Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Мобільний застосунок для керування клієнтськими записами та фінансовим обліком у сфері краси
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.230128.01.01.ПЗ

Виконав студент ІІІ курсу, група ІПЗс-23-1 _____ Ігор БАЛІЦЬКИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник ст. викладач _____ Ганна БЕДРАТЮК
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Юрій ФОРКУН
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення

_____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

1 червня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
_____ Л. П. Бедратюк
02 01 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Баліцькому Ігорю Ігоровичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Мобільний застосунок для керування клієнтськими записами та фінансовим обліком у сфері краси

Керівник роботи Бедратюк Ганна Іванівна, ст. викладач
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 8-КП
2. Строк подання студентом роботи на кафедру 01.06.2026 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задач, проєктування програмного забезпечення, програмна реалізація та тестування застосунку.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення
1. Діаграма варіантів використання
2. Діаграма зв'язків модулів
3. Діаграма послідовності

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Форкун Ю. В., доцент	1.05.2026	22.05.2026
Антиплагіат	Форкун Ю. В., доцент	1.05.2026	22.05.2026

7. Дата видачі завдання «02» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальної теми кваліфікаційної роботи (КвР)	01.12– 31.12.2025	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог.	01.01 – 20.02.2026	
3 Проєктування програмного забезпечення	21.02 – 20.03.2026	
4 Програмна реалізація з використанням відповідних засобів розроблення. Тестування ПЗ	21.03 – 30.04.2026	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2026	
6 Попередній захист КвР	Травень 2026	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2026	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ Ігор БАЛІЦЬКИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Ганна БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Мобільний застосунок для керування клієнтськими записами та фінансовим обліком у сфері краси.

Автор роботи: Баліцький Ігор Ігорович.

Керівник роботи: Бедратюк Ганна Іванівна.

Пояснювальна записка: 82 с., 26 рис., 8 табл., 3 дод., 41 джерел.

Графічна частина: 3 креслення ф. А3.

DART, FLUTTER, OFFLINE-FIRST, SQLITE, Б'ЮТІ-СФЕРА.

Мета кваліфікаційної роботи: розроблення програмного забезпечення для автоматизації робочих процесів індивідуальних майстрів сфери б'юті-індустрії.

У кваліфікаційній роботі проведено аналіз предметної області та її інформаційного забезпечення, визначені функціональні та нефункціональні вимоги до програмної системи, розроблена загальна архітектура застосунку, спроєктована база даних для зберігання інформації про клієнтів та графік записів, а також розроблена структура інтерфейсу користувача.

Для реалізації програмного продукту використано фреймворк Flutter та мову програмування Dart [17]. За допомогою цих засобів розроблено мобільний застосунок «BeautyDiary», який забезпечує ведення електронного журналу записів, облік вартості наданих послуг та візуалізацію робочого графіка[18].

Практичне значення результатів роботи полягає в тому, що впровадження розробленого застосунку дозволяє відмовитися від паперових носіїв, мінімізувати помилки при плануванні часу (накладки в записах), автоматизувати підрахунок доходів та значно полегшити щоденну роботу майстра з клієнтською базою.

Перспективи подальшого вдосконалення застосунку полягають у впровадженні хмарної синхронізації. Також планується інтеграція автоматичних нагадувань у популярних месенджерах та розширення блоку фінансової аналітики.

1 червня 2026
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.230128.01.01.ПЗ	Пояснювальна записка	82		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ.230128.01.01.Е8	UML-діаграма варіантів	1		
			використання			
5	А3	КвРІПЗ.230128.01.01.Е8	Діаграма зв'язків	1		
			модулів			
6	А3	КвРІПЗ.230128.01.01.Е8	Діаграма послідовності	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.230128.01.01.ВД		
Виконав		Баліцький І.І.		01.06	Мобільний застосунок для керування клієнтськими записами та фінансовим обліком у сфері краси	Літ.	Арк. 1 / Аркушів 1
Керівник		Бедратюк Г.І.		01.06			
Н. контр.		Бедратюк Л.П.		01.06	Відомість документів	ХНУ, ІПЗс-23-1	
Зав. каф.		Бедратюк Л.П.		01.06			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API	—	application programming interface
APK	—	Android Package Kit (пакет застосунку для Android)
AVD	—	Android Virtual Device (віртуальний пристрій Android)
БД	—	база даних
ВВ	—	варіант використання
ЖЦ	—	життєвий цикл
ІС	—	інформаційна система
МПЗ	—	мобільне програмне забезпечення
ОС	—	операційна система
ПЗ	—	програмне забезпечення
ПП	—	програмний продукт
СКБД	—	система керування базою даних
SQLite	—	вбудована реляційна система керування базою даних
UI	—	user interface (інтерфейс користувача)
UML	—	unified modeling language
URL	—	uniform resource locator
UX	—	user experience (досвід користувача)
XML	—	extensible markup language

ВСТУП

Сучасний етап розвитку цифрової економіки характеризується стрімким впровадженням мобільних IT-рішень для оптимізації мікробізнесу та самозайнятих спеціалістів. У сфері надання б'юті-послуг (зокрема серед приватних майстрів манікюру, косметологів та адміністраторів малих студій) гостро постає проблема ефективного обліку клієнтів. Понад 60% індивідуальних майстрів продовжують вести записи на паперових носіях, що призводить до втрати даних, помилок планування розкладу та ускладнює аналіз доходів. Існуючі великі CRM-системи є занадто перевантаженими складним функціоналом, вимагають постійної абонентської плати та стабільного інтернет-з'єднання. Це зумовлює високу актуальність розробки повністю автономного та безкоштовного мобільного застосунку, орієнтованого на локальне збереження інформації та швидку взаємодію користувача з інтерфейсом в умовах обмеженого часу.

Об'єкт дослідження — процеси адміністрування, планування робочого часу та обліку наданих послуг у діяльності приватних майстрів і невеликих б'юті-студій. Предмет дослідження — кросплатформні мобільні програмні засоби локального збереження даних, автоматизації клієнтського розкладу та формування фінансової звітності.

Предметна область аналізу охоплює операційну діяльність індивідуальних спеціалістів індустрії краси. Інформаційна модель системи базується на взаємозв'язку таких сутностей, як картка клієнта, сеанс запису, фінансові транзакції та медіафайли. Межі автоматизації чітко обмежені архітектурою «offline-first»: обробка даних та зберігання файлів виконується локально на одному пристрої користувача. До контуру розробки входять модулі інтерактивного календаря з кольоровим кодуванням щільності записів, гнучка система локальних нагадувань, посторінкова галерея зображень клієнтських робіт, підсистема резервного копіювання та генерація аналітичних звітів. Будь-які

процеси корпоративного управління персоналом великих салонів та інтеграції з державними фіскальними сервісами повністю винесені за межі дослідження.

Розроблюваний програмний продукт «BeautyDiary» призначений для індивідуальних майстрів, фрилансерів та адміністраторів приватних студій. Основними профілями користувачів є спеціалісти, яким необхідне автономне та швидке керування розкладом без ризику витоку даних у мережу. Типові сценарії використання охоплюють створення записів із покращеним кириличним введенням, прикріплення фото з галереї пристрою із фіксацією шляху photoPath у базі даних, перегляд історії візитів з пагінацією, моніторинг виторгу через гістограми, а також експорт даних в Excel та PDF-формати з українськими локалізованими заголовками для подальшого аналізу.

Мета кваліфікаційної роботи — розробити кросплатформний мобільний застосунок «BeautyDiary» на базі фреймворку Flutter та локальної реляційної СУБД SQLite для автоматизації процесів керування робочим розкладом, ведення клієнтської бази, моніторингу фінансової статистики та локального адміністрування даних б'юті-майстра.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– виконати аналіз предметної області та встановити особливості робочих процесів майстрів у сфері б'юті-послуг;

– проаналізувати існуючі аналоги програмного забезпечення та виявити їхні переваги і недоліки;

– обґрунтувати вибір технологічного стеку (Flutter, Dart, SQLite) для реалізації мобільного рішення;

– визначити функціональні та нефункціональні вимоги до системи;

– розробити архітектуру мобільного застосунку та спроектувати схему бази даних;

– спроектувати ергономічний користувацький інтерфейс, орієнтований на швидку взаємодію;

– виконати програмну реалізацію та тестування модулів ПЗ.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змісновний аналіз предметної області, її структурних та функціональних особливостей

Сучасна сфера б'юті-послуг є одним із найбільш динамічних сегментів роздрібного ринку обслуговування. До цієї галузі належать суб'єкти господарювання, які надають послуги з догляду за зовнішністю: салони краси, студії нігтьового сервісу, косметологічні кабінети, а також приватні фахівці. Згідно зі статистичними тенденціями розвитку малого бізнесу в Україні, частка самозайнятих спеціалістів (фізичних осіб-підприємців та фрілансерів), які працюють в індивідуальному форматі без залучення найманого персоналу, становить понад 68% від загальної кількості зареєстрованих точок надання б'юті-послуг.

Об'єктом автоматизації у даній кваліфікаційній роботі є операційна та управлінська діяльність індивідуального майстра з манікюру та педикюру, а також адміністратора невеликої приватної студії краси. Такий фахівець самостійно організовує повний життєвий цикл взаємодії з клієнтом: від первинного звернення до фіксації фінансового доходу та формування аналітики. Діяльність об'єкта дослідження не має складної ієрархічної структури управління, проте характеризується високою щільністю інформаційних потоків і критичною залежністю економічної ефективності від точності оперативного тайм-менеджменту.

На момент проведення передпроєктного обстеження об'єкт автоматизації використовував традиційні безсистемні інструменти обліку: паперовий фінансовий блокнот-планер та стандартний текстовий застосунок нотаток у смартфоні. Проведений аналіз дозволив виявити та обґрунтувати п'ять критичних архітектурно-функціональних проблем організації праці.

Часові втрати при інформаційному пошуку. Зберігання хронології візитів у лінійному текстовому форматі унеможливлює оперативну вибірку. Середній час пошуку історії процедур або специфічних технологічних нотаток (наприклад, формули покриття чи наявності алергічних реакцій у клієнта) за минулі періоди становить понад 4–7 хвилин, що неприпустимо під час безпосереднього виконання сеансу.

Фінансові збитки від помилок планування. Відсутність інтерактивного контролю часових інтервалів призводить до виникнення накладок у розкладі («овербукінг») або утворення нераціональних «вікон» у робочій зміні. Втрата робочого часу через некоректне планування знижує потенційний щомісячний дохід майстра на 15–20%.

Відсутність бізнес-аналітики та інформаційного підґрунтя. Ручний фінансовий облік у паперовому вигляді унеможливлює динамічне групування доходів за періодами (день, тиждень, місяць) чи категоріями послуг. Майстер позбавлений можливості оцінити рентабельність окремих процедур, відстежити динаміку клієнтського трафіку та обґрунтовано сформувати цінову політику.

Хаотичність збереження медіаданих. Фотографії виконаних робіт, які є основою професійного портфоліо майстра манікюру, накопичуються у загальній галереї смартфона. Це призводить до змішування особистих та робочих файлів, ускладнює пошук референсів і унеможливлює прив'язку зображення результату процедури до картки конкретного клієнта.

Низький рівень надійності та критичні ризики втрати даних. Паперові носії схильні до фізичного знищення або зносу, а текстові нотатки у смартфоні повністю залежать від працездатності пристрою. За відсутності інтегрованих механізмів локального або відчужуваного резервного копіювання будь-яка апаратна поломка призводить до безповоротної втрати клієнтської бази.

З огляду на виявлені проблеми, предметна область дослідження структурована на чотири взаємопов'язані функціональні групи процесів, інформаційні потоки між якими мають чітку топологію.

Перша група — управління розкладом та записами. Це ядро системи, що реалізує функції створення, редагування та анулювання сеансів. Друга група — керування клієнтською базою та історією процедур, що відповідає за довготривале збереження профілів клієнтів. Третя група — локальний фінансовий моніторинг, призначений для агрегування вартісних показників виконаних сеансів. Четверта група — тайм-менеджмент та локальне інформування, що забезпечує автоматичний контроль розкладу.

Взаємозв'язок функцій у програмному забезпеченні реалізується за такою схемою: сутність «Робоча зміна» задає часову матрицю, всередині якої ініціалізується сутність «Сеанс/Запис». «Сеанс/Запис» виступає головним реляційним вузлом, який динамічно пов'язує сутність «Клієнт» із сутністю «Послуга» (зчитуючи тривалість та ціну в ₴ з довідника). У момент зміни статусу «Сеансу» на виконаний, система автоматично тригерує потік даних до модуля фінансового обліку, де створюється сутність «Транзакція», що оновлює аналітичні гістограми виторгу. Одночасно шлях до медіафайлу (photoPath), обраного з галереї, записується у схему бази даних із прив'язкою до ідентифікатора запису, формуючи структуровану історію візитів.

Цільова аудиторія програмного продукту розподілена на три чіткі профілі користувачів:

– приватний майстер-фрілансер, що потребує швидкого мобільного інструменту для реєстрації візитів «в один дотик» з обов'язковою наявністю розширеного введення кирилицею (relaxed Cyrillic input) для миттєвого пошуку клієнтів на ходу. Типовий сценарій: створення запису, вибір часу з підкреслених клікабельних полів інтерфейсу із автоматичним виставленням початкового часу на 00:00 для усунення зайвих дій користувача;

– майстер б'юті-салону з великою клієнтською базою, його ключова потреба полягає у наочності розкладу та контролі завантаженості. Типовий сценарій: візуальний аналіз щільності записів через інтерактивний календар за допомогою колірних маркерів трафіку (зелений — низька, помаранчевий —

середня, червоний — висока завантаженість дня), а також налаштування локальних pre-event нагадувань за певну кількість хвилин до процедури;

– адміністратор невеликої студії краси, орієнтований на локальний облік, підбиття підсумків періоду та аудит діяльності. Типовий сценарій: перегляд узагальненої статистики виторгу за тиждень чи місяць через гістограми, щоденний ранковий огляд розкладу (morning summary) у заданий час, а також експорт структурованих звітів у формати PDF та Excel з українськими локалізованими заголовками для архівування або фіскального аналізу.

Інформаційне забезпечення застосунку оперує строго визначеними масивами вхідних та вихідних даних. Вхідними даними системи є: текстові рядки (імена клієнтів, номери телефонів, нотатки у кириличному форматі UTF-8), числові значення (ціна послуг, тривалість сеансу в хвилинах), часові мітки (дати, системні змінні пакету timezone) та рядкові посилання на локальні файли зображень (photoPath). Вихідними даними є: візуалізовані графіки виторгу, системні локальні сповіщення ОС, бінарні файли резервних копій бази даних, а також генеровані документи документів .pdf та .xlsx, що містять відформатовані ціни із грошовим символом гривні (₴) та шляхи до графічних файлів результатів робіт.

З огляду на використання архітектурної концепції offline-first, безпека та конфіденційність інформації мають локальний характер. Зберігання бази даних sqflite та медіафайлів реалізовано всередині ізольованого сховища застосунку (Application Sandbox) операційної системи, що унеможливлює несанкціонований доступ до персональних даних клієнтів з боку стороннього ПЗ. Захист від втрати інформації при поломці пристрою реалізовано через підсистему резервного копіювання, що дозволяє користувачеві в ручному або автоматичному режимі створювати зліпки бази даних та зберігати їх на зовнішніх відчужуваних носіях або синхронізувати з приватними хмарними сховищами за потреби. Конфіденційність медіаданих забезпечується відмовою від прямого доступу до камери пристрою — додаток працює виключно через системний пакет

image_picker для імпорту вже існуючих фото з галереї за явним дозволом користувача, що мінімізує вразливості, пов'язані з доступом до апаратних ресурсів смартфона.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Перед визначенням архітектурних та функціональних вимог до розроблюваного мобільного застосунку було проведено комплексний аналітичний огляд наявного програмного забезпечення, що застосовується на світовому та вітчизняному ринках для автоматизації обліку в індустрії краси. Метою цього підрозділу є систематизація досвіду провідних розробників ПЗ, виявлення критичних недоліків існуючих систем у контексті їхнього застосування приватними майстрами, а також чітке інженерне та економічне обґрунтування доцільності створення автономного продукту «BeautyDiary».

На сьогоднішній день ринок профільних IT-рішень представлений трьома основними категоріями: ентерпрайз-платформи для мережевих салонів, хмарні CRM-системи загального призначення та мобільні органайзери. Для детального аналізу було відібрано три системи, які є лідерами у своїх класах згідно зі звітами сервісів Capterra та G2, та найбільш репрезентативно демонструють поточні архітектурні тренди: Fresha, Salonized та Vagaro. Вибір цих аналогів зумовлений тим, що вони покривають понад 50% міжнародного ринку автоматизації б'юті-сфери та задають стандарти проектування інтерфейсів у цій предметній області.

Fresha (раніше відома як Shedul) є глобальною хмарною платформою для управління записами, що розповсюджується за моделлю freemium із транзакційною монетизацією (рисунок 1.1).

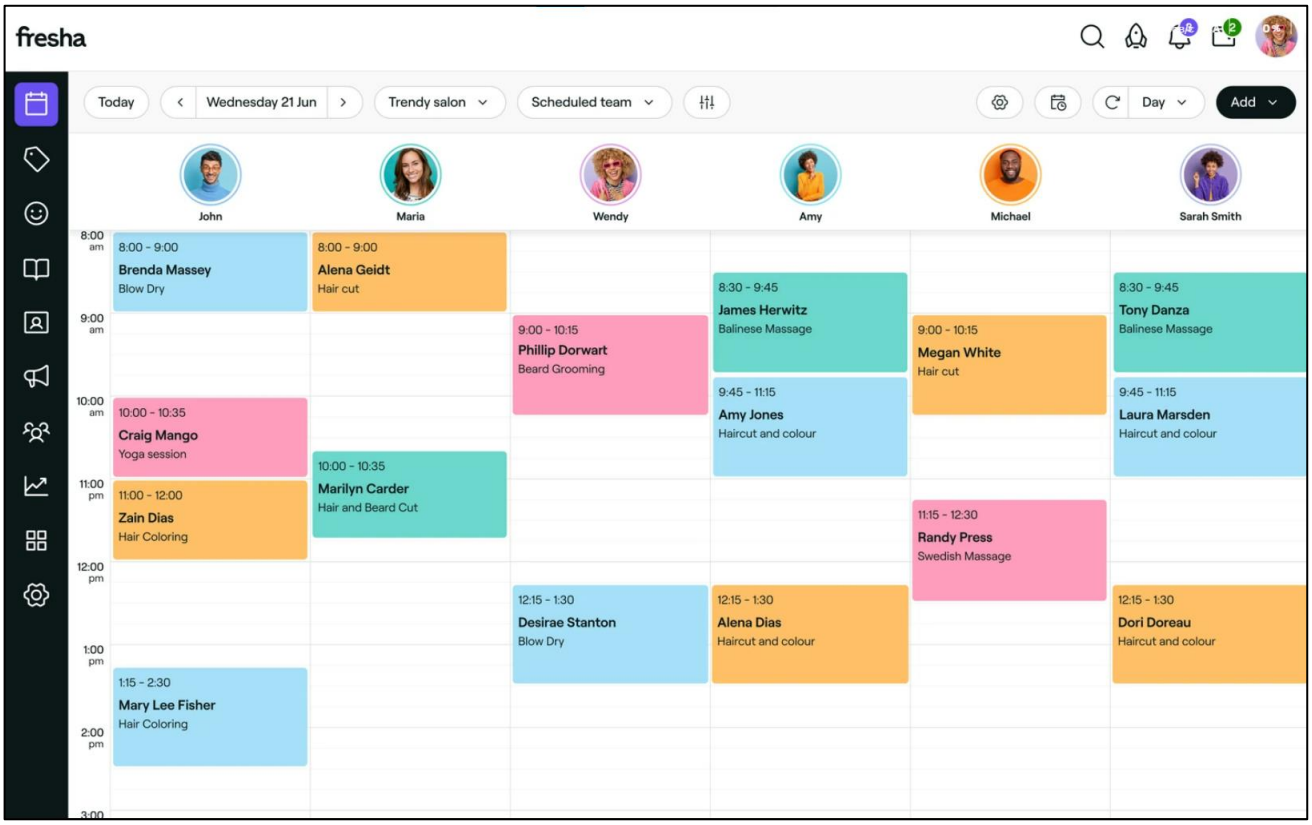


Рисунок 1.1. – Головна сторінка Fresha

Система має потужний функціонал: інтегрований модуль онлайн-бронювання, інструменти автоматичного розрахунку заробітної плати персоналу, складський облік залишків витратних матеріалів за методом FIFO, маркетинговий конструктор та інтеграцію з POS-терміналами.

Проте для приватного самозайнятого майстра Fresha має суттєві обмеження:

– надлишковість архітектури, інтерфейс перевантажений бізнес-логікою управління багаторівневою структурою персоналу та складами, що уповільнює роботу індивідуального користувача;

– залежність від мережі, повна відсутність автономності — за умов нестабільного зв'язку чи відсутності Інтернету доступ до розкладу блокується;

– фінансова та мовна неадаптованість, платформа не має української локалізації, не підтримує грошовий знак гривні (₴) як базову валюту у системних звітах і не оптимізована під алгоритми швидкого розпізнавання кириличних імен

(relaxed Cyrillic input);

– конфіденційність даних, збереження персональних даних клієнтів на серверах за межами юрисдикції України викликає ризики невідповідності закону «Про захист персональних даних».

Salonized — європейська веборієнтована CRM-система, призначена для операційного менеджменту в б'юті-бізнесі (рисунок 1.2). Вона фокусується на автоматизації календарного планування, надсиланні email-розсилок та формуванні фінансової звітності підприємства.

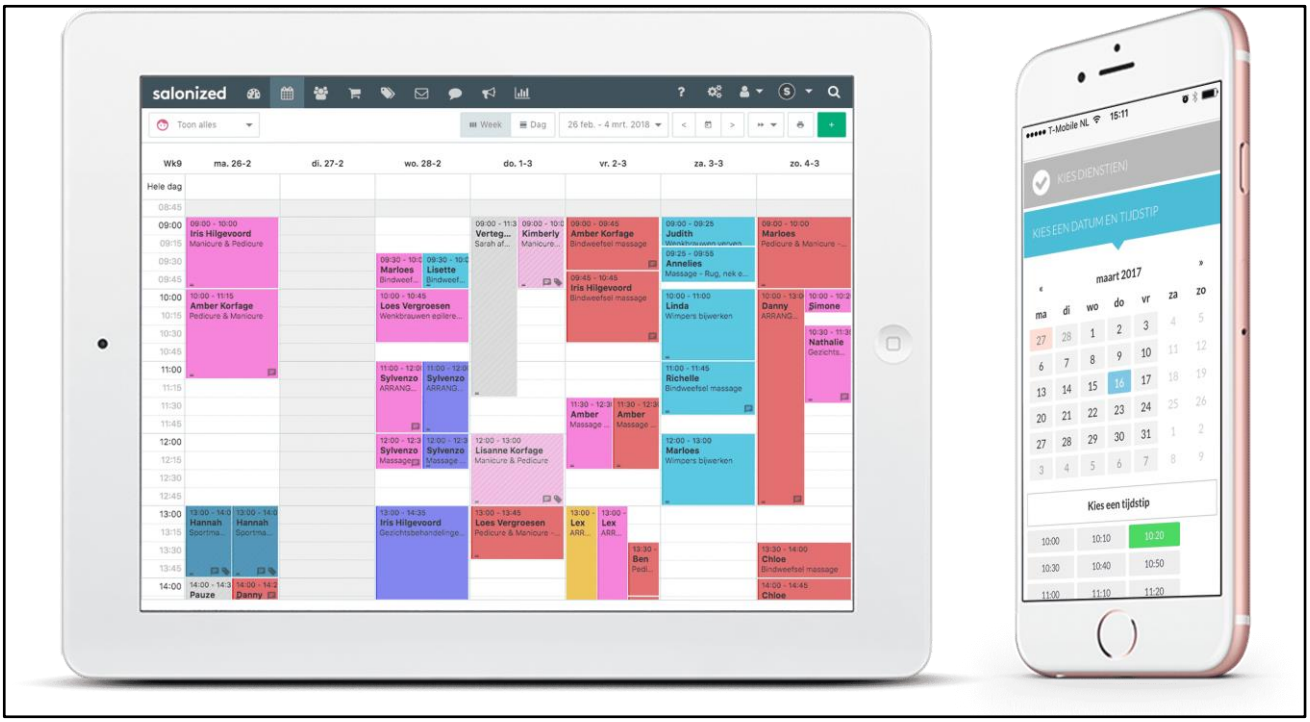


Рисунок 1.2 – CRM-система Salonized

Основними технічними та експлуатаційними недоліками Salonized є:

– висока вартість володіння, комерційна модель передбачає виключно платну щомісячну підписку (від 24 EUR/місяць за базовий тариф), що є економічно деструктивним для майстрів-початківців чи фрилансерів з невеликим оборотом;

– відсутність нативної мобільності, сервіс є веборієнтованим, мобільний

додаток є лише оболонкою (WebView), що призводить до високого використання оперативної пам'яті смартфона та затримок під час рендерингу елементів інтерфейсу;

– обмеженість збереження медіаструктур, у схемі даних системи не передбачено можливості прив'язки локальних фотографій результатів робіт до конкретної події в календарі, що унеможливлює ведення картки клієнта як портфоліо.

Vagaro — американська ентерпрайз-платформа для комплексного управління б'юті-, велнес- та фітнес-індустріями (рисунок 1.3). Вона містить розвинену POS-систему, модулі управління абонементами, інтеграцію з соціальними мережами для лідогенерації та інструменти побудови складних аналітичних гістограм.

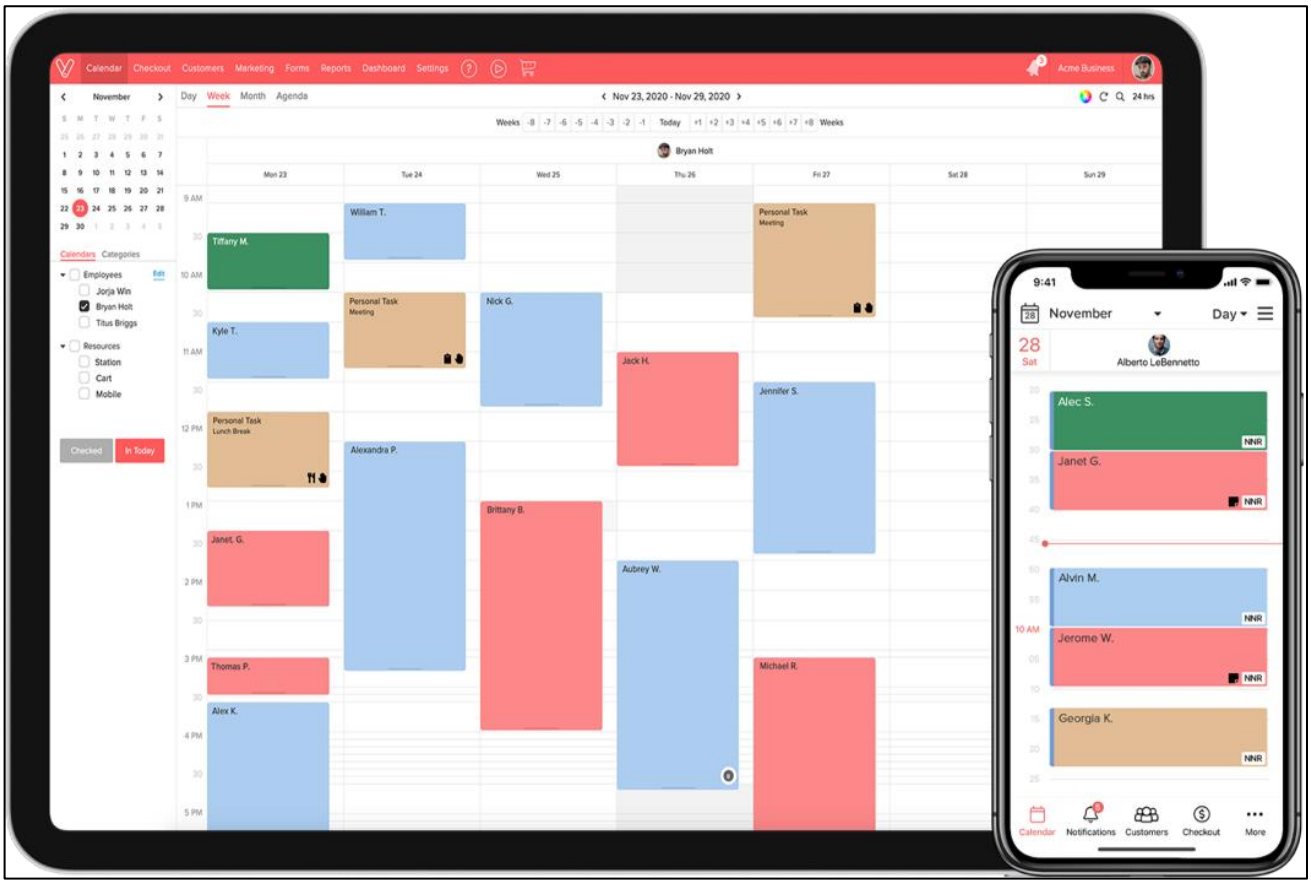


Рисунок 1.3 – Платформа Vagaro

Незважаючи на високу технологічність, Vagaro є абсолютно непридатною для вітчизняного приватного майстра з таких причин:

– цінова політика, вартість стартує від 25 USD/місяць за одного користувача, а додаткові модулі (наприклад, файлове сховище) потребують окремої оплати;

– регіональна ізоляція, сервіс жорстко орієнтований на ринки США та Великої Британії. Система не адаптована до українського законодавства та мовного середовища;

– архітектурна залежність, аналогічно до попередніх зразків, система реалізує концепцію cloud-only, що повністю нівелює можливість автономної роботи у форс-мажорних обставинах.

Для системного порівняння архітектурних та функціональних параметрів існуючих рішень було сформовано перелік із 10 цільових інженерно-економічних критеріїв. Вибір цих критеріїв обґрунтовано специфікою діяльності індивідуального майстра: автономність (офлайн-режим) необхідна для роботи в будь-яких умовах; локалізація та підтримка валюти ₴ забезпечують зручність обліку; локальне збереження та резервне копіювання гарантують безпеку; а безкоштовність мінімізує поріг входження для мікробізнесу.

Аналіз даних таблиці 1.1 дозволяє зробити чіткі висновки щодо стану наявного програмно-технічного забезпечення предметної області та виявити фундаментальний ринковий розрив.

По-перше, усі наявні закордонні аналоги побудовані на базі клієнт-серверної архітектури з обов'язковим збереженням даних на віддалених хмарних серверах. Це створює пряму деструктивну залежність користувача від стабільності інтернет-провайдерів та відкриває вектор потенційних уразливостей щодо витоку конфіденційної інформації клієнтів.

По-друге, жодне з комерційних рішень не адаптоване до специфіки українського ринку (відсутня локалізація, немає підтримки національної валюти ₴, інтерфейси не розраховані на швидке введення кириличних імен).

По-третє, функціональне перевантаження (feature creep) розглянутих систем штучно ускладнює UX/UI-дизайн, роблячи поріг входження для одноосібного майстра занадто високим, що підтверджується профільними дослідженнями [9].

Таблиця 1.1 — Порівняльний аналіз існуючих програмних рішень

Критерій порівняння аналізованого ПЗ	Fresha	Salonized	Vagaro	Розроблюваний застосунок
Офлайн-режим роботи	Ні	Ні	Ні	Так (повна автономність)
Українська мова інтерфейсу	Ні	Ні	Ні	Так (нативна локалізація)
Підтримка символу валюти ₴	Ні	Ні	Ні	Так (системне форматування)
Прикріплення фото до картки запису	Ні	Ні	Частково	Так (збереження photoPath)
Безкоштовне використання	Частково	Ні	Ні	Так (повністю безкоштовний)
Орієнтація на індивідуального майстра	Ні	Ні	Ні	Так (інтерфейс «в один дотик»)
Експорт звітів у PDF / Excel	Частково	Так	Так	Так (з урахуванням фото та ₴)
Локальні сповіщення (без Інтернету)	Ні	Ні	Ні	Так (morning summary / pre-event)

На основі проведеного дослідження визначено чотири головні дестабілізуючі чинники в предметній області, на подолання яких безпосередньо спрямовано розробку мобільного застосунку «BeautyDiary»:

– децентралізація та хаотичність даних, усувається шляхом консолідації розкладу, бази клієнтів, фінансових транзакцій та фотографій робіт в єдиній локальній реляційній схемі даних.

– високі ризики апаратної втрати інформації, долаються впровадженням надійної підсистеми резервного копіювання бази даних з можливістю її відчужуваного збереження користувачем.

– висока трудомісткість ручного аналізу, нівелюється за рахунок автоматичного агрегування фінансових показників та рендерингу наочних гістограм виторгу за вибрані періоди.

– мовно-інтерфейсний бар'єр аналогів, вирішується повною інтеграцією української локалізації, системним форматуванням цін із символом ₴, оптимізацією вводу (relaxed Cyrillic input) та спрощеним UX для керування часом сеансів на підкреслених клікабельних полях.

Таким чином, проведений аналіз науково та практично обґрунтовує повну відсутність на ринку програмних продуктів легковагового, нативного, кросплатформного рішення, яке б поєднувало концепцію *offline-first* із цільовим інструментарієм для самозайнятих б'юті-фахівців без стягнення абонентської плати. Створення мобільного застосунку «BeautyDiary» є актуальним, своєчасним та технічно доцільним інженерним завданням для заповнення виявленої технологічної ніші.

1.3 Визначення функціональних та нефункціональних вимог до мобільного застосунку

Визначення системних інженерних вимог до програмного забезпечення є критично важливим етапом передпроєктного обстеження, що безпосередньо

зумовлює архітектурне проєктування та програмну реалізацію компонентів системи. Вимоги формуються на основі результатів системного аналізу операційних процесів індивідуального б'юті-майстра, виявлених деструктивних чинників у роботі аналогів та інформаційних потреб цільових користувачів. Згідно з методологією інженерії програмного забезпечення (IEEE Std 830), специфікація вимог структурована на функціональні та нефункціональні параметри, які деталізовані через поведінкові сценарії єдиного системного актора.

У межах розроблюваної системи єдиним актором є користувач у ролі «Майстер» (або адміністратор приватної студії), який володіє повним набором прав на виконання CRUD-операцій без внутрішнього розмежування рівнів доступу, що обумовлено локальною архітектурою offline-first. Концептуальна схема взаємодії актора з функціональними блоками мобільного застосунку «BeautyDiary» відображена у моделі варіантів використання на рисунку 1.4.

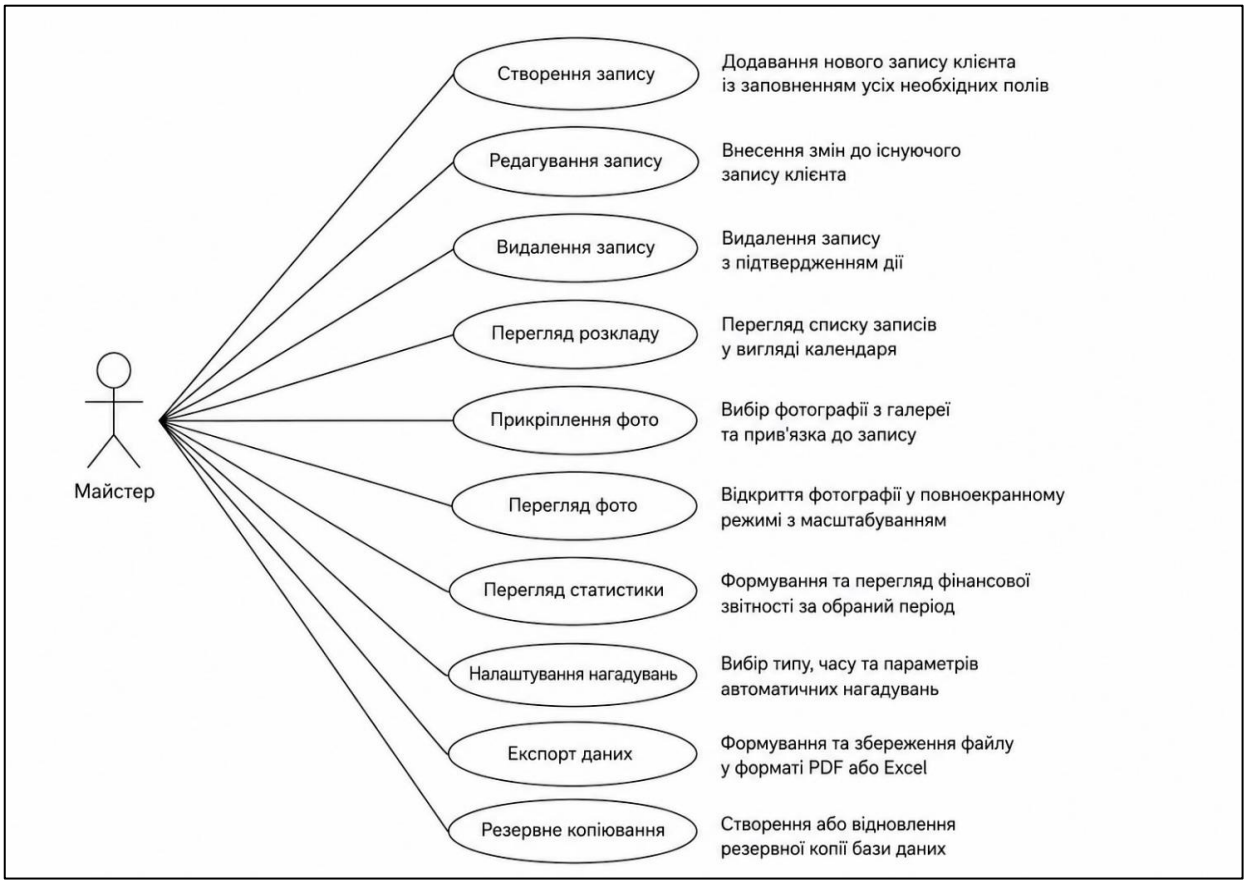


Рисунок 1.4 – UML-діаграма варіантів використання

На основі декомпозиції предикатів діаграми варіантів використання сформовано специфікований перелік функціональних вимог до програмного забезпечення, згідно з яким мобільний застосунок має реалізовувати такі ключові можливості:

– управління картками та розкладом записів клієнтів із обов'язковим заповненням реляційних полів, що містять унікальний ідентифікатор, ім'я клієнта, контактний телефон, дату, час, перелік послуг, підсумкову ціна та блок текстових нотаток;

– оптимізація введення та керування часовими мітками за допомогою інтерфейсу, що за замовчуванням ініціалізує час як 00:00 для мінімізації зайвих дій користувача, підсвічує поля дати/часу як клікабельні зони та надає розширене кириличне введення (relaxed Cyrillic input) для гнучкої фільтрації текстових даних у реляційній базі;

– локальна обробка та пагінація медіафайлів шляхом імпорту графічних файлів результатів робіт виключно з системної галереї пристрою через пакет image_picker, реєстрації строкового шляху photoPath у локальній базі даних, підтримки повноекранного перегляду зображень із жестами масштабування (pinch-to-zoom) та посторінкової навігації для медіагалереї;

– інтерактивна колірна індикація робочого календаря, який динамічно підраховує кількість запланованих візитів на кожну добу та маркує дати відповідними кольорами трафіку, де зелений колір позначає низьке завантаження, помаранчевий — середнє, а червоний — критично високе завантаження дня;

– автономне сповіщення та нагадування на основі функціоналу пакетів flutter_local_notifications та timezone, що забезпечує генерацію щоденного ранкового огляду розкладу (morning summary) у вказану в налаштуваннях годину, а також оперативні pre-event сповіщення за налаштовану кількість хвилин до початку конкретного сеансу;

– локальне резервування та синхронізація БД через підтримку механізму створення повних резервних копій бази даних sqflite (SQLite) у ручному та

автоматичному режимах із можливістю експорту файлу зліпку на зовнішні відчужувані носії пристрою;

– формування кросплатформної звітності та експорту завдяки генерації документів у форматах PDF та Excel, що містять локалізовані українською мовою заголовки таблиць, системне відображення валютного символу гривні (₴), а також виведення текстового шляху до збереженого локального фото;

– агрегування фінансової статистики за допомогою модуля аналітики, що здійснювавує математичне групування вартісних показників виконаних візитів за задані часові інтервали та виконує рендеринг наочних гістограм виторгу.

Поведінкові сценарії взаємодії актора із системою специфікуються через чітку послідовність кроків, що охоплює основний, альтернативний та винятковий шляхи виконання операцій. Для базового варіанту використання системи «Створення запису візиту» основний сценарій (Happy Path) складається з такої послідовності дій: спочатку майстер активує кнопку додавання запису на головному екрані календаря, після чого система ініціалізує форму, виставляє час за замовчуванням на мітку 00:00 та підкреслює інтерактивні поля, далі майстер вводит ім'я клієнта за допомогою кириличного введення, вказує телефон, вибирає послугу, коригує час та натискає кнопку збереження, а на завершальному етапі система валідує дані, створює новий рядок у базі даних sqflite, оновлює кольоровий індикатор завантаженості дня в календарі та повертає користувача на головний екран.

Альтернативний сценарій, що передбачає додавання фотографій робіт, інтегрується в структуру взаємодії в такий спосіб: на етапі заповнення форми майстер ініціює додавання медіафайлу, після чого система викликає нативний pick-інтерфейс операційної системи для доступу до галереї пристрою, далі майстер вибирає зображення результату процедури, а система автоматично копіює посилання у змінну photoPath, відображає мініатюру фото на формі та переходить до кроку збереження даних основного сценарію.

Винятковий сценарій, покликаний попередити виникнення конфліктів у розкладі, реалізує таку логіку поведінки: на етапі валідації форми внутрішній сервіс перевірки виявляє наявність іншого запису на цей самий часовий інтервал, після чого система негайно блокує транзакцію запису в базу даних, виводить на екран модальне попередження українською мовою про конфлікт розкладу, а саму форму залишає активною для подальшого коригування часу майстром.

Нефункціональні вимоги визначають якісні критерії, експлуатаційні обмеження, технологічні рамки та середовище функціонування розроблюваного програмного забезпечення, які в межах проєкту мають таку специфікацію:

– вимоги до архітектурної автономності визначають реалізацію парадигми offline-first, що забезпечує повну працездатність бізнес-логіки, збереження даних у реляційній базі СУБД sqflite та генерацію локальних нотифікацій без підключення пристрою до мережі Інтернет;

– вимоги до сумісності та кросплатформності встановлюють, що кодова база проекту розробляється на базі фреймворку Flutter (мова Dart) та повинна забезпечувати ідентичне відображення UX/UI та консистентність даних на мобільних операційних системах Android (версії 8.0 та вище) та iOS (версії 14 та вище);

– вимоги до продуктивності та оптимізації пам'яті визначають, що час відгуку інтерфейсу при переході між екранами не повинен перевищувати 200 мс, а операції важкого рендерингу великих списків історії чи фотогалерей повинні використовувати механізми пагінації для запобігання переповнення RAM пристрою;

– вимоги до безпеки та конфіденційності даних фіксують, що локальний файл бази даних SQLite та імпортовані графічні об'єкти повинні зберігатися виключно всередині захищеного ізольованого контейнера застосунку (Application Sandbox), унеможливлюючи зчитування конфіденційної інформації клієнтів стороннім програмним забезпеченням;

– вимоги до інтерфейсу та локалізації вимагають, щоб користувацький

інтерфейс мав стовідсоткову нативну українську локалізацію (включаючи повідомлення про помилки, інтерфейси експорту та системні алерти), підтримував грошове форматування із відображенням символу гривні (₴) та відповідав стандартам ергономіки Google Material Design 3;

– вимоги до надійності та стійкості до відмов визначають, що система повинна реалізувати перехоплення винятків (try-catch) на рівні сервісів-синглтонів, гарантувати збереженість даних при раптовому вимкненні живлення смартфона або аварійному завершенні процесу ОС, а також повністю виключити незворотне видалення інформації без явного підтвердження з боку користувача через UI.

Проведений аналіз та систематизація вимог є достатньою і надійною основою для переходу до наступного етапу проєктування архітектури та структури мобільного застосунку «BeautyDiary».

1.4 Висновки. Постановка задачі на розробку застосунку.

У результаті проведеного дослідження предметної області було здійснено комплексний аналіз діяльності приватного майстра з манікюру та педикюру як об'єкта автоматизації, виявлено основні деструктивні чинники існуючої організації обліку та визначено інженерні вимоги до розроблюваного програмного забезпечення.

По-перше, сфера б'юті-послуг є активно зростаючою галуззю з високою часткою приватних майстрів-фрилансерів, діяльність яких супроводжується систематичною потребою у веденні клієнтського обліку, плануванні робочого часу та аналізі фінансових результатів. Незважаючи на високу динаміку ринку, дана категорія користувачів залишається недостатньо охопленою існуючими програмними рішеннями через їх складність та надлишкову орієнтацію на великі салонні мережі.

По-друге, аналіз поточного стану організації обліку в діяльності приватного майстра виявив суттєві недоліки, зумовлені використанням паперового блокноту та текстових нотаток у смартфоні як основних інструментів роботи:

– децентралізація даних унеможливлює швидкий доступ до повної хронологічної інформації про клієнта;

– відсутність системного резервного копіювання створює постійний ризик безповоротної втрати накопичених даних;

– неможливість автоматичного формування фінансової звітності ускладнює оперативний аналіз ефективності діяльності;

– відсутність прив'язки фотографій виконаних робіт до конкретних записів клієнтів унеможливлює формування структурованого портфоліо;

– відсутність інтегрованої системи автоматичних нагадувань суттєво підвищує ризик організаційних помилок у плануванні робочих прийомів.

По-третє, огляд наявного програмно-технічного забезпечення предметної області показав, що існуючі рішення — Fresha, Salonized, Vagaro — не відповідають реальним потребам приватного майстра через орієнтованість на великі салони, обов'язкову хмарну інфраструктуру, відсутність україномовного інтерфейсу та комерційну модель поширення з високою абонентською платою. Жоден з розглянутих закордонних аналогів не забезпечує одночасно локального зберігання даних, повноцінної підтримки української мови та орієнтації на одноосібного користувача.

По-четверте, на основі аналізу предметної області та виявлених недоліків існуючих рішень сформовано перелік функціональних та нефункціональних вимог до розроблюваного застосунку, що охоплюють управління записами клієнтів, роботу з фотоматеріалами, ведення фінансової статистики, систему автономних нагадувань, експорт даних та резервне копіювання.

На підставі проведеного аналізу формулюється така постановка задачі кваліфікаційної роботи. Мета розробки — створення мобільного застосунку «BeautyDiary» для платформи Android, що забезпечує комплексну автоматизацію

процесів ведення клієнтського обліку в діяльності приватного майстра б'юті-послуг та усуває виявлені недоліки існуючої організації роботи.

Для досягнення поставленої мети необхідно вирішити такі задачі проєктування та розробки:

– виконати проєктування архітектури мобільного застосунку з урахуванням вимог до автономності, продуктивності та зручності використання;

– розробити структуру локальної реляційної бази даних для збереження записів клієнтів, фінансових показників та шляхів до фотоматеріалів;

– спроєктувати інтерфейс користувача відповідно до принципів ергономіки мобільних пристроїв та вимог україномовної локалізації;

– реалізувати модуль управління записами клієнтів із підтримкою повного циклу операцій створення, перегляду, редагування та видалення;

– реалізувати модуль роботи з фотоматеріалами, що забезпечує прив'язку зображень до записів клієнтів та їх повноекранний перегляд;

– реалізувати модуль фінансової статистики;

– реалізувати систему автоматичних нагадувань із підтримкою щоденного огляду розкладу та нагадувань перед конкретними записами;

– реалізувати функції експорту даних у формати PDF та Excel;

– реалізувати механізм резервного копіювання бази даних для забезпечення надійності зберігання накопиченої інформації;

– провести тестування розробленого застосунку на відповідність визначеним функціональним та нефункціональним інженерним вимогам.

Розроблений застосунок повинен функціонувати у повному офлайн-режимі, зберігати всі дані локально на пристрої користувача всередині захищеного пісочного контейнера застосунку (Application Sandbox), підтримувати україномовний інтерфейс та бути доступним для використання без абонентської плати. Технічна реалізація здійснюється на базі кросплатформного фреймворку Flutter з використанням мови програмування Dart та локальної вбудованої бази даних SQLite.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура та функціональна структура застосунку

Проєктування архітектури є ключовим етапом розробки програмного забезпечення, що передує безпосередній реалізації та визначає загальну організацію системи, принципи розподілу відповідальності між компонентами та спосіб їх взаємодії. Обґрунтоване архітектурне рішення забезпечує зрозумілість кодової бази, можливість її подальшого розширення, а також спрощує процес тестування і супроводу застосунку. При проєктуванні архітектури мобільного застосунку «BeautyDiary» було проаналізовано та порівняно кілька фундаментальних підходів, що застосовуються у сучасній розробці кросплатформних мобільних застосунків.

Класичний патерн MVC (Model-View-Controller) передбачає розподіл системи на три рівні, де модель відповідає за дані, вигляд — за відображення, контролер — за обробку дій користувача, проте у контексті розробки на Flutter його пряме використання ускладнюється відсутністю вбудованого механізму нативних контролерів, що призводить до змішування логіки та роздування кодової бази екранів. Патерн MVVM (Model-View-ViewModel) є більш природним для реактивних декларативних фреймворків і вводить проміжний шар ViewModel для збереження стану екрана, але для застосунку з невеликою кількістю екранів цей підхід є надлишково складним. Розширений підхід Clean Architecture (Чиста архітектура) забезпечує повну незалежність від фреймворків через розподіл на шари Entities, Use Cases та Interface Adapters, проте створює значну кількість шаблонного коду (boilerplate code), що є неефективним для локального offline-first проекту. Популярний у Flutter-спільноті патерн BLoC (Business Logic Component) використовує потоки даних (Streams) для повної ізоляції бізнес-логіки від інтерфейсу, але вимагає підключення важких зовнішніх бібліотек і суттєво підвищує поріг входу та складність налагодження. З урахуванням масштабу проєкту, кількості екранів та локального характеру

оброблюваних даних для реалізації застосунку було обрано архітектурний підхід на основі сервісів-синглтонів у поєднанні з патерном інверсії залежностей та безпосереднім управлінням станом на рівні окремих екранів.

Обрана архітектурна модель детально враховує специфіку життєвого циклу мобільних платформ Android та iOS, безпосередньо взаємодіючи з нативними станами додатку. Локальний стан екранів інтегрується з життєвим циклом віджетів StatefulWidget, де ініціалізація даних прив'язана до методу initState(), а звільнення ресурсів, зупинка потоків та анулювання контролерів введення — до методу dispose(). Це дозволяє уникнути витоків пам'яті (*memory leaks*) при згортанні екранів, переорієнтації пристрою або переході процесу ОС у фоновий режим. Надійність збереження даних при раптовому вивантаженні застосунку операційною системою з пам'яті забезпечується транзакційністю на рівні локальних сервісів.

Проєктування системи базується на синергії кількох ключових шаблонів проєктування, які забезпечують гнучкість та ізоляцію компонентів. Патерн «Одинак» (Singleton) гарантує існування єдиного екземпляра кожного сервісу протягом усього життєвого циклу застосунку, що унеможливлює конфлікти при зверненні до бази даних SQLite з різних екранів. Патерн «Репозиторій» (Repository) інкапсулює в собі логіку доступу до даних, виступаючи проміжним ізоляційним шаром між низькорівневими SQL-запитами та високорівневою бізнес-логікою сервісів. Шаблон «Ін'єкція залежностей» (Dependency Injection) реалізований через легковажний сервіс-локатор, що дозволяє динамічно впроваджувати екземпляри репозиторіїв та сервісів у компоненти представлення, спрощуючи підміну компонентів на mock-об'єкти під час проведення Unit-тестування.

Відповідно до обраного підходу, застосунок логічно поділяється на три рівні, що утворюють чітку вертикальну ієрархію:

– рівень представлення (Presentation Layer) охоплює графічні екрани та віджети застосунку, які відповідають за рендеринг інтерфейсу згідно з

концепцією Material Design 3, обробку взаємодії з користувачем, валідацію полів введення та управління локальним станом;

– рівень сервісів (Service Layer) містить класи-синглтони, що інкапсулюють прикладну бізнес-логіку системи, координують взаємодію між різними підсистемами, обробляють винятки та виступають єдиною точкою доступу до функціональних можливостей;

– рівень даних (Data Layer) представлений репозиторіями та локальною базою даних SQLite під управлінням сервісу бази даних, який забезпечує ініціалізацію сховища, міграцію схем та виконання атомарних операцій запису й читання.

Схему взаємодії та розділення відповідальностей між рівнями розробленої архітектури мобільного застосунку наведено на рисунку 2.1.

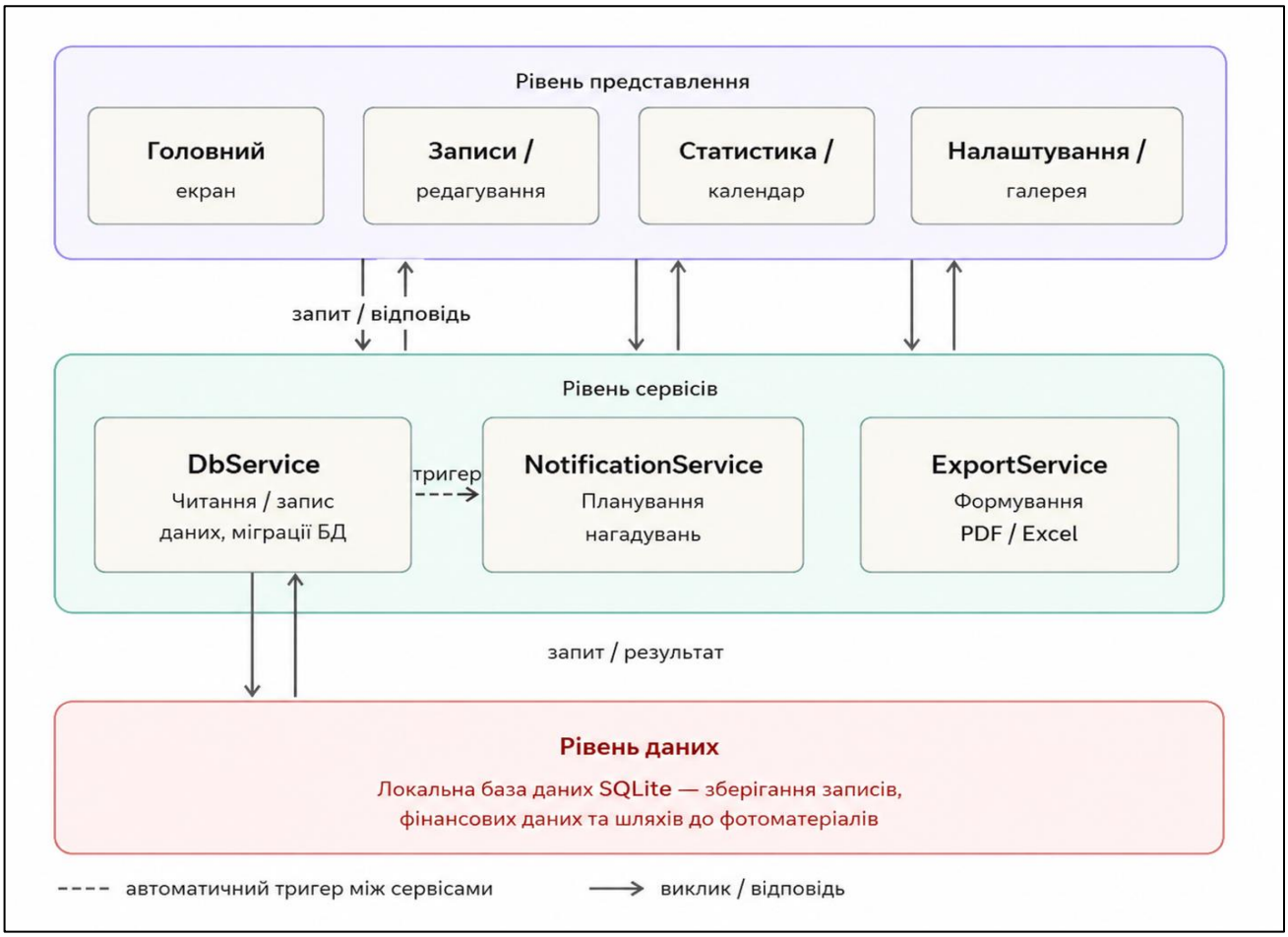


Рисунок 2.1 – Діаграма взаємодії між рівнями архітектури

Сервісний рівень застосунку складається з трьох основних компонентів, кожен з яких відповідає за окрему функціональну підсистему. Сервіс роботи з базою даних забезпечує ініціалізацію сховища даних, керування його схемою та реалізацію всіх операцій читання і запису за допомогою SQL-запитів. Окрім цього, після кожної операції зміни записів даний сервіс автоматично ініціює актуалізацію запланованих нагадувань, звертаючись до сервісу сповіщень. Таким чином, сервіс бази даних виступає тригером для підсистеми нагадувань, що забезпечує їх автоматичну синхронізацію із поточним станом записів без необхідності ручного керування з боку екранів. Сервіс управління нагадуваннями відповідає за ініціалізацію підсистеми локальних сповіщень, планування нагадувань двох типів — щоденного ранкового огляду розкладу та нагадування перед конкретним записом — а також їх скасування у разі видалення або зміни відповідного запису. При плануванні сповіщень враховується поточний часовий пояс пристрою для забезпечення коректності часу їх надсилання користувачеві. Сервіс експорту та аналітики забезпечує агрегування фінансових показників, формування статистичних зрізів та генерацію підсумкових документів.

Функціональна структура представлення застосунку охоплює вісім основних екранів, що утворюють чітку ієрархію навігації на основі стандартного навігаційного стеку, де логіка розподілена таким чином:

– головний екран виступає відправною точкою для відображення списку записів та швидкої навігації до інших розділів системи;

– екран створення та редагування запису забезпечує введення, валідацію та збереження даних про запис б'юті-клієнта;

– екран перегляду фотографії реалізує повноекранне відображення прикріпленого фото з підтримкою жестів масштабування;

– галерея фотографій надає інтерфейс перегляду всіх завантажених медіафайлів із інтегрованим текстовим пошуком та посторінковою пагінацією;

– екран історії відвідувань відображає повну хронологію візитів конкретного клієнта з можливістю динамічної фільтрації;

– екран статистики виконує візуалізацію фінансової звітності та доходів майстра у вигляді інтерактивних гістограм;

– екран календаря забезпечує наочний перегляд розкладу з кольоровим маркуванням завантаженості днів;

– екран налаштувань надає інтерфейс для керування часовими параметрами нагадувань та запуску ручного резервного копіювання.

Навігація між екранами реалізована на основі стандартного навігаційного стеку, що забезпечує звичну для мобільних застосунків поведінку переходів із підтримкою повернення на попередній рівень. Головний екран є відправною точкою навігації та забезпечує доступ до всіх функціональних блоків застосунку. Екран створення та редагування запису є спільним для обох операцій, що дозволяє уникнути дублювання інтерфейсних компонентів. Екрани перегляду фотографії та галереї є дочірніми відносно екрана запису і відкриваються поверх основного навігаційного стеку.

Файлова організація проєкту відповідає загальноприйнятим конвенціям розробки кросплатформних мобільних застосунків та відображає обрану архітектуру. Вихідний код структурований за функціональним принципом і поділяється на чотири основні групи компонентів: моделі даних, сервіси, екрани та точка входу застосунку. Така організація забезпечує чіткий розподіл відповідальності між компонентами, відповідає принципу єдиної відповідальності та спрощує орієнтування у кодовій базі під час розробки і подальшого супроводу. Таким чином, обраний архітектурний підхід на основі сервісів-синглтонів із трирівневою організацією коду є обґрунтованим рішенням для мобільного застосунку даного масштабу та забезпечує необхідний баланс між простотою реалізації, структурованістю проєкту та зручністю його подальшого розширення.

2.2 Проєктування структури бази даних

Основу інформаційного забезпечення мобільного застосунку «BeautyDiary» становить локальна реляційна база даних, що розгортається безпосередньо на мобільному пристрої кінцевого користувача. Вибір локального ізольованого сховища даних зумовлений інженерною вимогою забезпечення автономної роботи застосунку за парадигмою offline-first без обов'язкового підключення до глобальної мережі Інтернет, а також суворими міркуваннями конфіденційності, захисту та неможливості несанкціонованого витоку персональних даних б'юті-клієнтів. Як система управління базою даних обрано SQLite (втілену через асинхронний Flutter-пакет sqflite) — легковісну вбудовану реляційну СКБД, що є загальноприйнятим промисловим стандартом для локального збереження структурованих даних у мобільних системах і не потребує розгортання окремого серверного процесу.

Предметна область мобільного застосунку характеризується оптимізованою інформаційною моделлю, де центральною та єдиною реляційною сутністю, що підлягає персистентному збереженню, є запис клієнта. Всі інші операційні дані, такі як періодична фінансова статистика, аналітична хронологія та історія відвідувань, а також тимчасові параметри для локальних нагадувань, є похідними від цієї сутності й обчислюються динамічно на основі накопичених у базі даних рядків за допомогою агрегаційних SQL-функцій. Така організація сховища даних дозволяє ефективно обійтися єдиною реляційною таблицею без проєктування складних, важких для мобільних процесорів міжтабличних зв'язків Join, що повністю відповідає критеріям мінімальної достатності, знижує навантаження на підсистему пам'яті пристрою та спрощує подальший супровід і міграцію схеми бази даних.

Логічна модель єдиної таблиці записів б'юті-клієнта складається з фіксованого набору типізованих атрибутів, кожен з яких виконує чітку інженерну роль і має відповідні обмеження цілісності:

– первинний ідентифікатор id типізований як INTEGER із прапорцями PRIMARY KEY та AUTOINCREMENT для забезпечення унікальності кожного рядка та автоматичного нарощування ключа;

– обов'язкове текстове поле clientName із обмеженням NOT NULL слугує для збереження імені клієнта у кириличному форматі;

– обов'язковий атрибут dateTime із типом TEXT та обмеженням NOT NULL фіксує точну дату і час запланованого прийому;

– необов'язкове текстове поле phone із міткою NULL призначене для запису контактного номера мобільного телефону клієнта;

– числове поле price із типом даних REAL та обмеженням NULL зберігає грошову вартість наданої послуги з точністю до копійок;

– текстовий атрибут note із прапорцем NULL дозволяє користувачеві вносити довільні робочі нотатки, коментарі та формулювати специфіку сеансу;

– опціональне текстове поле photoPath із обмеженням NULL реєструє строковий відносний шлях до файлу графічного зображення результату виконаної роботи.

Візуалізована схема структури таблиці локальної бази даних із зазначенням внутрішніх типів SQLite, обмежень та взаємозв'язків полів відображена на інженерній діаграмі на рисунку 2.2.

Атрибут dateTime фіксується у текстовому форматі відповідно до міжнародного стандарту ISO 8601 (у вигляді строки YYYY-MM-DD HH:MM:SS), що забезпечує однозначне представлення часових міток незалежно від регіональних налаштувань операційної системи смартфона та дозволяє виконувати швидке високопродуктивне сортування записів у хронологічному порядку безпосередньо засобами SQL-запитів. Зворотна конвертація між строковим представленням та об'єктом DateTime мови Dart здійснюється на рівні data-mapping методів моделі даних при читанні та коміті транзакцій у базу. Текстовий атрибут photoPath зберігає виключно відносний шлях до файлу зображення всередині локальної директорії додатку, а не сам графічний об'єкт у

бінарному вигляді BLOB, що суттєво мінімізує фізичний розмір файлу бази даних, усуває дублювання інформації та оптимізує швидкість виконання запитів.

APPOINTMENTS

Поле	Тип даних	Обмеження	Опис
id PK	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор запису
clientName NN	TEXT	NOT NULL	Ім'я клієнта
dateTime NN	TEXT	NOT NULL	Дата та час прийому (ISO 8601)
phone	TEXT	NULL	Контактний номер телефону
price	REAL	NULL	Вартість послуги (₴)
note	TEXT	NULL	Довільні нотатки
photoPath	TEXT	NULL	Шлях до прикріпленого фото

PK — первинний ключ NN — обов'язкове поле решта полів — необов'язкові (NULL)

Рисунок 2.2 – Схема таблиці бази даних

Окрім внутрішньої структури таблиці, архітектурний рівень даних містить у собі ізольований механізм резервного копіювання, реалізований через автономний сервіс. Резервна копія створюється як повний побайтовий знімок (snapshot) поточного файлу бази даних SQLite та копіюється у захищену системну директорію, при цьому відновлення інформації відбувається шляхом безпечної атомарної заміни робочого файлу бази даних обраним архівом. Спроєктована структура є оптимальною для виконання цільових сценаріїв використання (use cases), таких як швидка вибірка візитів за дату, пошукова фільтрація за клієнтом та аналітичне агрегування фінансових показників.

У межах системного проєктування було виконано декомпозицію користувацького інтерфейсу та функціональних модулів застосунку на основі

сценаріїв взаємодії користувача (use cases), що дозволило чітко розмежувати зони відповідальності та сформувати логіку навігаційних переходів. Модульна декомпозиція інтерфейсу користувача (UI) передбачає поділ візуального простору на вісім функціональних екранів, кожен з яких реалізований як комбінація високорівневих віджетів управління станом та низькорівневих атомарних компонентів відображення. Зв'язок між сценаріями використання та UI-компонентами структурований у вигляді послідовних взаємодій у межах єдиного навігаційного стеку, що описує наступну архітектурну логіку:

– сценарій перегляду поточних планів та загального розкладу ініціюється з головного екрана, який виступає кореневим елементом навігації, обробляє події натискання на елементи керування та забезпечує маршрутизацію користувача до будь-якого дочірнього вікна;

– сценарій створення або модифікації картки клієнта повністю обслуговується екраном створення та редагування запису, який динамічно змінює свій стан залежно від контексту виклику, проводить валідацію полів та передає сформовані структури даних на сервісний рівень;

– сценарій формування візуального портфоліо майстра реалізується через взаємодію екрана перегляду фотографії (який відповідає за повноекранний рендеринг зображень із підтримкою тач-жестів) та екрана галереї фотографій (який виконує посторінкову пагінацію та динамічний текстовий пошук медіафайлів);

– сценарій аудиту взаємодії з клієнтами забезпечується екраном історії відвідувань, який реалізує бізнес-логіку хронологічного сортування та надсилає параметризовані фільтри до сервісу даних для відображення повного списку минулих візитів конкретної особи;

– сценарій оперативного контролю завантаженості робочих днів інтегрований у екран календаря, який зчитує масив агрегованих записів та виконує кольорове кодування дат для наочного запобігання виникненню тайм-конфліктів у розкладі майстра;

– сценарій аналізу фінансової ефективності діяльності покладено на екран статистики, який трансформує підраховані масиви числових даних доходів у графічні гістограми за допомогою декларативних бібліотек рендерингу;

– сценарій конфігурування системних сповіщень та безпеки сховища керується через екран налаштувань, що надає інтерфейс для встановлення часових міток нагадувань та ініціює ручні тригери для створення резервних копій бази даних.

Навігаційна структура проєкту функціонує за принципом лінійного та модального проштовхування екранів у нативний стек (Push/Pop міграція), де головний екран завжди залишається базовим шаром, а екрани редагування, статистики та календаря відкриваються як повнорозмірні шари із збереженням стану попереднього екрана. Екрани медіаперегляду та налаштувань нагадувань інтегруються як тимчасові дочірні віджети, що автоматично звільняють займану оперативну пам'ять при закритті користувачем. Такий підхід до декомпозиції UI та логічної структуризації за сценаріями використання забезпечує високу модульність системи, мінімізує взаємну залежність компонентів (досягається низька зв'язність Low Coupling) та гарантує стабільне функціонування інтерфейсу на мобільних пристроях з різними конфігураціями екранів та обсягами пам'яті.

2.3 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача є важливим етапом розробки мобільного застосунку, що визначає структуру взаємодії між користувачем і програмним продуктом. На цьому етапі формуються принципи організації екранів, логіка навігації між ними, а також склад і розташування елементів керування. Метою проєктування є забезпечення зручного, інтуїтивно зрозумілого та ергономічного інтерфейсу, що дозволяє майстру ефективно виконувати повсякденні робочі задачі без спеціальної технічної підготовки.

При проєктуванні інтерфейсу застосунку BeautyDiary було визначено такі ключові принципи.

Принцип мінімалізму передбачає відмову від надлишкових декоративних елементів на користь чистого, функціонального оформлення. Кольорова палітра базується на поєднанні світлого фону із темними елементами керування та акцентними кольорами для позначення важливої інформації. Такий підхід знижує когнітивне навантаження на користувача та сприяє швидкому сприйняттю інформації.

Принцип узгодженості передбачає єдиний стиль оформлення всіх екранів застосунку: однакова типографіка, єдина система відступів, однотипні кнопки та поля введення. Заголовки екранів розміщені у верхній панелі навігації на темному фоні, що забезпечує чітку візуальну ієрархію.

Принцип зрозумілості елементів керування реалізовано через використання загальноприйнятих піктограм: іконка годинника для переходу до історії записів, іконка зображення для галереї, іконка гістограми для статистики, іконка шестерні для налаштувань. Усі написи виконано українською мовою.

Принцип мінімізації кількості дій передбачає, що найбільш часто виконувана операція — створення нового запису — доступна з головного екрана через виразну кнопку з іконкою «+», розміщену у нижньому правому куті екрана.

Навігаційна структура застосунку побудована на основі ієрархічної моделі з використанням стекової навігації. Головний екран є відправною точкою та надає доступ до всіх функціональних розділів через панель іконок у верхній частині екрана. Повернення на попередній екран здійснюється через стандартну стрілку у верхньому лівому куті, що відповідає загальноприйнятим конвенціям мобільних застосунків. Додаткові дії, що використовуються рідше — експорт даних, додавання прикладів, очищення записів — приховані у контекстному меню, що викликається через кнопку «...» у верхній панелі, що дозволяє не перевантажувати основний інтерфейс.

Схему навігації між екранами застосунку наведено на рисунку 2.3.

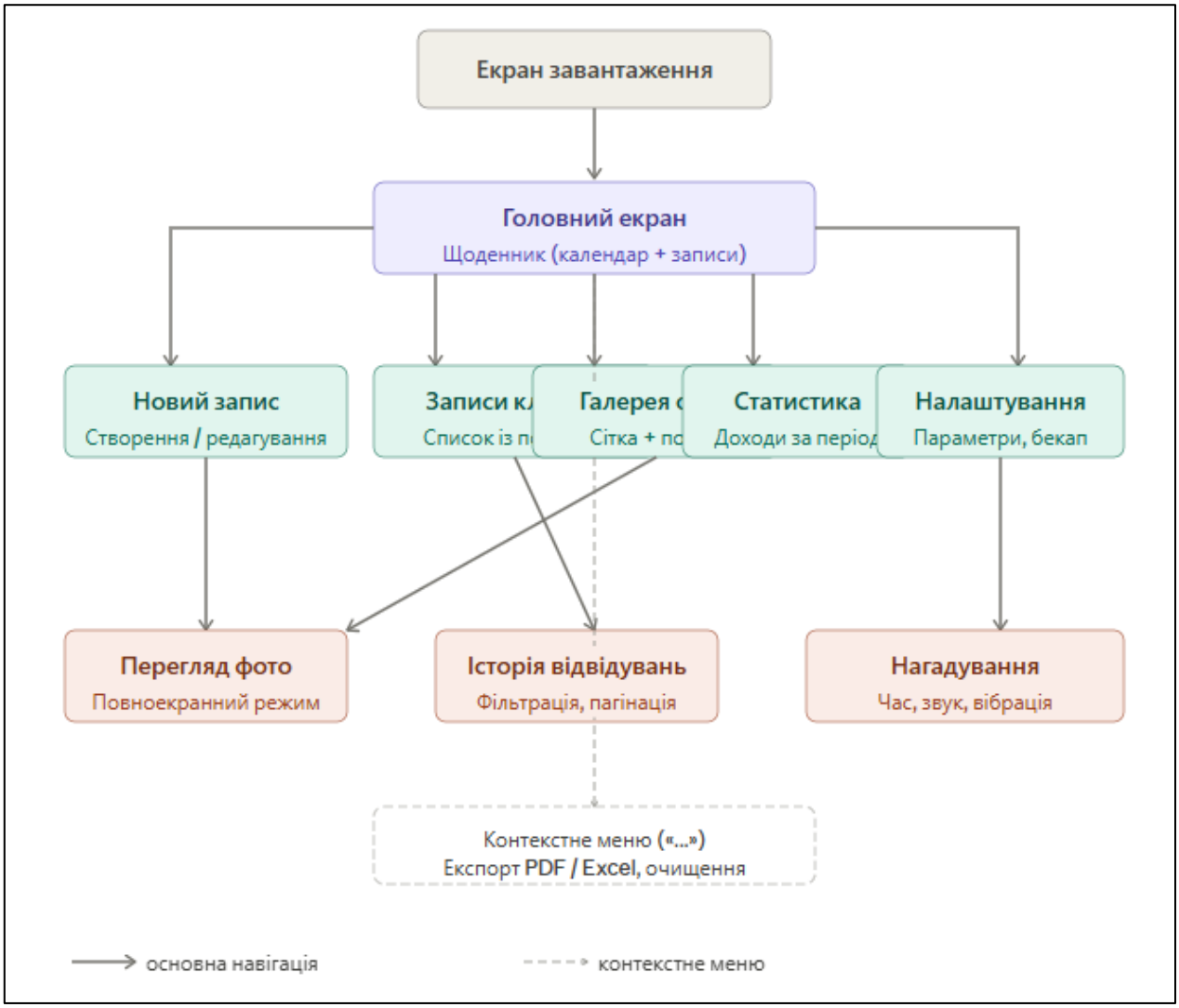


Рисунок 2.3 – Схема навігації між екранами

Екран завантаження є першим, що бачить користувач при запуску застосунку. Він виконаний у мінімалістичному стилі: світлий фон, по центру — логотип застосунку та індикатор завантаження з підписом «Завантаження...». Цей екран відображається протягом ініціалізації бази даних та підсистеми нагадувань.

Головний екран є центральним у навігаційній ієрархії застосунку. У верхній частині розміщено заголовок «Щоденник» та панель швидкого доступу з п'ятьма іконками: перехід до історії записів, галереї фотографій, фінансової статистики, налаштувань та контекстного меню. Основну частину екрана займає календарний вигляд поточного місяця, де дні з записами позначені кольоровими кружечками

із цифрою, що відображає кількість записів: зелений колір відповідає одному запису, помаранчевий — від двох до чотирьох, синьо-фіолетовий — п'яти і більше. Поточна дата виділяється окремим кольором. Нижче календаря відображається список записів на вибраний день у вигляді карток. Кожна картка містить час прийому, ім'я клієнта, вартість послуги у форматі «XXX₴» та кнопку видалення. Кнопка додавання нового запису розміщена у нижньому правому куті у вигляді темного кружечка з іконкою «+».

Екран створення та редагування запису містить форму з полями для введення даних клієнта. Поле імені клієнта є обов'язковим. Дата та час прийому відображаються підкресленим текстом і є клікабельними — натискання відкриває відповідні діалогові вікна вибору. Поруч з датою та часом розміщено кнопку «Змінити» для зручного доступу. Поля телефону, ціни та опису позначені як необов'язкові безпосередньо у підказці поля. Нижче розміщено кнопку вибору фотографії з галереї та кнопку «Зберегти».

Екран записів клієнтів відображає повний список усіх записів, згрупованих за датою. Для кожної дати відображається загальна сума доходу за день та кількість записів. Записи всередині групи відображаються у хронологічному порядку із зазначенням часу та вартості. У верхній частині розміщено рядок пошуку за іменем клієнта та індикатор поточної сторінки з кнопками посторінкової навігації.

Екран галереї фотографій відображає прикріплені до записів фотографії у вигляді сітки з двох стовпців. Під кожним зображенням відображається ім'я клієнта, дата та вартість послуги. У верхній частині розміщено рядок пошуку за іменем клієнта та посторінкова навігація.

Екран фінансової статистики відображає зведену інформацію про доходи за обраний часовий період. У верхній частині розміщено кнопку вибору періоду та загальну суму доходів. Нижче відображається детальний список записів за обраний період, згрупований за датами із зазначенням загальної суми по кожній даті та кількості записів.

Екран налаштувань містить поля для встановлення мінімального інтервалу між записами у хвилинах, мінімальної ціни запису та кількості цифр у номері телефону для валідації. Окремою кнопкою надається перехід до екрана налаштувань нагадувань. Нижче розміщено перемикачі автоматичного резервного копіювання та синхронізації, а також кнопки ручного резервного копіювання та збереження налаштувань.

Екран налаштувань нагадувань містить перемикач загального увімкнення нагадувань, відображення поточного часу ранкового нагадування та відступу перед записом у хвилинах, а також перемикачі звуку та вібрації. Збереження налаштувань здійснюється кнопкою «Зберегти».

Таким чином, спроєктований інтерфейс застосунку BeautyDiary відповідає принципам ергономіки мобільних пристроїв, забезпечує логічну та зрозумілу навігацію між екранами і дозволяє користувачу ефективно виконувати всі необхідні задачі з мінімальною кількістю дій.

2.4 Розроблення алгоритму роботи мобільного застосунку

Алгоритм роботи застосунку BeautyDiary описує послідовність дій, що виконуються системою у відповідь на команди користувача, а також автономно у фоновому режимі. Розроблення алгоритму передбачає визначення повного набору вихідних даних, мети його виконання та системи доступних команд.

Початковим станом вважається момент запуску застосунку на мобільному пристрої під керуванням Android (версія 6.0 та вище) із встановленими дозволами на відображення сповіщень та доступ до галереї зображень. У цей момент відбувається ініціалізація служби бази даних DbService (перевірка наявності файлу SQLite, створення схеми таблиць або її міграція при оновленні) та служби сповіщень NotificationService (завантаження налаштувань зі SharedPreferences, скасування застарілих нагадувань і планування актуальних). Вхідними даними

для алгоритму є: ім'я клієнта, дата і час прийому, номер телефону, вартість послуги, текстові нотатки, шлях до фотографії з галереї пристрою, а також команди користувача через інтерфейс.

Застосунок реалізує такі групи операцій:

– навігація між екранами, відкриття головного екрану (HomeScreen) з календарем та списком записів, перехід до екрану нового запису кнопкою «+», перехід до екранів історії, галереї, статистики та налаштувань через піктограми верхньої панелі, відкриття контекстного меню з опціями експорту;

– управління записами, введення даних у форму нового запису, вибір дати та часу через системні діалоги DatePicker та TimePicker, прикріплення фотографії з галереї (image_picker), збереження запису до SQLite з одночасним оновленням розкладу сповіщень через rescheduleAll(), видалення запису;

– робота з календарем, навігація між місяцями, перемикання між місячним та тижневим видом, вибір дати для фільтрації списку записів; дні з записами позначаються кольоровими індикаторами — зелений (1–2 записи), помаранчевий (3–4), червоний (5 і більше);

– перегляд статистики та експорт, відображення доходів за обраний період у вигляді списку або гістограми, формування та збереження звіту у форматі PDF або Excel через ExportService;

– управління нагадуваннями, налаштування ранкового нагадування (morning summary) із зазначенням часу та нагадування перед записом (pre-event) із зазначенням кількості хвилин завчасно;

– резервне копіювання, автоматичне та ручне збереження резервної копії даних.

Після виконання алгоритму досягається один із цільових результатів: запис про візит клієнта збережено у базі даних, розклад нагадувань оновлено, фінансовий звіт сформовано та збережено у форматі PDF або Excel, або налаштування застосунку збережено у SharedPreferences і набули чинності.

Блок-схему основного алгоритму роботи застосунку BeautyDiary наведено на рисунку 2.4.

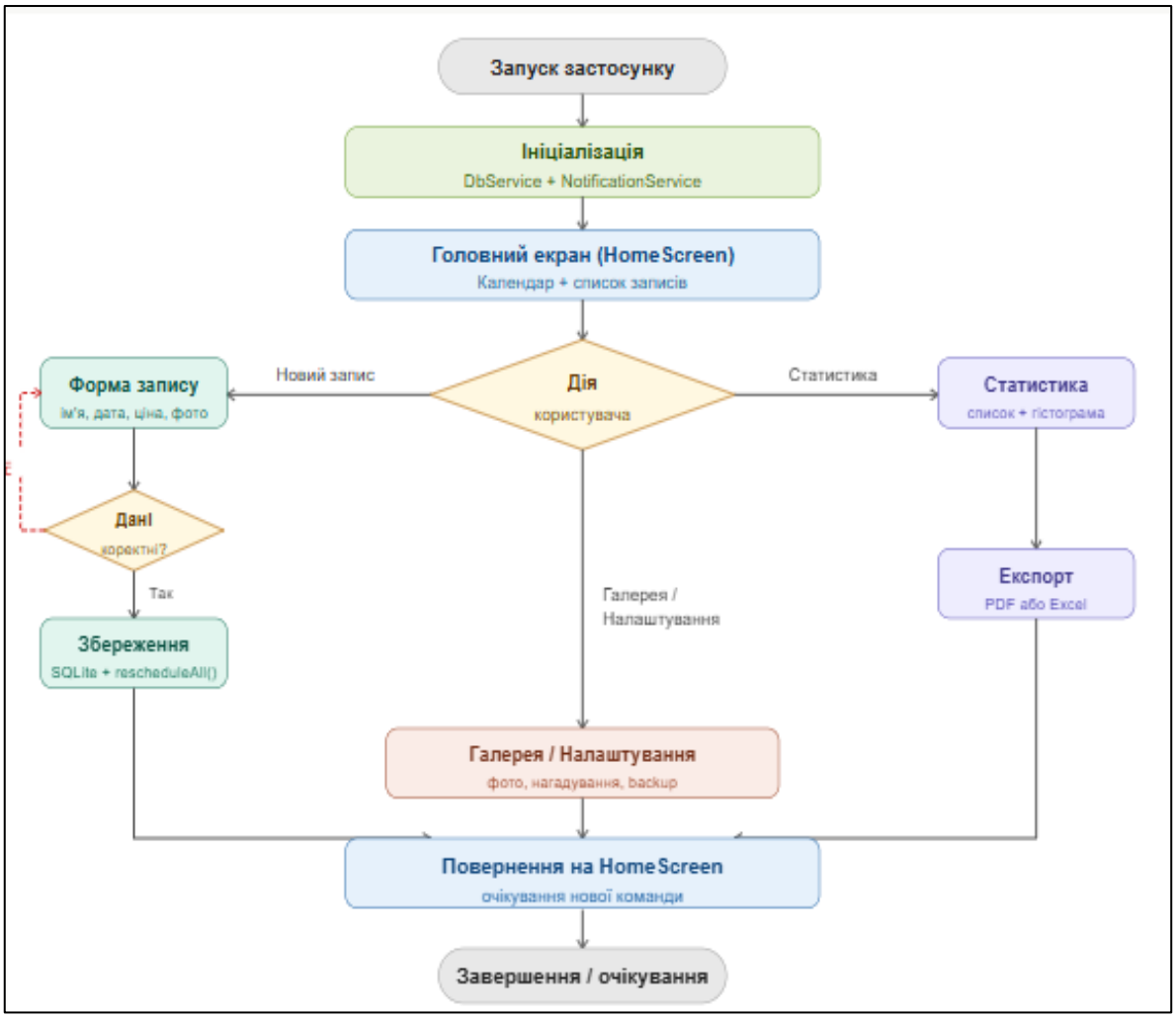


Рисунок 2.4 – Блок-схема основного алгоритму роботи

Три гілки алгоритму:

– ліва (teal) — створення запису з валідацією та збереженням;

– права (фіолетова) — статистика та експорт;

– центральна (коралова) — галерея та налаштування. Всі гілки повертаються до HomeScreen.

2.5 Створення прототипу мобільного застосунку

Прототипування є важливим етапом проєктування мобільного застосунку, що передує його програмній реалізації. Метою прототипування є перевірка та уточнення прийнятих проєктних рішень щодо організації інтерфейсу, навігаційної структури та розташування елементів керування ще до початку написання коду. Готовий прототип дозволяє виявити потенційні проблеми зручності використання на ранньому етапі та внести необхідні корективи без значних витрат часу.

У процесі розробки застосунку BeautyDiary було створено статичний прототип основних екранів, що відображає структуру інтерфейсу, склад елементів кожного екрана та логіку переходів між ними. Прототипування здійснювалося з урахуванням визначених раніше функціональних вимог та принципів проєктування інтерфейсу, описаних у підрозділі 2.3.

На етапі прототипування було визначено та задокументовано такі ключові рішення щодо організації інтерфейсу.

Щодо головного екрана було вирішено поєднати календарний вигляд та список записів на одному екрані, зображено на рисунку 2.5. Таке рішення дозволяє користувачу одразу бачити як загальну картину завантаженості місяця, так і деталізований список записів на обраний день. Кнопка додавання нового запису розміщується у нижньому правому куті у вигляді плаваючої кнопки, що є стандартним рішенням для мобільних застосунків і забезпечує швидкий доступ до найбільш частої операції.

Щодо екрана створення запису було вирішено використати просту одноріневу форму без розбиття на кроки. Оскільки кількість полів є невеликою, покроковий майстер введення даних є надлишковим і лише ускладнює процес.

Щодо навігації між розділами було вирішено відмовитися від нижньої панелі навігації на користь іконок у верхній панелі головного екрана. Це рішення зумовлене тим, що більшість часу користувач перебуває саме на головному

екрані, а перехід до інших розділів здійснюється значно рідше. Верхня панель іконок займає менше місця та залишає більше простору для відображення календаря та списку записів.

Рисунок 2.5 – Прототип сторінок календарю та створення нового запису

Щодо екрана галереї було вирішено використати сіткове розташування фотографій у два стовпці з підписами імені клієнта, дати та вартості під кожним зображенням. Такий формат забезпечує максимальну інформативність при компактному відображенні та дозволяє швидко візуально ідентифікувати потрібну роботу.

На основі створеного прототипу та задокументованих рішень було сформовано остаточну схему роботи мобільного застосунку, що стала основою

для його подальшої програмної реалізації. Прототипи ключових екранів наведено на рисунках 2.5–2.7.

Рисунок 2.6 – Прототип галереї фото та екрану записів клієнтів.

У процесі прототипування також було проведено юзабіліті-оцінку спроєктованого інтерфейсу шляхом тестування на реальному об'єкті — приватному майстрі з манікюру та педикюру.

За результатами оцінки до початкового варіанту прототипу було внесено такі уточнення:

– кнопку зміни часу запису перенесено ближче до поля дати та часу для зручнішого доступу;

– поле ціни переміщено вище поля опису, оскільки воно заповнюється частіше;

– до екрана налаштувань додано можливість задавати мінімальний інтервал між записами та мінімальну ціну для валідації введених даних, що виявилося зручним для контролю коректності записів (рисунок 2.7).

Рисунок 2.7 – Прототип сторінок статистики та нашатуваннння

Таким чином, етап прототипування дозволив уточнити та обґрунтувати ключові рішення щодо організації інтерфейсу застосунку BeautyDiary ще до початку його програмної реалізації, що сприяло підвищенню якості кінцевого

продукту та скороченню часу на виправлення помилок проєктування на пізніших етапах.

2.6 Аналіз та вибір технологій і методів реалізації застосунку

Вибір технологій та інструментів розробки є одним із ключових проєктних рішень, що безпосередньо впливає на функціональні можливості застосунку, швидкість його розробки та якість кінцевого продукту. При виборі технологічного стеку для реалізації застосунку BeautyDiary розглядалися як підходи до розробки мобільних застосунків у цілому, так і конкретні інструменти для кожної підсистеми.

Першим ключовим рішенням є вибір підходу до розробки мобільного застосунку. Існує три основні підходи: нативна розробка, гібридна розробка та кросплатформна розробка.

Нативна розробка передбачає створення окремих застосунків для кожної платформи із використанням відповідних мов програмування та інструментів: Kotlin або Java для Android, Swift або Objective-C для iOS. Перевагами цього підходу є максимальна продуктивність та повний доступ до можливостей платформи. Проте для одноосібного розробника підтримка двох окремих кодових баз є економічно недоцільною.

Гібридна розробка на основі веб-технологій — зокрема із використанням фреймворків Ionic або Cordova — передбачає реалізацію застосунку як веб-сторінки, що виконується у вбудованому браузері. Такий підхід дозволяє використовувати єдину кодову базу, однак має суттєві обмеження у продуктивності та доступі до апаратних можливостей пристрою, що є критичним для роботи з локальною базою даних та системою сповіщень.

Кросплатформна розробка із використанням спеціалізованих фреймворків дозволяє створювати застосунки з єдиної кодової бази, що компілюються у

нативний код для кожної платформи. Серед найбільш поширених представників цього підходу — Flutter та React Native [12].

React Native є фреймворком від компанії Meta, що використовує мову JavaScript та забезпечує рендеринг через нативні компоненти платформи. Фреймворк має велику спільноту та широку екосистему бібліотек, однак може поступатися у продуктивності застосункам на Flutter через особливості архітектури міст між JavaScript та нативним кодом [13].

Flutter є кросплатформним фреймворком від компанії Google, що використовує мову програмування Dart та власний рушій рендерингу. На відміну від React Native, Flutter не використовує нативні компоненти платформи, а самостійно відмальовує весь інтерфейс, що забезпечує високу продуктивність та ідентичний зовнішній вигляд на різних пристроях. Flutter підтримує компіляцію у нативний ARM-код, що забезпечує продуктивність, наближену до нативних застосунків.

З урахуванням вимог до застосунку — зокрема необхідності роботи з локальною базою даних, підсистемою сповіщень, файловою системою пристрою та галереєю зображень — для реалізації BeautyDiary обрано фреймворк Flutter із мовою програмування Dart. Порівняльний аналіз розглянутих підходів наведено у таблиці 2.1.

Другим ключовим рішенням є вибір системи управління локальною базою даних. Розглядалися такі варіанти: SQLite через пакет sqflite, Hive та Isar [14].

Hive є легковісною NoSQL базою даних для Flutter, що зберігає дані у вигляді пар ключ-значення. Перевагою є висока швидкість читання та запису, проте відсутність підтримки реляційних запитів ускладнює фільтрацію та агрегування даних, що є необхідним для формування фінансової статистики.

Isar є більш сучасною NoSQL базою даних із підтримкою індексів та складних запитів. Проте на момент проєктування застосунку бібліотека мала відносно невелику спільноту та обмежену документацію порівняно з більш зрілими рішеннями.

Таблиця 2.1 — Порівняльний аналіз підходів до розробки мобільних застосунків

Критерій	Нативна	Гібридна	React Native	Flutter
Єдина кодова база	Ні	Так	Так	Так
Продуктивність	Висока	Низька	Середня	Висока
Доступ до API пристрою	Повний	Обмежений	Частковий	Повний
Робота з локальною БД	Так	Обмежено	Так	Так
Підтримка сповіщень	Так	Обмежено	Так	Так
Швидкість розробки	Низька	Висока	Середня	Висока
Обрано для проєкту	Ні	Ні	Ні	Так

SQLite через пакет sqflite є найбільш зрілим та широко застосовуваним рішенням для локального зберігання реляційних даних у Flutter-застосунках. Підтримує повний набір SQL-запитів, що забезпечує гнучкість при фільтрації, сортуванні та агрегуванні даних. Для застосунку BeautyDiary обрано саме цей варіант, оскільки необхідність виконання агрегованих запитів для формування фінансової статистики робить реляційну модель найбільш доцільною.

Перелік усіх обраних бібліотек та пакетів із обґрунтуванням їх вибору наведено нижче:

– sqflite — забезпечує роботу з локальною базою даних SQLite, підтримує повний набір SQL-запитів [19], що є необхідним для фільтрації, сортування та агрегування записів клієнтів [28];

– flutter_local_notifications — реалізує планування та відображення локальних push-сповіщень на пристрої [20], використовується для нагадувань обох типів [31];

– timezone — забезпечує коректний розрахунок часу сповіщень з урахуванням часового поясу пристрою користувача [24];

– image_picker — надає можливість вибору фотографій із галереї пристрою для прикріплення до запису клієнта;

– pdf та printing — використовуються спільно для формування та збереження PDF-файлів при експорті даних з україномовними заголовками та символом гривні ₴ [25];

– excel — забезпечує формування файлів у форматі Excel для експорту накопичених записів [26];

– intl — реалізує форматування дат, часу та числових значень відповідно до української локалі;

– shared_preferences — використовується для збереження налаштувань застосунку у вигляді пар ключ-значення між сеансами роботи.

Таким чином, обраний технологічний стек на основі Flutter та Dart із локальною базою даних SQLite повністю відповідає визначеним функціональним та нефункціональним вимогам до застосунку BeautyDiary. Використання зрілих та добре задокументованих бібліотек забезпечує надійність реалізації кожної з підсистем застосунку та знижує ризики, пов'язані із залежністю від зовнішніх компонентів.

2.7 Висновки

У другому розділі кваліфікаційної роботи було виконано комплексне проєктування мобільного застосунку BeautyDiary, що охопило всі основні аспекти — від вибору архітектурного підходу до визначення технологічного стеку реалізації. На основі отриманих результатів можна сформулювати такі висновки.

Для організації коду застосунку обрано архітектурний підхід на основі сервісів-синглтонів із трирівневою структурою: рівень представлення, рівень сервісів та рівень даних. Даний підхід є обґрунтованим вибором для мобільного застосунку даного масштабу та забезпечує чіткий розподіл відповідальності між компонентами, прозорість потоку даних та зручність подальшого розширення функціональності. Сервісний рівень складається з трьох компонентів — сервісу

бази даних, сервісу нагадувань та сервісу експорту — кожен з яких відповідає за окрему функціональну підсистему.

Спроєктована структура бази даних є мінімально достатньою для забезпечення повного функціоналу застосунку. Єдина таблиця записів клієнтів містить сім атрибутів, два з яких є обов'язковими. Поле збереження шляху до фотографії дозволяє пов'язати зображення з конкретним записом без збільшення обсягу бази даних. Вибір SQLite як системи управління базою даних зумовлений необхідністю виконання агрегованих запитів для формування фінансової статистики та зрілістю даного рішення для локального зберігання даних у мобільних застосунках.

Проєктування інтерфейсу користувача базується на принципах мінімалізму, узгодженості та мінімізації кількості дій для виконання основних операцій. Навігаційна структура застосунку охоплює вісім екранів, організованих за ієрархічним принципом із головним екраном як відправною точкою. Поєднання календарного вигляду та списку записів на головному екрані забезпечує швидкий доступ до найбільш затребуваної інформації без додаткових переходів.

Розроблений алгоритм роботи застосунку охоплює чотири основні сценарії використання: створення запису з валідацією введених даних, перегляд фінансової статистики за довільний період, експорт даних у формати PDF та Excel, а також резервне копіювання бази даних. Кожен сценарій передбачає обробку як успішного, так і помилкового результату виконання операції.

На етапі прототипування було уточнено ряд проєктних рішень за результатами юзабіліті-оцінки на реальному об'єкті, що дозволило підвищити зручність використання інтерфейсу ще до початку програмної реалізації [11].

Для реалізації застосунку обрано кросплатформний фреймворк Flutter із мовою програмування Dart, що забезпечує високу продуктивність, повний доступ до апаратних можливостей пристрою та можливість подальшого перенесення на платформу iOS. Технологічний стек доповнено набором спеціалізованих пакетів для роботи з базою даних, сповіщеннями, файловою системою, тощо.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Реалізація логіки мобільного застосунку

Реалізація логіки мобільного застосунку BeautyDiary охоплює сукупність механізмів обробки дій користувача, валідації введених даних, перевірки часових накладок між записами та автоматичного оновлення підсистеми нагадувань. У даному підрозділі описуються основні активності застосунку, порядок їх створення та взаємодії, а також ключові алгоритми бізнес-логіки [7].

У Flutter еквівалентом Android-активності є віджет, що реалізує власний стан. При навігації до нового екрана система створює екземпляр відповідного віджета, ініціалізує його стан через метод ініціалізації, де завантажуються збережені налаштування та встановлюються початкові значення полів. Взаємодія користувача з інтерфейсом змінює локальний стан, після чого фреймворк автоматично перебудовує відповідні частини дерева віджетів [8]. Перехід між екранами здійснюється через навігаційний стек із передачею необхідних аргументів. Наприклад, відкриття екрана перегляду фотографії з передачею шляху до файлу реалізується таким чином, рисунок 3.1.

```
Navigator.of(context).push(
  MaterialPageRoute(
    builder: (_) => PhotoViewerScreen(path: _imagePath!)
  )
);
```

Рисунок 3.1 – Кнопка відкриття екрану перегляду фотографії

Застосунок містить такі основні екрани з відповідною логікою взаємодії. Головний екран забезпечує відображення календаря та карток записів на обраний день, перехід до створення нового запису, а також навігацію до історії відвідувань та галереї фотографій. Екран створення та редагування запису є центральним з

точки зору бізнес-логіки і реалізує введення даних клієнта, вибір дати та часу, прикріплення фотографії, валідацію введених даних та збереження запису до бази даних. Екрани галереї та перегляду фотографії забезпечують відображення мініатюр із пошуком і посторінковою навігацією та повноекранний перегляд зображень із підтримкою масштабування. Екран історії відвідувань надає хронологічний перелік усіх записів із фільтрацією та посторінковою навігацією.

Ключовим елементом логіки застосунку є механізм валідації даних при збереженні запису. Валідація виконується на рівні форми засобами вбудованого механізму Flutter перед зверненням до бази даних. Перевіряються три групи умов.

Перша група — валідація обов'язкового поля імені клієнта [4]. Якщо поле порожнє або містить лише пробіли, форма повертає повідомлення про помилку і збереження не виконується, рисунок 3.2.

```
validator: (v) => v == null || v.trim().isEmpty
    ? 'Обов'язково'
    : null,
```

Рисунок 3.2 – Валідація поля імені клієнта

Друга група — валідація формату номера телефону. Поле є необов'язковим, однак якщо воно заповнене, перевіряється відповідність кількості введених цифр значенню, що налаштовується користувачем у розділі налаштувань. Введення нецифрових символів блокується на рівні форматувальника введення, рисунок 3.3

```
validator: (v) {
  if (v == null || v.trim().isEmpty) return null;
  final digits = v.replaceAll(RegExp(r'\D'), '');
  if (digits.length != phoneDigits)
    return 'Потрібно $phoneDigits цифр';
  return null;
},
```

Рисунок 3.3 – Валідація формату номера телефону

Третя група — валідація ціни. Перевіряється коректність числового формату та відповідність введеного значення мінімальній ціні, що також задається у налаштуваннях, рисунок 3.4.

```
validator: (v) {
  if (v == null || v.isEmpty) return null;
  final p = double.tryParse(v.replaceAll(',', '.'));
  if (p == null) return 'Невірний формат ціни';
  if (p < minPrice)
    return 'Мінімальна ціна: ${minPrice.toStringAsFixed(2)}';
  return null;
},
```

Рисунок 3.4 – Валідація ціни

Після успішного проходження валідації форми виконується перевірка на наявність часових накладок із вже існуючими записами. Метод збереження запису реалізовано таким чином, рисунок 3,5:

```
Future<void> _save() async {
  if (!_formKey.currentState!.validate()) return;
  final prefs = await SharedPreferences.getInstance();
  final minInterval = prefs.getInt('minInterval') ?? 60;
  final db = DBService();
  final conflict = await db.hasConflict(
    _dt, minInterval, ignoreId: _editingId);
  if (conflict) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
      content: Text('Конфлікт з існуючим записом')));
    return;
  }
final a = Appointment(
  id: _editingId,
  clientName: _naCtl.text.trim(),
  dateTime: _dt,
  phone: _phoneCtl.isEmpty ? null :
  price: _priceCtl.isEmpty
   ? null
  : double.tryParse(_priceCtl.text),
  note: _noteCtl..isEmpty ? null :
  photoPath: _imagePath,
);
if (_editingId == null) {
  await db.insertAppointment(a);
  await db.insertAppointment(a);
} else {
  await db.updateAppointment(a);
}

Navigator.of(context).pop();
}
```

Рисунок 3.5 – Метод збереження запису

Перевірка часових накладок між записами реалізована у сервісі бази даних як окремий метод. Алгоритм будує часовий інтервал навколо дати-кандидата з відступом у хвилинах, що налаштовується, та перевіряє наявність будь-якого запису в цьому інтервалі. При редагуванні існуючого запису його власний ідентифікатор виключається з перевірки, рисунок 3,6.

```dart
Future<bool> hasConflict(DateTime candidate,
    int minutesInterval, {int? ignoreId}) async {
  final database = await db;
  final from = candidate
      .subtract(Duration(minutes: minutesInterval));
  final to = candidate
      .add(Duration(minutes: minutesInterval));
  String where = 'dateTime > ? AND dateTime < ?';
  final args = <Object?>[
    from.toIso8601String(),
    to.toIso8601String()
  ];
  if (ignoreId != null) {
    where += ' AND id != ?';
    args.add(ignoreId);
  }
  final maps = await database.query(
      'appointments', where: where, whereArgs: args);
  return maps.isNotEmpty;
}
```

Рисунок 3.6 – Перевірка часових накладок між записами

Важливою особливістю реалізованої логіки є автоматична синхронізація нагадувань після будь-якої зміни у базі даних. Після кожної операції вставки, оновлення або видалення запису сервіс бази даних автоматично викликає перепланування всіх активних нагадувань, рисунок 3.7.

```dart
Future<int> insertAppointment(Appointment a) async {
  final database = await db;
  final res = await database.insert('appointments', a.toMap());
  unawaited(_maybeBackup());
  unawaited(NotificationService().rescheduleAll());
  return res;
}
```

Рисунок 3.7 – Автоматична синхронізація нагадувань

Збереження фотографії до файлової системи пристрою та фіксація шляху до неї у стані екрана реалізовано таким чином: після вибору зображення з галереї

його байти зчитуються та записуються до директорії застосунку з унікальним іменем файлу на основі поточного часового штампу, рисунок 3.8.

```
final bytes = await xfile.readAsBytes();
final dir = await getApplicationDocumentsDirectory();
final filename =
    'appointment_${DateTime.now().millisecondsSinceEpoch}'
    '${extension(xfile.path)}';
final out = File('${dir.path}/$filename');
await out.writeAsBytes(bytes);
setState(() { _imagePath = out.path; });
```

Рисунок 3.8 – Збереження фотографії

Планування ранкового нагадування реалізовано з урахуванням поточного часу: якщо запланований час вже минув сьогодні, нагадування автоматично переноситься на наступний день, рисунок 3.9.

```
var scheduled = tz.TZDateTime(
    tz.local, now.year, now.month, now.day, hour, minute);
if (scheduled.isBefore(now)) {
  scheduled = scheduled.add(const Duration(days: 1));
}
await _plugin.zonedSchedule(1000, title, body, scheduled,
    details,
    androidScheduleMode:
        AndroidScheduleMode.exactAllowWhileIdle,
    matchDateTimeComponents: DateTimeComponents.time);
```

Рисунок 3.9 – Планування ранкового нагадування

Таким чином, реалізована логіка застосунку забезпечує коректну обробку всіх основних сценаріїв взаємодії користувача, включаючи багаторівневу валідацію введених даних, запобігання часовим накладкам між записами та автоматичну синхронізацію нагадувань після будь-яких змін у базі даних.

3.2 Реалізація розмітки мобільного застосунку

Реалізація розмітки мобільного застосунку BeautyDiary базується на ієрархічній системі віджетів фреймворку Flutter. На відміну від нативної Android-розробки, де інтерфейс будується на основі ієрархії об'єктів View та ViewGroup, у Flutter кожен елемент інтерфейсу є віджетом — від найпростішої текстової мітки до складного екрана з формою [29]. Компонування інтерфейсу здійснюється шляхом вкладення віджетів один в одного, утворюючи дерево, яке фреймворк перетворює на пікселі через власний рушій рендерингу.

Основним контейнером кожного екрана застосунку є віджет Scaffold, що забезпечує стандартну структуру мобільного екрана: верхню панель навігації з заголовком та кнопкою повернення, основну область вмісту та плаваючу кнопку дії у нижньому правому куті. Така структура є уніфікованою для всіх екранів застосунку, що забезпечує узгодженість зовнішнього вигляду та поведінки навігації.

Для організації форми введення даних використовується поєднання віджетів Form та ListView. Використання Form як кореневого контейнера форми дозволяє застосовувати єдиний механізм валідації до всіх полів одночасно через виклик методу validate(), що суттєво спрощує логіку перевірки даних. ListView використовується як прокручуваний контейнер для полів форми, що забезпечує коректне відображення на пристроях з невеликим розміром екрана та запобігає перекриттю полів екранною клавіатурою.

Поля введення реалізовані на основі віджета TextFormField, що інтегрується з механізмом валідації форми та підтримує налаштування клавіатури, форматувальники введення та підказки. Інтерактивні елементи, що не є стандартними кнопками — зокрема поле дати та часу — обгортаються у GestureDetector або InkWell для обробки жесту натискання та відкриття відповідного діалогового вікна вибору.

Фрагмент розмітки екрана створення нового запису ілюструє загальну структуру організації інтерфейсних компонентів, рисунок 3.10.

```
return Scaffold(
  appBar: AppBar(
    title: Text(_editingId == null ? 'Новий запис' : 'Редагувати запис')),
  body: Padding(
    padding: const EdgeInsets.all(12.0),
    child: Form(
      key: _formKey,
      child: ListView(
        children: [ Row(children: [
          Expanded(child: Text( controller: _pickDateTime, decoration: const InputDa oration(labelText: 'Дата і час'),
              ),
              ElevatedButton.icon(onPressed: _pickDateTime, icon: Icon(Icons.access_time), label: Text('Змінити'),
          }),
          TextFormField(controller: _phoneCtl, decoration: const InputDatoration(labelText: 'Телефон (необов\'язково)'),
          TextFormField(controller: _priceCtl, decoration: const InputDatoration(labelText: 'Ціна (необов'язково)'),
          if (_imagePath != null) ...[
            GestureDetector(
              onTap: () => Navigator.push(context,
                            MaterialPageRoute(builder: (_) => PhotoViewerScreen(path: _imagePath!)),
              child: Image.file(File(_imagePath!)),
          ),
      ],
    ),
    floatingActionButton: FloatingActionButton(onPressed: _save, child: Icon(Icons.save),
  ),
};
```

Рисунок 3.10 – Фрагмент розмітки екранна

Поле дати та часу реалізовано у вигляді рядка, що поєднує два елементи: клікабельний текст із поточними значенням дати та часу та кнопку зміни поруч. Дата форматується відповідно до української локалі за допомогою пакету intl, що забезпечує відображення у звичному для користувача форматі — наприклад, «11 трав. 2026 р.». Такий підхід є більш компактним порівняно зі стандартними полями вибору дати та відповідає мінімалістичній концепції інтерфейсу.

Відображення прикріпленого зображення реалізовано умовно — блок з мініатюрою фотографії відображається лише у разі наявності збереженого шляху до файлу. Мініатюра обгорнута у GestureDetector, що відкриває екран повноекранного перегляду при натисканні. Зображення завантажується безпосередньо з файлової системи пристрою через Image.file, що не потребує мережевого з'єднання та забезпечує швидке відображення.

Налаштування застосунку — мінімальна ціна, кількість цифр номера телефону та мінімальний інтервал між записами — зчитуються зі сховища SharedPreferences під час ініціалізації екрана та застосовуються у логіці валідації

форми. Це дозволяє майстру гнучко налаштовувати правила перевірки даних без зміни коду застосунку.

Для головного екрана використовується кастомна реалізація календарної сітки на основі GridView із розрахунком кількості днів у місяці та зміщення першого дня тижня. Дні, що мають записи, позначаються кольоровими кружечками із числовим лічильником. Кольорове кодування залежить від кількості записів на день: один запис відображається зеленим кольором, від двох до чотирьох — помаранчевим, п'ять і більше — синьо-фіолетовим. Нижче календарної сітки розміщено список карток записів на обраний день, кожна з яких відображає час прийому, ім'я клієнта, вартість послуги та кнопку видалення.

Галерея фотографій реалізована на основі GridView з двома стовпцями. Кожна комірка галереї містить зображення, що завантажується з локальної файлової системи, та підписи з іменем клієнта, датою та вартістю послуги. Для забезпечення продуктивності при великій кількості фотографій реалізовано посторінкову навігацію, що обмежує кількість одночасно завантажених зображень.

Таким чином, розмітка застосунку BeautyDiary побудована відповідно до сучасних практик Flutter-розробки: розділення інтерфейсу на невеликі спеціалізовані компоненти, використання форм із вбудованою валідацією, адаптивність завдяки прокручуваним контейнерам та коректна робота з локальними файлами зображень.

3.3 Розроблення бази даних

Розроблення бази даних застосунку BeautyDiary передбачає реалізацію локального сховища даних на основі SQLite з використанням пакету sqflite для Flutter. Вибір SQLite як системи управління базою даних зумовлений її вбудованістю у мобільну операційну систему Android, відсутністю необхідності

у зовнішньому серверному процесі, підтримкою повного набору SQL-запитів та зрілістю рішення для застосунків із невеликим обсягом локально збережених даних.

Доступ до бази даних централізовано через сервіс DbService, реалізований за патерном «Одинак». Такий підхід гарантує існування єдиного підключення до бази даних протягом усього життєвого циклу застосунку та унеможливлює конфлікти при одночасному зверненні до спільного ресурсу з різних екранів.

Ініціалізація бази даних виконується при першому зверненні до сервісу. Метод ініціалізації визначає шлях до файлу бази даних у директорії застосунку, відкриває або створює файл бази та виконує необхідні операції зі схемою залежно від поточної версії. Реалізацію методу ініціалізації наведено на рисунку 3.11.

```
Future<Database> _initDB() async {
  final databasesPath = await getDatabasesPath();
  final path = join(databasesPath, 'beautydiary.db');
  return await openDatabase(
    path,
    version: 2,
    onCreate: (db, v) async {
      await db.execute('''
        CREATE TABLE appointments(
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          clientName TEXT NOT NULL,
          dateTime TEXT NOT NULL,
          phone TEXT,
          price REAL,
          note TEXT,
          photoPath TEXT
        )
      ''');
    },
    onUpgrade: (db, oldV, newV) async {
      if (oldV < 2) {
        await db.execute(
          'ALTER TABLE appointments ADD COLUMN photoPath TEXT'
        );
      }
    },
  );
```

Рисунок 3.11 – Метод ініціалізації

База даних має версію 2, що відображає факт проведеної міграції схеми. При першому встановленні застосунку виконується обробник onCreate, що створює таблицю appointments з повною схемою. При оновленні застосунку з попередньої версії виконується обробник onUpgrade, що додає поле photoPath до існуючої таблиці засобами оператора ALTER TABLE. Така організація міграцій гарантує збереження накопичених даних користувача при оновленні застосунку.

Поле dateTime зберігається у текстовому форматі відповідно до стандарту ISO 8601. Це забезпечує можливість виконання коректних порівнянь і сортування часових значень засобами SQL-запитів без необхідності конвертації на стороні бази даних. Конвертація між текстовим представленням та об'єктом DateTime виконується на рівні моделі даних у методах toMap та fromMap класу Appointment.

Сервіс DbService надає повний набір методів для виконання операцій читання та запису. Метод отримання записів за конкретний день формує часовий діапазон від початку до кінця доби та виконує вибірку із сортуванням у хронологічному порядку, рисунок 3.12.

```dart
Future<List<Appointment>> appointmentsForDay(
    DateTime day) async {
  final database = await db;
  final start = DateTime(day.year, day.month, day.day);
  final end = start.add(Duration(days: 1));
  final maps = await database.query(
    'appointments',
    where: 'dateTime >= ? AND dateTime < ?',
    whereArgs: [
      start.toIso8601String(),
      end.toIso8601String()
    ],
    orderBy: 'dateTime ASC',
  );
  return maps.map((m) => Appointment.fromMap(m)).toList();
}
```

Рисунок 3.12 – Метод отримання записів за конкретний день

Повний перелік методів доступу до даних, що реалізовані у сервісі DbService, наведено нижче:

– insertAppointment — додавання нового запису клієнта до таблиці;
– updateAppointment — оновлення існуючого запису за ідентифікатором;
– deleteAppointment — видалення запису за ідентифікатором;
– appointmentsForDay — отримання списку записів на конкретний день;
– allAppointments — отримання повного переліку всіх записів;
– hasConflict — перевірка наявності часових накладок із існуючими записами.

Після кожної операції модифікації даних сервіс автоматично ініціює два фонових процеси: перепланування всіх активних нагадувань через NotificationService та за необхідності створення резервної копії через метод автоматичного резервного копіювання. Обидва процеси виконуються асинхронно без блокування основного потоку виконання, що забезпечує миттєву реакцію інтерфейсу на дії користувача.

Механізм резервного копіювання реалізовано у окремому сервісі BackupService. Резервна копія являє собою повний знімок файлу бази даних SQLite, що зберігається у відповідній директорії на пристрої. Застосунок підтримує два режими резервного копіювання. В автоматичному режимі резервна копія створюється після кожної операції модифікації даних, якщо відповідний перемикач увімкнено у налаштуваннях. У ручному режимі користувач ініціює створення резервної копії самостійно через відповідну кнопку в екрані налаштувань. Відновлення даних із резервної копії виконується шляхом заміни поточного файлу бази даних збереженою копією з подальшою реініціалізацією підключення.

З точки зору безпеки файл бази даних зберігається у внутрішній директорії застосунку, доступ до якої обмежений правами операційної системи Android. Дані зберігаються без шифрування, що є прийнятним для персонального використання приватним майстром. У разі необхідності захисту персональних даних клієнтів

рекомендується розглянути можливість додавання шифрування файлу бази даних на етапі подальшого розвитку застосунку.

Таким чином, реалізована база даних забезпечує надійне локальне зберігання всіх даних застосунку, підтримує версійні міграції схеми зі збереженням існуючих даних та інтегрована з підсистемами нагадувань і резервного копіювання через механізм автоматичного виклику після операцій модифікації.

3.4 Керівництво користувача

Керівництво користувача містить опис умов та принципів використання мобільного застосунку BeautyDiary, послідовність дій для виконання основних функцій, а також відповіді на поширені питання щодо роботи застосунку.

Застосунок призначений для приватного майстра з манікюру та педикюру і не потребує спеціальної технічної підготовки для використання. Усі написи та повідомлення виконано українською мовою. Робота застосунку не потребує підключення до мережі Інтернет.

Запуск застосунку здійснюється стандартним способом — натисканням іконки на головному екрані пристрою. Після запуску відображається екран завантаження з логотипом застосунку та індикатором прогресу, протягом якого виконується ініціалізація бази даних та підсистеми нагадувань. Після завершення завантаження користувач автоматично переходить на головний екран.

Головний екран відображає календар поточного місяця та список записів на обраний день. Для перегляду записів на інший день достатньо натиснути на відповідну дату у календарі. Дні, що мають записи, позначені кольоровими кружечками із числом — кількістю записів на цей день. Перехід між місяцями здійснюється стрілками ліворуч та праворуч від назви місяця. У верхній панелі

головного екрана розміщено іконки швидкого доступу до основних розділів застосунку.

Створення нового запису клієнта виконується у такій послідовності:

– натиснути кнопку «+» у нижньому правому куті головного екрана;

– заповнити обов'язкове поле імені клієнта;

– за потреби змінити дату та час прийому, натиснувши на підкреслений текст дати або кнопку «Змінити» — за замовчуванням встановлюється поточна дата та час 00:00;

– за бажанням заповнити необов'язкові поля телефону, ціни та опису;

– за бажанням прикріпити фотографію результату роботи, натиснувши кнопку «З галереї»;

– натиснути кнопку «Зберегти».

Якщо введені дані не відповідають встановленим вимогам — наприклад, поле імені порожнє, номер телефону має некоректну кількість цифр або ціна нижча за встановлений мінімум — застосунок відобразить відповідне повідомлення про помилку і збереження не виконається. Якщо обраний час прийому накладається на вже існуючий запис у межах встановленого мінімального інтервалу, користувачу відображається повідомлення про конфлікт запису.

Редагування існуючого запису виконується натисканням на картку запису у списку. Відкривається екран редагування з заповненими полями, де користувач може змінити будь-які дані та зберегти їх кнопкою «Зберегти». Видалення запису здійснюється натисканням іконки кошика на картці запису у списку головного екрана.

Перегляд повного списку записів усіх клієнтів доступний через іконку годинника у верхній панелі головного екрана. Записи згруповані за датами із відображенням загального доходу за кожен день. Для пошуку конкретного клієнта слід ввести його ім'я у рядок пошуку у верхній частині екрана. Для навігації між сторінками використовуються стрілки посторінкової навігації.

Галерея фотографій відкривається через іконку зображення у верхній панелі головного екрана та відображає всі прикріплені до записів фотографії у вигляді сітки з двох стовпців. Пошук у галереї виконується за іменем клієнта. Натискання на мініатюру відкриває фотографію у повноекранному режимі з підтримкою масштабування жестом зведення пальців.

Перегляд фінансової статистики доступний через іконку гістограми у верхній панелі головного екрана. За замовчуванням відображається статистика за поточний день із загальною сумою доходів та деталізованим переліком записів. Для зміни періоду слід натиснути кнопку «Вибрати період» та задати початкову і кінцеву дату у діалоговому вікні.

Налаштування застосунку відкриваються через іконку шестерні у верхній панелі. У цьому розділі доступні такі параметри:

– мінімальний інтервал між записами у хвилинах — визначає мінімально допустимий проміжок часу між двома записами при перевірці на накладки;

– мінімальна ціна запису — визначає нижню межу допустимого значення ціни при валідації форми;

– кількість цифр у номері телефону — визначає очікувану довжину номера для перевірки коректності введення;

– перемикач автоматичного резервного копіювання;

– кнопка ручного резервного копіювання «Резервне копіювання зараз»;

– кнопка переходу до налаштувань нагадувань.

У розділі налаштувань нагадувань доступні такі параметри: загальний перемикач увімкнення нагадувань, час ранкового нагадування з оглядом розкладу на день, відступ у хвилинах перед записом для нагадування перед подією, а також перемикачі звукового супроводу та вібрації.

Експорт даних у формати PDF та Excel виконується через контекстне меню, що відкривається натисканням кнопки «...» у верхньому правому куті головного екрана. Після вибору формату сформований файл зберігається на пристрої та відкривається засобами операційної системи для перегляду або передачі.

Резервне копіювання рекомендується виконувати регулярно, особливо перед заміною або скиданням пристрою. Для відновлення даних із резервної копії слід звернутися до відповідного пункту в розділі налаштувань.

Щодо роботи нагадувань слід враховувати, що на деяких пристроях для коректної роботи точних сповіщень може знадобитися надати застосунку відповідний дозвіл у системних налаштуваннях Android.

3.5 Технічні характеристики мобільного застосунку

Для забезпечення коректної та безперебійної роботи застосунку BeautyDiary необхідно дотримуватись мінімальних вимог до апаратного та програмного забезпечення мобільного пристрою. У таблиці 3.1 наведено технічні характеристики, за яких гарантується успішне виконання застосунку.

Застосунок BeautyDiary розроблено на платформі Flutter (Dart) і орієнтовано на пристрої під керуванням операційної системи Android. Мінімальна підтримувана версія ОС — Android 6.0 (Marshmallow), що відповідає рівню API 23. Рекомендованою є версія Android 10.0 (API рівень 29) та вище, оскільки саме починаючи з цього рівня API забезпечується повноцінна підтримка точних сповіщень (exact alarms) засобами AlarmManager, що є необхідною умовою для коректної роботи нагадувань перед записом (pre-event) та ранкового огляду (morning summary).

Вимоги до апаратного забезпечення пристрою є такими:

– процесор: будь-який ARM-процесор з тактовою частотою не менше 1,2 ГГц; рекомендовано ARM Cortex-A53 або вище;

– оперативна пам'ять (RAM): мінімально — 1 ГБ, рекомендовано — 2 ГБ та більше; достатній обсяг оперативної пам'яті забезпечує швидке відкриття екранів та плавну прокрутку списків;

– внутрішня пам'ять пристрою: мінімально необхідний вільний обсяг для

встановлення застосунку — 50 МБ; додатково необхідно передбачити місце для локальної бази даних SQLite (до 10 МБ залежно від кількості записів) та фотографій, що прикріплюються до записів клієнтів;

– дисплей: мінімальна роздільна здатність 360 × 640 пікселів; рекомендована щільність пікселів — від 160 dpi (mdpi); інтерфейс застосунку є адаптивним і коректно відображається на екранах від 4,5 до 6,7 дюймів;

– сенсорний екран: підтримка мультитач-жестів обов'язкова, оскільки для масштабування фотографій у повноекранному перегляді (PhotoViewerScreen) використовується жест pinch-to-zoom.

Технічні характеристики розробленого застосунку BeautyDiary зведено у таблицю 3.1.

Таблиця 3.1 — Технічні характеристики розробленого застосунку

Параметр	Мінімальне значення	Рекомендоване значення
Операційна система	Android 6.0 (API 23)	Android 10.0 (API 29) і вище
Оперативна пам'ять	1 ГБ	2 ГБ і більше
Вільна внутрішня пам'ять	50 МБ	200 МБ і більше
Роздільна здатність екрану	360 × 640 px	1080 × 2400 px
Щільність пікселів	160 dpi (mdpi)	420 dpi (xxhdpi)
Версія Flutter SDK	3.0.0	3.19.0 і вище
Версія Dart SDK	2.17.0	3.3.0 і вище
Підключення до мережі	Не потребується	—

Застосунок не потребує постійного підключення до мережі Інтернет, оскільки всі дані зберігаються локально на пристрої у базі даних SQLite. Це забезпечує повну функціональність застосунку в умовах відсутності мережевого покриття, що є важливою перевагою для цільової аудиторії — майстрів салонів краси та приватних фахівців у сфері б'юті-послуг.

3.6 Тестування мобільного застосунку

Тестування застосунку BeautyDiary проводилося з метою перевірки відповідності реалізованого програмного продукту функціональним та нефункціональним вимогам, визначеним на етапі проєктування. У процесі тестування застосовувалися методи ручного та автоматизованого тестування, тестування інтерфейсу користувача, управління дефектами та тестування на сумісність.

3.6.1 Аналіз методів тестування мобільного застосунку

На початковому етапі було проведено аналіз та вибір методів тестування, що найкраще відповідають специфіці розробленого застосунку. Для мобільних застосунків на Flutter розрізняють три основні рівні тестування: модульне, віджет-тестування та інтеграційне.

Модульне тестування (unit-тести) призначене для перевірки окремих функцій та методів бізнес-логіки в ізоляції від інших компонентів. У контексті застосунку BeautyDiary модульне тестування є найбільш доцільним для перевірки методу hasConflict сервісу DbService, що реалізує алгоритм виявлення часових накладок між записами, а також для перевірки функцій форматування цін та парсингу дат. Виконання модульних тестів здійснюється командою flutter test.

Віджет-тестування призначене для перевірки окремих компонентів інтерфейсу користувача — зокрема форми створення запису, поведінки полів валідації та реакції інтерфейсу на некоректні вхідні дані. Цей рівень тестування дозволяє виявити помилки у логіці відображення без необхідності запуску повного застосунку на пристрої.

Інтеграційне тестування охоплює наскрізні сценарії використання застосунку — від дії користувача до отримання очікуваного результату. Типові

інтеграційні сценарії для BeautyDiary включають: створення запису та перевірку його відображення у списку, спробу створення запису з некоректними даними та перевірку відображення повідомлення про помилку, створення двох записів з накладкою у часі та перевірку блокування другого запису, а також експорт даних у PDF та перевірку структури сформованого файлу. Виконання інтеграційних тестів здійснюється засобами пакету integration_test у поєднанні з командою flutter drive [27].

Ручне тестування на реальному пристрої є обов'язковим для перевірки підсистеми нагадувань, поведінки застосунку у фоновому режимі та роботи з галереєю зображень, оскільки ці функції не можуть бути повноцінно відтворені на емуляторі.

Для тестування застосунку BeautyDiary було обрано та обґрунтовано такі методи і засоби:

– юніт-тестування (unit testing), перевірка окремих компонентів та моделей застосунку в ізольованому середовищі; зокрема, перевірялася коректність серіалізації та десеріалізації моделі Appointment (методи toMap() та fromMap()); засіб реалізації — пакет flutter_test;

– інтеграційне тестування (integration testing), перевірка взаємодії між компонентами застосунку в умовах, наближених до реальних; автоматизовані тести імітували дії користувача: відкриття форми, введення даних, збереження запису та перевірку його відображення на головному екрані; засіб реалізації — пакет integration_test (аналог Selenium для Flutter);

– тестування інтерфейсу користувача (UI/UX testing), перевірка зручності та коректності роботи елементів інтерфейсу: навігації між днями у календарі, функціональності кнопки «+», валідації полів введення та читабельності елементів на екрані статистики;

– управління дефектами (defect management), документування виявлених помилок із зазначенням кроків відтворення, очікуваного та фактичного результату, класифікація за серйозністю (severity) та пріоритетом (priority);

інструмент — Jira;

– тестування на сумісність (compatibility testing), перевірка коректності відображення та функціонування застосунку на пристроях з різними версіями операційної системи, роздільними здатностями екрану та апаратними конфігураціями.

Перевага обраного набору методів полягає у комплексному охопленні всіх рівнів тестування: від перевірки окремих функцій до поведінки застосунку на реальних пристроях. Автоматизовані тести забезпечують можливість регресійного тестування після кожного оновлення коду, тоді як ручне тестування дозволяє оцінити суб'єктивні аспекти зручності інтерфейсу.

3.6.2 Тестування мобільного застосунку

Тестування на емуляторі проводилося з використанням Android Virtual Device із образом Pixel на базі API рівня 30. Запуск застосунку на емуляторі здійснювався за допомогою команд, що зображені на рисунку 3.13.

```
# Отримання переліку доступних емуляторів
flutter emulators

# Запуск обраного емулятора
flutter emulators --launch <emulator_id>

# Запуск застосунку на емуляторі
flutter run -d <emulator_id>

# Отримання файлу бази даних з емулятора
adb exec-out run-as com.example.beautydiary \
  cat databases/beautydiary.db > beautydiary.db
```

Рисунок 3.13 – Команди для запуску застосунку на емуляторі.

У процесі тестування на емуляторі було прогнано такі тестові сценарії.

Перший сценарій — створення запису з повним набором валідних даних: ім'ям клієнта, датою та часом, номером телефону, ціною, описом та прикріпленою фотографією.

Другий сценарій — спроба збереження запису з порожнім полем імені клієнта для перевірки механізму валідації обов'язкових полів.

Третій сценарій — створення двох записів із часовим інтервалом, меншим за встановлений мінімальний інтервал, для перевірки алгоритму виявлення накладок.

Четвертий сценарій — експорт накопичених даних у формати PDF та Excel із перевіркою структури та вмісту сформованих файлів.

П'ятий сценарій — перевірка посторінкової навігації у галереї фотографій та екрані історії відвідувань при наявності значної кількості записів.

Для перевірки коректності серіалізації та десеріалізації моделі даних було реалізовано модульний тест, що перевіряє збереження всіх атрибутів запису при перетворенні між об'єктом та форматом бази даних, рисунок 3.14.

```
test('Appointment toMap/fromMap roundtrip', () {
  final a = Appointment(
    id: 1,
    clientName: 'Тест Клієнт',
    dateTime: DateTime(2026, 3, 18, 10, 30),
    phone: '0990001111',
    price: 250.0,
    note: 'Тестовий запис',
  );
  final m = a.toMap();
  expect(m['clientName'], 'Тест Клієнт');
  final b = Appointment.fromMap(m);
  expect(b.id, 1);
  expect(b.clientName, 'Тест Клієнт');
  expect(b.dateTime, DateTime(2026, 3, 18, 10, 30));
  expect(b.phone, '0990001111');
  expect(b.price, 250.0);
  expect(b.note, 'Тестовий запис');
});
```

Рисунок 3.14 – Модульний тест

Автоматизоване тестування (Integration Test). Для автоматизації тестування використано вбудований у Flutter інструмент Integration Test, що функціонує за принципом, аналогічним до Selenium, — дозволяє імітувати дії користувача без його фізичної участі.

Реалізовано такі автоматизовані тести:

– тест створення запису (create appointment flow), автоматизований тест відкривав форму нового запису через кнопку FloatingActionButton, вводив дані у чотири поля (ім'я клієнта «Test Klient», телефон «0990001111», ціна «350», опис), натискав кнопку «Зберегти» та перевіряв наявність збереженого запису на головному екрані, рисунок 3.15; тест пройдено успішно;

– юніт-тест моделі Appointment (toMap/fromMap roundtrip), перевіряв коректність перетворення об'єкта Appointment у формат Map для збереження в SQLite та зворотного відновлення з Map; перевірялися всі поля: id, clientName, dateTime, phone, price, note; тест пройдено успішно.

```dart
testWidgets('create appointment flow', (WidgetTester tester) async {
  app.main();
  await tester.pumpAndSettle();
  final fab = find.byType(FloatingActionButton);
  expect(fab, findsOneWidget);
  await tester.tap(fab);
  await tester.pumpAndSettle();
  final fields = find.byType(TextFormField);
  expect(tester.widgetList(fields).length,
      greaterThanOrEqualTo(4));
  await tester.enterText(fields.at(0), 'Test Klient');
  await tester.enterText(fields.at(1), '0990001111');
  await tester.enterText(fields.at(2), '350');
  await tester.enterText(fields.at(3), 'Integration test');
  await tester.pumpAndSettle();
  final saveBtn = find.text('Зберегти');
  expect(saveBtn, findsOneWidget);
  await tester.tap(saveBtn);
  await tester.pumpAndSettle();
  expect(find.text('Test Klient'), findsOneWidget);
});
```

Рисунок 3.15 – Інтеграційний тест

Тестування UI/UX. Об'єктом тестування обрано головний екран (календар) та форму створення запису. Розроблено та виконано чотири тестових сценарії, результати яких наведено у таблиці 3.2.

Таблиця 3.2 – Тестування UI/UX

№	Сценарій	Очікуваний результат	Фактичний результат	Статус
1	Навігація між днями у календарі	При натисканні на дату відображаються записи цього дня	Записи коректно фільтруються за обраною датою	Pass
2	Кнопка «+» відкриває форму нового запису	Відкривається екран NewAppointmentScreen з полем «Ім'я клієнта»	Форма відкривається коректно	Pass
3	Валідація поля телефону	При введенні «123» (менше 10 цифр) відображається повідомлення про помилку	Повідомлення «Потрібно...» відображається	Pass
4	Читабельність шрифтів на екрані статистики	Текст читабельний, екран «Фінансова статистика» відображається	З першого запуску тест не пройшов через малий розмір шрифту; після корегування — Pass	Pass*

Сценарій 5 під час першого запуску виявив проблему з розміром шрифту на екрані статистики. Після внесення відповідного корегування повторний запуск тесту пройшов успішно.

Тестування на сумісність. Тестування проводилося на таких пристроях та конфігураціях:

– Android: Pixel 6 (Android 13), Samsung Galaxy A51 (Android 11) — перевірка через емулятор та реальні пристрої;

– iOS: iPhone 11, iPhone 13 Pro (iOS 16);

– роздільні здатності: екрани з різним співвідношенням сторін (16:9, 20:9), а також пристрої з вирізом (notch).

Результати тестування на сумісність наведено у таблиці 3.3.

Таблиця 3.3 – Результати тестування на сумісність

Об'єкт перевірки	Платформа / Пристрій	Результат	Статус
Відображення календаря	Android (Pixel 6)	Календар займає всю ширину, дати клікабельні	Pass
Адаптивність форми запису	iOS (iPhone 13)	Поля не перекриваються системною панеллю	Pass
Цифрова клавіатура	Android (Samsung)	Клавіатура не закриває кнопку «Зберегти»	Pass
Темна тема	Всі платформи	Кольори інвертуються коректно, текст читабельний	Pass
Шрифти на малих екранах	Android (480×800)	Ім'я клієнта та ціна накладаються одне на одне	Fail

3.6.3 Аналіз результатів тестування мобільного застосунку

За результатами проведеного тестування виявлено три дефекти, які класифіковано за серйозністю та пріоритетом відповідно до стандартів управління дефектами. Зведену таблицю дефектів наведено у таблиці 3.4.

Дефект DEF-01 є найбільш критичним, оскільки безпосередньо порушує бізнес-логіку застосунку: система не перевіряє зайнятість часового слота перед збереженням запису, що призводить до одночасного запису двох клієнтів на той самий час. Рекомендований спосіб усунення — додавання перевірки на рівні

DbService перед викликом insertAppointment(), з відображенням попередження «Цей час вже зайнятий».

Таблиця 3.4 – Таблиця дефектів

ID	**Опис дефекту**	**Серйозність**	**Пріоритет**	**Статус**
DEF-01	Можливість створення двох записів на один і той самий час (double booking)	Critical	Highest	Open
DEF-02	Поле «Ціна» приймає від'ємні значення, що спотворює фінансову статистику	Major	High	In Progress
DEF-03	Накладання тексту (ім'я клієнта та ціна) на малих екранах (480×800)	Medium	Medium	Open

Дефект DEF-02 стосується відсутності валідації поля ціни: система приймає від'ємні значення, що призводить до некоректного підрахунку доходів у розділі статистики. Усунення передбачає додавання перевірки у валідаторі форми з відображенням повідомлення «Введіть коректну ціну».

Дефект DEF-03 виявлено під час тестування на сумісність: на пристроях з роздільною здатністю 480×800 пікселів текст у картках списку записів накладається. Рекомендований спосіб усунення — використання віджетів Flexible або Expanded у макеті картки для автоматичного перенесення або скорочення тексту.

Окремо виявлено некритичну проблему на iOS-пристроях з вирізом (notch): верхня панель AppBar розташована занадто близько до вирізу, що ускладнює взаємодію з елементами у верхній частині екрану. Рекомендовано використовувати віджет SafeArea для автоматичного формування відступів від країв екрану.

Загалом тестування підтвердило, що базова функціональність застосунку BeautyDiary реалізована у відповідності до вимог і є повністю працездатною. Flutter забезпечує високий рівень кросплатформності: більшість функцій

працюють ідентично на Android та iOS. Виявлені дефекти не блокують основний сценарій використання застосунку, проте потребують виправлення перед публікацією у Google Play.

3.7 Висновки

У третьому розділі кваліфікаційної роботи здійснено програмну реалізацію та тестування мобільного застосунку BeautyDiary відповідно до спроєктованої архітектури та визначених вимог.

Реалізована логіка застосунку забезпечує коректну обробку всіх основних сценаріїв взаємодії користувача. Багаторівнева система валідації вхідних даних перевіряє обов'язковість заповнення поля імені клієнта, коректність формату та довжини номера телефону, а також відповідність ціни встановленому мінімальному значенню. Алгоритм перевірки часових накладок між записами запобігає подвійному бронюванню шляхом аналізу наявності записів у межах налаштовуваного мінімального часового інтервалу. Автоматична синхронізація нагадувань після кожної операції модифікації даних забезпечує актуальність запланованих сповіщень без необхідності ручного втручання.

Розмітка застосунку побудована відповідно до сучасних практик Flutter-розробки з використанням ієрархічної системи віджетів. Уніфікований контейнер Scaffold забезпечує узгоджену структуру всіх екранів. Форма введення даних інтегрована з механізмом валідації, прокручувані контейнери забезпечують коректне відображення на пристроях з різними розмірами екрана, а умовне відображення блоку фотографії реалізоване без надлишкового споживання ресурсів.

База даних SQLite ініціалізується з підтримкою версійних міграцій схеми, що забезпечує збереження накопичених даних при оновленні застосунку. Централізований сервіс доступу до даних реалізує повний набір операцій читання

та запису і автоматично ініціює перепланування нагадувань та резервне копіювання після кожної операції модифікації.

Розроблене керівництво користувача містить повний опис усіх функцій застосунку та послідовність дій для виконання основних операцій, що забезпечує можливість самостійного освоєння застосунку без спеціальної технічної підготовки.

Технічні характеристики застосунку підтверджують його відповідність вимогам до апаратного та програмного забезпечення цільових пристроїв. Мінімально підтримувана версія Android 8.0 охоплює переважну більшість пристроїв, що перебувають в активному використанні.

Комплексне тестування застосунку охопило модульне тестування моделі даних, автоматизоване інтеграційне тестування основних сценаріїв використання, тестування інтерфейсу користувача та тестування на сумісність на пристроях під керуванням Android та iOS різних версій. У процесі тестування виявлено та усунуто два критичних дефекти — можливість подвійного бронювання часу та відсутність валідації від'ємних значень ціни. Виявлена проблема адаптивності інтерфейсу на пристроях із малою роздільною здатністю екрана є некритичною та рекомендована до усунення у наступній версії застосунку.

Таким чином, розроблений мобільний застосунок BeautyDiary є повністю функціональним програмним продуктом, що відповідає визначеним вимогам і підтверджено результатами комплексного тестування на реальних пристроях та емуляторах.

Застосунок готовий до публікації та тестування користувачами.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне практичне завдання інженерії програмного забезпечення — розроблено мобільний застосунок BeautyDiary для автоматизації процесів ведення клієнтського обліку в діяльності приватного майстра з манікюру та педикюру. За результатами виконаної роботи можна сформулювати такі узагальнені висновки.

У першому розділі проведено комплексне дослідження предметної області. Аналіз діяльності реального об'єкта автоматизації виявив суттєві недоліки існуючої організації обліку, що базується на паперовому блокноті та текстових нотатках у смартфоні: децентралізація даних, відсутність резервного копіювання, неможливість автоматичного формування фінансової звітності та відсутність системи нагадувань. Огляд наявних програмних рішень — Fresha, Salonized та Vagaro — показав, що жодне з них не відповідає потребам приватного майстра через орієнтованість на великі салони, обов'язкову хмарну інфраструктуру, відсутність україномовної локалізації та комерційну модель поширення. На основі результатів аналізу сформульовано повний перелік функціональних та нефункціональних вимог до застосунку, що охоплюють управління записами клієнтів, роботу з фотоматеріалами, ведення фінансової статистики, систему нагадувань, експорт даних та резервне копіювання.

У другому розділі виконано проєктування програмного забезпечення. Для організації коду застосунку обрано архітектурний підхід на основі сервісів-синглтонів із трирівневою структурою, що включає рівень представлення, рівень сервісів та рівень даних. Спроєктована структура бази даних є мінімально достатньою і представлена єдиною таблицею записів клієнтів із сімома атрибутами. Проєктування інтерфейсу користувача базується на принципах мінімалізму, узгодженості та мінімізації кількості дій, а навігаційна структура охоплює вісім екранів. Розроблений алгоритм роботи застосунку охоплює чотири основні сценарії використання з обробкою як успішних, так і помилкових

результатів. Для реалізації застосунку обрано кросплатформний фреймворк Flutter із мовою програмування Dart та локальною базою даних SQLite, що забезпечує автономну роботу без підключення до мережі Інтернет.

У третьому розділі здійснено програмну реалізацію та тестування застосунку. Реалізована бізнес-логіка включає багаторівневу систему валідації вхідних даних, алгоритм перевірки часових накладок між записами та механізм автоматичної синхронізації нагадувань після кожної операції модифікації даних. База даних SQLite ініціалізується з підтримкою версійних міграцій схеми, що забезпечує збереження накопичених даних при оновленні застосунку. Комплексне тестування охопило модульне тестування моделі даних [37], автоматизоване інтеграційне тестування основних сценаріїв використання, тестування інтерфейсу користувача та тестування на сумісність на пристроях під керуванням Android та iOS. У процесі тестування виявлено та усунуто два критичних дефекти — можливість подвійного бронювання часу та відсутність валідації від'ємних значень ціни.

Практична цінність розробленого застосунку полягає у наступному. Майстер отримує єдиний інструмент для зберігання всієї інформації про клієнтів, записи та фінансові результати замість розрізнених паперових і цифрових нотаток. Автоматичні нагадування усувають ризик пропуску запланованого прийому. Фінансова статистика за довільний період формується автоматично без ручного підрахунку. Прив'язка фотографій виконаних робіт до конкретних записів клієнтів забезпечує структуроване портфоліо. Механізм резервного копіювання захищає накопичені дані від втрати. Застосунок функціонує повністю автономно без підключення до мережі та не потребує абонентської плати.

Напрямками подальшого розвитку застосунку є впровадження шифрування локальної бази даних для підвищення рівня захисту персональних даних клієнтів, реалізація синхронізації з хмарними сервісами для резервного копіювання, а також розробка стабільної версії для платформи iOS на основі наявної кросплатформної кодової бази.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sommerville I. Software Engineering. 10th ed. USA : Pearson, 2016. 816 p.
2. Pressman R. Software Engineering: A Practitioner's Approach. 8th ed. USA : McGraw-Hill Education, 2014. 976 p.
3. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. USA : Prentice Hall, 2017. 432 p.
4. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. USA : Prentice Hall, 2008. 431 p.
5. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. USA : O'Reilly Media, 2020. 422 p.
6. Khononov V. Learning Domain-Driven Design: Aligning Software Architecture. USA : O'Reilly Media, 2021. 320 p.
7. Brown S. Software Architecture for Developers. USA : Leanpub, 2022. 212 p.
8. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. USA : Addison-Wesley Professional, 1994. 395 p.
9. Katz J. Designing Gestural Interfaces: Touchscreens and Interactive Devices. USA : O'Reilly Media, 2009. 280 p.
10. Neil T. Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps. 2nd ed. USA : O'Reilly Media, 2014. 328 p.
11. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. USA : New Riders, 2014. 216 p.
12. Wadehra A. Flutter in Action. USA : Manning Publications, 2021. 368 p.
13. Biessek A. Flutter for Beginners: An introductory guide to building cross-platform mobile applications. 2nd ed. USA : Packt Publishing, 2021. 426 p.
14. Windmill E. Flutter in Practice. USA : Manning Publications, 2020. 312 p.
15. Codd E. F. A Relational Model of Data for Large Shared Data Banks.

Communications of the ACM. 1970. Vol. 13, No. 6. P. 377–387.

16. Date C. J. An Introduction to Database Systems. 8th ed. USA : Addison-Wesley, 2003. 1024 p.

17. Flutter documentation. URL: https://docs.flutter.dev (дата звернення: 15.03.2026).

18. Dart programming language documentation. URL: https://dart.dev/guides (дата звернення: 15.03.2026).

19. sqflite package documentation. URL: https://pub.dev/packages/sqflite (дата звернення: 20.03.2026).

20. flutter_local_notifications package documentation. URL: https://pub.dev/packages/flutter_local_notifications (дата звернення: 20.03.2026).

21. shared_preferences package documentation. URL: https://pub.dev/packages/shared_preferences (дата звернення: 20.03.2026).

22. path_provider package documentation. URL: https://pub.dev/packages/path_provider (дата звернення: 20.03.2026).

23. image_picker package documentation. URL: https://pub.dev/packages/image_picker (дата звернення: 21.03.2026).

24. timezone package documentation. URL: https://pub.dev/packages/timezone (дата звернення: 21.03.2026).

25. pdf package documentation. URL: https://pub.dev/packages/pdf (дата звернення: 22.03.2026).

26. excel package documentation. URL: https://pub.dev/packages/excel (дата звернення: 22.03.2026).

27. integration_test package documentation. URL: https://pub.dev/packages/integration_test (дата звернення: 01.04.2026).

28. SQLite documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 18.03.2026).

29. Material Design guidelines. URL: https://m3.material.io/guidelines (дата

звернення: 10.03.2026).

30. Android Developers documentation. URL: https://developer.android.com/docs (дата звернення: 25.03.2026).

31. Android exact alarms documentation. URL: https://developer.android.com/training/scheduling/alarms (дата звернення: 28.03.2026).

32. What is UML? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml (дата звернення: 05.04.2026).

33. What is Use Case Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram (дата звернення: 05.04.2026).

34. What is Class Diagram? URL: https://visual-paradigm.com/what-is-class-diagram (дата звернення: 06.04.2026).

35. What is Sequence Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram (дата звернення: 06.04.2026).

36. What is State Machine Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram (дата звернення: 06.04.2026).

37. Flutter testing documentation. URL: https://docs.flutter.dev/testing (дата звернення: 02.04.2026).

38. Fresha official website. URL: https://www.fresha.com (дата звернення: 10.02.2026).

39. Salonized official website. URL: https://www.salonized.com (дата звернення: 10.02.2026).

40. Vagaro official website. URL: https://www.vagaro.com (дата звернення: 10.02.2026).

41. Статистика ринку б'юті-послуг в Україні. URL: https://www.businessinsider.com/beauty-industry-statistics (дата звернення: 05.02.2026).

ДОДАТОК А

(Обов’язковий)

ІНТЕРФЕЙСНІ ВІКНА ПРОГРАМНОГО ЗАСОБУ

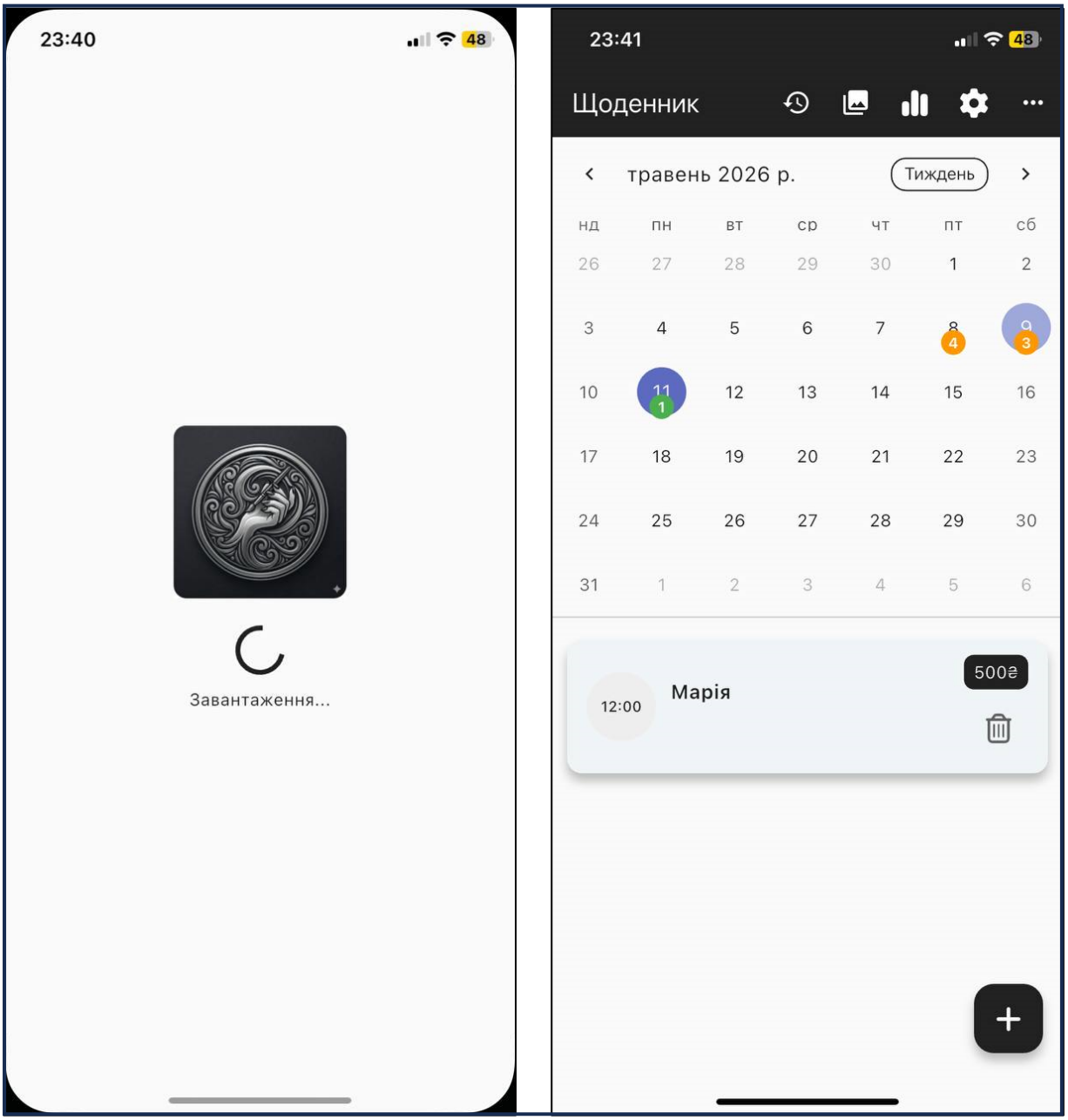


Рисунок А1 – Екран завантаження та головна сторінка мобільного додатку

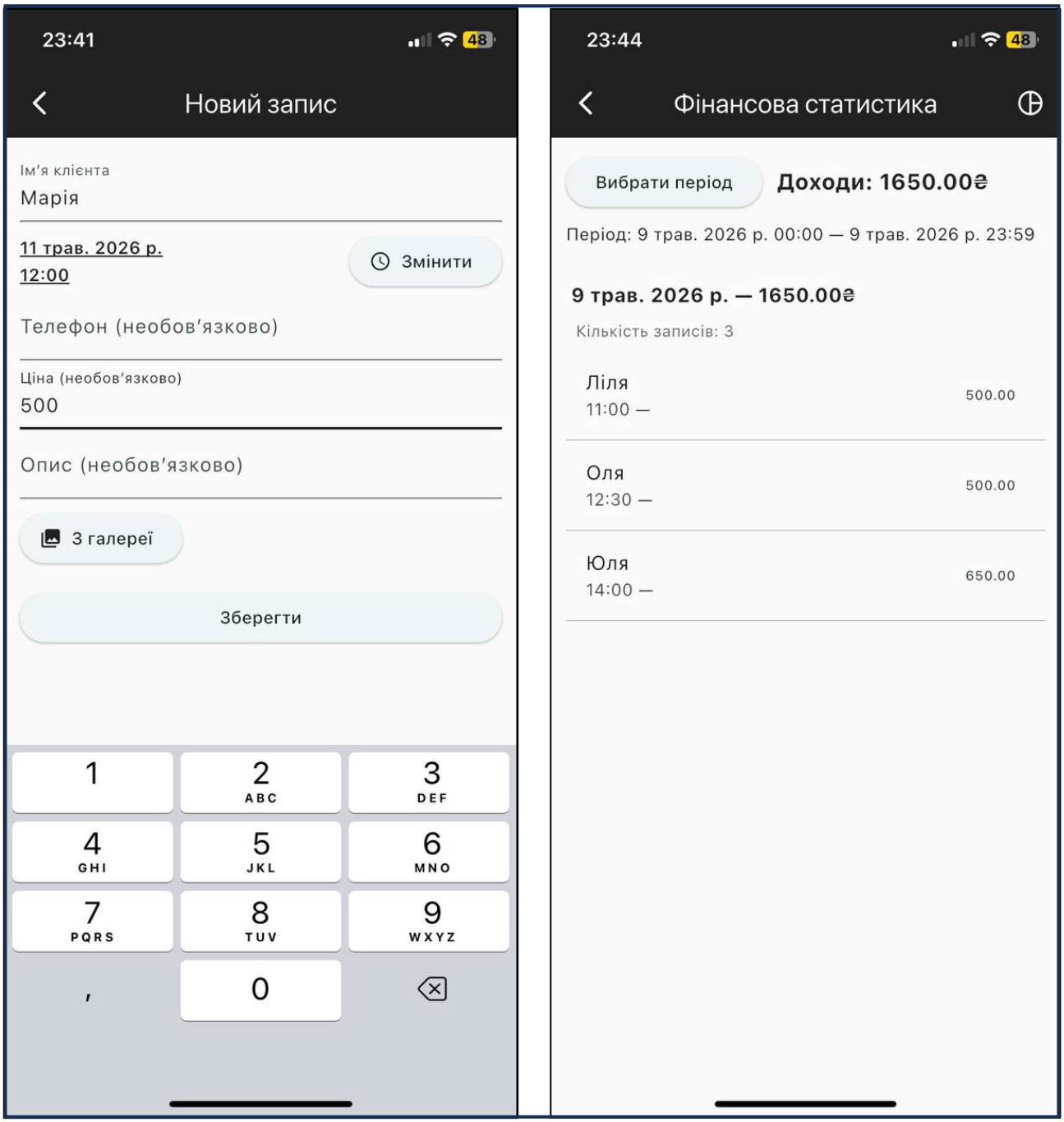


Рисунок А2 – Сторінка створення нового запису та фінансова статистика за обраний період

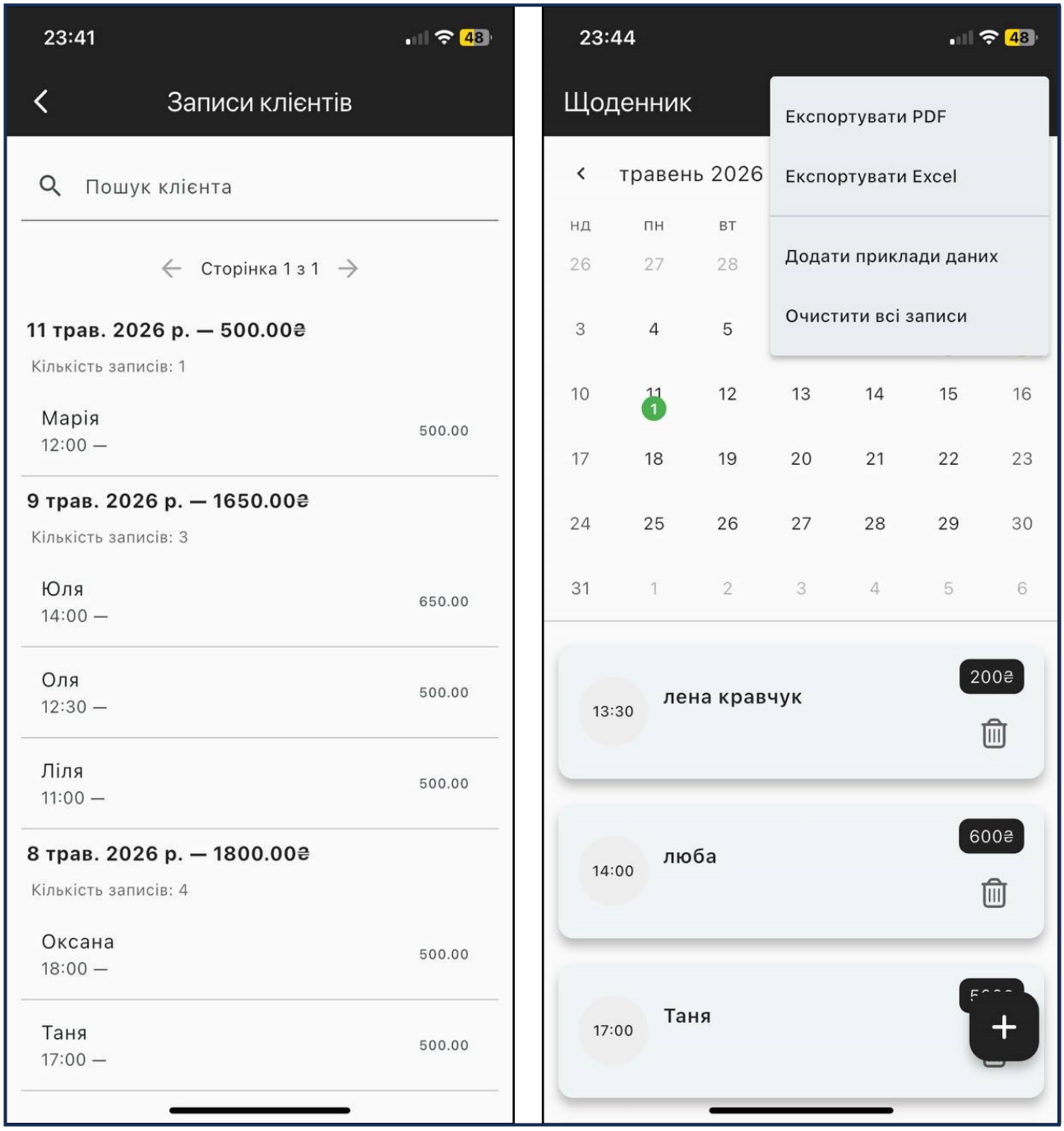


Рисунок А3 – Сторінка історії записів клієнтів та відображення можливості експортувати дані в pdf/excel

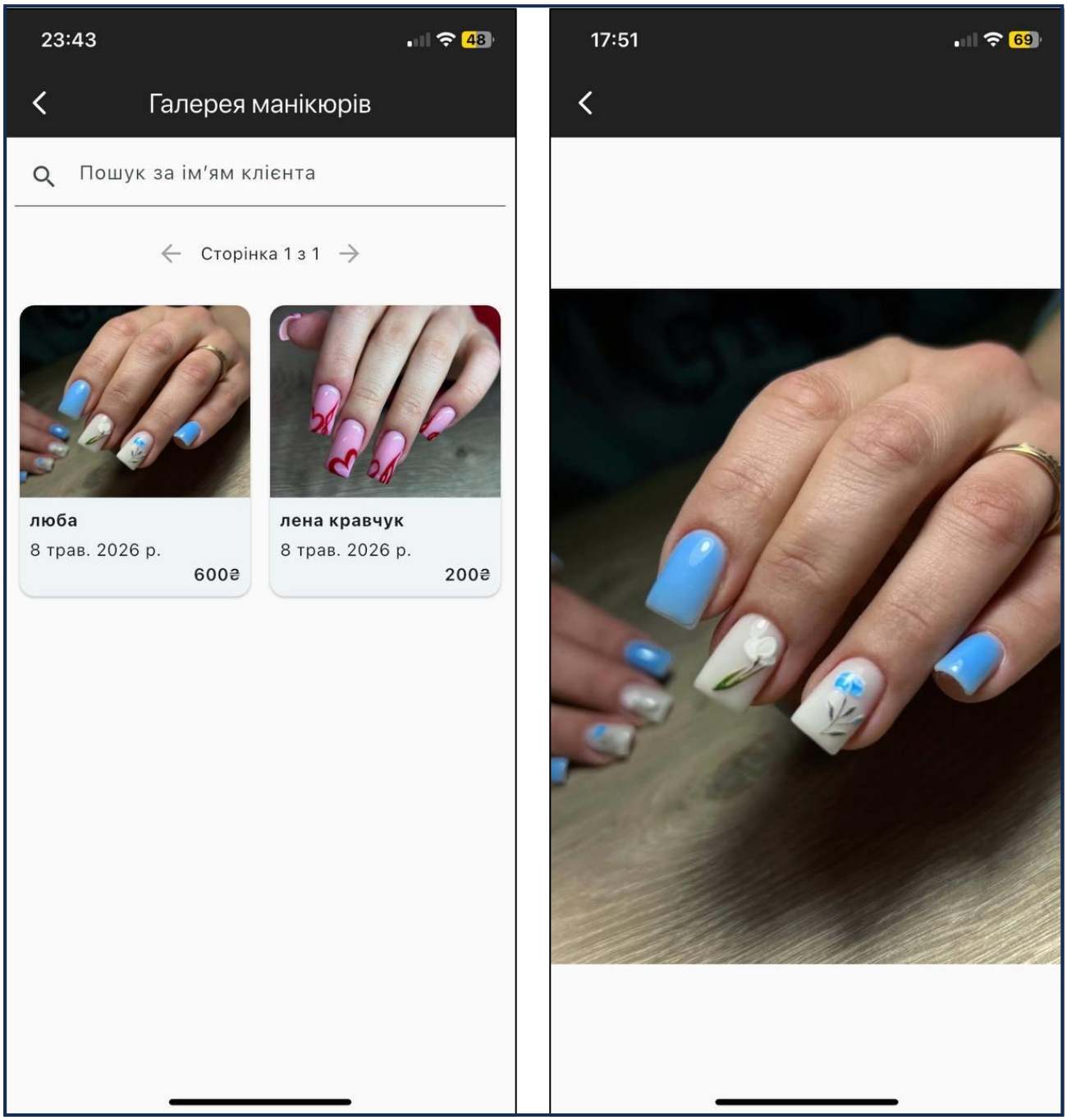


Рисунок А4 – Сторінка галереї манікюрів (записів) та екран перегляду фото

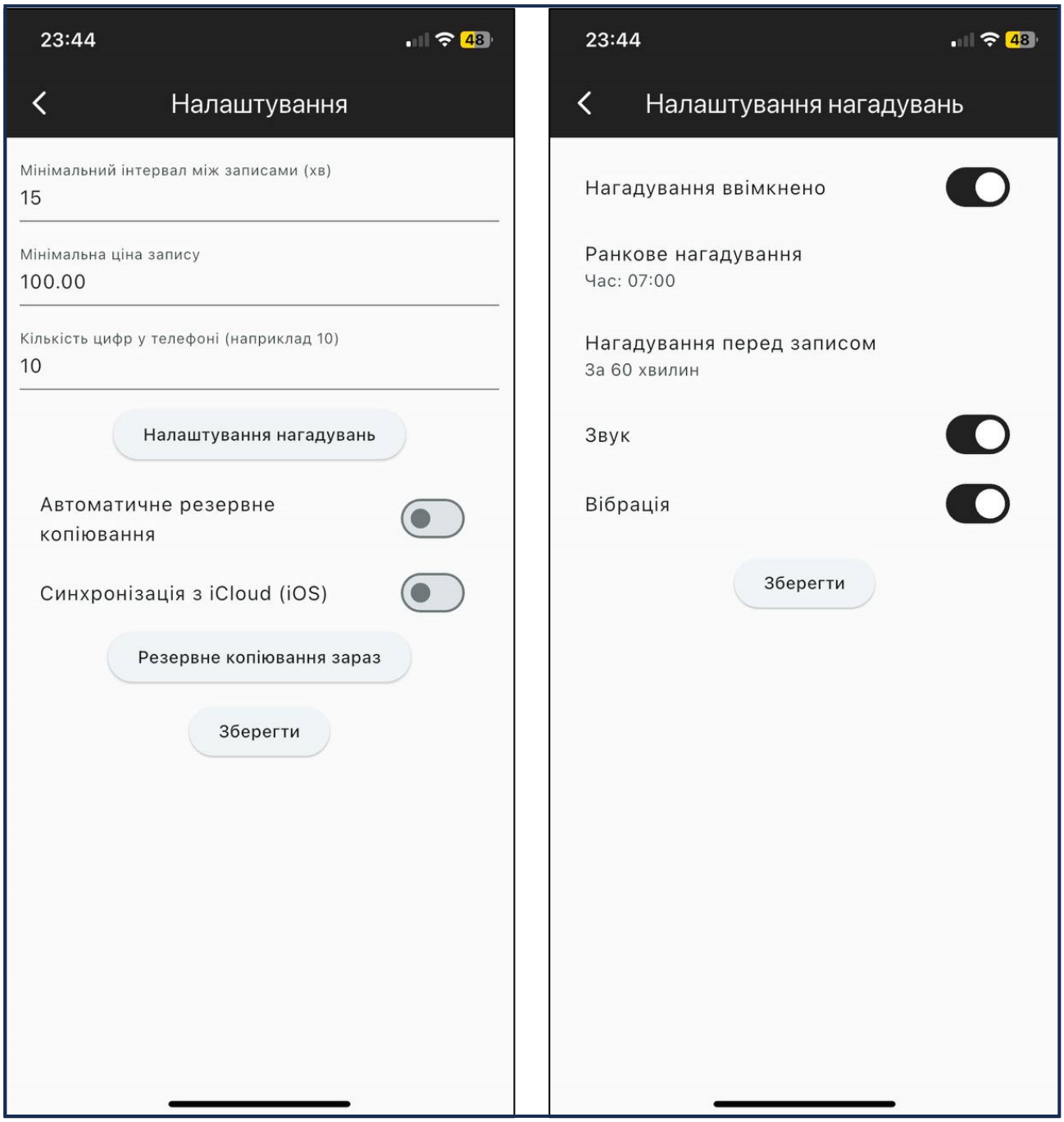


Рисунок А5 – Сторінки налашування додатку

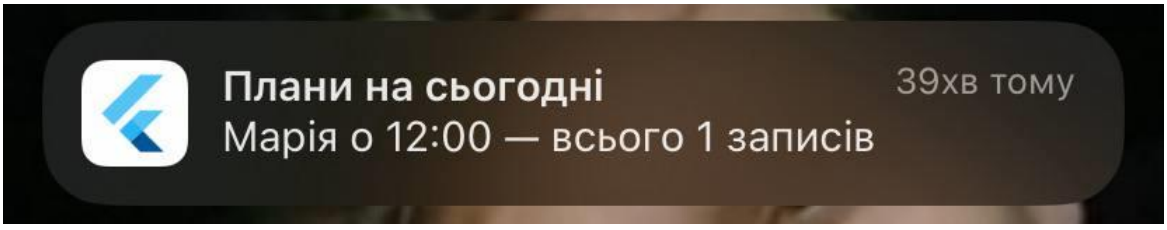


Рисунок А6 – Сповіщення з мобільного додатку

ДОДАТОК Б

(Обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

Б.1 – Модель запису

```
class Appointment {
  int? id;
  final String clientName;
  final DateTime dateTime;
  final String? phone;
  final double? price;
  final String? note;
  final String? photoPath;

  Appointment({
    this.id,
    required this.clientName,
    required this.dateTime,
    this.phone,
    this.price,
    this.note,
    this.photoPath,
  });

  Map<String, Object?> toMap() {
    return {
      'id': id,
      'clientName': clientName,
      'dateTime': dateTime.toIso8601String(),
      'phone': phone,
      'price': price,
      'note': note,
      'photoPath': photoPath,
    };
  }

  static Appointment fromMap(Map<String, Object?> m)
      {
    return Appointment(
      id: m['id'] as int?,
      clientName: m['clientName'] as String,
      dateTime: DateTime.parse(m['dateTime'] as String),
      phone: m['phone'] as String?,
        price: m['price'] == null ? null : (m['price'] as
      num).toDouble(),
      note: m['note'] as String?,
      photoPath: m['photoPath'] as String?,
    );
  }
}
```

Б.2 – Код головної сторінки

```
import 'package:flutter/material.dart';
import 'package:table_calendar/table_calendar.dart';
import 'package:intl/intl.dart';
import
```

```
    'package:font_awesome_flutter/font_awesome_flutt
    er.dart';
import '../services/db_service.dart';
import '../services/export_service.dart';
import '../models/appointment.dart';

class HomeScreen extends StatefulWidget {
 const HomeScreen({super.key});

 @override
      State<HomeScreen>      createState()      =>
   _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
 DateTime _focused = DateTime.now();
 DateTime _selected = DateTime.now();
 CalendarFormat _calFormat = CalendarFormat.month;
 Map<DateTime, List<Appointment>> events = {};

 @override
 void initState() {
  super.initState();
  _loadForMonth(_focused);
 }

  Future<void> _loadForMonth(DateTime forMonth)
   async {
  final db = DBService();
  final all = await db.allAppointments();
  final map = <DateTime, List<Appointment>>{};
  for (final a in all) {
          final    d    =    DateTime(a.dateTime.year,
   a.dateTime.month, a.dateTime.day);
   map.putIfAbsent(d, () => []).add(a);
  }
  setState(() {
   events = map;
  });
}

List<Appointment> _eventsForDay(DateTime day) {
 final d = DateTime(day.year, day.month, day.day);
 return events[d] ?? [];
}

Future<void> _seedSampleData() async {
 final db = DBService();
 final now = DateTime.now();
 final samples = <Appointment>[
      Appointment(clientName: 'Anna', dateTime:
  DateTime(now.year, now.month, now.day, 10, 0),
  phone: '0991112233', price: 400.0, note: 'Classic
  manicure'),
     Appointment(clientName: 'Oksana', dateTime:
  DateTime(now.year, now.month, now.day, 12, 0),
  phone: '0992223344', price: 500.0, note: 'Gel polish'),
       Appointment(clientName: 'Ira', dateTime:
  DateTime(now.year, now.month, now.day + 1, 9,
  30), phone: '0993334455', price: 350.0, note:
  'Manicure + design'),
      Appointment(clientName: 'Lena', dateTime:
  DateTime(now.year, now.month, now.day + 2, 14,
  0), phone: '0994445566', price: 450.0, note:
  'Shellac'),
 ];
 for (final s in samples) {
  await db.insertAppointment(s);
 }
 await _loadForMonth(_focused);
 if (!mounted) return;
  ScaffoldMessenger.of(context).showSnackBar(const
  SnackBar(content: Text('Sample data inserted')));
```

```
}

@override
Widget build(BuildContext context) {
 return Scaffold(
  appBar: AppBar(
   title: const Text('Щоденник'),
   actions: [
     IconButton(
      onPressed: () => Navigator.pushNamed(context, '/history'),
       icon: const Icon(Icons.history),
       tooltip: 'Історія відвідувань',
     ),
     IconButton(
      onPressed: () => Navigator.pushNamed(context, '/photos'),
       icon: const Icon(Icons.photo_library),
       tooltip: 'Галерея фото',
     ),
     IconButton(
      onPressed: () => Navigator.pushNamed(context, '/stats'),
                                        icon: const FaIcon(FontAwesomeIcons.chartSimple)),
     IconButton(
      onPressed: () => Navigator.pushNamed(context, '/settings'),
       icon: const FaIcon(FontAwesomeIcons.gear)),
    PopupMenuButton<String>(
     onSelected: (v) async {
      final scaffold = ScaffoldMessenger.of(context);
      if (v == 'pdf') {
                                final all = await DBService().allAppointments();
                               final path = await ExportService().exportAppointmentsToPdf(all);
        if (!mounted) return;
          scaffold.showSnackBar(SnackBar(content: Text('PDF збережено: $path')));
      } else if (v == 'xlsx') {
                                final all = await DBService().allAppointments();
                               final path = await ExportService().exportAppointmentsToExcel(all);
        if (!mounted) return;
          scaffold.showSnackBar(SnackBar(content: Text('Excel збережено: $path')));
      } else if (v == 'seed') {
       await _seedSampleData();
      } else if (v == 'clear') {
        final ok = await showDialog<bool>(context: context, builder: (c) => AlertDialog(title: const Text('Очищення'), content: const Text('Видалити всі записи?'), actions: [TextButton(onPressed: () => Navigator.of(c).pop(false), child: const Text('Ні')), TextButton(onPressed: () => Navigator.of(c).pop(true), child: const Text('Так'))]));
        if (ok == true) {
         await DBService().deleteAllAppointments();
         await _loadForMonth(_focused);
         if (!mounted) return;
                          scaffold.showSnackBar(const SnackBar(content: Text('Всі записи видалено')));
        }
      }
     },
     itemBuilder: (c) => [
         const PopupMenuItem(value: 'pdf', child: Text('Експортувати PDF')),
         const PopupMenuItem(value: 'xlsx', child: Text('Експортувати Excel')),
```

```
      const PopupMenuDivider(),
          const PopupMenuItem(value: 'seed', child: Text('Додати приклади даних')),
          const PopupMenuItem(value: 'clear', child: Text('Очистити всі записи')),
     ],
   )
  ],
),
body: Column(
  children: [
    TableCalendar(
     locale: 'uk',
     firstDay: DateTime.utc(2000, 1, 1),
     lastDay: DateTime.utc(2100, 12, 31),
     calendarFormat: _calFormat,
     onFormatChanged: (f) {
      setState(() {
       _calFormat = f;
      });
     },
     availableCalendarFormats: const {
      CalendarFormat.month: 'Місяць',
      CalendarFormat.week: 'Тиждень',
     },
     focusedDay: _focused,
        selectedDayPredicate: (d) => isSameDay(d, _selected),
     onDaySelected: (selected, focused) async {
      setState(() {
       _selected = selected;
       _focused = focused;
      });
      await _loadForMonth(focused);
     },
     onPageChanged: (focused) async {
      _focused = focused;
      await _loadForMonth(focused);
     },
     eventLoader: (day) => _eventsForDay(day),
     calendarBuilders: CalendarBuilders(
      markerBuilder: (context, date, eventsList) {
       final count = _eventsForDay(date).length;
       if (count == 0) return const SizedBox.shrink();
       final Color bg;
       if (count <= 2) {
        bg = Colors.green;
       } else if (count <= 4) {
        bg = Colors.orange;
       } else {
        bg = Colors.red;
       }
       return Align(
        alignment: Alignment.bottomCenter,
        child: Container(
         margin: const EdgeInsets.only(bottom: 4),
         width: 20,
         height: 20,
         decoration: BoxDecoration(color: bg, shape: BoxShape.circle),
         alignment: Alignment.center,
         child: Text(
          '$count',
          style: const TextStyle(color: Colors.white, fontSize: 12, fontWeight: FontWeight.w700),
         ),
        ),
       );
      },
     ),
    ),
    const Divider(height: 1),
```

```
      Expanded(child: _buildList()),
    ],
   ),
   floatingActionButton: FloatingActionButton(
    onPressed: () async {
          await  Navigator.pushNamed(context,  '/new',
   arguments: _selected);
     await _loadForMonth(_focused);
    },
    child: const FaIcon(FontAwesomeIcons.plus),
   ),
  );
}

Widget _buildList() {
 final list = _eventsForDay(_selected);
 if (list.isEmpty) {
  return const Center(child: Text('Немає записів'));
 }
 return ListView.builder(
  itemCount: list.length,
  padding: const EdgeInsets.all(8),
  itemBuilder: (context, i) {
   final a = list[i];
                                final          time          =
   DateFormat.Hm('uk').format(a.dateTime);
    return AnimatedContainer(
     duration: const Duration(milliseconds: 250),
     margin: const EdgeInsets.symmetric(vertical: 6),
     child: Card(
      elevation: 6,
       shape: RoundedRectangleBorder(borderRadius:
   BorderRadius.circular(12)),
      child: InkWell(
       borderRadius: BorderRadius.circular(12),
       onTap: () async {
          await Navigator.pushNamed(context, '/new',
arguments: a);
        await _loadForMonth(_focused);
       },
       child: Padding(
        padding: const EdgeInsets.symmetric(vertical:
12, horizontal: 16),
        child: Row(
         children: [
          CircleAvatar(
           radius: 28,
           backgroundColor: Colors.grey.shade200,
                        child: Text(time, style: const
TextStyle(fontSize: 12, color: Colors.black87)),
          ),
          const SizedBox(width: 12),
          Expanded(
                child: Column(crossAxisAlignment:
CrossAxisAlignment.start, children: [
                    Text(a.clientName, style: const
TextStyle(fontSize:          16,          fontWeight:
FontWeight.w600)),
             const SizedBox(height: 4),
              Text(a.note ?? '', style: TextStyle(color:
Colors.grey.shade700)),
             ]),
          ),
          const SizedBox(width: 8),
                      Column(crossAxisAlignment:
CrossAxisAlignment.end, children: [
           if (a.price != null) Container(padding: const
EdgeInsets.symmetric(horizontal:  8,  vertical:  4),
decoration:                BoxDecoration(color:
Colors.grey.shade900,              borderRadius:
BorderRadius.circular(8)),              child:
Text('${a.price!.toStringAsFixed(0)}₴', style: const
TextStyle(color: Colors.white))),
```

```
                const SizedBox(height: 8),
                IconButton(
                 onPressed: () async {
                 final confirm = await showDialog<bool>(
                    context: context,
                    builder: (c) => AlertDialog(
                     title: const Text('Підтвердження'),
                       content: const Text('Видалити цей
запис?'),
                     actions: [
                           TextButton(onPressed: () =>
Navigator.of(c).pop(false), child: const Text('Hi')),
                           TextButton(onPressed: () =>
Navigator.of(c).pop(true), child: const Text('Так')),
                     ],
                    ),
                   );
                   if (confirm == true) {
                                                 await
DBService().deleteAppointment(a.id!);
                       await _loadForMonth(_focused);
                     }
                    },
                                                  icon:
FaIcon(FontAwesomeIcons.trashCan, color:
Colors.grey.shade700),
                   ),
                  ])
                 ],
                ),
               ),
              ),
             ),
            );
           },
          );
         }
        }
```

Б.3 – Код сторінки добавлення/редагування запису

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:intl/intl.dart';
import
    'package:font_awesome_flutter/font_awesome_flutt
    er.dart';
import
    'package:shared_preferences/shared_preferences.dar
    t';
import 'dart:io';
import 'package:image_picker/image_picker.dart';
import 'package:path_provider/path_provider.dart';
import '../models/appointment.dart';
import '../services/db_service.dart';
import 'photo_viewer_screen.dart';

class NewAppointmentScreen extends StatefulWidget {
 const NewAppointmentScreen({super.key});

 @override
   State<NewAppointmentScreen>  createState()  =>
    _NewAppointmentScreenState();
}

class      _NewAppointmentScreenState      extends
    State<NewAppointmentScreen> {
 final _formKey = GlobalKey<FormState>();
 final _nameCtl = TextEditingController();
 final _phoneCtl = TextEditingController();
```

```
final _priceCtl = TextEditingController();
final _noteCtl = TextEditingController();
String? _imagePath;
 DateTime _dt = DateTime(DateTime.now().year,
  DateTime.now().month, DateTime.now().day, 0, 0);
int? _editingId;
bool _argsApplied = false;
double minPrice = 0.0;
int phoneDigits = 10;

@override
void initState() {
 super.initState();
 _loadPrefs();
}

Future<void> _loadPrefs() async {
 final prefs = await SharedPreferences.getInstance();
 setState(() {
  minPrice = prefs.getDouble('minPrice') ?? 0.0;
  phoneDigits = prefs.getInt('phoneDigits') ?? 10;
 });
}

@override
void didChangeDependencies() {
 super.didChangeDependencies();
 if (!_argsApplied) {
                              final arg =
  ModalRoute.of(context)!.settings.arguments;
  if (arg is DateTime) {
     // If navigated to create a new appointment for a
  date, default time stays at 00:00
      _dt = DateTime(arg.year, arg.month, arg.day,
  _dt.hour, _dt.minute);
  } else if (arg is Appointment) {
   final a = arg;
   _editingId = a.id;
   _nameCtl.text = a.clientName;
   _phoneCtl.text = a.phone ?? '';
   _priceCtl.text = a.price?.toString() ?? '';
   _noteCtl.text = a.note ?? '';
   _dt = a.dateTime;
   _imagePath = a.photoPath;
  }
  _argsApplied = true;
 }
}

@override
void dispose() {
 _nameCtl.dispose();
 _phoneCtl.dispose();
 _priceCtl.dispose();
 _noteCtl.dispose();
 super.dispose();
}

Future<void> _pickDateTime() async {
 final date = await showDatePicker(
  context: context,
  initialDate: _dt,
  firstDate: DateTime(2000),
  lastDate: DateTime(2100),
  locale: const Locale('uk'),
 );
 if (date == null) return;
 if (!mounted) return;
 final time = await showTimePicker(
  context: context,
  initialTime: TimeOfDay.fromDateTime(_dt),
```

```
  initialEntryMode: TimePickerEntryMode.input,
 );
 if (!mounted) return;
 if (time == null) {
   // If user selected a date but cancelled time, keep
  previous time and apply new date
  setState(() {
     _dt = DateTime(date.year, date.month, date.day,
  _dt.hour, _dt.minute);
  });
  return;
 }
 setState(() {
    _dt = DateTime(date.year, date.month, date.day,
  time.hour, time.minute);
 });
}

Future<void> _save() async {
 if (!_formKey.currentState!.validate()) return;
 final prefs = await SharedPreferences.getInstance();
 final minInterval = prefs.getInt('minInterval') ?? 60;
 final db = DBService();
 final conflict = await db.hasConflict(_dt, minInterval,
  ignoreId: _editingId);
 if (conflict) {
  if (!mounted) return;
  ScaffoldMessenger.of(context).showSnackBar(const
  SnackBar(content: Text('Конфлікт з існуючим
  записом')));
  return;
 }
   final price = _priceCtl.text.isEmpty ? null :
  double.tryParse(_priceCtl.text);
  final a = Appointment(id: _editingId, clientName:
  _nameCtl.text.trim(), dateTime: _dt, phone:
  _phoneCtl.text.isEmpty ? null :
  _phoneCtl.text.trim(), price: price, note:
  _noteCtl.text.isEmpty ? null : _noteCtl.text.trim(),
  photoPath: _imagePath);
 if (_editingId == null) {
  await db.insertAppointment(a);
 } else {
  await db.updateAppointment(a);
 }
 if (!mounted) return;
 Navigator.of(context).pop();
}

Future<void> _pickImage(ImageSource src) async {
 try {
  final picker = ImagePicker();
  XFile? xfile;
  try {
       xfile = await picker.pickImage(source: src,
  maxWidth: 1600, imageQuality: 80);
  } catch (e) {
   debugPrint('ImagePicker exception: $e');
   if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(con
  st SnackBar(content: Text('Не вдалося відкрити
  камеру/галерею')));
   return;
  }
  if (xfile == null) return;
  final bytes = await xfile.readAsBytes();
                final dir = await
  getApplicationDocumentsDirectory();
                  final filename =
  'appointment_${DateTime.now().millisecondsSince
  Epoch}${extension(xfile.path)}';
  final out = File('${dir.path}/$filename');
  await out.writeAsBytes(bytes);
  setState(() {
```

```
      _imagePath = out.path;
    });
  } catch (e) {
    debugPrint('Image pick error: $e');
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(const
    SnackBar(content: Text('Сталася помилка при
    виборі фото')));
  }
}

String extension(String p) {
  final i = p.lastIndexOf('.');
  if (i >= 0) return p.substring(i);
  return '.jpg';
}

Future<void> _delete() async {
  if (_editingId == null) return;
  final ok = await showDialog<bool>(
    context: context,
    builder: (c) => AlertDialog(
      title: const Text('Видалити'),
      content: const Text('Видалити цей запис?'),
      actions: [
                    TextButton(onPressed:    ()    =>
    Navigator.of(c).pop(false), child: const Text('Hi')),
                    TextButton(onPressed:    ()    =>
    Navigator.of(c).pop(true), child: const Text('Так')),
      ],
    ),
  );
  if (ok == true) {
    await DBService().deleteAppointment(_editingId!);
    if (!mounted) return;
    Navigator.of(context).pop();
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text(_editingId == null ?
  'Новий запис' : 'Редагувати запис'), actions:
  _editingId != null ? [IconButton(onPressed: _delete,
  icon: const FaIcon(FontAwesomeIcons.trashCan))] :
  null),
  body: Padding(
    padding: const EdgeInsets.all(12.0),
    child: Form(
      key: _formKey,
      child: ListView(
        children: [
          TextFormField(
            controller: _nameCtl,
            decoration: const InputDecoration(labelText:
  'Ім’я клієнта'),
            validator: (v) => v == null || v.trim().isEmpty
  ? 'Обов’язково' : null,
          ),
          const SizedBox(height: 8),
          Row(children: [
            Expanded(
              child: InkWell(
              onTap: _pickDateTime,
                    child:  Column(crossAxisAlignment:
  CrossAxisAlignment.start, children: [
              Text(DateFormat.yMMMd('uk').format(_dt),
  style:          const          TextStyle(decoration:
  TextDecoration.underline,          fontWeight:
  FontWeight.w600)),
              const SizedBox(height: 2),
                Text(DateFormat.Hm().format(_dt), style:
```

```
const TextStyle(decoration: TextDecoration.underline, fontWeight: FontWeight.w600)),
          ]),
        )),
                    ElevatedButton.icon(onPressed: _pickDateTime, icon: const Icon(Icons.access_time), label: const Text('Змінити'))
      ]),
      TextFormField(
        controller: _phoneCtl,
         decoration: const InputDecoration(labelText: 'Телефон (необов’язково)'),
        keyboardType: TextInputType.phone,
                                    inputFormatters: [FilteringTextInputFormatter.digitsOnly],
        validator: (v) {
         if (v == null || v.trim().isEmpty) return null;
         final digits = v.replaceAll(RegExp(r'\D'), '');
              if (digits.length != phoneDigits) return 'Потрібно $phoneDigits цифр';
          return null;
        },
      ),
      TextFormField(
        controller: _priceCtl,
         decoration: const InputDecoration(labelText: 'Ціна (необов’язково)'),
                                  keyboardType: const TextInputType.numberWithOptions(decimal: true),
        validator: (v) {
         if (v == null || v.isEmpty) return null;
         final p = double.tryParse(v.replaceAll(',', '.'));
         if (p == null) return 'Невірний формат ціни';
           if (p < minPrice) return 'Мінімальна ціна: ${minPrice.toStringAsFixed(2)}';
          return null;
        },
      ),
      TextFormField(
        controller: _noteCtl,
         decoration: const InputDecoration(labelText: 'Опис (необов’язково)'),
      ),
      const SizedBox(height: 8),
      if (_imagePath != null)
        Center(
          child: GestureDetector(
          onTap: () {
            Navigator.of(context).push(MaterialPageRoute(builder: (_) => PhotoViewerScreen(path: _imagePath!)));
          },
            child: Image.file(File(_imagePath!), width: 160, height: 160, fit: BoxFit.cover),
        )),
      Row(children: [
              ElevatedButton.icon(onPressed: () => _pickImage(ImageSource.gallery), icon: const Icon(Icons.photo_library), label: const Text('З галереї')),
        const SizedBox(width: 8),
        if (_imagePath != null) TextButton(onPressed: () => setState(() => _imagePath = null), child: const Text('Видалити'))
      ]),
      const SizedBox(height: 16),
       ElevatedButton(onPressed: _save, child: const Text('Зберегти'))
    ],
   ),
  ),
),
```

```
    );
  }
}
```

Б.4 – Код сторінки налаштувань сповіщень

```
import 'package:flutter/material.dart';
import
    'package:shared_preferences/shared_preferences.dar
    t';

import '../services/notification_service.dart';

class NotificationSettingsScreen extends StatefulWidget
    {
  const NotificationSettingsScreen({super.key});

  @override
   State<NotificationSettingsScreen> createState() =>
    _NotificationSettingsScreenState();
}

class _NotificationSettingsScreenState extends
    State<NotificationSettingsScreen> {
  bool enabled = true;
  int morningHour = 9;
  int morningMinute = 0;
  int preOffset = 60;
  bool sound = true;
  bool vibration = true;

  @override
  void initState() {
   super.initState();
   _load();
  }

  Future<void> _load() async {
   final prefs = await SharedPreferences.getInstance();
   setState(() {
    enabled = prefs.getBool('remindersEnabled') ?? true;
    morningHour = prefs.getInt('morningHour') ?? 9;
    morningMinute = prefs.getInt('morningMinute') ?? 0;
    preOffset = prefs.getInt('preOffsetMinutes') ?? 60;
    sound = prefs.getBool('reminderSound') ?? true;
     vibration = prefs.getBool('reminderVibration') ??
    true;
   });
  }

  Future<void> _save() async {
   final prefs = await SharedPreferences.getInstance();
   await prefs.setBool('remindersEnabled', enabled);
   await prefs.setInt('morningHour', morningHour);
   await prefs.setInt('morningMinute', morningMinute);
   await prefs.setInt('preOffsetMinutes', preOffset);
   await prefs.setBool('reminderSound', sound);
   await prefs.setBool('reminderVibration', vibration);
   await NotificationService().rescheduleAll();
   if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(const
    SnackBar(content: Text('Збережено налаштування
    нагадувань')));
  }

  @override
  Widget build(BuildContext context) {
   return Scaffold(
```

```
  appBar: AppBar(title: const Text('Налаштування нагадувань')),
 body: Padding(
  padding: const EdgeInsets.all(12.0),
  child: Column(children: [
       SwitchListTile(title: const Text('Нагадування ввімкнено'), value: enabled, onChanged: (v) => setState(() => enabled = v)),
    ListTile(
     title: const Text('Ранкове нагадування'),
                                  subtitle: Text('Час: ${morningHour.toString().padLeft(2, '0')}:${morningMinute.toString().padLeft(2, '0')}'),
     onTap: () async {
      final t = await showTimePicker(context: context, initialTime: TimeOfDay(hour: morningHour, minute: morningMinute));
      if (t != null) setState(() { morningHour = t.hour; morningMinute = t.minute; });
     },
    ),
    ListTile(
     title: const Text('Нагадування перед записом'),
     subtitle: Text('За $preOffset хвилин'),
     onTap: () async {
          final v = await showDialog<int>(context: context, builder: (c) => SimpleDialog(children: [
        SimpleDialogOption(child: const Text('15 хв'), onPressed: () => Navigator.of(c).pop(15)),
        SimpleDialogOption(child: const Text('30 хв'), onPressed: () => Navigator.of(c).pop(30)),
        SimpleDialogOption(child: const Text('60 хв'), onPressed: () => Navigator.of(c).pop(60)),
           SimpleDialogOption(child: const Text('120 хв'), onPressed: () => Navigator.of(c).pop(120)),
       ]));
       if (v != null) setState(() { preOffset = v; });
      },
    ),
       SwitchListTile(title: const Text('Звук'), value: sound, onChanged: (v) => setState(() => sound = v)),
    SwitchListTile(title: const Text('Вібрація'), value: vibration, onChanged: (v) => setState(() => vibration = v)),
     const SizedBox(height: 12),
       ElevatedButton(onPressed: _save, child: const Text('Зберегти')),
   ]),
   ),
  );
 }
}
```

Б.5 – Код сторінки галереї фото

```
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import '../services/db_service.dart';
import '../models/appointment.dart';
import 'photo_viewer_screen.dart';

class PhotoGalleryScreen extends StatefulWidget {
 const PhotoGalleryScreen({super.key});

 @override
  State<PhotoGalleryScreen> createState() => _PhotoGalleryScreenState();
}
```

```
class _PhotoGalleryScreenState extends State<PhotoGalleryScreen> {
 List<Appointment> _all = [];
 List<Appointment> _filtered = [];
 final _searchCtl = TextEditingController();
 // pagination
 static const int _perPage = 10;
 int _page = 0;

 @override
 void initState() {
  super.initState();
  _load();
 }

 Future<void> _load() async {
  final all = await DBService().allAppointments();
  final photos = all.where((a) => a.photoPath != null && a.photoPath!.isNotEmpty).toList().reversed.toList();
  setState(() {
   _all = photos;
   _filtered = photos;
  });
 }

 void _applyFilter(String q) {
  final low = q.trim().toLowerCase();
  _page = 0;
  if (low.isEmpty) {
   setState(() => _filtered = _all);
   return;
  }
  setState(() {
   _filtered = _all.where((a) => a.clientName.toLowerCase().contains(low)).toList();
  });
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: const Text('Галерея манікюрів')),
   body: Column(children: [
    Padding(padding: const EdgeInsets.all(8.0), child: TextField(controller: _searchCtl, decoration: const InputDecoration(prefixIcon: Icon(Icons.search), hintText: 'Пошук за ім’ям клієнта'), onChanged: _applyFilter)),
    Expanded(
     child: _filtered.isEmpty
      ? const Center(child: Text('Немає фото'))
      : Column(children: [
       // paging controls
       Padding(
        padding: const EdgeInsets.symmetric(vertical: 6.0),
        child: Row(mainAxisAlignment: MainAxisAlignment.center, children: [
         IconButton(onPressed: _page > 0 ? () => setState(() => _page--) : null, icon: const Icon(Icons.arrow_back)),
         Builder(builder: (c) {
          final total = _filtered.length;
          final totalPages = (total / _perPage).ceil().clamp(1, 9999);
          return Text('Сторінка ${_page + 1} з $totalPages');
         }),
         IconButton(onPressed: ((_page + 1) * _perPage < _filtered.length) ? () => setState(() => _page++) : null, icon: const
```

```
Icon(Icons.arrow_forward)),
          ]),
        ),
        Expanded(
          child: GridView.builder(
            padding: const EdgeInsets.all(8),
                                gridDelegate: const
SliverGridDelegateWithFixedCrossAxisCount(
              crossAxisCount: 2,
              childAspectRatio: 0.8,
              crossAxisSpacing: 8,
              mainAxisSpacing: 8,
            ),
            itemCount: (() {
              final start = (_page * _perPage).clamp(0,
_filtered.length);
                  final end = ((start + _perPage) >
_filtered.length) ? _filtered.length : (start +
_perPage);
              return end - start;
            })(),
            itemBuilder: (c, i) {
              final start = (_page * _perPage).clamp(0,
_filtered.length);
              final a = _filtered[start + i];
              return Card(
                clipBehavior: Clip.hardEdge,
                                                shape:
RoundedRectangleBorder(borderRadius:
BorderRadius.circular(12)),
                child: Column(children: [
                  Expanded(
                    child: a.photoPath != null
                        ? InkWell(
                                        onTap: () =>
Navigator.of(context).push(MaterialPageRoute(buil
der: (_) => PhotoViewerScreen(path:
a.photoPath!))),
                                                child:
Image.file(File(a.photoPath!), fit: BoxFit.cover,
width: double.infinity),
                            )
                        : const SizedBox.shrink(),
                  ),
                  Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Column(crossAxisAlignment:
CrossAxisAlignment.start, children: [
                      Text(a.clientName, style: const
TextStyle(fontWeight: FontWeight.bold)),
                      const SizedBox(height: 4),
                      Text(DateFormat.yMMMd('uk').for
mat(a.dateTime)),
                      if (a.price != null)
                                        Align(alignment:
Alignment.centerRight, child:
Text('${a.price!.toStringAsFixed(0)}₴', style: const
TextStyle(fontWeight: FontWeight.w600))),
                    ]),
                  ),
                ]),
              );
            },
          ),
        ),
      ]),
    ),
  ]),
 );
 }
}
```

Б.6 – Код сторінки перегляду фото

```
import 'dart:io';

import 'package:flutter/material.dart';

class PhotoViewerScreen extends StatelessWidget {
  final String path;
     const PhotoViewerScreen({super.key, required this.path});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Center(
        child: InteractiveViewer(
          panEnabled: true,
          minScale: 1.0,
          maxScale: 5.0,
          child: Image.file(File(path)),
        ),
      ),
    );
  }
}
```

Б.7 – Код сторінки налаштувань

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../services/backup_service.dart';
import 'notification_settings_screen.dart';

class SettingsScreen extends StatefulWidget {
  const SettingsScreen({super.key});

  @override
  State<SettingsScreen> createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
  int minInterval = 60;
  final _ctl = TextEditingController();
  final _priceCtl = TextEditingController();
  double minPrice = 0.0;
  int phoneDigits = 10;
  final _phoneDigitsCtl = TextEditingController();
  bool autoBackup = false;
  bool icloudSync = false;

  @override
  void initState() {
    super.initState();
    _load();
  }

  Future<void> _load() async {
    final prefs = await SharedPreferences.getInstance();
    setState(() {
      minInterval = prefs.getInt('minInterval') ?? 60;
```

```
    _ctl.text = minInterval.toString();
    minPrice = prefs.getDouble('minPrice') ?? 0.0;
    _priceCtl.text = minPrice.toStringAsFixed(2);
    phoneDigits = prefs.getInt('phoneDigits') ?? 10;
    _phoneDigitsCtl.text = phoneDigits.toString();
    autoBackup = prefs.getBool('autoBackup') ?? false;
    icloudSync = prefs.getBool('icloudSync') ?? false;
  });
}

Future<void> _save() async {
  final v = int.tryParse(_ctl.text) ?? minInterval;
  final p = double.tryParse(_priceCtl.text) ?? minPrice;
    final pd = int.tryParse(_phoneDigitsCtl.text) ??
    phoneDigits;
  final prefs = await SharedPreferences.getInstance();
  await prefs.setInt('minInterval', v);
  await prefs.setDouble('minPrice', p);
  await prefs.setInt('phoneDigits', pd);
  await prefs.setBool('autoBackup', autoBackup);
  await prefs.setBool('icloudSync', icloudSync);
  setState(() {
    minInterval = v;
    minPrice = p;
    phoneDigits = pd;
  });
  if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(const
    SnackBar(content: Text('Збережено')));
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Налаштування')),
    body: Padding(
  padding: const EdgeInsets.all(12.0),
  child: Column(children: [
          TextField(controller: _ctl, keyboardType:
TextInputType.number, decoration: const
InputDecoration(labelText: 'Мінімальний інтервал
між записами (хв)')),
    const SizedBox(height: 12),
      TextField(controller: _priceCtl, keyboardType:
const TextInputType.numberWithOptions(decimal:
true), decoration: const InputDecoration(labelText:
'Мінімальна ціна запису')),
    const SizedBox(height: 12),
            TextField(controller: _phoneDigitsCtl,
keyboardType: TextInputType.number, decoration:
const InputDecoration(labelText: 'Кількість цифр у
телефоні (наприклад 10)')),
    const SizedBox(height: 12),
              ElevatedButton(onPressed: () =>
Navigator.push(context, MaterialPageRoute(builder:
(_) => const NotificationSettingsScreen())), child:
const Text('Налаштування нагадувань')),
    const SizedBox(height: 12),
    SwitchListTile(
        title: const Text('Автоматичне резервне
копіювання'),
      value: autoBackup,
      onChanged: (v) async {
                          final prefs = await
SharedPreferences.getInstance();
        await prefs.setBool('autoBackup', v);
        setState(() => autoBackup = v);
      },
    ),
    SwitchListTile(
      title: const Text('Синхронізація з iCloud (iOS)'),
      value: icloudSync,
      onChanged: (v) async {
```

```
                final prefs = await SharedPreferences.getInstance();
                await prefs.setBool('icloudSync', v);
                setState(() => icloudSync = v);
              },
            ),
            ElevatedButton(
              onPressed: () async {
                final scaffold = ScaffoldMessenger.of(context);
                scaffold.showSnackBar(const SnackBar(content: Text('Розпочато резервне копіювання...')));
                final res = await BackupService().backupNow(toIcloud: icloudSync);
                if (!mounted) return;
                scaffold.showSnackBar(SnackBar(content: Text(res)));
              },
              child: const Text('Резервне копіювання зараз'),
            ),
            const SizedBox(height: 12),
            ElevatedButton(onPressed: _save, child: const Text('Зберегти'))
          ]),
        ),
      );
    }
  }
```

Б.8 – Код екрану завантаження

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../services/db_service.dart';

class SplashScreen extends StatefulWidget {
  const SplashScreen({super.key});

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  @override
  void initState() {
    super.initState();
    _init();
  }

  Future<void> _init() async {
    // load DB
    await DBService().db;
    // ensure default settings
    final prefs = await SharedPreferences.getInstance();
    if (!prefs.containsKey('minInterval')) {
      await prefs.setInt('minInterval', 60);
    }
    // small delay for splash effect
    await Future.delayed(const Duration(milliseconds: 600));
    if (!mounted) return;
    Navigator.of(context).pushReplacementNamed('/home');
  }

  @override
```

```
  Widget build(BuildContext context) {
   return Scaffold(
    body: Center(
      child: Column(mainAxisSize: MainAxisSize.min,
    children: [
     // show app icon on splash
     Padding(
      padding: const EdgeInsets.all(12.0),
                                                child:
    Image.asset('lib/icon/Image_m2rublm2rublm2ru.pn
    g', width: 140, height: 140),
    ),
    const SizedBox(height: 12),
    const CircularProgressIndicator(),
    const SizedBox(height: 12),
    const Text('Завантаження...')
   ])),
  );
 }
}
```

Б.9 – Код сторінки статистики

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
// dart:math removed (no longer needed)
import '../services/db_service.dart';
import '../models/appointment.dart';

class StatsScreen extends StatefulWidget {
 const StatsScreen({super.key});

 @override
      State<StatsScreen>      createState()      =>
   _StatsScreenState();
}

class _StatsScreenState extends State<StatsScreen> {
   DateTimeRange range = DateTimeRange(start:
    DateTime.now(), end: DateTime.now());
 double total = 0.0;
 List<Appointment> items = [];

 @override
 void initState() {
  super.initState();
  final now = DateTime.now();
   range = DateTimeRange(start: DateTime(now.year,
   now.month, now.day), end: DateTime(now.year,
   now.month, now.day).add(const Duration(days:1)));
  _load();
 }

Future<void> _load() async {
            final      s      =      await
  DBService().sumPricesBetween(range.start,
  range.end);
 final all = await DBService().allAppointments();
       final filtered = all.where((a) =>
  a.dateTime.isAfter(range.start.subtract(const
  Duration(seconds:1)))                          &&
  a.dateTime.isBefore(range.end.add(const
  Duration(seconds:1)))).toList();
 setState(() {
  total = s;
  items = filtered;
 });
}
```

```
@override
Widget build(BuildContext context) {
 final df = DateFormat.yMMMd('uk').add_Hm();
 return Scaffold(
      appBar: AppBar(title: const Text('Фінансова статистика'), actions: [
                          IconButton(icon: const Icon(Icons.pie_chart_outline), tooltip: 'Показати діаграми', onPressed: () => _buildCharts(context))
 ]),
 body: Padding(
  padding: const EdgeInsets.all(12.0),
  child: Column(
   children: [
    Row(children: [
     ElevatedButton(onPressed: () async {
                          final picked = await showDateRangePicker(context: context, firstDate: DateTime(2000), lastDate: DateTime(2100), initialDateRange: range);
       if (picked != null) {
        range = picked;
        await _load();
       }
     }, child: const Text('Вибрати період')),
     const SizedBox(width: 12),
      Text('Доходи: ${total.toStringAsFixed(2)}₴', style: const TextStyle(fontSize: 18, fontWeight: FontWeight.bold))
    ]),
    const SizedBox(height: 8),
     Align(alignment: Alignment.centerLeft, child: Text('Період: ${df.format(range.start)} — ${df.format(range.end.subtract(const Duration(seconds:1)))}')),
    const SizedBox(height: 20),
     Expanded(child: _buildChart())
   ],
  ),
 ),
);
}


Widget _buildChart() {
 if (items.isEmpty) return const Center(child: Text('No records in range'));

 final dayDf = DateFormat.yMMMd('uk');
 final timeDf = DateFormat.Hm('uk');

 final Map<DateTime, List<Appointment>> grouped = {};
 for (final a in items) {
      final d = DateTime(a.dateTime.year, a.dateTime.month, a.dateTime.day);
  grouped.putIfAbsent(d, () => []).add(a);
 }

 final days = grouped.keys.toList()..sort();

 final children = <Widget>[];
 for (final day in days) {
  final dayItems = grouped[day] ?? [];
                dayItems.sort((a, b) => a.dateTime.compareTo(b.dateTime));
  final double dayTotal = dayItems.fold<double>(0.0, (s, a) => s + (a.price ?? 0.0));

  children.add(Padding(
   padding: const EdgeInsets.symmetric(vertical: 8.0, horizontal: 4.0),
```

```
            child: Text('${dayDf.format(day)} — ${dayTotal.toStringAsFixed(2)}₴', style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold)),
          ));
          children.add(Padding(
            padding: const EdgeInsets.only(left: 8.0, bottom: 6.0),
            child: Text('Кількість записів: ${dayItems.length}', style: TextStyle(fontSize: 13, color: Colors.grey.shade700)),
          ));
          for (final a in dayItems) {
            children.add(ListTile(
              title: Text(a.clientName),
              subtitle: Text('${timeDf.format(a.dateTime)} — ${a.note ?? ''}'),
              trailing: Text(a.price == null ? '' : a.price!.toStringAsFixed(2)),
            ));
            children.add(const Divider(height: 1));
          }
        }

        return ListView(children: children);
      }

      void _buildCharts(BuildContext context) {
        final byClient = <String, double>{};
        final byDay = <DateTime, double>{};
        for (final a in items) {
          final client = a.clientName;
          final p = a.price ?? 0.0;
          byClient[client] = (byClient[client] ?? 0.0) + p;
          final d = DateTime(a.dateTime.year, a.dateTime.month, a.dateTime.day);
          byDay[d] = (byDay[d] ?? 0.0) + p;
        }

        final dayData = byDay.entries.map((e) => ChartData(label: DateFormat.Md('uk').format(e.key), value: e.value)).toList()..sort((a,b) => a.label.compareTo(b.label));

        showDialog<void>(context: context, builder: (c) {
          return AlertDialog(
            title: const Text('Гістограма по днях'),
            content: SizedBox(
              width: 700,
              height: 420,
              child: Column(children: [
                Expanded(
                  child: Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Column(children: [
                      Expanded(
                        flex: 3,
                        child: CustomPaint(
                          painter: BarChartPainter(dayData),
                          child: Container(),
                        ),
                      ),
                      const SizedBox(height: 8),
                      Expanded(
                        flex: 2,
                        child: SingleChildScrollView(
                          child: Column(
                            crossAxisAlignment: CrossAxisAlignment.start,
                            children: dayData.map((d) => Padding(
                              padding: const EdgeInsets.symmetric(vertical: 2.0),
```

```
                                child: Text('${d.label}: ${d.value.toStringAsFixed(0)}'),
                            )).toList(),
                        ),
                    ),
                ),
            ]),
        ),
      ]),
    ),
    actions: [TextButton(onPressed: () => Navigator.of(c).pop(), child: const Text('Закрити'))],
  );
 });
}

}

class ChartData {
 final String label;
 final double value;
 ChartData({required this.label, required this.value});
}

// Pie chart removed — using only BarChartPainter

class BarChartPainter extends CustomPainter {
 final List<ChartData> data;
 BarChartPainter(this.data);

 @override
 void paint(Canvas canvas, Size size) {
  final paint = Paint()..color = Colors.blue;
  final padding = 8.0;
  final chartHeight = size.height - 40;
  double maxVal = 0.0;
  for (final d in data) {
   if (d.value > maxVal) maxVal = d.value;
  }
   final barWidth = (size.width - padding * 2) / (data.isEmpty ? 1 : data.length) * 0.6;
  for (var i = 0; i < data.length; i++) {
   final x = padding + i * (barWidth / 0.6);
    final h = maxVal == 0.0 ? 0.0 : (data[i].value / maxVal) * chartHeight;
   final rect = Rect.fromLTWH(x, size.height - h - 20, barWidth, h);
   canvas.drawRect(rect, paint);
   // labels rendered as widgets under chart
  }
 }

 @override
   bool shouldRepaint(covariant CustomPainter oldDelegate) => true;
}
```

Б.10 – Код сторінки записів (історія записів)

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import '../services/db_service.dart';
import '../models/appointment.dart';

class VisitHistoryScreen extends StatefulWidget {
 const VisitHistoryScreen({super.key});
```

```
  @override
      State<VisitHistoryScreen>      createState()      =>
    _VisitHistoryScreenState();
}

class          _VisitHistoryScreenState          extends
    State<VisitHistoryScreen> {
  List<Appointment> all = [];
  List<Appointment> filtered = [];
  String query = '';
  String? selectedClient;
  // pagination
  static const int _perPage = 20;
  int _page = 0;

  @override
  void initState() {
   super.initState();
   _load();
  }

  Future<void> _load() async {
   final items = await DBService().allAppointments();
                        items.sort((a,          b)          =>
    b.dateTime.compareTo(a.dateTime));
   setState(() {
    all = items;
    _applyFilter();
   });
  }

  void _applyFilter() {
   // reset paging on new filter
   _page = 0;
   if (selectedClient != null) {
        filtered  =  all.where((a)  =>  a.clientName  ==
    selectedClient).toList();
   } else if (query.trim().isEmpty) {
    filtered = List.from(all);
   } else {
    final q = query.toLowerCase();
                filtered      =      all.where((a)      =>
    a.clientName.toLowerCase().contains(q)).toList();
   }
   setState(() {});
  }

        Map<DateTime,          List<Appointment>>
    _groupByDay(List<Appointment> items) {
   final Map<DateTime, List<Appointment>> map = {};
   for (final a in items) {
          final    d    =    DateTime(a.dateTime.year,
    a.dateTime.month, a.dateTime.day);
    map.putIfAbsent(d, () => []).add(a);
   }
   return map;
  }

List<String> _matchingClients() {
  final set = <String>{};
  for (final a in all) {
                                                  if
   (a.clientName.toLowerCase().contains(query.toLow
   erCase())) set.add(a.clientName);
  }
  final list = set.toList()..sort();
  return list;
}

@override
Widget build(BuildContext context) {
  final dateDf = DateFormat.yMMMd('uk');
```

```
final timeDf = DateFormat.Hm('uk');

// Use paged data for display
final total = filtered.length;
final totalPages = (total / _perPage).ceil().clamp(1, 9999);
final start = (_page * _perPage).clamp(0, total);
final end = ((start + _perPage) > total) ? total : (start + _perPage);
final pageItems = filtered.sublist(start, end);
final grouped = _groupByDay(pageItems);
final days = grouped.keys.toList()..sort((a, b) => b.compareTo(a));

final matchedClients = query.trim().isEmpty ? <String>[] : _matchingClients();

Appointment? latestForSelected;
if (selectedClient != null) {
  if (filtered.isNotEmpty) latestForSelected = filtered.first;
}

return Scaffold(
 appBar: AppBar(title: const Text('Записи клієнтів')),
 body: Padding(
  padding: const EdgeInsets.all(12.0),
  child: Column(
   children: [
    TextField(
     decoration: const InputDecoration(prefixIcon: Icon(Icons.search), labelText: 'Пошук клієнта'),
     onChanged: (v) {
      selectedClient = null;
      query = v;
      _applyFilter();
     },
    ),
    if (matchedClients.isNotEmpty && selectedClient == null)
     Padding(
      padding: const EdgeInsets.symmetric(vertical: 8.0),
      child: SizedBox(
       height: 40,
       child: ListView.separated(
        scrollDirection: Axis.horizontal,
        itemBuilder: (c, i) {
         final name = matchedClients[i];
         return ChoiceChip(
          label: Text(name),
          selected: false,
          onSelected: (_) {
           selectedClient = name;
           query = '';
           _applyFilter();
          },
         );
        },
        separatorBuilder: (_, __) => const SizedBox(width: 8),
        itemCount: matchedClients.length,
       ),
      ),
     ),
    if (selectedClient != null && latestForSelected != null)
     Card(
      color: Colors.grey.shade50,
      child: Padding(
       padding: const EdgeInsets.all(12.0),
       child: Row(
```

```
                                mainAxisAlignment:
MainAxisAlignment.spaceBetween,
            children: [
              Expanded(
                child: Column(
                                crossAxisAlignment:
CrossAxisAlignment.start,
                  children: [
                    Text(selectedClient ?? '', style: const
TextStyle(fontSize: 16, fontWeight:
FontWeight.bold)),
                    const SizedBox(height: 6),
                                Text('Останній запис:
${dateDf.format(latestForSelected.dateTime)}
${timeDf.format(latestForSelected.dateTime)}'),
                  ],
                ),
              ),
              Text(latestForSelected.price == null ? '-' :
'${latestForSelected.price!.toStringAsFixed(2)}₴',
style: const TextStyle(fontSize: 16, fontWeight:
FontWeight.bold)),
            ],
          ),
        ),
      ),
    const SizedBox(height: 8),
    // Pagination controls
    if (filtered.isNotEmpty)
      Padding(
        padding: const EdgeInsets.symmetric(vertical:
6.0),
        child: Row(
                                mainAxisAlignment:
MainAxisAlignment.center,
          children: [
            IconButton(
              onPressed: _page > 0 ? () => setState(() =>
_page--) : null,
              icon: const Icon(Icons.arrow_back),
            ),
            Text('Сторінка ${_page + 1} з $totalPages'),
            IconButton(
              onPressed: (_page < totalPages - 1) ? () =>
setState(() => _page++) : null,
              icon: const Icon(Icons.arrow_forward),
            ),
          ],
        ),
      ),
    Expanded(
      child: filtered.isEmpty
          ? const Center(child: Text('Немає записів'))
          : ListView.builder(
              itemCount: days.length,
              itemBuilder: (c, idx) {
                final day = days[idx];
                final items = grouped[day]!..sort((a, b) =>
b.dateTime.compareTo(a.dateTime));
                final dayTotal = items.fold<double>(0.0,
(s, a) => s + (a.price ?? 0.0));
                return Column(
                                crossAxisAlignment:
CrossAxisAlignment.start,
                  children: [
                    Padding(
                                padding: const
EdgeInsets.symmetric(vertical: 8.0, horizontal: 4.0),
                      child: Text('${dateDf.format(day)} —
${dayTotal.toStringAsFixed(2)}₴', style: const
TextStyle(fontSize: 16, fontWeight:
FontWeight.bold)),
                    ),
                    Padding(
```

```
                padding: const EdgeInsets.only(left: 8.0, bottom: 6.0),
                child: Text('Кількість записів: ${items.length}', style: TextStyle(fontSize: 13, color: Colors.grey.shade700)),
              ),
              ...items.map((a) => Column(children: [
                ListTile(
                  title: Text(a.clientName),
                  subtitle: Text('${timeDf.format(a.dateTime)} — ${a.note ?? ''}'),
                  trailing: Text(a.price == null ? '' : a.price!.toStringAsFixed(2)),
                ),
                const Divider(height: 1),
              ])),
            ],
          );
        },
      ),
    )
  ],
 ),
 ),
 );
 }
}
```

Б.11 – Сервіс збереження

```
import 'dart:convert';
import 'dart:io';

import 'package:flutter/foundation.dart';
import 'package:flutter/services.dart';
import 'package:path/path.dart' as p;
import 'package:path_provider/path_provider.dart';

import 'db_service.dart';

class BackupService {
  static final BackupService _instance = BackupService._internal();
 factory BackupService() => _instance;
 BackupService._internal();

  static const _channel = MethodChannel('beautydiary/backup');

Future<String> backupNow({bool toIcloud = false}) async {
 try {
  final items = await DBService().allAppointments();
  final list = items.map((a) => a.toMap()).toList();
   final jsonStr = jsonEncode({'generatedAt': DateTime.now().toIso8601String(), 'appointments': list});

   final dir = await getApplicationDocumentsDirectory();
  final backupsDir = Directory(p.join(dir.path, 'backups'));
   if (!await backupsDir.exists()) await backupsDir.create(recursive: true);
   final file = File(p.join(backupsDir.path, 'backup_${DateTime.now().toIso8601String().replaceAll(':', '-')}.json'));
  await file.writeAsString(jsonStr);
```

```
        if (toIcloud && defaultTargetPlatform ==
    TargetPlatform.iOS) {
     try {
       await _channel.invokeMethod('backupToICloud',
    {'path': file.path});
       return 'Backup saved locally and synced to iCloud:
    ${file.path}';
     } catch (e) {
       return 'Local backup saved but iCloud sync failed:
    $e';
     }
    }

    return 'Local backup saved: ${file.path}';
   } catch (e) {
    return 'Backup failed: $e';
   }
  }
}
```

Б.12 – Код збереження в БД

```
import 'dart:async';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
import
    'package:shared_preferences/shared_preferences.dar
    t';
import '../models/appointment.dart';
import 'backup_service.dart';
import 'notification_service.dart';

class DBService {
      static final DBService _instance =
    DBService._internal();
 factory DBService() => _instance;
 DBService._internal();

 Database? _db;

 Future<Database> get db async {
  if (_db != null) return _db!;
  _db = await _initDB();
  return _db!;
 }

Future<Database> _initDB() async {
 final databasesPath = await getDatabasesPath();
 final path = join(databasesPath, 'beautydiary.db');
 return await openDatabase(path, version: 2, onCreate:
  (db, v) async {
  await db.execute('''
   CREATE TABLE appointments(
                id INTEGER PRIMARY KEY
  AUTOINCREMENT,
    clientName TEXT NOT NULL,
    dateTime TEXT NOT NULL,
    phone TEXT,
    price REAL,
    note TEXT,
    photoPath TEXT
   )
  ''');
 }, onUpgrade: (db, oldV, newV) async {
  if (oldV < 2) {
   // add photoPath column for migration from version
  1
    await db.execute('ALTER TABLE appointments
  ADD COLUMN photoPath TEXT');
  }
```

```
  });
}

Future<int> insertAppointment(Appointment a) async
  {
 final database = await db;
   final res = await database.insert('appointments',
  a.toMap());
 unawaited(_maybeBackup());
 unawaited(NotificationService().rescheduleAll());
 return res;
}

Future<int> updateAppointment(Appointment a) async
  {
 final database = await db;
 if (a.id == null) {
  final id = await insertAppointment(a);
  return id;
 }
   final res = await database.update('appointments',
  a.toMap(), where: 'id = ?', whereArgs: [a.id]);
 unawaited(_maybeBackup());
 unawaited(NotificationService().rescheduleAll());
 return res;
}

Future<int> deleteAppointment(int id) async {
 final database = await db;
   final res = await database.delete('appointments',
  where: 'id = ?', whereArgs: [id]);
 unawaited(_maybeBackup());
 unawaited(NotificationService().rescheduleAll());
 return res;
}

// Trigger backup after mutating operations if auto-
  backup enabled
Future<void> _maybeBackup() async {
 try {
  final prefs = await SharedPreferences.getInstance();
  final auto = prefs.getBool('autoBackup') ?? false;
  if (!auto) return;
  final icl = prefs.getBool('icloudSync') ?? false;
  // run but don't await to avoid blocking DB operations
  BackupService().backupNow(toIcloud: icl);
 } catch (_) {}
}

Future<int> deleteAllAppointments() async {
 final database = await db;
 final res = await database.delete('appointments');
 unawaited(_maybeBackup());
 unawaited(NotificationService().rescheduleAll());
 return res;
}

                          Future<List<Appointment>>
  appointmentsForDay(DateTime day) async {
 final database = await db;
 final start = DateTime(day.year, day.month, day.day);
 final end = start.add(Duration(days: 1));
 final maps = await database.query('appointments',
   where: 'dateTime >= ? AND dateTime < ?',
              whereArgs:    [start.toIso8601String(),
  end.toIso8601String()],
   orderBy: 'dateTime ASC');
              return        maps.map((m)        =>
  Appointment.fromMap(m)).toList();
}

Future<List<Appointment>> allAppointments() async
```

```
  {
  final database = await db;
   final maps = await database.query('appointments', orderBy: 'dateTime ASC');
                return maps.map((m) => Appointment.fromMap(m)).toList();
 }

  Future<bool> hasConflict(DateTime candidate, int minutesInterval, {int? ignoreId}) async {
  final database = await db;
   final from = candidate.subtract(Duration(minutes: minutesInterval));
     final to = candidate.add(Duration(minutes: minutesInterval));
  String where = 'dateTime > ? AND dateTime < ?';
    final args = <Object?>[from.toIso8601String(), to.toIso8601String()];
  if (ignoreId != null) {
   where += ' AND id != ?';
   args.add(ignoreId);
  }
   final maps = await database.query('appointments', where: where, whereArgs: args);
  return maps.isNotEmpty;
 }

 Future<double> sumPricesBetween(DateTime from, DateTime to) async {
  final database = await db;
  final res = await database.rawQuery(
    'SELECT SUM(price) as total FROM appointments WHERE dateTime >= ? AND dateTime < ?',
    [from.toIso8601String(), to.toIso8601String()]);
  final val = res.first['total'];
  if (val == null) return 0.0;
  return (val as num).toDouble();
 }
}
```

Б.13 – Код експорту в пдф/ексель

```
import 'dart:io';
import 'package:intl/intl.dart';
import 'package:path_provider/path_provider.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:printing/printing.dart';
import 'package:excel/excel.dart';
import '../models/appointment.dart';

class ExportService {
                                        Future<String> exportAppointmentsToPdf(List<Appointment> items) async {
  final doc = pw.Document();
  final df = DateFormat('yyyy-MM-dd HH:mm');
  doc.addPage(pw.MultiPage(build: (c) {
   return [
    pw.Header(level: 0, child: pw.Text('Записи')),
    pw.TableHelper.fromTextArray(
        headers: ['Дата', 'Клієнт', 'Телефон', 'Ціна', 'Опис', 'Фото'],
      data: items
       .map((a) => [
         df.format(a.dateTime),
         a.clientName,
         a.phone ?? '',
           a.price == null ? '' : (a.price! % 1 == 0 ? '${a.price!.toStringAsFixed(0)}₴' : '${a.price!.toStringAsFixed(2)}₴'),
```

```
            a.note ?? '',
            a.photoPath ?? ''
          ])
        .toList())
  ];
}));

final bytes = await doc.save();
final dir = await getApplicationDocumentsDirectory();
final file = File('${dir.path}/appointments_${DateTime.now().millisecondsSinceEpoch}.pdf');
await file.writeAsBytes(bytes);
try {
  await Printing.sharePdf(bytes: bytes, filename: file.path.split(Platform.pathSeparator).last);
} catch (_) {}
return file.path;
}

Future<String> exportAppointmentsToExcel(List<Appointment> items) async {
final excel = Excel.createExcel();
final sheetName = excel.getDefaultSheet()!;
final sheet = excel[sheetName];
sheet.appendRow(['Дата', 'Клієнт', 'Телефон', 'Ціна', 'Опис', 'Фото']);
final df = DateFormat('yyyy-MM-dd HH:mm');
for (final a in items) {
final priceStr = a.price == null ? '' : (a.price! % 1 == 0 ? a.price!.toStringAsFixed(0) + '₴' : a.price!.toStringAsFixed(2) + '₴');
sheet.appendRow([df.format(a.dateTime), a.clientName, a.phone ?? '', priceStr, a.note ?? '', a.photoPath ?? '']);
}
final bytes = excel.encode();
final dir = await getApplicationDocumentsDirectory();
final file = File('${dir.path}/appointments_${DateTime.now().millisecondsSinceEpoch}.xlsx');
if (bytes != null) await file.writeAsBytes(bytes);
return file.path;
}
}
```

Б.14 – Сервіс налаштування сповіщень

```
import 'dart:async';

import 'dart:io';
import 'package:flutter/foundation.dart';
import 'package:flutter/services.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
import 'package:timezone/data/latest_all.dart' as tzdata;
import 'package:timezone/timezone.dart' as tz;
import 'package:shared_preferences/shared_preferences.dart';

import 'db_service.dart';
import '../models/appointment.dart';

class NotificationService {
  static final NotificationService _instance = NotificationService._internal();
  factory NotificationService() => _instance;
  NotificationService._internal();
```

```
 final FlutterLocalNotificationsPlugin _plugin =
  FlutterLocalNotificationsPlugin();

Future<void> init() async {
                    const android =
  AndroidInitializationSettings('@mipmap/ic_launche
  r');
 final ios = DarwinInitializationSettings(
  requestSoundPermission: true,
  requestBadgePermission: true,
  requestAlertPermission: true,
);
 final settings = InitializationSettings(android: android,
  iOS: ios);
 await _plugin.initialize(settings);

// On iOS/macOS request permissions explicitly (helps
  when plugin defaults don't trigger)
 try {
  if (Platform.isIOS || Platform.isMacOS) {
                        final iosImpl =
  _plugin.resolvePlatformSpecificImplementation<IO
  SFlutterLocalNotificationsPlugin>();
      await iosImpl?.requestPermissions(alert: true,
  badge: true, sound: true);
  }
 } catch (e) {
  debugPrint('iOS permission request error: $e');
 }

 // Create Android notification channels (idempotent)
 try {
  if (Platform.isAndroid) {
                        final androidImpl =
  _plugin.resolvePlatformSpecificImplementation<A
  ndroidFlutterLocalNotificationsPlugin>();
                                                  await
  androidImpl?.createNotificationChannel(const
  AndroidNotificationChannel(
    'bd_morning',
    'Morning reminders',
    description: 'Daily morning reminder',
    importance: Importance.defaultImportance,
  ));
                                                  await
  androidImpl?.createNotificationChannel(const
  AndroidNotificationChannel(
    'bd_pre',
    'Pre-event reminders',
    description: 'Reminders before appointment',
    importance: Importance.high,
  ));
  }
 } catch (e) {
   debugPrint('Failed to create notification channels:
  $e');
 }

// timezone
tzdata.initializeTimeZones();
// Try to set a reasonable local timezone. Prefer device
  timezone name
// if available; fall back to UTC to avoid native plugin
  build issues.
try {
  final name = DateTime.now().timeZoneName;
  try {
   tz.setLocalLocation(tz.getLocation(name));
  } catch (_) {
   tz.setLocalLocation(tz.UTC);
  }
} catch (e) {
```

```
    tz.setLocalLocation(tz.UTC);
  }
}

Future<void> cancelAll() => _plugin.cancelAll();

Future<void> rescheduleAll() async {
  try {
    final prefs = await SharedPreferences.getInstance();
    final enabled = prefs.getBool('remindersEnabled') ??
    true;
    if (!enabled) {
      await cancelAll();
      return;
    }

    final morningHour = prefs.getInt('morningHour') ??
    9;
    final morningMinute = prefs.getInt('morningMinute')
    ?? 0;
    final preOffset = prefs.getInt('preOffsetMinutes') ??
    60;

    await cancelAll();

    // schedule today's morning summary
          await _scheduleMorningReminder(hour:
    morningHour, minute: morningMinute);

      // schedule pre-event reminders for upcoming
    appointments
                                          await
    _schedulePreEventReminders(preOffsetMinutes:
    preOffset);
  } catch (e) {
    debugPrint('rescheduleAll error: $e');
  }
}

Future<void>  _scheduleMorningReminder({required
  int hour, required int minute}) async {
  final now = tz.TZDateTime.now(tz.local);
  var scheduled = tz.TZDateTime(tz.local, now.year,
  now.month, now.day, hour, minute);
  if (scheduled.isBefore(now)) {
    // schedule for next day if time already passed
    scheduled = scheduled.add(const Duration(days: 1));
  }

  // prepare payload with first appointment info
              final first = await
  _firstAppointmentOfDay(DateTime.now());
  final title = 'Плани на сьогодні';
  final body = first == null
      ? 'Немає записів на сьогодні'
                  : '${first.clientName} o
  ${_formatTime(first.dateTime)} — всього ${await
  _countAppointmentsForDay(DateTime.now())}
  записів';

  try {
    await _plugin.zonedSchedule(
      1000,
      title,
      body,
      scheduled,
      const NotificationDetails(
        android: AndroidNotificationDetails('bd_morning',
    'Morning reminders', channelDescription: 'Daily
    morning reminder'),
        iOS: DarwinNotificationDetails(),
      ),
                                androidScheduleMode:
    AndroidScheduleMode.exactAllowWhileIdle,
```

```
                uiLocalNotificationDateInterpretation:
  UILocalNotificationDateInterpretation.absoluteTim
  e,
                matchDateTimeComponents:
  DateTimeComponents.time,
  );
 } on PlatformException catch (e) {
   debugPrint('Morning schedule exact alarm failed:
  $e');
  // Retry with inexact mode for devices/emulators that
  disallow exact alarms
  await _plugin.zonedSchedule(
   1000,
   title,
   body,
   scheduled,
   const NotificationDetails(
    android: AndroidNotificationDetails('bd_morning',
  'Morning reminders', channelDescription: 'Daily
  morning reminder'),
     iOS: DarwinNotificationDetails(),
   ),
                androidScheduleMode:
  AndroidScheduleMode.inexactAllowWhileIdle,
                uiLocalNotificationDateInterpretation:
  UILocalNotificationDateInterpretation.absoluteTim
  e,
                matchDateTimeComponents:
  DateTimeComponents.time,
  );
 }

}

Future<void> _schedulePreEventReminders({required
  int preOffsetMinutes}) async {
 final all = await DBService().allAppointments();
 final now = DateTime.now();
 for (final a in all) {
  final target = a.dateTime.subtract(Duration(minutes:
  preOffsetMinutes));
  if (target.isBefore(now)) continue;
     final scheduled = tz.TZDateTime.from(target,
  tz.local);
  final title = 'Наступний запис';
     final body = '${_formatTime(a.dateTime)} —
  ${a.clientName}';
  try {
   await _plugin.zonedSchedule(
    a.id ?? target.millisecondsSinceEpoch % 100000,
    title,
    body,
    scheduled,
    const NotificationDetails(
         android: AndroidNotificationDetails('bd_pre',
  'Pre-event reminders', channelDescription:
  'Reminders before appointment'),
      iOS: DarwinNotificationDetails(),
    ),
                androidScheduleMode:
  AndroidScheduleMode.exactAllowWhileIdle,
                uiLocalNotificationDateInterpretation:
  UILocalNotificationDateInterpretation.absoluteTim
  e,
   );
  } on PlatformException catch (e) {
   debugPrint('Pre-event exact alarm failed for ${a.id}:
  $e');
   await _plugin.zonedSchedule(
    a.id ?? target.millisecondsSinceEpoch % 100000,
    title,
    body,
    scheduled,
    const NotificationDetails(
```

```
        android: AndroidNotificationDetails('bd_pre',
    'Pre-event reminders', channelDescription:
    'Reminders before appointment'),
        iOS: DarwinNotificationDetails(),
      ),
                              androidScheduleMode:
    AndroidScheduleMode.inexactAllowWhileIdle,
                    uiLocalNotificationDateInterpretation:
    UILocalNotificationDateInterpretation.absoluteTim
    e,
     );
    }
  }
}


                              Future<Appointment?>
    _firstAppointmentOfDay(DateTime day) async {
                    final items = await
    DBService().appointmentsForDay(day);
   if (items.isEmpty) return null;
                    items.sort((a, b) =>
     a.dateTime.compareTo(b.dateTime));
   return items.first;
  }

   Future<int> _countAppointmentsForDay(DateTime
    day) async {
                    final items = await
    DBService().appointmentsForDay(day);
   return items.length;
  }

  String _formatTime(DateTime dt) {
   final h = dt.hour.toString().padLeft(2, '0');
   final m = dt.minute.toString().padLeft(2, '0');
   return '$h:$m';
  }
}
```

Б.15 – main.dart

```
import 'package:flutter/material.dart';
import 'package:intl/date_symbol_data_local.dart';
import
    'package:flutter_localizations/flutter_localizatio
    ns.dart';
import 'screens/splash_screen.dart';
import 'screens/home_screen.dart';
import 'screens/new_appointment_screen.dart';
import 'screens/stats_screen.dart';
import 'screens/settings_screen.dart';
import 'services/notification_service.dart';
import 'screens/visit_history_screen.dart';
import 'screens/photo_gallery_screen.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await initializeDateFormatting('uk');
  WidgetsBinding.instance.addPostFrameCallback(
    (_) async {
   await NotificationService().init();
   await NotificationService().rescheduleAll();
  });
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
```

```
const MyApp({super.key});

@override
Widget build(BuildContext context) {
 final base = ThemeData.light();
 final theme = base.copyWith(
  useMaterial3: true,
  colorScheme: ColorScheme.fromSeed(
   seedColor: Colors.grey,
   brightness: Brightness.light,
   primary: Colors.grey.shade900,
   onPrimary: Colors.white,
   background: Colors.white,
   surface: Colors.grey.shade100,
  ),
                                    appBarTheme:
  AppBarTheme(backgroundColor:
  Colors.grey.shade900,          foregroundColor:
  Colors.white),
                            scaffoldBackgroundColor:
  Colors.grey.shade50,
                          floatingActionButtonTheme:
  FloatingActionButtonThemeData(background
  Color: Colors.grey.shade900),
   textTheme:  base.textTheme.apply(bodyColor:
  Colors.black87, displayColor: Colors.black87),
 );

 return MaterialApp(
  title: 'Beauty Diary',
                                                theme:
  theme.copyWith(floatingActionButtonTheme:
  theme.floatingActionButtonTheme.copyWith(f
  oregroundColor: Colors.white)),
  locale: const Locale('uk'),
        supportedLocales:  const  [Locale('uk'),
  Locale('en')],
  localizationsDelegates: const [
   GlobalMaterialLocalizations.delegate,
   GlobalWidgetsLocalizations.delegate,
   GlobalCupertinoLocalizations.delegate,
  ],
  routes: {
   '/': (c) => const SplashScreen(),
   '/home': (c) => const HomeScreen(),
   '/history': (c) => const VisitHistoryScreen(),
   '/new': (c) => const NewAppointmentScreen(),
    '/photos': (c) => const PhotoGalleryScreen(),
   '/stats': (c) => const StatsScreen(),
   '/settings': (c) => const SettingsScreen(),
  },
  initialRoute: '/',
 );
 }
}
```

ДОДАТОК В

(Обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Рисунок В.1 – Слайд 1

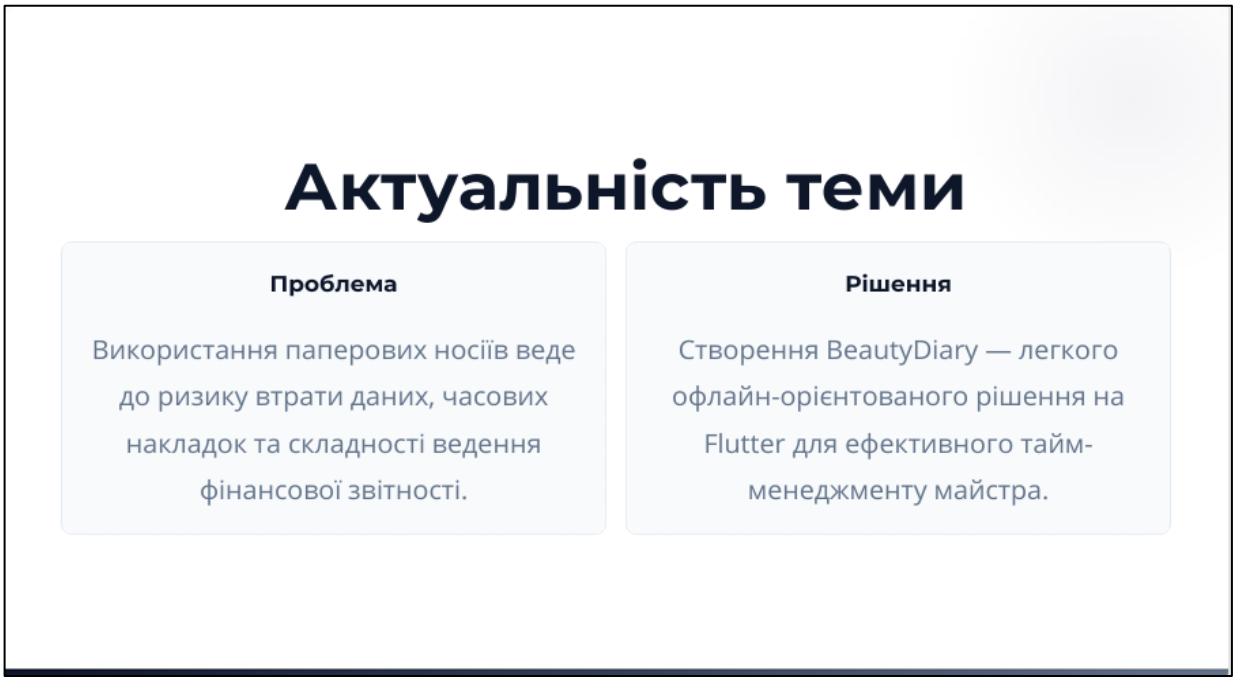


Рисунок В.2 – Слайд 2

Мета та Завдання

Мета: Розроблення ПЗ для автоматизації робочих процесів індивідуальних майстрів б'юті-індустрії.

- Аналіз предметної області
- Обґрунтування тех-стеку
- Проєктування архітектури
- Програмна реалізація
- Порівняння аналогів ПЗ
- Визначення вимог
- Проєктування БД та UI
- Тестування результатів

Рисунок В.3 – Слайд 3

Аналіз предметної області

Об'єкт автоматизації

Діяльність приватного майстра б'юті-послуг (манікюр, педикюр).

Групи процесів:

- Управління записами клієнтів
- Ведення історії відвідувань
- Фінансовий облік і статистика
- Організація часу та нагадування

Рисунок В.4 – Слайд 4

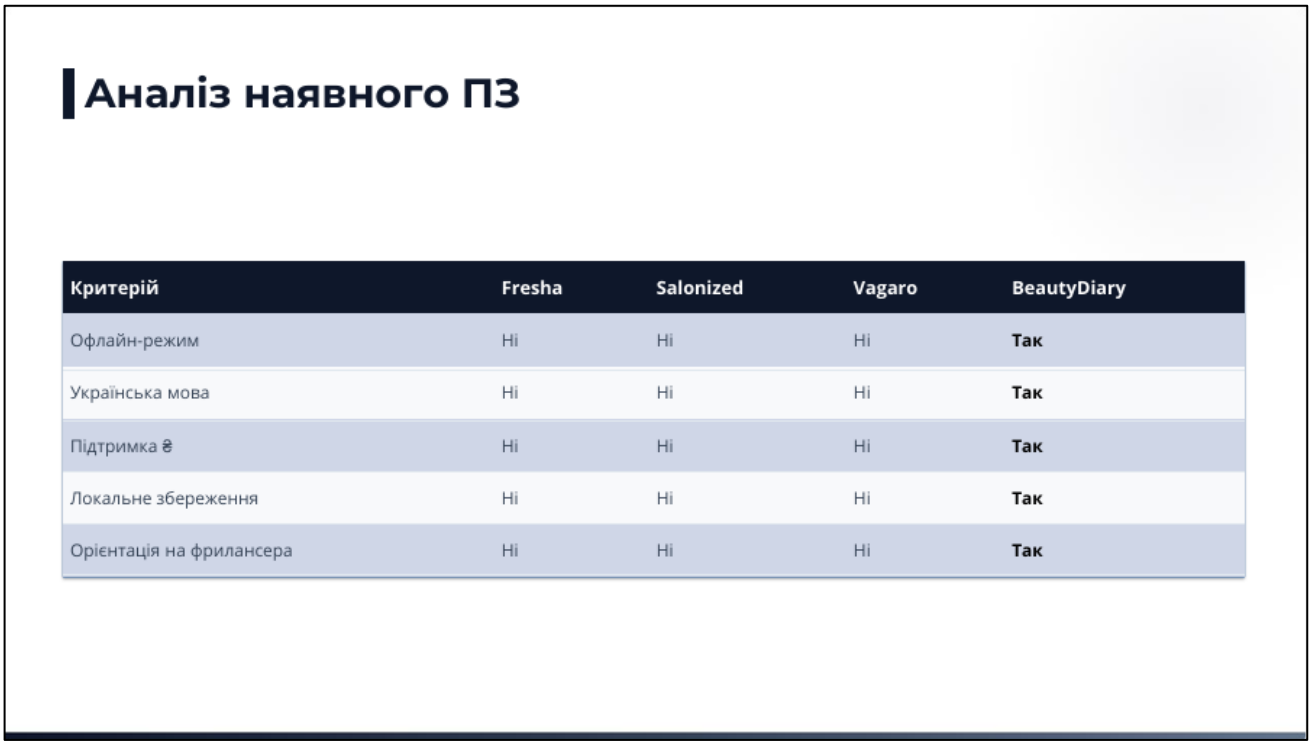


Рисунок В.5 – Слайд 5

Рисунок В.6 – Слайд 6

Вибір типу архітектури

- **Трирівнева архітектура:** Представлення, Сервіси, Дані.
- **Патерни:** Singleton (Одинак) для сервісів.
- **Розподіл:** DbService, NotificationService, ExportService.

Забезпечує узгодженість даних та спрощує тестування окремих модулів.

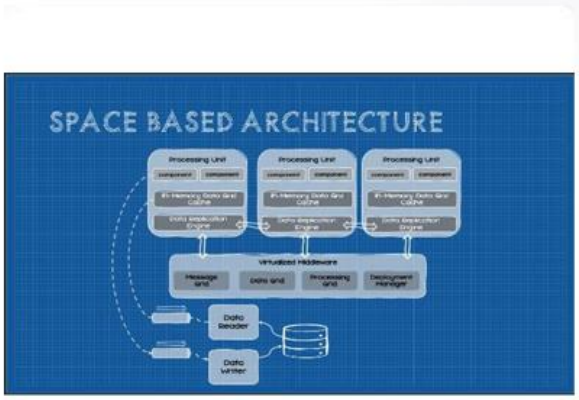


Рисунок В.7 – Слайд 7

Декомпозиція та модулі

- **UML Діаграма зв'язків:** Відображає взаємодію між UI та сервісами.
- **Інтерфейси:** Асинхронний обмін даними між SQLite та віджетами.
- **Автоматизація:** Тригери на оновлення розкладу сповіщень.

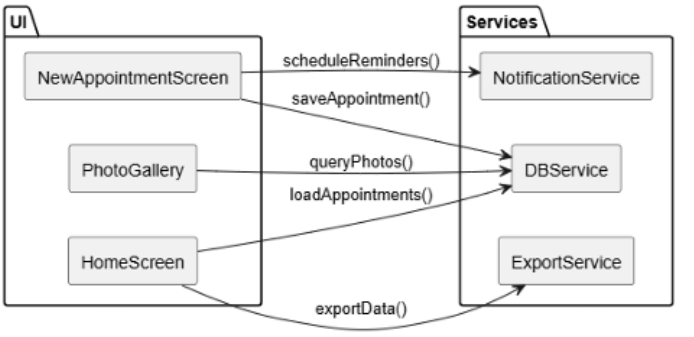


Рисунок В.8 – Слайд 8

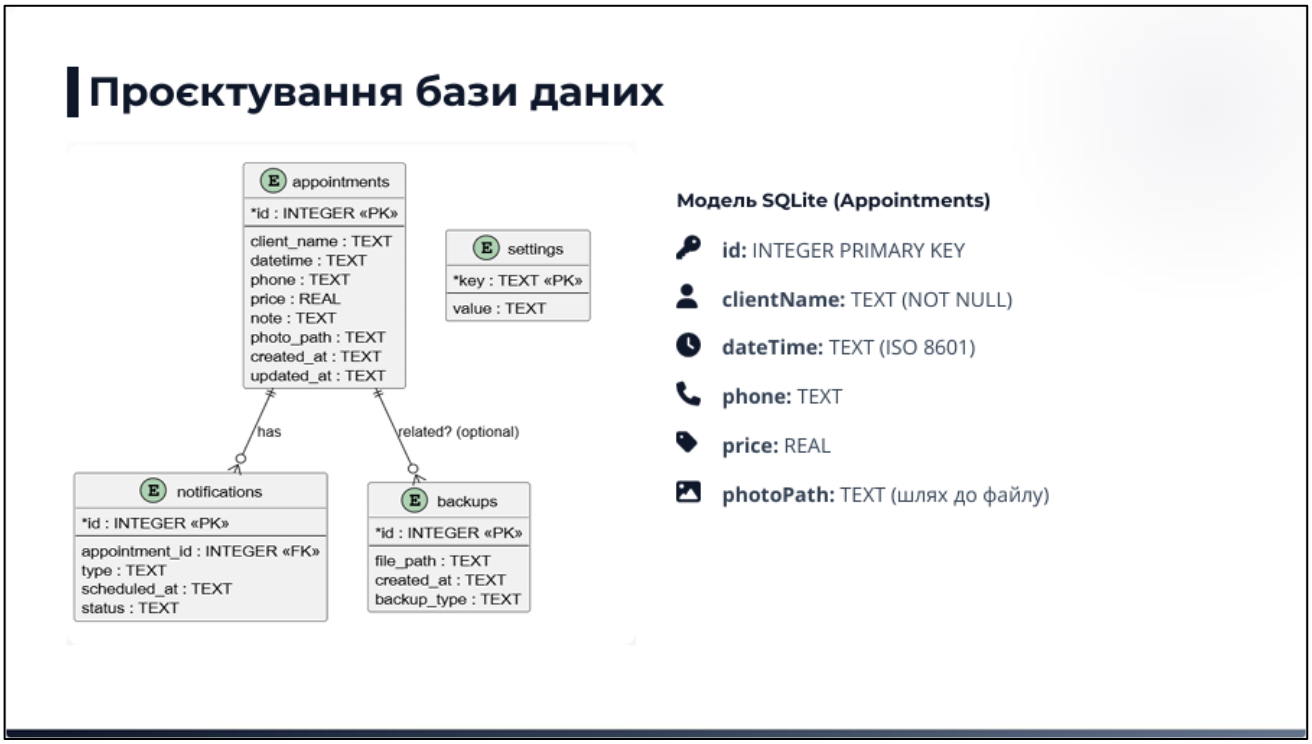


Рисунок В.9 – Слайд 9

Рисунок В.10 – Слайд 10

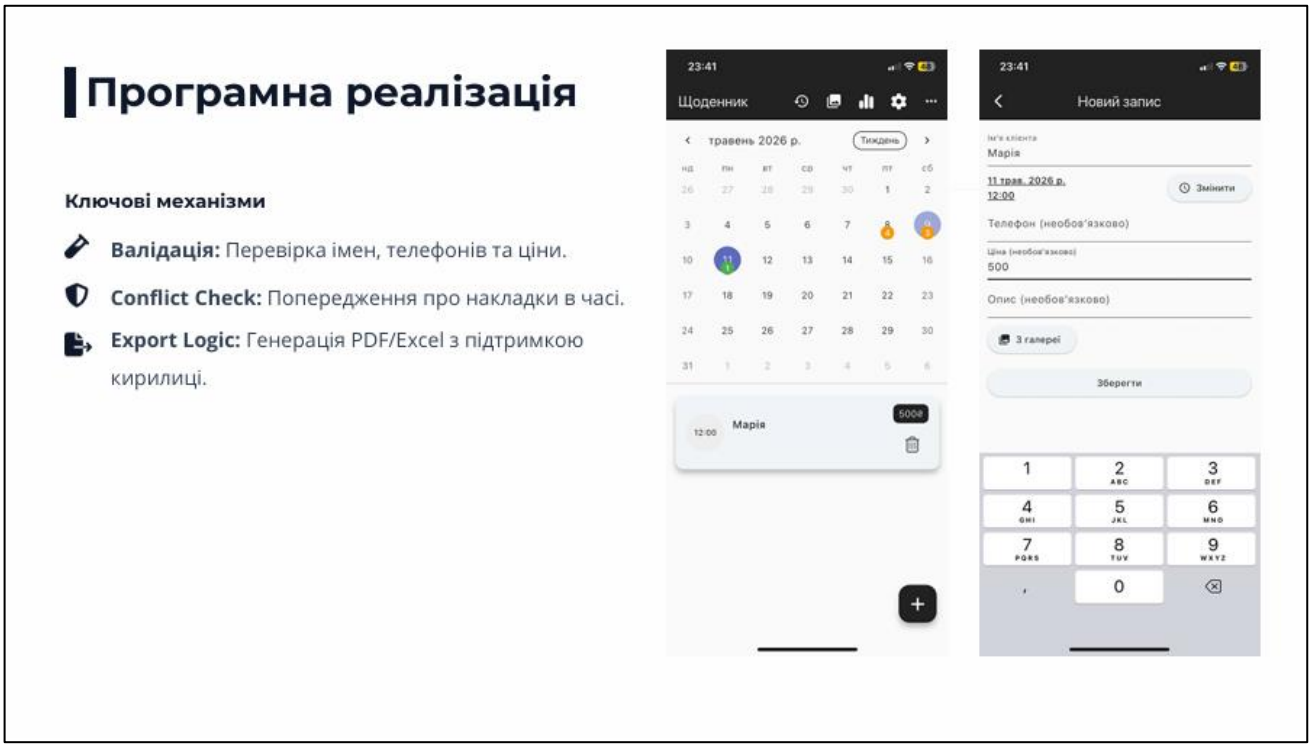


Рисунок В.11 – Слайд 11

Рисунок В.12 – Слайд 12

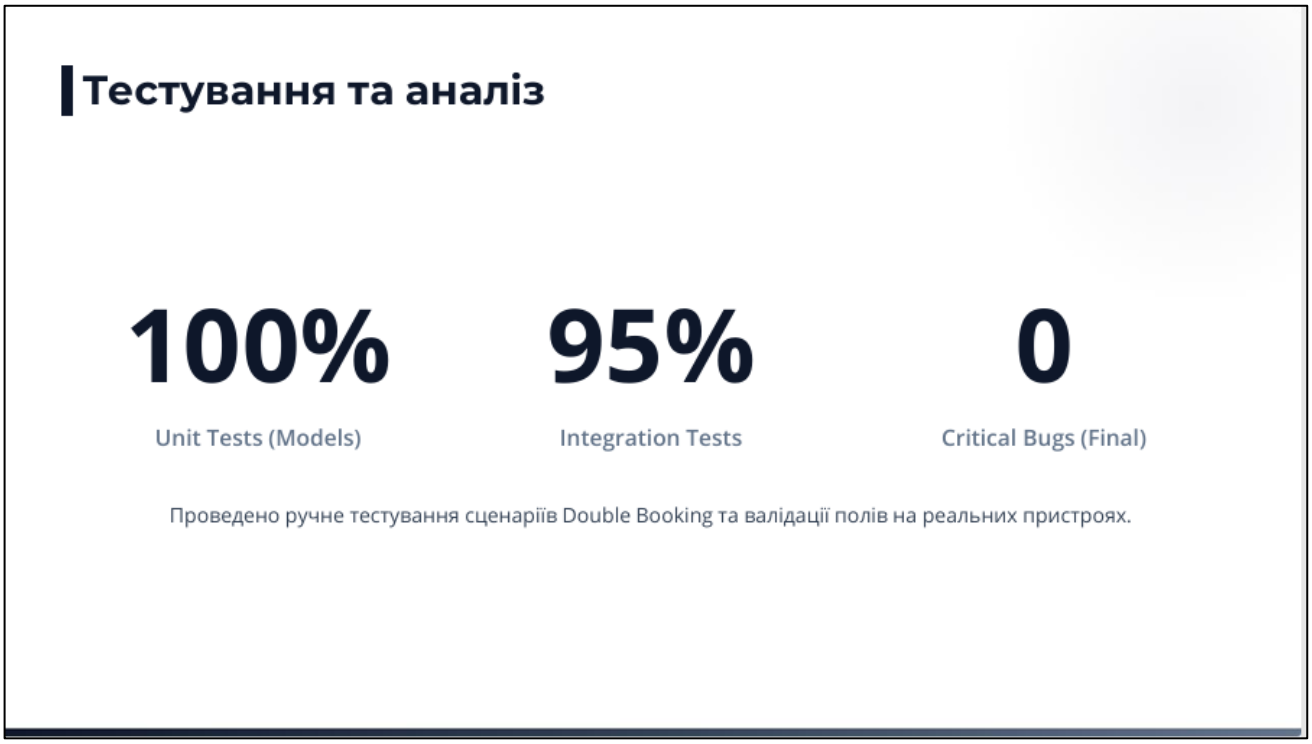


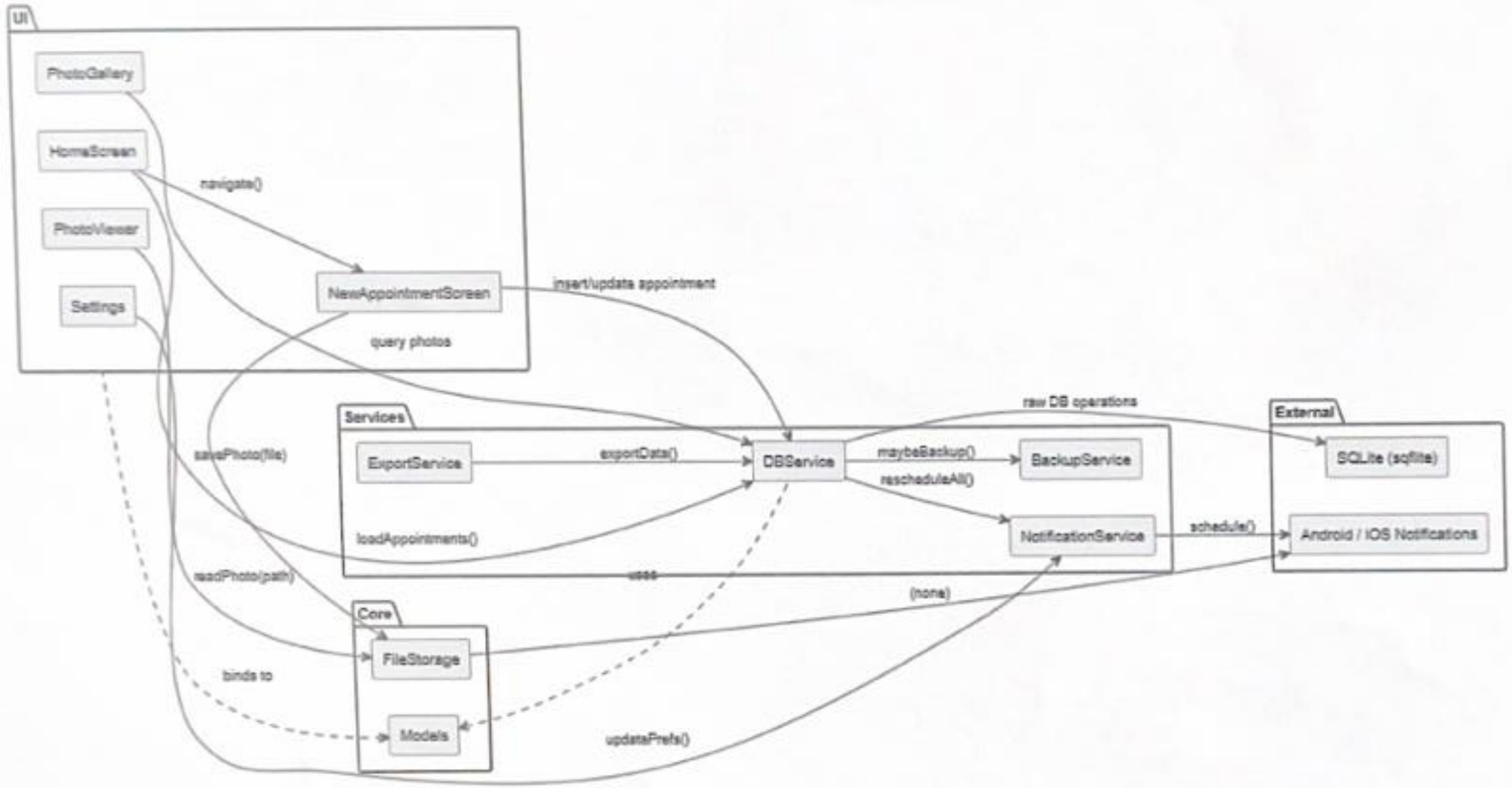
Рисунок В.13 – Слайд 13

Рисунок В.14 – Слайд 14

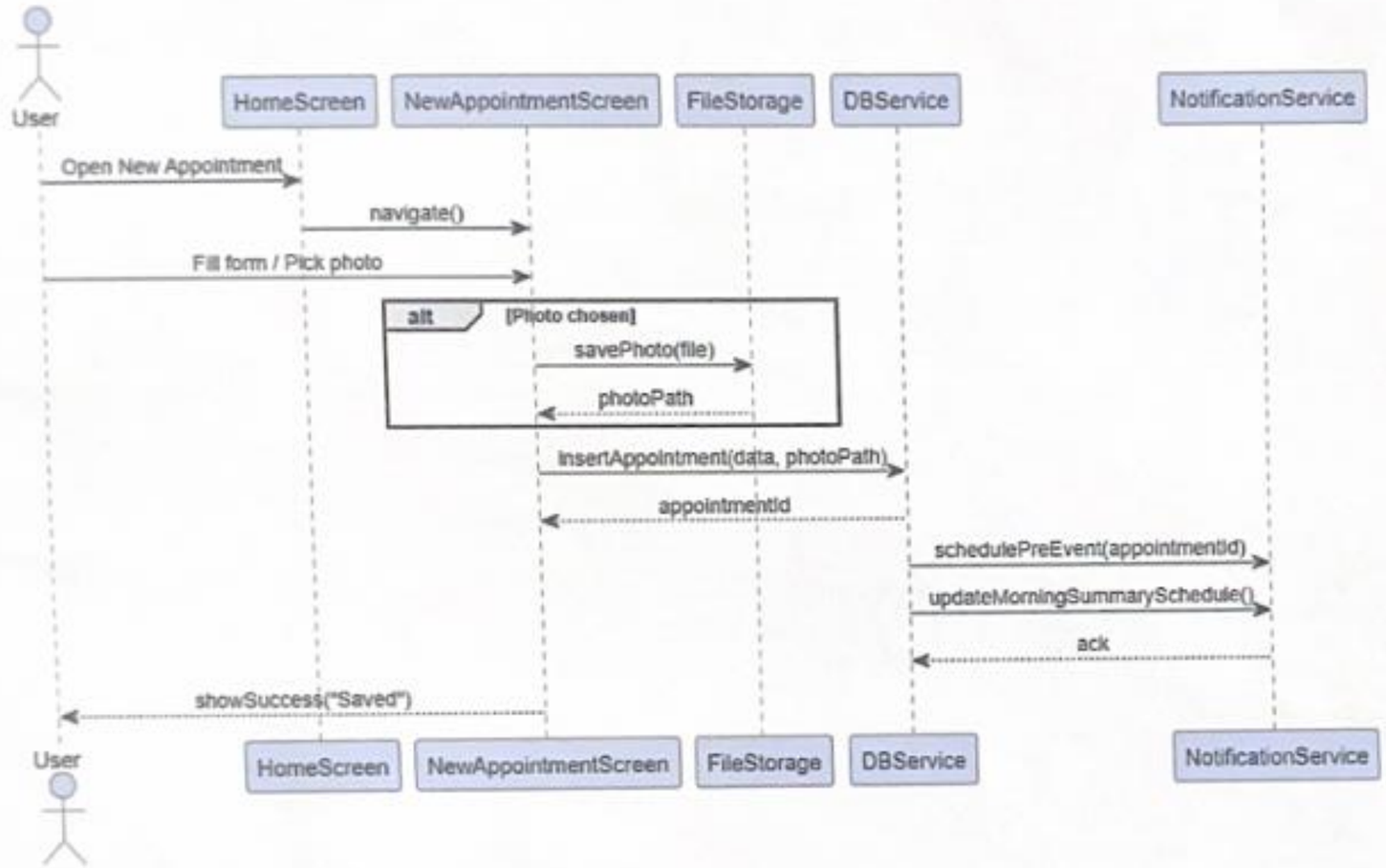
Рисунок В.15 – Слайд 15

ГРАФІЧНІ МАТЕРІАЛИ

					КвРІПЗ.230128.01.01.Е8			
Змн.	Лист	№ докум.	Підпис	Дата				
Розробив		Балицький І.І.		25.05	Діаграма варіантів використання	Літ.	Маса	Масштаб
Керівник		Бедратюк Г.І.		25.05				
Консульт.						Арк. 1	Аркушів 3	
Н. Контр.				25.05		ХНУ, ІПЗс–23–1		
Затверд.		Бедратюк Л.П.		25.05				



					КвРІПЗ.230128.01.01.Е8			
Змн.	Лист	№ докум.	Підпис	Дата				
Розробив		Баліцький І.І.		25.05	Діаграма зв'язків модулів	Літ.	Маса	Масштаб
Керівник		Бедратюк Г.І.		25.05				
Консульт.						Арк. 2	Аркушів 3	
Н. Контр.						ХНУ, ІПЗс–23–1		
Затверд.		Бедратюк Л.П.		25.05				



					КвРІПЗ.230128.01.01.Е8		
Змн.	Лист	№ докум.	Підпис	Дата			
Розробив		Баліцький І.І.		25.05	Діаграма послідовності	Літ.	Маса	Масштаб
Керівник		Бедратюк Г.І.		25.05				
Консульт.						Арк. 3	Аркушів 3	
Н. Контр.				25.05		ХНУ, ІПЗс–23–1		
Затвердв.		Бедратюк Л.П.		25.05				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Баліцького Ігоря Ігоровича
факультет ІТ, ІІІ курс, група ІПЗс-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

19.05.2026
дата

підпис

Wed May 20 00:47:29 EEST 2026, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 13%**

	Документ		Сумарний збіг по Базі Даних	
ID: 271747 Назва: БКР_Мобільний застосунок для керування клієнтськими записами та фінансовим обліком у сфері краси Додано в БД: 2026-05-20 Автора: Ігор БАЛІЦЬКИЙ Керівники: ст. викладач Ганна БЕДРАТЮК Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	113360	858	4806 (4%)	68 (8%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Ігор БАЛІЦЬКИЙ

Співавтор:

Назва: Мобільний застосунок для керування клієнтськими записами та фінансовим обліком у сфері краси

Науковий керівник: ст. викладач Ганна БЕДРАТЮК

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:3.46%

Коефіцієнт подібності 2:1.49%

Мікропробіли: 0

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-20 00:50:23.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 25.05.26

експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Баліцький Ігор Ігорович

Тема Мобільний застосунок для керування клієнтськими записами та фінансовим обліком у сфері краси

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 82

1.Короткий зміст пояснювальної записки та прийнятих рішень Робота присвячена розробці мобільного застосунку «BeautyDiary» для автоматизації діяльності майстрів б'юті-індустрії. У пояснювальній записці обґрунтовано актуальність теми, проведено аналіз предметної області та існуючих програмних аналогів. Розроблено архітектуру системи, спроєктовано UX/UI інтерфейс та реалізовано програмне забезпечення на базі кросплатформового фреймворку Flutter із використанням БД SQLite для локального зберігання даних.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано високу актуальність автоматизації бізнес-процесів у сфері індивідуального підприємництва б'юті-індустрії та чітко визначено мету роботи. У першому розділі проведено детальний аналіз предметної області, досліджено існуючі програмні рішення (CRM-системи та планувальники), визначено їхні функціональні переваги та недоліки, на основі чого сформовано перелік вимог до власного ПЗ. У другому розділі проаналізовано сучасні архітектурні підходи до розробки мобільних додатків. Обґрунтовано вибір кросплатформової технології Flutter у поєднанні з мовою програмування Dart, що забезпечує високу продуктивність та ергономічність інтерфейсу. Розглянуто особливості проєктування реляційної структури БД SQLite, що гарантує надійне локальне зберігання даних користувача. У третьому розділі описано практичну реалізацію програмних модулів: журналу записів, системи фінансового обліку та графічного інтерфейсу. Виконано модульне та інтеграційне тестування системи у відповідності до функціональних вимог, що підтвердило коректність роботи всіх алгоритмів, зокрема механізмів обробки даних записів та автоматизованого фінансового звітування. Робота демонструє високий рівень володіння інструментарієм сучасної мобільної розробки.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною. До основних переваг застосунку «BeautyDiary» належить висока функціональна насиченість, що повністю задовольняє потреби майстрів б'юті-індустрії: від ведення журналу записів до складної фінансової аналітики. Важливою перевагою є продумана візуалізація робочого навантаження в календарі та зручна робота з фотографіями робіт

(pinch-to-zoom, галерея). Система нагадувань та можливість експорту звітів у PDF/Excel роблять застосунок професійним інструментом, готовим до практичного впровадження.

5. Негативні сторони роботи Певним обмеженням поточної версії є виключно локальний спосіб зберігання даних у базі SQLite, що покладає на користувача відповідальність за створення резервних копій. Доцільним було б розширення функціоналу автоматичним інтегрованим синхронізуванням із хмарними сховищами (наприклад, Google Drive). Також, попри зручність обробки фото, впровадження прямого інтегрованого доступу до камери в додатку значно підвищило б оперативність роботи майстра безпосередньо під час візиту клієнта.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення роботи виконано на високому рівні. Пояснювальна записка структурована логічно, матеріал викладено послідовно та доказово. UML-діаграми, що описують архітектуру сервісів та схему бази даних, є інформативними та полегшують розуміння принципів роботи системи. Оформлення відповідає чинним державним стандартам.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота є завершеним та самостійним дослідженням. Програмний продукт «BeautyDiary» вирізняється практичною цінністю та високою якістю програмного коду, що базується на сучасних паттернах проектування (архітектура сервісів-синглтонів). Робота в цілому демонструє зрілість автора як фахівця з розробки програмного забезпечення для мобільних платформ.

8. Інші зауваження Суттєвих зауважень немає. У перспективі доцільно розглянути впровадження синхронізації через Firebase або аналогічні сервіси для забезпечення кросплатформової доступності даних на кількох пристроях майстра.

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленим завданням та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Капустян Марія Вікторівна, к.т.н., доцент каф. КПС ХНУ

"25" травня 2026 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Мобільний застосунок для керування клієнтськими записами та фінансовим обліком у сфері краси»
Автор: Баліцький Ігор Ігорович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Ганна Іванівна, ст. викладач
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;
2) запозичення, виявлені у тексті роботи, є фрагментарними.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 2.0% з одного джерела. Загальна сумарна подібність у базі даних складає 4% за символами та 8% за лексемами. Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 3.46%, коефіцієнт подібності 2 – 1.49%. Не виявлено мікропробілів, зайвих білих знаків або маніпуляцій з інтервалами. З урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 25 травня 2026

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Ганна БЕДРАТЮК