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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для планування задач в салоні краси на платформі Node.js з використанням Kotlin»

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

_____ Олександр ДОВБНЯ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Олександр ДОВБНЯ

Керівник: Олена НЕГОДЕНКО
к.т.н., доцент

Рецензент: _____

Київ 2024

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
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Довбні Олександру Володимировичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку для планування задач в салоні краси на платформі Node.js з використанням Kotlin»
керівник кваліфікаційної роботи к.т.н., доцент Олена НЕГОДЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи:
   1. Вимоги до продуктивності та безпеки системи;
   2. Розробка архітектури та бази даних для збереження інформації;
   3. Впровадження технічних обмежень та інтеграційних можливостей.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Опис методів та інструментів, що використовувалися під час дослідження.
   2. Аналіз отриманих результатів та їх відповідності поставленим цілям.
   3. Розроблення рекомендацій щодо створення архітектури застосунку та оптимізація системи на базі Node.js.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Use case diagram.
5. Схема баз даних
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Сучасні інтелектуальні інформаційні технології в науці та освіті	28.02-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих рішень та технологій. Розробка архітектури додатку. Визначення структури бази даних.	14.03-18.03.2024.	
4	Backend (Node.js), Frontend (Kotlin)	19.03-09.04.2024	
5	Підключення фронтенду до бекенду.	10.04-17.04.2024	
6	Аналіз та виправлення виявлених помилок.	11.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Олександр ДОВБНЯ
(підпис)

Керівник
кваліфікаційної роботи _____ Олена НЕГОДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи: 75 сторінок, 44 рисунки, 1 таблиця та 22 джерела.

Об'єктом дослідження – процес планування та управління часом, який виникає в умовах сучасного ритму життя і вимагає систематичного підходу для досягнення оптимальних результатів у професійній та особистій сферах.

Предмет дослідження – функціональні можливості та характеристики програмного забезпечення для планування часу, яке розробляється. Конкретно, досліджуються інструменти управління завданнями, системи пріоритетів, а також методи підвищення ефективності та продуктивності користувача через оптимальне використання часу.

Мета роботи – - створення оптимальних умов для організації робочого та особистого життя користувачів. Головною метою додатку є надання ефективного та зручного інструменту для управління завданнями, встановлення пріоритетів, раціонального розподілу часу та досягнення високих результатів у вирішенні поставлених завдань. Дане програмне забезпечення спрямоване на полегшення процесу прийняття рішень, підвищення продуктивності та ефективності роботи, а також забезпечення більшої гнучкості та контролю над часом, що допоможе користувачам досягати успіху як у професійній, так і особистій сферах життя.

Методи дослідження – для досягнення поставленої мети використовуються методи аналізу вимог користувачів, проектування архітектури програмного забезпечення, програмування з використанням мов програмування Node.js та Kotlin, а також тестування та вдосконалення розроблених компонентів з урахуванням отриманих результатів.

У даній роботі досліджено проблему ефективного планування задач у салоні краси, що стає ключовим аспектом для забезпечення якісного обслуговування клієнтів та оптимізації робочих процесів для персоналу.

На основі досліджень у сфері планування та організації обслуговування в салонах краси виявлено значну потребу у вдосконаленні інструментів для планування задач.

Галузь використання – сфера послуг краси та догляду за собою.

KOTLIN, ПЛАТФОРМА NODE.JS, РОЗРОБКА, АРХІТЕКТУРА, ІНТЕГРАЦІЯ, ПЛАНУВАННЯ, САЛОН КРАСИ

ЗМІСТ

ВСТУП

Актуальність дослідження. Актуальність даного дослідження полягає у пошуку та реалізації ефективних інструментів для планування та організації робочих процесів у салонах краси. Застосування мобільних застосунків у цій галузі може значно полегшити роботу персоналу, забезпечити більшу зручність для клієнтів та підвищити загальну ефективність діяльності салонів краси. Проектування та розробка мобільного застосунку для планування задач у салоні краси на платформі Node.js з використанням Kotlin є важливим кроком у напрямку впровадження інноваційних технологій у цю сферу та може сприяти подальшому розвитку бізнесу у галузі краси та догляду за собою.

У сучасному світі розвиток інформаційних технологій змінює підходи до управління бізнесом у різних галузях. Сфера послуг краси не є винятком, адже популярність салонів краси та студій з косметології постійно зростає, а разом із цим зростає і конкуренція. Забезпечення якісного обслуговування та задоволення потреб клієнтів стає надзвичайно важливим завданням для власників та персоналу салонів краси. У цьому контексті розробка мобільних застосунків для планування та управління процесами у салоні краси стає актуальною проблемою, яка сприяє оптимізації робочих процесів та покращенню якості обслуговування.

Розробка мобільного застосунку на платформі Node.js з використанням Kotlin відкриває широкі можливості для створення високоефективних та функціональних додатків. Використання Node.js дозволяє створювати потужні серверні додатки з високою швидкістю та масштабованістю, а Kotlin надає зручний та безпечний інструментарій для розробки мобільних додатків для платформи Android. Поєднання цих технологій дозволить створити додаток, який буде працювати ефективно, має високу продуктивність та забезпечить зручний інтерфейс користувача для клієнтів та персоналу салону краси.

Об'єкт дослідження – процес планування та управління часом, який виникає в умовах сучасного ритму життя і вимагає систематичного підходу для досягнення оптимальних результатів у професійній та особистій сферах.

Предмет дослідження – функціональні можливості та характеристики програмного забезпечення для планування часу, яке розробляється. Конкретно, досліджуються інструменти управління завданнями, системи пріоритетів, а також методи підвищення ефективності та продуктивності користувача через оптимальне використання часу.

Мета роботи створення оптимальних умов для організації робочого та особистого життя користувачів. Головною метою додатку є надання ефективного та зручного інструменту для управління завданнями, встановлення пріоритетів, раціонального розподілу часу та досягнення високих результатів у вирішенні поставлених завдань. Дане програмне забезпечення спрямоване на полегшення процесу прийняття рішень, підвищення продуктивності та ефективності роботи, а також забезпечення більшої гнучкості та контролю над часом, що допоможе користувачам досягати успіху як у професійній, так і особистій сферах життя.

Наукові завдання:

Провести огляд літератури щодо методів та технік управління часом, тайм-менеджменту, та психології продуктивності.

Визначити основні проблеми та потреби в управлінні часом у сучасному світі.

Провести аналіз існуючих програмних засобів для планування часу та управління завданнями.

Розробити концепцію додатка для планування часу, визначивши основні функціональні можливості та особливості інтерфейсу користувача.

Розробити прототип додатка на основі виявлених вимог та концепції роботи з користувачем.

Провести тестування прототипу з метою виявлення можливих недоліків та отримання фідбеку від користувачів.

Додати необхідні корективи до прототипу на основі результатів тестування та отриманого фідбеку.

1 ТЕОРЕТИЧНІ АСПЕКТИ ПЛАНУВАННЯ ЗАДАЧ В САЛОНІ КРАСИ

1.1 Фактори, що впливають на ефективність планування в салоні краси

Ефективне планування в салоні краси є важливим для забезпечення високої якості обслуговування клієнтів, оптимізації використання ресурсів і підвищення продуктивності роботи. У цьому розділі ми розглянемо ключові фактори, що впливають на ефективність планування в салоні краси, з метою визначення вимог до мобільного застосунку для планування задач.

Управління записами клієнтів є однією з ключових функцій, що впливає на ефективність роботи салону краси. Від того, наскільки зручно та точно ведеться запис клієнтів, залежить рівень їх задоволеності, уникнення конфліктів у розкладі та загальна організованість роботи салону.

Одним з основних завдань є створення та підтримка точного графіку записів. Важливо забезпечити клієнтам можливість запису на прийом через різні канали, такі як веб-сайт або мобільний додаток, що зменшує навантаження на адміністратора та надає клієнтам зручний спосіб запису. Окрім онлайн-запису, обробка записів, зроблених по телефону або при особистому візиті, повинна бути швидкою та точною. Автоматичні нагадування про майбутні прийоми через SMS або електронну пошту допомагають зменшити кількість пропущених візитів.

Облік історії візитів клієнтів є важливим для надання персоналізованих послуг та підвищення рівня задоволеності клієнтів. Це включає збереження персональних даних клієнтів, інформації про всі надані послуги, включаючи дату, тип послуги, використані матеріали та майстра, який виконував роботу. Додаткові нотатки, такі як алергії на певні матеріали або вподобання щодо стилю обслуговування, також повинні зберігатися для підвищення якості обслуговування.

Ефективне управління записами клієнтів неможливе без точного відстеження наявності персоналу. Це включає ведення графіку роботи кожного майстра, враховуючи їх доступність, відпустки, лікарняні та інші відсутності. Важливо забезпечити рівномірний розподіл клієнтів між майстрами, щоб уникнути перевантаження деяких з них та простоїв інших, що сприяє оптимізації роботи салону.

Для уникнення конфліктів у розкладі необхідно враховувати забронюваний час, щоб система автоматично блокувала вже зайняті часові слоти, уникнувши подвійного бронювання. Гнучкість у записах дозволяє легко змінювати або скасовувати записи за необхідності з автоматичним оновленням графіку, що запобігає можливим непорозумінням та забезпечує плавність роботи.

Ефективне управління записами включає також аналіз даних для прийняття обґрунтованих рішень. Звіти про завантаженість майстрів, популярність певних послуг та часових проміжків допомагають керівництву салону краси оптимізувати робочий процес. Оцінка рівня задоволеності клієнтів через збір та аналіз відгуків допомагає виявити сильні та слабкі сторони в наданні послуг, що дозволяє вчасно коригувати стратегію обслуговування.

Оптимізація розкладу персоналу є критично важливою для забезпечення ефективної роботи салону краси. Це включає створення гнучкого та збалансованого розкладу, який враховує потреби клієнтів, доступність майстрів та оптимальне використання ресурсів салону. При розробці мобільного застосунку для планування задач на платформі Node.js з використанням Kotlin можна забезпечити високу ефективність та автоматизацію цього процесу.

Основним завданням є автоматизоване складання розкладу персоналу, яке враховує наявні замовлення, побажання клієнтів та доступність майстрів. Використання Node.js забезпечує швидку обробку даних та високу продуктивність серверної частини застосунку, що дозволяє оперативно створювати та оновлювати розклади. Kotlin, як мова програмування для Android, забезпечує зручність розробки клієнтської частини застосунку, де майстри можуть бачити свій розклад у реальному часі.

Для створення оптимального розкладу важливо враховувати робочі зміни та заплановані відпустки кожного співробітника. В системі, розробленій на Node.js, можна реалізувати модуль для управління графіком роботи персоналу, який дозволяє адміністраторам легко вносити зміни та автоматично коригувати розклад відповідно до відсутностей. Це знижує ризик виникнення конфліктів у графіку та дозволяє підтримувати високу якість обслуговування клієнтів.

Одним з ключових аспектів оптимізації є балансування навантаження між різними майстрами. Застосування алгоритмів машинного навчання на платформі Node.js дозволяє аналізувати історичні дані про завантаженість майстрів та прогнозувати майбутні навантаження. Це дозволяє рівномірно розподілити клієнтів між співробітниками, запобігаючи перевантаженню деяких майстрів та простоїв інших. Kotlin забезпечує інтерактивність мобільного додатку, де майстри можуть бачити свою поточну та майбутню завантаженість.

Гнучкість у зміні розкладу є важливою для адаптації до непередбачених змін, таких як термінові відміни клієнтів або раптові відпустки майстрів. Застосунок, побудований на Node.js з використанням Kotlin, дозволяє адміністраторам та майстрам легко вносити зміни у розклад через зручний інтерфейс. Використання WebSocket в Node.js забезпечує миттєве оновлення даних у режимі реального часу, що дозволяє всім користувачам завжди бачити актуальну інформацію.

Оптимізація розкладу потребує регулярного аналізу даних та оцінки ефективності. Застосунок на платформі Node.js може генерувати детальні звіти про завантаженість майстрів, кількість обслугованих клієнтів та інші ключові показники ефективності. Використання бібліотек для візуалізації даних, таких як D3.js, дозволяє створювати інформативні графіки та діаграми, які допомагають адміністраторам приймати обґрунтовані рішення щодо оптимізації розкладу. Kotlin забезпечує інтеграцію цих звітів у мобільний додаток, роблячи їх доступними для перегляду в будь-який момент.

Управління ресурсами та матеріалами в салоні краси є важливим аспектом, що впливає на загальну ефективність його роботи. Це включає в себе контроль за запасами, планування закупівель та раціональне використання матеріалів.

Застосування сучасних технологій, таких як платформа Node.js у поєднанні з Kotlin, дозволяє автоматизувати ці процеси, підвищуючи точність і зменшуючи втрати.

Система на базі Node.js може бути розроблена для автоматизованого обліку запасів. Використовуючи Kotlin для написання клієнтської частини мобільного застосунку, можна створити зручний інтерфейс для персоналу салону, який дозволить швидко і легко вводити дані про витрати матеріалів та залишки. Це забезпечить актуальну інформацію про стан запасів у режимі реального часу, що сприятиме своєчасному поповненню необхідних матеріалів і уникненню їх дефіциту.

Ефективне планування закупівель є ключовим елементом управління матеріалами. Застосування алгоритмів прогнозування попиту, написаних на Node.js, дозволить створити систему, яка автоматично аналізує історичні дані про використання матеріалів і прогнозує майбутні потреби. Зокрема, інтеграція машинного навчання може забезпечити точніші прогнози, враховуючи сезонні коливання та специфіку конкретного салону. Використання Kotlin дозволить реалізувати ці функції у мобільному застосунку, забезпечуючи зручний доступ до інформації та можливість швидко коригувати замовлення.

Застосування сучасних IT-рішень дозволяє не лише автоматизувати облік і планування, але й оптимізувати використання матеріалів. Наприклад, можна розробити модулі для відстеження ефективності використання різних видів продукції, що дозволить виявляти та усувати надлишкові витрати. Інтерфейси, створені на Kotlin, можуть включати графічні елементи для візуалізації даних, що допоможе персоналу краще розуміти, як ефективніше використовувати ресурси.

Ще одним важливим аспектом є інтеграція системи управління матеріалами з постачальниками. Використовуючи API, розроблені на Node.js, можна створити автоматизовані процеси замовлення, які будуть базуватися на актуальних даних про запаси та потреби салону. Це дозволить скоротити час на оформлення замовлень і мінімізувати ймовірність помилок. Мобільний застосунок на Kotlin

забезпечить легкий доступ до функцій замовлення та відстеження статусу постачання.

1.2 Розроблення мобільних додатків між платформами

Розробка мобільних додатків — це процес створення програмного забезпечення, спеціально розробленого для роботи на мобільних пристроях, таких як смартфони, планшети, переносні пристрої та розумні окуляри доповненої реальності.

Нативна розробка — це основна методологія розробки додатків, яку надають постачальники: Google для Android і Apple для iOS. Нативна розробка передбачає використання специфічних для платформи мов, інструментів і комплектів розробки програмного забезпечення (SDK) (рис. 1.1). Розробка Android використовує:

- Kotlin або Java як мови програмування
- Android Studio як інтегроване середовище розробки (IDE)

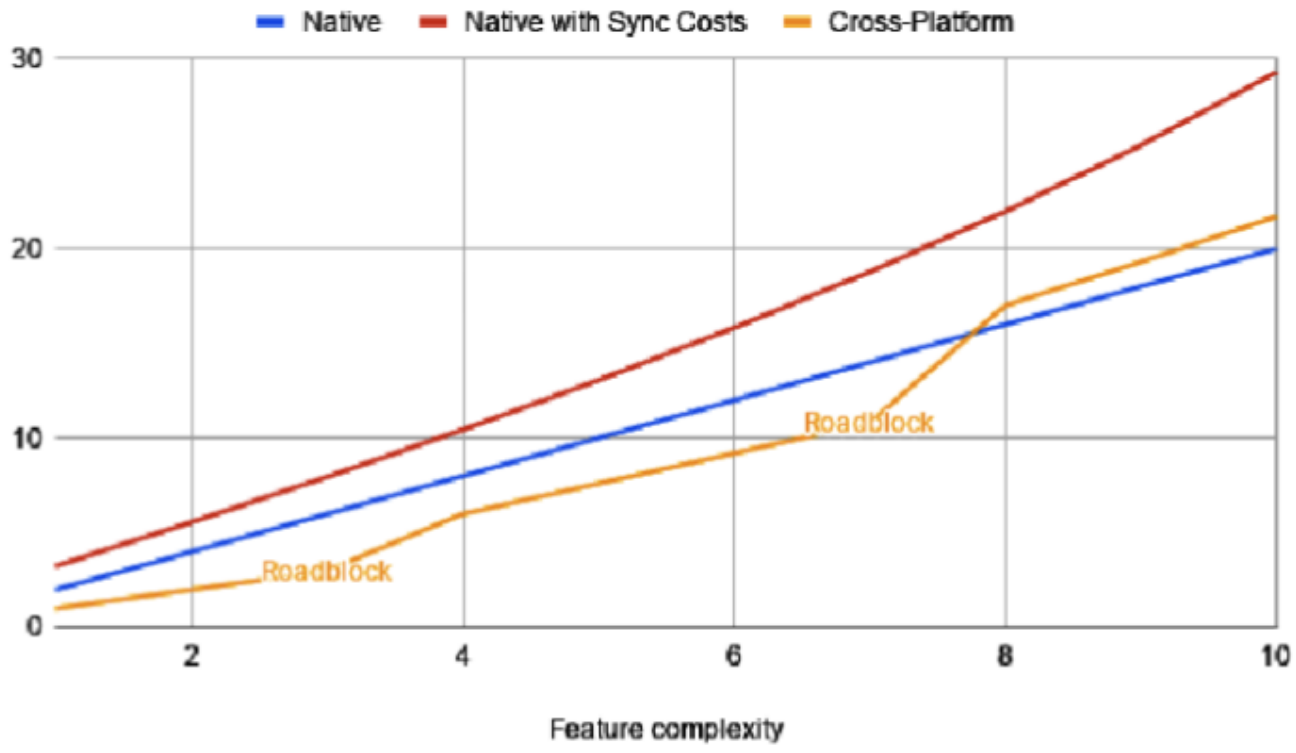


Рис. 1.1 Вартість кросплатформенності

Kotlin виділяється як статично типізована, універсальна мова програмування, відома та цінована своєю лаконічністю, безпекою та практичністю. Він розроблений для повної інтеграції з Java, таким чином забезпечує повну взаємодію з існуючими бібліотеками Java. Однак варто зазначити, що створення Kotlin Multiplatform вплинуло на взаємодію Kotlin і Java.

Kotlin особливо популярний у розробці Android. Щоб зрозуміти важливість у розробці Android, важливо зазначити, що на Google I/O 2019 компанія Google оголосила, що Kotlin стала першою та офіційною мовою програмування для програм Android, фактично замінивши Java (розробники Android, 2024 р.). Java все ще залишається придатним варіантом для створення програм. Однак деякі передові бібліотеки, такі як Jetpack Compose, не підтримують Java, що ще більше поглиблює перехід до Kotlin.

Kotlin Multiplatform дозволяє розробникам ефективно повторно використовувати код на різних платформах, таких як Android, iOS, веб-додатки, настільні та серверні додатки, зберігаючи можливість використовувати власний код, коли це необхідно. Надання розробнику можливості приймати рішення, коли використовувати власний код, є інноваційним підхід, завдяки якому Kotlin Multiplatform виділяється серед більш усталених міжплатформних фреймворків. Замість того, щоб намагатися сприяти розробці, приймаючи рішення щодо конкретної платформи, KMP визнає як необхідність спільного використання впровадження, так і переваги впровадження для конкретної платформи. Таким чином, KMP не надає шару-оболонки для нативних платформ, а намагається бути інструментом, який забезпечує спільний доступ до коду між платформами.

KMP дозволяє розробникам створювати платформно-агностичний код у Kotlin, який потім компілюється за допомогою одного з трьох основних компіляторів екосистеми Kotlin: Kotlin/JVM для Android, сервера та робочого столу, Kotlin/JS для Інтернету та Kotlin/Native для iOS та macOS. Враховуючи це, KMP стає зручним вибором для розробників Android і Kotlin, коли вимоги до проекту виходять за межі однієї платформи. Завдяки цьому підходу ми можна припустити, що взаємодія KMP з екосистемою Android дорівнює нулю, оскільки

Kotlin/JVM є основною частиною екосистеми Android. В інших випадках накладні витрати залежать від продуктивності компілятора.

Одним із обмежень Kotlin Multiplatform є його обмеження щодо використання бібліотек Java під час націлювання на платформи, які потребують іншої компіляції, ніж Kotlin/JVM. Можна припустити, що оскільки Kotlin було розроблено для взаємодії з Java, KMP дозволив би мати посилання на Java у спільному платформно-агностичному модулі. Однак, оскільки інші платформи за своєю суттю не підтримують Java, використання простих бібліотек Java стає неможливим (Надь, 2022). Розробники, які використовують KMP для платформ, відмінних від Android, Desktop і Server, повинні враховувати це під час розробки спільної кодової бази.

Kotlin Multiplatform дотримується модульної конструкції, щоб полегшити обмін кодом. Кожна збірка проекту з Kotlin Multiplatform містить незалежний від платформи спільний модуль на основі Kotlin і кількість індивідуальних модулів для платформи (рис. 1.2).

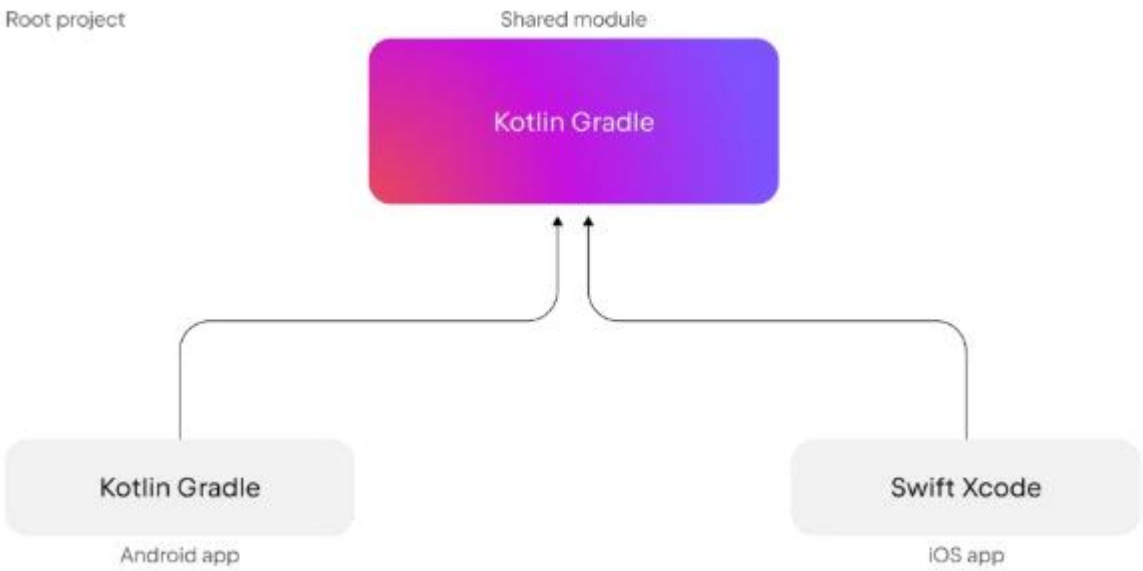


Рис. 1.2 Базова структура проекту

Однією з найважливіших особливостей KMP є механізм очікування-дійсності. Це основний інструмент, коли потрібно написати специфічні для платформи реалізації для використання в спільному модулі. Ідея механізму

очікування-дійсності схожа на інтерфейс із парадигми об’єктно-орієнтованого програмування (рис. 1.3). Загальний вихідний набір спільного модуля вимагає визначення очікуваної декларації, і кожна конкретна платформа має реалізувати її. Компілятор запевняє, що очікуване визначення має свою фактичну реалізацію в кожному з наборів вихідних кодів для конкретної платформи

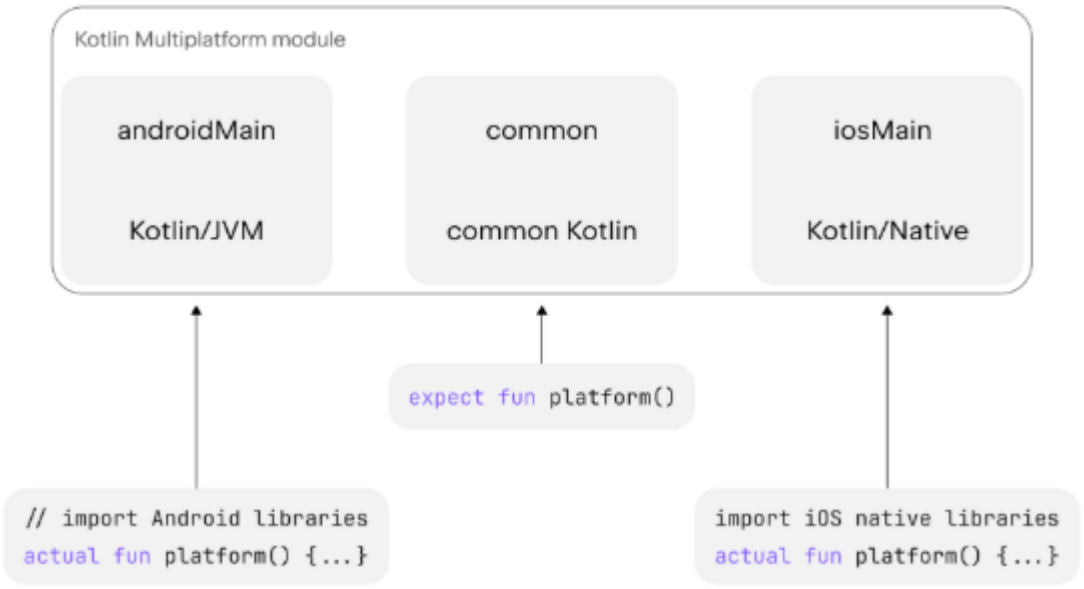


Рис. 1.3 Очікувано-фактичний механізм

Офіційна документація Kotlin Multiplatform Development (2024) рекомендує розробникам використовувати існуючі мультиплатформенні бібліотеки Kotlin, перш ніж поспішати самостійно писати код для конкретної платформи. Ці бібліотеки вже подбали про написання очікувано-фактичного механізму для проблем, які залежать від платформи. Однак деякі з цих бібліотек вимагають від розробників впровадження деяких частин специфічних для платформи API вручну. Таким чином, під час роботи з KMP важливо розуміти систему очікування-дійсність.

1.3 Аналіз та порівняння існуючих програмних засобів для планування часу та управління завданнями

Аналіз та порівняння існуючих програмних засобів для планування часу та управління завданнями може бути корисним етапом при виборі інструменту для планування задач. Для цього можна розглянути такі критерії:

- нагадування про заплановану задачу
- додавання додаткових локацій проведення події
- розподілення задач між собою
- інтеграція з іншими додатками
- зручність у використанні додатку

Результати проведеного аналізу наведені в таблиці 1.1.

Робочий час - це період, протягом якого працівник виконує свої робочі обов'язки або виконує роботу для роботодавця. Він може бути фіксованим, наприклад, з 9:00 до 17:00, або може бути гнучким, коли працівник самостійно встановлює графік роботи.

Управління часом - це процес планування, організації та контролю власного часу з метою ефективного використання його для досягнення поставлених цілей та завдань. Управління часом включає в себе такі аспекти, як складання розкладу, пріоритезація завдань, уникнення відволікань та планування перерв для відновлення енергії.

Керування часом - це ширший термін, який охоплює стратегії, методи та інструменти, що використовуються для управління часом як на робочому місці, так і в особистому житті. Це може включати в себе використання технологій, методів планування, аналізу та вдосконалення ефективності, а також стратегії збереження часу та встановлення пріоритетів. Керування часом спрямоване на досягнення більшої продуктивності, ефективності та зниження стресу.

Jira - це програмний продукт, розроблений компанією Atlassian, призначений для керування проектами та задачами (рис. 1.4). Ця система дозволяє командам ефективно планувати, відстежувати та керувати робочими завданнями, спільно

працювати над проектами, а також вести звітність і аналізувати прогрес виконання завдань. Jira надає широкий набір функцій, включаючи створення та призначення завдань, встановлення термінів виконання, організацію робочих процесів у вигляді канбан-дошок або Scrum-дошок, відстеження часу та багато іншого. Вона є однією з найпопулярніших систем управління проектами у світі технологій і використовується в багатьох компаніях для керування різноманітними видами проектів.

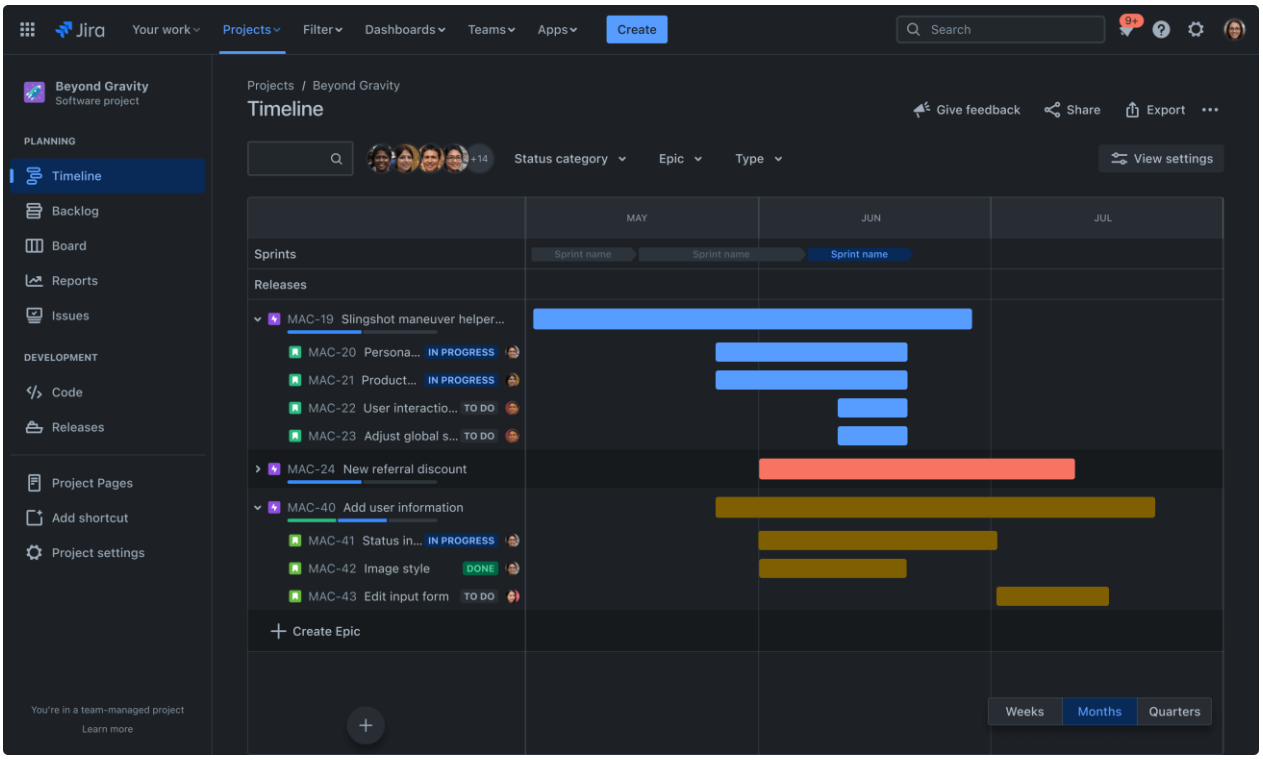


Рис. 1.4 Jira, веб-сервіс

Jira надає широкий набір функціональності для управління проектами та задачами. Основні функції Jira включають:

1. Створення та призначення завдань: Користувачі можуть створювати нові завдання, призначати їх собі або іншим учасникам команди, встановлювати терміни виконання та призначати пріоритети.

2. Організація робочих процесів: Jira дозволяє налаштовувати робочі потоки для відстеження процесу виконання завдань. Це може бути канбан-дошка, Scrum-дошка або інший тип робочого потоку, що відповідає потребам команди.

3. Відстеження прогресу: Користувачі можуть відстежувати статус виконання завдань, переглядати прогрес у реальному часі та отримувати сповіщення про зміни.

4. Звітність та аналітика: Jira надає можливості для створення звітів та аналізу продуктивності команди. Це допомагає керівникам отримувати інформацію про робочий процес, виявляти тенденції та приймати обґрунтовані рішення.

5. Інтеграція з іншими інструментами: Jira підтримує інтеграцію з різними іншими інструментами розробки програмного забезпечення, такими як GitHub, Bitbucket, Confluence та інші, що дозволяє зручно обмінюватися даними між різними інструментами та платформами

Ці функції дозволяють командам ефективно керувати проектами, спільно працювати та досягати поставлених цілей.

Todoist - це популярний онлайн-сервіс для управління задачами та списками справ (рис. 1.5). Він дозволяє користувачам створювати, організовувати та відстежувати свої завдання та проекти, надаючи зручний інтерфейс та низку корисних функцій. Користувачі можуть встановлювати терміни виконання завдань, додавати підзадачі, встановлювати пріоритети, використовувати різні кольори та мітки для організації та категоризації задач. Todoist також має синхронізацію з різними пристроями та можливість спільної роботи над проектами для команд.

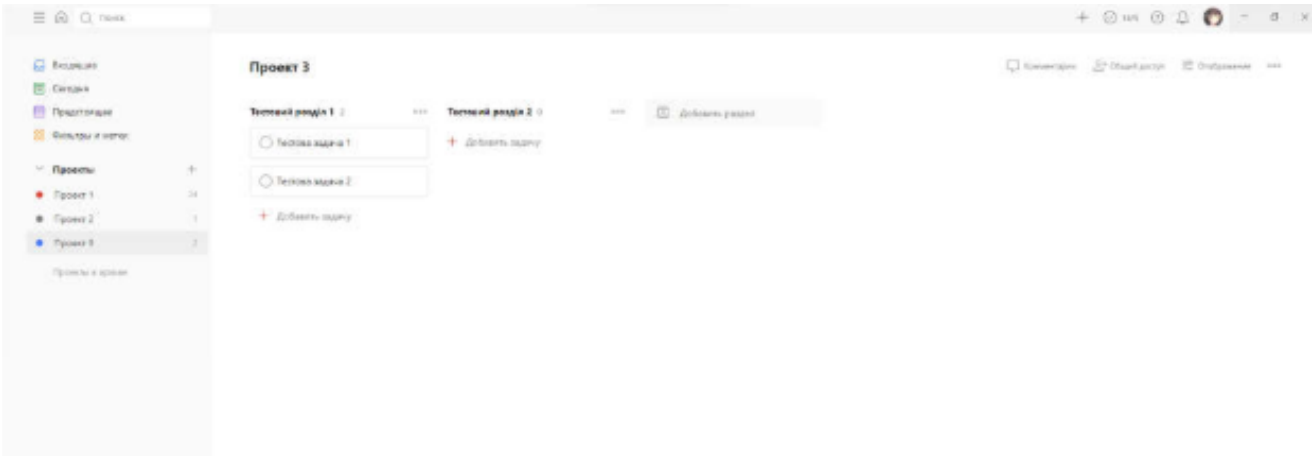


Рис. 1.5 Todoist, веб-сервіс

Основні функції Todoist включають:

1. Створення завдань: Користувачі можуть швидко створювати нові завдання та присвоювати їм опис, терміни виконання та пріоритети.

2. Організація завдань за проектами: Todoist дозволяє користувачам групувати завдання за проектами, що допомагає зберігати їх упорядкованими та легко знаходити потрібні завдання.

3. Планування термінів виконання: Користувачі можуть встановлювати терміни виконання для кожного завдання, щоб визначити, коли вони мають бути завершені.

4. Поділ завдань: Todoist дозволяє розбивати великі завдання на менші підзавдання, щоб полегшити їх виконання та відстежувати прогрес.

5. Нагадування: Користувачі можуть встановлювати нагадування про наближення термінів виконання завдань, щоб не пропустити важливі дедлайни.

6. Планування завдань за пріоритетом: Todoist дозволяє встановлювати пріоритети для кожного завдання, щоб визначити їх важливість та порядок виконання.

7. Інтеграція з іншими сервісами: Todoist підтримує інтеграцію з різними іншими сервісами та інструментами, такими як Google Календар, Dropbox, Slack, яка дозволяє зручно обмінюватися даними та спільно працювати з іншими інструментами.

Загалом, Todoist допомагає користувачам ефективно організовувати свій час, планувати завдання та досягати своїх цілей.

Google Завдання - це безкоштовний інструмент для управління завданнями, що надається компанією Google. Завдяки цьому інструменту користувачі можуть організовувати свої завдання, створювати списки, встановлювати терміни виконання та відстежувати прогрес (рис. 1.6).

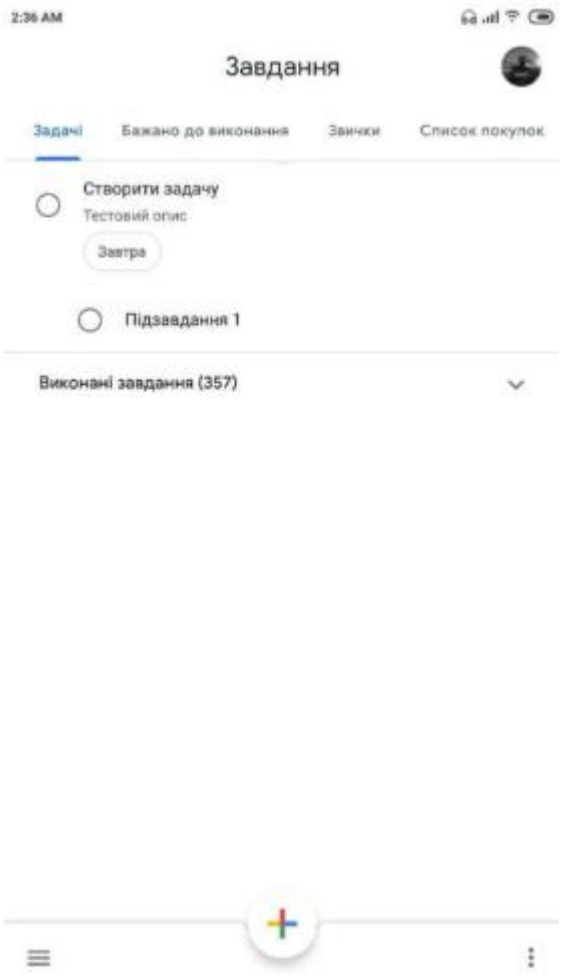


Рис. 1.6 Google Завдання, мобільна версія

Основні функції Google Завдання включають:

1. Створення завдань: Користувачі можуть швидко створювати нові завдання та присвоювати їм назви та опис.

2. Організація за списками: Завдання можна групувати за різними списками, наприклад, "Робочі завдання", "Особисті справи", "Покупки" тощо.

3. Встановлення термінів виконання: Користувачі можуть встановлювати терміни виконання для кожного завдання, щоб визначити, коли вони мають бути завершені.

4. Поділ завдань: Завдання можна розбивати на менші підзавдання для полегшення їх виконання та відстеження прогресу.

5. Нагадування: Користувачі можуть встановлювати нагадування про наближення термінів виконання завдань, щоб не пропустити важливі дедлайни.

6. Інтеграція з іншими сервісами Google: Google Завдання інтегровано з іншими сервісами Google, такими як Gmail, Google Календар, Google Диск, що дозволяє зручно обмінюватися даними та спільно працювати з іншими інструментами.

Google Завдання - це зручний і простий інструмент для організації та ведення списків завдань, який допомагає користувачам ефективно керувати своїм часом та завданнями.

Trello - це інтерактивна онлайн-дошка, яка дозволяє користувачам організовувати свої завдання у вигляді карток на дошці. Вона базується на концепції канбан-дошки, яка спрощує управління завданнями та проектами шляхом візуального відображення їх статусу та переміщенням між колонками.

Він доступний як веб-додаток, а також є мобільні додатки для різних платформ, таких як iOS та Android. Це дозволяє користувачам використовувати Trello на будь-якому пристрої та в будь-якому місці. Він також може бути інтегрований з різними інструментами, такими як Trello, Slack, Jira та Google Drive.

Додаток пропонує безкоштовний план з основними функціями та платний план з розширеними функціями. Платні плани Trello Business Class і Trello Enterprise пропонують додаткові функції, такі як Розширена звітність і обмеження доступу для великих команд і організацій.

Trello базується на системі дощок, списків та карток. Користувачі можуть створити нову панель управління проектом, розмістити на ній список завдань і створити флеш-карту для кожного завдання. Опис картки може містити 10-місячну дату, коментарі, жовтня та інші подробиці. Крім того, Trello має такі функції, як теги, контрольні списки, вкладені списки, публікації та події. Користувачі можуть переміщувати карти між списками, змінювати статус, призначати менеджерів завдань та додавати мітки та кольорові мітки для класифікації. декомунізація.Deadline Trello також підтримує співпрацю, де кілька користувачів

можуть співпрацювати над проектом, додавати коментарі для скасування підписки або відстежувати зміни та оновлення. Це гарна ідея.

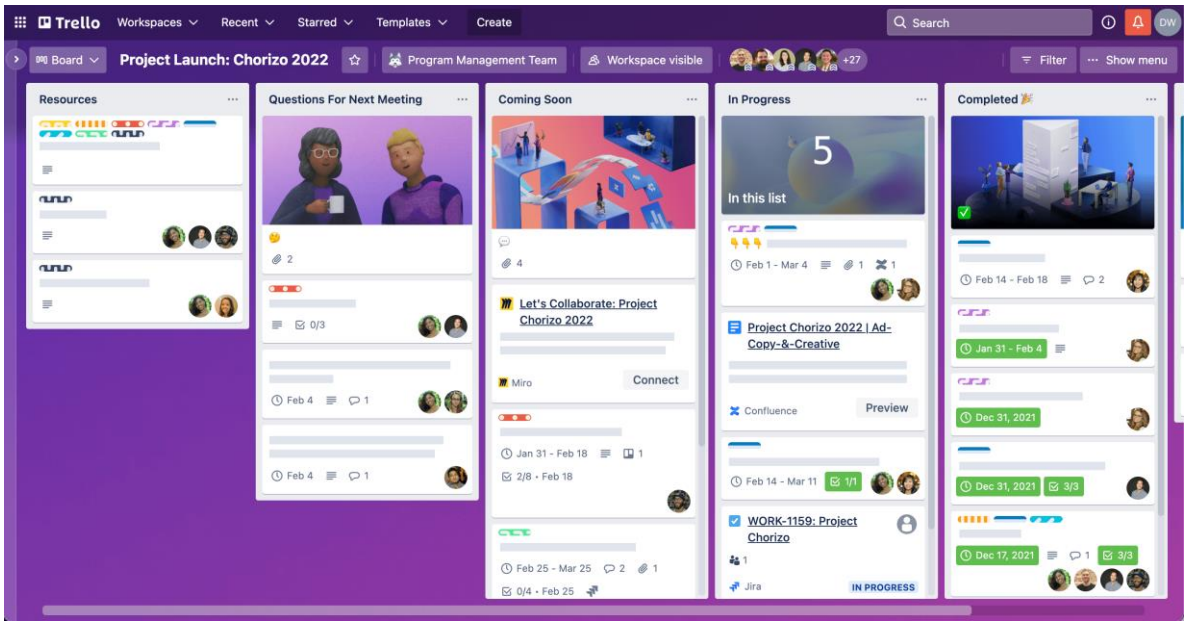


Рис. 1.7 Trello веб-інтерфейс

Основні характеристики Trello включають:

Дошки: Користувачі можуть створювати різні дошки для різних проектів або завдань.

Картки: Кожне завдання або проект представлено у вигляді картки, яку можна розміщувати на дошці.

Списки: Картки можна організовувати у вигляді списків, які представляють різні стадії або категорії завдань.

Пошук та фільтрація: Користувачі можуть швидко знаходити та фільтрувати завдання за різними критеріями.

Коментарі та додатки: Користувачі можуть обговорювати завдання, додавати коментарі та прикріплювати файли до карток.

Ділитися: Користувачі можуть ділитися своїми дошками з колегами або співробітниками, щоб спільно працювати над проектами.

Загалом, Trello є потужним інструментом для управління завданнями та проектами, який дозволяє організовувати роботу в команді, відстежувати прогрес та підвищувати продуктивність.

Таблиця 1.1

Аналіз та порівняння існуючих програмних засобів

	Jira	Todoist	Google Завдання	Trello	Time Management
Нагадування про заплановану задачу	-	-	+	+	+
Додавання додаткових локацій проведення події	-	-	-	-	+
Розподілення задач між собою	Завдяки поміткам чи під-задачами	Є можливість створити проект	Є можливість	Є можливість створити проект	Завдяки власним тегам
Інтеграція з іншими додатками	Присутня, але не з багатьма	Присутня	Присутня, але не з багатьма	Присутня, але не з багатьма	Відсутня

Висновок до розділу 1

Проаналізовано основні фактори, що впливають на ефективність планування в салоні краси, включаючи управління часом, розподіл ресурсів, кваліфікацію персоналу та адаптацію до попиту клієнтів. Визначено, що ретельне планування є ключовим для оптимізації робочих процесів і підвищення загальної продуктивності салону.

Визначено важливість планування задач для забезпечення високої якості обслуговування в салоні краси. Зазначено, що правильне планування дозволяє уникнути перевантажень, скоротити час очікування клієнтів та забезпечити індивідуальний підхід, що сприяє підвищенню задоволення клієнтів.

Розроблено рекомендації щодо покращення планування задач з метою підвищення продуктивності персоналу. Зазначено, що ефективне планування дозволяє рівномірно розподілити робоче навантаження, знизити рівень стресу серед працівників і підвищити їхню мотивацію, що в кінцевому підсумку веде до поліпшення якості обслуговування клієнтів та підвищення конкурентоспроможності салону краси.

2 ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЛАНУВАННЯ ЗАДАЧ В САЛОНІ КРАСИ

У сучасному світі індустрія краси швидко розвивається, що вимагає підвищення ефективності управління салонами краси та поліпшення якості обслуговування клієнтів. Одним із ключових аспектів успішної діяльності салону є ефективне планування задач, яке дозволяє оптимізувати робочий процес, зменшити час очікування для клієнтів та підвищити задоволеність обслуговуванням. У зв'язку з цим, розробка мобільного додатку для планування задач у салоні краси стає актуальною і необхідною.

2.1 Проектування архітектури системи мобільного додатку

Проектування архітектури мобільного додатку для планування задач у салоні краси є ключовим етапом розробки, оскільки від правильності вибору архітектурних рішень залежить стабільність, масштабованість та зручність використання додатку. У цьому розділі розглядаються основні компоненти системи, їх взаємодія, а також технології, які будуть використані для реалізації проекту.

Основні компоненти архітектури включають клієнтську та серверну частини, а також зовнішні сервіси. Клієнтська частина складається з мобільного додатку, який є основним інтерфейсом для користувачів, що надає функції планування, керування записами, перегляду розкладу та іншої інформації. Крім того, важливим аспектом є UI/UX дизайн, що забезпечить зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

Серверна частина включає API сервер, який буде реалізований за допомогою Node.js. Цей сервер відповідатиме за обробку запитів від мобільного додатку, взаємодію з базою даних та іншими зовнішніми сервісами. Для зберігання даних

про клієнтів, записи, розклад майстрів, використані матеріали тощо буде використана реляційна база даних PostgreSQL. Аутентифікація та авторизація будуть реалізовані за допомогою JWT (JSON Web Tokens), що забезпечить безпеку доступу до даних та функцій додатку.

Зовнішні сервіси включатимуть платіжні шлюзи для обробки онлайн-платежів за послуги та СМС і email повідомлення для надсилання нагадувань та повідомлень клієнтам. Взаємодія компонентів системи передбачає, що користувачі мобільного додатку (клієнти салону та співробітники) взаємодіють з додатком, надсилаючи запити до серверної частини через API. API сервер обробляє запити, виконує необхідні операції (наприклад, створення нового запису, оновлення розкладу) та взаємодіє з базою даних для зберігання та отримання інформації. Дані можуть бути отримані або оновлені за запитом від API сервера. Зовнішні сервіси використовуються для обробки платежів та надсилання повідомлень, при цьому API сервер взаємодіє з цими сервісами для виконання відповідних операцій.

Використані технології включають Flutter для кросплатформеного розробки мобільного додатку, Node.js для реалізації API сервера, PostgreSQL для зберігання структурованих даних, JWT для безпечного управління доступом до даних та функцій додатку, а також інтеграцію з платіжними шлюзами та СМС і email сервісами для забезпечення додаткових функцій.

Проектування архітектури системи мобільного додатку для планування задач у салоні краси забезпечує основу для його подальшої розробки та впровадження. Правильний вибір технологій та архітектурних рішень гарантує стабільну роботу додатку, його масштабованість та зручність використання для кінцевих користувачів.

В додатку Time Management користувач буде мати можливість:

– Створення задачі з вихідними даними, мати можливість перегляду статистики по закінченим задачам та змогу створення подій з вхідними даними.

– Реєстрація .

– Вхід за логіном та паролем.

– Відновлення паролю.

– Створення тегів.

На рисунку 2.1 наведено схематичне зображення можливостей користувача.

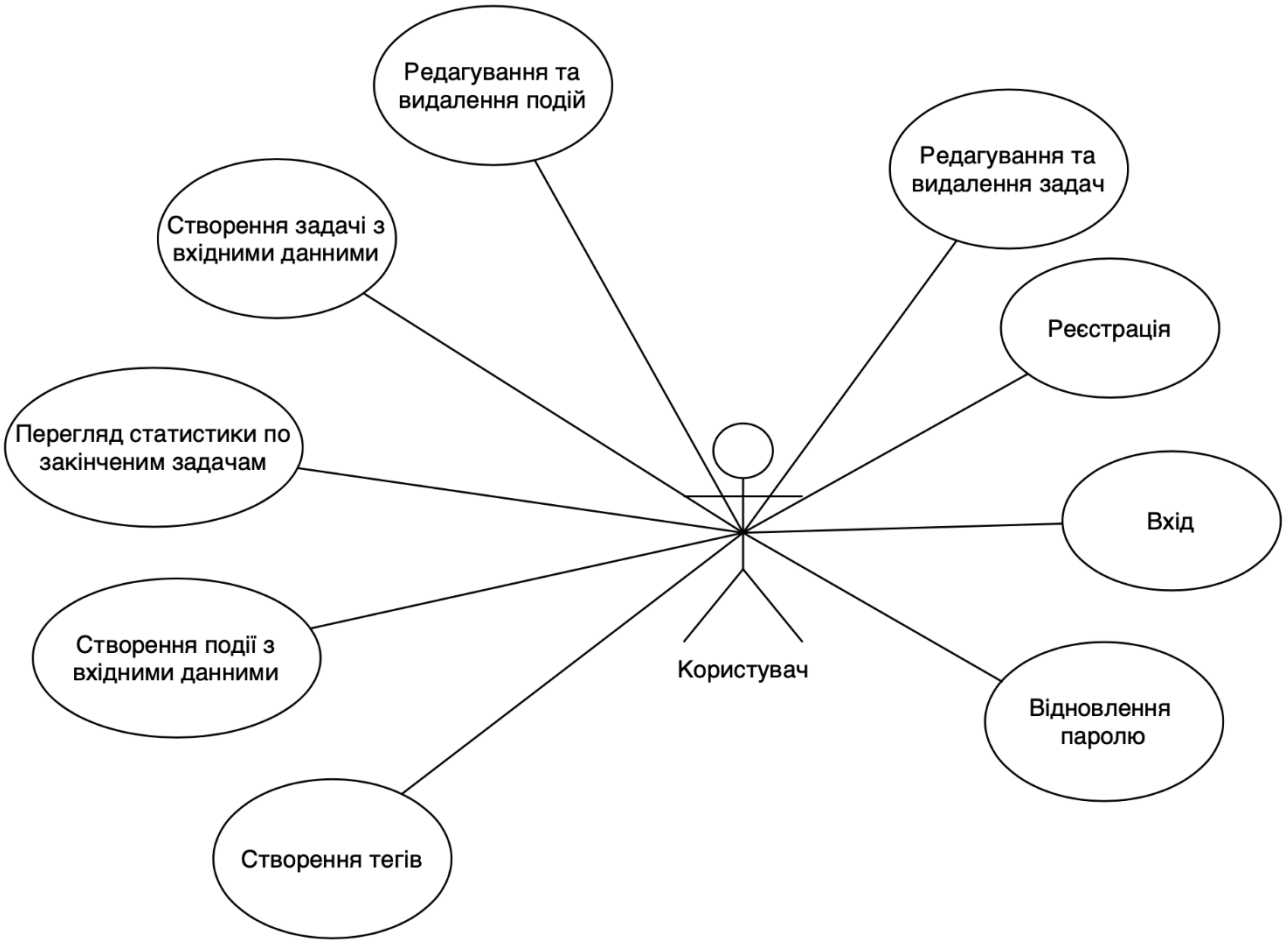


Рис. 2.1 Схематичне зображення можливостей користувача

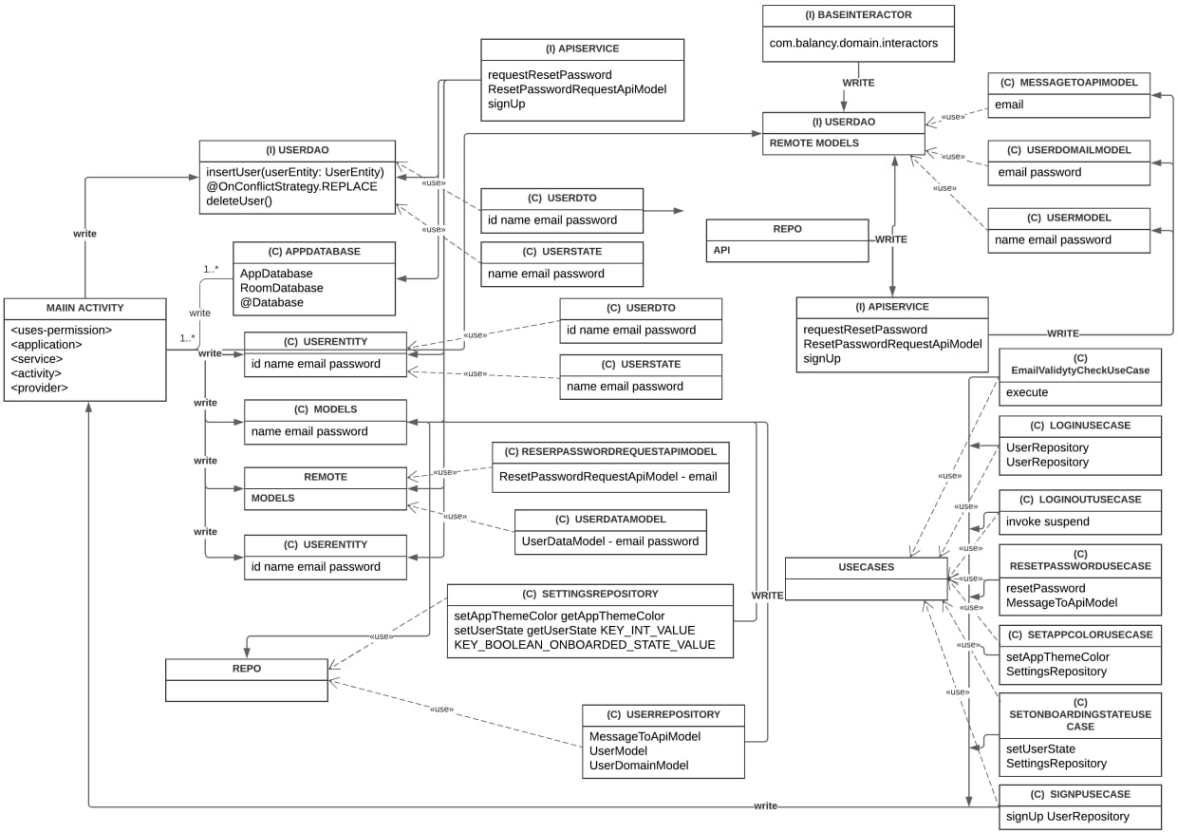


Рис. 2.2 Діаграма класів мобільного застосунку

2.2 Проектування та моделювання вимог до програмного забезпечення

Процес проектування та моделювання вимог до програмного забезпечення є одним з найважливіших етапів у життєвому циклі розробки програмного забезпечення. Від його успішного виконання залежить якість кінцевого продукту, а також його відповідність очікуванням користувачів та замовників.

На першому етапі відбувається збір вимог, який включає в себе взаємодію з майбутніми користувачами, замовниками та іншими зацікавленими сторонами. Основна мета цього етапу – зрозуміти, які функціональні та нефункціональні вимоги мають бути реалізовані в програмному забезпеченні. Використовуються різні методи збору вимог, такі як інтерв'ю, анкети, робочі групи, аналіз документів та спостереження за роботою користувачів.

Зібрані вимоги необхідно ретельно проаналізувати для виявлення можливих суперечностей, дублювань та неоднозначностей. Аналіз вимог дозволяє уточнити

деталі та створити узгоджений набір вимог, які будуть зрозумілими для всіх учасників проекту. На цьому етапі створюються діаграми прецедентів (use case diagrams), які допомагають візуалізувати взаємодію користувачів з системою.

Моделювання вимог передбачає створення різноманітних моделей, які допомагають зрозуміти та описати вимоги до системи. Основними видами моделей є:

1. Функціональні моделі – описують функціональні можливості системи та взаємодію між ними. До таких моделей належать діаграми прецедентів, діаграми активності та діаграми станів.

2. Нефункціональні моделі – описують вимоги, які не стосуються конкретних функцій системи, але є важливими для її роботи. Це можуть бути вимоги до продуктивності, безпеки, надійності та інші.

3. Інформаційні моделі – описують структуру та організацію даних, які використовуються в системі. Це можуть бути діаграми класів, діаграми об'єктів та ER-діаграми.

Після моделювання вимог необхідно провести їх верифікацію та валідизацію. Верифікація вимог полягає в перевірці їх відповідності початковим потребам користувачів та замовників. Валідизація вимог – це процес підтвердження того, що вимоги є правильними та повними, а також можуть бути реалізовані в рамках заданих обмежень.

Завершальним етапом є документування вимог. Створюється специфікація вимог до програмного забезпечення (SRS – Software Requirements Specification), яка містить повний та структурований опис всіх вимог. Цей документ є основою для подальшого проектування, розробки, тестування та впровадження програмного забезпечення.

Проектування та моделювання вимог до програмного забезпечення є складним та багатоетапним процесом, який вимагає уваги до деталей та залучення різних фахівців. Однак, правильне виконання цього процесу є запорукою успішної розробки якісного програмного забезпечення, яке відповідає очікуванням

користувачів та замовників. В таблиці 2.1 наведено категорії та опис до кожної з них.

Таблиця 2.1

Опис атрибутів якості додатку

Категорія	Опис атрибуту якості
Надійність	Працездатність системи якнайдовше.
Продуктивність	Реакція та швидкодія операцій
Масштабованість	Можливість витримувати підвищені навантаження без шкоди для продуктивності системи або здатності швидко розширюватися.
Безпека	Здатність системи запобігати шкідливим або випадковим діям, які виходять за рамки передбачуваного використання.
Зручність використання	Зручність встановлення додатку та доступність в використанні.
Дизайнерські рішення	Впровадження механізмів автоматичного відновлення системи після збоїв.
Зручність в користуванні	Зручність використання визначає, наскільки легко та ефективно користувачі можуть взаємодіяти з системою, швидко виконувати необхідні дії та навчатися роботі з додатком.
Підтримуваність	Підтримуваність забезпечує легкість обслуговування, оновлення та модифікації системи без значних зусиль, що зменшує час простою та витрати на підтримку.
Сумісність	Сумісність системи дозволяє їй інтегруватися з іншими додатками та платформами, забезпечуючи безперебійний обмін даними та розширення функціональності.
Портативність	Портативність дозволяє системі працювати на різних платформах та пристроях з мінімальними змінами, що забезпечує гнучкість та зручність для користувачів.

2.3 Проектування інтерфейсу для мобільного додатку

Прототипування включає в себе процес створення моделі або макета мобільного додатка для перевірки його структури, функціональності та загальної концепції. Це сприяє уникненню непорозумінь між учасниками проекту, а також виявленню та виправленню помилок на ранніх етапах розробки. Такий процес допомагає структурувати думки та зменшити ризик виконання зайвої роботи. Тестування майбутнього додатка за допомогою фокус-групи надає корисну зворотну інформацію від потенційних користувачів. Для цього додатка використано концепцію "швидкого прототипування", що дозволило оперативно створити основні вікна майбутнього програмного продукту. Розробка макету здійснювалася з використанням стандартних інструментів Windows. Основні критерії при розробці включали в себе створення елементів в єдиному стилі, уникнення взаємного пересічення елементів та розробку інтуїтивно-зрозумілої системи навігації. Створені макети дизайну представлені на відповідних рисунках.

Якщо ми звернемося до головного вікна нашого додатку, ще до початку створення, ми можемо побачити декілька типів зображень розташованих в **drawable** представлених нижче:

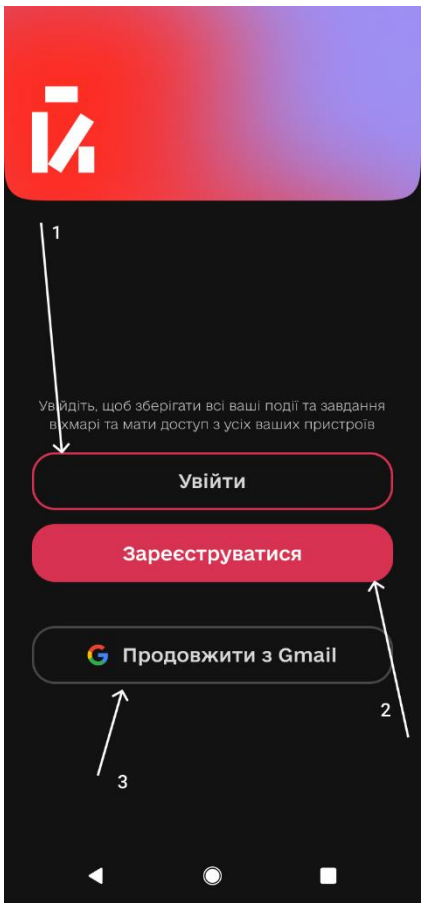


Рис. 2.1 Контекстний логотип додатку

Стрілками та позначеннями до них показані основні елементи екранів мобільного додатку [12].

На рисунку, який представлений вище показано головне activity додатку.

Роз’яснення до деяких елементів наведено нижче:

1. Перше поле відповідає за можливість входу в додаток з аккаунту який вже інсує.

2. Друге поле відповідає за можливість реєстрації в додатку по послідуючим заповненням своїх персональних даних

3. Третє поле дає можливість зареєструватися за допомогою Gmail.

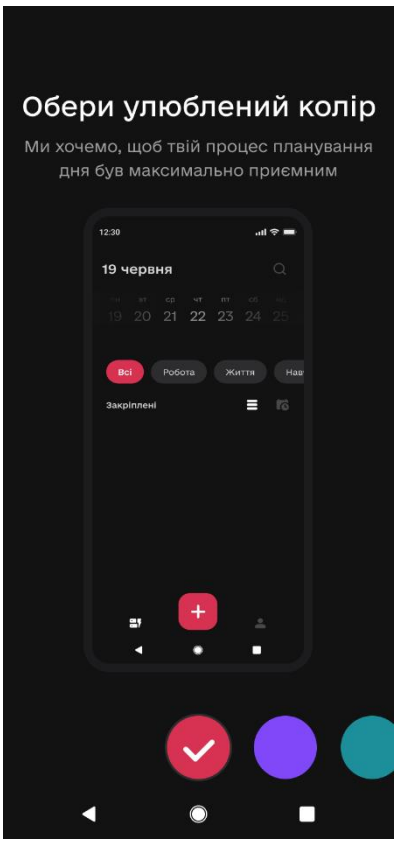


Рис. 2.2 Вибір кольору інтерфейсу головного вікна

Як зазначено на рисунку 2.2 у користувача є можливість самому підібрати колір інтерфейсу з представлених [13]:

Роз’яснення відносно представлених елементів нижче:

1. Користувачу на вибір дається перелік кольорів щоб змінити колір інтерфейсу.
2. При спробі вибрати будь-який варіант, можна зразу переглянути, як буде виглядати додаток та визначитись з кольором [14].

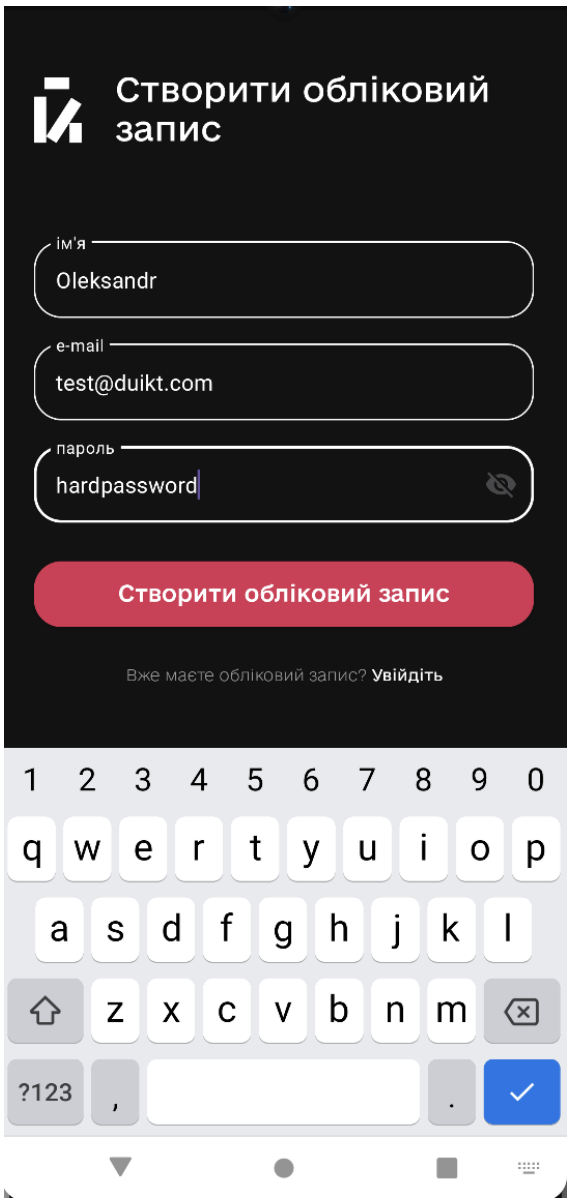


Рис. 2.3 Реєстрація в додаток

Тут можемо побачити, як саме виглядає вікно реєстрації користувача:

- Створення імені
- На яку електронну адресу буду здійснена реєстрація
- Пароль до облікового запису

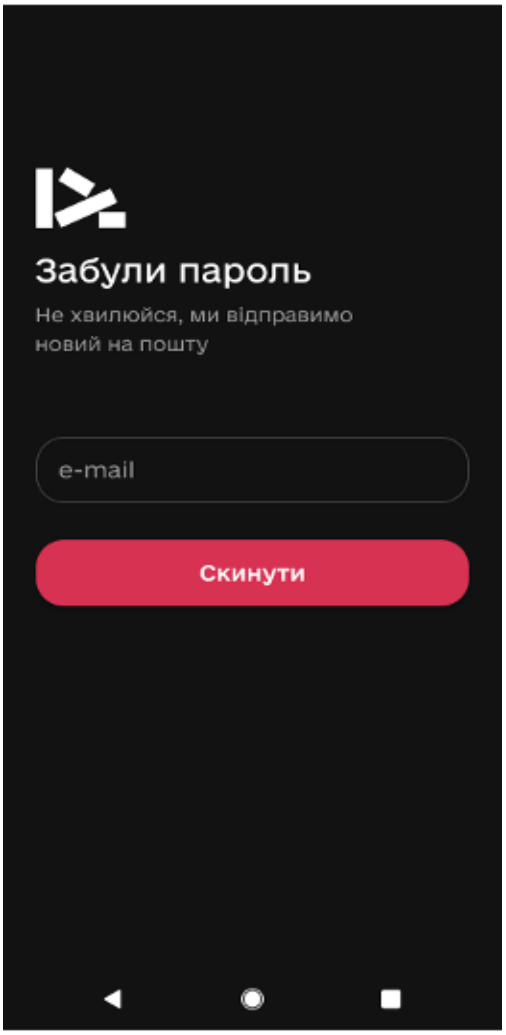


Рис. 2.4 Вікно скидання паролю

На рисунку 2.4 представлені елементи які відповідають за можливість скидання паролю, для відновлення свого облікового запису.

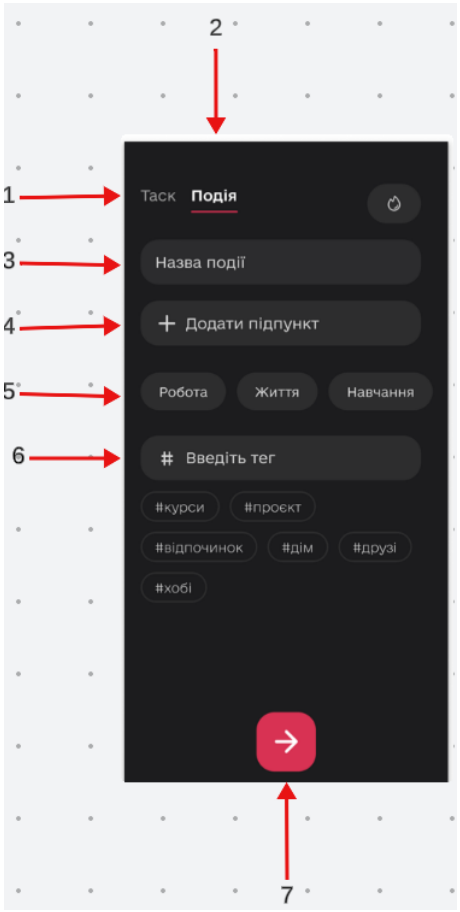


Рис. 2.5 Вікно створення події

На рисунку 2.5 представлене поле яке відповідає за створення таску або необхідної для користувача події.

1. Спочатку користувач повинен вибрати, що він хоче створити (Таск або подія).
2. Задає назву обраному елементу.
3. За бажання може створити підпункт.
4. Приклади, які підпункти зазвичай можна використовувати.
5. Створення тегів для необхідної події.
6. Приклади тегів які зазвичай можна використовувати.
7. Поле продовження створення задачі.

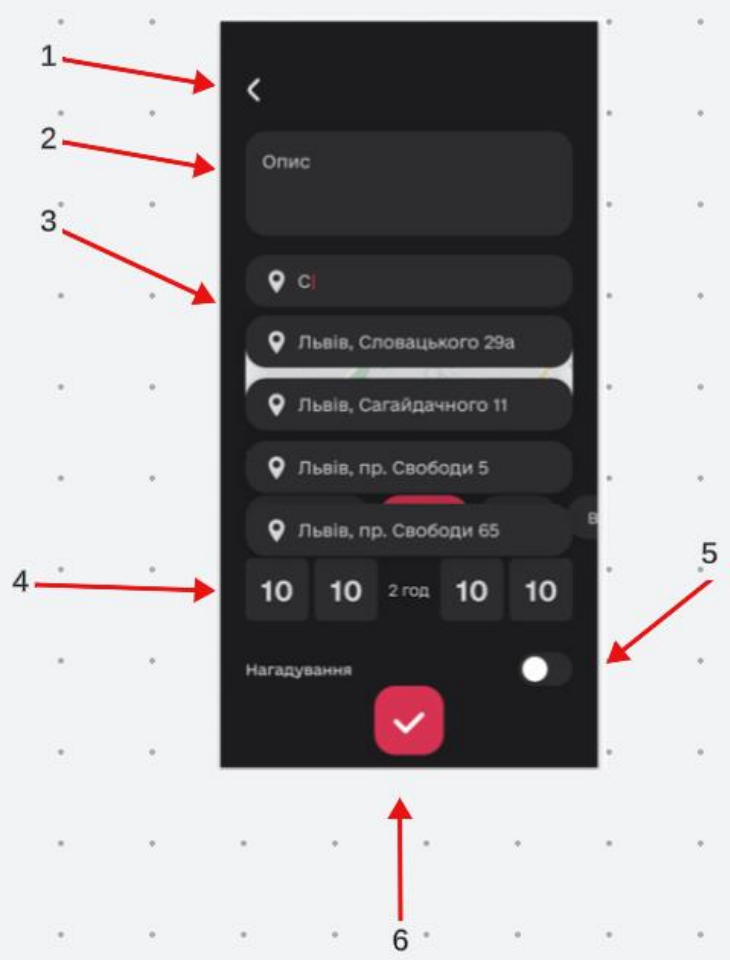


Рис. 2.6 Вікно встановлення параметрів місця та часу

Як можемо побачити що тут у нас є поля які відповідають за встановлення локації де буде відбуватися наша подія та час коли вона відбудеться, за необхідності був створений елемент нагадування, щоб користувач не забувся про подію. Після встановлення цих параметрів подія буде записана.

На рисунку 2.7 показаний розмір логотипу самого додатку (те як він відтворений фронтально).

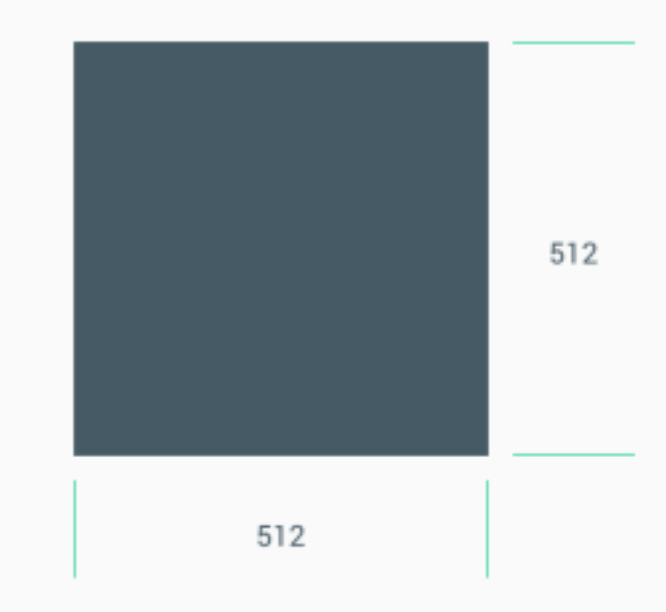


Рис. 2.7 Розмір фронтального елемента

На рис. 2.8 показано контури об'єкта, який лежить всередині квадрата.

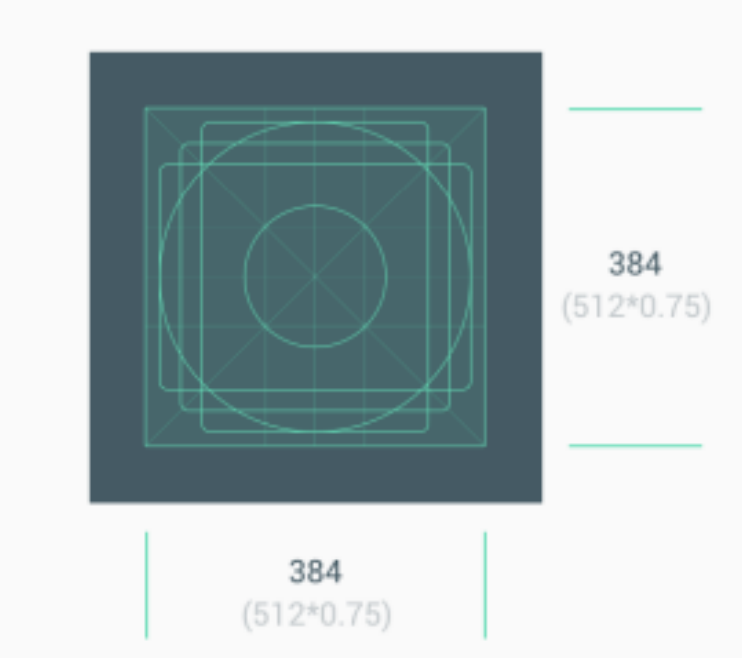


Рис. 2.8 Контур елемента

За основу логотипу на даний момент було вибрано шаблонний варіант Android Studio з мінімалістичним стилем за трьома кольорами: Сірий, білий та зелений.

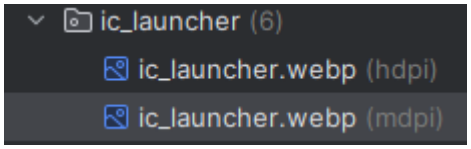


Рис. 2.9 Логотип мобільного додатку в директорії формата webp

Рис. 2.10 Логотип мобільного додатку

Так виглядатиме наш додаток, якщо ми захочемо його передати або завантажити для публічного використання в маркетплейс.

Висновок до розділу 2

Проектування архітектури системи включало визначення основних компонентів та їх взаємодії. Була розроблена багаторівнева архітектура, що забезпечує надійність, масштабованість та безпеку додатку. Важливою складовою стало використання сучасних технологій та підходів, що дозволяють забезпечити високу продуктивність системи та її легку підтримку.

Проектування та моделювання вимог до програмного забезпечення було визначено ключові функції та характеристики мобільного додатку. Зокрема, акцент був зроблений на можливості створення та управління задачами, реєстрації користувачів, входу за логіном та паролем, відновлення паролю та створення тегів. Ці вимоги були задокументовані та проаналізовані для забезпечення їх повного виконання під час розробки додатку.

Проектування інтерфейсу користувача стало важливим етапом, оскільки від зручності та інтуїтивності інтерфейсу залежить задоволеність користувачів. Було створено макети інтерфейсу, що відповідають сучасним стандартам UX/UI дизайну, з урахуванням особливостей використання мобільних пристроїв. Інтерфейс забезпечує простоту навігації, легкість доступу до основних функцій та привабливий зовнішній вигляд.

Проектування мобільного додатку для планування задач в салоні краси створили міцну основу для його подальшої розробки та впровадження. Застосовані методології та підходи дозволяють забезпечити високу якість програмного продукту, його надійність, зручність у використанні та відповідність потребам кінцевих користувачів.

3. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЛАНУВАННЯ ЗАДАЧ В САЛОНІ КРАСИ

3.1 Реалізація роботи програми, вибір технології та розробка архітектури додатку

Огляд процесу розробки додатку Time Management для салону краси.

Розробка мобільного додатку Time Management для салону краси - завдання, що вимагає уважного вибору програмних засобів для досягнення найкращих результатів. У цьому огляді розглянемо, чому TypeScript, Kotlin, Android Studio, TypeORM та PostgreSQL є відмінними інструментами для створення мобільних додатків та як їх інтеграція допомагає забезпечити ефективність та надійність.

Програмні засоби реалізації додатку:

TypeScript є перевагою у розробці мобільних додатків завдяки своїй типізації, що допомагає виявляти помилки на етапі компіляції, зменшуючи кількість можливих проблем у виконанні програми. Його синтаксис, який є розширенням JavaScript, дозволяє швидко створювати інтуїтивно зрозумілі та легкі у розвитку додатки.

Kotlin - це мова програмування, яка стала офіційною мовою розробки для платформи Android. Вона відрізняється відмінною читабельністю коду, безпекою та можливістю використання на різних платформах. Kotlin пропонує ефективнішу альтернативу для розробки мобільних додатків порівняно з Java, забезпечуючи швидкий та надійний додаток.

Android Studio - це інтегроване середовище розробки (IDE) для розробки програм для платформи Android. Воно має широкий набір інструментів для розробки і тестування додатків, включаючи емулятори, дебагери та інші корисні функції, що сприяють швидкій та ефективній розробці.

TypeORM - це ORM (Object-Relational Mapping) для TypeScript та JavaScript, яке дозволяє розробникам працювати з базами даних, використовуючи об'єктно-

орієнтовані принципи. Він надає можливість легко взаємодіяти з базою даних PostgreSQL, спрощуючи роботу зі зберіганням та отриманням даних.

PostgreSQL - це потужна та надійна система керування базами даних, яка використовується для зберігання даних у додатках. Вона відзначається високою швидкодією, надійністю та розширюваністю, що робить її ідеальним вибором для мобільних додатків з великим обсягом даних.

Інтеграція TypeScript, Kotlin, Android Studio, TypeORM та PostgreSQL дозволяє створити потужний та надійний мобільний додаток Time Management для салону краси. TypeScript та Kotlin забезпечують читабельний та безпечний код, Android Studio - зручне середовище розробки, TypeORM - зручне взаємодія з базою даних, а PostgreSQL - надійне зберігання даних. Це дозволяє створити продуктивний та ефективний додаток, який задовольняє потреби клієнтів салонів краси.

Розробка мобільного додатку Time Management для салону краси використовуючи TypeScript, Kotlin, Android Studio, TypeORM та PostgreSQL - це оптимальний вибір, що дозволяє забезпечити якість, надійність та ефективність програмного забезпечення. Цей стек технологій забезпечує розробникам широкі можливості для створення інноваційного та конкурентоспроможного додатку для салонів краси.

Архітектура програмного забезпечення.

Архітектура додатку TimeManagement представлена на рисунку 3.1 та 3.3:

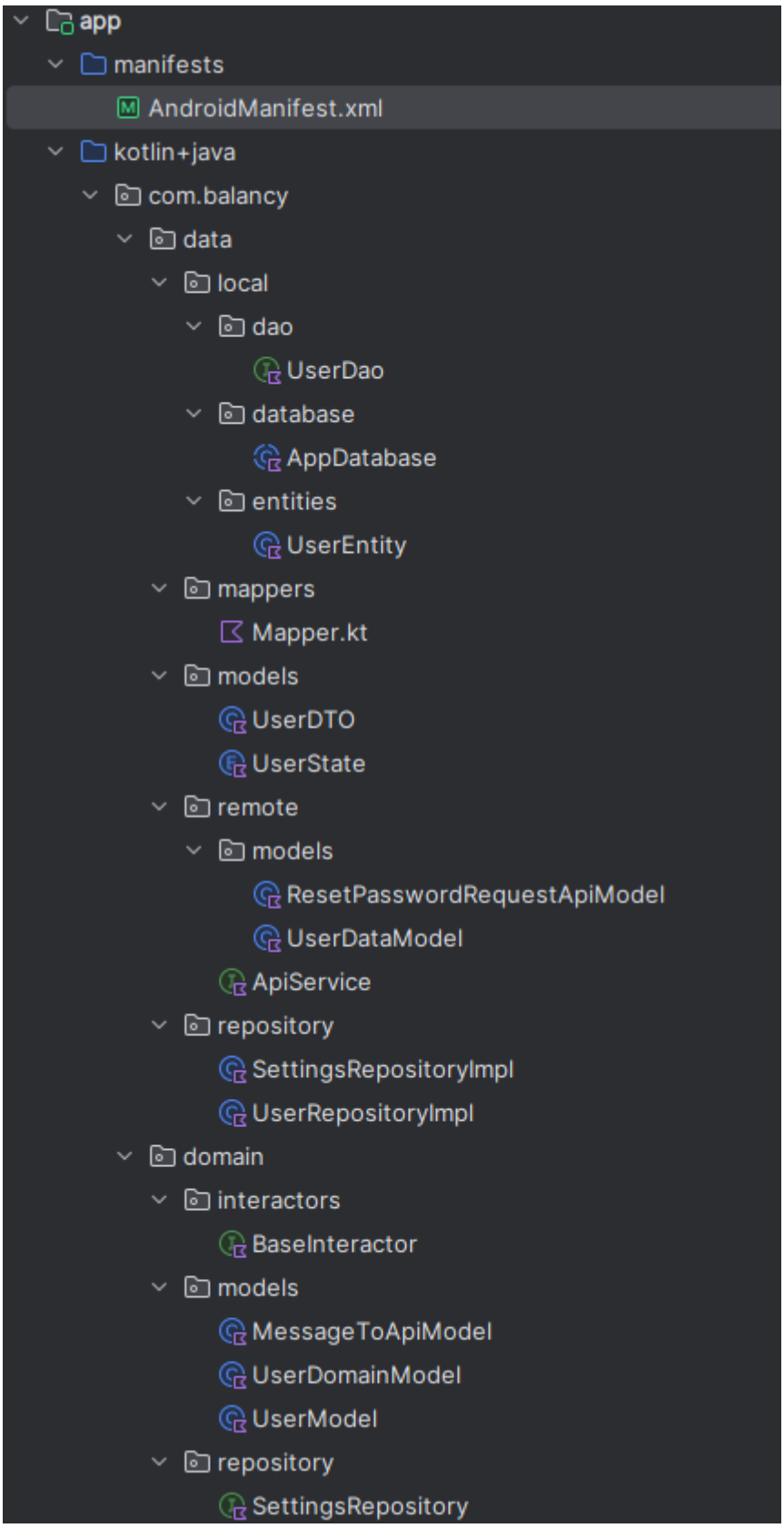


Рис. 3.1 Архітектура розробленого додатку

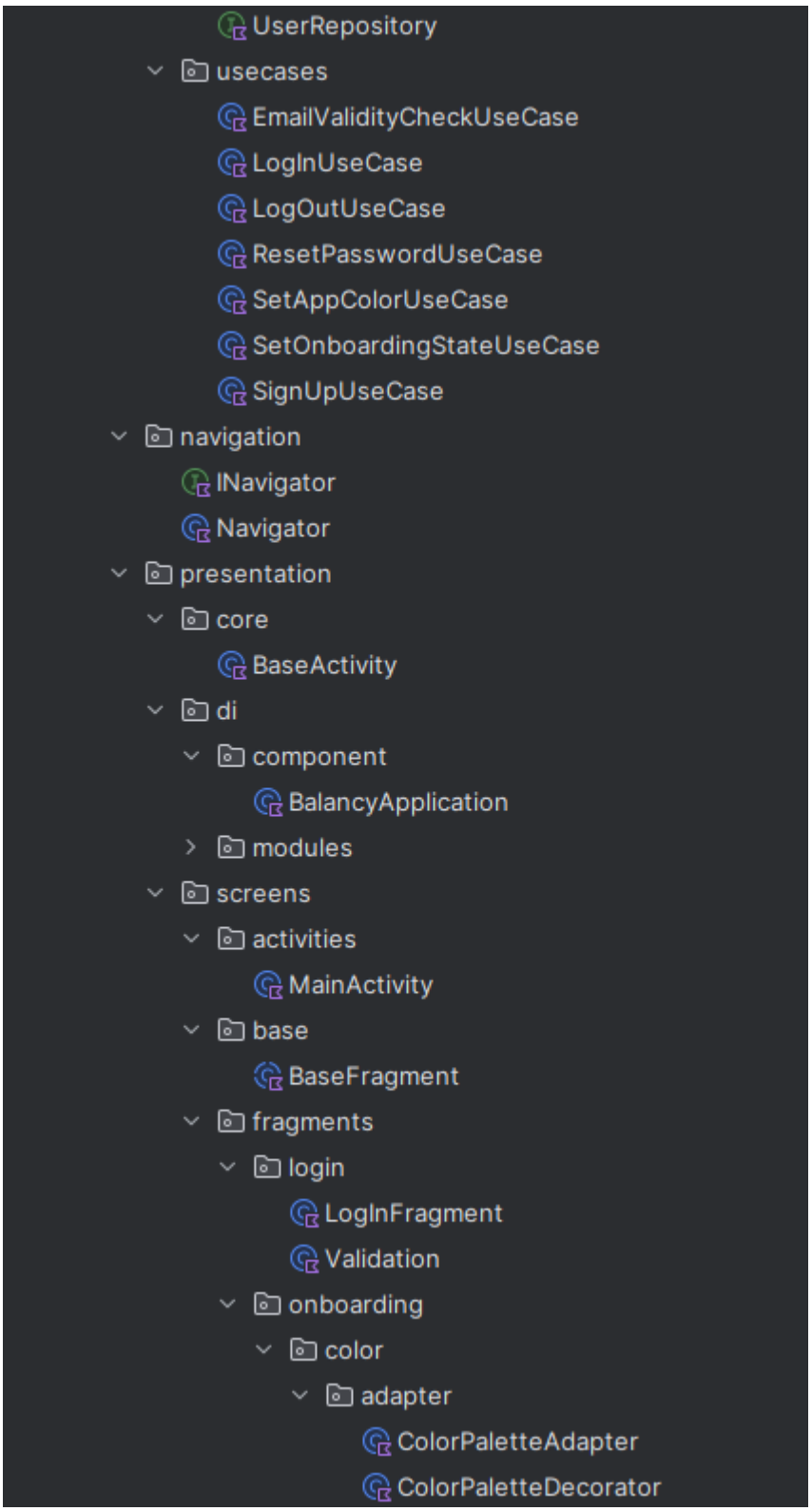


Рис. 3.2 Архітектура розробленого додатку (продовження)

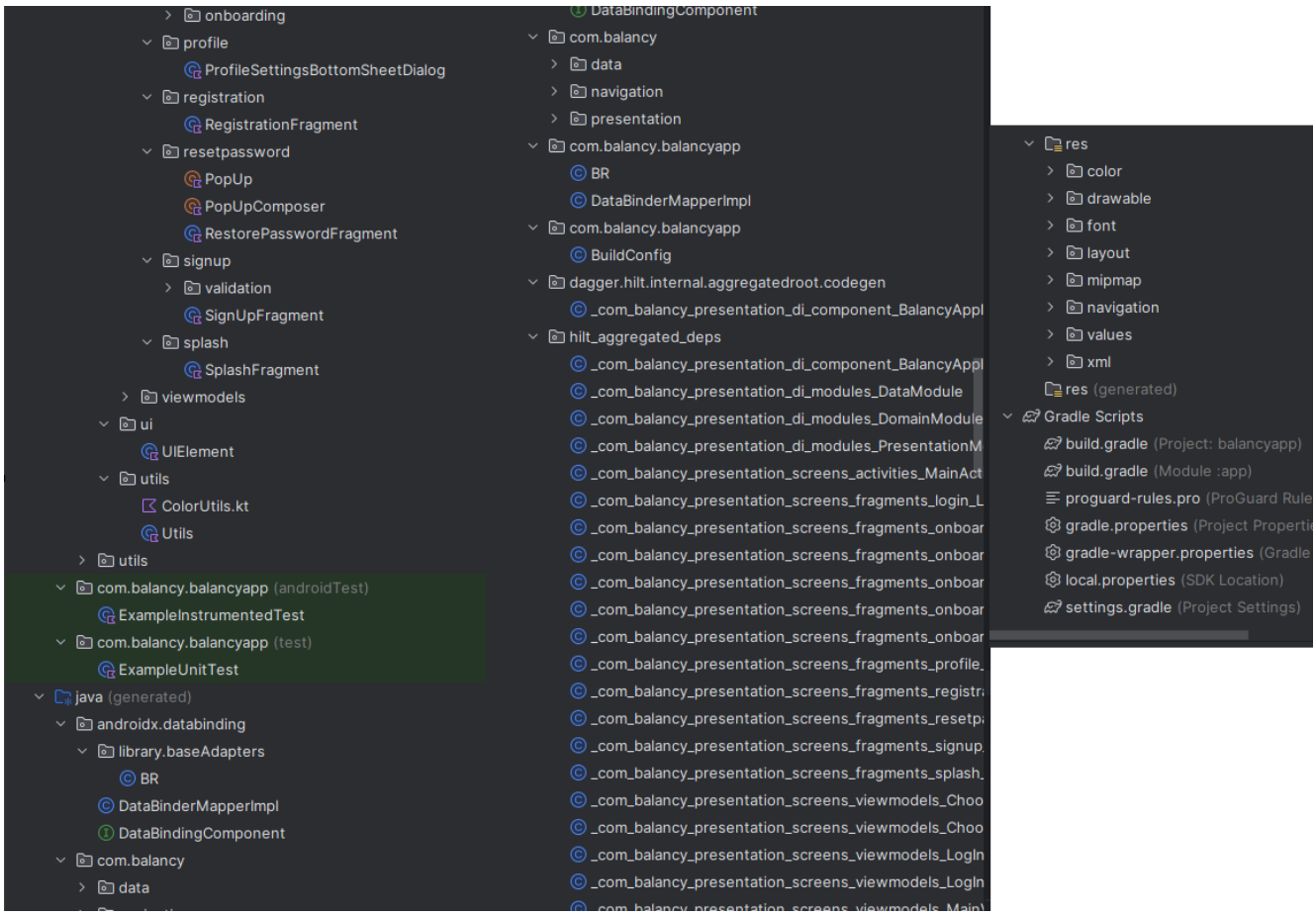


Рис. 3.3 Архітектура розробленого додатку (продовження)

У рамках додатку, як описано у розділі 3.1 та 3.3, присутні такі ключові елементи:

1. AndroidManifest.xml: Цей XML-файл є своєрідним інформаційним центром програми для операційної системи Android. Він містить важливі дані про програму, такі як ім'я Java-пакета, що є унікальним ідентифікатором, опис компонентів програми, перелік необхідних дозволів для доступу до захищених API та взаємодії з іншими додатками. Також вказується мінімальний та максимальний рівень API Android, необхідний для правильної роботи додатку.

Структура AndroidManifest.xml відображена на рисунку 3.4:

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:name="com.balancy.presentation.di.component.BalancyApplication"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:label="balancyapp"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:icon="@mipmap/ic_launcher"
        android:supportsRtl="true"
        android:theme="@style/Balancy.Pink"
        tools:targetApi="31">
        <activity
            android:name="com.balancy.presentation.screens.activities.MainActivity"
            android:exported="true"
            android:fitsSystemWindows="true"
            android:windowSoftInputMode="adjustResize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

Рис. 3.4 Структура AndroidManifest.xml

Опису структури вмісту AndroidManifest.xml.

manifest: Основний елемент, який обгортає весь вміст файлу AndroidManifest.xml. Він вказує на те, що цей файл є дескриптором додатка.

xmlns:android="http://schemas.android.com/apk/res/android": XML-простір імен, який визначає префікс "android" для всіх елементів XML, які належать до Android SDK.

xmlns:tools="http://schemas.android.com/tools": XML-простір імен, який визначає префікс "tools" для інструментів, що використовуються для розробки Android додатків.

<uses-permission android:name="android.permission.INTERNET" />: Цей елемент визначає, що додаток використовує дозвіл на доступ до Інтернету.

<application>: Основний елемент, який містить інформацію про додаток. Він містить атрибути, такі як ім'я додатка, дозволи, теми та інші метадані.

android:name="com.balancy.presentation.di.component.BalancyApplication": Вказує на ім'я класу додатка, який реалізує Application клас.

android:allowBackup="true": Визначає, чи може система автоматично створювати резервні копії додатку.

android:dataExtractionRules="@xml/data_extraction_rules": Вказує на правила для вилучення даних.

android:fullBackupContent="@xml/backup_rules": Вказує на правила резервного копіювання даних.

android:label="@string/app_name": Ім'я додатка, яке відображається на екрані.

android:roundIcon="@mipmap/ic_launcher_round": Іконка додатка, яка відображається на круглій формік на деяких пристроях.

android:icon="@mipmap/ic_launcher": Основна іконка додатка.

android:supportsRtl="true": Вказує на підтримку написання зправа наліво.

android:theme="@style/Balancy.Pink": Вказує на тему додатка.

<activity>: Елемент, який визначає активіті (екран) в додатку.

android:name="com.balancy.presentation.screens.activities.MainActivity": Вказує на ім'я класу головного активіті.

android:exported="true": Визначає, чи може ця активіті бути запущена ззовні додатка.

android:fitsSystemWindows="true": Вказує, чи має активіті займати область під системними панелями.

android:windowSoftInputMode="adjustResize": Вказує, як система повинна адаптувати розмір активіті під клавіатуру при її відкритті.

<intent-filter>: Елемент, який визначає намір активіті.

<action android:name="android.intent.action.MAIN" />: Вказує на те, що ця активіті є головною.

<category android:name="android.intent.category.LAUNCHER" />: Вказує на те, що ця активіті є точкою входу в додаток.

Перейдемо до розгляду основної архітектурної складової нашого додатку:

Інтерфейс DAO (Data Access Object) розроблений для взаємодії з локальною базою даних у застосунку на платформі Android, реалізовану за допомогою Room Persistence Library. Складові інтерфейсу:

- package com.balancy.data.local.dao: Вказує на те, що цей інтерфейс знаходиться в пакеті "com.balancy.data.local.dao".
- @Dao: Аннотація, що позначає цей інтерфейс як DAO. Room використовує цю аннотацію для генерації необхідного коду для взаємодії з базою даних.
- interface UserDao: Оголошує інтерфейс DAO для роботи з базою даних.
- @Insert(onConflict = OnConflictStrategy.REPLACE): Аннотація, що позначає метод як метод для вставки даних в базу даних. Параметр onConflict вказує, яким чином обробляти конфлікти, в даному випадку, якщо зустрінеться конфлікт, відбудеться заміна (REPLACE) даних.
- suspend fun insertUser(userEntity: UserEntity): Метод для вставки користувача (об'єкту типу UserEntity) в базу даних. Оскільки це асинхронна операція, він використовує ключове слово suspend.
- @Query("DELETE FROM user"): Аннотація, що позначає метод як SQL-запит для видалення всіх записів з таблиці "user".
- suspend fun deleteUser(): Метод для видалення всіх користувачів з бази даних. Як і в попередньому випадку, це асинхронна операція, тому використовується ключове слово suspend.

Рис. 3.5 Структура Інтерфейсу DAO

Клас бази даних для локального сховища даних у застосунку на платформі Android, використовуючи Room Persistence Library. Структурний опис класу:

- package com.balancy.data.local.database: Вказує на те, що цей клас знаходиться в пакеті "com.balancy.data.local.database".
- @Database(entities = [UserEntity::class], version = 1, exportSchema = false): Аннотація, що позначає клас як базу даних. Параметр entities вказує на список сутностей (таблиць), які входять до цієї бази даних, version - версію бази даних, а exportSchema - показник, який визначає, чи потрібно експортувати схему бази даних для роботи з нею.
- abstract class AppDatabase: RoomDatabase(): Визначає абстрактний клас бази даних, який розширює RoomDatabase.
- abstract fun userDao(): UserDao: Абстрактний метод, який повертає DAO (Data Access Object) для взаємодії з таблицею користувачів.
- companion object {...}: Об'єкт-компаньйон, який містить константу DATABASE_NAME, яка визначає назву бази даних.

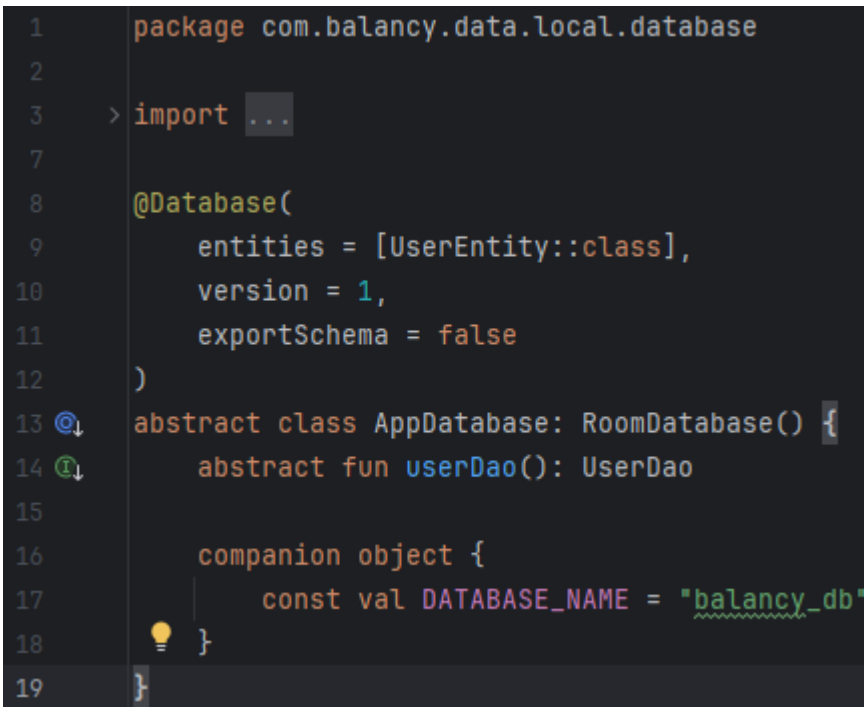


Рис. 3.6 Структура Класу AppDatabase

Сутність (Entity) користувача для локального сховища даних у застосунку на платформі Android, використовуючи Room Persistence Library. Структурний опис:

- package com.balancy.data.local.entities: Вказує на те, що цей клас знаходиться в пакеті "com.balancy.data.local.entities".
- @Entity(tableName = "user"): Аннотація, що позначає клас як сутність бази даних (таблицю) з назвою "user".
- data class UserEntity: Клас даних, що представляє користувача в базі даних.
- @PrimaryKey(autoGenerate = false): Аннотація, що позначає поле "id" як первинний ключ таблиці. Параметр autoGenerate вказує, що значення цього поля не буде генеруватися автоматично, а буде задане вручну. У цьому випадку, значення id завжди дорівнюватиме 1.
- val id: Int = 1: Поле, яке визначає унікальний ідентифікатор користувача.
- val name: String: Поле, яке зберігає ім'я користувача.
- val email: String: Поле, яке зберігає email користувача.
- val password: String: Поле, яке зберігає пароль користувача.

```kotlin
package com.balancy.data.local.entities

import ...

@Entity(tableName = "user")
data class UserEntity(
    @PrimaryKey(autoGenerate = false)
    val id: Int = 1,
    val name: String,
    val email: String,
    val password: String
)
```

Рис. 3.7 Структура сутності користувача

Набір функцій-розміщувачів (mappers), які забезпечують конвертацію об'єктів між різними шарами додатку. Розглянемо структуру:

- MessageToApiModel.mapToResetPasswordRequestApiModel(): Ця функція приймає об'єкт типу MessageToApiModel і конвертує його в об'єкт типу ResetPasswordRequestApiModel, який використовується для запиту на скидання пароля. В об'єкті ResetPasswordRequestApiModel передається лише email користувача.
- UserModel.mapToCreateUserApi(): Ця функція конвертує об'єкт UserModel в об'єкт UserDTO, який використовується для створення нового користувача на сервері. В об'єкті UserDTO передаються ім'я, email та пароль користувача.
- UserModel.mapToUserEntity(): Ця функція конвертує об'єкт UserModel в об'єкт UserEntity, який використовується для зберігання користувача в локальній базі даних. В об'єкті UserEntity передаються ім'я, email та пароль користувача.
- UserDomainModel.mapToUserDataModule(): Ця функція конвертує об'єкт UserDomainModel в об'єкт UserDataModel, який використовується для передачі даних між шарами додатку на віддалений сервер. В об'єкті UserDataModel передаються email та пароль користувача.

– UserDomainModel.mapToUserEntity(): Ця функція конвертує об'єкт UserDomainModel в об'єкт UserEntity для зберігання в локальній базі даних. У цій функції поле "name" задається пустим рядком, оскільки об'єкт UserDomainModel не містить інформації про ім'я користувача.

```
package com.balancy.data.mappers

import ...

fun MessageToApiModel.mapToResetPasswordRequestApiModel(): ResetPasswordRequestApiModel =
    ResetPasswordRequestApiModel(email)

fun UserModel.mapToCreateUserApi(): UserDTO {
    return UserDTO(
        name = name,
        email = email,
        password = password
    )
}
fun UserModel.mapToUserEntity(): UserEntity {
    return UserEntity(
        name = name,
        email = email,
        password = password
    )
}

fun UserDomainModel.mapToUserDataModule(): UserDataModel = UserDataModel(email, password)


fun UserDomainModel.mapToUserEntity(): UserEntity =
    UserEntity(name = " ", email = email, password = password)

```

Рис. 3.8 Структура функцій-розміщувачів

Об'єкти даних (models) UserDTO та UserState:

– UserDTO: Це data class, який представляє об'єкт даних користувача для використання в рівні даних додатку. Він містить поля name, email та password, які представляють ім'я, електронну пошту та пароль користувача відповідно.

– UserState: Це перерахування, яке визначає стани користувача. Воно має три значення: NEW_USER, ONBOARDED_USER та LOGGED_IN_USER, які

представляють стан нового користувача, користувача, який завершив процес введення даних та ввійшов у систему, і користувача, який вже увійшов у систему відповідно.

```kotlin
package com.balancy.data.model

data class UserDTO(
    val name: String,
    val email: String,
    val password: String
)
```

```kotlin
package com.balancy.data.models

enum class UserState {
    NEW_USER,
    ONBOARDED_USER,
    LOGGED_IN_USER
}
```

Рис. 3.9 Об'єкти даних UserDTO та UserState

Об'єкти моделей для взаємодії з віддаленим сервером ResetPasswordRequestApiModel та UserDataModel :

- ResetPasswordRequestApiModel: Це data class, який представляє об'єкт запиту на скидання пароля на віддаленому сервері. Він має одне поле email, яке відображає електронну пошту користувача, для якого потрібно скинути пароль.
- UserDataModel: Це data class, який представляє об'єкт даних користувача для використання на віддаленому сервері. Він має два поля: email і password, які відображають електронну пошту та пароль користувача відповідно.

```kotlin
package com.balancy.data.remote.models

data class ResetPasswordRequestApiModel(val email: String)
```

```kotlin
package com.balancy.data.remote.models

data class UserDataModel(val email: String, val password: String)
```

Рис. 3.10 Об'єкти даних UserDTO та UserState

Реалізація репозиторію на рівні даних для зберігання та отримання налаштувань додатку та стану користувача. Розберемо методи:

- setAppThemeColor(color: Int): Цей метод приймає значення кольору та зберігає його в об'єкті SharedPreferences під ключем KEY_INT_VALUE.
- getAppThemeColor(): Цей метод повертає збережений колір теми додатку, який зберігається в об'єкті SharedPreferences під ключем KEY_INT_VALUE.
- setUserState(state: UserState): Цей метод приймає стан користувача (UserState) та зберігає його в об'єкті SharedPreferences під ключем KEY_BOOLEAN_ONBOARDED_STATE_VALUE. Перед збереженням значення стану користувача перетворюється в ціле число (ordinal).
- getUserState(): Цей метод повертає стан користувача, який зберігається в об'єкті SharedPreferences під ключем KEY_BOOLEAN_ONBOARDED_STATE_VALUE. Значення збереженого стану спочатку отримується як ціле число, а потім конвертується у відповідне значення перерахування UserState за допомогою методу entries.

```
package com.balancy.data.repository

import ...

class SettingsRepositoryImpl @Inject constructor(
    private val sharedPreferences: SharedPreferences
) : SettingsRepository {

    override suspend fun setAppThemeColor(color: Int) {
        sharedPreferences.edit().putInt(KEY_INT_VALUE, color).apply()
    }

    override fun getAppThemeColor(): Int = sharedPreferences.getInt(KEY_INT_VALUE, defValue: 1)
    override suspend fun setUserState(state: UserState) {
        sharedPreferences.edit().putInt(KEY_BOOLEAN_ONBOARDED_STATE_VALUE, state.ordinal).apply()
    }

    override fun getUserState(): UserState = UserState.entries[sharedPreferences.getInt(
        KEY_BOOLEAN_ONBOARDED_STATE_VALUE, UserState.NEW_USER.ordinal
    )]



    companion object {
        const val KEY_INT_VALUE = "theme"
        const val KEY_BOOLEAN_ONBOARDED_STATE_VALUE = "onboarding"
    }
}
```

Рис. 3.11 Структура репозиторію SettingsRepositoryImpl

Реалізацію репозиторію на рівні даних для роботи з користувачами. Предсталені методи:

- resetPassword(messageToApiModel: MessageToApiModel): Цей метод викликається для скидання пароля користувача. Він спочатку викликає функцію mapToResetPasswordRequestApiModel(), щоб перетворити об'єкт MessageToApiModel на об'єкт ResetPasswordRequestApiModel, який використовується для створення запиту на скидання пароля. Потім він виконує запит на скидання пароля через ApiService, використовуючи об'єкт map. Якщо виникає помилка, метод повертає код помилки або повідомлення про помилку.
- signUp(userModel: UserModel): Цей метод викликається для реєстрації нового користувача. Спочатку він викликає функцію mapToCreateUserApi(), щоб перетворити об'єкт UserModel на об'єкт UserDTO, який використовується для відправки запиту на реєстрацію на сервері. Після успішної реєстрації на сервері, користувацький об'єкт також зберігається в локальній базі даних через метод insertUser() в UserDao.
- logIn(userDomainModel: UserDomainModel): Цей метод викликається для входу користувача в систему. Він викликає функцію requestLogIn() через ApiService для виконання запиту на вхід користувача, використовуючи дані користувача, перетворені з об'єкта UserDomainModel на об'єкт UserDataModel. Після успішного входу, дані користувача також зберігаються в локальній базі даних.
- logOut(): Цей метод викликається для виходу користувача з системи. Він видаляє дані користувача з локальної бази даних за допомогою методу deleteUser() в UserDao.

```kotlin
package com.balancy.data.repository

import ...

class UserRepositoryImpl @Inject constructor(
    private val userDao: UserDao,
    private val apiService: ApiService
) : UserRepository {
    override suspend fun resetPassword(messageToApiModel: MessageToApiModel): String {
        val map = messageToApiModel.mapToResetPasswordRequestApiModel()
        return try {
            apiService.requestResetPassword(map)
        } catch (e: retrofit2.HttpException) {
            e.code().toString()
        } catch (e: java.net.UnknownHostException) {
            e.toString()
        }
    }

    override suspend fun signUp(userModel: UserModel) {
        try {
            apiService.signUp(userModel.mapToCreateUserApi())
            userDao.insertUser(userModel.mapToUserEntity())
        } catch (e: retrofit2.HttpException) {
            throw e
        }
    }

    override fun logIn(userDomainModel: UserDomainModel): Flow<String?> = flow {
        val response= apiService.requestLogIn(userDomainModel.mapToUserDataModule())
        userDao.insertUser(userDomainModel.mapToUserEntity())
        emit(response.body())
    }.flowOn(Dispatchers.IO)

    override suspend fun logOut() = userDao.deleteUser()
}
```

Рис. 3.12 Структура репозиторію UserRepositoryImpl

Інтерфейс ***BaseInteractor*** визначає загальні методи або функціональність, які будуть узагальнені для всіх інтеракторів у додатку. Найчастіше інтерактори використовуються для виконання конкретних операцій чи бізнес-логіки, які не підходять або не можуть бути виконані на інших рівнях архітектури, таких як презентери або репозиторії. Ось декілька можливих варіантів для методів цього інтерфейсу:

– execute(): Метод, який виконує основну бізнес-логіку чи операцію, яка повинна бути виконана інтерактором.
– cancel(): Метод, який скасовує виконання операції, якщо вона ще не завершилася.
– observe(): Метод, який дозволяє наблюдати за станом виконання операції, якщо вона виконується асинхронно.
– dispose(): Метод, який звільняє всі ресурси, пов'язані з виконанням операції.

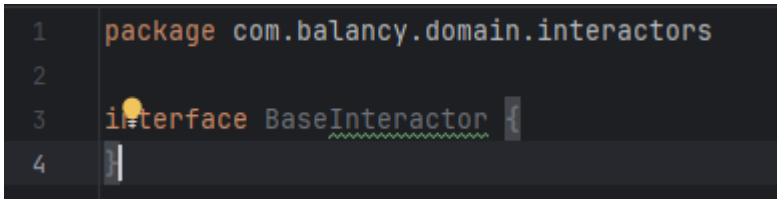


Рис. 3.13 Інтерфейс *BaseInteractor*

Наступний код визначає моделі даних на рівні домену (бізнес-логіки) у додатку. Ось детальний опис кожної моделі:

- MessageToApiModel: Це data class, який представляє об'єкт моделі для використання у взаємодії з API на рівні домену. Він містить поле email, яке відображає електронну адресу, до якої буде відправлено повідомлення.
- UserDomainModel: Це data class, який представляє об'єкт моделі користувача на рівні домену. Він містить поля email і password, які відображають електронну адресу та пароль користувача відповідно.
- UserModel: Це data class, який представляє об'єкт моделі користувача на рівні домену. Він містить поля name, email та password, які відображають ім'я, електронну адресу та пароль користувача відповідно.

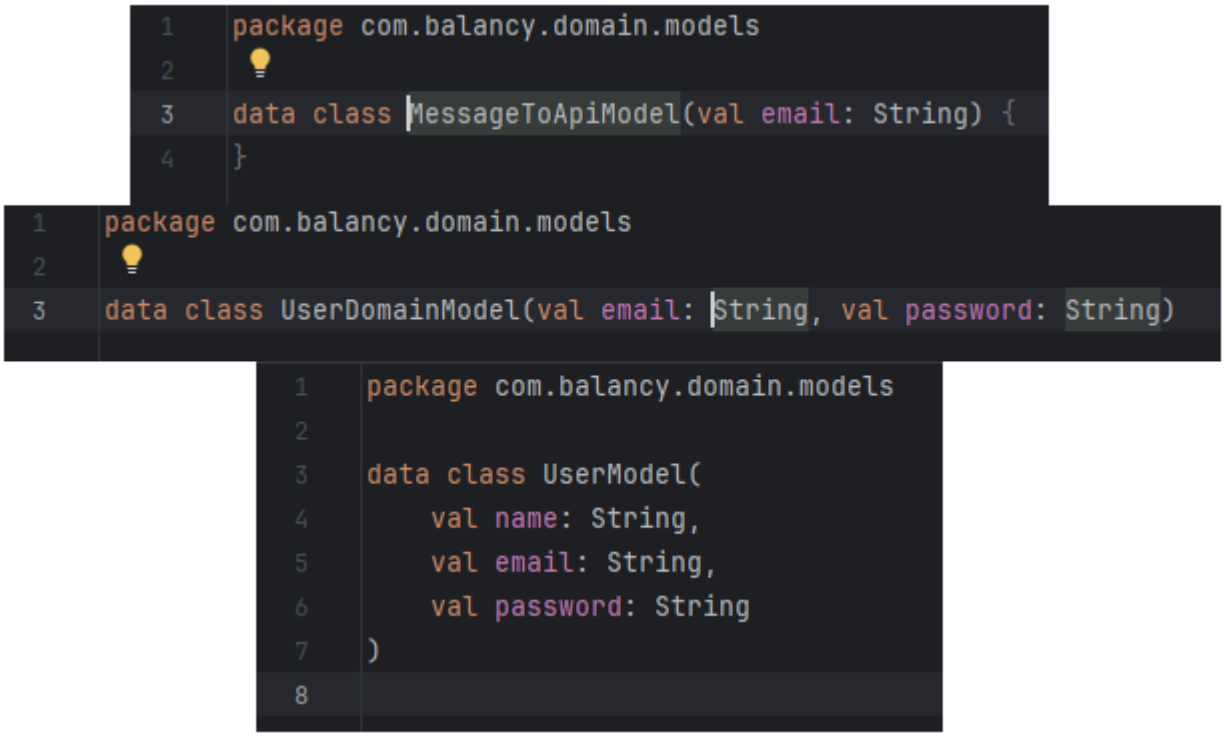


Рис. 3.14 Моделі даних на рівні домену

Інтерфейси SettingsRepository та UserRepository визначають контракти для репозиторіїв на рівні домену, які використовуються для взаємодії з даними та виконання операцій бізнес-логіки. Давайте розглянемо кожен інтерфейс:

1. SettingsRepository: Цей інтерфейс визначає методи для роботи з налаштуваннями додатку, такими як кольори теми та стан користувача.

– setAppThemeColor(color: Int): Метод для встановлення кольору теми додатку.
– getAppThemeColor(): Int: Метод для отримання поточного кольору теми додатку.
– setUserState(state: UserState): Метод для встановлення стану користувача.
– getUserState(): UserState: Метод для отримання поточного стану користувача.

2. UserRepository: Цей інтерфейс визначає методи для роботи з користувачами, такими як реєстрація, вхід, вихід та скидання пароля.

• resetPassword(massageToApiModel: MessageToApiModel): String: Метод для скидання пароля користувача. Повертається рядок з результатом операції (наприклад, код помилки або повідомлення про успіх).

- signUp(userModel: UserModel): Метод для реєстрації нового користувача.
- logIn(userDomainModel: UserDomainModel): Flow<String?>: Метод для входу користувача в систему. Використовується Flow для асинхронного отримання результату входу.
- logOut(): Метод для виходу користувача з системи.

```
package com.balancy.domain.repository

import com.balancy.data.models.UserState


interface SettingsRepository {
    suspend fun setAppThemeColor(color: Int)

    fun getAppThemeColor(): Int

    suspend fun setUserState(state: UserState)
    fun getUserState(): UserState
}
```

```
package com.balancy.domain.repository

import ...

interface UserRepository {
    suspend fun resetPassword(massageToApiModel: MessageToApiM

    suspend fun signUp(userModel: UserModel)

    fun logIn(userDomainModel: UserDomainModel): Flow<String?>

    suspend fun logOut()
}
```

Рис. 3.15 Контракти для репозиторіїв на рівні домену

Ці класи представляють використання в додатку, які виконують конкретні операції бізнес-логіки. Давайте розглянемо кожен із них:

1. EmailValidityCheckUseCase: Цей клас використовується для перевірки правильності формату електронної пошти за допомогою регулярного виразу. Метод `execute()` приймає рядок з електронною поштою та повертає `true`, якщо формат електронної пошти є вірним, та `false` у протилежному випадку.

2. LogInUseCase: Цей клас відповідає за операцію входу користувача в систему. Він викликає метод `logIn()` у `UserRepository`, передаючи об'єкт `UserDomainModel`, і повертає `Flow<String?>`, що представляє результат входу.

3. LogOutUseCase: Цей клас відповідає за операцію виходу користувача з системи. Він викликає метод `logOut()` у `UserRepository`.

4. ResetPasswordUseCase: Цей клас відповідає за операцію скидання пароля користувача. Він викликає метод `resetPassword()` у `UserRepository`, передаючи об'єкт `MessageToApiModel`, та перевіряє, чи результат є успішним (код статусу "201").

5. SetAppColorUseCase: Цей клас відповідає за встановлення кольору теми додатку. Він викликає метод `setAppThemeColor()` у `SettingsRepository`, передаючи кольорове значення.

6. SetOnboardingStateUseCase: Цей клас відповідає за встановлення стану onboarding для користувача. Він викликає метод `setUserState()` у `SettingsRepository`, передаючи стан користувача.

7. SignUpUseCase: Цей клас відповідає за операцію реєстрації нового користувача. Він викликає метод `signUp()` у `UserRepository`, передаючи дані користувача (ім'я, електронна пошта, пароль).

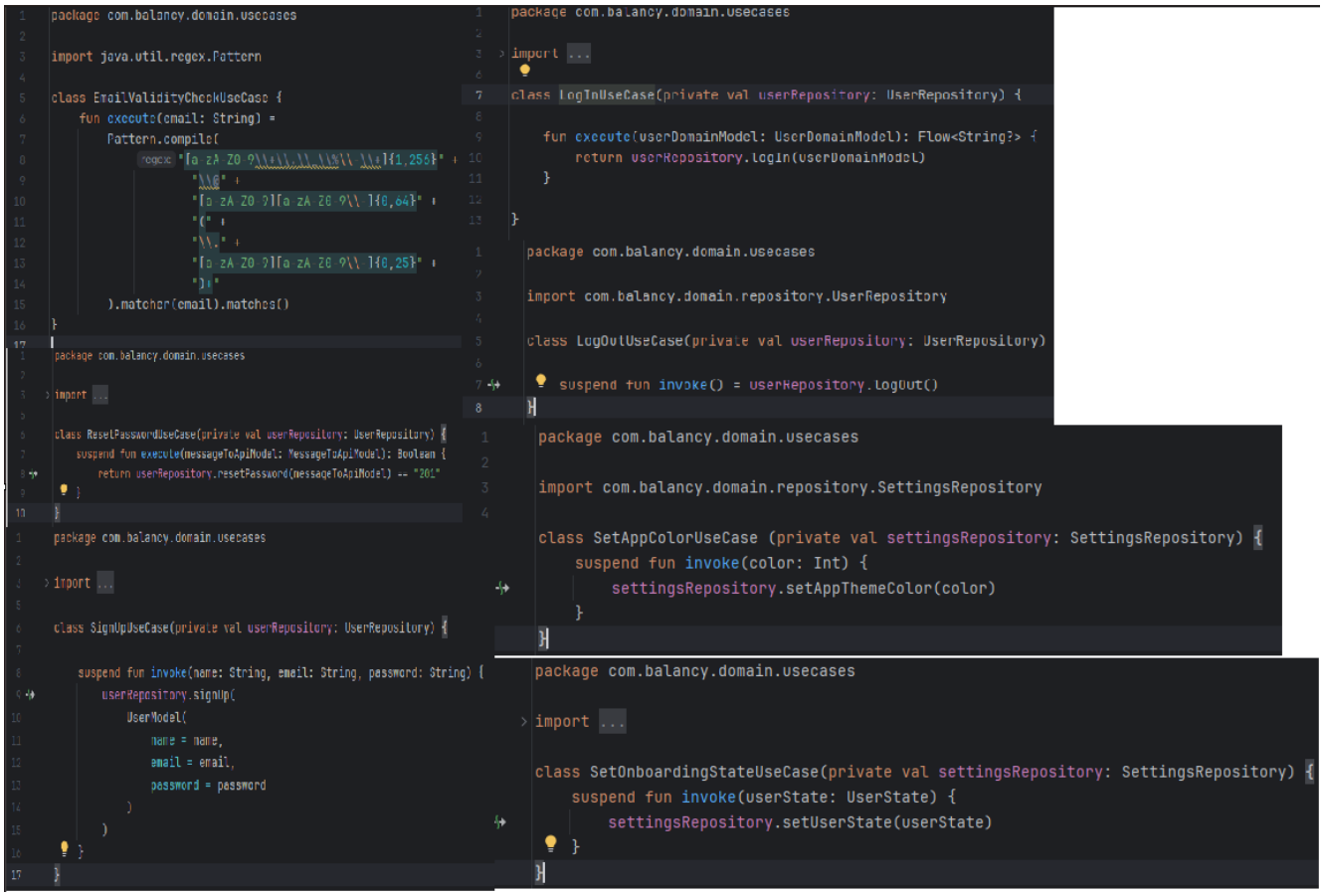


Рис. 3.16 Операції бізнес-логіки

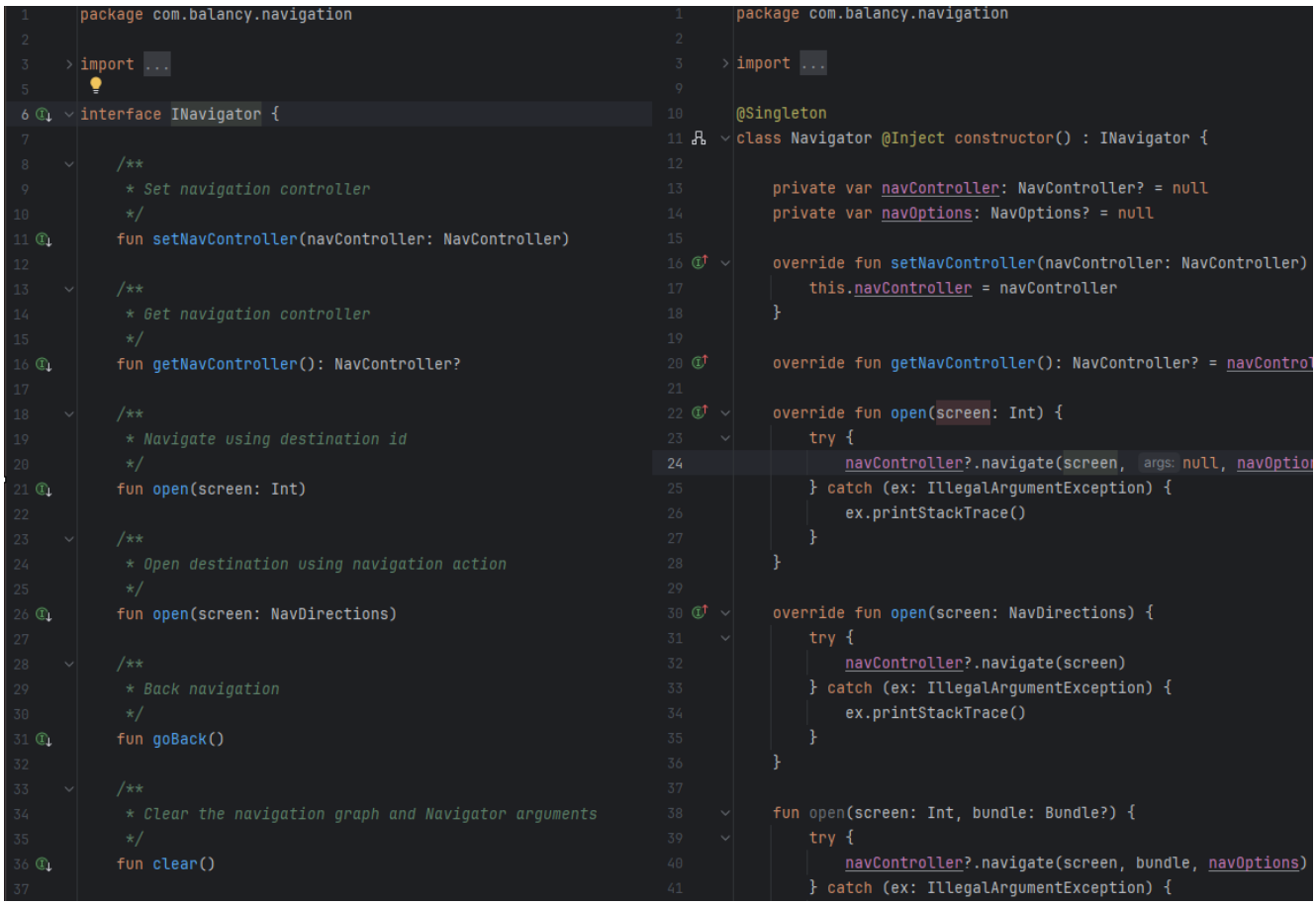
Наступний код визначає навігатор, який використовується для навігації між екранами в додатку. Давайте розглянемо кожний метод інтерфейсу та його реалізацію:

1. INavigator:

- setNavController(navController: NavController): Встановлює контролер навігації.
- getNavController(): NavController?: Отримує поточний контролер навігації.
- open(screen: Int): Відкриває екран за допомогою ідентифікатора призначення.
- open(screen: NavDirections): Відкриває екран за допомогою навігаційної дії.
- goBack(): Виконує навігацію назад.
- clear(): Очищує граф навігації та аргументи навігатора.

2. Navigator:

- setNavController(navController: NavController): Реалізація методу, який встановлює контролер навігації.
- getNavController(): NavController?: Реалізація методу, який повертає поточний контролер навігації.
- open(screen: Int): Реалізація методу, який відкриває екран за допомогою ідентифікатора призначення.
- open(screen: NavDirections): Реалізація методу, який відкриває екран за допомогою навігаційної дії.
- open(screen: Int, bundle: Bundle?): Відкриває екран із переданим пакетом аргументів.
- openAndPopUp(destinationScreen: Int, popUpScreen: Int): Відкриває екран із стеком екранів, який очищається до вказаного екрану.
- goBack(): Реалізація методу, який виконує навігацію назад.
- clear(): Реалізація методу, який очищує граф навігації та аргументи навігатора.



```
package com.balancy.navigation

import ...

interface INavigator {

    /**
     * Set navigation controller
     */
    fun setNavController(navController: NavController)

    /**
     * Get navigation controller
     */
    fun getNavController(): NavController?

    /**
     * Navigate using destination id
     */
    fun open(screen: Int)

    /**
     * Open destination using navigation action
     */
    fun open(screen: NavDirections)

    /**
     * Back navigation
     */
    fun goBack()

    /**
     * Clear the navigation graph and Navigator arguments
     */
    fun clear()
```

```
package com.balancy.navigation

import ...

@Singleton
class Navigator @Inject constructor() : INavigator {

    private var navController: NavController? = null
    private var navOptions: NavOptions? = null

    override fun setNavController(navController: NavController)
        this.navController = navController
    }

    override fun getNavController(): NavController? = navContro

    override fun open(screen: Int) {
        try {
            navController?.navigate(screen, args: null, navOptio
        } catch (ex: IllegalArgumentException) {
            ex.printStackTrace()
        }
    }

    override fun open(screen: NavDirections) {
        try {
            navController?.navigate(screen)
        } catch (ex: IllegalArgumentException) {
            ex.printStackTrace()
        }
    }

    fun open(screen: Int, bundle: Bundle?) {
        try {
            navController?.navigate(screen, bundle, navOptions)
        } catch (ex: IllegalArgumentException) {
```

Рис. 3.17 Навігації між екранами в додатку

BaseActivity зазвичай використовується як базовий клас для всіх активностей у додатку. Його можна розширити, щоб надати загальні функціональність та поведінку для всіх активностей.

- Менеджмент життєвого циклу: Реалізація методів onCreate(), onStart(), onResume(), onPause(), onStop() та onDestroy() для обробки відповідних подій життєвого циклу активності.
- Навігація: Методи для навігації між екранами, такі як navigateTo(destination: Int) чи goBack().
- Обробка помилок та винятків: Методи для відображення повідомлень про помилки або обробки винятків у випадку непередбачуваних ситуацій.
- Взаємодія з іншими компонентами: Методи для взаємодії з фрагментами, сервісами чи іншими компонентами додатку.
- Управління станом: Можливість зберігати та відновлювати стан активності під час зміни конфігурації пристрою.

- Управління дозволами: Обробка запитів на дозвіл у відповідь на дії користувача.
- Загальні налаштування та ініціалізація: Ініціалізація загальних компонентів та ресурсів, таких як інтерфейс користувача, мова, тема.

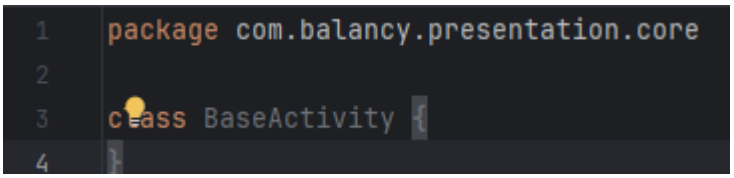


Рис. 3.18 Клас BaseActivity

Компоненти рівня презентації (presentation layer) додатку, такі як активності, фрагменти та класи для валідації. Давайте розглянемо кожну частину окремо:

- BalancyApplication: Цей клас представляє додаток Balancy та використовує анотацію @HiltAndroidApp для інтеграції з Dagger Hilt.
- MainActivity: Основна активність додатка, яка відображає головний екран. Вона використовує viewModels() для створення ViewModel та налаштовує тему додатку за допомогою методів getAppThemeColor() та applyNewThemeStyle().
- BaseFragment: Базовий клас для всіх фрагментів додатка, який забезпечує спільну логіку та функціональність, таку як отримання ViewBinding, установка навігатора та очищення ViewBinding після знищення фрагменту.
- LogInFragment: Фрагмент для екрану входу в систему. Цей фрагмент використовує viewModels() для створення ViewModel та спостерігає за LiveData потоками для обробки подій натискання на кнопки та відображення помилок.
- Validation: Клас, що містить методи для валідації електронної пошти та обробки введення користувача на екрані входу в систему.

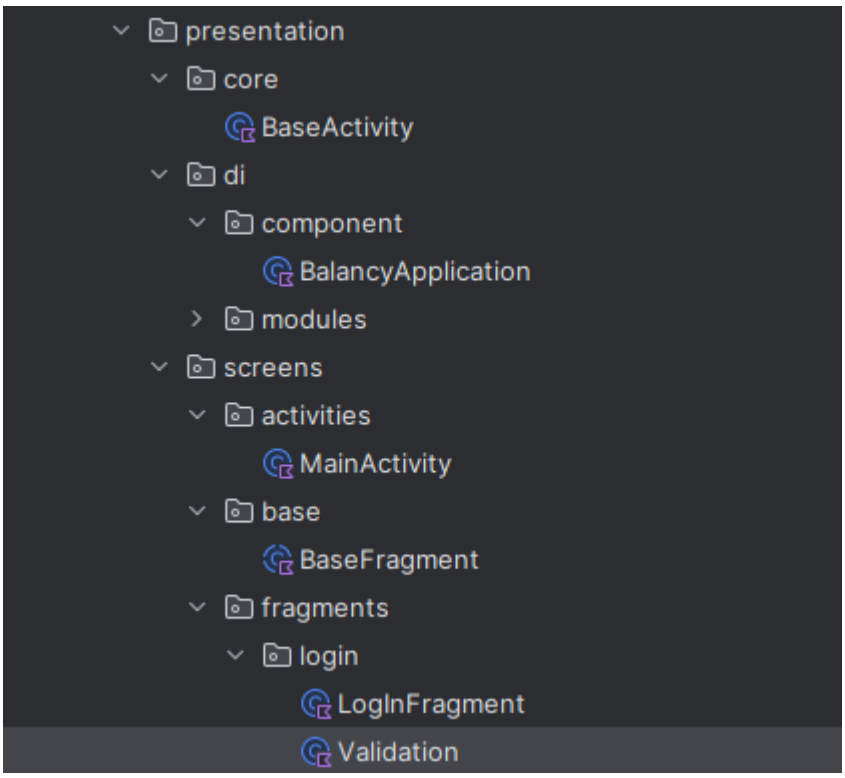


Рис. 3.19 Рівень презентації активностей додатку

Фрагменти, адаптери та моделі, що використовуються в процесі обрання кольору під час налаштування в додатку. Ось опис кожного компонента:

- ColorPaletteDecorator: Цей клас використовується як розкладник для елементів RecyclerView, щоб встановити відступи між елементами.
- ColorPaletteListener: Інтерфейс, який визначає методи для зміни зображення та встановлення кольору з палітри кольорів.
- ColorImageList: Об'єкт, який містить список ресурсів зображень, які будуть використовуватися для представлення кольорових палітр.
- ColorPaletteList: Об'єкт, який містить список об'єктів ColorPalette, що представляють різні кольорові палітри.
- ColorPalette: Модель, яка представляє окрему кольорову палітру.
- ChooseColorFragment: Фрагмент, який відображає палітру кольорів для обрання. Він використовує ColorPaletteAdapter для відображення списку кольорових палітр у RecyclerView. Цей фрагмент також відповідає за навігацію на екран реєстрації після обрання кольору.

– SecondOnboardingFragment та ThirdOnboardingFragment: Фрагменти, які представляють другу та третю сторінки екрану onboarding. Вони використовуються для відображення інформації та вказівок під час проходження першого запуску додатку.

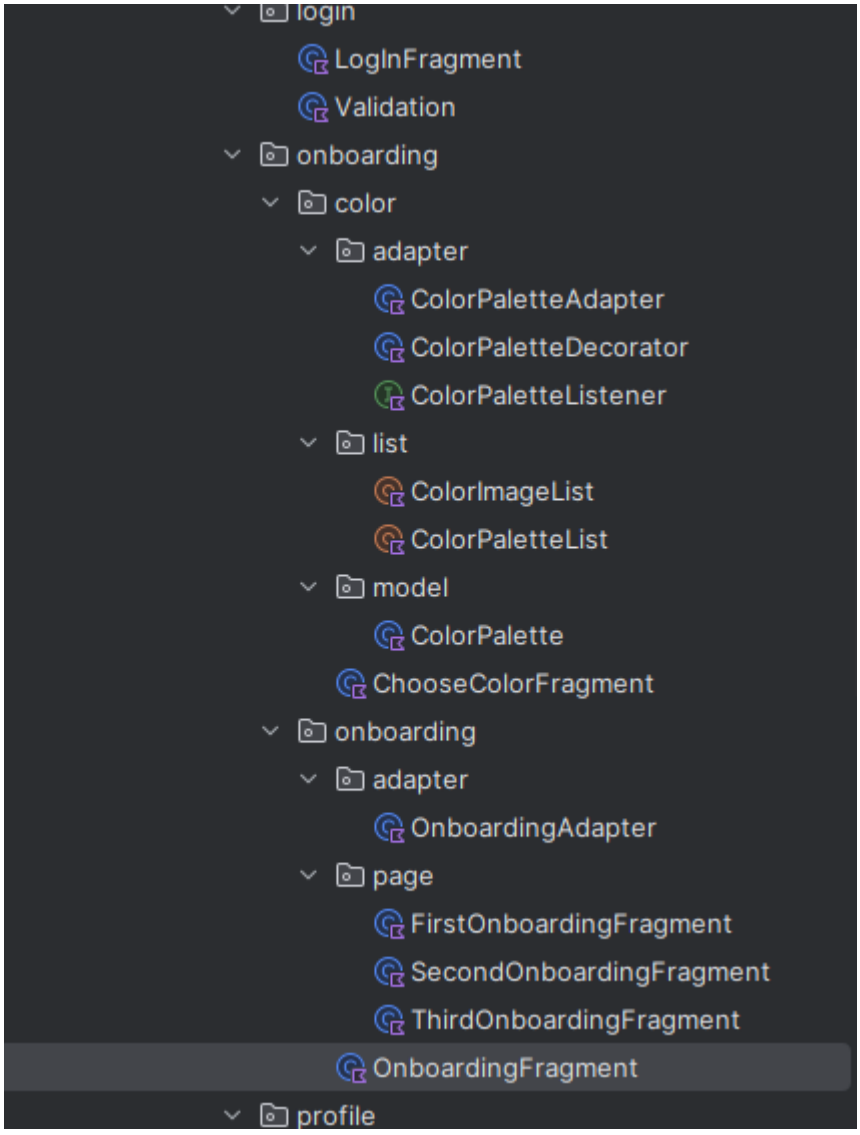


Рис. 3.20 Процеси обрання кольору

1. ProfileSettingsBottomSheetDialog: Це діалогове вікно нижнього аркуша, яке містить різні елементи керування профілем, такі як редагування профілю, налаштування, зміна кольору тощо. Кожен з цих елементів має відповідний обробник подій, який надсилає відповідний інтент або виконує певні дії.

2. RegistrationFragment: Це фрагмент для екрану реєстрації. Він містить дві кнопки, які ведуть до фрагментів входу та реєстрації.
3. RestorePasswordFragment: Це фрагмент, який відповідає за екран скидання пароля. Користувач вводить свою адресу електронної пошти, після чого відправляє запит на скидання пароля. Під час цього процесу також проводиться валідація введених даних, і користувач отримує сповіщення про успішне скидання пароля.
4. SignUpFragment: Цей фрагмент обробляє функціонал реєстрації користувачів. Він включає перевірку форми для полів імені, адреси електронної пошти та пароля. Він також забезпечує розгалужений рядок для клікабельного посилання "Sign In", переходи між різними станами (з клавіатурою і без неї) і навігацію до екрану реєстрації після успішного входу в систему.
5. SplashFragment: Цей фрагмент є частиною функціональності заставки. Він використовує MotionLayout для анімації переходів на заставці. Залежно від стану користувача (новий користувач, зареєстрований користувач або користувач, який увійшов до системи), він переходить на відповідний екран після завершення анімації заставки.
6. UIElement: Цей клас видається порожнім і, можливо, наразі не використовується.
7. Utils: Цей клас також видається порожнім і, можливо, наразі не використовується.

TextChecker: Це реалізація TextWatcher, яка використовується для загального моніторингу змін у тексті. Вона дозволяє вам визначати власні дії при зміні тексту.

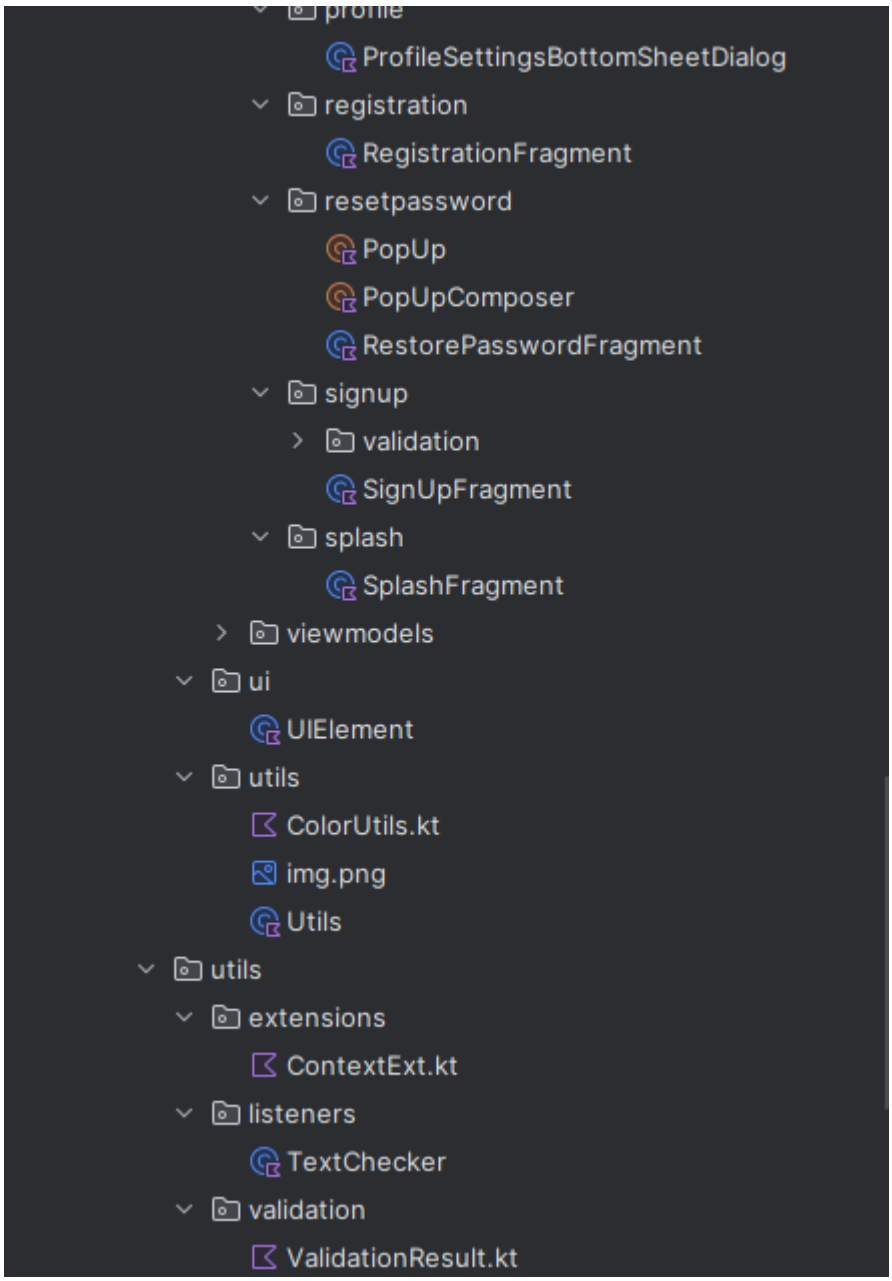


Рис. 3.21 Фрагменти екранних форм

Клас BR:

- Цей клас зазвичай генерується бібліотекою Android Data Binding під час компіляції.
- Він містить посилання на змінні та вирази, що використовуються в XML-файлах макетів.
- У вашому прикладі він визначає константу _all, яка може використовуватися всередині бібліотеки Data Binding.

Клас DataBinderMapperImpl:

- Цей клас також є частиною бібліотеки Android Data Binding.
- Він відповідає за зіставлення ресурсів макета з відповідними класами зв'язування.
- У вашому прикладі він розширює MergedDataBinderMapper і додає маппер для зв'язування даних вашого додатку.

Інтерфейс DataBindingComponent:

- Цей інтерфейс є частиною бібліотеки Android Data Binding.
- Він слугує гачком для вбудовування кастомної поведінки у класи прив'язки даних.
- Ви можете використовувати його для створення власних реалізацій адаптерів, конвертерів та інших компонентів для прив'язки даних.

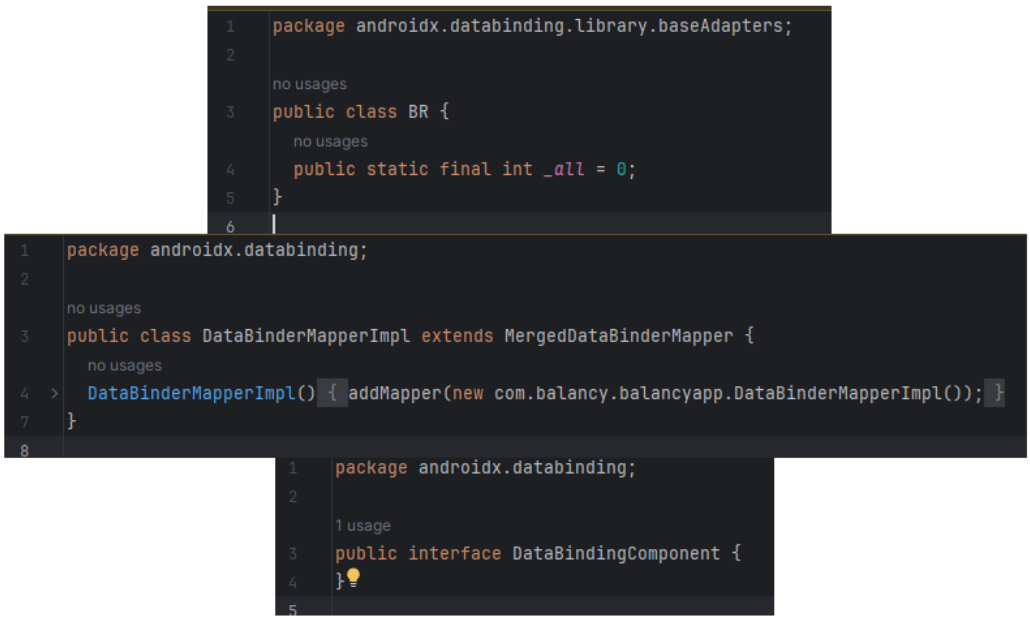


Рис. 3.22 Прив’язка даних до ОС Android

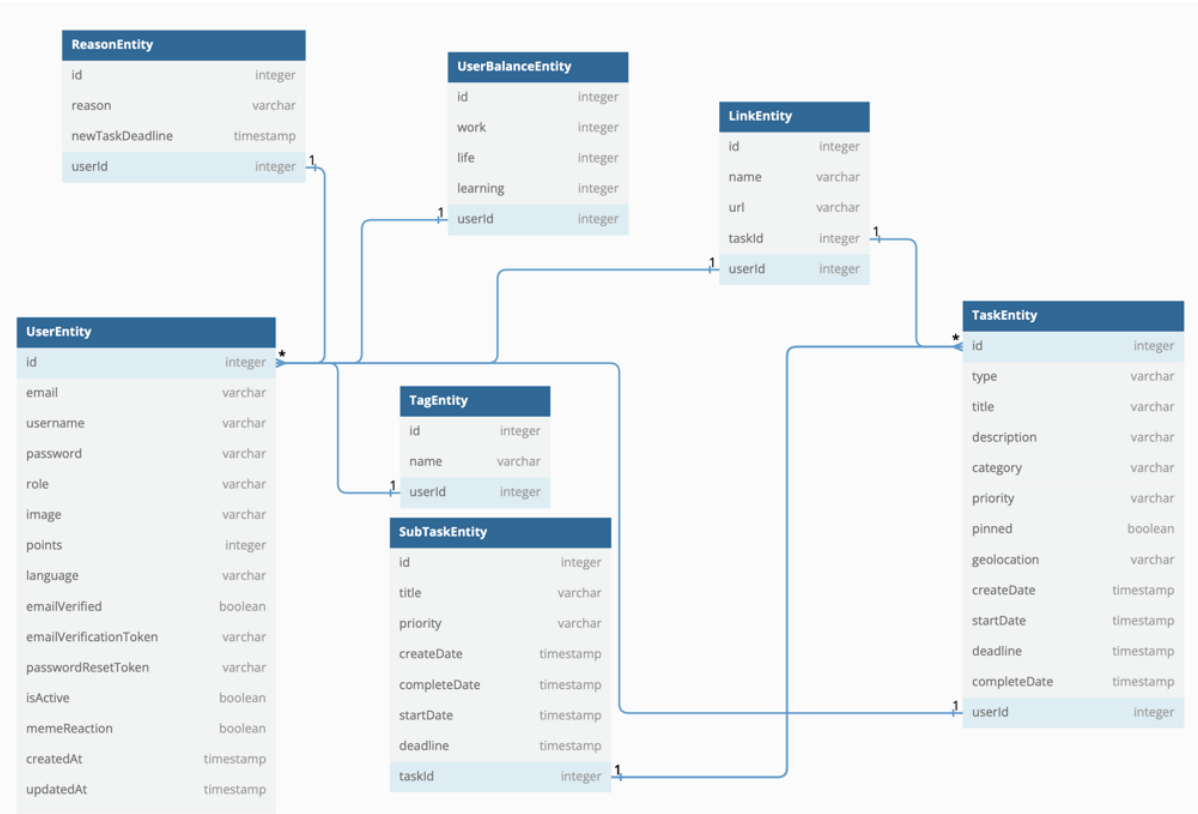


Рис. 3.23 Схематична реалізація бази даних

3.2 Опис функціонування програми у відповідності до діаграми діяльності

Після першого завантаження додатку ми потрапляємо на вікно авторизації в нього, з можливості вибору існуючого облікового запису, створення нового, або під’єднатись за допомогою Google. На рисунку 3.21 показано саме вікно реєстрації:

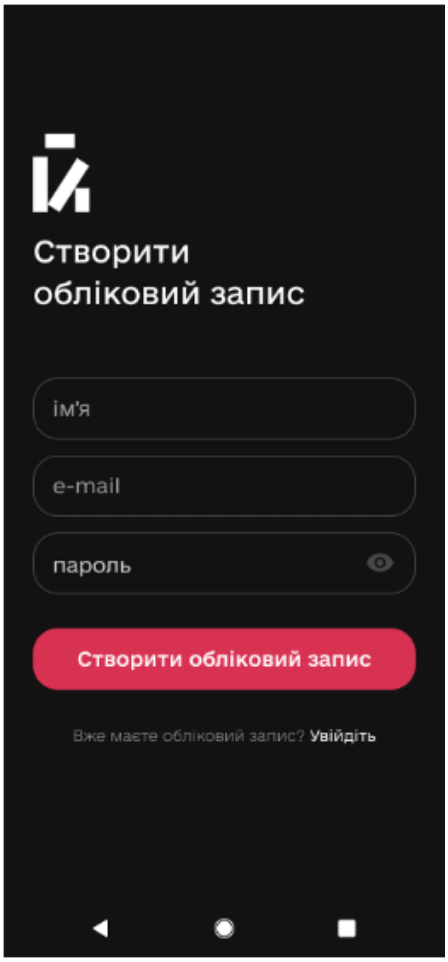


Рис. 3.24 Головна вікно додатку при першому запуску

Якщо користувач вже працював з додатком для цього йому потрібно лише ввести свої облікова дані:

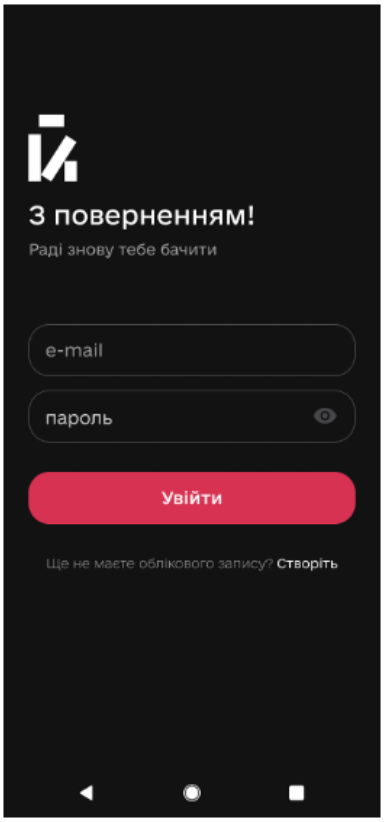


Рис. 3.25 Авторизація в додаток

У випадку якщо користувач не пам’ятає облікових даних створено спеціальне поле відновлення запису.

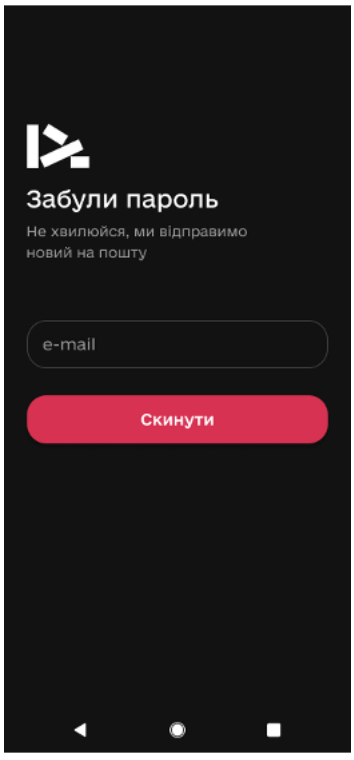


Рис. 3.26 Відновлення облікового запису

Після того, як користувач зайшов в обліковий запис йому доступне поле створення події або задачі, створити назву, вибрати підпункти (охарактеризувати) та додати необхідні теги, якщо цього потребує задача:

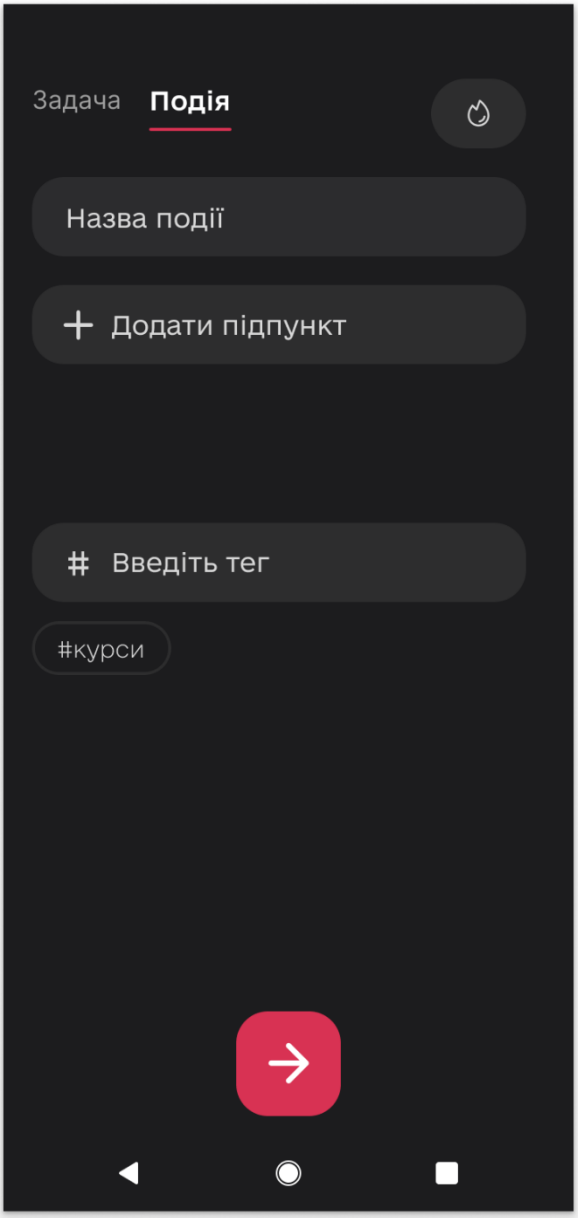


Рис. 3.27 Створення нової події

Далі задачу можна описати, щоб при нагадуванні було зрозуміло куди вона відноситься, також додати геолокацію та час виконання задачі. В разі якщо була налаштована подія, то виключно (що за подія, де відбувається в який час).

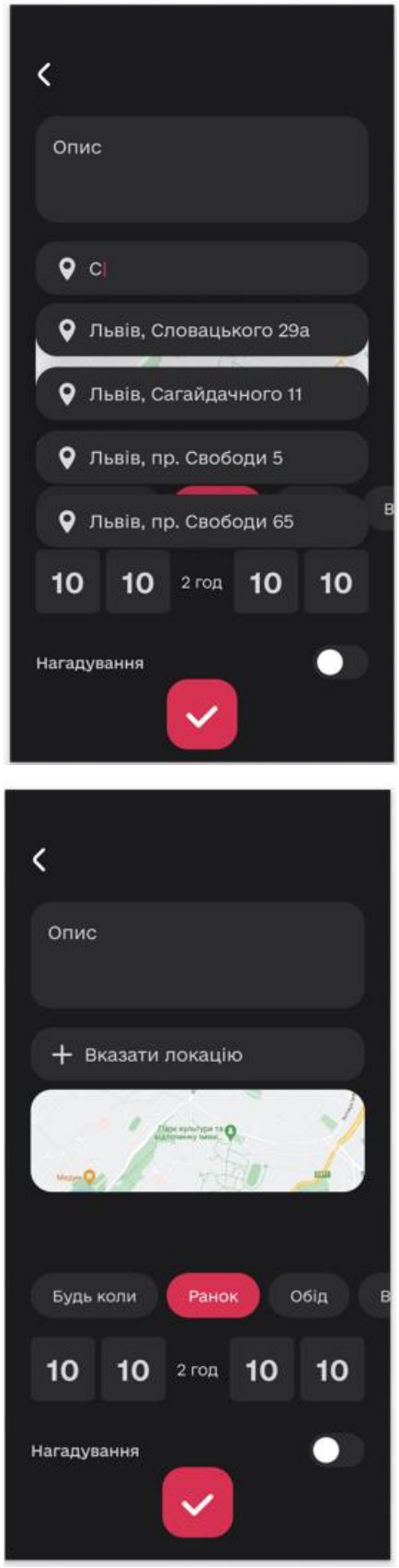


Рис. 3.28 Налаштування місця та часу події

Після затвердження задача або подія буде налаштована та створена.

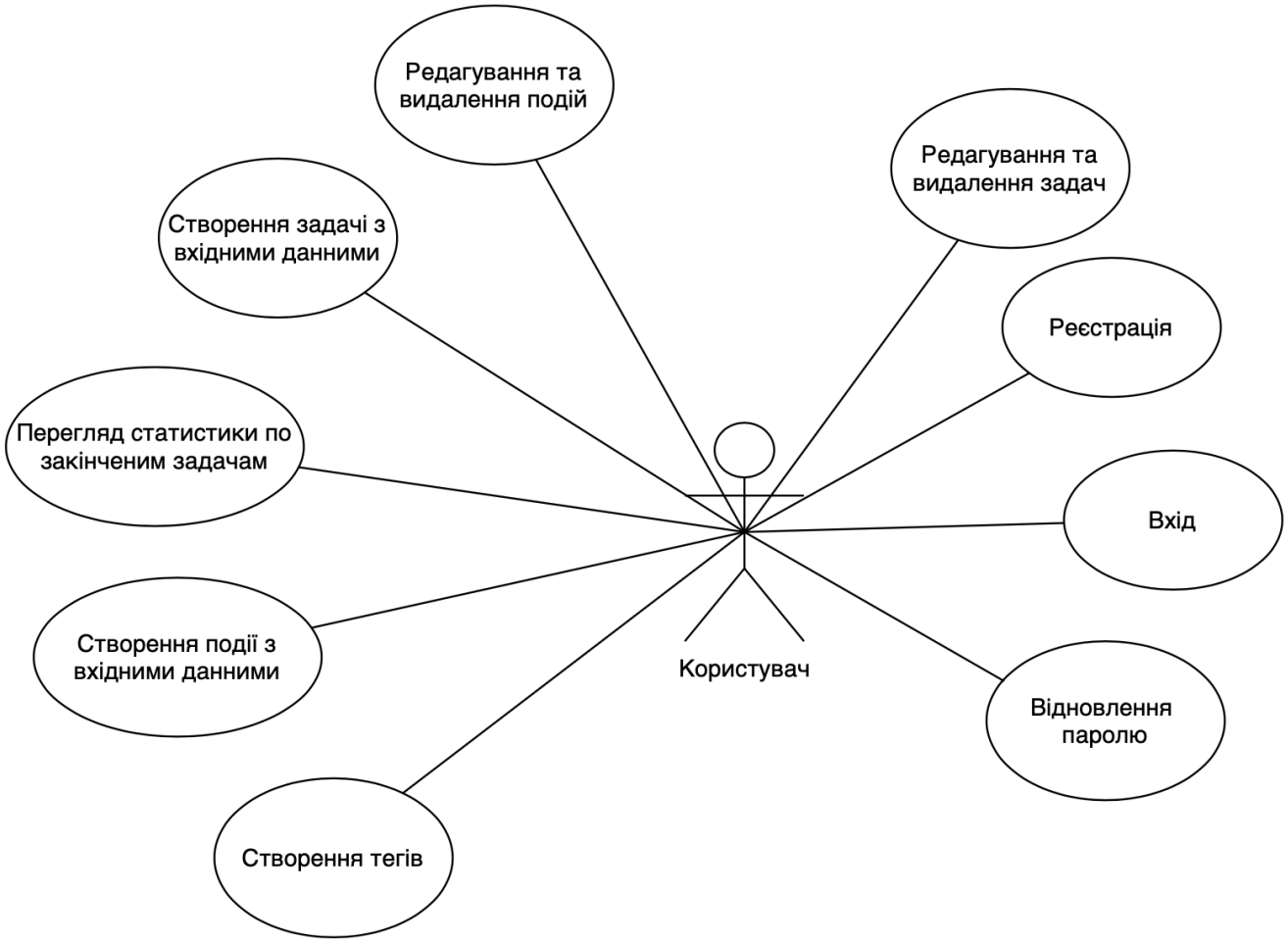


Рис. 3.29 Діаграма діяльності

Висновок до розділу 3

Проаналізовано та реалізовано основні компоненти мобільного додатку для планування задач у салоні краси. Вибрано відповідні технології, які забезпечують надійність та продуктивність додатку. Реалізовано архітектуру, яка дозволяє ефективно управляти задачами та ресурсами салону. Зазначено, що правильний вибір технологій і добре продумана архітектура є критичними для успішної реалізації та функціонування додатку.

Розроблено та описано функціонування мобільного додатку згідно з діаграмою діяльності. Це включає детальний опис процесів, які відбуваються у програмі, що дозволяє користувачам легко взаємодіяти з додатком та ефективно використовувати його функції. Зазначено, що діаграми діяльності є важливим інструментом для візуалізації процесів та полегшення розуміння роботи додатку, що сприяє його ефективному використанню та подальшому вдосконаленню.

Проаналізовано та реалізовано мобільний додаток для планування задач у салоні краси, включаючи вибір технологій, розробку архітектури та детальний опис функціонування програми. Визначено, що добре продумана архітектура, правильний вибір технологій і чітке розуміння процесів роботи програми є ключовими факторами успішної реалізації додатку. Це забезпечує ефективне управління задачами, підвищення продуктивності салону та задоволення клієнтів.

ВИСНОВКИ

Було проведено огляд літератури з методів управління часом, тайм-менеджменту та психології продуктивності, що дозволило отримати глибоке розуміння теоретичних основ цих понять.

Визначені основні проблеми та потреби в управлінні часом у сучасному світі, що послужило важливою основою для розробки додатка.

Проведений аналіз існуючих програмних засобів дозволив визначити їх переваги та недоліки, а також виявити можливості для покращення.

На основі цих висновків була розроблена концепція додатка для планування часу з визначенням ключових функціональних можливостей та особливостей інтерфейсу користувача.

Далі був розроблений прототип додатка який був протестований з метою виявлення можливих недоліків. На основі результатів тестування були внесені необхідні корективи до прототипу, що дозволило покращити функціональність та зручність використання додатка.

Завершальним етапом було розроблення остаточної версії додатка для планування часу, яка враховує всі виявлені вимоги та відповідає потребам користувачів.

Робота пройшла апробацію:

Довбня О.В., Негоденко О.В. Розробка мобільного застосунку для планування задач. IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 р., Київ, Державний університет інформаційних технологій. Збірник тез. К.: ДУІКТ, 2024. Подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Squire, L. R. – Knowlton, B. – Musen, G. The Structure and Organization of Memory [online]. Annual Review of Psychology, 1993. [cit. 2020/04/09]. URL: https://www.annualreviews.org/ doi/pdf/10.1146/annurev.ps.44.020193.002321 (дата звернення: 24.03.2024).

2. Clifton, C. et al. Language Comprehension and Production [online]. Comprehensive handbook of psychology, 2013. [cit. 2020/04/09]. URL: https://www.researchgate.net/publication/269700690_Language_Comprehension_and_ Production (дата звернення: 25.03.2024).

3. Greg A. Jamieson and Kim J. Vicente. Ecological interface design for petrochemical applications: supporting operator adaptation, continuous learning, and distributed, collaborative work. Computers & Chemical Engineering, 25(7-8):1055–1074, August 2001.

4. Polat K. Computer aided diagnosis of ECG data on the least square support vector machine / K. Polat, B. Akdemir, S. Gbne // Digit Signal Process, 2008. – Vol. 18. – № 1. – P. 25-32.

5. L. Bass, P. Clements, R. Kazman. Software Architecture in Practice. 2-nd edition, Addison-Wesley, 2003.

6. Давидов М.В., Демчук А.Б., Лозинська О.В. Програмне забезпечення мобільних пристроїв: навчальний посібник – Львів: Видавництво «Новий Світ-2000» 2020. – 218 с

7. Власій О.О., Винничук М.Д. Розробка мобільних додатків засобами блочного програмування: Навчально-методичний посібник. – Івано-Франківськ: Прикарпатський національний університет імені Василя Стефаника, 2021. − 130 с.

8. Павлиш В. А. Основи інформаційних технологій і систем: Навчальний посібник. / Павлиш В. А., Гліненко Л. К. - Львів: Видавництво Львівської політехніки, 2021. − 500 с.

9. Дворецький М. Л., Нездолій Ю. О., Дворецька С. В., Кандиба І. О. Розробка мобільних застосунків для OS Android : навч. посіб. – Миколаїв : Вид-во

ЧНУ ім. Петра Могили, 2021. – 140 с.

10. Rafael V. Exploring Intelligent Decision Support Systems. Current State and New Trends / V. Rafael. – Munich : Springer International Publishing AG, 2018. – 237 p.

11. Hrabovskyi Y. Methods of Developing the Event-agency Site / Y. Hrabovskyi // Збірник наукових праць Харківського національного університету Повітряних Сил. – 2021. – Вип. 4 (70). – С. 70-76.

12. Hrabovskyi Y., Brusiltseva Yu. The methodology of developing a mobile application design for creating a genealogical tree // Поліграфія і видавнича справа. 2022. № 1 (83). С. 66-78

13. Hrabovskyi Y., Brynza N., Vilkhivska O. Development of information visualization methods for use in multimedia applications. EUREKA: Physics and Engineering. 2020. № 1. Pp. 3–17.

14. Hrabovskyi Y., Fedorchenko V. Development of the optimization model of the interface of multimedia edition. EUREKA: Physics and Engineering. 2019. № 3. Pp. 3–12.

15. Safonov I. Adaptive Image Processing Algorithms for Printing. Springer. 2018. 304 p.

16. Billinghurst M. Emerging Technologies of Augmented Reality: Interfaces and Design. / Billinghurst Mark, Thomas Bruce, 2007. - p. 209 - ISBN 978-1-5990-4066-0.

17. Patrick D. Archeoguide: System Architecture of a Mobile Outdoor Augmented Reality System. / Patrick D. Karigiannis, John N., 2009. - p. 345 - ISBN 9780769517810.

18. Hemrajani A. Getting Started with iBeacon/ Anil Hemrajani, Scott W. Ambler and Rod Johnson – USA: Sams Publishing, 2016. – 318 p

19. Castillo-Garcia P. Indoor Navigation Strategies for Aerial Autonomous Systems / Pedro Castillo-Garcia, Laura Munoz, Hernandez Pedro Gil – ButterworthHeinemann, 2016. – 300 p. – ISBN 9780128051894

20. Сучасні підходи до розроблення і впровадження інформаційних систем URL: https://pidruchniki.com/1181092047726/informatika/cuchasni_pidhodi_rozroblenn ya_vprovadzhennya_informatsiynih_sistem. (Дата звернення 25.04.2024).

21. Мови програмування для мобільної розробки URL: https://code.tutsplus.com/uk/articles/mobiledevelopment-languages--cms-29138. (Дата звернення 25.04.2024).

22. Методичні рекомендації з комерціалізації розробок, створених в результаті науково-технічної діяльності – К.: Наказ Державного комітету України з питань науки, інновацій та інформатики.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільного застосунку для планування задач в салоні краси на платформі Node.js з використанням Kotlin

Виконав студент 4 курсу
групи ПД-44
Прізвище Довбня Олександр Володимирович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Негоденко Олена Василівна

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи –** покращити процес планування задач в салоні краси за рахунок мобільного застосунку, створеного на платформі Node.js з використанням Kotlin.
- **Об'єкт дослідження** – процес планування задач в салоні краси.
- **Предмет дослідження** – мобільний застосунок для планування задач в салоні краси.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. *Проаналізувати існуючі аналоги додатків для планування часу, визначити переваги та недоліки.*
2. *Розробити функціональні та нефункціональні вимоги до мобільного застосунку.*
3. *Дослідити та вибрати засоби розробки мобільного застосунку, розробити модель. архітектури додатку та його дизайн.*
4. *Розробити мобільний застосунок на основі обраних інструментів та визначених вимог.*
5. *Провести тестування мобільного застосунку.*

3

Аналіз аналогів

Характеристики	Jira	ToDoist	Time Management
Нагадування про заплановану задачу	-	-	+
Додавання додаткових локацій проведення події	-	-	+
Розподілення задач	Завдяки поміткам чи під-задачами	Є можливість створити проект	Завдяки власним тегам
Інтеграція з іншими додатками	Присутня, але не з багатьма	Присутня	Відсутня

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Створення завдань з визначенням назви, терміну виконання та пріоритету.
2. Організація завдань у списки категорії.
3. Перегляд та редагування завдань за поточний день, тиждень.
4. Відзначення виконаних завдань та їх архівування.
5. Перегляд статистики виконаних завдань.

Нефункціональні вимоги

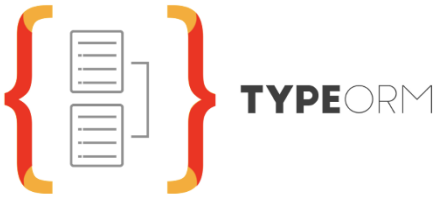
1. Стабільність роботи програми, забезпечення відсутності помилок та відмов у роботі.
2. Обробка клієнтських помилок
3. Хешування приватних даних користувача
4. Локалізація українською мовою

5

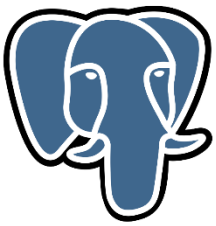
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

TypeScript

Nest.js



Kotlin

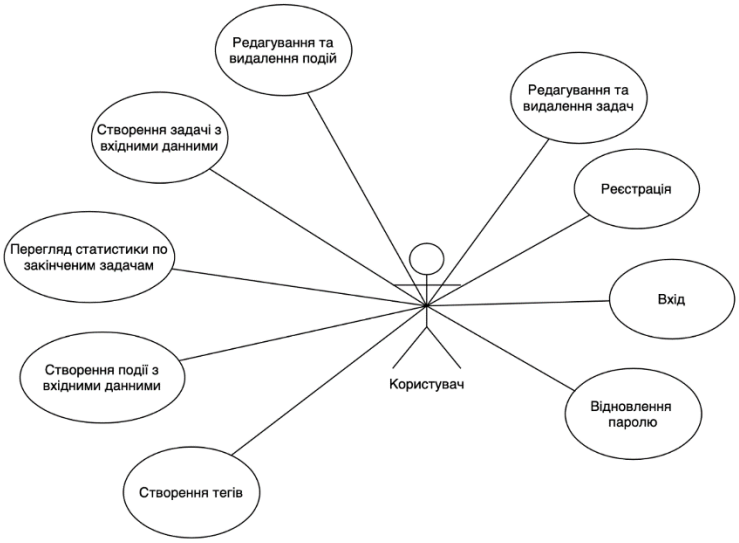


Postgres

Android Studio

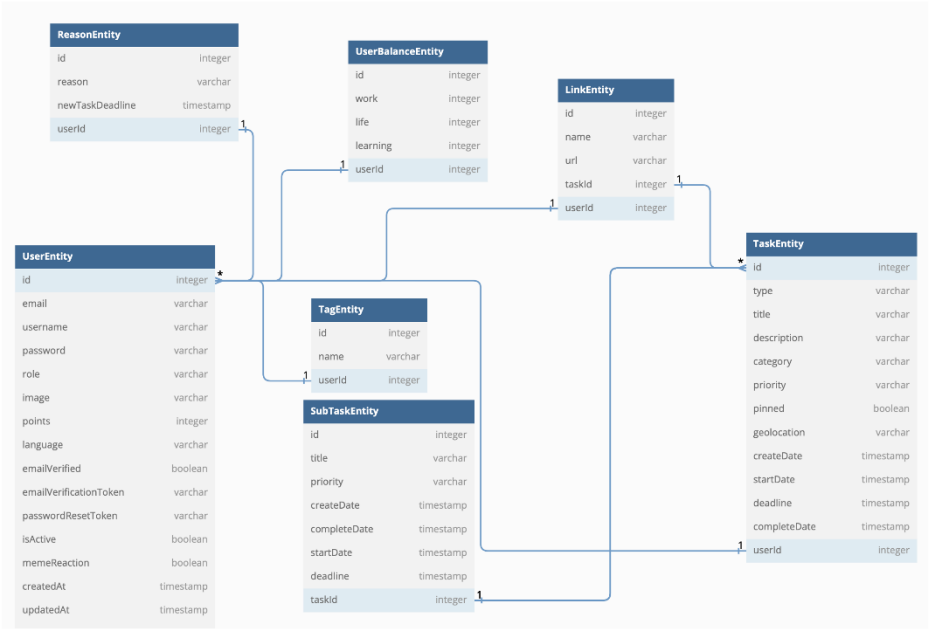
6

Use Case Diagram



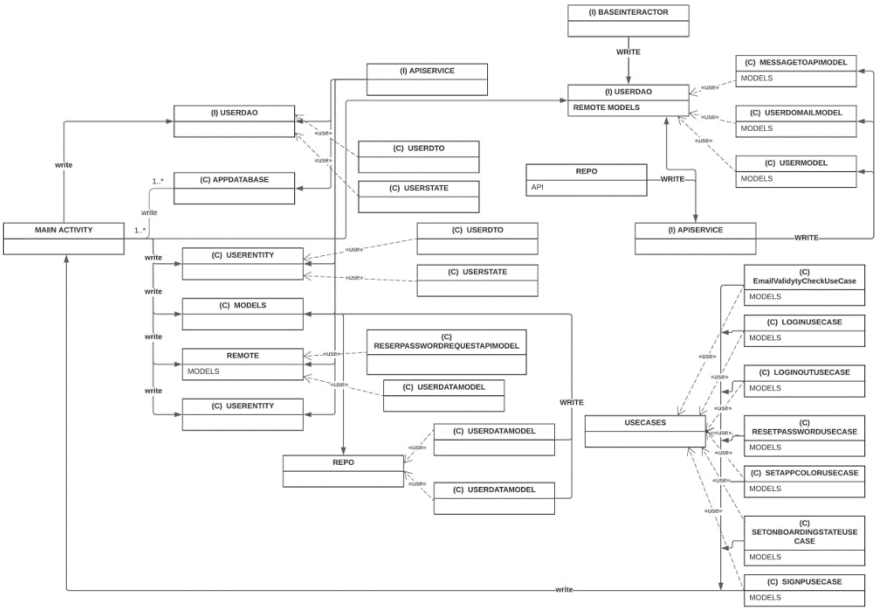
7

Схема бази даних



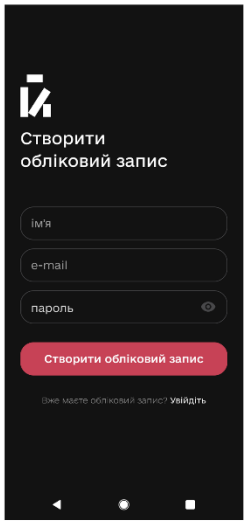
8

Діаграма класів

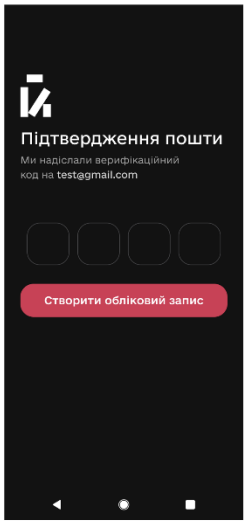


9

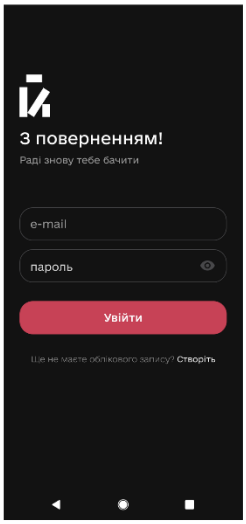
Екранні форми



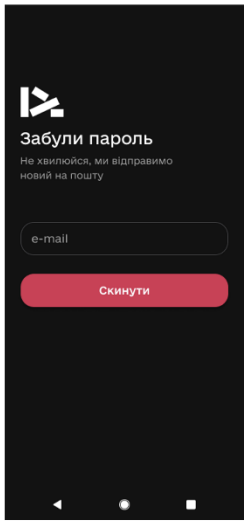
Реєстрація



Підтвердження пошти



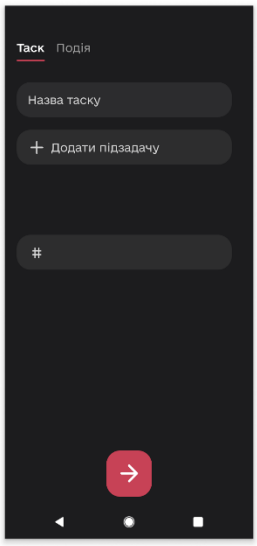
Вхід



Відновлення доступу

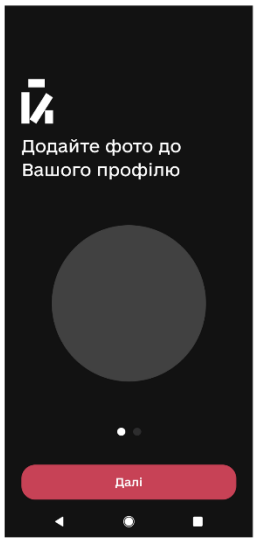
10

Екранні форми



Створення задачі

Перегляд задач



Додавання фото

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Довбня О.В., Негоденко О.В. Розробка мобільного застосунку для планування задач. IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 р., Київ, Державний університет інформаційних технологій. Збірник тез. К.: ДУІКТ, 2024. Подано до друку.

ВИСНОВКИ

1. Проаналізовано додатки для планування та управління часом: Jira, ToDoist та виділено основні переваги та недоліки.
2. Створено технічне завдання, обрано стек технологіій та розроблено функціональні та нефункціональні вимоги до мобільного застосунку.
3. Обрано технології: Nestj.js в якості фреймворка, TypeScript та Kotlin як мово програмування, системи управління базами даних PostgreSql, ORM - TypeOrm, WebStorm для розробки бекенд частики, AndroidStudio – клієнтської.
4. Спроектовано архітектуру мобільного застосунку на основі клієнт-серверної моделі, яка розділяє його на три частини: серверну, клієнтську та базу даних.
5. Архітектура мобільного застосунку враховує всі функціональні та нефункціональні вимоги.
6. Розроблено мобільний застосунок для планування часу в салоні краси, використовуючи патерн Controller-Service-Repository для серверної частини, модульний підхід для клієнтської частини та REST API для забезпечення взаємодії між серверною та клієнтською частинами.
7. Проведено тестування, яке підтвердило відповідність функціональним і нефункціональним вимогам.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```kotlin
package com.balancy.data.local.dao

import ...

@Dao
interface UserDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUser(userEntity: UserEntity)

    @Query("DELETE FROM user")
    suspend fun deleteUser()

}
```

```kotlin
package com.balancy.data.local.database

import ...

@Database(
    entities = [UserEntity::class],
    version = 1,
    exportSchema = false
)
abstract class AppDatabase: RoomDatabase() {
    abstract fun userDao(): UserDao

    companion object {
        const val DATABASE_NAME = "balancy_db"
    }
}
```

```kotlin
package com.balancy.data.local.entities

import ...

@Entity(tableName = "user")
data class UserEntity(
    @PrimaryKey(autoGenerate = false)
    val id: Int = 1,
    val name: String,
    val email: String,
    val password: String
)
```

```kotlin
package com.balancy.data.mappers

import ...

fun MessageToApiModel.mapToResetPasswordRequestApiModel(): ResetPasswordRequestApiModel =
    ResetPasswordRequestApiModel(email)

fun UserModel.mapToCreateUserApi(): UserDTO {
    return UserDTO(
        name = name,
        email = email,
        password = password
    )
}
fun UserModel.mapToUserEntity(): UserEntity {
    return UserEntity(
        name = name,
        email = email,
        password = password
    )
}

fun UserDomainModel.mapToUserDataModule(): UserDataModel = UserDataModel(email, password)


fun UserDomainModel.mapToUserEntity(): UserEntity =
    UserEntity(name = " ", email = email, password = password)

```

```kotlin
package com.balancy.data.models

data class UserDTO(
    val name: String,
    val email: String,
    val password: String
)
```

```kotlin
package com.balancy.data.models

enum class UserState {
    NEW_USER,
    ONBOARDED_USER,
    LOGGED_IN_USER
}
```

```kotlin
package com.balancy.data.remote.models

data class ResetPasswordRequestApiModel(val email: String)
```

```kotlin
package com.balancy.data.remote.models

data class UserDataModel(val email: String, val password: String)
```

```kotlin
package com.balancy.data.repository

import ...

class SettingsRepositoryImpl @Inject constructor(
    private val sharedPreferences: SharedPreferences
) : SettingsRepository {

    override suspend fun setAppThemeColor(color: Int) {
        sharedPreferences.edit().putInt(KEY_INT_VALUE, color).apply()
    }

    override fun getAppThemeColor(): Int = sharedPreferences.getInt(KEY_INT_VALUE, defValue: 1)
    override suspend fun setUserState(state: UserState) {
        sharedPreferences.edit().putInt(KEY_BOOLEAN_ONBOARDED_STATE_VALUE, state.ordinal).apply()
    }

    override fun getUserState(): UserState = UserState.entries[sharedPreferences.getInt(
        KEY_BOOLEAN_ONBOARDED_STATE_VALUE, UserState.NEW_USER.ordinal
    )]



    companion object {
        const val KEY_INT_VALUE = "theme"
        const val KEY_BOOLEAN_ONBOARDED_STATE_VALUE = "onboarding"
    }
}
```

```kotlin
package com.balancy.data.repository

import ...

class UserRepositoryImpl @Inject constructor(
    private val userDao: UserDao,
    private val apiService: ApiService
) : UserRepository {
    override suspend fun resetPassword(messageToApiModel: MessageToApiModel): String {
        val map = messageToApiModel.mapToResetPasswordRequestApiModel()
        return try {
            apiService.requestResetPassword(map)
        } catch (e: retrofit2.HttpException) {
            e.code().toString()
        } catch (e: java.net.UnknownHostException) {
            e.toString()
        }
    }

    override suspend fun signUp(userModel: UserModel) {
        try {
            apiService.signUp(userModel.mapToCreateUserApi())
            userDao.insertUser(userModel.mapToUserEntity())
        } catch (e: retrofit2.HttpException) {
            throw e
        }
    }

    override fun logIn(userDomainModel: UserDomainModel): Flow<String?> = flow { this: FlowCollector<S
        val response= apiService.requestLogIn(userDomainModel.mapToUserDataModule())
        userDao.insertUser(userDomainModel.mapToUserEntity())
        emit(response.body())
    }.flowOn(Dispatchers.IO)

    override suspend fun logOut() = userDao.deleteUser()
}
```

```kotlin
package com.balancy.domain.repository

import com.balancy.data.models.UserState


interface SettingsRepository {
    suspend fun setAppThemeColor(color: Int)

    fun getAppThemeColor(): Int

    suspend fun setUserState(state: UserState)
    fun getUserState(): UserState
}
```

```kotlin
package com.balancy.domain.repository

import ...

interface UserRepository {
    suspend fun resetPassword(massageToApiModel: MessageToApiM

    suspend fun signUp(userModel: UserModel)

    fun logIn(userDomainModel: UserDomainModel): Flow<String?>

    suspend fun logOut()
}
```

```kotlin
package com.balancy.domain.usecases

import java.util.regex.Pattern

class EmailValidityCheckUseCase {
    fun execute(email: String) =
        Pattern.compile(
            regex "[a-zA-Z0-9\\+\\.\\_\\%\\-\\+]{1,256}" +
                    "\\@" +
                    "[a-zA-Z0-9][a-zA-Z0-9\\-]{0,64}" +
                    "(" +
                    "\\." +
                    "[a-zA-Z0-9][a-zA-Z0-9\\-]{0,25}" +
                    ")+"
        ).matcher(email).matches()
}
```

```kotlin
package com.balancy.domain.usecases

import ...

class ResetPasswordUseCase(private val userRepository: UserRepository) {
    suspend fun execute(messageToApiModel: MessageToApiModel): Boolean {
        return userRepository.resetPassword(messageToApiModel) == "201"
    }
}
```

```kotlin
package com.balancy.domain.usecases

import ...

class SignUpUseCase(private val userRepository: UserRepository) {

    suspend fun invoke(name: String, email: String, password: String) {
        userRepository.signUp(
            UserModel(
                name = name,
                email = email,
                password = password
            )
        )
    }
}
```

```kotlin
package com.balancy.domain.usecases

import ...

class LogInUseCase(private val userRepository: UserRepository) {

    fun execute(userDomainModel: UserDomainModel): Flow<String?> {
        return userRepository.logIn(userDomainModel)
    }

}
```

```kotlin
package com.balancy.domain.usecases

import com.balancy.domain.repository.UserRepository

class LogOutUseCase(private val userRepository: UserRepository)

    suspend fun invoke() = userRepository.logOut()
}
```

```kotlin
package com.balancy.domain.usecases

import com.balancy.domain.repository.SettingsRepository

class SetAppColorUseCase (private val settingsRepository: SettingsRepository) {
    suspend fun invoke(color: Int) {
        settingsRepository.setAppThemeColor(color)
    }
}
```

```kotlin
package com.balancy.domain.usecases

import ...

class SetOnboardingStateUseCase(private val settingsRepository: SettingsRepository) {
    suspend fun invoke(userState: UserState) {
        settingsRepository.setUserState(userState)
    }
}
```

```kotlin
package com.balancy.navigation

import ...

interface INavigator {

    /**
     * Set navigation controller
     */
    fun setNavController(navController: NavController)

    /**
     * Get navigation controller
     */
    fun getNavController(): NavController?

    /**
     * Navigate using destination id
     */
    fun open(screen: Int)

    /**
     * Open destination using navigation action
     */
    fun open(screen: NavDirections)

    /**
     * Back navigation
     */
    fun goBack()

    /**
     * Clear the navigation graph and Navigator arguments
     */
    fun clear()
```

```kotlin
package com.balancy.navigation

import ...

@Singleton
class Navigator @Inject constructor() : INavigator {

    private var navController: NavController? = null
    private var navOptions: NavOptions? = null

    override fun setNavController(navController: NavController)
        this.navController = navController
    }

    override fun getNavController(): NavController? = navContro

    override fun open(screen: Int) {
        try {
            navController?.navigate(screen, args: null, navOptio
        } catch (ex: IllegalArgumentException) {
            ex.printStackTrace()
        }
    }

    override fun open(screen: NavDirections) {
        try {
            navController?.navigate(screen)
        } catch (ex: IllegalArgumentException) {
            ex.printStackTrace()
        }
    }

    fun open(screen: Int, bundle: Bundle?) {
        try {
            navController?.navigate(screen, bundle, navOptions)
        } catch (ex: IllegalArgumentException) {
```

```java
package androidx.databinding.library.baseAdapters;

no usages
public class BR {
  no usages
  public static final int _all = 0;
}
```

```java
package androidx.databinding;

no usages
public class DataBinderMapperImpl extends MergedDataBinderMapper {
  no usages
  DataBinderMapperImpl() { addMapper(new com.balancy.balancyapp.DataBinderMapperImpl()); }
}
```

```java
package androidx.databinding;

1 usage
public interface DataBindingComponent {
}
```