolicies, coverage));

    TextView quickTitle = tv("Швидкі дії", 20, DARK, Typeface.BOLD);
    quickTitle.setPadding(0, dp(22), 0, dp(8));
    c.addView(quickTitle);

    GridLayout quick = new GridLayout(this);
    quick.setColumnCount(4);
    addQuickAction(quick, "■", "Подати\нзаявку", v -> showProducts());
    addQuickAction(quick, "☎", "Чат з\нменеджером", v -> showChat());
    addQuickAction(quick, "✓", "Мої\нполіси", v -> showPolicies());
    addQuickAction(quick, "□", "Оплатити\нполіс", v ->
showPaymentScreen());
    c.addView(quick);

    TextView servicesTitle = tv("Наші послуги", 20, DARK,
Typeface.BOLD);

```

```

servicesTitle.setPadding(0, dp(18), 0, dp(8));
c.addView(servicesTitle);

GridLayout services = new GridLayout(this);
services.setColumnCount(2);

addServiceCard(services, "─", "Автострахування", "КАСКО, ОСЦПВ
та інші програми", v -> showAutoInsurance());
addServiceCard(services, "♥", "Медичне страхування", "Захист
здоров'я для вас і близьких", v -> showMedicalInsurance());
addServiceCard(services, "●", "Страхування життя", "Фінансова
безпека для родини", v -> showLifeInsurance());
addServiceCard(services, "✈", "Страхування подорожей", "Поліс для
поїздок за кордон", v -> showTravelInsurance());
addServiceCard(services, "≡", "Мої поліси", "Перегляд та управління
полісами", v -> showPolicies());
addServiceCard(services, "💬", "Чат з менеджером", "Отримайте
допомогу швидко та зручно", v -> showChat());
c.addView(services);

setScreen(c, "Головна");
}

private LinearLayout summaryCard(int activePolicies, double coverage) {
    LinearLayout card = vbox();
    card.setPadding(dp(16), dp(16), dp(16), dp(16));
    card.setBackground(blueGradient());

    LinearLayout top = hbox();

    TextView bigIcon = tv("✓", 34, BLUE, Typeface.BOLD);

```

```

        bigIcon.setGravity(Gravity.CENTER);
        bigIcon.setBackground(bg(Color.WHITE, 42));
        LinearLayout.LayoutParams iconLp = new
LinearLayout.LayoutParams(dp(78), dp(78));
        iconLp.setMargins(0, 0, dp(16), 0);
        top.addView(bigIcon, iconLp);

        LinearLayout left = vbox();
        left.addView(tv("Ваші активні поліси", 15, Color.WHITE,
Typeface.NORMAL));
        left.addView(tv(String.valueOf(activePolicies), 34, Color.WHITE,
Typeface.BOLD));
        left.addView(tv("Поліси в дії", 15, Color.rgb(218, 234, 255),
Typeface.NORMAL));
        top.addView(left, new LinearLayout.LayoutParams(0,
LinearLayout.LayoutParams.WRAP_CONTENT, 1));

        LinearLayout right = vbox();
        right.setGravity(Gravity.RIGHT);
        TextView status = tv("● Усе гаразд", 13, Color.rgb(12, 95, 70),
Typeface.BOLD);
        status.setGravity(Gravity.CENTER);
        status.setPadding(dp(8), dp(4), dp(8), dp(4));
        status.setBackground(bg(Color.WHITE, 20));
        right.addView(status);

        TextView amount = tv("Загальна сума\нпокриття", 13, Color.rgb(218,
234, 255), Typeface.NORMAL);
        amount.setGravity(Gravity.RIGHT);
        amount.setPadding(0, dp(12), 0, dp(2));

```

```
right.addView(amount);
```

```
TextView sum = tv(String.format(Locale.US, "%.0f ¢", coverage), 22,
Color.WHITE, Typeface.BOLD);
```

```
sum.setGravity(Gravity.RIGHT);
```

```
right.addView(sum);
```

```
top.addView(right);
```

```
card.addView(top);
```

```
View line = new View(this);
```

```
line.setBackgroundColor(Color.argb(90, 255, 255, 255));
```

```
LinearLayout.LayoutParams lineLp = new LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.MATCH_PARENT, dp(1)
```

```
);
```

```
lineLp.setMargins(0, dp(14), 0, dp(14));
```

```
card.addView(line, lineLp);
```

```
LinearLayout bottom = hbox();
```

```
TextView cal = tv("■", 28, Color.WHITE, Typeface.BOLD);
```

```
cal.setGravity(Gravity.CENTER);
```

```
LinearLayout.LayoutParams calLp = new
LinearLayout.LayoutParams(dp(48), dp(48));
```

```
calLp.setMargins(0, 0, dp(12), 0);
```

```
bottom.addView(cal, calLp);
```

```
LinearLayout txt = vbox();
```

```
txt.addView(tv("Найближче завершения", 14, Color.WHITE,
Typeface.NORMAL));
```

```
txt.addView(tv("ОСЦПБ (auto)", 18, Color.WHITE, Typeface.BOLD));
```

```
bottom.addView(txt, new LinearLayout.LayoutParams(0,
LinearLayout.LayoutParams.WRAP_CONTENT, 1));
```

```
TextView date = tv("12.06.2025  >", 16, Color.WHITE, Typeface.BOLD);
bottom.addView(date);
card.addView(bottom);
```

```
LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.MATCH_PARENT,
    LinearLayout.LayoutParams.WRAP_CONTENT
);
lp.setMargins(0, dp(12), 0, dp(8));
card.setLayoutParams(lp);
return card;
}
```

```
private void addQuickAction(GridLayout grid, String icon, String label,
View.OnClickListener listener) {
    LinearLayout item = vbox();
    item.setGravity(Gravity.CENTER);
    item.setPadding(dp(8), dp(10), dp(8), dp(10));
    item.setBackground(borderBg(Color.WHITE, BORDER, 18, 1));
    item.setOnClickListener(listener);

    TextView i = tv(icon, 27, BLUE, Typeface.BOLD);
    i.setGravity(Gravity.CENTER);
    item.addView(i);

    TextView t = tv(label, 13, TEXT, Typeface.NORMAL);
    t.setGravity(Gravity.CENTER);
```

```
t.setPadding(0, dp(6), 0, 0);
item.addView(t);
```

```
GridLayout.LayoutParams lp = new GridLayout.LayoutParams();
lp.width = 0;
lp.height = dp(110);
lp.columnSpec = GridLayout.spec(GridLayout.UNDEFINED, 1f);
lp.setMargins(dp(4), dp(4), dp(4), dp(4));
grid.addView(item, lp);
}
```

```
private void addServiceCard(GridLayout grid, String icon, String title,
String desc, View.OnClickListener listener) {
```

```
    LinearLayout item = hbox();
    item.setGravity(Gravity.CENTER_VERTICAL);
    item.setPadding(dp(10), dp(12), dp(10), dp(12));
    item.setBackground(borderBg(Color.WHITE, BORDER, 18, 1));
    item.setOnClickListener(listener);
```

```
    TextView ic = tv(icon, 27, Color.WHITE, Typeface.BOLD);
    ic.setGravity(Gravity.CENTER);
    ic.setBackground(bg(LIGHT_BLUE, 14));
    ic.setTextColor(BLUE);
```

```
    LinearLayout.LayoutParams icLp = new
LinearLayout.LayoutParams(dp(58), dp(58));
    icLp.setMargins(0, 0, dp(10), 0);
    item.addView(ic, icLp);
```

```
    LinearLayout text = vbox();
    text.addView(tv(title, 14, DARK, Typeface.BOLD));
```

```

        TextView d = tv(desc, 11.5f, MUTED, Typeface.NORMAL);
        d.setPadding(0, dp(2), 0, 0);
        text.addView(d);
        item.addView(text, new LinearLayout.LayoutParams(0,
        LinearLayout.LayoutParams.WRAP_CONTENT, 1));

```

```

        TextView arrow = tv(">", 28, Color.rgb(126, 139, 160),
        Typeface.NORMAL);
        item.addView(arrow);

```

```

        GridLayout.LayoutParams lp = new GridLayout.LayoutParams();
        lp.width = 0;
        lp.height = dp(96);
        lp.columnSpec = GridLayout.spec(GridLayout.UNDEFINED, 1f);
        lp.setMargins(dp(4), dp(6), dp(4), dp(6));
        grid.addView(item, lp);
    }

```

```

public void showProducts() {
    base();
    LinearLayout c = contentBase();
    c.addView(appHeader("Страховий сервіс", true));
    c.addView(tv("Наші послуги", 26, DARK, Typeface.BOLD));
    c.addView(padBottom("Оберіть потрібний страховий продукт", 12));

    c.addView(productCard("Автострахування", "ОСЦПВ, КАСКО, зелена
картка", "від 3 200 ₪", v -> showAutoInsurance()));
    c.addView(productCard("Медичне страхування", "Лікування,
консультації, невідкладна допомога", "від 4 500 ₪", v ->
showMedicalInsurance()));
}

```

```

        c.addView(productCard("Страхування життя", "Фінансовий захист
клієнта та родини", "від 6 200 ₪", v -> showLifeInsurance()));

        c.addView(productCard("Страхування подорожей", "Медичний захист
за кордоном", "від 950 ₪", v -> showTravelInsurance()));

        setScreen(c, "Головна");
    }

    private LinearLayout productCard(String title, String desc, String price,
View.OnClickListener listener) {
        LinearLayout card = cardLayout(Color.WHITE, BORDER, 18);
        card.addView(tv(title, 18, DARK, Typeface.BOLD));
        card.addView(padBottom(desc, 6));
        card.addView(tv(price, 16, BLUE, Typeface.BOLD));
        card.setOnClickListener(listener);
        return card;
    }

    public void showAutoInsurance() {
        base();
        LinearLayout c = contentBase();
        c.addView(appHeader("Страховий сервіс", true));
        c.addView(tv("Автострахування", 26, DARK, Typeface.BOLD));
        c.addView(padBottom("ОСЦПВ, КАСКО та інші програми для
автомобіля", 14));

        c.addView(cardText("Що входить", "ОСЦПВ • КАСКО • зелена картка
• допомога при ДТП • консультація менеджера"));

        LinearLayout panel = cardLayout(Color.WHITE, BORDER, 18);

```

```
panel.addView(inputLabel("Марка авто"));
EditText brand = input("Марка авто", "Toyota Camry");
panel.addView(brand);
```

```
panel.addView(inputLabel("Рік випуску"));
EditText year = input("Рік випуску", "2018");
year.setInputType(InputType.TYPE_CLASS_NUMBER);
panel.addView(year);
```

```
panel.addView(inputLabel("Номерний знак"));
EditText plate = input("Номерний знак", "AA1234AB");
panel.addView(plate);
```

```
panel.addView(inputLabel("Тип страхування"));
EditText type = input("Тип страхування", "ОСЦПВ");
panel.addView(type);
```

```
panel.addView(priceBox("Попередній розрахунок", "Орієнтовна  
вартість: 3 200 €"));
```

```
Button send = button("Подати заявку", BLUE, Color.WHITE);
send.setOnClickListener(v -> {
    int parsedYear = parseInt(year.getText().toString().trim());
    long id = dbHelper.insertApplication(
        currentUserId,
        1,
        brand.getText().toString().trim(),
        parsedYear,
        plate.getText().toString().trim(),
        type.getText().toString().trim(),
```

3200

```

    );
    Toast.makeText(this, "Заявку №" + id + " на автострахування
подано", Toast.LENGTH_LONG).show();
    showApplications();
});
panel.addView(send, new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
dp(52)));

c.addView(panel);
setScreen(c, "Заявки");
}

public void showMedicalInsurance() {
    base();
    LinearLayout c = contentBase();
    c.addView(appHeader("Страховий сервіс", true));
    c.addView(tv("Медичне страхування", 26, DARK, Typeface.BOLD));
    c.addView(padBottom("Захист здоров'я для вас і ваших близьких",
14));

    c.addView(cardText("Що покриває поліс", "Консультації лікарів •
діагностика • стаціонар • невідкладна допомога • медикаменти за програмою"));
    c.addView(cardText("Доступні пакети", "Базовий — 4 500 ₪
\nСтандарт — 8 900 ₪\nПреміум — 14 500 ₪"));

    LinearLayout panel = cardLayout(Color.WHITE, BORDER, 18);
    panel.addView(inputLabel("ПІБ застрахованої особи"));
    EditText name = input("ПІБ", "Владислав Клієнт");

```

```
panel.addView(name);
```

```
panel.addView(inputLabel("Дата народження"));
EditText birth = input("Дата народження", "01.01.2000");
panel.addView(birth);
```

```
panel.addView(inputLabel("Телефон"));
EditText phone = input("Телефон", "+380501234567");
panel.addView(phone);
```

```
panel.addView(inputLabel("Пакет страхування"));
EditText plan = input("Пакет", "Стандарт");
panel.addView(plan);
```

```
panel.addView(priceBox("Попередній розрахунок", "Орієнтовна  
вартість: 8 900 ₴"));
```

```
Button send = button("Подати заявку на медичне страхування", BLUE,  
Color.WHITE);
```

```
send.setOnClickListener(v -> {
    long id = dbHelper.insertApplication(
        currentUserId,
        2,
        "Медичне страхування: " + name.getText().toString().trim(),
        0,
        phone.getText().toString().trim(),
        "Пакет: " + plan.getText().toString().trim(),
        8900
    );
});
```

```

        Toast.makeText(this, "Заявку №" + id + " на медичне страхування
        подано", Toast.LENGTH_LONG).show();

```

```

        showApplications();

```

```

    });

```

```

        panel.addView(send, new
        LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
        dp(52)));

```

```

        c.addView(panel);

```

```

        setScreen(c, "Заявки");

```

```

    }

```

```

public void showLifeInsurance() {

```

```

    base();

```

```

    LinearLayout c = contentBase();

```

```

    c.addView(appHeader("Страховий сервіс", true));

```

```

    c.addView(tv("Страхування життя", 26, DARK, Typeface.BOLD));

```

```

    c.addView(padBottom("Фінансова безпека для клієнта та родини",
    14));

```

```

        c.addView(cardText("Призначення", "Фінансовий захист родини •
        накопичення • підтримка у складних життєвих обставинах"));

```

```

        c.addView(cardText("Переваги", "Гнучка сума покриття • різні строки
        дії • індивідуальні умови договору"));

```

```

    LinearLayout panel = cardLayout(Color.WHITE, BORDER, 18);

```

```

    panel.addView(inputLabel("ПІБ застрахованої особи"));

```

```

    EditText name = input("ПІБ", "Владислав Клієнт");

```

```

    panel.addView(name);

```

```

panel.addView(inputLabel("Бажана сума покриття"));
EditText coverage = input("Сума покриття", "500000");
coverage.setInputType(InputType.TYPE_CLASS_NUMBER);
panel.addView(coverage);

panel.addView(inputLabel("Строк дії договору"));
EditText term = input("Строк", "12 місяців");
panel.addView(term);

panel.addView(priceBox("Попередній розрахунок", "Орієнтовна
вартість: 6 200 ₪"));

Button send = button("Подати заявку на страхування життя", BLUE,
Color.WHITE);
send.setOnClickListener(v -> {
    long id = dbHelper.insertApplication(
        currentUserId,
        3,
        "Страхування життя: " + name.getText().toString().trim(),
        0,
        "LIFE",
        "Покриття: " + coverage.getText().toString().trim() + " ₪, строк: "
+ term.getText().toString().trim(),
        6200
    );
    Toast.makeText(this, "Заявку №" + id + " на страхування життя
подано", Toast.LENGTH_LONG).show();
    showApplications();
});

```

```

        panel.addView(send, new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
dp(52)));

```

```

        c.addView(panel);
        setScreen(c, "Заявки");
    }

```

```

public void showTravelInsurance() {
    base();
    LinearLayout c = contentBase();
    c.addView(appHeader("Страховий сервіс", true));
    c.addView(tv("Страхування подорожей", 26, DARK,
Typeface.BOLD));
    c.addView(padBottom("Поліс для виїзду за кордон", 14));

    c.addView(cardText("Що покриває", "Медична допомога за кордоном
• багаж • скасування подорожі • підтримка 24/7"));

```

```

    LinearLayout panel = cardLayout(Color.WHITE, BORDER, 18);
    panel.addView(inputLabel("Країна подорожі"));
    EditText country = input("Країна", "Польща");
    panel.addView(country);

```

```

    panel.addView(inputLabel("Дата початку"));
    EditText start = input("Дата початку", "01.07.2026");
    panel.addView(start);

```

```

    panel.addView(inputLabel("Дата завершення"));
    EditText end = input("Дата завершення", "14.07.2026");

```

```

panel.addView(end);

panel.addView(priceBox("Попередній розрахунок", "Орієнтовна
вартість: 950 €"));

Button send = button("Подати заявку на travel-поліс", BLUE,
Color.WHITE);
send.setOnClickListener(v -> {
    long id = dbHelper.insertApplication(
        currentUserId,
        4,
        "Подорож: " + country.getText().toString().trim(),
        0,
        "TRAVEL",
        start.getText().toString().trim() + " — " +
end.getText().toString().trim(),
        950
    );
    Toast.makeText(this, "Заявку №" + id + " на страхування подорожей
подано", Toast.LENGTH_LONG).show();
    showApplications();
});
panel.addView(send, new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
dp(52)));

c.addView(panel);
setScreen(c, "Заявки");
}

```

```

private LinearLayout priceBox(String title, String amount) {
    LinearLayout calc = cardLayout(LIGHT_BLUE, Color.rgb(210, 230,
255), 16);
    calc.addView(tv(title, 15, BLUE, Typeface.BOLD));
    calc.addView(tv(amount, 20, BLUE, Typeface.BOLD));
    return calc;
}

private int parseInt(String value) {
    try {
        return Integer.parseInt(value);
    } catch (Exception e) {
        return 0;
    }
}

public void showApplications() {
    base();
    LinearLayout c = contentBase();
    c.addView(appHeader("Страховий сервіс", true));
    c.addView(tv("Заявки", 26, DARK, Typeface.BOLD));
    c.addView(padBottom("Відстежуйте статус поданих заявок", 14));

    c.addView(cardText("Остання заявка",
dbHelper.getLastApplicationInfo(currentUserId)));
    c.addView(cardText("Статус обробки", "Нова заявка передана
менеджеру. Очікуйте уточнення деталей."));

    Button create = button("Створити нову заявку", BLUE, Color.WHITE);
    create.setOnClickListener(v -> showProducts());
}

```

```

        c.addView(create, new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
dp(52)));

        setScreen(c, "Заявки");
    }

    public void showPolicies() {
        base();
        LinearLayout c = contentBase();
        c.addView(appHeader("Страховий сервіс", true));
        c.addView(tv("Мої поліси", 26, DARK, Typeface.BOLD));
        c.addView(padBottom("Перегляд та управління вашими полісами",
14));

        List<String[]> policies = dbHelper.getPolicies(currentUserId);
        for (String[] p : policies) {
            LinearLayout card = cardLayout(Color.WHITE, BORDER, 18);
            card.addView(tv(p[1], 18, DARK, Typeface.BOLD));
            card.addView(padBottom("№ " + p[0], 4));

            TextView status = tv(
                p[2],
                14,
                p[2].contains("Активний") ? GREEN : ORANGE,
                Typeface.BOLD
            );
            card.addView(status);

            card.addView(padBottom("Діє до: " + p[3], 2));

```

```

        card.addView(tv("Покриття: " + p[4] + " ₪", 14, TEXT,
Typeface.NORMAL));

        card.addView(tv("Вартість: " + p[5] + " ₪", 14, TEXT,
Typeface.NORMAL));

        Button pay = button("Оплатити / переглянути", LIGHT_BLUE,
BLUE);

        pay.setOnClickListener(v -> showPaymentScreen());

        card.addView(pay, new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
dp(48)));

        c.addView(card);
    }

    setScreen(c, "Поліси");
}

public void showPaymentScreen() {
    base();
    LinearLayout c = contentBase();
    c.addView(appHeader("Страховий сервіс", true));
    c.addView(tv("Оплата полісу", 26, DARK, Typeface.BOLD));
    c.addView(padBottom("Оберіть активний поліс і спосіб оплати", 14));

    c.addView(cardText("До оплати", "ОСЦПВ (auto)\nСума: 3 200 ₪
\nСпосіб: банківська картка"));

    Button pay = button("Оплатити", BLUE, Color.WHITE);

```

```

        pay.setOnClickListener(v -> Toast.makeText(this, "Оплату успішно
підтверджено", Toast.LENGTH_LONG).show());

        c.addView(pay, new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
dp(52)));

```

```

        setScreen(c, "Поліси");
    }

```

```

public void showChat() {
    base();
    LinearLayout c = contentBase();
    c.addView(appHeader("Страховий сервіс", true));
    c.addView(tv("Чат з менеджером", 26, DARK, Typeface.BOLD));
    c.addView(padBottom("Отримайте консультацію щодо послуг і
полісів", 14));

    List<String[]> messages = dbHelper.getChatMessages(currentUserId);
    for (String[] msg : messages) {
        boolean manager = msg[0].contains("Менеджер");
        LinearLayout bubble = cardLayout(
            manager ? Color.WHITE : LIGHT_BLUE,
            manager ? BORDER : Color.rgb(210, 230, 255),
            18
        );
        bubble.addView(tv(msg[0], 13, manager ? BLUE : DARK,
Typeface.BOLD));
        bubble.addView(tv(msg[1], 15, TEXT, Typeface.NORMAL));
        c.addView(bubble);
    }
}

```

```

EditText input = input("Введіть повідомлення", "");
c.addView(input);

Button send = button("Надіслати", BLUE, Color.WHITE);
send.setOnClickListener(v -> {
    String text = input.getText().toString().trim();
    if (text.length() == 0) {
        Toast.makeText(this, "Введіть повідомлення",
Toast.LENGTH_SHORT).show();
        return;
    }
    dbHelper.addUserMessage(currentUserId, text);
    Toast.makeText(this, "Повідомлення надіслано",
Toast.LENGTH_SHORT).show();
    showChat();
});
c.addView(send, new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
dp(52)));

setScreen(c, "Чат");
}

public void showProfile() {
    base();
    LinearLayout c = contentBase();
    c.addView(appHeader("Страховий сервіс", true));
    c.addView(tv("Профіль", 26, DARK, Typeface.BOLD));
    c.addView(padBottom("Особисті дані та налаштування сервісу", 14));
}

```

```

        c.addView(cardText("Клієнт",
dbHelper.getUserFullName(currentUserId) +
"\nclient@example.com\n+380501234567"));

        c.addView(cardText("Безпека", "PIN-код, Face ID, підтвердження
важливих дій"));

        c.addView(cardText("Сповіщення", "Нагадування про оплату, статус
заявок, оновлення полісів"));

        c.addView(cardText("Підтримка", "Онлайн-чат, FAQ, контакти
страхової компанії"));

        Button logout = button("Вийти", RED, Color.WHITE);
        logout.setOnClickListener(v -> showLogin());
        c.addView(logout, new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
dp(52)));

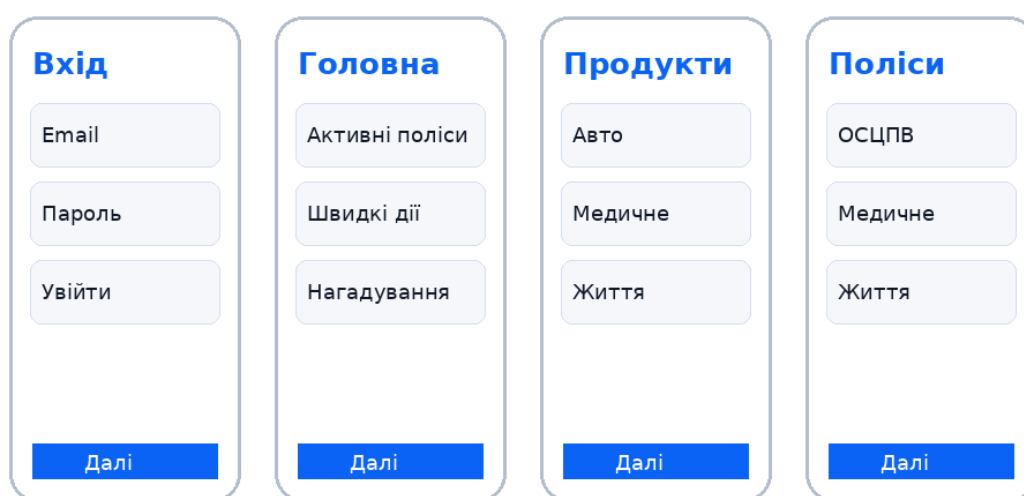
        setScreen(c, "Профіль");
    }
}

```

ДОДАТОК Б

Інструкція з відкриття демонстраційного Android Studio-проєкту

До роботи додано архів InsuranceApp_AndroidStudio.zip, який містить демонстраційний Android Studio-проєкт мобільного застосунку страхової компанії. Проєкт реалізовано мовою Java у середовищі Android Studio та містить базові екрани: авторизація, головна сторінка, страхові продукти, оформлення заявки, мої поліси, чат і профіль користувача.

Демонстраційні екрани Android-застосунку

Виконавець: Голота Владислав Сергійович

Рис Б.1 Демонстраційні екрани Android-застосунку

Склад проєкту:

- settings.gradle — підключення модуля app;
- build.gradle — налаштування Android Gradle Plugin;
- app/build.gradle — параметри застосунку: applicationId, minSdk, targetSdk, compileSdk;
- AndroidManifest.xml — опис головної Activity та параметрів застосунку;
- MainActivity.java — програмна реалізація всіх демонстраційних екранів;
- README.md — коротка інструкція для відкриття й запуску проєкту.

Порядок запуску проєкту в Android Studio:

1. Розпакувати архів InsuranceApp_AndroidStudio.zip у папку без кириличних символів і пробілів, наприклад C:\Projects\InsuranceApp_AndroidStudio.
2. Відкрити Android Studio та обрати File → Open.
3. У вікні вибору вказати кореневу папку InsuranceApp_AndroidStudio.
4. Дочекатися завершення Gradle Sync. За потреби дозволити Android Studio завантажити Android Gradle Plugin та компоненти SDK.
5. Відкрити Device Manager і створити емулятор Android або підключити фізичний смартфон з увімкненим USB debugging.
6. Натиснути Run. Після збірки застосунків встановиться на емулятор або смартфон.
7. Для демонстрації перевірити послідовно екрани: вхід, головна сторінка, продукти, заявка, поліси, чат, профіль.

Обмеження демонстраційної версії:

Це навчальний прототип для захисту й опису у бакалаврській роботі. У реальному промисловому застосунку потрібно додати серверну частину, повноцінну базу даних, шифрування, захищену авторизацію, API страхової компанії, платіжний модуль, push-сповіщення та повноцінне тестування безпеки.

ДОДАТОК В



ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ІІІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Голота Владислав Сергійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

Моя бакалаврська кваліфікаційна робота (проект) на тему «**Мобільний програмний застосунок для споживачів послуг страхової компанії**» є результатом моєї особистої праці.

1. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
2. Щодо використання інструментів ІІІ зазначаю, що (обрати необхідне):
 - [+] інструменти ІІІ (вказати назву: ChatGPT, Gemini, Grok.com були використані для:
 - [+] перекладу іншомовних джерел;
 - [+] стилістичного редагування та перевірки граматики;
 - [+] допомога у відлагодженні програмного коду;
 - [+] при узагальненні та редагуванні списку літератури.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«22» травня 2026 р.



(підпис)

Владислав Голота