ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для контролю споживання кофеїну»

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

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

_____ Роман ЧОРНИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Роман ЧОРНИЙ

Керівник: Ірина ЩЕРБИНА
к.т.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чорному Роману Олександровичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку для контролю споживання кофеїну»
керівник кваліфікаційної роботи к.т.н., доцент Ірина ЩЕРБИНА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про архітектурні шаблони проектування мобільних застосунків (MVVM), опис методів локального збереження даних на платформі Android (Room), технічна документація бібліотек для візуалізації даних (MPAndroidChart) та розробки користувацького інтерфейсу (Android Jetpack).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів та технологій для контролю рівня споживання кофеїну.
2. Проектування архітектури та структури даних мобільного застосунку.
3. Програмна реалізація системи контролю споживання кофеїну.

4. Тестування розробленого програмного застосунку.

5. Перелік графічного матеріалу: *презентація*
   1. Порівняльний аналіз існуючих аналогів.
   2. Таблиця функціональних та нефункціональних вимог.
   3. Схема обраного стеку технологій.
   4. Діаграма варіантів використання (Use Case).
   5. Діаграма класів (Class Diagram).
   6. Схема архітектури застосунку (MVVM).
   7. Схема структури бази даних.
   8. Екранні форми розробленого застосунку.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Формулювання вимог та вибір технологій	14.03–21.03.2025	
4	Проектування застосунку: архітектура, UML-діаграми, структура БД	22.03–04.04.2025	
5	Програмна реалізація застосунку	05.04–20.04.2025	
6	Тестування застосунку	21.04–28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Роман ЧОРНИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЩЕРБИНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 48 стор., 5 табл., 18 рис., 24 джерел.

Мета роботи – спрощення процесу контролю за споживанням кофеїну для користувачів за рахунок розробки мобільного застосунку.

Об'єкт дослідження – процес контролю споживання кофеїновмісних продуктів за допомогою мобільних технологій.

Предмет дослідження – програмне забезпечення для контролю споживання кофеїну, зокрема його архітектура, функціональні можливості та методи реалізації.

Короткий зміст роботи: В роботі проаналізовано предметну галузь контролю споживання кофеїну, досліджено існуючі мобільні застосунки-аналоги. Розроблено архітектуру застосунку за шаблоном MVVM та програмно реалізовані ключові функціональні можливості: додавання та редагування записів, відстеження денного ліміту, візуалізація статистики, система налаштувань та сповіщень. В роботі використано мова програмування Kotlin, архітектурний шаблон MVVM, бібліотеки Android Jetpack (Room, LiveData, ViewModel), бібліотека для візуалізації даних MPAndroidChart, середовище розробки Android Studio, система контролю версій Git.

Сферою використання застосунку є надання інструменту для користувачів, що прагнуть свідомо контролювати щоденне споживання кофеїну з метою дотримання медичних рекомендацій, покращення якості сну та підтримки здорового способу життя.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, ANDROID, KOTLIN, MVVM, ROOM, КОНТРОЛЬ КОФЕЇНУ, БАЗА ДАНИХ, СТАТИСТИКА, UI, MPANDROIDCHART, ANDROID JETPACK.

ЗМІСТ

ВСТУП

Актуальність теми. Споживання кофеїну є невід'ємною частиною повсякденного життя мільйонів людей у всьому світі. Кава, чай, енергетичні напої та інші продукти використовуються для підвищення концентрації та бадьорості. Водночас, надмірне або неконтрольоване споживання кофеїну пов'язане з низкою ризиків для здоров'я, включаючи порушення сну, підвищення рівня тривожності та негативний вплив на серцево-судинну систему. Для певних груп населення, зокрема вагітних жінок та людей з медичними протипоказаннями, контроль добової норми кофеїну є медичною необхідністю.

Основна складність для користувачів полягає у відсутності простого механізму для відстеження загальної кількості спожитого кофеїну з різних джерел, а також у варіативності його вмісту в напоях. Існуючі мобільні застосунки часто пропонують або обмежений функціонал, що не задовольняє потреби в аналітиці, або є перевантаженими та не забезпечують належного рівня кастомізації. В таких умовах виникає потреба у розробці спеціалізованого програмного забезпечення, здатного автоматизувати та систематизувати процес контролю споживання кофеїну, надаючи користувачам точний та доступний інструмент для моніторингу своїх звичок.

Мета роботи. Метою даної кваліфікаційної роботи є розробка мобільного застосунку для контролю споживання кофеїну, який надає користувачам інструментарій для фіксації випитих напоїв, відстеження денної норми, візуалізації статистики та отримання своєчасних сповіщень. Система має на меті забезпечити локальне збереження даних, налаштування індивідуальних лімітів та можливість аналізувати динаміку споживання за обрані періоди.

Об'єкт дослідження. Процес контролю добового споживання кофеїну користувачами за допомогою мобільних технологій. Аналіз методів збору, збереження, обробки та представлення даних для підвищення ефективності моніторингу та формування усвідомлених звичок у користувачів.

Предметом дослідження є методи та технології розробки нативного мобільного застосунку для платформи Android, включаючи проєктування архітектури за шаблоном MVVM, реалізацію логіки взаємодії з локальною базою даних за допомогою Room, розробку користувацького інтерфейсу з використанням компонентів Android Jetpack та інтеграцію бібліотек для візуалізації даних.

Розробка такої системи дозволить не лише поглибити знання у сфері сучасної мобільної розробки на мові Kotlin, але й зробити внесок у створення програмного продукту, що сприяє підвищенню обізнаності користувачів про власне здоров'я. Очікується, що результатом роботи стане функціональний та стабільний мобільний застосунок, готовий до використання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ЗАСОБІВ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1 Аналіз предметної галузі контролю споживання кофеїну

Кофеїн є найпоширенішою психоактивною речовиною у світі, яка природним чином міститься в каві, чаї, какао-бобах, а також додається до складу енергетичних напоїв, безалкогольних напоїв та деяких лікарських засобів. Його популярність зумовлена стимулюючим впливом на центральну нервову систему, що проявляється у підвищенні рівня бадьорості, покращенні концентрації уваги та тимчасовому зниженні втоми.

Фізіологічний вплив кофеїну полягає в блокуванні аденозинових рецепторів у мозку. Аденозин – це нейромедіатор, який накопичується протягом дня і викликає відчуття сонливості. Блокуючи його дію, кофеїн підтримує стан збудження. Помірне споживання, за даними численних досліджень, може бути пов'язане з певними позитивними ефектами. Однак, перевищення індивідуальної норми або неконтрольоване споживання може призвести до низки негативних наслідків:

Порушення сну: Вживання кофеїну, особливо в другій половині дня, може значно ускладнити процес засинання, скоротити тривалість фази глибокого сну та погіршити загальну якість нічного відпочинку.

Підвищена тривожність: У високих дозах кофеїн може викликати або посилювати симптоми тривожних розладів, такі як нервозність, прискорене серцебиття та відчуття неспокою.

Вплив на серцево-судинну систему: Хоча для більшості здорових людей цей вплив є тимчасовим, особам із гіпертонією або іншими серцевими захворюваннями рекомендується обмежувати споживання.

Згідно з рекомендаціями Управління з санітарного нагляду за якістю харчових продуктів та медикаментів США (FDA) та Європейського агентства з

безпеки харчових продуктів (EFSA), безпечною добовою дозою для більшості здорових дорослих вважається до 400 мг кофеїну. Для вагітних жінок та жінок, що годують груддю, ця норма зазвичай знижується до 200 мг на добу.

На основі цих факторів можна виділити кілька ключових груп цільової аудиторії для програмних засобів контролю кофеїну:

1. Офісні працівники та студенти: Використовують кофеїн для підтримки розумової працездатності протягом дня. Для них важливо не перевищувати добову норму, щоб уникнути негативного впливу на сон та підвищення рівня стресу.
2. Спортсмени: Вживають кофеїн як ергогенну добавку для підвищення витривалості та продуктивності. Для них ключовим є точне дозування та час прийому відносно тренувань.
3. Особи з медичними показаннями: Пацієнти з серцево-судинними захворюваннями, тривожними розладами чи проблемами шлунково-кишкового тракту, яким лікар рекомендував суворий контроль або повну відмову від кофеїну.
4. Вагітні жінки та матері, що годують груддю: Потребують ретельного дотримання зниженої норми споживання для безпеки дитини.
5. Особи, що прагнуть до здорового способу життя: Контролюють споживання кофеїну як частину загальної стратегії покращення самопочуття, сну та зменшення залежності від стимуляторів.

Основною проблемою для всіх цих груп є складність ручного відстеження. Вміст кофеїну значно варіюється не тільки між різними типами напоїв, але й в межах одного типу залежно від способу приготування, об'єму порції та виробника. Це робить процес підрахунку неточним та трудомістким. Відсутність зручного інструменту також призводить до нерегулярного внесення даних та унеможливлює об'єктивний аналіз власних звичок.

Таким чином, актуальність розробки спеціалізованого мобільного застосунку полягає в необхідності надати користувачам інструмент для

автоматизованого, точного та систематичного контролю споживання кофеїну, що сприятиме формуванню усвідомлених та здорових звичок.

1.1.1 Фізіологічний вплив кофеїну та медичні аспекти контролю

Фізіологічний вплив кофеїну на організм людини є складним і багатогранним. Його основний механізм дії полягає в блокуванні аденозинових рецепторів у центральній нервовій системі. Аденозин – це нейромедіатор, що накопичується в мозку протягом дня і, зв'язуючись зі своїми рецепторами, викликає відчуття втоми та сонливості. Кофеїн, маючи схожу з аденозином молекулярну структуру, займає його місце на рецепторах, тим самим перешкоджаючи його дії та підтримуючи стан бадьорості.

Помірне споживання кофеїну асоціюється з низкою позитивних ефектів, таких як покращення когнітивних функцій, підвищення концентрації уваги, швидкості реакції та короткочасної пам'яті. Однак, неконтрольоване або надмірне споживання може призвести до значних негативних наслідків для здоров'я:

Порушення сну: Це найпоширеніший негативний ефект. Кофеїн може не тільки ускладнювати процес засинання, але й скорочувати тривалість фази глибокого сну, яка є критично важливою для фізичного та психічного відновлення організму. Період напіввиведення кофеїну становить в середньому 5-6 годин, тому вживання його в другій половині дня може суттєво погіршити якість нічного відпочинку.

Підвищена тривожність: У високих дозах (зазвичай понад 400 мг) кофеїн може викликати або посилювати симптоми тривожних розладів, такі як нервозність, дратівливість, прискорене серцебиття та навіть панічні атаки.

Вплив на серцево-судинну систему: Кофеїн викликає тимчасове підвищення артеріального тиску та частоти серцевих скорочень. Хоча для більшості здорових людей цей ефект не є небезпечним, особам із гіпертонією або іншими серцевими захворюваннями рекомендується суворо обмежувати споживання.

Згідно з рекомендаціями провідних світових організацій, таких як Управління з санітарного нагляду за якістю харчових продуктів та медикаментів США (FDA) та Європейське агентство з безпеки харчових продуктів (EFSA), безпечною добовою дозою для більшості здорових дорослих вважається до 400 мг кофеїну. Для вагітних жінок та жінок, що годують груддю, ця норма зазвичай знижується до 200 мг на добу, щоб уникнути потенційних ризиків для дитини.

1.1.2 Психологічні та поведінкові аспекти відстеження звичок

Розробка застосунку для контролю споживання кофеїну є не лише технічною, а й поведінковою задачею. Ефективність таких інструментів значною мірою залежить від їхньої здатності впливати на звички користувача. Ключовим психологічним механізмом, що лежить в основі цього процесу, є самомоніторинг.

Самомоніторинг – це процес систематичного спостереження та запису власної поведінки. Дослідження в галузі психології здоров'я показують, що сам акт відстеження звички значно підвищує усвідомленість та є першим кроком до її зміни. Коли користувач регулярно фіксує кожен випитий напій, він отримує об'єктивний зворотний зв'язок про реальний обсяг споживання, який часто виявляється вищим, ніж суб'єктивні уявлення.

Мобільний застосунок у цьому контексті виступає інструментом для створення ефективного циклу зворотного зв'язку:

1. Тригер: Подія споживання напою.
2. Дія: Користувач відкриває застосунок і вносить дані.
3. Винагорода/Зворотний зв'язок: Застосунок миттєво оновлює дані на головному екрані, показуючи прогрес відносно денного ліміту. Візуалізація (наприклад, заповнення прогрес-бару) слугує наочним підтвердженням дії.
4. Аналіз: Перегляд історії та статистики дозволяє користувачу аналізувати свою поведінку в динаміці, виявляти патерни (наприклад, підвищене споживання у стресові дні) та приймати більш обґрунтовані рішення в майбутньому.

Таким чином, розроблюваний продукт має не просто фіксувати дані, а й сприяти формуванню усвідомлених звичок через надання точного, своєчасного та легкодоступного зворотного зв'язку.

1.2 Аналіз існуючих програмних аналогів

З метою визначення функціональних прогалин на ринку та формування вимог до власного продукту було проведено аналіз чотирьох мобільних застосунків, призначених для контролю споживання кофеїну.

1. Caffeine Tracker

Опис: Застосунок, функціонал якого сфокусований на фіксації спожитого кофеїну. Головний екран надає користувачу можливість додати напій з наявного переліку. Екранні форми додатку «Caffeine Tracker» наведено на рисунку 1.1

Реалізований функціонал:

- Додавання записів про споживання з передвстановленого списку.
- Відображення сумарної кількості спожитого кофеїну за день.

Відсутній функціонал:

- Візуалізація даних у вигляді графіків чи діаграм.
- Можливість додавання власних типів напоїв.
- Механізм встановлення персонального денного ліміту.

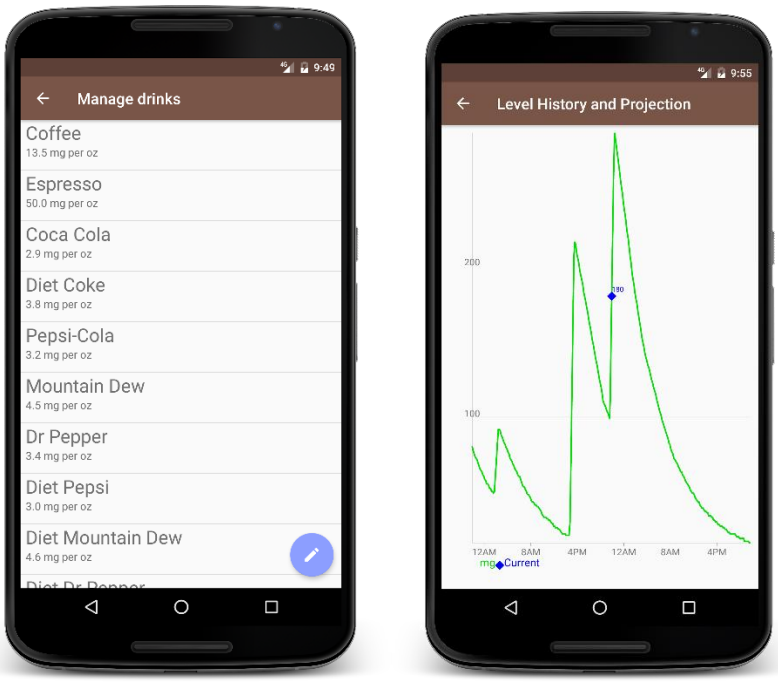


Рис. 1.1 Екранні форми застосунку «Caffeine Tracker»

2. Coffee Plus

Опис: Програмний продукт, що розширює базовий функціонал відстеження, додаючи інструменти для аналізу даних. Екранні форми додатку «Coffe Plus» наведено на рисунку 1.2

Реалізований функціонал:

- Візуалізація даних споживання у вигляді стовпчастих діаграм за тиждень та місяць.
- Можливість додавання власних напоїв з вказанням вмісту кофеїну.
- Налаштування денного ліміту споживання.

Особливості:

- Інтерфейс містить велику кількість елементів керування.
- Безкоштовна версія інтегрує рекламні блоки.

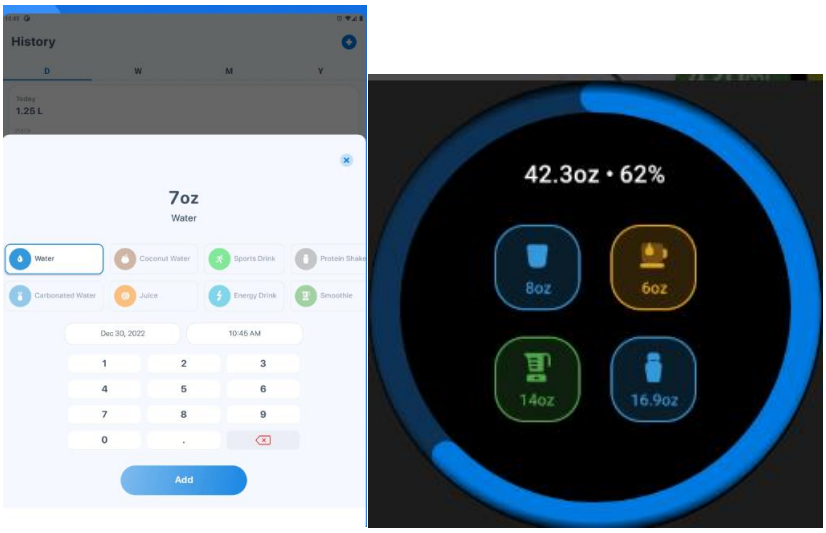


Рис. 1.2 Екранні форми застосунку «Coffee Plus»

3. Caffeine Clock

Опис: Особливістю даного застосунку є наявність прогностичної функції, яка розраховує час зниження концентрації кофеїну в організмі. Екранні форми додатку «Caffeine Clock» наведено на рисунку 1.3

Реалізований функціонал:

- Прогнозування часу, коли ефект від спожитого кофеїну зменшиться.

Відсутній функціонал:

- Ведення та перегляд історії споживання за минулі дні.
- Інструменти для аналізу даних за періоди.

Особливості:

- Графічний інтерфейс виконаний у стилі попередніх версій мобільних ОС.
- Обмежений перелік доступних для додавання напоїв.

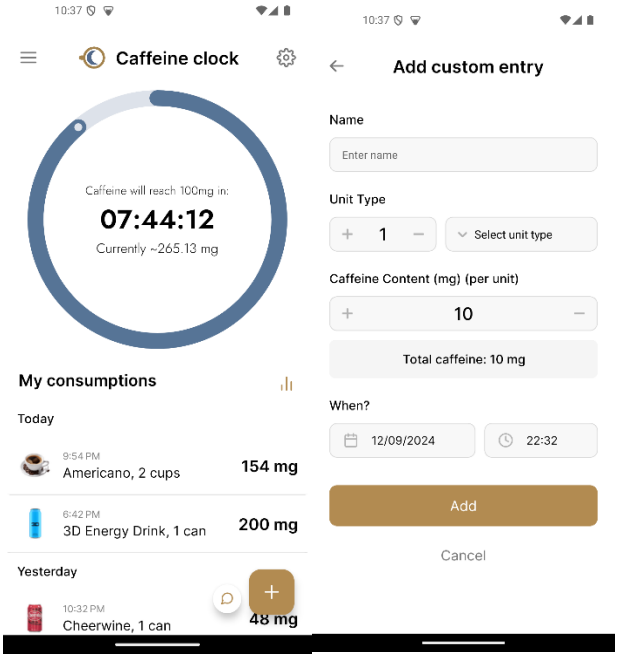


Рис. 1.3 Екранні форми застосунку «Caffeine Clock»

4. CaffeInMe

Опис: Застосунок для відстеження споживання кофеїну з можливістю встановлення добової норми. Екранні форми додатку «CaffeInMe» наведено на рисунку 1.4

Реалізований функціонал:

- Встановлення та відстеження добового ліміту споживання.
- Додавання записів про споживання.

Відсутній функціонал:

- Перегляд детальної історії споживання за минулі періоди.
- Інструменти для керування збереженими даними (наприклад, видалення записів за обраний період).

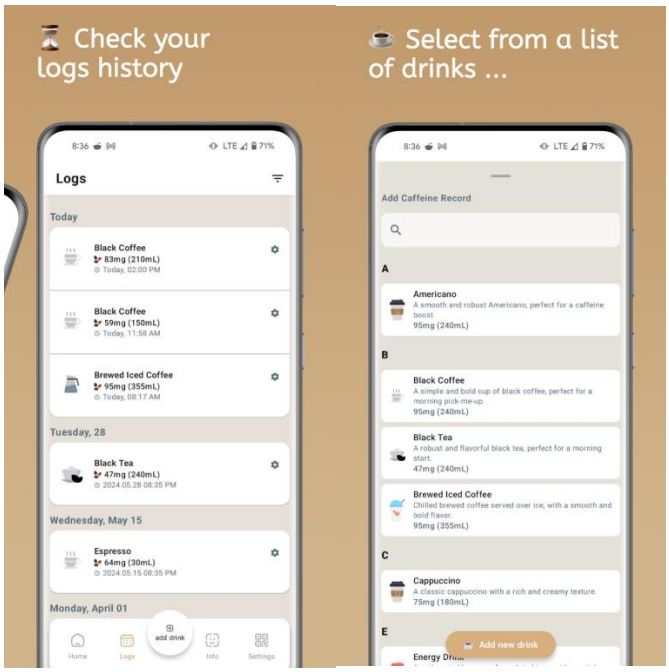


Рис. 1.4 Екранні форми застосунку «CaffeInMe»

Порівняльний аналіз та висновки

Для наочного порівняння характеристик існуючих додатків було складено зведену таблицю 1.1.

Таблиця 1.1

Порівняльна характеристика додатків для контролю кофеїну

Характеристика	CaffeInMe	Coffe Plus	Caffeine Tracker	Caffeine Clock	LatteNo
Відстеження споживання кофеїну	+	+	+	+	+
Візуалізація даних (графіки)	-	+	-	-	+
Налаштування персональних цілей	+	+	-	-	+
Додавання власних напоїв	-	+	-	+	+
Перегляд історії споживання	-	+	+	-	+
Очищення даних за період	-	-	-	-	+

Аналіз існуючих рішень показує, що на ринку відсутній застосунок, який би інтегрував повний набір інструментів для контролю споживання кофеїну. Це

обґрунтовує доцільність розробки нового продукту "LatteNo", який має на меті заповнити виявлені функціональні прогалини.

1.3 Огляд технологій та методів розробки мобільних застосунків

Вибір стеку технологій є ключовим етапом, що визначає продуктивність, стабільність та можливості подальшого розвитку програмного продукту.

Нативна розробка. Для даного проекту було обрано підхід нативної розробки для платформи Android. На відміну від кросплатформних рішень (React Native, Flutter), нативна розробка забезпечує:

Максимальну продуктивність: Прямий доступ до всіх API операційної системи та апаратних ресурсів пристрою.

Оптимальний користувацький досвід (UX): Можливість створювати інтерфейси, що повністю відповідають гайдлайнам Material Design.

Високу стабільність: Менша залежність від сторонніх фреймворків, що знижує ризик виникнення специфічних помилок.

Мова програмування: Kotlin є офіційно рекомендованою мовою для Android-розробки з 2019 року. Її вибір обґрунтований наступними технічними перевагами:

1. Лаконічність та читабельність коду. Код, написаний на Kotlin, є значно коротшим і виразнішим у порівнянні з його аналогом на Java. Це дозволяє писати менше шаблонного коду, що не тільки прискорює процес розробки, але й робить програму легшою для розуміння та подальшої підтримки.
2. Підвищена надійність та менша кількість збоїв. Kotlin має вбудовані механізми, які допомагають уникнути однієї з найчастіших помилок у програмуванні — звернення до неіснуючого об'єкта (NullPointerException). Це відбувається ще на етапі написання коду, що в результаті робить фінальний застосунок більш стабільним та менш схильним до раптових збоїв під час роботи.

3. Повна взаємосумісність з Java. Платформа Android історично базується на Java. Kotlin сумісна з Java, що дозволяє розробникам використовувати величезну кількість існуючих бібліотек, інструментів та фреймворків, написаних на Java, без будь-яких обмежень. Це значно розширює можливості та прискорює розробку.
4. Офіційна підтримка Google та активна спільнота. Google активно розвиває та підтримує Kotlin як пріоритетну мову для Android. Це гарантує якісну документацію, постійні оновлення інструментів (в Android Studio) та довгострокову актуальність технології. Крім того, велика та активна спільнота розробників забезпечує швидкий пошук рішень для будь-яких технічних завдань.

Архітектурний шаблон: MVVM (Model-View-ViewModel) для структурування коду було обрано архітектурний шаблон MVVM, рекомендований Google. Він розділяє застосунок на три основні компоненти:

Model: Шар даних. Відповідає за бізнес-логіку та доступ до даних (локальна база даних, мережа).

View: Шар представлення. Відповідає за відображення даних на екрані (Activity, Fragment). Він не містить жодної логіки, окрім взаємодії з користувачем.

ViewModel: Посередник, що готує дані з Model для відображення у View. Він не має посилань на View, що робить його незалежним від життєвого циклу UI-компонентів та спрощує тестування.

Для реалізації цього шаблону використовуються компоненти з Android Jetpack, такі як ViewModel, LiveData. LiveData є спостережуваним (observable) тримачем даних, який автоматично сповіщає View про зміни, забезпечуючи реактивність інтерфейсу.

Локальне сховище даних: Room для збереження даних на пристрої обрано бібліотеку Room з Android Jetpack. Вона надає абстракційний шар над вбудованою базою даних SQLite і має наступні переваги:

- Перевірка SQL-запитів на етапі компіляції.
- Мінімізація шаблонного коду.

- Інтеграція з LiveData для автоматичного оновлення UI при зміні даних у базі.

Вибір даного стеку технологій (нативна розробка, Kotlin, MVVM, Room) дозволяє створити сучасний, продуктивний та легкий у підтримці мобільний застосунок.

1.4 Висновки до першого розділу

У першому розділі кваліфікаційної роботи було проведено комплексний аналіз, що став основою для подальшого проєктування та розробки мобільного застосунку.

Було досліджено предметну галузь, пов'язану з контролем споживання кофеїну. Визначено, що актуальність задачі зумовлена медичними рекомендаціями, проблемами зі сном та загальним трендом на здоровий спосіб життя. Сформовано портрети ключових груп цільової аудиторії та виявлено основні проблеми ручного відстеження, що обґрунтовує потребу в автоматизованому програмному рішенні.

Проведено порівняльний аналіз чотирьох існуючих мобільних застосунків-аналогів. Встановлено, що на ринку відсутній продукт, який би поєднував у собі повний набір необхідних функцій, таких як візуалізація статистики та гнучке керування даними. Це дозволило визначити функціональні прогалини та сформувати конкурентні переваги майбутнього застосунку.

На основі аналізу підходів до розробки було сформовано стек технологій. Прийнято рішення про використання нативної розробки на мові Kotlin з застосуванням архітектурного шаблону MVVM та компонентів Android Jetpack (Room, LiveData, ViewModel), що забезпечить створення продуктивного, стабільного та масштабованого програмного продукту.

Результати, отримані в даному розділі, слугують фундаментом для формулювання технічних вимог, проєктування архітектури та подальшої програмної реалізації застосунку.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Формулювання функціональних та нефункціональних вимог

На основі аналізу предметної галузі та існуючих програмних рішень, проведеного в попередньому розділі, на даному етапі виконується проєктування майбутнього мобільного застосунку. Цей процес включає визначення детальних технічних вимог, вибір та опис архітектури, а також проєктування структури локальної бази даних.

Формулювання вимог є фундаментальним етапом проєктування, що визначає, яким функціоналом має володіти система та яким критеріям якості вона повинна відповідати.

Функціональні вимоги описують конкретні дії, які система повинна виконувати. На основі проведеного аналізу було сформовано наступний перелік функціональних вимог:

1. Керування записами про споживання:

Система повинна надавати користувачу можливість додавати нові записи про спожитий кофеїн.

Процес додавання має включати вибір напою з передвстановленого списку або введення даних про власний напій (назва, кількість кофеїну).

Система повинна дозволяти редагувати існуючі записи (змінювати назву, кількість кофеїну).

Система повинна забезпечувати можливість видалення окремих записів з історії.

2. Відстеження денного ліміту:

Застосунок повинен відображати поточну сумарну кількість спожитого кофеїну за добу.

Користувач повинен мати можливість встановлювати та змінювати свій індивідуальний денний ліміт.

Інтерфейс має візуально відображати прогрес споживання відносно встановленого ліміту.

3. Аналітика та історія:

Система повинна зберігати та відображати історію споживання за минулі дні.

Застосунок має надавати статистику споживання у вигляді графіка за обраний період (останні 7 днів).

Користувач повинен мати можливість переглядати записи за будь-яку обрану дату в минулому.

4. Керування даними:

Система повинна надавати функцію видалення всіх записів за обраний користувачем період часу.

5. Система сповіщень:

Застосунок повинен надсилати сповіщення при перевищенні встановленого денного ліміту.

Система має реалізовувати функцію щоденних нагадувань у встановлений користувачем час.

Нефункціональні вимоги визначають атрибути якості системи та обмеження, в рамках яких вона має функціонувати.

1. Продуктивність: Застосунок повинен забезпечувати швидкий відгук інтерфейсу (менше 200 мс на стандартні операції) та не споживати надмірну кількість ресурсів пристрою (CPU, пам'ять).
2. Надійність: Система повинна працювати стабільно без збоїв та втрати даних при стандартних сценаріях використання.
3. Сумісність: Застосунок має коректно функціонувати на пристроях під управлінням ОС Android версії 7.0 (API 24) та вище.
4. Безпека та приватність: Усі дані користувача (історія споживання, налаштування) повинні зберігатися локально на пристрої, без передачі на зовнішні сервери.

5. Простота встановлення: Інсталяція та перший запуск застосунку не повинні вимагати від користувача додаткових складних налаштувань.

2.2 Проєктування архітектури застосунку

Для забезпечення гнучкості, масштабованості та тестованості коду, застосунок проектується на основі архітектурного шаблону MVVM (Model-View-ViewModel).

Загальну структуру архітектури MVVM, адаптовану для даного проекту, показано на рисунку 2.1.

Архітектура складається з трьох основних шарів:

View Layer (Шар представлення): Включає в себе UI контролери (Activity та Fragments). Його єдина відповідальність — відображення даних, отриманих від ViewModel, та передача дій користувача (натискання кнопок, введення тексту) у ViewModel.

ViewModel Layer (Шар моделі подання): Клас ViewModel виступає посередником. Він отримує запити від View, обробляє їх, взаємодіє з шаром даних (Model) і надає готові для відображення дані назад у View через LiveData. Важливою особливістю є те, що ViewModel не має прямих посилань на View, що робить його незалежним від життєвого циклу UI.

Model Layer (Шар даних): В даному проекті реалізований через патерн "Репозиторій" (Repository), який є єдиним джерелом даних для ViewModel. Репозиторій керує доступом до локальної бази даних (Room) і, за потреби, може бути розширений для роботи з віддаленими джерелами (API).

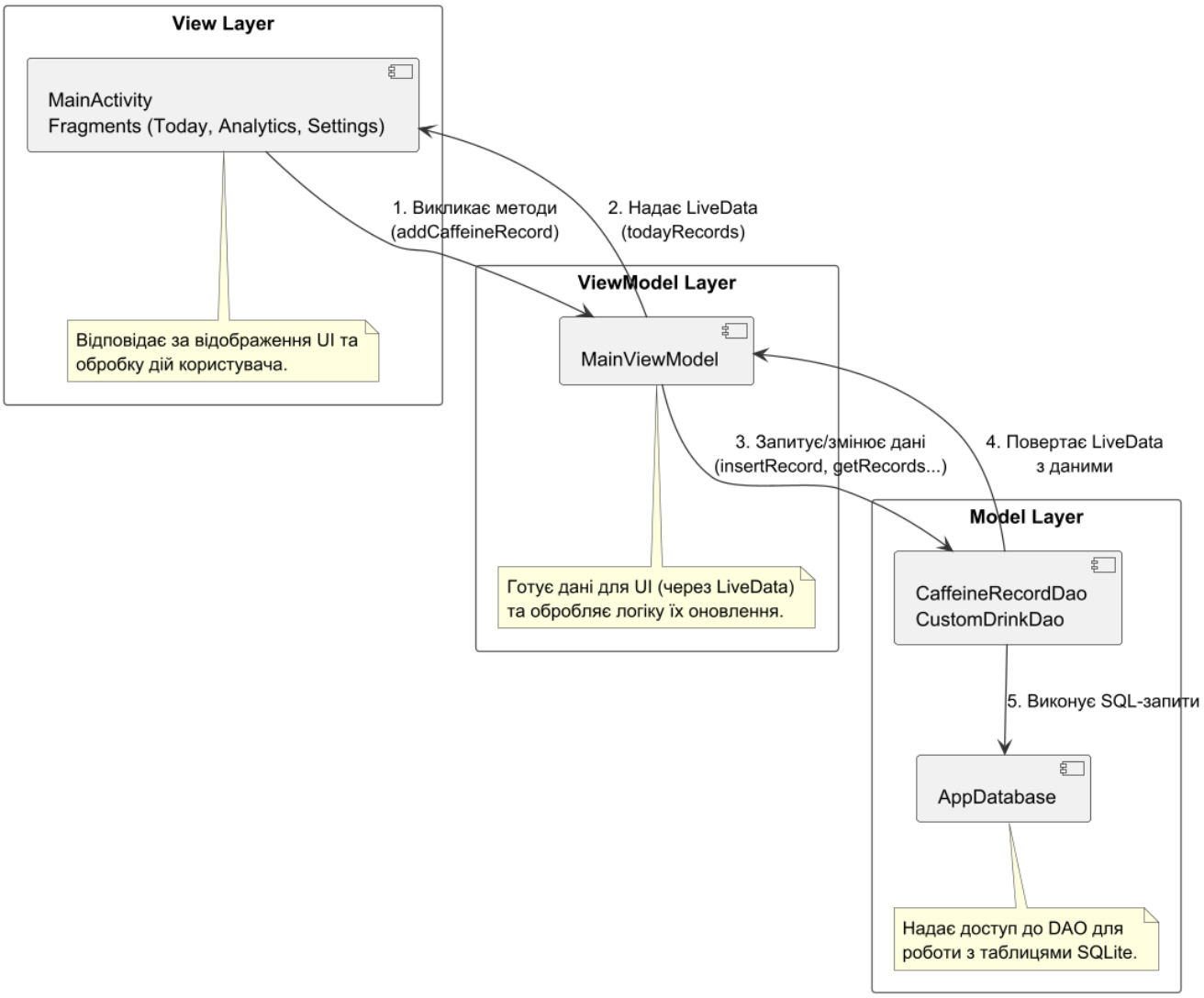


Рис. 2.1. Архітектурна схема застосунку за шаблоном MVVM

Діаграма варіантів використання (Use Case Diagram) Для моделювання взаємодії користувача з системою розроблено діаграму варіантів використання. Вона візуалізує основні функції, які застосунок надає актору "Користувач", та логічні зв'язки між ними.

На діаграмі показано, що користувач може виконувати такі основні дії, як "Додати споживання кофеїну", "Переглянути історію", "Налаштувати ліміт" та інші. Деякі варіанти використання, як-от "Обрати напій зі списку", є включеними (<<include>>), а інші, наприклад "Редагувати запис", розширюють (<<extend>>) базову функціональність (рисунок 2.2).

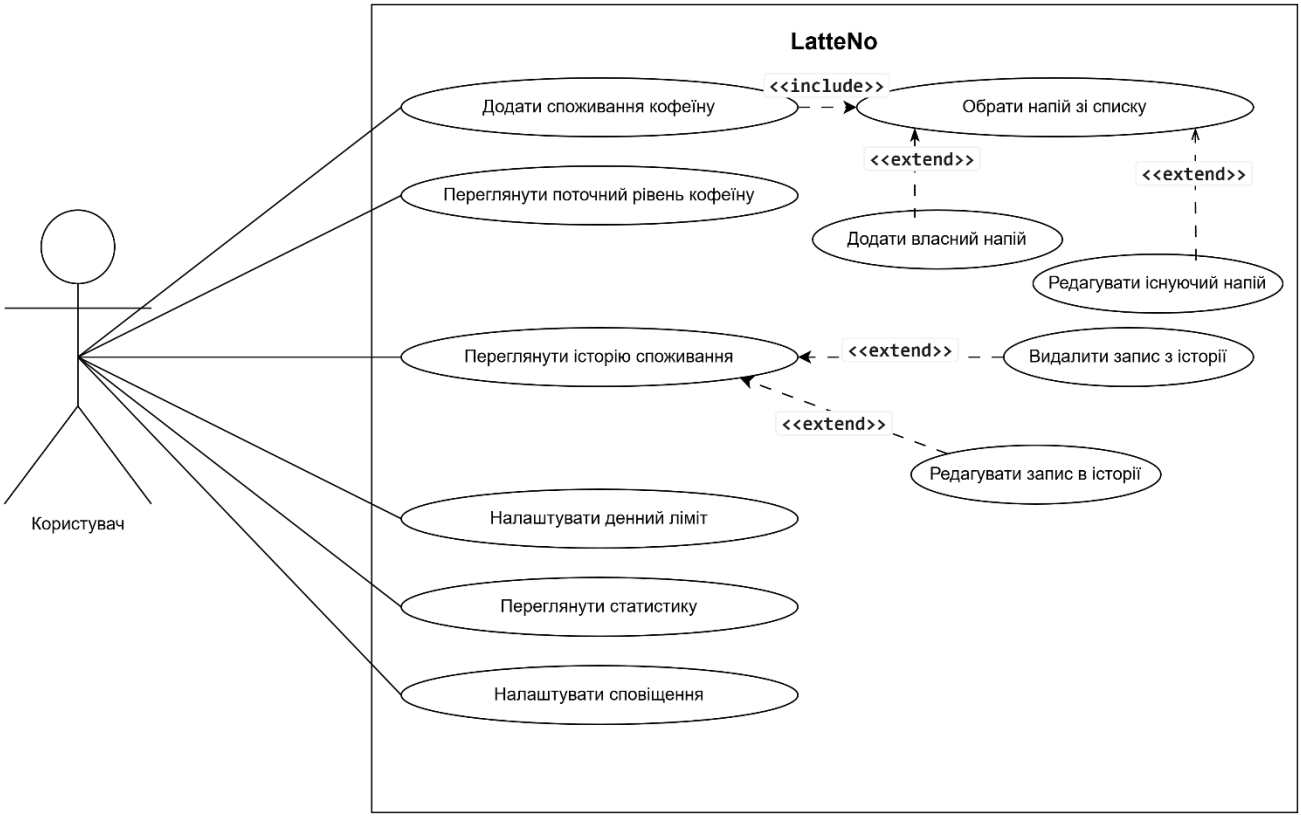


Рис. 2.2. Діаграма варіантів використання

Діаграма класів. Для деталізації статичної структури системи та уникнення перевантаження однієї схеми, проєктування класів представлено у вигляді двох окремих діаграм. Перша ілюструє взаємодію між шарами UI Layer та ViewModel, а друга — між ViewModel та Data Layer.

Діаграма, показана на рисунку 2.3, демонструє як компоненти користувацького інтерфейсу взаємодіють з MainViewModel.

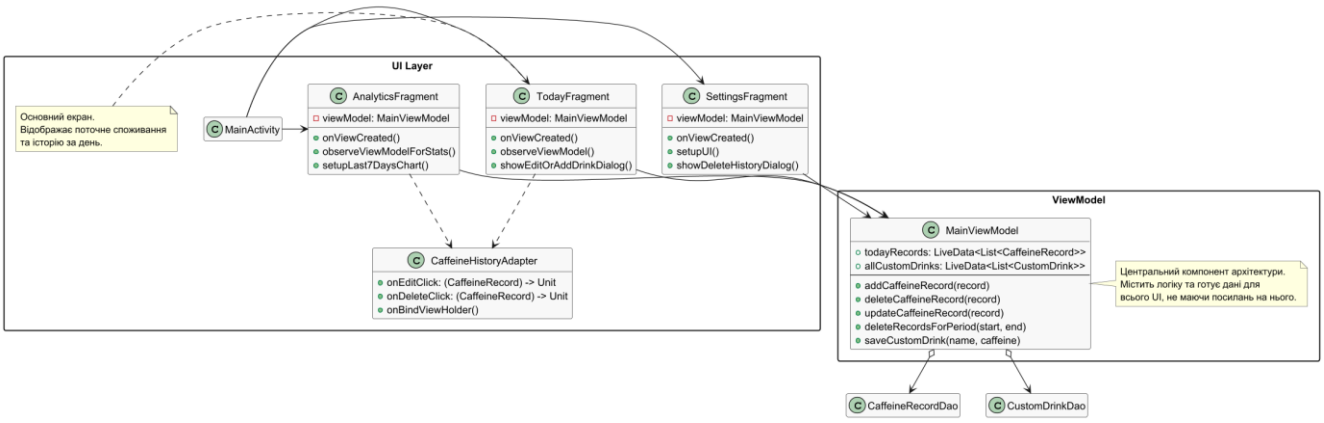


Рис. 2.3 Діаграма класів взаємодії UI Layer та ViewModel

Як показано на рисунку 2.3, класи фрагментів (TodayFragment, AnalyticsFragment, SettingsFragment) містять посилання на екземпляр MainViewModel. Вони викликають його публічні методи (наприклад, addCaffeineRecord()) у відповідь на дії користувача. Водночас, вони підписані на LiveData поля (todayRecords, allCustomDrinks) з MainViewModel для автоматичного оновлення інтерфейсу при зміні даних. CaffeineHistoryAdapter використовується фрагментами для відображення списків.

Діаграма класів взаємодії ViewModel та Data Layer (рисунок 2.4) деталізує взаємодію MainViewModel з компонентами шару даних.

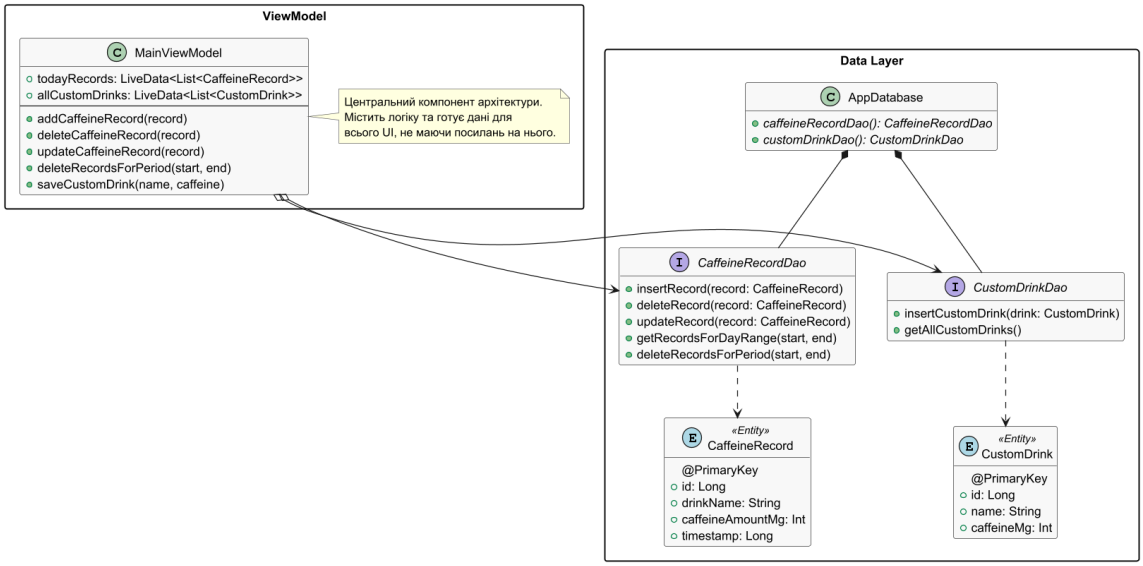


Рис. 2.4 Діаграма класів взаємодії ViewModel та Data Layer

Як показано на рисунку 2.4 видно, що MainViewModel не взаємодіє з базою даних напряму. Замість цього він використовує інтерфейси доступу до даних (CaffeineRecordDao, CustomDrinkDao) як контракт для виконання операцій з даними. Ці DAO, у свою чергу, працюють із сутностями (CaffeineRecord, CustomDrink), які представляють собою моделі таблиць у базі даних. Такий підхід забезпечує слабку зв'язаність компонентів та робить систему більш гнучкою до змін.

2.2.1 Проєктування користувацького досвіду

Проектування користувацького досвіду та інтерфейсу є критичним етапом, що визначає, наскільки ефективно та приємно користувач зможе взаємодіяти з функціоналом застосунку. Головною метою на цьому етапі було створення інтуїтивно зрозумілого та мінімалістичного інтерфейсу, що дозволяє виконувати ключові дії з мінімальною кількістю кроків.

Основні сценарії взаємодії (User Flows): Ключовий сценарій "Додавання запису про споживання" було спроектовано наступним чином:

1. На головному екрані користувач натискає кнопку «+» (Floating Action Button).
2. Система відкриває діалогове вікно додавання запису.
3. Користувач обирає напій з випадаючого списку або вводить назву вручну.
4. Якщо напій обрано зі списку, поле "Кількість кофеїну" заповнюється автоматично.
5. Користувач підтверджує дію, натискаючи кнопку "Додати".
6. Діалогове вікно закривається, головний екран оновлюється, відображаючи новий запис та оновлений сумарний показник споживання.

Принципи дизайну та візуальне оформлення: Для забезпечення консистентності та відповідності стандартам платформи Android було вирішено дотримуватися Material Design 3. Це забезпечує впізнаваність елементів керування для користувачів та сучасний вигляд застосунку. Кольорова схема була обрана з урахуванням психології сприйняття: основний акцентний колір (фіолетовий) використовується для ключових елементів, тоді як решта інтерфейсу виконана в нейтральних сірих та білих тонах, щоб не відволікати користувача від основної інформації – даних про споживання.

2.3 Проєктування структури бази даних

Дизайн Для локального збереження даних використовується бібліотека Room, яка працює поверх SQLite. База даних буде містити дві основні таблиці:

для зберігання записів про споживання та для зберігання кастомних напоїв, створених користувачем.

Сутність CaffeineRecord

Ця сутність представляє таблицю caffeine_records, в якій зберігається кожен окремий запис про спожитий напій. Структура сутності наведена в таблиці 2.1.

Таблиця 2.1

Структура сутності CaffeineRecord

Назва поля	Тип даних	Опис	Обмеження
id	Long	Унікальний ідентифікатор запису	Первинний ключ, автоінкремент
drinkName	String	Назва спожитого напою	-
caffeineAmountMg	Int	Кількість кофеїну в напої (в мг)	-
timestamp	Long	Час створення запису (в мілісекундах)	-

Сутність CustomDrink

Ця сутність представляє таблицю custom_drinks, де зберігаються напої, створені власноруч користувачем для швидкого доступу в майбутньому. Структура сутності наведена в таблиці 2.2.

Таблиця 2.2

Структура сутності CustomDrink

Назва поля	Тип даних	Опис	Обмеження
id	Long	Унікальний ідентифікатор напою	Первинний ключ, автоінкремент
name	String	Назва кастомного напою	Унікальне значення
caffeineMg	Int	Кількість кофеїну в напої (в мг)	-

Інтерфейси доступу до даних (DAO). Для взаємодії з таблицями створюються DAO (Data Access Object) — інтерфейси, що містять методи для виконання SQL-запитів.

- CaffeineRecordDao буде містити методи для вставки, оновлення, видалення та отримання записів CaffeineRecord.
- CustomDrinkDao буде містити методи для додавання та отримання кастомних напоїв CustomDrink.

Як видно на схемі (рисунок 2.5), між таблицями caffeine_records та custom_drinks відсутній прямий зв'язок на рівні зовнішніх ключів (Foreign Key). Цей зв'язок є логічним і реалізується на рівні застосунку: при додаванні нового запису в caffeine_records, назва напою (drinkName) може бути або взята з таблиці custom_drinks, або введена користувачем вручну. Такий підхід забезпечує гнучкість, оскільки дозволяє фіксувати унікальні напої без обов'язкового збереження їх у списку кастомних.

Для взаємодії з цими таблицями в архітектурі застосунку передбачені інтерфейси доступу до даних (DAO): CaffeineRecordDao та CustomDrinkDao, які інкапсулюють всю логіку виконання SQL-запитів.

Рис. 2.5 Схема бази даних

2.3.2 Обґрунтування реляційної моделі та вибору типів даних

При проектуванні локальної бази даних було прийнято рішення використовувати дві окремі таблиці: caffeine_records для зберігання всіх

транзакцій споживання та custom_drinks для збереження напоїв, створених користувачем.

Важливим архітектурним рішенням є відсутність жорсткого зовнішнього ключа (Foreign Key) між цими таблицями. Такий підхід був обраний для забезпечення більшої гнучкості системи. Він дозволяє користувачу фіксувати унікальні або разові напої (наприклад, "Кава від бабусі"), не вимагаючи обов'язкового збереження цього напою в таблицю custom_drinks. Таким чином, таблиця кастомних напоїв слугує лише для шаблонів швидкого доступу, а не як жорсткий довідник.

Вибір типів даних для полів сутностей був обґрунтований наступним чином:

id: Long: Тип Long для первинного ключа є стандартом для баз даних Room/SQLite, оскільки він надає достатньо великий діапазон значень для унікальної ідентифікації записів протягом тривалого часу використання застосунку.

timestamp: Long: Для збереження часу створення запису також було обрано тип Long. Це дозволяє зберігати час у форматі Unix-часу.

caffeineAmountMg: Int: Кількість кофеїну зберігається як ціле число (Int), оскільки для даної задачі не потрібна дробова точність, а використання Int є більш ефективним з точки зору використання пам'яті у порівнянні з Double або Float.

2.4 Висновки до другого розділу

У другому розділі кваліфікаційної роботи було виконано етап проектування мобільного застосунку для контролю споживання кофеїну.

На основі результатів аналізу, проведеного в попередньому розділі, було сформульовано детальний перелік функціональних та нефункціональних вимог. Ці вимоги стали технічним базисом для подальшого проектування системи.

Було спроектовано архітектуру застосунку на основі шаблону MVVM. Для візуалізації архітектури та взаємодії користувача з системою розроблено діаграми UML, зокрема діаграму варіантів використання та діаграму класів, що визначає статичну структуру програмних компонентів.

Також було спроектовано структуру локальної бази даних. Визначено дві основні сутності – CaffeineRecord та CustomDrink, описано їхні атрибути та обмеження. Спроектовано інтерфейси доступу до даних (DAO) для взаємодії з базою даних.

Отримані в даному розділі проектні рішення, включаючи вимоги, архітектурні схеми та структуру бази даних, є основою для подальшої програмної реалізації застосунку, яка буде описана в наступному розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ КОНТРОЛЮ СПОЖИВАННЯ КОФЕЇНУ

На основі проектних рішень, розроблених у попередньому розділі, на даному етапі було виконано програмну реалізацію мобільного застосунку. Цей розділ описує використане середовище розробки, інструменти, а також детально розглядає реалізацію ключових функціональних модулів системи.

3.1 Опис середовища розробки та інструментів

Розробка мобільного застосунку "LatteNo" здійснювалася з використанням сучасного та стандартизованого стеку інструментів для платформи Android.

Середовище розробки (IDE): Android Studio - офіційна інтегрована середа розробки, що надає повний набір інструментів для написання коду, тестування та збирання проекту.

Мова програмування: Kotlin.

Система збирання: Gradle. Використовується для автоматизації процесу збирання, керування залежностями та налаштування конфігурацій проекту.

Система контролю версій: Git. Для відстеження змін у коді, роботи з гілками та забезпечення цілісності проекту використовується Git. Як віддалений репозиторій для зберігання коду використовується сервіс GitHub.

Емуляція та тестування: Для тестування застосунку використовувався вбудований в Android Studio емулятор Android Virtual Device (AVD) з образом системи Android 13 (API 33).

3.2 Реалізація основних функціональних можливостей

Основний функціонал застосунку включає роботу з базою даних для збереження записів та взаємодію з користувацьким інтерфейсом для їх відображення та редагування.

Реалізація шару даних (Data Layer). Шар даних реалізовано за допомогою бібліотеки Room. Було створено клас AppDatabase, що успадковується від RoomDatabase. Цей клас є центральною точкою доступу до бази даних та визначає сутності (CaffeineRecord, CustomDrink) і абстрактні методи для отримання DAO (Data Access Objects). Приклад реалізації цього класу показано на рисунку 3.1.

```
@Database(entities = [CaffeineRecord::class, CustomDrink::class], version = 2, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {

    abstract fun caffeineRecordDao(): CaffeineRecordDao
    abstract fun customDrinkDao(): CustomDrinkDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock: this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                     name: "caffeine_tracker_database"
                )
                    .fallbackToDestructiveMigration()
                    .build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Рис. 3.1. Фрагмент коду класу AppDatabase

Для кожної сутності створено відповідний DAO-інтерфейс, який описує всі можливі операції з даними. Як видно з рисунка 3.2, CaffeineRecordDao містить методи для виконання CRUD-операцій (Create, Read, Update, Delete). Важливо, що методи, які повертають дані для відображення (getRecordsForDayRange),

повертають об'єкт LiveData. Цей підхід дозволяє автоматично оновлювати UI при будь-якій зміні даних у таблиці.

```kotlin
@Dao
interface CaffeineRecordDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertRecord(record: CaffeineRecord)

    @Query("SELECT * FROM caffeine_records WHERE timestamp >= :startTime AND timestamp <= :endTime ORDER BY timestamp DESC")
    fun getRecordsForDayRange(startTime: Long, endTime: Long): LiveData<List<CaffeineRecord>>

    @Delete
    suspend fun deleteRecord(record: CaffeineRecord)

    @Update
    suspend fun updateRecord(record: CaffeineRecord)

    @Query("SELECT * FROM caffeine_records WHERE timestamp >= :startTime AND timestamp <= :endTime ORDER BY timestamp DESC")
    fun getRecordsForPeriod(startTime: Long, endTime: Long): LiveData<List<CaffeineRecord>>

    @Query("DELETE FROM caffeine_records WHERE timestamp >= :startTime AND timestamp <= :endTime")
    suspend fun deleteRecordsForPeriod(startTime: Long, endTime: Long)
}
```

Рис. 3.2. Фрагмент коду інтерфейсу CaffeineRecordDao

Реалізація ViewModel. Клас MainViewModel виступає як центральний компонент архітектури, що з'єднує логіку даних та інтерфейс. Він ініціалізує доступ до DAO та надає для UI LiveData об'єкти, на які підписуються фрагменти. Як показано на рисунку 3.3, всі операції запису чи видалення даних (наприклад, addCaffeineRecord) виконуються асинхронно в корутинах за допомогою viewModelScope.launch, що запобігає блокуванню головного потоку.

```kotlin
    fun addCaffeineRecord(record: CaffeineRecord) {
        viewModelScope.launch {
            caffeineRecordDao.insertRecord(record)
        }
    }

    fun deleteCaffeineRecord(record: CaffeineRecord) {
        viewModelScope.launch {
            caffeineRecordDao.deleteRecord(record)
        }
    }

    fun saveCustomDrink(name: String, caffeineMg: Int) {
        viewModelScope.launch {
            val existingDrink = customDrinkDao.getCustomDrinkByName(name)
            if (existingDrink == null) { // Додаємо тільки якщо напою з такою назвою ще немає
                customDrinkDao.insertCustomDrink(CustomDrink(name = name, caffeineMg = caffeineMg))
            }
            // Якщо напій вже існує, можна вирішити оновлювати його або нічого не робити.
        }
    }

    // Функція для редагування запису історії
    fun updateCaffeineRecord(record: CaffeineRecord) {
        viewModelScope.launch {
            caffeineRecordDao.updateRecord(record)
        }
    }
```

Рис. 3.3. Фрагмент коду класу MainViewModel

Реалізація користувацького інтерфейсу (UI Layer). Інтерфейс реалізовано за допомогою фрагментів. Наприклад, TodayFragment відповідає за головний екран. У методі onViewCreated() він отримує екземпляр MainViewModel та підписується на оновлення todayRecords. Коли LiveData сповіщає про зміни, новий список передається в CaffeineHistoryAdapter, який оновлює RecyclerView.

Для додавання та редагування записів використовується діалогове вікно, яке викликається з TodayFragment. Воно містить поля для введення назви напою та кількості кофеїну. Після підтвердження користувачем, дані передаються у MainViewModel, який викликає відповідний метод DAO для збереження в базі даних.

3.2.1 Реалізація механізму взаємодії з базою даних через Room та DAO

Взаємодія з локальною базою даних SQLite інкапсульована в шарі доступу до даних (DAO). Бібліотека Room на етапі компіляції генерує конкретну реалізацію для кожного DAO-інтерфейсу, перетворюючи анотації на відповідні SQL-запити.

Наприклад, анотації в інтерфейсі CaffeineRecordDao, яка показана на рисунку 3.2, виконують наступні функції:

@Insert(onConflict = OnConflictStrategy.REPLACE): Вказує, що метод insertRecord() буде виконувати операцію вставки. Стратегія REPLACE означає, що якщо запис з таким самим первинним ключем вже існує, він буде замінений на новий.

@Query("SELECT * ..."): Ця анотація дозволяє виконувати довільні SQL-запити. Room перевіряє синтаксичну коректність цих запитів під час компіляції, що запобігає помилкам під час виконання програми.

suspend fun: Ключове слово suspend позначає, що функція є корутиною. Room має вбудовану підтримку корутин, що дозволяє виконувати операції з базою даних у фоновому потоці без ризику блокування основного потоку UI.

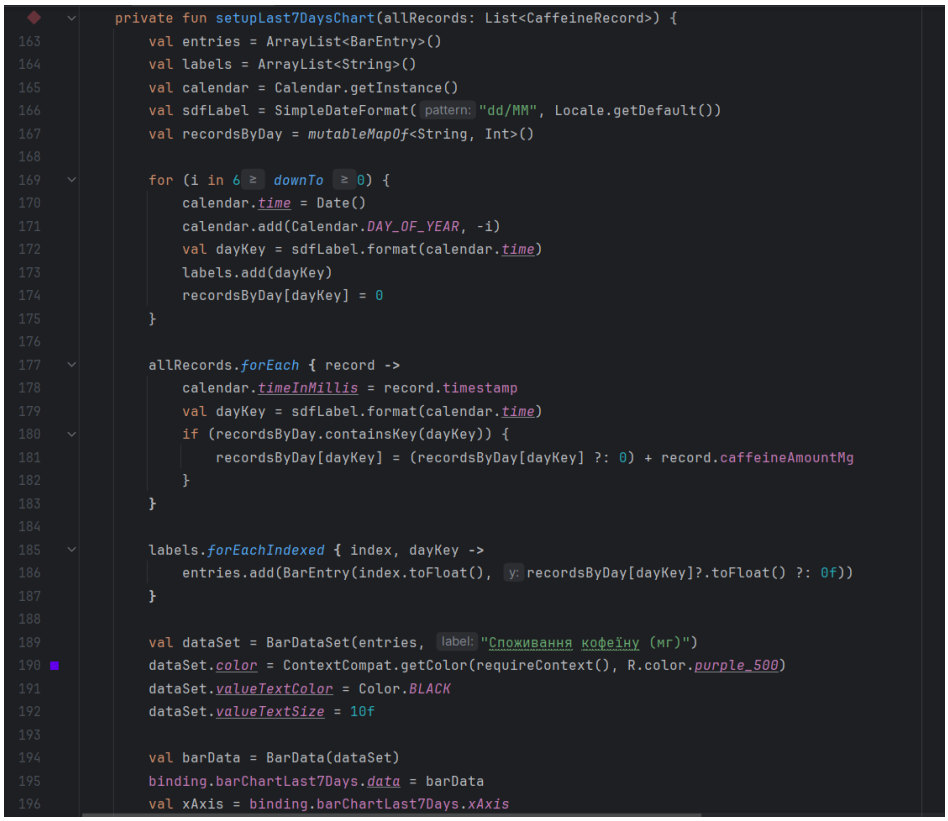
Виклик цих методів з MainViewModel відбувається асинхронно, як показано на рисунку 3.3. Це забезпечує реактивність та стабільність роботи користувацького інтерфейсу.

3.3 Реалізація модуля статистики та візуалізації

Для візуалізації статистики споживання було інтегровано бібліотеку MPAndroidChart. Цей модуль реалізовано у фрагменті AnalyticsFragment.

Процес формування графіка складається з наступних кроків:

1. Отримання даних: AnalyticsFragment через MainViewModel запитує історію споживання за останні 7 днів.
2. Агрегація даних: Отримані дані групуються за датою, і для кожного дня розраховується сумарна кількість спожитого кофеїну.
3. Формування набору даних: На основі агрегованих даних створюється набір BarEntry, де кожний елемент відповідає одному дню (стовпцю на графіку).
4. Налаштування та відображення діаграми: Створений набір даних передається в об'єкт BarChart. На рисунку 3.4 показано, як виконується детальне налаштування зовнішнього вигляду діаграми: осі, підписи, кольори, анімації та форматування значень.



```
private fun setupLast7DaysChart(allRecords: List<CaffeineRecord>) {
    val entries = ArrayList<BarEntry>()
    val labels = ArrayList<String>()
    val calendar = Calendar.getInstance()
    val sdfLabel = SimpleDateFormat( pattern: "dd/MM", Locale.getDefault())
    val recordsByDay = mutableMapOf<String, Int>()

    for (i in 6 ≥ downTo ≥ 0) {
        calendar.time = Date()
        calendar.add(Calendar.DAY_OF_YEAR, -i)
        val dayKey = sdfLabel.format(calendar.time)
        labels.add(dayKey)
        recordsByDay[dayKey] = 0
    }

    allRecords.forEach { record ->
        calendar.timeInMillis = record.timestamp
        val dayKey = sdfLabel.format(calendar.time)
        if (recordsByDay.containsKey(dayKey)) {
            recordsByDay[dayKey] = (recordsByDay[dayKey] ?: 0) + record.caffeineAmountMg
        }
    }

    labels.forEachIndexed { index, dayKey ->
        entries.add(BarEntry(index.toFloat(), y: recordsByDay[dayKey]?.toFloat() ?: 0f))
    }

    val dataSet = BarDataSet(entries, label: "Споживання кофеїну (мг)")
    dataSet.color = ContextCompat.getColor(requireContext(), R.color.purple_500)
    dataSet.valueTextColor = Color.BLACK
    dataSet.valueTextSize = 10f

    val barData = BarData(dataSet)
    binding.barChartLast7Days.data = barData
    val xAxis = binding.barChartLast7Days.xAxis
```

Рис. 3.4. Фрагмент коду для налаштування графіка статистики

Окрім графіка, в цьому ж фрагменті реалізовано функціонал перегляду історії за будь-яку обрану дату. Для цього використовується системний DatePickerDialog, який дозволяє користувачу обрати дату. Після вибору, AnalyticsFragment робить новий запит до ViewModel для отримання записів за вказаний день.

Реалізація системи налаштувань та сповіщень

Функціонал налаштувань реалізовано у SettingsFragment. Для збереження налаштувань користувача, таких як денний ліміт, час нагадувань та статус перемикачів сповіщень, використовується механізм SharedPreferences.

Керування денним лімітом: Користувач може змінити денний ліміт через діалогове вікно. Нове значення зберігається у SharedPreferences. Для того, щоб інші частини застосунку (наприклад, TodayFragment) реагували на зміну ліміту, використовується OnSharedPreferenceChangeListener, який відстежує зміни у SharedPreferences та оновлює відповідні змінні.

Система сповіщень: Система сповіщень реалізована за допомогою NotificationManager та AlarmManager.

Сповіщення про перевищення ліміту: При кожному додаванні нового запису TodayFragment перевіряє, чи не перевищує сумарне споживання встановлений ліміт. Якщо так, і якщо відповідне налаштування увімкнене, викликається метод з NotificationUtils для створення та показу сповіщення.

Щоденні нагадування: Коли користувач вмикає нагадування у SettingsFragment, застосунок використовує AlarmManager для планування щоденної задачі. Як видно з рисунка 3.5, створюється PendingIntent, який буде виконано у вказаний час. AlarmManager в указаний час запускає ReminderBroadcastReceiver, який, у свою чергу, викликає метод з NotificationUtils для показу нагадування. Скасування нагадування також відбувається через AlarmManager.

```
46      private val requestNotificationPermissionLauncher =
47          registerForActivityResult(ActivityResultContracts.RequestPermission()) { isGranted: Boolean ->
48              if (isGranted) {
49                  Toast.makeText(requireContext(), text: "Дозвіл на сповіщення надано!", Toast.LENGTH_SHORT).show()
50                  updateSwitchesAfterPermissionResult(true)
51              } else {
52                  Toast.makeText(requireContext(), text: "Дозвіл на сповіщення не надано.", Toast.LENGTH_SHORT).show()
53                  updateSwitchesAfterPermissionResult(false)
54              }
55          }
56
57      private val requestExactAlarmPermissionLauncher =
58          registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
59              if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
60                  val alarmManager = requireContext().getSystemService(Context.ALARM_SERVICE) as android.app.AlarmManager
61                  if (alarmManager.canScheduleExactAlarms()) {
62                      Toast.makeText(requireContext(), text: "Дозвіл на точні нагадування надано.", Toast.LENGTH_SHORT).show()
63                      val reminderEnabled = sharedPreferences.getBoolean(Constants.KEY_NOTIFY_REMINDER, false)
64                      if(reminderEnabled) {
65                          NotificationUtils.scheduleDailyReminder(requireContext(), reminderHour, reminderMinute, enabled: true)
66                      }
67                  } else {
68                      Toast.makeText(requireContext(), text: "Дозвіл на точні нагадування не надано.", Toast.LENGTH_SHORT).show()
69                      binding.switchReminderNotification.isChecked = false // Скидаємо, якщо немає дозволу
70                      sharedPreferences.edit().putBoolean(Constants.KEY_NOTIFY_REMINDER, false).apply()
71                  }
72              }
73          }
```

Рис. 3.5. Фрагмент коду для планування щоденних нагадувань

Особливу увагу було приділено роботі з дозволами. Для Android 13 (API 33) і вище перед показом сповіщень застосунок запитує у користувача дозвіл POST_NOTIFICATIONS.

3.3.1 Алгоритм агрегації та підготовки даних для графіка

Процес перетворення сирих даних з бази даних у візуальний графік складається з чітко визначених кроків. Алгоритм, реалізований у AnalyticsFragment, можна описати наступним чином:

1. Запит даних. AnalyticsFragment через MainViewModel викликає метод getRecordsForPeriod(), передаючи йому часові межі (наприклад, останні 7 днів). ViewModel повертає LiveData, що містить список об'єктів CaffeineRecord.
2. Агрегація даних за датою. Отриманий список записів обробляється. Для цього створюється словник (Map), де ключем є дата у форматі рядка (наприклад, "08/06"), а значенням — сумарна кількість кофеїну, спожитого за цю дату. Програма ітерується по списку CaffeineRecord і для кожного запису додає його caffeineAmountMg до відповідного значення у словнику.
3. Трансформація даних для бібліотеки. Дані з агрегованого словника перетворюються у формат, зрозумілий для бібліотеки MPAndroidChart. Для цього створюється список об'єктів BarEntry, де кожен об'єкт представляє один стовпець на графіку і містить координати по осі X (індекс дня) та Y (сума кофеїну).
4. Створення та налаштування набору даних. Створений список BarEntry передається в конструктор об'єкта BarDataSet. На цьому етапі налаштовуються візуальні параметри, такі як колір стовпців, розмір підписів значень тощо.
5. Відображення графіка. Об'єкт BarDataSet передається в об'єкт BarChart, після чого викликається метод invalidate(), який перемальовує графік на екрані з новими даними.

Цей покроковий процес забезпечує ефективну обробку даних та їх коректне візуальне представлення.

3.4 Реалізація системи налаштувань та сповіщень

Для забезпечення персоналізації користувацького досвіду та проактивної взаємодії з користувачем, у застосунку було реалізовано модуль налаштувань та систему сповіщень. Цей функціонал реалізовано у SettingsFragment та NotificationUtils.

Керування налаштуваннями за допомогою SharedPreferences

Для збереження простих даних налаштувань користувача, таких як денний ліміт кофеїну, статус перемикачів сповіщень та час щоденних нагадувань, було обрано механізм SharedPreferences. Це стандартний для платформи Android інструмент для зберігання невеликих обсягів даних у форматі "ключ-значення".

Коли користувач змінює налаштування (наприклад, встановлює новий денний ліміт у діалоговому вікні), відповідне значення зберігається у SharedPreferences. Для того, щоб інші компоненти застосунку (наприклад, TodayFragment, який відображає прогрес-бар) реагували на ці зміни, використовується OnSharedPreferenceChangeListener. Цей слухач реєструється у фрагменті та автоматично спрацьовує при будь-якій зміні даних у SharedPreferences, дозволяючи оновити інтерфейс та внутрішню логіку застосунку в реальному часі.

Реалізація системи сповіщень

Система сповіщень є ключовим елементом для залучення користувача та виконання превентивної функції. Вона реалізована за допомогою системних сервісів NotificationManager та AlarmManager.

1. Сповіщення про перевищення ліміту. Цей тип сповіщень є реактивним. Логіка реалізована у TodayFragment. Кожного разу, коли додається новий запис про споживання, метод updateDailyIntake() перевіряє, чи не перевищує сумарна кількість кофеїну встановлений денний ліміт. Якщо ліміт перевищено і відповідне налаштування увімкнене, викликається метод з NotificationUtils для створення та показу сповіщення.

2. Щоденні нагадування. Цей тип сповіщень є запланованим і вимагає взаємодії кількох компонентів:
   - AlarmManager: Системний сервіс, що дозволяє виконувати код у визначений час у майбутньому, навіть якщо застосунок неактивний.
   - PendingIntent: Об'єкт, що містить намір (Intent), який буде виконано AlarmManager. У нашому випадку, це намір запустити ReminderBroadcastReceiver.
   - ReminderBroadcastReceiver: Компонент, який отримує системний сигнал від AlarmManager у заданий час. Його єдина функція – викликати метод з NotificationUtils для показу нагадування.

Робота з дозволами (Permissions)

Особливу увагу було приділено коректній роботі з дозволами, що є обов'язковим для сучасних версій Android.

Дозвіл на сповіщення: Для Android 13 (API 33) і вище, перед показом будь-якого сповіщення, застосунок повинен отримати від користувача дозвіл POST_NOTIFICATIONS. Запит цього дозволу реалізовано в SettingsFragment при активації відповідних перемикачів.

Дозвіл на точні будильники: Для надійної роботи AlarmManager на Android 12 (API 31) і вище може знадобитися дозвіл SCHEDULE_EXACT_ALARM. Застосунок перевіряє його наявність і, в разі потреби, може направити користувача до системних налаштувань для його надання.

Такий підхід до реалізації налаштувань та сповіщень забезпечує гнучкість, надійність та відповідність сучасним вимогам платформи Android.

3.5 Висновки до третього розділу

У третьому розділі було детально описано процес програмної реалізації мобільного застосунку "LatteNo".

Було описано використане середовище розробки та інструментарій, включаючи Android Studio, Kotlin та Gradle.

Розглянуто реалізацію ключових компонентів застосунку відповідно до архітектури MVVM. Детально описано створення шару даних за допомогою бібліотеки Room, реалізацію логіки у MainViewModel з використанням корутин для асинхронних операцій, а також взаємодію ViewModel з користувацьким інтерфейсом на прикладі TodayFragment.

Описано реалізацію модуля статистики та візуалізації даних. Продемонстровано процес інтеграції бібліотеки MPAndroidChart для побудови графіків споживання та використання DatePickerDialog для перегляду історії.

Також детально розглянуто реалізацію системи налаштувань та сповіщень. Описано використання SharedPreferences для збереження налаштувань користувача та AlarmManager і NotificationManager для реалізації щоденних нагадувань та сповіщень про перевищення ліміту.

Результатом роботи, описаної в даному розділі, є функціональний програмний продукт, готовий до етапу тестування.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

На даному етапі кваліфікаційної роботи проводиться тестування розробленого програмного продукту. Метою цього етапу є перевірка функціональної коректності застосунку, його стабільності, відповідності раніше визначеним вимогам, а також виявлення та усунення потенційних дефектів перед завершенням розробки.

4.1 Організація процесу тестування

Процес тестування було організовано з використанням комбінації різних видів та рівнів тестування для забезпечення комплексного покриття функціоналу.
Мета тестування:

Перевірити, чи реалізовано всі функціональні вимоги, визначені в розділі 2.1.

Переконатися у стабільності роботи застосунку при виконанні стандартних користувацьких сценаріїв.

Виявити та задокументувати будь-які дефекти або невідповідності в роботі програми.

Оцінити відповідність користувацького інтерфейсу спроектованим макетам та його інтуїтивну зрозумілість.

Види тестування:

Функціональне тестування: Основний вид тестування, спрямований на перевірку коректності роботи кожної функції системи. Проводилося вручну на основі розроблених тест-кейсів.

Тестування графічного інтерфейсу (GUI Testing): Перевірка правильності відображення всіх елементів інтерфейсу, їхнього розміщення, читабельності тексту та коректності роботи на екранах з різною щільністю пікселів.

Тестування юзабіліті (Usability Testing): Неформальна оцінка того, наскільки легко та інтуїтивно користувач може взаємодіяти із застосунком для досягнення своїх цілей.

Для проведення тестування використовувався емулятор Android Virtual Device (AVD) з образом системи Android 13 (API 33), а також фізичний пристрій з ОС Android 12 для перевірки роботи в реальних умовах.

4.2 Розробка тестових сценаріїв та тест-кейсів

Для систематичної перевірки функціоналу було розроблено набір тестових сценаріїв та детальних тест-кейсів, що покривають основні сценарії використання застосунку. У таблиці 4.1 наведено приклади ключових тест-кейсів.

Таблиця 4.1

Приклади тест-кейсів для функціонального тестування

ID	Назва тест-кейсу	Передумови	Кроки виконання	Очікуваний результат
TC-01	Додавання запису з існуючого списку	Застосунок запущено, користувач на головному екрані.	1.Натиснути кнопку +. 2. У діалозі обрати "Еспресо (60 мг)" зі списку. 3. Натиснути "Додати".	1.Діалог закривається. 2. На головному екрані з'являється новий запис "Еспресо". 3.Загальна кількість спожитого кофеїну збільшується на 60 мг.
TC-02	Додавання власного напою	Застосунок запущено, користувач на головному екрані.	1.Натиснути +. 2. Ввести "Матча лате" в поле назви. 3. Ввести "70" в поле кількості кофеїну. 4. Натиснути "Додати".	1. Діалог закривається. 2. Додано новий запис "Матча лате". 3. Цей напій не з'являється у списку для швидкого вибору.

Продовження таблиці 4.1

Приклади тест-кейсів для функціонального тестування

ID	Назва тест-кейсу	Передумови	Кроки виконання	Очікуваний результат
ТС-03	Редагування існуючого запису	На головному екрані є хоча б один запис.	1.Натиснути іконку редагування біля запису. 2.У діалозі змінити кількість кофеїну з "60" на "80". 3.Натиснути "Зберегти".	1.Діалог закривається. 2.Кількість кофеїну в записі оновилася на "80 мг". 3.Загальна денна кількість кофеїну перерахувалася.
ТС-04	Зміна денного ліміту	Користувач на екрані "Налаштування".	1.Натиснути кнопку "Змінити" біля денного ліміту. 2.У діалозі ввести "350". 3.Натиснути "Зберегти".	1.Діалог закривається. 2.Текст на екрані "Налаштування" оновився на "Поточний ліміт: 350 мг". 3. На головному екрані максимальне значення прогрес-бару змінилося на 350.
ТС-05	Перевірка спрацювання сповіщення	Денний ліміт встановлено на 100 мг. Поточне споживання 80 мг. Сповіщення увімкнені.	1.Додати новий напій з кількістю кофеїну "30 мг".	1.У системній панелі з'являється сповіщення з текстом про перевищення ліміту.
ТС-06	Видалення історії за період	В базі даних є записи за декілька днів. Користувач на екрані "Налаштування".	1. Натиснути "Видалити історію за період". 2. У календарі обрати початкову та кінцеву дати. 3. Підтвердити видалення у діалозі.	1.З'являється повідомлення про успішне видалення. 2.Записи за обраний період зникають з історії на екрані "Аналітика".

4.3 Аналіз результатів тестування

Тестування проводилося шляхом послідовного виконання всіх розроблених тест-кейсів. Результати виконання ключових тест-кейсів наведено в таблиці 4.2.

Таблиця 4.2.

Результати виконання тест-кейсів

ID	Назва тест-кейсу	Фактичний результат	Статус
ТС-01	Додавання запису з існуючого списку	Діалог закрито, на головному екрані з'явився новий запис "Еспресо". Загальна кількість кофеїну оновилася. Результат відповідає очікуваному.	Успішно
ТС-02	Додавання власного напою	Діалог закрито, додано новий запис "Матча латте". Напій не з'явився у списку швидкого вибору. Результат відповідає очікуваному.	Успішно
ТС-03	Редагування існуючого запису	Діалог закрито, кількість кофеїну в записі оновилася на "80 мг". Загальна денна кількість перерахувалася. Результат відповідає очікуваному.	Успішно
ТС-04	Зміна денного ліміту	Діалог закрито, текст ліміту на екрані "Налаштування" та максимальне значення прогрес-бару оновилися. Результат відповідає очікуваному.	Успішно
ТС-05	Перевірка спрацювання сповіщення	Після додавання напою, що перевищує ліміт, у системній панелі пристрою з'явилося сповіщення. Результат відповідає очікуваному.	Успішно
ТС-06	Видалення історії за період	Після підтвердження видалення з'явилося повідомлення про успішну операцію. Записи за обраний період зникли з історії на екрані "Аналітика". Результат відповідає очікуваному.	Успішно

За результатами функціонального тестування критичних дефектів, що блокують роботу основного функціоналу, виявлено не було. Застосунок продемонстрував стабільну роботу та відповідність усім функціональним вимогам, визначеним у розділі 2.1.

4.4 Інструкція користувача

Дана інструкція описує основні сценарії використання мобільного застосунку "LatteNo".

1. Головний екран. Після запуску застосунку відкривається головний екран (рисунок 4.1). На ньому відображається загальна кількість спожитого кофеїну за поточний день, встановлений денний ліміт та історія споживання. Для додавання нового запису необхідно натиснути кнопку "+".

Рис. 4.1. Головний екран застосунку

2. Додавання напою. Після натискання на кнопку "+" відкривається діалогове вікно (рисунок 4.2). Тут можна обрати напій зі списку або ввести назву та кількість кофеїну вручну. Після заповнення полів необхідно натиснути "Додати".

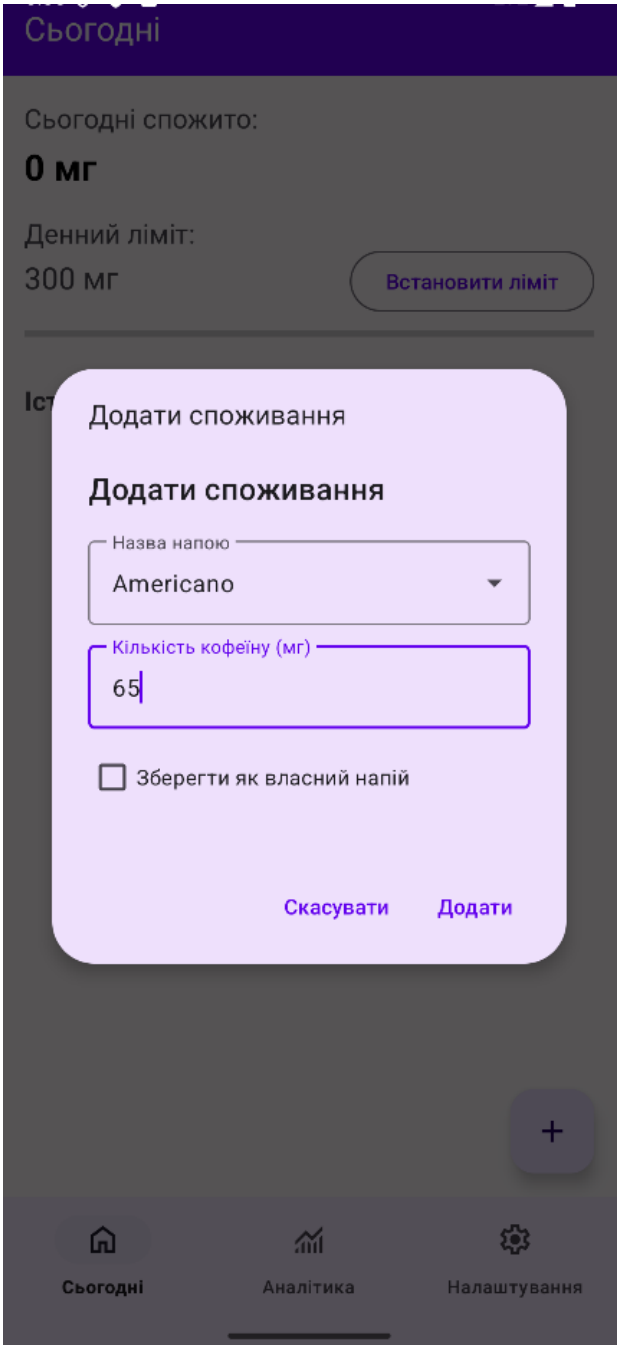


Рис. 4.2. Діалог додавання запису

3. Екран аналітики. На вкладці "Аналітика" (рисунок 4.3) відображається графік споживання за останні 7 днів та середні значення. Також тут можна обрати будь-яку дату, щоб переглянути історію споживання за цей день.

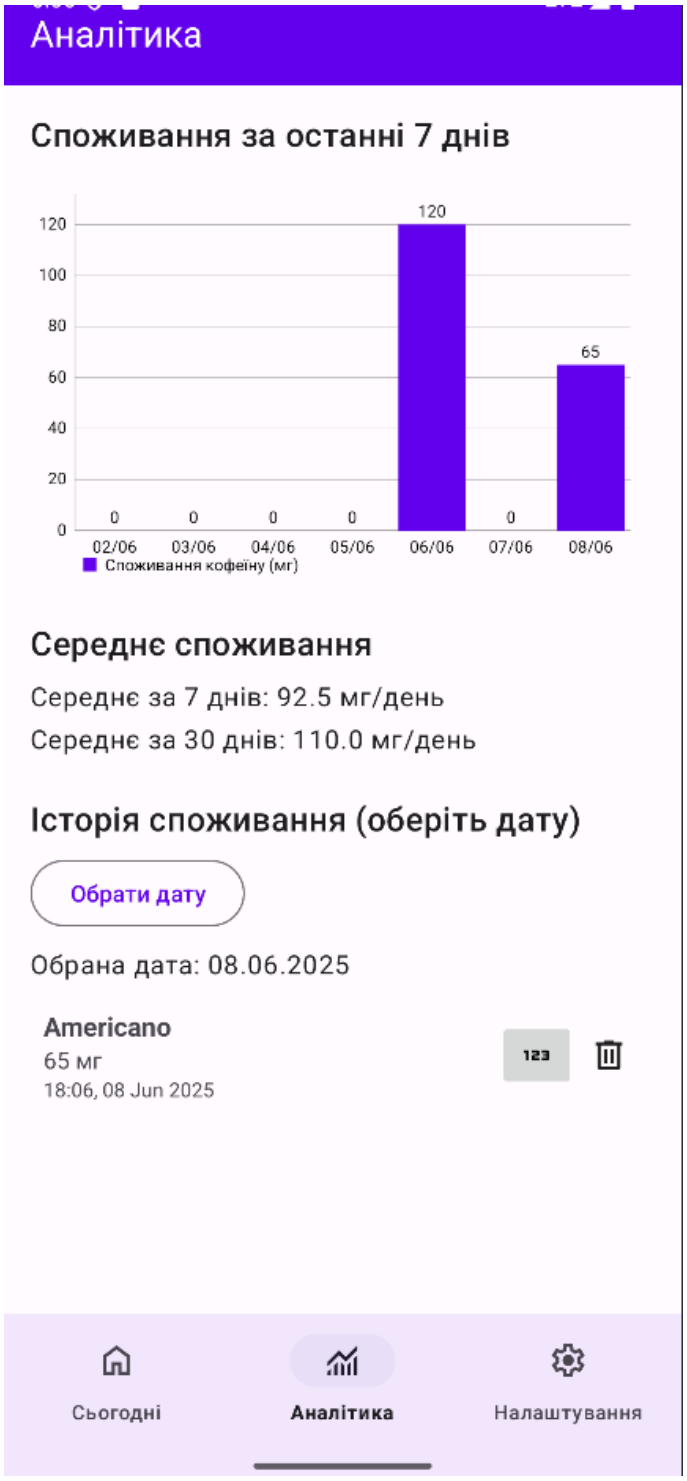


Рис. 4.3. Екран аналітики

4. Екран налаштувань. На вкладці "Налаштування" (рисунок 4.4) користувач може змінити денний ліміт, увімкнути або вимкнути сповіщення та щоденні нагадування, а також видалити історію за обраний період.

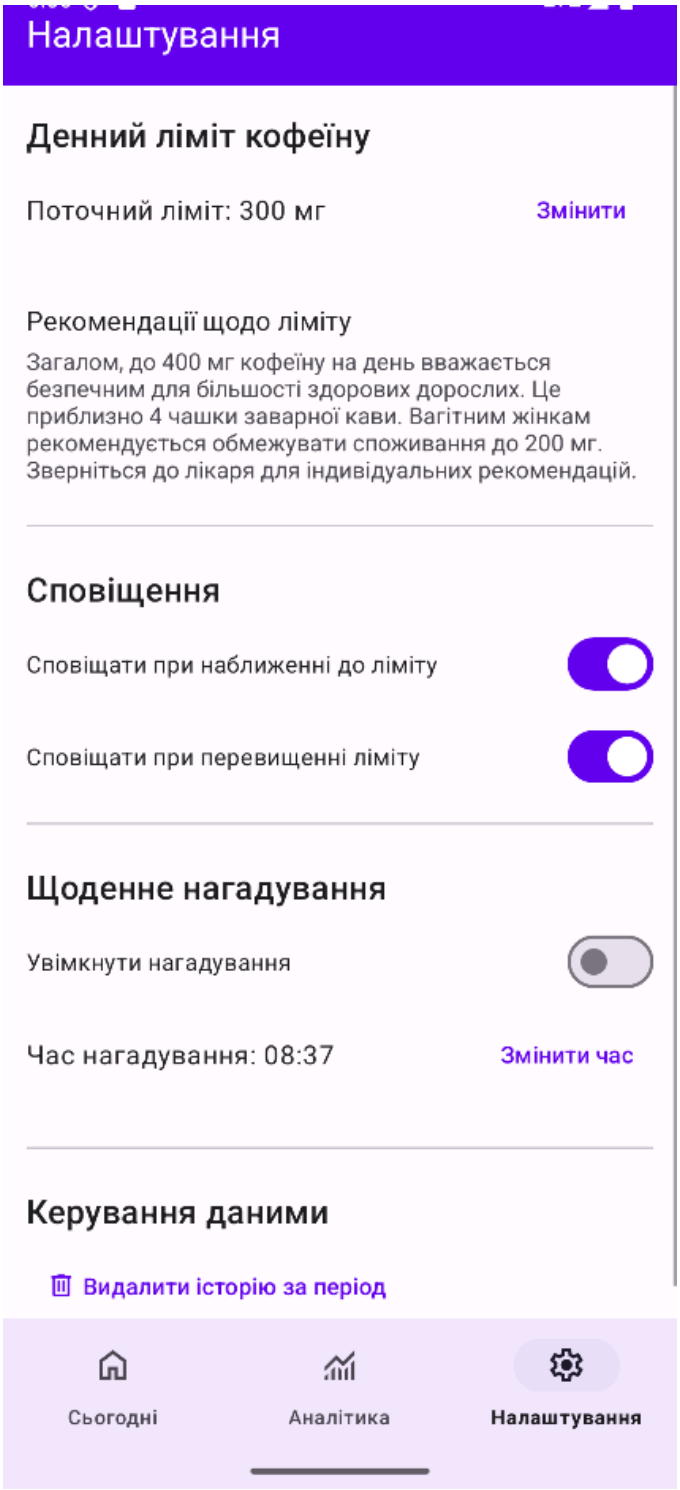


Рис. 4.4. Екран налаштувань

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу розробки мобільного застосунку для контролю споживання кофеїну, що надає користувачам інструмент для моніторингу та формування усвідомлених звичок.

У процесі виконання роботи були отримані наступні результати:

1. У першому розділі було проведено комплексний аналіз предметної галузі, в ході якого визначено актуальність проблеми контролю споживання кофеїну, потреби цільової аудиторії, а також функціональні прогалини існуючих програмних аналогів. На основі цього було обґрунтовано вибір стеку технологій для розробки, зокрема мови Kotlin та архітектури MVVM, як найбільш відповідних для створення сучасного та надійного Android-застосунку.
2. У другому розділі було виконано етап проектування системи. Сформульовано детальний перелік функціональних та нефункціональних вимог до майбутнього продукту. Спроектовано архітектуру застосунку за шаблоном MVVM та розроблено UML-діаграми (Use Case, Class Diagram) для візуалізації його структури та взаємодій. Також було спроектовано структуру локальної бази даних для зберігання даних користувача.
3. У третьому розділі детально описано процес програмної реалізації застосунку. Реалізовано ключові функціональні модулі: керування записами про споживання (додавання, редагування, видалення), відстеження денного ліміту, візуалізація статистики за допомогою бібліотеки MPAndroidChart, а також система налаштувань та сповіщень.
4. У четвертому розділі було проведено тестування розробленого програмного продукту. Розроблено набір тест-кейсів, що покривають основні користувацькі сценарії. Результати тестування підтвердили відповідність застосунку заявленим вимогам, його стабільну роботу та відсутність критичних дефектів.

Таким чином, мета кваліфікаційної роботи — розробка мобільного застосунку для контролю споживання кофеїну — є повністю досягнутою. Результатом роботи є готовий до використання програмний продукт для платформи Android, що надає користувачам весь необхідний функціонал для ефективного моніторингу своїх звичок.

Практична значущість роботи полягає у створенні інструменту, що може бути використаний широким колом користувачів для підвищення обізнаності про власне здоров'я, дотримання медичних рекомендацій та покращення якості життя.

Напрямками подальшого розвитку проекту можуть бути розширення бази даних напоїв, інтеграція з фітнес-трекерами для врахування індивідуальних показників користувача, а також реалізація хмарної синхронізації даних між різними пристроями.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Чорний Р.О., Щербина І.С. Розробка мобільного застосунку для моніторингу споживання кофеїну: сучасні підходи та перспективи використання // II Міжнародна науково-практична конференція The Future of Science, Technology and Economy» International Scientific Unity (ISU), , 11.06.2025 – м. Софія, Болгарія, 2025. – (подано до друку)
2. Чорний Р.О., Щербина І.С. Використання мобільних технологій для контролю індивідуальних показників здоров'я: приклад розробки застосунку для моніторингу споживання кофеїну // VIII Всеукраїнська студентська конференція «Розвиток сучасної науки: актуальні питання теорії та практики» ГО «Молодіжна наукова ліга», 20.06.2025 – м. Запоріжжя, Україна, 2025. – (подано до публікації)

ПЕРЕЛІК ПОСИЛАНЬ

1. Григорук П. М. Вплив кофеїну на когнітивні функції та якість сну людини. *Український журнал медицини, біології та спорту*. 2023. Т. 8, № 4. С. 55–61.
2. Коваленко О. С. Застосування архітектурного шаблону MVVM при розробці мобільних застосунків на платформі Android. *Вісник Національного технічного університету "ХПІ". Серія: Нові рішення в сучасних технологіях*. 2022. № 2(10). С. 25–30.
3. Захарова І. В. Використання бібліотеки Room для організації локального сховища даних в Android-додатках. *Інформаційні технології та системи*. 2022. № 1(15). С. 45–51.
4. Бондаренко С. В. Психологічні аспекти формування звичок за допомогою мобільних технологій. *Психологія і суспільство*. 2022. № 3(89). С. 98–107.
5. Шевченко М. В., Ткаченко А. П. Методи та засоби тестування програмного забезпечення мобільних систем. *Проблеми інформатики та моделювання : матеріали XIV Міжнар. наук.-техн. конф.* (Київ, 17-19 травня 2022 р.). Київ: КНУ ім. Тараса Шевченка, 2022. С. 112–114.
6. Клименко Ю. Асинхронне програмування в Kotlin з використанням корутин. *Інформаційні технології в освіті та науці : матеріали V Всеукр. наук.-практ. конф.* (Мелітополь, 21-23 квітня 2023 р.). Мелітополь: МДПУ ім. Б. Хмельницького, 2023. С. 210–212.
7. Sommerville I. Software Engineering. 11th ed. Pearson, 2023. 840 p.
8. Temple J. L., Bernard C., Lipshultz S. E. et al. The Safety of Ingested Caffeine: A Comprehensive Review. *Frontiers in Psychiatry*. 2022. Vol. 13. Art. 977278. DOI: 10.3389/fpsyt.2022.977278.
9. Poole R., Kennedy O. J., Roderick P. et al. Coffee consumption and health: umbrella review of meta-analyses of multiple health outcomes. *BMJ*. 2022. Vol. 378. Art. e068665.

10. Gardner B., Lally P. Does habit weaken the relationship between intention and behaviour? A test of the habit-intention interaction hypothesis. *Social and Personality Psychology Compass*. 2022. Vol. 16, no. 8. Art. e12693.
11. Kumar S., Nilsen W. J., Abernethy A. et al. Mobile Health Technology Evaluation and Implementation. *Frontiers in Digital Health*. 2023. Vol. 5. Art. 1133917.
12. Hanoch Y., Miron-Shatz T., Tsvik K. Mobile Health (mHealth) and its implications for patient-centeredness in a digital era. *Journal of Medical Internet Research*. 2022. Vol. 24, no. 5. Art. e34237.
13. Lee M., Lee H., Kim J. A study on the user experience of health tracking applications. *Journal of Digital Interaction Design*. 2023. Vol. 16, no. 2. P. 45–56.
14. Петренко І. Проєктування користувацького досвіду (UX) для мобільних додатків: сучасні підходи та патерни. *DOU.ua*. 2023. URL: https://dou.ua/lenta/articles/mobile-ux-patterns/ (дата звернення: 10.05.2025).
15. Рекомендації щодо добового споживання кофеїну / Центр громадського здоров'я МОЗ України. 2023. URL: https://phc.org.ua/news/skilki-kavi-mozhna-piti-dopomoga-kavomanam (дата звернення: 28.04.2025).
16. *Настанова до збірки знань з управління проєктами (Настанова PMBOK). 7-е видання* / Інститут Проєктного Менеджменту Україна. 2023. URL: https://pmiukraine.org/pmbok7 (дата звернення: 15.04.2025).
17. *Офіційна документація мови програмування Kotlin: What's new in Kotlin 1.9.20*. JetBrains. 2023. URL: https://kotlinlang.org/docs/whatsnew1920.html (дата звернення: 11.05.2025).
18. *Guide to app architecture*. Android Developers. 2023. URL: https://developer.android.com/jetpack/guide (дата звернення: 11.05.2025).

19. *Room Persistence Library Overview*. Android Developers. 2024. URL: https://developer.android.com/training/data-storage/room (дата звернення: 12.05.2025).
20. *Jahoda P. MPAndroidChart Library*. GitHub Repository. 2024. URL: https://github.com/PhilJay/MPAndroidChart (дата звернення: 12.05.2025).
21. *Notifications in Android 13 and higher*. Android Developers. 2023. URL: https://developer.android.com/develop/ui/views/notifications/notification-permission (дата звернення: 20.05.2025).
22. *Material Design 3 Guidelines*. Google. 2024. URL: https://m3.material.io/ (дата звернення: 15.05.2025).
23. *SharedPreferences | Android Developers*. 2024. URL: https://developer.android.com/training/data-storage/shared-preferences (дата звернення: 22.05.2025).
24. *Git Documentation*. 2024. URL: https://git-scm.com/doc (дата звернення: 14.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільного застосунку для контролю споживання кофеїну

Виконав студент 4 курсу

групи ПД-43
Чорний Роман Олександрович
Керівникроботи

к.т.н, доцент, доцент кафедри ІПЗ Щербина Ірина Сергіївна

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** Спростити контроль заспоживанням кофеїну за допомогою мобільного застосунку
- **Об’єкт дослідження**Процес споживання кофеїновмісних продуктівта методи його контролю за допомогою мобільних технологій.
- **Предмет дослідження:** Технології та засоби для відстеження рівня кофеїну, зокрема мобільні застосунки.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати існуючі мобільні застосунки для контролю споживання кофеїну та визначити їх переваги й недоліки.
2. Сформулювати функціональні та нефункціональні вимоги до мобільного застосунку «LatteNo».
3. Спроектувати архітектуру мобільного застосунку, структуру бази даних та інтерфейс користувача.
4. Розробити основний функціонал застосунку: додавання напоїв, відстеження денної норми, історія споживання, налаштування лімітів, візуалізація статистики, нагадування для користувача та система сповіщень
5. Реалізувати локальне збереження даних та налаштувань.
6. Провести тестування розробленого мобільного застосунку.

АНАЛІЗ АНАЛОГІВ

Характеристика	CaffeInMe	Coffe Plus	Caffeine Tracker	Caffeine Clock	**LatteNo**
Відстеження споживання кофеїну	+	+	+	+	+
Візуалізація даних (графіки)	-	+	-	-	+
Налаштування персональних цілей	+	+	-	-	+
Додавання власних напоїв	-	+	-	+	+
Перегляд історії споживання кофеїну	-	+	+	-	+
Очищення даних за період	-	-	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональнівимоги:

1. Можливість додавати записи про спожитий кофеїн, обираючи напій зі списку або вводячи дані вручну.
2. Відображення поточного рівня спожитого кофеїну відносно встановленого денного ліміту.
3. Ведення та перегляд історії споживання кофеїну.
4. Надання статистики споживання у вигляді графіка
5. Можливість користувача встановлювати та змінювати денний ліміт кофеїну.
6. Надсилання сповіщень при наближенні/перевищенні денного ліміту.
7. Можливістьочищення даних за період.

Нефункціональні вимоги:

1. Забезпечення стабільної роботи без збоїв при типових сценаріях використання.
2. Сумісність з більшістю сучасних Android-пристроїв (Android 7.0/API 24+)
3. Локальне збереження даних користувачана пристрої.
4. Інсталяція та запуск без додаткового налаштування зі сторони користувача.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Kotlin

Android

android studio

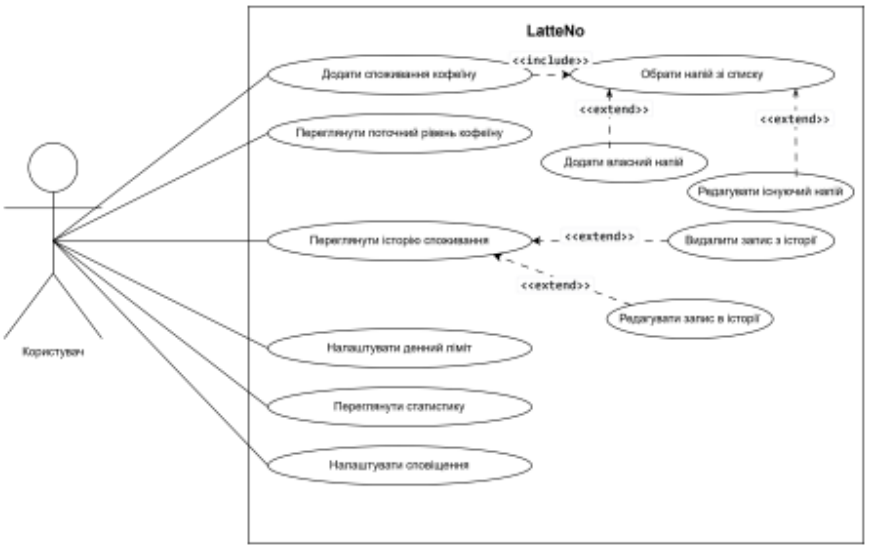
MPAndroidChart

created by Philipp Jahoda

GitHub

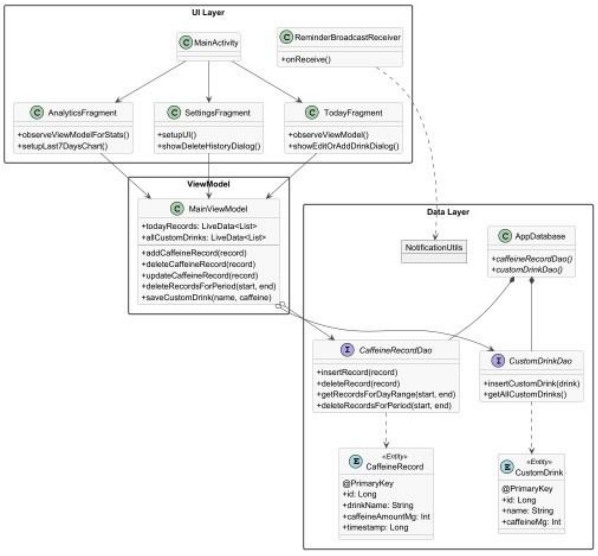
6

Діаграма варіантів використання



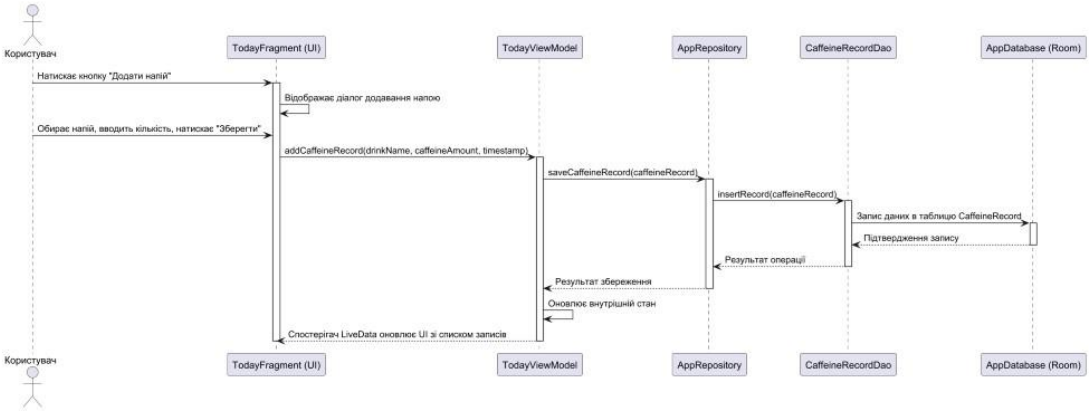
7

Діаграма класів



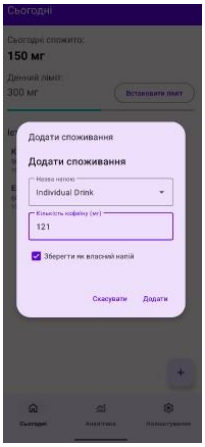
8

Діаграма послідовності: Додавання запису про споживання кофеїну

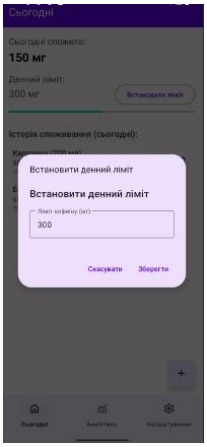


ЕКРАННІ ФОРМИ

Головна сторінка застосунку

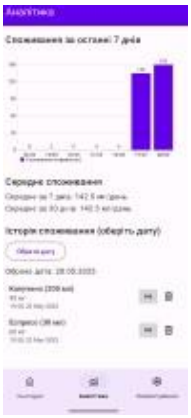


Меню додавання власного напою

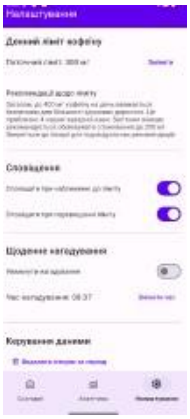


Зміна денного ліміту кофеїну

ЕКРАННІ ФОРМИ



Сторінка аналітики з графіком та історією споживання



Налаштування застосунку та сповіщень

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Чорний Р.О., Щербина І.С.Розробкамобільного застосунку для моніторингу споживання кофеїну: сучасні підходи та перспективи використання // II Міжнародна науково-практична конференціяThe Future of Science, Technology and Economy» International Scientific Unity (ISU)(м. Софія, Болгарія), 11.06.2025 – Софія, 2025. – (подано до друку)

2. Чорний Р.О., Щербина І.С. Використання мобільних технологій для контролю індивідуальних показників здоров'я: приклад розробки застосунку для моніторингу споживання кофеїнуVIII Всеукраїнськастудентська конференція «Розвитоксучасної науки: актуальні питання теорії та практики» ГО «Молодіжнанаукова ліга» (м. Запоріжжя, Україна), 20.06.2025 – Запоріжжя, 2025. – (подано до публікації)

ВИСНОВКИ

1. Проведено аналіз існуючих аналогів, визначено їхні переваги та недоліки, що дозволило виділити ключові функціональні відмінності майбутнього застосунку.
2. Сформульовано функціональні та нефункціональні вимоги, що лягли в основу архітектури проєкту.
3. Реалізовано основний функціонал: додавання напоїв, підрахунок денної норми кофеїну, ведення історії, візуалізація статистики, встановлення лімітів та система сповіщень.
4. Для локального збереження структурованих даних використано бібліотеку від Google Room, що забезпечує взаємодію з вбудованою в Android базою даних SQLite, а для збереження налаштувань користувача – компонент Jetpack DataStore.
5. Проведено тестування функціоналу.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

Лістинг класу MainViewModel

```
class MainViewModel(application: Application) :
AndroidViewModel(application) {

    private val caffeineRecordDao: CaffeineRecordDao
    private val customDrinkDao: CustomDrinkDao

    val todayRecords: LiveData<List<CaffeineRecord>>
    val allCustomDrinks: LiveData<List<CustomDrink>>


    init {
        val database = AppDatabase.getDatabase(application)
        caffeineRecordDao = database.caffeineRecordDao()
        customDrinkDao = database.customDrinkDao()

        allCustomDrinks =
customDrinkDao.getAllCustomDrinks()

        val calendar = Calendar.getInstance()
        val startOfDay = calendar.apply {
            set(Calendar.HOUR_OF_DAY, 0);
set(Calendar.MINUTE, 0); set(Calendar.SECOND, 0);
set(Calendar.MILLISECOND, 0)
        }.timeInMillis
        val endOfDay = calendar.apply {
            set(Calendar.HOUR_OF_DAY, 23);
set(Calendar.MINUTE, 59); set(Calendar.SECOND, 59);
set(Calendar.MILLISECOND, 999)
        }.timeInMillis

        todayRecords =
caffeineRecordDao.getRecordsForDayRange(startOfDay,
endOfDay)
    }

    fun addCaffeineRecord(record: CaffeineRecord) {
        viewModelScope.launch {
            caffeineRecordDao.insertRecord(record)
        }
    }

    fun deleteCaffeineRecord(record: CaffeineRecord) {
        viewModelScope.launch {
            caffeineRecordDao.deleteRecord(record)
        }
    }

    fun saveCustomDrink(name: String, caffeineMg: Int) {
        viewModelScope.launch {
            val existingDrink =
customDrinkDao.getCustomDrinkByName(name)
            if (existingDrink == null) { // Додаємо тільки якщо
напою з такою назвою ще немає

customDrinkDao.insertCustomDrink(CustomDrink(name =
name, caffeineMg = caffeineMg))
            }
            // Якщо напій вже існує, можна вирішити
оновлювати його або нічого не робити.
        }
    }

    // Функція для редагування запису історії
    fun updateCaffeineRecord(record: CaffeineRecord) {
        viewModelScope.launch {
            caffeineRecordDao.updateRecord(record)
        }
    }

    fun getRecordsForDay(startTime: Long, endTime: Long):
LiveData<List<CaffeineRecord>> {
        return
caffeineRecordDao.getRecordsForDayRange(startTime,
endTime)
    }

    // Метод для AnalyticsFragment для отримання записів
за період (для графіків, середніх)
    fun getRecordsForPeriod(startTime: Long, endTime:
Long): LiveData<List<CaffeineRecord>> {
        return
caffeineRecordDao.getRecordsForPeriod(startTime,
endTime)
    }

    fun deleteRecordsForPeriod(startTime: Long, endTime:
Long) {
        viewModelScope.launch {

caffeineRecordDao.deleteRecordsForPeriod(startTime,
endTime)
        }
    }
}
```

Лістинг файлу CaffeineRecordDao.kt

```
@Dao
interface CaffeineRecordDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertRecord(record: CaffeineRecord)

    @Query("SELECT * FROM caffeine_records WHERE
timestamp >= :startTime AND timestamp <= :endTime
ORDER BY timestamp DESC")
    fun getRecordsForDayRange(startTime: Long, endTime:
Long): LiveData<List<CaffeineRecord>>

    @Delete
    suspend fun deleteRecord(record: CaffeineRecord)

    @Update
    suspend fun updateRecord(record: CaffeineRecord)

    @Query("SELECT * FROM caffeine_records WHERE
timestamp >= :startTime AND timestamp <= :endTime
ORDER BY timestamp DESC")
    fun getRecordsForPeriod(startTime: Long, endTime:
Long): LiveData<List<CaffeineRecord>>

    @Query("DELETE FROM caffeine_records WHERE
timestamp >= :startTime AND timestamp <= :endTime")
    suspend fun deleteRecordsForPeriod(startTime: Long,
endTime: Long)
}
```

Лістинг файлу AnalyticsFragment

```
class AnalyticsFragment : Fragment() {

    private var _binding: FragmentAnalyticsBinding? = null
    private val binding get() = _binding!!
```

```
    private val viewModel: MainViewModel by
activityViewModels()
    private lateinit var analyticsHistoryAdapter:
CaffeineHistoryAdapter
    private val selectedDateCalendar: Calendar =
Calendar.getInstance()

    // Список для AutoCompleteTextView в діалозі
редагування/додавання
    private val combinedDrinkListForDialogAnalytics =
mutableListOf<Pair<String, Int>>()
    private lateinit var drinkNameAdapterForDialogAnalytics:
ArrayAdapter<String>

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentAnalyticsBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View,
savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Спостерігаємо за кастомними напоями для
оновлення списку в діалозі

viewModel.allCustomDrinks.observe(viewLifecycleOwner)
{ customDrinks ->

updateCombinedDrinkListForDialogAnalytics(customDrinks
?: emptyList())
        }

        setupRecyclerView()
        setupDatePicker()
        observeViewModelForStats()

updateSelectedDateRecords(System.currentTimeMillis())
    }

    private fun
updateCombinedDrinkListForDialogAnalytics(customDrinks
: List<CustomDrink>) {
        combinedDrinkListForDialogAnalytics.clear()
        Constants.PREDEFINED_DRINKS.forEach {

combinedDrinkListForDialogAnalytics.add(Pair("${it.name}
(${it.caffeineMg} мг)", it.caffeineMg))
        }
        customDrinks.forEach {
            val drinkDisplayName = "${it.name}
(${it.caffeineMg} мг)"
            if (combinedDrinkListForDialogAnalytics.none {
item -> item.first.equals(drinkDisplayName, ignoreCase =
true) }) {

combinedDrinkListForDialogAnalytics.add(Pair(drinkDispla
yName, it.caffeineMg))
            }
        }

        val displayNames =
combinedDrinkListForDialogAnalytics.map { it.first
}.distinct().sorted()

        if
(!::drinkNameAdapterForDialogAnalytics.isInitialized) {
            drinkNameAdapterForDialogAnalytics =
ArrayAdapter(requireContext(),
android.R.layout.simple_dropdown_item_1line,
displayNames)
        } else {
            drinkNameAdapterForDialogAnalytics.clear()

drinkNameAdapterForDialogAnalytics.addAll(displayNames
)

drinkNameAdapterForDialogAnalytics.notifyDataSetChange
d()
        }
    }


    private fun setupRecyclerView() {
        analyticsHistoryAdapter = CaffeineHistoryAdapter(
            onDeleteClick = { record ->
showDeleteConfirmationDialog(record) },
            onEditClick = { record ->
showEditOrAddDrinkDialogAnalytics(record) }
        )
        binding.rvAnalyticsHistory.apply {
            adapter = analyticsHistoryAdapter
            layoutManager =
LinearLayoutManager(requireContext())
        }
    }

    private fun setupDatePicker() {
        binding.btnSelectDate.setOnClickListener {
            val dateSetListener =
DatePickerDialog.OnDateSetListener { _, year, month,
dayOfMonth ->
                selectedDateCalendar.set(Calendar.YEAR, year)
                selectedDateCalendar.set(Calendar.MONTH,
month)

selectedDateCalendar.set(Calendar.DAY_OF_MONTH,
dayOfMonth)

updateSelectedDateRecords(selectedDateCalendar.timeInMil
lis)
            }
            DatePickerDialog(
                requireContext(),
                dateSetListener,
                selectedDateCalendar.get(Calendar.YEAR),
                selectedDateCalendar.get(Calendar.MONTH),

selectedDateCalendar.get(Calendar.DAY_OF_MONTH)
            ).show()
        }
        val sdf = SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault())
        binding.tvSelectedDate.text = "Обрана дата:
${sdf.format(selectedDateCalendar.time)}"
    }

    private fun updateSelectedDateRecords(timestamp: Long)
{
        val sdf = SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault())
        binding.tvSelectedDate.text = "Обрана дата:
${sdf.format(Date(timestamp))}"
```

```
        val calendar = Calendar.getInstance().apply {
timeInMillis = timestamp }
        val startOfDay = calendar.apply {
            set(Calendar.HOUR_OF_DAY, 0);
set(Calendar.MINUTE, 0); set(Calendar.SECOND, 0);
set(Calendar.MILLISECOND, 0)
        }.timeInMillis
        val endOfDay = calendar.apply {
            set(Calendar.HOUR_OF_DAY, 23);
set(Calendar.MINUTE, 59); set(Calendar.SECOND, 59);
set(Calendar.MILLISECOND, 999)
        }.timeInMillis

        viewModel.getRecordsForDay(startOfDay,
endOfDay).observe(viewLifecycleOwner, Observer { records
->

analyticsHistoryAdapter.submitList(records?.sortedByDesce
nding { it.timestamp })
        })
    }

    private fun observeViewModelForStats() {
        val thirtyDaysAgo = Calendar.getInstance().apply {
add(Calendar.DAY_OF_YEAR, -30) }.timeInMillis
        val now = System.currentTimeMillis()

        viewModel.getRecordsForPeriod(thirtyDaysAgo,
now).observe(viewLifecycleOwner) { allRecords ->
            if (allRecords.isNullOrEmpty()) {
                binding.barChartLast7Days.clear()
                binding.barChartLast7Days.invalidate()
                binding.tvAverageIntake7Days.text = "Середнє за
7 днів: 0 мг/день"
                binding.tvAverageIntake30Days.text = "Середнє
за 30 днів: 0 мг/день"
                return@observe
            }
            setupLast7DaysChart(allRecords)
            calculateAndDisplayAverages(allRecords)
        }
    }

    private fun setupLast7DaysChart(allRecords:
List<CaffeineRecord>) {
        val entries = ArrayList<BarEntry>()
        val labels = ArrayList<String>()
        val calendar = Calendar.getInstance()
        val sdfLabel = SimpleDateFormat("dd/MM",
Locale.getDefault())
        val recordsByDay = mutableMapOf<String, Int>()

        for (i in 6 downTo 0) {
            calendar.time = Date()
            calendar.add(Calendar.DAY_OF_YEAR, -i)
            val dayKey = sdfLabel.format(calendar.time)
            labels.add(dayKey)
            recordsByDay[dayKey] = 0
        }

        allRecords.forEach { record ->
            calendar.timeInMillis = record.timestamp
            val dayKey = sdfLabel.format(calendar.time)
            if (recordsByDay.containsKey(dayKey)) {
                recordsByDay[dayKey] = (recordsByDay[dayKey]
?: 0) + record.caffeineAmountMg
            }
        }

        labels.forEachIndexed { index, dayKey ->
            entries.add(BarEntry(index.toFloat(),
recordsByDay[dayKey]?.toFloat() ?: 0f))
        }

        val dataSet = BarDataSet(entries, "Споживання
кофеїну (мг)")
        dataSet.color =
ContextCompat.getColor(requireContext(),
R.color.purple_500)
        dataSet.valueTextColor = Color.BLACK
        dataSet.valueTextSize = 10f

        val barData = BarData(dataSet)
        binding.barChartLast7Days.data = barData
        val xAxis = binding.barChartLast7Days.xAxis
        xAxis.valueFormatter =
IndexAxisValueFormatter(labels)
        xAxis.position = XAxis.XAxisPosition.BOTTOM
        xAxis.granularity = 1f
        xAxis.setDrawGridLines(false)
        xAxis.setDrawAxisLine(true)
        binding.barChartLast7Days.axisLeft.axisMinimum = 0f
        binding.barChartLast7Days.axisRight.isEnabled = false
        binding.barChartLast7Days.description.isEnabled =
false
        binding.barChartLast7Days.animateY(1000)
        binding.barChartLast7Days.invalidate()
    }

    private fun calculateAndDisplayAverages(allRecords:
List<CaffeineRecord>) {
        val sevenDaysAgoTimestamp =
Calendar.getInstance().apply {
add(Calendar.DAY_OF_YEAR, -7);
set(Calendar.HOUR_OF_DAY, 0); set(Calendar.MINUTE,
0); set(Calendar.SECOND, 0);
set(Calendar.MILLISECOND, 0) }.timeInMillis
        val recordsLast7Days = allRecords.filter { it.timestamp
>= sevenDaysAgoTimestamp }
        val totalCaffeine7Days = recordsLast7Days.sumOf {
it.caffeineAmountMg }
        val distinctDays7 = recordsLast7Days.map {
            val cal = Calendar.getInstance()
            cal.timeInMillis = it.timestamp
            cal.set(Calendar.HOUR_OF_DAY, 0)
            cal.set(Calendar.MINUTE, 0)
            cal.set(Calendar.SECOND, 0)
            cal.set(Calendar.MILLISECOND, 0)
            cal.timeInMillis
        }.distinct().count()
        val average7Days = if (distinctDays7 > 0)
totalCaffeine7Days.toDouble() / distinctDays7 else 0.0
        binding.tvAverageIntake7Days.text =
String.format(Locale.getDefault(), "Середнє за 7 днів: %.1f
мг/день", average7Days)

        val thirtyDaysAgoTimestamp =
Calendar.getInstance().apply {
add(Calendar.DAY_OF_YEAR, -30);
set(Calendar.HOUR_OF_DAY, 0); set(Calendar.MINUTE,
0); set(Calendar.SECOND, 0);
set(Calendar.MILLISECOND, 0) }.timeInMillis
        val recordsLast30Days = allRecords.filter { it.timestamp
>= thirtyDaysAgoTimestamp }
        val totalCaffeine30Days = recordsLast30Days.sumOf {
it.caffeineAmountMg }
        val distinctDays30 = recordsLast30Days.map {
```

```
            val cal = Calendar.getInstance()
            cal.timeInMillis = it.timestamp
            cal.set(Calendar.HOUR_OF_DAY, 0)
            cal.set(Calendar.MINUTE, 0)
            cal.set(Calendar.SECOND, 0)
            cal.set(Calendar.MILLISECOND, 0)
            cal.timeInMillis
        }.distinct().count()
        val average30Days = if (distinctDays30 > 0)
totalCaffeine30Days.toDouble() / distinctDays30 else 0.0
        binding.tvAverageIntake30Days.text =
String.format(Locale.getDefault(), "Середнє за 30 днів:
%.1f мг/день", average30Days)
    }

    // Функції діалогів, адаптовані з TodayFragment
    private fun
showEditOrAddDrinkDialogAnalytics(recordToEdit:
CaffeineRecord?) { // recordToEdit не може бути null тут
        if (recordToEdit == null) return // На вкладці
аналітики ми тільки редагуємо

        val dialogBinding =
DialogAddDrinkBinding.inflate(LayoutInflater.from(require
Context()))
        val actvDrinkName = dialogBinding.actvDrinkName
        val etCaffeineAmount =
dialogBinding.etCaffeineAmount
        val cbSaveAsCustom = dialogBinding.cbSaveAsCustom
// Не використовуємо

        cbSaveAsCustom.visibility = View.GONE // Завжди
приховано при редагуванні

        if
(!::drinkNameAdapterForDialogAnalytics.isInitialized) {
            // Якщо адаптер ще не створений
            val displayNames =
combinedDrinkListForDialogAnalytics.map { it.first
}.distinct().sorted()
            drinkNameAdapterForDialogAnalytics =
ArrayAdapter(requireContext(),
android.R.layout.simple_dropdown_item_1line,
displayNames)
        }

actvDrinkName.setAdapter(drinkNameAdapterForDialogAn
alytics)

        // Знаходимо повну назву для відображення
        val fullDisplayName =
combinedDrinkListForDialogAnalytics.find {
            it.first.startsWith(recordToEdit.drinkName,
ignoreCase = true) && it.second ==
recordToEdit.caffeineAmountMg
        }?.first ?: "${recordToEdit.drinkName}
(${recordToEdit.caffeineAmountMg} мг)"

        actvDrinkName.setText(fullDisplayName, false)

etCaffeineAmount.setText(recordToEdit.caffeineAmountMg.
toString())

        actvDrinkName.setOnItemClickListener { _, _, position,
_ ->
            val selectedDisplayName =
drinkNameAdapterForDialogAnalytics.getItem(position)
            val selectedDrinkPair =
combinedDrinkListForDialogAnalytics.find { it.first ==
selectedDisplayName }
            selectedDrinkPair?.let {
                etCaffeineAmount.setText(it.second.toString())
            }
        }

        val dialog =
MaterialAlertDialogBuilder(requireContext())
            .setTitle("Редагувати запис")
            .setView(dialogBinding.root)
            .setPositiveButton(R.string.save_button, null)
            .setNegativeButton(R.string.cancel_button, null)
            .create()

        dialog.setOnShowListener {
            val positiveButton =
dialog.getButton(AlertDialog.BUTTON_POSITIVE)
            positiveButton.setOnClickListener {
                var drinkNameFromInput =
actvDrinkName.text.toString().trim()
                val caffeineAmountStr =
etCaffeineAmount.text.toString().trim()

                val pattern = " \\(\\d+(\\.\\d+)? мг\\)$".toRegex()
                val cleanDrinkName = if
(pattern.containsMatchIn(drinkNameFromInput)) {
                    pattern.replace(drinkNameFromInput, "")
                } else {
                    drinkNameFromInput
                }

                if (cleanDrinkName.isBlank() ||
caffeineAmountStr.isBlank()) {
                    Toast.makeText(requireContext(),
getString(R.string.input_error_fill_fields),
Toast.LENGTH_SHORT).show()
                    return@setOnClickListener
                }
                val caffeineAmount =
caffeineAmountStr.toIntOrNull()
                if (caffeineAmount == null || caffeineAmount <= 0)
{
                    Toast.makeText(requireContext(),
getString(R.string.input_error_positive_number),
Toast.LENGTH_SHORT).show()
                    return@setOnClickListener
                }

                val updatedRecord = recordToEdit.copy(
                    drinkName = cleanDrinkName,
                    caffeineAmountMg = caffeineAmount
                )
                viewModel.updateCaffeineRecord(updatedRecord)
                dialog.dismiss()
            }
        }
        dialog.show()
    }
```