СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

на тему «Розробка мобільного додатку для оптимального управління часом»

Виконав: студент групи ІПЗ-21д	_____ (підпис)	П.Ю. Подоляка (ініціали і прізвище)
Керівник	_____ (підпис)	Д.В. Ратов (ініціали і прізвище)
Завідувач кафедри	_____ (підпис)	О.І. Захожай (ініціали і прізвище)

Рецензент _____В.О. Лифар_____

Київ – 2025

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування
Освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Захожай О.І.
“___” _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Подоляка Павло Юрійович
(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка мобільного додатку для оптимального управління часом
керівник роботи Ратов Денис Валентинович, доцент, к.т.н.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “_27_” _травня_2025 року №67/15.15-С

2. Строк подання студентом роботи 16.06.2025р.
3. Вихідні дані до роботи: Об’єктом даної роботи є процес створення додатку для оптимального управління часом

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Вступ. Опис мети та цілей розробки. Аналітичний огляд. Опис інформаційної моделі. Опис архітектури додатку Розробка програмного забезпечення Опис процесу створення додатку для оптимального управління часом Висновки. Перелік використаних джерел.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 01.04.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Отримання завдання та збір матеріалів	01.04.25	виконано
2	Огляд предметної області	18.04.25	виконано
3	Формулювання вимог до системи та розробка основних алгоритмів	04.05.25	виконано
4	Розробка практичної частини	01.06.25	виконано
5	Оформлення пояснювальної записки	12.06.25	виконано
6	Підготовка та подання роботи до захисту	16.06.25	виконано

Студент _____ П.Ю. Подоляка
підпис (ініціали і прізвище)

Керівник роботи _____ Д.В. Ратов
підпис (ініціали і прізвище)

РЕФЕРАТ

Текст – 91с., рис. – 32, табл. – 1, додатків – 1, літературних джерел – 4

У ході виконання даної дипломної роботи було проведено дослідження процесу розробки мобільного додатку для оптимального управління часом, від ідеї до реалізації. Основні характеристики додатку: підвищення продуктивності, планування, розклад, тайм-менеджмент.

Був проведений аналіз позитивних і негативних властивостей мобільних додатків. Він допоміг виявити помилки, яких вдалося уникнути.

Також були визначені оптимальні підходи до розробки програмного забезпечення. Зокрема обрано чітку архітектуру, яка має гарну масштабованість та можливість тестування.

У результаті виконання роботи було створене програмне рішення у вигляді мобільного додатку для оптимального управління часом. Вийшло дотриматися більшості критеріїв. У результаті ми отримали робочий продукт зі зручним, простим та інтуїтивно зрозумілим інтерфейсом. Цей продукт можна практично використати для створення власного розпорядку дня, створити розклад на тиждень, а також запланувати завдання на майбутнє.

Ключові слова: МОБІЛЬНИЙ ДОДАТОК, РОЗРОБКА, АРХІТЕКТУРА, СТРУКТУРА, ФУНКЦІЯ, МЕТОД, ІНТЕРФЕЙС, БАЗА ДАНИХ, БІБЛІОТЕКА, КРИТЕРІЇ, АНАЛІЗ.

ЗМІСТ

ВСТУП

Актуальність. Якщо звернути увагу на появу проблем, а потім, і їх наслідків, виникнення переживань і стресу, можна помітити, що багато з них є наслідком ліні, слабкої дисципліни або, взагалі, її відсутності.

Сучасний світ вимогливий, стрімкий, непередбачуваний. Щоб встигати за змінами, відповідати вимогам суспільства, рухатися до успіху, треба вміти контролювати себе. Зараз із появою нових, корисних і цікавих речей, з'являється і велика кількість зовнішніх подразників для відволікання уваги. У людей зі слабкою дисципліною часто виникають проблеми із зосередженням на певній справі, навіть важливій. Швидка втрата концентрації і перемикання на інше заняття витрачає дорогоцінний час, відведений на виконання завдання. А коли це входить у звичку, починається протистояння себе з собою. Іноді ми намагаємося змусити себе зробити щось корисне, і починаємо вже робити, а потім, різко перемикаємо увагу на більш цікаве заняття. Але цікаве не дорівнює корисне. І таке часто відбувається несвідомо. На початку ми можемо не помічати, що таке явище є причиною нових проблем. З часом приходить розуміння і думки: «Час вже з цим покінчити!».

Об'єкт досліджень: процес розробки мобільного додатку для системи Android.

Предмет досліджень: додаток для оптимального управління часом.

Мета дослідження: дослідити процес розробки мобільних додатків для операційної системи Android, а також розробити додаток для оптимального управління часом.

Завдання дослідження:

1. Ознайомитись із процесом розробки мобільних додатків;
2. Проаналізувати переваги та недоліки готових рішень;
3. Ознайомитися із середовищем розробки Android Studio
4. Сформувати вимоги до підходу розробки;
5. Розробити дизайн графічного інтерфейсу додатку;
6. Програмна реалізація додатку

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД

1.3. Актуальність

Давайте розглянемо доволі поширене явище під назвою прокрастинація: прокрастинація – це поведінка людини, яка призводить до розтягнення або перенесення термінів виконання завдання. Тобто людина відкладає справу на інший час, хоча є можливість займатися нею зараз, при цьому відсутній критичний фактор, який міг бути чинником відтермінування. Явище може відбуватися, як перед виконанням роботи: коли ми тільки обираємо завдання на поточний час, так і у процесі виконання завдання [1].

Звичайне повідомлення про нову публікацію на сторінці у людини, за якою ви слідкуєте, випадкова реклама чи нудна одноманітна робота може легко змусити нас відволіктися на більш просте і цікаве заняття. Найбільша кількість подразників людської уваги, частіше за все, йде від електронних пристроїв. Чому так відбувається і чи справді саме гаджети так погано на нас впливають?

Смартфони і подібні «розумні» пристрої надають доступ до інформації майже з усього світу. Ми отримуємо коротку і цікаву інформацію у дуже великих об’ємах і при цьому нам навіть не треба виходити з дому. Ми звикаємо часто перемикати нашу увагу, з часом відчуваємо недостатню цікавість до отримуваної інформації, що призводить до довгого процесу пошуку більш цікавих публікацій. Це забирає наш час. Сюди додається відсутність чіткого графіку і нерівномірно розподілене навантаження, що в сумі вливає на подальші результати і нашу емоційну рівновагу. А разом із несподіваними чинниками, які від нас не залежать, поступово починається вигорання.

Дана проблема особливо актуальна для людей без жорсткого графіку, або людей, які відчувають сильний дискомфорт від суворого розпорядку дня, а потім потрапляють у середовище з більш вільним вибором. Ці переходи

можуть повторюватися, їх можна порівняти із гойдалкою. Наприклад, порівнюючи зі школою, навчання в університеті дозволяє гнучкий графік, але, нажаль, пізніше він може стати неконтрольованим. Без належного самоконтролю у нас з’являється звичка відкладати справи на інший час. На цьому етапі ми заново вчимося притримуватися жорстко сформованого розпорядку, схожого на той, що був у період навчання у школі. Але зміни вже вносяться на основі власного досвіду, як гарного, так і отриманого шляхом аналізу допущених помилок. Схожі проблеми можна помітити у фрілансерів. Але це не означає, що людям з іншим родом занять не притаманна лінь і прокрастинація.

1.2. Проблематика сучасних мобільних додатків

Варто розглянути проблеми сучасних мобільних додатків. Це допоможе сформувати певні критерії для створення власного додатку.

1. Інтерфейс користувача:

Перенасичений інтерфейс – користувач бачить перед собою багато елементів керування, що може заважати йому користуватися додатком, особливо у ситуації, коли він використовує лише декілька функцій із усіх запропонованих.

Незрозумілий інтерфейс – випадок схожий на попередній, але тут справа у нелогічності або складності розташування елементів на екрані. Навіть, якщо це зрозуміло для вузькоспеціалізованого фахівця, це не завжди розуміють звичайні користувачі.

Відволікаючі елементи – частини програми, можливо суто дизайнерські рішення, які відволікають увагу користувача від основного функціоналу.

2. Монетизація:

Велика кількість нав’язливої реклами – реклама заважає користуватися застосунком, що у подальшому може змусити користувача видалити додаток.

Сильні обмеження – базовий функціонал урізано, багато функцій, які мали бути базовими пропонуються до покупки. Користувач не бачить сенсу продовжувати використовувати додаток, цілком ймовірно почне використовувати краще рішення від конкурента.

3. Адаптація:

Підтримка однієї або декількох мов – додаток, як правило, публікується доступним для усього світу, тому підтримка більшої кількості мов, може значно збільшити кількість активних користувачів. В першу чергу варто додавати найбільш поширені мови, що додає додатку універсальності.

Адаптація під різні типи екранів – варто зробити адаптивний інтерфейс, який буде масштабуватися під різні розміри дисплею.

4. Інтернет з’єднання:

Неможливість додатку виконувати базові функції, без підключення до інтернету (не стосується окремих додатків, які розраховані виключно на роботу з використанням мережі). Такі обмеження часто є рішенням розробників, щоб не втрачати потенційний прибуток на рекламі у додатку.

5. Оптимізація:

Витік пам’яті – недоліки у коді програми, які перевантажують оперативну пам’ять пристрою, що призводить до зниження продуктивності чи аварійного завершення додатку.

Використання зайвих ресурсів – програма неефективно витрачає фізичні ресурси гаджета.

Зайва фонова активність – додаток може не зупиняти дії у фоні, хоча вони можуть бути потрібні тільки за певних обставин.

6. Приватність і безпека:

Збір даних – недобросовісні розробники можуть таємно збирати дані без згоди користувача.

Дозволи – у додатку може з’явитися вимога надати дозвіл на дії, які напряму не потрібні для правильної роботи застосунку, наприклад: доступ до камери, мікрофону, геолокації.

7. Підтримка:

Оновлення – відсутні своєчасні виправлення багів, немає розширеної підтримки різних пристроїв.

Взаємодія із аудиторією – розробники не звертають уваги на зауваження і пропозиції з боку користувачів.

Служба підтримки – відсутність зворотного зв’язку із розробниками.

1.3. Огляд і порівняння готових рішень

Дослідження «ринку конкурентів», є чудовим аналітичним підходом. Таким чином можна дізнатися способи монетизації, залучення аудиторії, а також порівняти додатки між собою та визначити позитивні сторони, які можна потім втілити у власному проекті.

Давайте візьмемо 5 додатків з Google Play для порівняння. Наприклад: два додатки від провідних IT-компаній – Google та Microsoft, а також три додатки, що мають одні з найбільших показників кількості завантажень, кількості відгуків та оцінок. Отже виходить такий список: Google Tasks, Microsoft To Do, TickTick, Any.do та To-Do List – Schedule Planner. Порівняльна таблиця 1.1 дозволить нам наочно відобразити певні критерії якості продуктів, а також можливості зацікавити та утримати користувачів. Дана таблиця

створена на основі мого особистого досвіду користування вище згаданими додатками [2].

Таблиця 1.1

Порівняльна таблиця готових рішень

Характеристика	Google Tasks	Microsoft To Do	TickTick	Any.do	To-Do List …
Зручність	9/10	8/10	9/10	6/10	7/10
Чи є потреба у вході до аккаунта?	Так, не можна пропустити	Так, не можна пропустити	Так, можна пропустити	Так, не можна пропустити	Так, можна пропустити
Чи покриває базові потреби?	Так, 7/10	Так, 8/10	Так, 9/10	Так, 9/10	Так, 9/10
Нагадування	Присутні	Присутні	Присутні	Присутні	Присутні
Календар	Інтеграція/Синхронізація	Інтеграція/ Синхронізація	Вбудований	Інтеграція/Синхронізація	Вбудований
Під завдання	Так	Так	Так	Так	Так
Нотатки	Так	Так	Так	Так	Так
Списки/ Категорії	Так	Так	Так	Так	Так
Повторення завдань	Так	Так	Так	Ні	Так
Мітки	Пріоритетні завдання	Теги	Пріоритетні завдання	Пріоритетні завдання	Теги

Продовження таблиці 1.1

Реклама	Відсутня	Відсутня	Частково присутня	Присутня	Присутня
Обмеження	Відсутні	Відсутні	Розблокування по підписці	Розблокування по підписці	Розблокування по підписці
Потреба в підключені до інтернету	Всі «offline» функції працюють як треба	Всі «offline» функції працюють як треба	Всі «offline» функції працюють як треба	Всі «offline» функції працюють як треба	Всі «offline» функції працюють як треба
Ціна	Безкоштовно	Безкоштовно	Freemium	Freemium	Freemium
Особливості	Простота і зручність. Інтеграції з Gmail, Google Calendar	Інтеграція з Outlook	Наявність вбудованих методів підвищення продуктивності, таких як Pomodoro тощо	Інтеграція сторонніх сервісів	Інтеграція сторонніх сервісів

1.4. Обґрунтування використання мови програмування Kotlin

Kotlin молода і сучасна мова програмування. Вона має відкритий вихідний код, що дає змогу використовувати її людям з усього світу, незалежно від статусу. Мова є строго типізованою, що підвищує надійність коду.

Розробка мови ведеться компанією JetBrains, відомою своїми інструментами для розробки програмного забезпечення з використанням різних мов програмування. Обов’язково варто зазначити, що розробка ведеться

за підтримки відомих і успішних компаній, таких як Google, Meta, Gradle та інших. Ці компанії є членами Kotlin Foundation і широко сприяють розробці і покращенням мови програмування [3].

Kotlin в повній мірі сумісна з Java, що дозволяє використовувати свої переваги у проектах, що первинно створювалися без її використання.

Особливу увагу варто приділити оптимізації використання коду, тобто щоб втілити певну ідею потрібно писати значно менше коду. Це підвищує продуктивність при роботі, а також полегшує читабельність.

У розробці мобільних додатків для платформи Android важливу роль займають асинхронні процеси. Співпрограми Kotlin допомагають розробникам у цьому. З ними створювати асинхронні процеси стало ще легше і зручніше.

Також інструменти Jetpack Compose позбавляють розробників від рутини при створенні інтерфейсу користувача для додатку. Значно скоротилося використання шаблонного коду, що безумовно прискорює розробку.

Окремо хочу виділити чудову якість офіційної документації до мови програмування Kotlin, а також документації зі створення додатків у середовищі розробки Android Studio. Все викладено зрозуміло для новачків, з гарними прикладами, дуже зручно вчитися і практикуватися опираючись на надані розробниками ресурси.

На останок варто сказати, що Google прямо рекомендує до використання мову Kotlin у проектах Android Studio.

РОЗДІЛ 2. ОПИС ІНФОРМАЦІЙНОЇ МОДЕЛІ ДОДАТКУ

2.1. Критерії до процесу розробки додатку

Скористаємося пунктами із проблематики і спробуємо поставити задачу для реалізації.

1. Інтерфейс користувача:

Потрібно зробити простий, інтуїтивно зрозумілий інтерфейс, у якому елементи управління будуть структуровано розташовані, а кількість функцій на одному екрані буде обмеженою. Обмеження дозволить нам притримуватися мінімалізму, відкинувши всі непотрібні або ті, що рідко використовуються функції на другий план. В основі мають бути добре налаштовані базові можливості.

2. Монетизація:

Варто притримуватися моделі «freemium» (якщо додаток комерційний). Ця модель добре показує свою ефективність. Це може бути безкоштовний додаток з рекламою, а покупка преміум версії прибиратиме її. При цьому реклама має бути не нав’язливою. Або преміум версія може надавати доступ до другорядних функцій, які покращують досвід використання додатку. Непоганим варіантом є можливість кастомізації зовнішнього вигляду програми. Не рекомендовано додавати сильні обмеження на функціонал і надавати доступ за підпискою.

3. Адаптація:

Багато популярних додатків підтримують більше 20 різних мов. І звісно треба налаштувати коректне масштабування інтерфейсу під різні розміри дисплеїв. Опціонально додати можливість змінювати розмір інтерфейсу.

4. Інтернет з’єднання:

Краще робити підключення до інтернет мережі необов’язковим. Вимога має з’являтися тільки у разі потреби використання онлайн функцій (якщо такі присутні у додатку).

5. Оптимізація:

Варто проводити якісне і докладне тестування аби виявити якомога більше недоліків роботи мобільного застосунку. Але перш за все треба слідкувати за правильним використанням процесів і контролювати їхнє завершення, якщо вони не роблять це автоматично. Також точно не буде зайвим правильне розподілення синхронних і асинхронних задач. Таким чином один процес не заважатиме іншому. Наприклад обчислення не блокуватимуть графічний інтерфейс.

6. Приватність і безпека:

Вимагати дозволи потрібно тільки на те, без чого додаток не працюватиме. Також корисно буде додати можливість поступового запити на дозволи, щоб надавати дозвіл тільки у момент, коли користувач намагається використати функцію додатку, що не може працювати без цього дозволу.

7. Підтримка:

Потрібно звертати увагу на відгуки аудиторії, вивчати проблеми, з якими вони зіткнулися та вчасно виправляти їх. Гарною практикою буде враховувати побажання користувачів. Наприклад запити на додавання нових функцій. Але перш за все треба оцінити всі ризики та переваги.

2.2. Опис підходів розробки та подробиць структури проекту

Щоб створити додаток будемо притримуватися шаблону архітектури MVVM, включаючи елементи Clean Architecture. Таким чином ми розподілимо проект на складові: дані, логіка, представлення [4].

MVVM архітектура пропонує чітке розподілення обов’язків між частинами програми. На рисунку 2.1 представлена проста схема цієї архітектури.

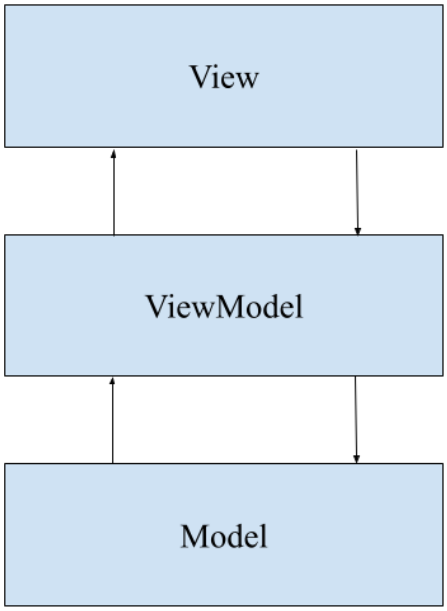


Рисунок 2.1

View – це шар який відповідає за графічне відображення. Це і є наш інтерфейс користувача, який ми бачимо при використанні додатку. Сюди можуть входити, як повні екрани, так і компоненти, з яких вони складаються. Займається обміном подіями і даними із ViewModel.

ViewModel – цей шар відповідає за взаємодію між представленням і даними. Він реагує на зміни у Model та View. Він є посередником передачі даних.

Model – тут зберігаються дані та знаходиться бізнес-логіка. Сюди можуть входити бази даних, інтерфейси взаємодії з цими базами, також логіка отримання і внесення даних.

Дана архітектура забезпечить не тільки чітку структуру, з якою зручно працювати, а і чудову масштабованість та просте тестування різних частин проекту у майбутньому.

2.2.1. Зберігання даних

Ми створюємо додаток виду To Do List, тому нам важливо зберігати дані про заплановані справи і завдання локально. Для цього скористаємося бібліотекою Room, яка забезпечить зручний і надійний спосіб роботи із базою даних SQLite.

Цей вибір зумовлений, раніше зробленим, аналізом деяких аспектів, які впливають на позитивний досвід користувача від використання додатку. У нас не завжди є безумовний доступ до інтернету, тому це не має впливати на правильну роботу програми. А ще цей підхід дозволить зменшити витрати на оплату сторонніх онлайн-сервісів.

Давайте розглянемо схематичний вигляд бібліотеки Room у структурі проекту, який зображений на рисунку 2.2

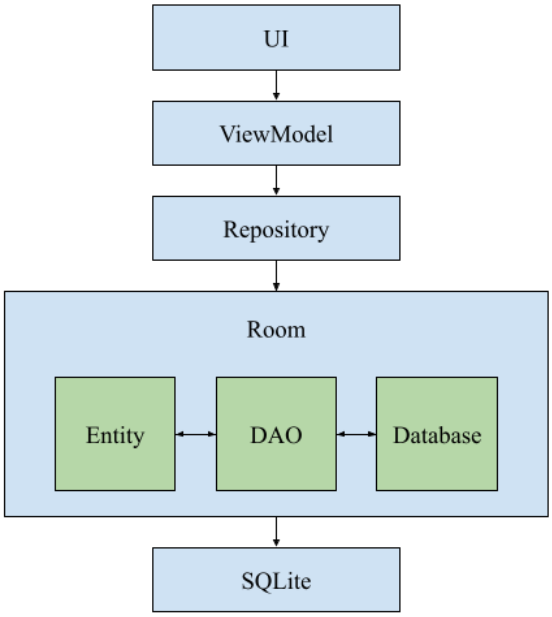


Рис. 2.2

Entity – являє собою таблицю у базі даних. Параметри в ній є стовпчиками таблиці, їх можна зручно налаштувати, щоб використовувати як характеристики до об’єкту, дані якого будуть внесені у таблицю. Також деяким параметрам, за потреби, можна додати можливість приймати порожні значення, що може бути корисно у окремих випадках. Entity у проекті матиме такий вигляд:

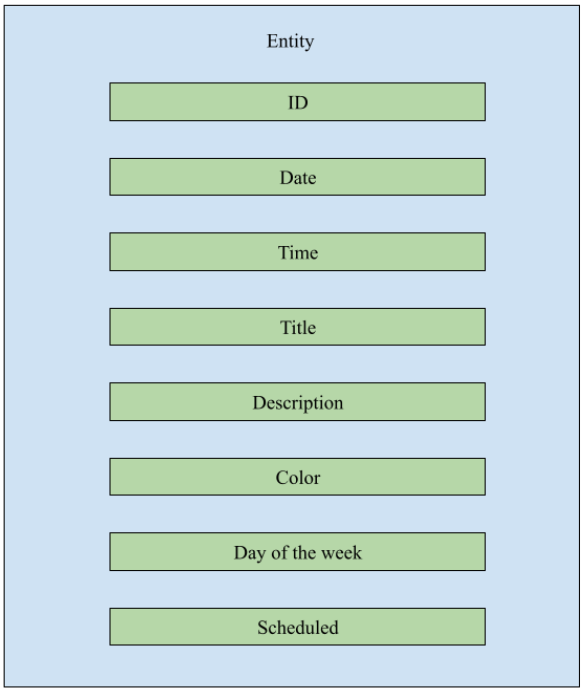


Рис. 2.3

Id – буде налаштовано на автоматичну генерацію, починаючи з нуля. Він буде ключовим параметром унікальності кожного об’єкту.

DAO (Data Access Object) – це інтерфейс для доступу до бази даних. Є способом створення запитів на читання, вставлення, оновлення та видалення даних із таблиці.

Database – являє собою основний елемент роботи з Entity та DAO, а також забезпечує можливість взаємодіяти із самою локальною базою даних. Це основний контролер, який керує усіма таблицями.

Також є додатковий елемент, це конвертери. Вони допомагають автоматизувати використання деяких параметрів, що потребують різного виду зберігання або представлення. Це обумовлено конкретними типами даних параметрів. Часто застосовується якщо використовуються такі дані, як дата і час.

2.2.2. Відображення

Задля продуктивності і зручності розробки, прийнято рішення використовувати інструмент для створення UI – Jetpack Compose, замість звичайної XML розмітки.

Програма матиме єдину точку входу – MainActivity, яка в свою чергу викликатиме головну Composable-функцію, яка об’єднає у собі всі елементи відображення.

Основна Composable-функція з елементами каркасу і елементами навігації представлена на рисунку 2.4 .

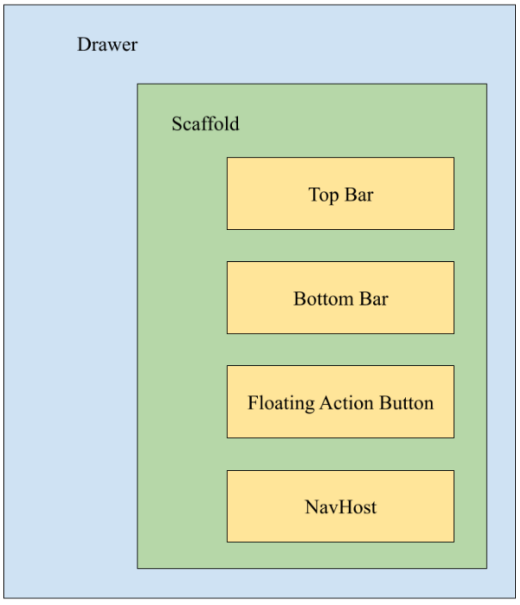


Рис. 2.4

Drawer – бокова «штора», являє собою меню для потенційного розміщення додаткових елементів, таких як: кнопки переходів на другорядні екрани, кнопки виклику діалогів різного призначення, від інформаційного до функціонального, елементи керування певними налаштуваннями додатку.

Scaffold – це каркас, який забезпечує дуже легкий спосіб для компонування UI. Він допомагає об’єднати верхнє меню, нижнє навігаційне меню, контент, що має бути розташованим між ними, а також плаваючу кнопку дій, яка знаходиться поверх контенту.

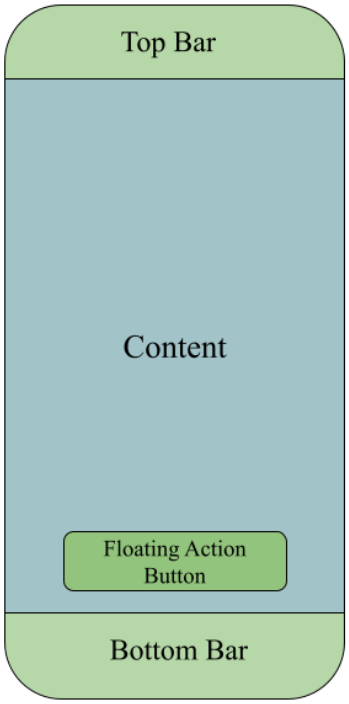


Рис. 2.5

Top Bar – графічний елемент, що знаходиться у верхній частині екрану телефону, може містити назву додатку, інструменти, кнопки для виклику інших типів меню.

Bottom Bar – нижнє навігаційне меню, що використовується для переходів між екранами. Наповненням Bottom Bar є Navigation Bar.

Floating Action Button – кнопка, що знаходиться поверх частини інтерфейсу додатка, яку не займають вищезгадані 2 елемента, у даному

випадку те місце займають екрани. Виконує роль запуску однієї з основних функцій програми.

Nav Host – хоча він зображений на схемі з рисунку 2.4, візуально користувач бачить контент одного з екранів. Nav Host є частиною роботи навігації у додатку. На рисунку 2.6 зображена загальна схема структури навігації.

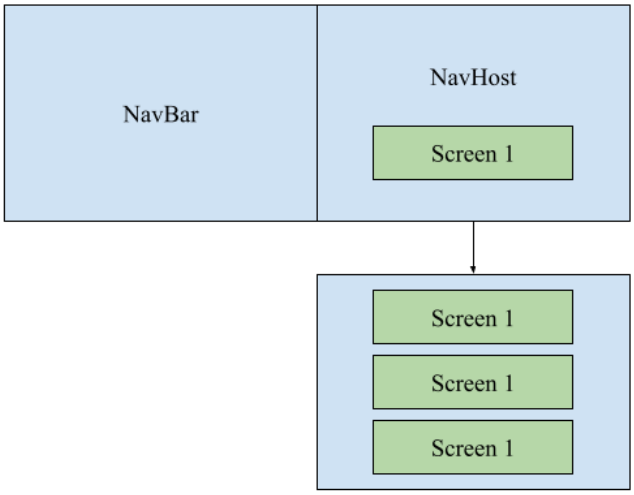


Рис. 2.6

Якщо казати про процес навігації, то він відбувається таким чином: користувач має почати взаємодію з функціональним елементом, наприклад кнопкою. Яка в свою чергу має бути правильно візуально і інформаційно оформлена, щоб користувач розумів куди він зможе перейти, натиснувши на неї. Далі кнопка має відправити команду переходу за певним маршрутом, Nav Controller приймає запит і маршрут. Далі Nav Host перевіряє чи існує пункт призначення за даним маршрутом і коли результат позитивний – передає потрібний екран на процес рендеру.

Процес навігації, з назвами етапів, схематично показано на рисунку 2.7

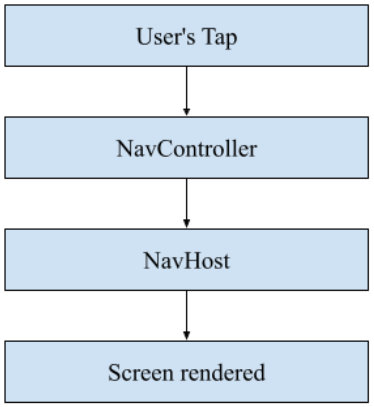


Рис 2.7

2.2.3. Сповіщення

У додатка мають бути власні сповіщення для нагадування про заплановані справи. Також як і у випадку зі збереженням даних, варто відмовитись від онлайн-сервісів і надати перевагу локальним сповіщенням.

Відсутність сповіщень негативно впливатиме на досвід використання додатку. Адже без нагадувань доведеться багато разів переглядати поставлені для себе задачі. А у нас все таки не нотатник, а інструмент планування, для якого важливо підстрахувати користувача, якщо він щось забув, тому варто врахувати людський фактор і додати нагадування. Але важливо пам’ятати, що нагадування не мають на меті замінити будильник, тому планування таких задач, як вчасно прокинутися зранку, не буде правильним способом використання застосунку.

Вимоги до роботи сповіщень наступні: надсилати сповіщення своєчасно, час сповіщення має відповідати часу отриманому Entity у параметр Time. Задля запобігання несподіваним ситуаціям, наприклад: телефон раптово розрядився і вимкнувся, або користувачеві для інших цілей довелося зробити

перезавантаження чи послідовне вимкнення та ввімкнення пристрою, у цьому разі варто налаштувати відновлення встановлених нагадувань.

Тобто ми впроваджуємо до нашої зв’язки data – logic – view ще одну складову – utils, але вона є другорядною. В утилітах ми маємо налаштувати в першу чергу сервіси, які будуть правильно отримувати дані з нашої бази даних. Отже встановлюємо один окремий зв’язок нашої ViewModel та Room для отримання інформації часу сповіщень, а далі треба створити канал сповіщень, налаштувати планувальник сповіщень, встановити логіку сповіщень, а вже потім додати Boot Receiver, щоб якраз відбувався процес відновлення сповіщень після ввімкнення телефону. І звісно не можна забувати налаштувати дозволи у Manifest.

РОЗДІЛ 3. СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ

3.1. Побудова графічного інтерфейсу

Для створення дизайну основних 3 екранів було обрано платформу Canva, через низький поріг входу. Її використання просте навіть для новачків, але вона має достатній функціонал для створення прототипів дизайну. Перший варіант дизайну для додатку був пробним і вже на етапі його розробки було вирішено, що це буде тестовий варіант, і правки вноситимуться уже під час верстки коду.

Готовий прототип має такий вигляд:

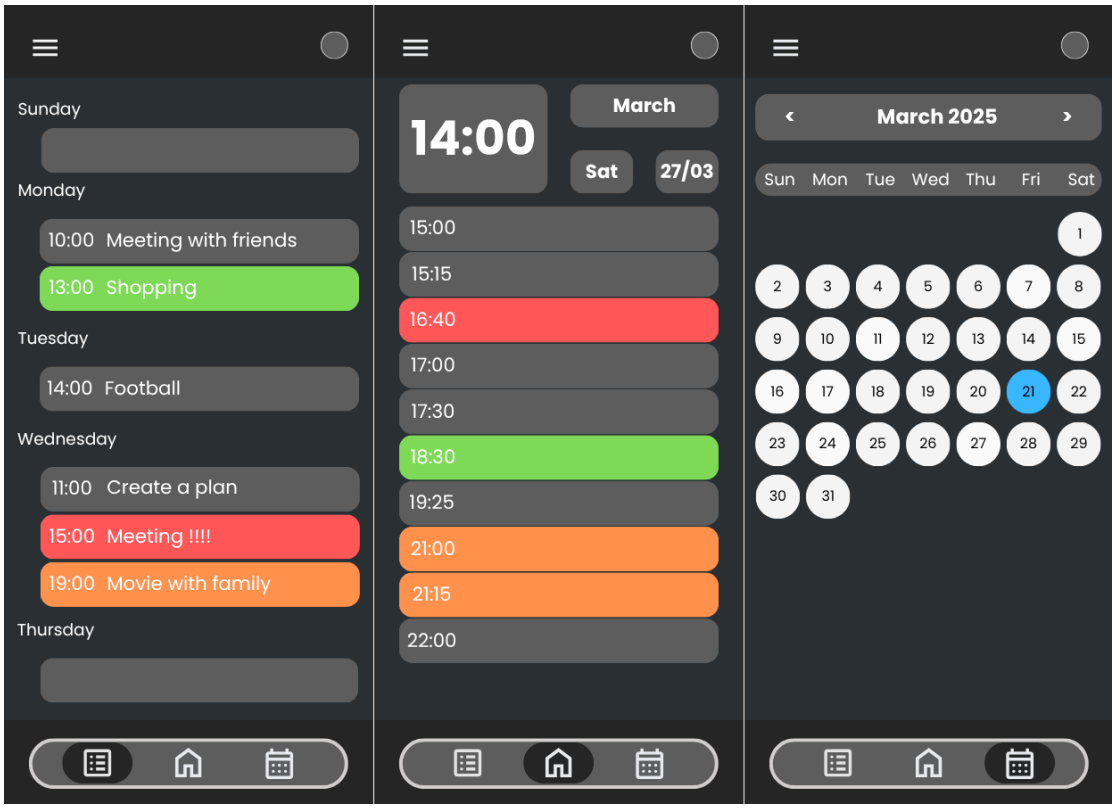


Рис. 3.1

Цей прототип пізніше був змінений і уподібнений до стилю бібліотеки Material 3, адже ми використовуємо framework Jetpack Compose, а ця бібліотека

вбудована і офіційно підтримується. Це нам надає переваги у зручності використання та наданні професійного вигляду схожого на продукти Google.

Далі робота йде у IDE Android Studio. Отже для початку створимо окремі компоненти для наповнення головної Composable функції проекту. Працюємо за схемою на рис. 2.4, але починаючи з компонентів і закінчуючи обгортками.

Важлива ремарка: далі буде показано частини коду основних елементів відображення, повний вихідний код програми знаходиться у додатку А.

Top Bar:

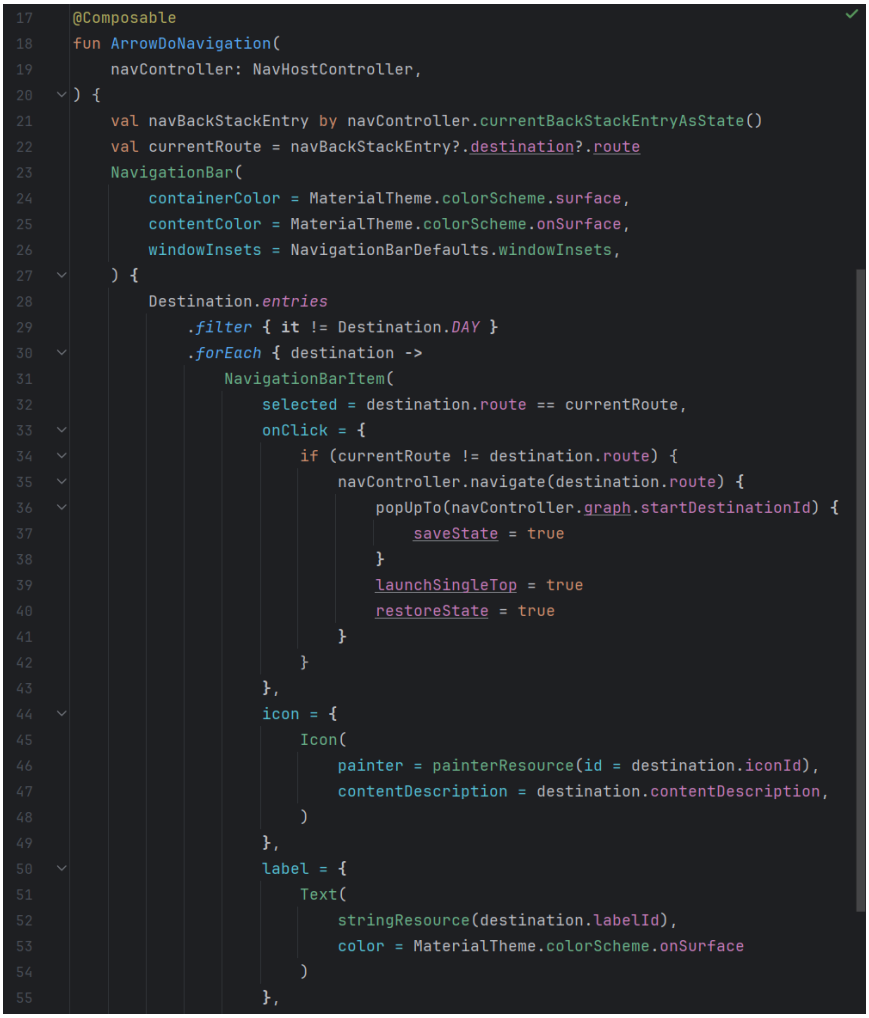
```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ArrowDoTopBar(
    onMenuClick: () -> Unit,
) {
    CenterAlignedTopAppBar(

        colors = TopAppBarDefaults.topAppBarColors(
            containerColor = MaterialTheme.colorScheme.surface,
            titleContentColor = MaterialTheme.colorScheme.onSurface,
            navigationIconContentColor = MaterialTheme.colorScheme.onSurface,
            actionIconContentColor = MaterialTheme.colorScheme.onSurface,
        ),
        title = {
            Text(
                text = stringResource("Arrow Do"),
                maxLines = 1,
                overflow = TextOverflow.Ellipsis
            )
        },
        navigationIcon = {
            IconButton(onClick = onMenuClick) {
                Icon(
                    imageVector = Icons.Filled.Menu,
                    contentDescription = "Menu"
                )
            }
        },
        actions = {
            IconButton(onClick = { }) {
                Icon(
                    imageVector = Icons.Filled.AccountCircle,
                    contentDescription = "User"
                )
            }
        }
    )
}
```

Рис. 3.2

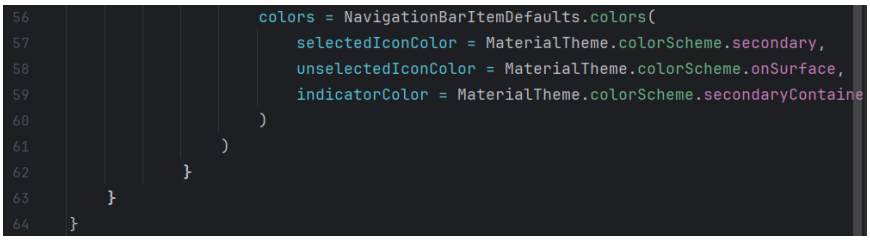
У цьому елементі ми задаємо зовнішній вигляд та додаємо функцію кліку по кнопці меню. Також ми передаємо callback для визначення поведінки при натисканні.

Bottom Bar:



```
@Composable
fun ArrowDoNavigation(
    navController: NavHostController,
) {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route
    NavigationBar(
        containerColor = MaterialTheme.colorScheme.surface,
        contentColor = MaterialTheme.colorScheme.onSurface,
        windowInsets = NavigationBarDefaults.windowInsets,
    ) {
        Destination.entries
            .filter { it != Destination.DAY }
            .forEach { destination ->
                NavigationBarItem(
                    selected = destination.route == currentRoute,
                    onClick = {
                        if (currentRoute != destination.route) {
                            navController.navigate(destination.route) {
                                popUpTo(navController.graph.startDestinationId) {
                                    saveState = true
                                }
                                launchSingleTop = true
                                restoreState = true
                            }
                        }
                    },
                    icon = {
                        Icon(
                            painter = painterResource(id = destination.iconId),
                            contentDescription = destination.contentDescription,
                        )
                    },
                    label = {
                        Text(
                            stringResource(destination.labelId),
                            color = MaterialTheme.colorScheme.onSurface
                        )
                    },
```

Рис. 3.3



```
                    colors = NavigationBarItemDefaults.colors(
                        selectedIconColor = MaterialTheme.colorScheme.secondary,
                        unselectedIconColor = MaterialTheme.colorScheme.onSurface,
                        indicatorColor = MaterialTheme.colorScheme.secondaryContaine
                    )
                )
            }
    }
}
```

Рис. 3.4

У Navigation Bar ми маємо цикл для кожного із трьох основних екранів, щоб правильно створити екземпляри кнопок для переходу у відповідності з їхніми маршрутами. У даному випадку код взаємодії Nav Host та класу сталих значень опускається для не нагромадження документа.

FAB (Floating Action Button):

```
@Composable
fun ArrowDoFAB(
    currentRoute: String?,
    onFabClick: () -> Unit
) {
    if (
        currentRoute == Destination.HOME.route ||
        currentRoute == Destination.WEEK.route ||
        currentRoute == Destination.DAY.route
    ) {
        FloatingActionButton(
            onClick = onFabClick,
            containerColor = MaterialTheme.colorScheme.primaryContainer,
            contentColor = MaterialTheme.colorScheme.onPrimaryContainer
        ) {
            Icon(Icons.Default.Add, contentDescription = "Add")
        }
    }
}
```

Рис. 3.5

Дана плаваюча кнопка викликається із головної функції, але налаштована бути присутньою лише на двох головних і одному додатковому екрані.

Drawer:

```
@Composable
fun ArrowDoDrawer(
    drawerState: DrawerState,
    content: @Composable () -> Unit,
    onAboutClick: () -> Unit
) {
    val scope = rememberCoroutineScope()
    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            ModalDrawerSheet(
                drawerContainerColor = MaterialTheme.colorScheme.surfaceBright,
                drawerContentColor = MaterialTheme.colorScheme.onSurface,
                modifier = Modifier.width(320.dp),
                ) {
                IconButton(
                    onClick = {
                        scope.launch {
                            drawerState.close()
                        }
                    }
                ) {
                    Icon(
                        Icons.Default.Close,
                        contentDescription = "Close",
                    )
                }
                HorizontalDivider()
                NavigationDrawerItem(
                    icon = { Icon(Icons.Default.Info, contentDescription = "About") },
                    label = { Text(text = stringResource("About")) },
                    selected = false,
                    onClick = {
                        scope.launch {
                            drawerState.close()
                        }
                        onAboutClick()
                    },
```

Рис. 3.6

```
                    colors = NavigationDrawerItemDefaults.colors(
                        unselectedIconColor = MaterialTheme.colorScheme.onSurface,
                        unselectedTextColor = MaterialTheme.colorScheme.onSurface,
                    )
                )
            }
        },
        gesturesEnabled = true,
        content = content
    )
}
```

Рис 3.7

Як все об’єднано у головній функції:

```
ArrowDoTheme {
    ArrowDoDrawer(
        drawerState = drawerState,
        onAboutClick = {
            aboutDialog.value = true
        },
        content = {
            Scaffold(
                topBar = {
                    ArrowDoTopBar(
                        onMenuClick = {
                            scope.launch {
                                drawerState.apply {
                                    if (isClosed) open() else close()
                                }
                            }
                        }
                    )
                },

                bottomBar = {
                    ArrowDoNavigation(
                        navController = navController
                    )
                },
                floatingActionButton = {
                    ArrowDoFAB(
                        currentRoute = currentRoute,
                        onFabClick = {
                            when (currentRoute) {
                                Destination.HOME.route -> addTaskDialog.value = true
                                Destination.WEEK.route -> addScheduledTaskDialog.value =
                                    true

                                else -> addDayTaskDialog.value = true
                            }
                        }
                    )
                }
```

Рис. 3.8

```
            ) { innerPadding ->
                AppNavHost(
                    navController,
                    startDestination,
                    modifier = Modifier.padding(innerPadding),
                    viewModel = viewModel
                )
            }
        }
    )
```

Рис. 3.9

Якщо говорити просто, то тут йдуть звичайні виклики раніше налаштованих Composable-функцій, звісно із передачею потрібних параметрів. Також у коді згадані виклики діалогів в залежності від екрану, на якому знаходиться плаваюча кнопка.

Наповнення екранів поки що опустимо. У результаті у нас виходить ось такий вигляд графічного інтерфейсу.

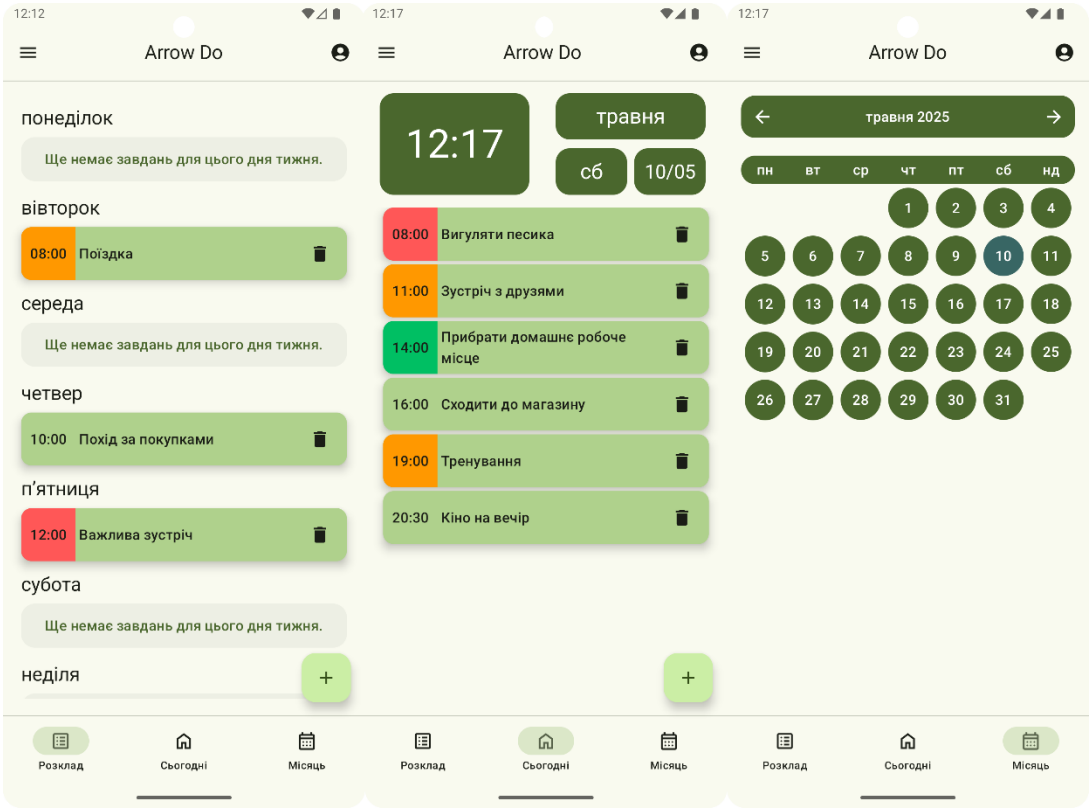


Рис. 3.10

3.2. Налаштування бази даних

Спочатку додаємо залежності у файл build.gradle.kts, для можливостей користуватися основними можливостями бібліотеки Room і взаємодією з життєвим циклом ViewModel, а також додаються розширення для Room. Розширення полегшують використання бібліотеки, а також дають можливості для роботи з асинхронними процесами.

```
dependencies {
    implementation(libs.androidx.room)
    implementation(libs.androidx.room.ktx)
    ksp(libs.androidx.room.ksp)
    implementation(libs.androidx.lifecycle.viewmodel)
```

Рис. 3.11

Далі створюємо екземпляр таблиці і додаємо параметри, вказуючи потрібний нам для використання тип даних. Код конверторів для типів даних локальних дати та часу, а також типу дня – опускаємо, але він буде включений у повний вихідний код у додатку А.

```
@Entity(tableName = "tasks")
data class Task(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val day: LocalDate? = null,
    val time: LocalTime,
    val title: String,
    val description: String,
    val color: Int,
    val dayType: DayOfWeek? = null,
    val scheduled: Boolean = false,
)
```

Рис. 3.12

Наступним кроком створюємо об’єкт для доступу до даних таблиці. Тут додаються асинхронні функції, які не блокують основний процес. Для запитів використовується SQL.

```
@Dao
interface TaskDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(task: Task)

    @Delete
    suspend fun delete(task: Task)

    @Query("SELECT * FROM tasks ORDER BY time ASC")
    fun getAllTasks(): Flow<List<Task>>

    @Query("SELECT * FROM tasks ORDER BY time ASC")
    fun getAllTasksForBoot(): List<Task>

    @Query("SELECT * FROM tasks WHERE day = :date ORDER BY time ASC")
    fun getDayTasks(date: LocalDate): Flow<List<Task>>
}
```

Рис. 3.13

Далі налаштовується абстрактний клас, що успадковується від RoomDatabase. У ньому отримується список таблиць, і описується створення бази даних. Також тут використовується Singleton-патерн, він забезпечує одноразове створення бази даних для усього додатку. Іншими словами – не дає створити зайві бази даних.

```
@Database(entities = [Task::class], version = 1, exportSchema = false)
@TypeConverters(Converter::class)
abstract class TaskDatabase : RoomDatabase() {
    abstract fun taskDao(): TaskDao

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

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

Рис. 3.14

Потім ми створюємо репозиторій у який закладаємо логіку роботи з даними, тобто отримання і внесення даних. Також у ньому ми використаємо можливості Kotlin Coroutines, тобто співпрограм, що дозволять відслідковувати зміни одразу після їх внесення. Такий підхід не блокуватиме основні процеси і забезпечить належну оптимізацію.

```
class TaskRepository(private val taskDao: TaskDao) {
    val allTasks: Flow<List<Task>> = taskDao.getAllTasks()

    fun getTasksForDay(date: LocalDate): Flow<List<Task>> {
        return taskDao.getDayTasks(date)
    }
    fun getTasksForToday(): Flow<List<Task>> {
        val today = LocalDate.now()
        val todayDayType = today.dayOfWeek
        return taskDao.getAllTasks().map { tasks ->
            tasks.filter { task ->
                (!task.scheduled && task.day == today) ||
                        (task.scheduled && task.dayType == todayDayType)
            }
        }
    }
    fun getTasksForTodayBoot(): List<Task> {
        val today = LocalDate.now()
        val todayDayType = today.dayOfWeek
        val allTasks = taskDao.getAllTasksForBoot()
        return allTasks.filter { task ->
            (!task.scheduled && task.day == today) ||
                    (task.scheduled && task.dayType == todayDayType)
        }

    }
    fun getTasksForDayType(dayOfWeek: DayOfWeek): Flow<List<Task>> {
        return taskDao.getAllTasks().map { tasks ->
            tasks.filter { task -> task.scheduled && task.dayType == dayOfWeek }
        }
    }
    suspend fun addTask(task: Task) {
        taskDao.insert(task)
    }
    suspend fun deleteTask(task: Task) {
        taskDao.delete(task)
    }
}
```

Рис. 3.15

А вже тільки після нього створюється ViewModel, для того щоб відокремити прямої взаємодії із Room.

```
open class TaskViewModel(private val repository: TaskRepository) : ViewModel() {
    val tasks: StateFlow<List<Task>> = repository.allTasks
        .stateIn(viewModelScope, SharingStarted.Lazily, emptyList())
    private val _selectedDate = mutableStateOf<LocalDate?>( value: null)
    val selectedDate: State<LocalDate?> = _selectedDate
    fun addTask(task: Task) {
        viewModelScope.launch {
            repository.addTask(task)
        }
    }
    fun deleteTask(task: Task) {
        viewModelScope.launch {
            repository.deleteTask(task)
        }
    }
    fun getTasksForDay(date: LocalDate): Flow<List<Task>> {
        return repository.getTasksForDay(date)
    }
    fun getTasksForToday(): Flow<List<Task>> {
        return repository.getTasksForToday()
    }
    fun getTasksForTodayBoot(): List<Task> {
        return repository.getTasksForTodayBoot()
    }
    fun getTasksForDayType(dayOfWeek: DayOfWeek): Flow<List<Task>> {
        return repository.getTasksForDayType(dayOfWeek)
    }
    fun setSelectedDate(date: LocalDate) {
        _selectedDate.value = date
    }
}
```

Рис. 3.16

Так як ми не використовуємо Hilt, нам треба налаштувати функцію створення ViewModel із параметрами. Для цього створимо клас Factory.

```
class TaskViewModelFactory(private val repository: TaskRepository) :ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(TaskViewModel::class.java)) {
            @Suppress( ...names: "UNCHECKED_CAST")
            return TaskViewModel(repository) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

Рис. 3.17

3.3. Створення основного об’єкту

Завдання які ми створюватимемо будуть мати вигляд невеликої картки. Карта матиме 2 текстових поля у яких зазначатиметься час на який заплановано завдання, назву завдання (не більше 30 символів) та кнопку видалити, а також кольоровий маркер (зелений, помаранчевий або червоний). Маркер може бути відсутнім, він є елементом який допоможе користувачеві візуально розмітити більш важливі завдання. Картка матиме вигляд зображений рисунку 3.18.

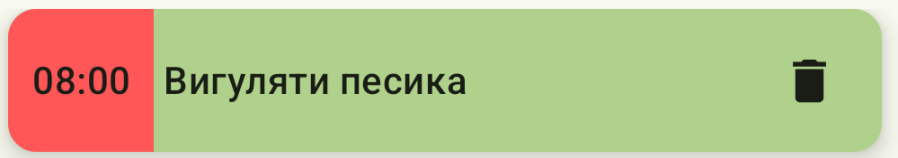
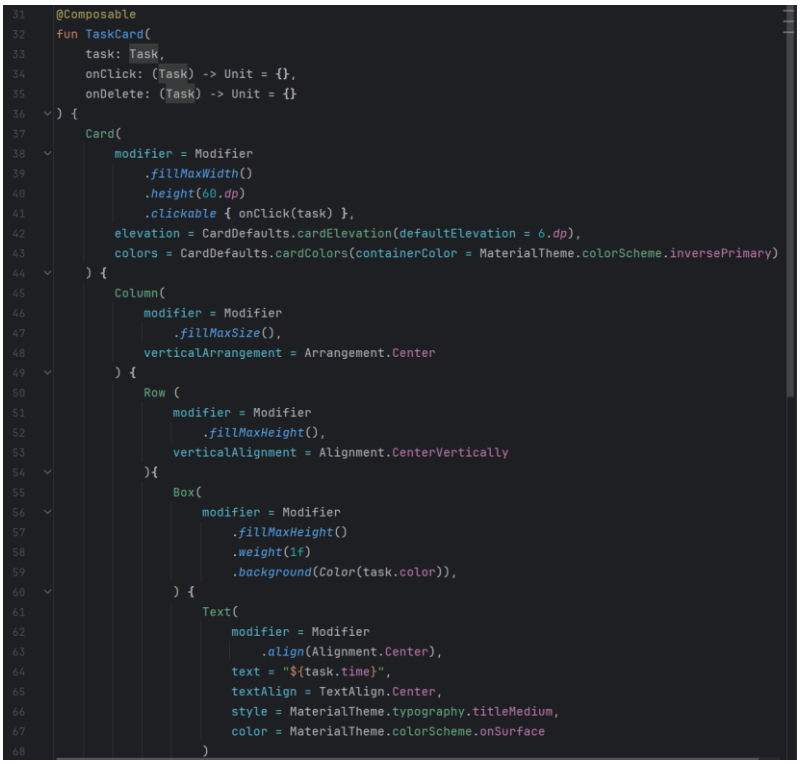


Рис. 3.18

Екземпляр картки достатньо створити один раз, він буде використовуватися у ролі шаблону візуального відображення завдання.



```
@Composable
fun TaskCard(
    task: Task,
    onClick: (Task) -> Unit = {},
    onDelete: (Task) -> Unit = {}
) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .height(60.dp)
            .clickable { onClick(task) },
        elevation = CardDefaults.cardElevation(defaultElevation = 6.dp),
        colors = CardDefaults.cardColors(containerColor = MaterialTheme.colorScheme.inversePrimary)
    ) {
        Column(
            modifier = Modifier
                .fillMaxSize(),
            verticalArrangement = Arrangement.Center
        ) {
            Row (
                modifier = Modifier
                    .fillMaxHeight(),
                verticalAlignment = Alignment.CenterVertically
            ){
                Box(
                    modifier = Modifier
                        .fillMaxHeight()
                        .weight(1f)
                        .background(Color(task.color)),
                ) {
                    Text(
                        modifier = Modifier
                            .align(Alignment.Center),
                        text = "${task.time}",
                        textAlign = TextAlign.Center,
                        style = MaterialTheme.typography.titleMedium,
                        color = MaterialTheme.colorScheme.onSurface
                    )
```

Рис. 3.19

```
                }
                Text(
                    modifier = Modifier
                        .weight(4f)
                        .padding(horizontal = 4.dp),
                    text = task.title,
                    textAlign = TextAlign.Start,
                    overflow = TextOverflow.Ellipsis,
                    style = MaterialTheme.typography.titleMedium,
                    color = MaterialTheme.colorScheme.onSurface
                )
                IconButton(
                    modifier = Modifier
                        .weight(1f),
                    onClick = {
                        onDelete(task)
                    },
                ) {
                    Icon(
                        imageVector = Icons.Default.Delete,
                        contentDescription = "DeleteTaskCard",
                        tint = MaterialTheme.colorScheme.onSurface
                    )
                }
            }
        }
    }
}
```

Рис 3.20

У шаблоні ми встановлюємо callbacks для подальшої логіки видалення та перегляду докладної інформації завдання.

На рисунку 3.21 приклад відображення створених завдань, узятий із домашнього екрану, у якому вказана логіка виклику функції видалення через ViewModel відповідаючи на callback натискання кнопки видалити, а також

поведінка виклику діалогу для огляду повної інформації вказаної користувачем.

```
173     } else items(tasks) { task ->
174         TaskCard(
175             task = task,
176             onDelete = { taskToDelete ->
177                 viewModel.deleteTask(taskToDelete)
178             },
179             onClick = { clickedTask ->
180                 taskToShow = clickedTask
181                 showTaskDialog.value = true
182             }
183         )
184         Spacer(modifier = Modifier.height(4.dp))
185     }
```

Рис. 3.21

3.4. Опис роботи додатку

На рисунку 3.10 можна побачити наповнення екранів. Тому почнемо по черзі розбирати яким чином були розподілені обов’язки між екранами.

1. Домашній екран:

Має показувати всі завдання заплановані на сьогоднішню дату. Це його головне завдання. Сюди також додаватимуться повторювані завдання, які будуть створені на екрані планування на тиждень, по мірі того як співпадатиме з день тижня, який зараз, із днем тижня у розкладі. Також для зручності було додано інформаційний блок, розташований у верхній частині екрану. У цьому блоці показується поточний час, який оновлюється щосекунди, але користувач бачитиме лише зміну хвилин і годин. Далі є інформація про те, який зараз місяць, ця інформація надається у вигляді слова – назви. Також у блок додано відображення назви поточного дня тижня (у скороченому варіанті), а також дата і місяць у вигляді цифрових значень.

2. Екран планування на тиждень:

Показує завдання заплановані на певний день тижня. Має вигляд списку з можливістю скролу. У списку містяться назви днів тижня і під ними групуються завдання, які відносяться до потрібного дня тижня.

3. Екран планування на місяць:

Має представити сітку днів на поточний місяць (або інший, якщо його було вибрано). Дати розташовані відповідно до дня тижня. Дні мають бути клікабельними. Вони вестимуть користувача на екран схожий на домашній, але показуватимуть завдання на обрану дату.

Процес створення завдання реалізований у вигляді діалогів. І як було вказано раніше, що плаваюча кнопка викликатиме потрібний діалог, в залежності від того, на якому екрані вона зараз знаходиться.

Ось так виглядає діалог домашнього екрану:

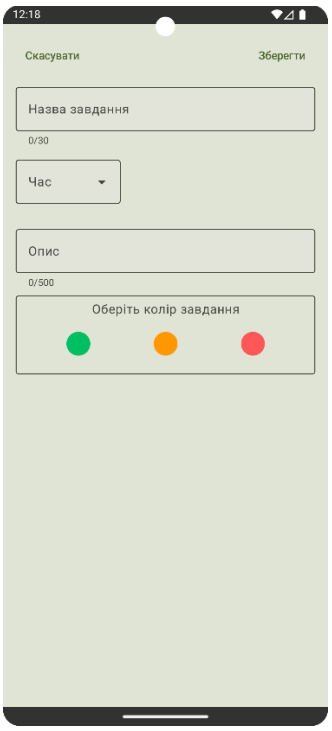


Рис. 3.22

Ви можете помітити, що ось тут, нарешті, використовується більше параметрів таблиці. Тут ми просимо користувача дати назву завданню. Встановити час для відображення і нагадування. Опціонально додати опис, якщо того вимагає завдання. У опису обмеження у 500 символів. Також другим необов'язковим параметром є колір для маркеру. Параметр дати тут вноситься автоматично. Так само автоматично і визначається сьогоднішня дата завдяки бібліотеці java.time. А параметри дня тижня і плановості – приймають значення null, адже нам не потрібно повторювати завдання, воно просто буде створено на сьогодні.

Рисунок 3.23 зображує нам діалог для створення завдання але на екрані розкладу на тиждень.

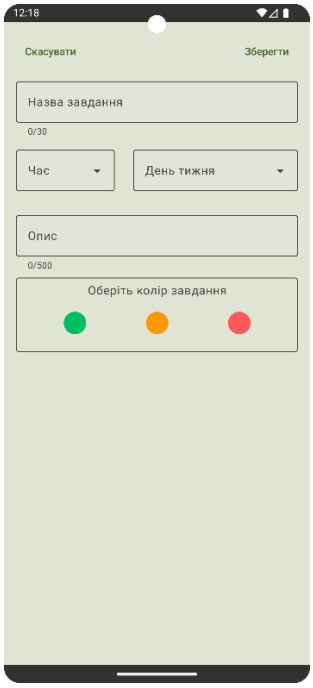


Рис. 3.23

У цьому випадку у нас додається можливість змінювати параметр дня тижня, який є обов'язковим. Але тепер нам неважливо, яка буде дата, адже це створення повторюваного завдання, тому ми її не вказуємо.

Якщо взяти обидва діалоги то можна виділити, що у нас 2 або 3 параметра є обов'язковими, тому тут встановлено обмеження, яке не дозволяє

спрацювати кнопці «Зберегти» за призначенням, замість цього є візуальні попередження, що дані поля мають бути заповненими. Це зроблено для запобігання небажаному внесенню неповноцінних даних у таблицю. Тому що, без Часу і назви завдання ми не зможемо запланувати нагадування і відобразити ці значення на картці. А без дня тижня у нас не буде працювати повторюване завдання.

Інформація з необов’язкових параметрів займає багато місця і не вміщається у картку. А робити картку великою нераціонально у нашому випадку. Тому нам потрібен діалог для показу повної інформації.

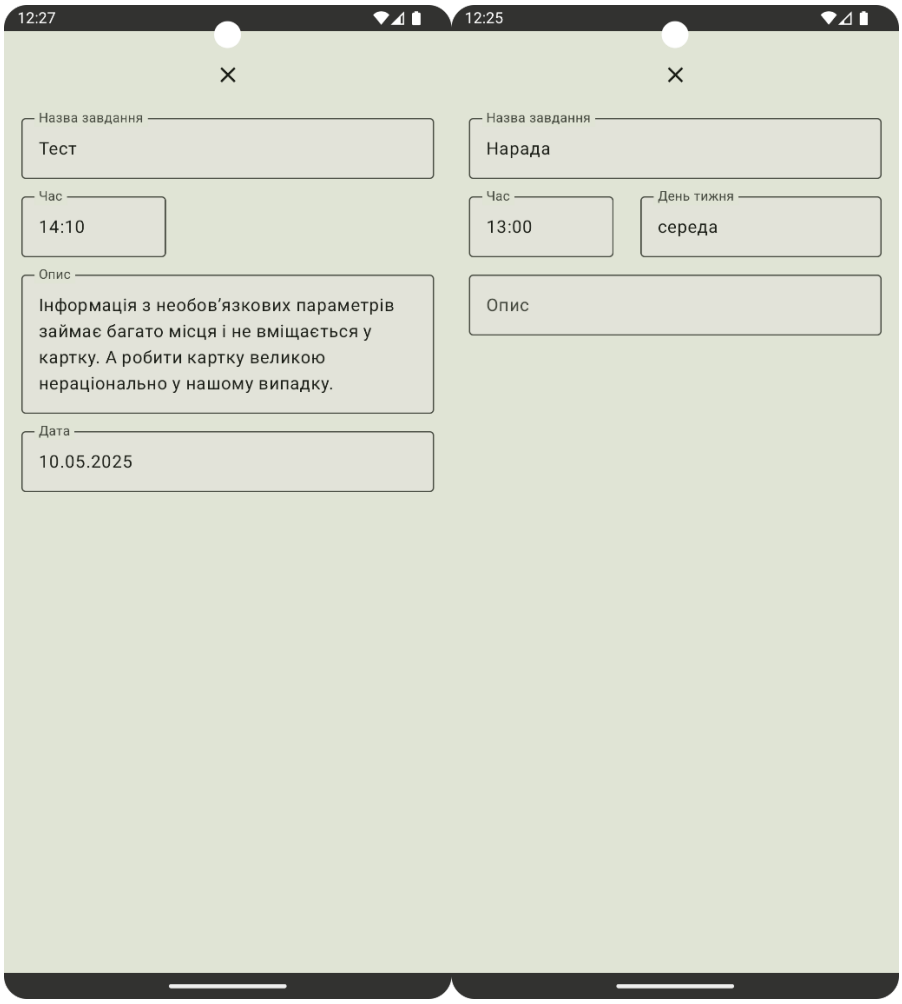
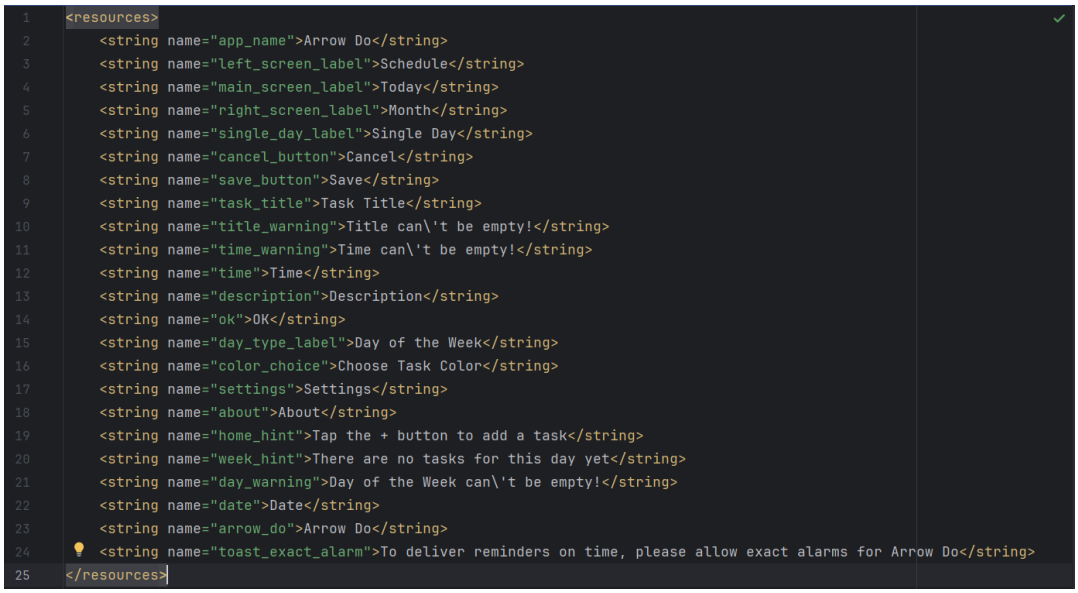


Рис. 3.24

За наявних конкретних значень у параметрі, для кожного типу завдань відповідно, ці параметри будуть відображені у діалозі, який викликається по натисканню на картку.

3.5. Локалізація

Локалізація додатку важлива для охоплення більшої аудиторії. Людям набагато приємніше буде користуватися додатком, якщо текст у ньому буде їхньою мовою. Щоб зручно і швидко виконати переклад додатку, треба заздалегідь винести всі рядки тексту у файл strings.xml у папці …/res/values, відповідно розмітивши їх. Після цього у коді треба зробити посилання на потрібний рядок за допомогою ідентифікатора name. Далі достатньо створити у теці …/res директорії із назвами values-xx (ось тут дві букви коду мови) і додати в них файл language.xml (для кожної мови можна створити відповідну назву). У цей файл треба перенести вміст із файлу strings.xml, замінивши вміст рядків на слова потрібною мовою, при цьому не можна змінювати name. І так можна для кожної мови, яку потрібно додати. Система сама потім підтягуватиме потрібний переклад в залежності від мови обраної в налаштуваннях телефону. Якщо ж у додатку не буде відповідної мови система використає стандартний варіант із файлу strings.xml. Наприклад: у цьому файлі може бути англомовний переклад, ми ще додали файл з перекладом українською, а у користувача системна мова німецька, тоді система покаже англійський переклад як стандартний.



```xml
<resources>
    <string name="app_name">Arrow Do</string>
    <string name="left_screen_label">Schedule</string>
    <string name="main_screen_label">Today</string>
    <string name="right_screen_label">Month</string>
    <string name="single_day_label">Single Day</string>
    <string name="cancel_button">Cancel</string>
    <string name="save_button">Save</string>
    <string name="task_title">Task Title</string>
    <string name="title_warning">Title can\'t be empty!</string>
    <string name="time_warning">Time can\'t be empty!</string>
    <string name="time">Time</string>
    <string name="description">Description</string>
    <string name="ok">OK</string>
    <string name="day_type_label">Day of the Week</string>
    <string name="color_choice">Choose Task Color</string>
    <string name="settings">Settings</string>
    <string name="about">About</string>
    <string name="home_hint">Tap the + button to add a task</string>
    <string name="week_hint">There are no tasks for this day yet</string>
    <string name="day_warning">Day of the Week can\'t be empty!</string>
    <string name="date">Date</string>
    <string name="arrow_do">Arrow Do</string>
    <string name="toast_exact_alarm">To deliver reminders on time, please allow exact alarms for Arrow Do</string>
</resources>
```

Рис. 3.25

ВИСНОВКИ

Під час виконання даної роботи був проведений докладний аналіз процесу розробки мобільних додатків та їх впливу на сучасне життя.

Було проведено порівняльний аналіз існуючих готових рішень, які також націлені на підвищення продуктивності та правильного розподілення часу. Це допомогло визначити позитивні і негативні фактори впливу на якість додатку в цілому, якщо дивитися з боку користувача. Цей аналіз дав змогу сформувати потрібні критерії для створення власного додатку.

Під час проведення аналізу обраної теми, стало зрозуміло, що суспільство потребує допомоги із вирішенням питання самоконтролю і планування. Також було виявлено основні проблеми, які має вирішувати додаток для планування. Не дивлячись на велику кількість готових рішень, помітно, що хоч в нас і є такий великий вибір, у всього є свої недоліки і є куди покращуватись. Ідеально підлаштуватись під всіх неможливо. Те що подобається одному, не подобається іншому. Тому можливо тільки намагатися досягти мети: влаштувати якомога більше користувачів. Але до цього реально наближатися, адже є не тільки відмінності у тому, якому підходу віддадуть перевагу люди, але й спільні риси які подобаються багатьом.

В результаті виконання дипломної роботи було створене програмне рішення у вигляді мобільного додатку для оптимального управління часом. Вийшло дотриматися більшості критеріїв. У результаті ми отримали робочий продукт із зручним, простим та інтуїтивно зрозумілим інтерфейсом. Тестування показали, що додаток справляється зі своїми задачами, а тому у нас є гарна основа. Покращення функцій є звичайною справою у життєвому циклі додатків, тому цей продукт готовий до майбутніх змін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. K. B. Klingsieck, “Procrastination: When Good Things Don’t Come to Those Who Wait,” *European Psychologist*, vol. 18, no. 1, pp. 24–34, Jan. 2013 [Електронний ресурс] – Режим доступу до ресурсу: https://doi.org/10.1027/1016-9040/a000138
2. Google Play [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/search?q=google%20tasks&c=apps&hl=en
3. Android Kotlin First Approach [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/kotlin/first
4. MVVM Architecture in Android using Kotlin: A Practical Guide [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@jecky999/mvvm-architecture-in-android-using-kotlin-a-practical-guide-73f8de1d9c58

ДОДАТКИ

Додаток А

```
package com.pavlopodoliaka.arrowdo
 import android.Manifest
 import android.app.AlarmManager
 import android.content.Context
 import android.content.Intent
 import android.content.pm.PackageManager
 import android.os.Build
 import android.os.Bundle
 import android.provider.Settings
 import android.widget.Toast
 import androidx.activity.ComponentActivity
 import androidx.activity.compose.setContent
 import androidx.activity.enableEdgeToEdge
 import androidx.core.app.ActivityCompat
 import androidx.core.content.ContextCompat
 import androidx.core.splashscreen.SplashScreen.Companion.installSplashScreen
 import androidx.lifecycle.ViewModelProvider
 import androidx.lifecycle.lifecycleScope
 import com.pavlopodoliaka.arrowdo.data.TaskDatabase
 import com.pavlopodoliaka.arrowdo.data.TaskRepository
 import com.pavlopodoliaka.arrowdo.ui.ArrowDoApp
 import com.pavlopodoliaka.arrowdo.utils.NotificationUtils
 import com.pavlopodoliaka.arrowdo.viewmodel.TaskViewModel
 import com.pavlopodoliaka.arrowdo.viewmodel.TaskViewModelFactory
 import kotlinx.coroutines.launch
 class MainActivity : ComponentActivity() {
   private lateinit var viewModel: TaskViewModel
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      installSplashScreen()
      val database = TaskDatabase.getDatabase(applicationContext)
      val repository = TaskRepository(database.taskDao())
      val factory = TaskViewModelFactory(repository)
      viewModel = ViewModelProvider(this, factory)[TaskViewModel::class.java]
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
         if (ContextCompat.checkSelfPermission(this, Manifest.permission.
 POST_NOTIFICATIONS)
            != PackageManager.PERMISSION_GRANTED
         ) {
            ActivityCompat.requestPermissions(
               this,
               arrayOf(Manifest.permission.POST_NOTIFICATIONS),
               1001
            )
         }
      }
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
         val alarmManager = getSystemService(Context.ALARM_SERVICE) as
AlarmManager
         if (!alarmManager.canScheduleExactAlarms()) {
            Toast.makeText(
               this,
```

```
                getString(R.string.toast_exact_alarm),
                Toast.LENGTH_LONG
            ).show()
            val intent = Intent(Settings.
ACTION_REQUEST_SCHEDULE_EXACT_ALARM)
            startActivity(intent)
        }
    }
    lifecycleScope.launch {
        viewModel.getTasksForToday().collect { todayTasks ->
            NotificationUtils.scheduleTodayNotifications(this@MainActivity
, todayTasks)
        }
    }
    enableEdgeToEdge()
    setContent {
        ArrowDoApp(viewModel = viewModel)
    }
  }
}
package com.pavlopodoliaka.arrowdo.ui
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.DrawerValue
import androidx.compose.material3.Scaffold
import androidx.compose.material3.rememberDrawerState
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Modifier
import androidx.navigation.compose.currentBackStackEntryAsState
import androidx.navigation.compose.rememberNavController
import com.pavlopodoliaka.arrowdo.data.Task
import com.pavlopodoliaka.arrowdo.ui.components.AboutDialog
import com.pavlopodoliaka.arrowdo.ui.components.AddDayTaskDialog
import com.pavlopodoliaka.arrowdo.ui.components.AddScheduledTaskDialog
import com.pavlopodoliaka.arrowdo.ui.components.AddTaskDialog
import com.pavlopodoliaka.arrowdo.ui.components.AppNavHost
import com.pavlopodoliaka.arrowdo.ui.components.ArrowDoDrawer
import com.pavlopodoliaka.arrowdo.ui.components.ArrowDoFAB
import com.pavlopodoliaka.arrowdo.ui.components.ArrowDoNavigation
import com.pavlopodoliaka.arrowdo.ui.components.ArrowDoTopBar
import com.pavlopodoliaka.arrowdo.ui.components.Destination
import com.pavlopodoliaka.arrowdo.ui.theme.ArrowDoTheme
import com.pavlopodoliaka.arrowdo.viewmodel.TaskViewModel
import kotlinx.coroutines.launch
import java.time.LocalTime
@Composable
fun ArrowDoApp(
  viewModel: TaskViewModel
) {
  val startDestination = Destination.HOME
  val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
  val scope = rememberCoroutineScope()
  val navController = rememberNavController()
  val navBackStackEntry by navController.currentBackStackEntryAsState()
  val currentRoute = navBackStackEntry?.destination?.route
  val addTaskDialog = remember { mutableStateOf(false) }
  val addScheduledTaskDialog = remember { mutableStateOf(false) }
  val addDayTaskDialog = remember { mutableStateOf(false) }
  val aboutDialog = remember { mutableStateOf(false) }
```

```
    val selectedDate by viewModel.selectedDate
    ArrowDoTheme {
        ArrowDoDrawer(
            drawerState = drawerState,
            onAboutClick = {
                aboutDialog.value = true
            },
            content = {
                Scaffold(
                    topBar = {
                        ArrowDoTopBar(
                            onMenuClick = {
                                scope.launch {
                                    drawerState.apply {
                                        if (isClosed) open() else close()
                                    }
                                }
                            }
                        )
                    },
                    bottomBar = {
                        ArrowDoNavigation(
                            navController = navController
                        )
                    },
                    floatingActionButton = {
                        ArrowDoFAB(
                            currentRoute = currentRoute,
                            onFabClick = {
                                when (currentRoute) {
                                    Destination.HOME.route -> addTaskDialog.
 value = true
                                    Destination.WEEK.route ->
addScheduledTaskDialog.value =
                                        true
                                    else -> addDayTaskDialog.value = true
                                }
                            }
                        )
                    }
                ) { innerPadding ->
                    AppNavHost(
                        navController,
                        startDestination,
                        modifier = Modifier.padding(innerPadding),
                        viewModel = viewModel
                    )
                }
            }
        )
        if (addTaskDialog.value) {
            AddTaskDialog(
                modifier = Modifier,
                onDismissRequest = {
                    addTaskDialog.value = false
                },
                onConfirmRequest = { title, time, description, date, color ->
                    val newTask = Task(
                        title = title,
                        time = LocalTime.parse(time),
                        description = description,
                        color = color,
```

```
                    day = date,
                    dayType = null,
                    scheduled = false
                )
                viewModel.addTask(newTask)
                addTaskDialog.value = false
            }
        )
    }
    if (addScheduledTaskDialog.value) {
        AddScheduledTaskDialog(
            modifier = Modifier,
            onDismissRequest = {
                addScheduledTaskDialog.value = false
            },
            onConfirmRequest = { dayType, title, time, description, color
->
                val newTask = Task(
                    title = title,
                    time = LocalTime.parse(time),
                    description = description,
                    color = color,
                    dayType = dayType,
                    scheduled = true
                )
                viewModel.addTask(newTask)
                addScheduledTaskDialog.value = false
            }
        )
    }
    if (addDayTaskDialog.value && selectedDate != null) {
        AddDayTaskDialog(
            modifier = Modifier,
            onDismissRequest = {
                addDayTaskDialog.value = false
            },
            onConfirmRequest = { title, time, description, date, color ->
                val newTask = Task(
                    title = title,
                    time = LocalTime.parse(time),
                    description = description,
                    color = color,
                    day = date,
                    dayType = null,
                    scheduled = false
                )
                viewModel.addTask(newTask)
                addDayTaskDialog.value = false
            },
            defaultDate = selectedDate!!
        )
    }
    if (aboutDialog.value) {
        AboutDialog(
            onDismiss = {
                aboutDialog.value = false
            },
        )
    }
  }
 }
package com.pavlopodoliaka.arrowdo.ui.theme
```

```
import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
// Set of Material typography styles to start with
val Typography = Typography(
  bodyLarge = TextStyle(
    fontFamily = FontFamily.Default,
    fontWeight = FontWeight.Normal,
    fontSize = 16.sp,
    lineHeight = 24.sp,
    letterSpacing = 0.5.sp
  )
  /* Other default text styles to override
  titleLarge = TextStyle(
    fontFamily = FontFamily.Default,
    fontWeight = FontWeight.Normal,
    fontSize = 22.sp,
    lineHeight = 28.sp,
    letterSpacing = 0.sp
  ),
  labelSmall = TextStyle(
    fontFamily = FontFamily.Default,
    fontWeight = FontWeight.Medium,
    fontSize = 11.sp,
    lineHeight = 16.sp,
    letterSpacing = 0.5.sp
  )
  */
)
package com.pavlopodoliaka.arrowdo.ui.theme
import androidx.compose.ui.graphics.Color
//LightTheme
val primaryLight = Color(0xFF4A672D)
val onPrimaryLight = Color(0xFFFFFFFF)
val primaryContainerLight = Color(0xFFCBEEA5)
val onPrimaryContainerLight = Color(0xFF334E18)
val secondaryLight = Color(0xFF57624A)
val onSecondaryLight = Color(0xFFFFFFFF)
val secondaryContainerLight = Color(0xFFDBE7C8)
val onSecondaryContainerLight = Color(0xFF3F4A34)
val tertiaryLight = Color(0xFF386664)
val onTertiaryLight = Color(0xFFFFFFFF)
val tertiaryContainerLight = Color(0xFFBBECE9)
val onTertiaryContainerLight = Color(0xFF1E4E4C)
val errorLight = Color(0xFFBA1A1A)
val onErrorLight = Color(0xFFFFFFFF)
val errorContainerLight = Color(0xFFFFDAD6)
val onErrorContainerLight = Color(0xFF93000A)
val backgroundLight = Color(0xFFF9FAEF)
val onBackgroundLight = Color(0xFF1A1D16)
val surfaceLight = Color(0xFFF9FAEF)
val onSurfaceLight = Color(0xFF1A1D16)
val surfaceVariantLight = Color(0xFFE0E4D5)
val onSurfaceVariantLight = Color(0xFF44483E)
val outlineLight = Color(0xFF74796C)
val outlineVariantLight = Color(0xFFC4C8BA)
val scrimLight = Color(0xFF000000)
val inverseSurfaceLight = Color(0xFF2F312A)
val inverseOnSurfaceLight = Color(0xFFF0F2E7)
val inversePrimaryLight = Color(0xFFAFD18C)
```

```
 val surfaceDimLight = Color(0xFFD9DBD0)
 val surfaceBrightLight = Color(0xFFF9FAEF)
 val surfaceContainerLowestLight = Color(0xFFFFFFFF)
 val surfaceContainerLowLight = Color(0xFFF3F5E9)
 val surfaceContainerLight = Color(0xFFEDEFE4)
 val surfaceContainerHighLight = Color(0xFFE8E9DE)
 val surfaceContainerHighestLight = Color(0xFFE2E3D9)
 //DarkTheme
 val primaryDark = Color(0xFFAFD18C)
 val onPrimaryDark = Color(0xFF1D3703)
 val primaryContainerDark = Color(0xFF334E18)
 val onPrimaryContainerDark = Color(0xFFCBEEA5)
 val secondaryDark = Color(0xFFBFCBAD)
 val onSecondaryDark = Color(0xFF29341F)
 val secondaryContainerDark = Color(0xFF3F4A34)
 val onSecondaryContainerDark = Color(0xFFDBE7C8)
 val tertiaryDark = Color(0xFFA0CFCD)
 val onTertiaryDark = Color(0xFF003735)
 val tertiaryContainerDark = Color(0xFF1E4E4C)
 val onTertiaryContainerDark = Color(0xFFBBECE9)
 val errorDark = Color(0xFFFFB4AB)
 val onErrorDark = Color(0xFF690005)
 val errorContainerDark = Color(0xFF93000A)
 val onErrorContainerDark = Color(0xFFFFDAD6)
 val backgroundDark = Color(0xFF11140E)
 val onBackgroundDark = Color(0xFFE2E3D9)
 val surfaceDark = Color(0xFF11140E)
 val onSurfaceDark = Color(0xFFE2E3D9)
val surfaceVariantDark = Color(0xFF44483E)
 val onSurfaceVariantDark = Color(0xFFC4C8BA)
 val outlineDark = Color(0xFF8E9285)
 val outlineVariantDark = Color(0xFF44483E)
 val scrimDark = Color(0xFF000000)
 val inverseSurfaceDark = Color(0xFFE2E3D9)
 val inverseOnSurfaceDark = Color(0xFF2F312A)
 val inversePrimaryDark = Color(0xFF4A672D)
 val surfaceDimDark = Color(0xFF11140E)
 val surfaceBrightDark = Color(0xFF373A33)
 val surfaceContainerLowestDark = Color(0xFF0C0F09)
 val surfaceContainerLowDark = Color(0xFF1A1D16)
 val surfaceContainerDark = Color(0xFF1E211A)
 val surfaceContainerHighDark = Color(0xFF282B24)
 val surfaceContainerHighestDark = Color(0xFF33362E)
package com.pavlopodoliaka.arrowdo.ui.theme
 import android.os.Build
 import androidx.compose.foundation.isSystemInDarkTheme
 import androidx.compose.material3.MaterialTheme
 import androidx.compose.material3.darkColorScheme
 import androidx.compose.material3.dynamicDarkColorScheme
 import androidx.compose.material3.dynamicLightColorScheme
 import androidx.compose.material3.lightColorScheme
 import androidx.compose.runtime.Composable
 import androidx.compose.ui.platform.LocalContext
 private val DarkColorScheme = darkColorScheme(
    primary = primaryDark,
    onPrimary = onPrimaryDark,
    primaryContainer = primaryContainerDark,
    onPrimaryContainer = onPrimaryContainerDark,
    secondary = secondaryDark,
    onSecondary = onSecondaryDark,
    secondaryContainer = secondaryContainerDark,
    onSecondaryContainer = onSecondaryContainerDark,
```

```
    tertiary = tertiaryDark,
    onTertiary = onTertiaryDark,
    tertiaryContainer = tertiaryContainerDark,
    onTertiaryContainer = onTertiaryContainerDark,
    error = errorDark,
    onError = onErrorDark,
    errorContainer = errorContainerDark,
    onErrorContainer = onErrorContainerDark,
    background = backgroundDark,
    onBackground = onBackgroundDark,
    surface = surfaceDark,
    onSurface = onSurfaceDark,
    surfaceVariant = surfaceVariantDark,
    onSurfaceVariant = onSurfaceVariantDark,
    outline = outlineDark,
    outlineVariant = outlineVariantDark,
    scrim = scrimDark,
    inverseSurface = inverseSurfaceDark,
    inverseOnSurface = inverseOnSurfaceDark,
    inversePrimary = inversePrimaryDark,
    surfaceDim = surfaceDimDark,
    surfaceBright = surfaceBrightDark,
    surfaceContainerLowest = surfaceContainerLowestDark,
    surfaceContainerLow = surfaceContainerLowDark,
    surfaceContainer = surfaceContainerDark,
    surfaceContainerHigh = surfaceContainerHighDark,
    surfaceContainerHighest = surfaceContainerHighestDark,
)
private val LightColorScheme = lightColorScheme(
    primary = primaryLight,
    onPrimary = onPrimaryLight,
    primaryContainer = primaryContainerLight,
    onPrimaryContainer = onPrimaryContainerLight,
    secondary = secondaryLight,
    onSecondary = onSecondaryLight,
    secondaryContainer = secondaryContainerLight,
    onSecondaryContainer = onSecondaryContainerLight,
    tertiary = tertiaryLight,
    onTertiary = onTertiaryLight,
    tertiaryContainer = tertiaryContainerLight,
    onTertiaryContainer = onTertiaryContainerLight,
    error = errorLight,
    onError = onErrorLight,
    errorContainer = errorContainerLight,
    onErrorContainer = onErrorContainerLight,
    background = backgroundLight,
    onBackground = onBackgroundLight,
    surface = surfaceLight,
    onSurface = onSurfaceLight,
    surfaceVariant = surfaceVariantLight,
    onSurfaceVariant = onSurfaceVariantLight,
    outline = outlineLight,
    outlineVariant = outlineVariantLight,
    scrim = scrimLight,
    inverseSurface = inverseSurfaceLight,
    inverseOnSurface = inverseOnSurfaceLight,
    inversePrimary = inversePrimaryLight,
    surfaceDim = surfaceDimLight,
    surfaceBright = surfaceBrightLight,
    surfaceContainerLowest = surfaceContainerLowestLight,
    surfaceContainerLow = surfaceContainerLowLight,
    surfaceContainer = surfaceContainerLight,
```

```
    surfaceContainerHigh = surfaceContainerHighLight,
    surfaceContainerHighest = surfaceContainerHighestLight,
  )
  @Composable
  fun ArrowDoTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = false,
    content: @Composable () -> Unit
  ) {
    val colorScheme = when {
      dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
        val context = LocalContext.current
        if (darkTheme) dynamicDarkColorScheme(context) else
dynamicLightColorScheme(context)
      }
      darkTheme -> DarkColorScheme
      else -> LightColorScheme
    }
    MaterialTheme(
      colorScheme = colorScheme,
      typography = Typography,
      content = content
    )
  }
package com.pavlopodoliaka.arrowdo.ui.screens
  import androidx.compose.foundation.background
  import androidx.compose.foundation.layout.Arrangement
  import androidx.compose.foundation.layout.Box
  import androidx.compose.foundation.layout.Column
  import androidx.compose.foundation.layout.Row
  import androidx.compose.foundation.layout.Spacer
  import androidx.compose.foundation.layout.fillMaxHeight
  import androidx.compose.foundation.layout.fillMaxSize
  import androidx.compose.foundation.layout.fillMaxWidth
  import androidx.compose.foundation.layout.height
  import androidx.compose.foundation.layout.padding
  import androidx.compose.foundation.layout.wrapContentHeight
  import androidx.compose.foundation.layout.wrapContentSize
  import androidx.compose.foundation.lazy.LazyColumn
  import androidx.compose.foundation.lazy.items
  import androidx.compose.foundation.shape.RoundedCornerShape
  import androidx.compose.material3.HorizontalDivider
  import androidx.compose.material3.MaterialTheme
  import androidx.compose.material3.Text
  import androidx.compose.runtime.Composable
  import androidx.compose.runtime.LaunchedEffect
  import androidx.compose.runtime.collectAsState
  import androidx.compose.runtime.getValue
  import androidx.compose.runtime.mutableStateOf
  import androidx.compose.runtime.remember
  import androidx.compose.runtime.setValue
  import androidx.compose.ui.Alignment
  import androidx.compose.ui.Modifier
  import androidx.compose.ui.draw.clip
  import androidx.compose.ui.res.stringResource
  import androidx.compose.ui.text.style.TextAlign
  import androidx.compose.ui.unit.dp
  import com.pavlopodoliaka.arrowdo.R
  import com.pavlopodoliaka.arrowdo.data.Task
  import com.pavlopodoliaka.arrowdo.ui.components.ShowTaskDialog
  import com.pavlopodoliaka.arrowdo.ui.components.TaskCard
```

```
import com.pavlopodoliaka.arrowdo.viewmodel.TaskViewModel
import kotlinx.coroutines.delay
import java.time.LocalDate
import java.time.LocalTime
import java.time.format.DateTimeFormatter
import java.util.Locale
@Composable
fun HomeScreen(
    viewModel: TaskViewModel,
    modifier: Modifier = Modifier
) {
    val tasks by viewModel.getTasksForToday().collectAsState(initial =
emptyList())
    val showTaskDialog = remember { mutableStateOf(false) }
    val currentLocale = Locale.getDefault()
    val today = LocalDate.now()
    var currentTime by remember { mutableStateOf(LocalTime.now()) }
    var taskToShow by remember { mutableStateOf<Task?>(null) }
    LaunchedEffect(Unit) {
        while (true) {
            currentTime = LocalTime.now()
            delay(1000L)
        }
    }
    Box(
        modifier = modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.surfaceBright),
    ) {
        HorizontalDivider(thickness = 1.dp)
        Row(
            modifier = Modifier
                .fillMaxWidth(0.9f)
                .fillMaxHeight(0.2f)
                .align(Alignment.TopCenter),
            horizontalArrangement = Arrangement.spacedBy(30.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            Text(
                modifier = Modifier
                    .fillMaxHeight(0.8f)
                    .weight(1f)
                    .clip(RoundedCornerShape(16.dp))
                    .background(MaterialTheme.colorScheme.primary)
                    .wrapContentSize(Alignment.Center),
                text = currentTime.format(DateTimeFormatter.ofPattern("HH:mm"
)),
                style = MaterialTheme.typography.displayMedium,
                textAlign = TextAlign.Center,
                color = MaterialTheme.colorScheme.onPrimary
            )
            Column(
                modifier = Modifier
                    .fillMaxHeight(0.8f)
                    .weight(1f),
                verticalArrangement = Arrangement.spacedBy(10.dp)
            ) {
                Text(
                    modifier = Modifier
                        .fillMaxWidth()
                        .fillMaxHeight()
                        .weight(1f)
```

```
                    .clip(RoundedCornerShape(16.dp))
                    .background(MaterialTheme.colorScheme.primary)
                    .wrapContentSize(Alignment.Center),
                style = MaterialTheme.typography.headlineSmall,
                textAlign = TextAlign.Center,
                color = MaterialTheme.colorScheme.onPrimary,
                text = today.format(DateTimeFormatter.ofPattern("MMMM",
currentLocale))
            )
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .weight(1f),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.spacedBy(8.dp)
            ) {
                Text(
                    modifier = Modifier
                        .weight(1f)
                        .fillMaxHeight()
                        .clip(RoundedCornerShape(16.dp))
                        .background(MaterialTheme.colorScheme.primary)
                        .wrapContentSize(Alignment.Center),
                    style = MaterialTheme.typography.headlineSmall,
                    textAlign = TextAlign.Center,
                    color = MaterialTheme.colorScheme.onPrimary,
                    text = today.format(DateTimeFormatter.ofPattern("E",
currentLocale))
                )
                Text(
                    modifier = Modifier
                        .weight(1f)
                        .fillMaxHeight()
                        .clip(RoundedCornerShape(16.dp))
                        .background(MaterialTheme.colorScheme.primary)
                        .wrapContentSize(Alignment.Center),
                    style = MaterialTheme.typography.titleLarge,
                    textAlign = TextAlign.Center,
                    color = MaterialTheme.colorScheme.onPrimary,
                    text = today.format(DateTimeFormatter.ofPattern("dd/MM"
 , currentLocale))
                )
            }
        }
    }
    HorizontalDivider(
        modifier = Modifier
            .align(Alignment.BottomCenter), thickness = 1.dp
    )
    LazyColumn(
        modifier = modifier
            .fillMaxWidth(0.9f)
            .fillMaxHeight(0.8f)
            .align(Alignment.BottomCenter),
    ) {
        if (tasks.isEmpty()) {
            item {
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .wrapContentHeight()
                        .clip(RoundedCornerShape(16.dp))
```

```
                    .background(MaterialTheme.colorScheme.
surfaceContainer),
                    contentAlignment = Alignment.Center
                ) {
                    Text(
                        modifier = Modifier
                            .padding(16.dp),
                        textAlign = TextAlign.Center,
                        text = stringResource(R.string.home_hint),
                        style = MaterialTheme.typography.titleMedium,
                        color = MaterialTheme.colorScheme.primary
                    )
                }
            }
        } else items(tasks) { task ->
            TaskCard(
                task = task,
                onDelete = { taskToDelete ->
                    viewModel.deleteTask(taskToDelete)
                },
                onClick = { clickedTask ->
                    taskToShow = clickedTask
                    showTaskDialog.value = true
                }
            )
            Spacer(modifier = Modifier.height(4.dp))
        }
    }
  }
  if (showTaskDialog.value && taskToShow != null) {
      ShowTaskDialog(
          task = taskToShow!!,
          onDismissRequest = {
              showTaskDialog.value = false
          }
      )
  }
}
package com.pavlopodoliaka.arrowdo.ui.screens
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.res.stringResource
```

```
import androidx.compose.ui.unit.dp
import com.pavlopodoliaka.arrowdo.R
import com.pavlopodoliaka.arrowdo.data.Task
import com.pavlopodoliaka.arrowdo.ui.components.ShowTaskDialog
import com.pavlopodoliaka.arrowdo.ui.components.TaskCard
import com.pavlopodoliaka.arrowdo.viewmodel.TaskViewModel
import java.time.DayOfWeek
import java.time.format.TextStyle
import java.util.Locale
@Composable
fun WeekScreen(
  viewModel: TaskViewModel,
  modifier: Modifier = Modifier
) {
  val daysOfWeek = remember {
    listOf(
      DayOfWeek.MONDAY,
      DayOfWeek.TUESDAY,
      DayOfWeek.WEDNESDAY,
      DayOfWeek.THURSDAY,
      DayOfWeek.FRIDAY,
      DayOfWeek.SATURDAY,
      DayOfWeek.SUNDAY
    )
  }
  val showTaskDialog = remember { mutableStateOf(false) }
  val currentLocale = Locale.getDefault()
  var taskToShow by remember { mutableStateOf<Task?>(null) }
  Box(
    modifier = Modifier
      .fillMaxSize()
      .background(MaterialTheme.colorScheme.surfaceBright),
  ) {
    HorizontalDivider(thickness = 1.dp)
    HorizontalDivider(
      modifier = Modifier
        .align(Alignment.BottomCenter), thickness = 1.dp
    )
    LazyColumn(
      modifier = modifier
        .padding(vertical = 20.dp)
        .fillMaxWidth(0.9f)
        .fillMaxHeight()
        .align(Alignment.TopCenter),
    ) {
      daysOfWeek.forEach { dayOfWeek ->
        val formattedDay = dayOfWeek.getDisplayName(TextStyle.FULL,
currentLocale)
        item {
          Spacer(modifier = Modifier.height(8.dp))
          val tasks by viewModel.getTasksForDayType(dayOfWeek)
            .collectAsState(initial = emptyList())
          Text(
            text = formattedDay.toString(),
            style = MaterialTheme.typography.titleLarge,
            color = MaterialTheme.colorScheme.onSurface
          )
          Spacer(modifier = Modifier.height(8.dp))
          if (tasks.isEmpty()) {
            Box(
              modifier = Modifier
                .fillMaxWidth()
```

```
                        .height(50.dp)
                        .clip(RoundedCornerShape(16.dp))
                        .background(MaterialTheme.colorScheme.
surfaceContainer),
                    contentAlignment = Alignment.Center
                ) {
                    Text(
                        text = stringResource(R.string.week_hint),
                        style = MaterialTheme.typography.titleMedium,
                        color = MaterialTheme.colorScheme.primary
                    )
                }
                Spacer(modifier = Modifier.height(8.dp))
            } else tasks.forEach { task ->
                TaskCard(
                    task = task,
                    onDelete = { taskToDelete ->
                        viewModel.deleteTask(taskToDelete)
                    },
                    onClick = { clickedTask ->
                        taskToShow = clickedTask
                        showTaskDialog.value = true
                    }
                )
                Spacer(modifier = Modifier.height(4.dp))
            }
        }
    }
  }
  if (showTaskDialog.value && taskToShow != null) {
    ShowTaskDialog(
        task = taskToShow!!,
        onDismissRequest = {
            showTaskDialog.value = false
        }
    )
  }
}
package com.pavlopodoliaka.arrowdo.ui.screens
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.automirrored.filled.ArrowForward
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
```

```
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.navigation.NavHostController
import com.pavlopodoliaka.arrowdo.ui.components.DayCell
import com.pavlopodoliaka.arrowdo.viewmodel.CalendarViewModel
import kotlinx.coroutines.delay
import java.time.DayOfWeek
import java.time.LocalDate
import java.time.format.DateTimeFormatter
import java.time.format.TextStyle
import java.util.Locale
data class Day(
  val date: LocalDate?,
  val isToday: Boolean = false
)
@Composable
fun MonthScreen(
  modifier: Modifier = Modifier,
  viewModel: CalendarViewModel = viewModel(),
  navController: NavHostController
) {
  var animationFinished by remember { mutableStateOf(false) }
  LaunchedEffect(Unit) {
    delay(200L)
    animationFinished = true
  }
  if (!animationFinished) {
    Box(
      modifier = Modifier.fillMaxSize()
    ) {}
  } else {
    val selectedMonth by viewModel.selectedMonth.collectAsState()
    val days by viewModel.days.collectAsState()
    val currentLocale = Locale.getDefault()
    val daysOfWeek = DayOfWeek.entries
    Column(
      modifier = modifier
        .fillMaxSize()
        .background(MaterialTheme.colorScheme.surfaceBright)
    ) {
      HorizontalDivider(thickness = 1.dp)
      Row(
        modifier = Modifier
          .fillMaxWidth()
          .padding(16.dp)
          .clip(RoundedCornerShape(16.dp))
          .background(MaterialTheme.colorScheme.primary),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceBetween
      ) {
        IconButton(
          onClick = viewModel::prevMonth
        ) {
          Icon(
```

```
                imageVector = Icons.AutoMirrored.Filled.ArrowBack,
                contentDescription = "Previous Month",
                tint = MaterialTheme.colorScheme.onPrimary
            )
        }
        Text(
            text = selectedMonth.format(
                DateTimeFormatter.ofPattern(
                    "MMMM yyyy",
                    currentLocale
                )
            ),
            style = MaterialTheme.typography.titleMedium,
            color = MaterialTheme.colorScheme.onPrimary
        )
        IconButton(
            onClick = viewModel::nextMonth
        ) {
            Icon(
                imageVector = Icons.AutoMirrored.Filled.ArrowForward,
                contentDescription = "Next Month",
                tint = MaterialTheme.colorScheme.onPrimary
            )
        }
    }
    LazyColumn(
        modifier = Modifier
            .weight(1f)
            .padding(vertical = 4.dp, horizontal = 16.dp)
    ) {
        item {
            Row(
                modifier = Modifier
                    .clip(RoundedCornerShape(16.dp))
                    .background(MaterialTheme.colorScheme.primary),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                daysOfWeek.forEach { dayOfWeek ->
                    Text(
                        modifier = Modifier
                            .weight(1f)
                            .padding(4.dp),
                        text = dayOfWeek.getDisplayName(TextStyle.SHORT
, currentLocale),
                        textAlign = TextAlign.Center,
                        style = MaterialTheme.typography.titleMedium,
                        color = MaterialTheme.colorScheme.onPrimary
                    )
                }
            }
        }
        items(days.chunked(7)) { week ->
            Row(
                modifier = Modifier
                    .fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                week.forEach { day ->
                    DayCell(
                        day = day,
                        modifier = Modifier
                            .padding(4.dp)
```

```
                        .weight(1f),
                    navController = navController
                )
            }
        }
    }
    HorizontalDivider(thickness = 1.dp)
  }
 }
}
package com.pavlopodoliaka.arrowdo.ui.screens
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.wrapContentHeight
import androidx.compose.foundation.layout.wrapContentSize
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import com.pavlopodoliaka.arrowdo.R
import com.pavlopodoliaka.arrowdo.data.Task
import com.pavlopodoliaka.arrowdo.ui.components.ShowTaskDialog
import com.pavlopodoliaka.arrowdo.ui.components.TaskCard
import com.pavlopodoliaka.arrowdo.viewmodel.TaskViewModel
import java.time.LocalDate
import java.time.format.DateTimeFormatter
import java.util.Locale
@Composable
fun SingleDayScreen(
  viewModel: TaskViewModel,
  modifier: Modifier = Modifier,
  date: LocalDate
) {
  LaunchedEffect(date) {
    viewModel.setSelectedDate(date)
  }
  val currentLocale = Locale.getDefault()
  val tasks by viewModel.getTasksForDay(date).collectAsState(initial =
emptyList())
  val showTaskDialog = remember { mutableStateOf(false) }
```

```
    var taskToShow by remember { mutableStateOf<Task?>(null) }
    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.surfaceBright),
    ) {
        HorizontalDivider(thickness = 1.dp)
        HorizontalDivider(
            modifier = Modifier
                .align(Alignment.BottomCenter), thickness = 1.dp
        )
        Text(
            modifier = Modifier
                .fillMaxHeight(0.15f)
                .fillMaxWidth()
                .padding(16.dp)
                .clip(RoundedCornerShape(16.dp))
                .background(MaterialTheme.colorScheme.primary)
                .align(Alignment.TopCenter)
                .wrapContentSize(Alignment.Center),
            style = MaterialTheme.typography.headlineSmall,
            textAlign = TextAlign.Center,
            color = MaterialTheme.colorScheme.onPrimary,
            text = date.format(DateTimeFormatter.ofPattern("dd MMMM",
currentLocale))
        )
        LazyColumn(
            modifier = modifier
                .fillMaxWidth(0.9f)
                .fillMaxHeight(0.8f)
                .align(Alignment.BottomCenter),
        ) {
            if (tasks.isEmpty()) {
                item {
                    Box(
                        modifier = Modifier
                            .fillMaxWidth()
                            .wrapContentHeight()
                            .clip(RoundedCornerShape(16.dp))
                            .background(MaterialTheme.colorScheme.
 surfaceContainer),
                        contentAlignment = Alignment.Center
                    ) {
                        Text(
                            modifier = Modifier
                                .padding(16.dp),
                            text = stringResource(R.string.home_hint),
                            style = MaterialTheme.typography.titleMedium,
                            color = MaterialTheme.colorScheme.primary
                        )
                    }
                }
            } else items(tasks) { task ->
                TaskCard(
                    task = task,
                    onDelete = { taskToDelete ->
                        viewModel.deleteTask(taskToDelete)
                    },
                    onClick = { clickedTask ->
                        taskToShow = clickedTask
                        showTaskDialog.value = true
                    }
```

```
            )
            Spacer(modifier = Modifier.height(4.dp))
        }
    }
  }
  if (showTaskDialog.value && taskToShow != null) {
    ShowTaskDialog(
      task = taskToShow!!,
}
)
onDismissRequest = {
showTaskDialog.value = false
}
}
package com.pavlopodoliaka.arrowdo.ui.components
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.navigation.NavHostController
import androidx.navigation.compose.currentBackStackEntryAsState
import com.pavlopodoliaka.arrowdo.ui.screens.Day
@Composable
fun DayCell(
  modifier: Modifier = Modifier,
  day: Day,
  navController: NavHostController
) {
  val navBackStackEntry by navController.currentBackStackEntryAsState()
  val currentRoute = navBackStackEntry?.destination?.route
  Box(
    modifier = modifier
      .aspectRatio(1f)
      .clip(shape = CircleShape)
      .background(
        when {
          day.isToday -> MaterialTheme.colorScheme.tertiary
          day.date != null -> MaterialTheme.colorScheme.primary
          else -> Color.Transparent
        }
      )
      .clickable {
        day.date?.let { date ->
          val targetRoute = "day/$date"
          if (currentRoute != targetRoute) {
            navController.navigate(targetRoute) {
              popUpTo(navController.graph.startDestinationId) {
                saveState = true
              }
              launchSingleTop = true
            }
          }
        }
```

```
            }
        ) {
            Text(
                modifier = Modifier
                    .align(Alignment.Center),
                text = day.date?.dayOfMonth?.toString() ?: "",
                style = MaterialTheme.typography.titleMedium,
                color = MaterialTheme.colorScheme.onPrimary
            )
        }
    }
}
package com.pavlopodoliaka.arrowdo.ui.components
import android.icu.util.Calendar
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowDropDown
import androidx.compose.material.icons.filled.Close
import androidx.compose.material3.BasicAlertDialog
import androidx.compose.material3.Button
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.ExposedDropdownMenuBox
import androidx.compose.material3.ExposedDropdownMenuDefaults
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.MenuAnchorType
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TimePicker
import androidx.compose.material3.TimePickerState
import androidx.compose.material3.rememberTimePickerState
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.toArgb
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextAlign
```

```
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.Dialog
import androidx.compose.ui.window.DialogProperties
import com.pavlopodoliaka.arrowdo.R
import com.pavlopodoliaka.arrowdo.data.Task
import java.time.DayOfWeek
import java.time.LocalDate
import java.time.format.DateTimeFormatter
import java.time.format.TextStyle
import java.util.Locale
//MAIN SCREEN DIALOG
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddTaskDialog(
    modifier: Modifier = Modifier,
    onDismissRequest: () -> Unit,
    onConfirmRequest: (
        title: String,
        time: String,
        description: String,
        date: LocalDate,
        color: Int
    ) -> Unit
) {
    var title by remember { mutableStateOf("") }
    var time by remember { mutableStateOf("") }
    var description by remember { mutableStateOf("") }
    val date by remember { mutableStateOf(LocalDate.now()) }
    var color by remember { mutableIntStateOf(Color.Transparent.toArgb()) }
    var showTimePicker by remember { mutableStateOf(false) }
    val saveButtonClicked = remember { mutableStateOf(false) }
    Dialog(
        onDismissRequest = { onDismissRequest() },
        properties = DialogProperties(
            usePlatformDefaultWidth = false,
            dismissOnBackPress = true,
        ),
    ) {
        Box(
            modifier = modifier
                .fillMaxSize()
                .background(MaterialTheme.colorScheme.surfaceVariant),
            contentAlignment = Alignment.TopCenter
        ) {
            LazyColumn(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.SpaceAround
            ) {
                item {
                    Row(
                        modifier = Modifier
                            .fillMaxWidth(),
                        horizontalArrangement = Arrangement.SpaceBetween
                    ) {
                        TextButton(onClick = { onDismissRequest() }) {
                            Text(stringResource(R.string.cancel_button))
                        }
                        TextButton(onClick = {
                            saveButtonClicked.value = false
```

```
                    if (title.isNotBlank() && time.isNotBlank()) {
                        onConfirmRequest(
                            title,
                            time,
                            description,
                            date,
                            color
                        )
                    } else {
                        saveButtonClicked.value = true
                    }
                }) {
                    Text(stringResource(R.string.save_button))
                }
            }
            Spacer(modifier = Modifier.height(8.dp))
            Column(
                verticalArrangement = Arrangement.SpaceBetween
            ) {
                OutlinedTextField(
                    value = title,
                    onValueChange = {
                        if (it.length <= 30) title = it
                    },
                    isError = title.isBlank() && saveButtonClicked.
value,
                    supportingText = {
                        if (title.isBlank() && saveButtonClicked.value)
                            Text(stringResource(R.string.title_warning
))
                        else Text(
                            "${title.length}/30"
                        )
                    },
                    label = { Text(stringResource(R.string.task_title
)) },
                    singleLine = true,
                    colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                    modifier = Modifier
                        .fillMaxWidth()
                )
                Spacer(modifier = Modifier.height(8.dp))
                OutlinedTextField(
                    value = time,
                    onValueChange = { },
                    isError = time.isBlank() && saveButtonClicked.value
,
                    supportingText = {
                        if (time.isBlank() && saveButtonClicked.value)
                            Text(stringResource(R.string.time_warning))
                    },
                    label = { Text(stringResource(R.string.time)) },
                    placeholder = { Text("00:00") },
                    singleLine = true,
                    readOnly = true,
                    trailingIcon = {
                        IconButton(
                            onClick = { showTimePicker = true }
                        ) {
                            Icon(
                                Icons.Default.ArrowDropDown,
```

```
                            contentDescription = "trailing icon"
                        )
                    }
                },
                colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                modifier = Modifier
                    .fillMaxWidth(0.35f)
            )
            if (showTimePicker) {
                TimePickerDialogCombined(
                    onConfirm = { timePickerState ->
                        val hour = timePickerState.hour.toString().
padStart(2, '0')
                        val minute =
                            timePickerState.minute.toString().
padStart(2, '0')
                        time = "$hour:$minute"
                        showTimePicker = false
                    },
                    onDismiss = { showTimePicker = false }
                )
            }
        }
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = description,
            onValueChange = {
                if (it.length <= 500) description = it
            },
            supportingText = { Text("${description.length}/500") },
            label = { Text(stringResource(R.string.description)) },
            colors = ExposedDropdownMenuDefaults.textFieldColors(),
            modifier = Modifier
                .fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(8.dp))
        ColorPicker(
            selectedColor = color,
            onSelected = { color = it }
        )
    }
  }
 }
}
//WEEK SCREEN DIALOG
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddScheduledTaskDialog(
  modifier: Modifier = Modifier,
  onDismissRequest: () -> Unit,
  onConfirmRequest: (
    dayType: DayOfWeek?,
    title: String,
    time: String,
    description: String,
    color: Int
  ) -> Unit
) {
  var dayType by remember { mutableStateOf<DayOfWeek?>(null) }
  var title by remember { mutableStateOf("") }
```

```
    var time by remember { mutableStateOf("") }
    var description by remember { mutableStateOf("") }
    var color by remember { mutableIntStateOf(Color.Transparent.toArgb()) }
    var showTimePicker by remember { mutableStateOf(false) }
    val saveButtonClicked = remember { mutableStateOf(false) }
    Dialog(
        onDismissRequest = { onDismissRequest() },
        properties = DialogProperties(
            usePlatformDefaultWidth = false,
            dismissOnBackPress = true,
        ),
    ) {
        Box(
            modifier = modifier
                .fillMaxSize()
                .background(MaterialTheme.colorScheme.surfaceVariant),
            contentAlignment = Alignment.TopCenter
        ) {
            LazyColumn(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.SpaceAround
            ) {
                item {
                    Row(
                        modifier = Modifier
                            .fillMaxWidth(),
                        horizontalArrangement = Arrangement.SpaceBetween
                    ) {
                        TextButton(onClick = { onDismissRequest() }) {
                            Text(stringResource(R.string.cancel_button))
                        }
                        TextButton(onClick = {
                            saveButtonClicked.value = false
                            if (title.isNotBlank() && time.isNotBlank() &&
dayType != null) {
                                onConfirmRequest(
                                    dayType,
                                    title,
                                    time,
                                    description,
                                    color
                                )
                            } else {
                                saveButtonClicked.value = true
                            }
                        }) {
                            Text(stringResource(R.string.save_button))
                        }
                    }
                    Spacer(modifier = Modifier.height(8.dp))
                    Column(
                        verticalArrangement = Arrangement.SpaceBetween
                    ) {
                        OutlinedTextField(
                            value = title,
                            onValueChange = {
                                if (it.length <= 30) title = it
                            },
                            isError = title.isBlank() && saveButtonClicked.
```

```
value,
                    supportingText = {
                        if (title.isBlank() && saveButtonClicked.value)
                            Text(stringResource(R.string.title_warning
))
                        else Text(
                            "${title.length}/30"
                        )
                    },
                    label = { Text(stringResource(R.string.task_title
)) },
                    singleLine = true,
                    colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                    modifier = Modifier
                        .fillMaxWidth()
                )
                Spacer(modifier = Modifier.height(8.dp))
                Row(
                    modifier = Modifier
                        .fillMaxWidth(),
                    horizontalArrangement = Arrangement.SpaceBetween
                ) {
                    OutlinedTextField(
                        value = time,
                        onValueChange = { },
                        isError = time.isBlank() && saveButtonClicked.
value,
                        supportingText = {
                            if (time.isBlank() && saveButtonClicked.
value)
                                Text(stringResource(R.string.
time_warning))
                        },
                        label = { Text(stringResource(R.string.time
)) },
                        placeholder = { Text("00:00") },
                        singleLine = true,
                        readOnly = true,
                        trailingIcon = {
                            IconButton(
                                onClick = { showTimePicker = true }
                            ) {
                                Icon(
                                    Icons.Default.ArrowDropDown,
                                    contentDescription = "trailing icon
"
                                )
                            }
                        },
                        colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                        modifier = Modifier
                            .fillMaxWidth(0.35f)
                    )
                    DayTypeDropdownMenu(
                        onDayTypeSelected = { selectedDay ->
                            dayType = selectedDay
                        },
                        isError = dayType == null,
                        saveButtonClicked = saveButtonClicked.value
                    )
```

```
                }
                if (showTimePicker) {
                    TimePickerDialogCombined(
                        onConfirm = { timePickerState ->
                            val hour = timePickerState.hour.toString().
padStart(2, '0')
                            val minute =
                                timePickerState.minute.toString().
padStart(2, '0')
                            time = "$hour:$minute"
                            showTimePicker = false
                        },
                        onDismiss = { showTimePicker = false }
                    )
                }
            }
            Spacer(modifier = Modifier.height(8.dp))
            OutlinedTextField(
                value = description,
                onValueChange = {
                    if (it.length <= 500) description = it
                },
                supportingText = { Text("${description.length}/500") },
                label = { Text(stringResource(R.string.description)) },
                colors = ExposedDropdownMenuDefaults.textFieldColors(),
                modifier = Modifier
                    .fillMaxWidth()
            )
            Spacer(modifier = Modifier.height(8.dp))
            ColorPicker(
                selectedColor = color,
                onSelected = { color = it }
            )
        }
    }
  }
}
//SINGLE DAY SCREEN DIALOG
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddDayTaskDialog(
  modifier: Modifier = Modifier,
  defaultDate: LocalDate,
  onDismissRequest: () -> Unit,
  onConfirmRequest: (
    title: String,
    time: String,
    description: String,
    date: LocalDate,
    color: Int
  ) -> Unit
) {
  var title by remember { mutableStateOf("") }
  var time by remember { mutableStateOf("") }
  var description by remember { mutableStateOf("") }
  val date by remember { mutableStateOf(defaultDate) }
  var color by remember { mutableIntStateOf(Color.Transparent.toArgb()) }
  var showTimePicker by remember { mutableStateOf(false) }
  val saveButtonClicked = remember { mutableStateOf(false) }
  Dialog(
    onDismissRequest = { onDismissRequest() },
```

```
        properties = DialogProperties(
            usePlatformDefaultWidth = false,
            dismissOnBackPress = true,
        ),
    ) {
        Box(
            modifier = modifier
                .fillMaxSize()
                .background(MaterialTheme.colorScheme.surfaceVariant),
            contentAlignment = Alignment.TopCenter
        ) {
            LazyColumn(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.SpaceAround
            ) {
                item {
                    Row(
                        modifier = Modifier
                            .fillMaxWidth(),
                        horizontalArrangement = Arrangement.SpaceBetween
                    ) {
                        TextButton(onClick = { onDismissRequest() }) {
                            Text(stringResource(R.string.cancel_button))
                        }
                        TextButton(onClick = {
                            saveButtonClicked.value = false
                            if (title.isNotBlank() && time.isNotBlank()) {
                                onConfirmRequest(
                                    title,
                                    time,
                                    description,
                                    date,
                                    color
                                )
                            } else {
                                saveButtonClicked.value = true
                            }
                        }) {
                            Text(stringResource(R.string.save_button))
                        }
                    }
                    Spacer(modifier = Modifier.height(8.dp))
                    Column(
                        verticalArrangement = Arrangement.SpaceBetween
                    ) {
                        OutlinedTextField(
                            value = title,
                            onValueChange = {
                                if (it.length <= 30) title = it
                            },
                            isError = title.isBlank() && saveButtonClicked.
value,
                            supportingText = {
                                if (title.isBlank() && saveButtonClicked.value)
                                    Text(stringResource(R.string.title_warning
))
                                else Text(
                                    "${title.length}/30"
                                )
```

```
                    },
                    label = { Text(stringResource(R.string.task_title
)) },
                    singleLine = true,
                    colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                    modifier = Modifier
                        .fillMaxWidth()
                )
                Spacer(modifier = Modifier.height(8.dp))
                OutlinedTextField(
                    value = time,
                    onValueChange = { },
                    isError = time.isBlank() && saveButtonClicked.value
,
                    supportingText = {
                        if (time.isBlank() && saveButtonClicked.value)
                            Text(stringResource(R.string.time_warning))
                    },
                    label = { Text(stringResource(R.string.time)) },
                    placeholder = { Text("00:00") },
                    singleLine = true,
                    readOnly = true,
                    trailingIcon = {
                        IconButton(
                            onClick = { showTimePicker = true }
                        ) {
                            Icon(
                                Icons.Default.ArrowDropDown,
                                contentDescription = "trailing icon"
                            )
                        }
                    },
                    colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                    modifier = Modifier
                        .fillMaxWidth(0.35f)
                )
                if (showTimePicker) {
                    TimePickerDialogCombined(
                        onConfirm = { timePickerState ->
                            val hour = timePickerState.hour.toString().
padStart(2, '0')
                            val minute =
                                timePickerState.minute.toString().
padStart(2, '0')
                            time = "$hour:$minute"
                            showTimePicker = false
                        },
                        onDismiss = { showTimePicker = false }
                    )
                }
            }
            Spacer(modifier = Modifier.height(8.dp))
            OutlinedTextField(
                value = description,
                onValueChange = {
                    if (it.length <= 500) description = it
                },
                supportingText = { Text("${description.length}/500") },
                label = { Text(stringResource(R.string.description)) },
                colors = ExposedDropdownMenuDefaults.textFieldColors(),
```

```
                modifier = Modifier
                    .fillMaxWidth()
            )
            Spacer(modifier = Modifier.height(8.dp))
            ColorPicker(
                selectedColor = color,
                onSelected = { color = it }
            )
        }
    }
  }
}
//TIME PICKER
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TimePickerDialogCombined(
  onConfirm: (TimePickerState) -> Unit,
  onDismiss: () -> Unit
) {
  val currentTime = Calendar.getInstance()
  val timePickerState = rememberTimePickerState(
    initialHour = currentTime.get(Calendar.HOUR_OF_DAY),
    initialMinute = currentTime.get(Calendar.MINUTE),
    is24Hour = true,
  )
  BasicAlertDialog(
    onDismissRequest = onDismiss
  ) {
    Surface(
      shape = RoundedCornerShape(16.dp),
      tonalElevation = 8.dp,
      color = MaterialTheme.colorScheme.surface,
      modifier = Modifier
        .fillMaxWidth()
        .padding(16.dp)
    ) {
      Column(
        modifier = Modifier
          .padding(16.dp),
      ) {
        TimePicker(
          state = timePickerState,
        )
        Row(
          modifier = Modifier
            .fillMaxWidth(),
          horizontalArrangement = Arrangement.SpaceEvenly
        ) {
          Button(onClick = onDismiss) {
            Text(stringResource(R.string.cancel_button))
          }
          Button(onClick = { onConfirm(timePickerState) }) {
            Text(stringResource(R.string.ok))
          }
        }
      }
    }
  }
}
//COLOR PICKER
@Composable
```

```
fun ColorPicker(
  selectedColor: Int?,
  onSelected: (Int) -> Unit,
  colors: List<Int> = listOf(
    Color(0xFF00BF63).toArgb(),
    Color(0xFFFF9800).toArgb(),
    Color(0xFFFF5757).toArgb()
  )
) {
  Box(
    modifier = Modifier
      .fillMaxWidth()
      .height(100.dp)
      .border(
        1.dp,
        MaterialTheme.colorScheme.onSurfaceVariant,
        shape = RoundedCornerShape(4.dp)
      ),
  ) {
    Text(
      modifier = Modifier
        .fillMaxWidth()
        .padding(8.dp)
        .align(Alignment.TopCenter),
      textAlign = TextAlign.Center,
      style = MaterialTheme.typography.bodyLarge,
      text = stringResource(R.string.color_choice),
      color = MaterialTheme.colorScheme.onSurfaceVariant
    )
    Spacer(modifier = Modifier.height(16.dp))
    Row(
      modifier = Modifier
        .fillMaxWidth()
        .padding(24.dp)
        .align(Alignment.BottomCenter),
      horizontalArrangement = Arrangement.SpaceAround
    ) {
      colors.forEach { colorS ->
        val isSelected = selectedColor != null && colorS ==
selectedColor
        Box(
          modifier = Modifier
            .size(30.dp)
            .clip(CircleShape)
            .background(Color(colorS))
            .border(
              width = if (isSelected) 4.dp else 0.dp,
              color = if (isSelected) MaterialTheme.colorScheme.
onSurface else Color.Transparent,
              shape = CircleShape
            )
            .clickable { onSelected(colorS) }
        )
      }
    }
  }
}
//DAY PICKER
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DayTypeDropdownMenu(
  onDayTypeSelected: (DayOfWeek) -> Unit,
```

```
    isError: Boolean,
    saveButtonClicked: Boolean
) {
    val currentLocale = Locale.getDefault()
    var expanded by remember { mutableStateOf(false) }
    val daysOfWeek = DayOfWeek.entries
    var dayType by remember { mutableStateOf("") }
    ExposedDropdownMenuBox(
        modifier = Modifier
            .fillMaxWidth(0.9f),
        expanded = expanded,
        onExpandedChange = { expanded = it }
    ) {
        OutlinedTextField(
            value = dayType,
            onValueChange = {},
            modifier = Modifier.menuAnchor(MenuAnchorType.PrimaryNotEditable,
true),
            readOnly = true,
            singleLine = true,
            label = { Text(stringResource(R.string.day_type_label)) },
            trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded
 = expanded) },
            colors = ExposedDropdownMenuDefaults.textFieldColors(),
            isError = isError && saveButtonClicked,
            supportingText = {
                if (isError && saveButtonClicked) {
                    Text(stringResource(R.string.day_warning))
                }
            }
        )
        ExposedDropdownMenu(
            expanded = expanded,
            onDismissRequest = { expanded = false },
        ) {
            daysOfWeek.forEach { dayOfWeek ->
                val formattedDay = dayOfWeek.getDisplayName(TextStyle.FULL,
currentLocale)
                DropdownMenuItem(
                    text = {
                        Text(
                            formattedDay,
                            style = MaterialTheme.typography.bodyLarge
                        )
                    },
                    onClick = {
                        dayType = formattedDay
                        onDayTypeSelected(dayOfWeek)
                        expanded = false
                    },
                    contentPadding = ExposedDropdownMenuDefaults.
 ItemContentPadding,
                )
            }
        }
    }
}
//VIEW TASK DIALOG
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ShowTaskDialog(
    task: Task,
```

```
    modifier: Modifier = Modifier,
    onDismissRequest: () -> Unit
) {
    val title = task.title
    val time = task.time
    val dayType = task.dayType
    val description = task.description
    val day = task.day
    val currentLocale = Locale.getDefault()
    Dialog(
        onDismissRequest = { onDismissRequest() },
        properties = DialogProperties(
            usePlatformDefaultWidth = false,
            dismissOnBackPress = true,
        ),
    ) {
        Box(
            modifier = modifier
                .fillMaxSize()
                .background(MaterialTheme.colorScheme.surfaceVariant),
            contentAlignment = Alignment.TopCenter
        ) {
            LazyColumn(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.SpaceAround
            ) {
                item {
                    IconButton(
                        onClick = { onDismissRequest() }
                    ) {
                        Icon(
                            Icons.Default.Close,
                            contentDescription = "Close",
                        )
                    }
                    Spacer(modifier = Modifier.height(8.dp))
                    Column(
                        verticalArrangement = Arrangement.SpaceBetween
                    ) {
                        OutlinedTextField(
                            value = title,
                            onValueChange = {},
                            readOnly = true,
                            label = { Text(stringResource(R.string.task_title
)) },
                            singleLine = true,
                            colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                            modifier = Modifier
                                .fillMaxWidth()
                        )
                        Spacer(modifier = Modifier.height(8.dp))
                        Row(
                            modifier = Modifier
                                .fillMaxWidth(),
                            horizontalArrangement = Arrangement.SpaceBetween
                        ) {
                            OutlinedTextField(
                                value = time.toString(),
```

```
                        onValueChange = {},
                        singleLine = true,
                        readOnly = true,
                        label = { Text(stringResource(R.string.time
)) },
                        colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                        modifier = Modifier
                            .fillMaxWidth(0.35f)
                    )
                    if (dayType != null) {
                        OutlinedTextField(
                            value = dayType.getDisplayName(TextStyle.
FULL, currentLocale),
                            onValueChange = {},
                            readOnly = true,
                            singleLine = true,
                            label = { Text(stringResource(R.string.
day_type_label)) },
                            colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                            modifier = Modifier
                                .fillMaxWidth(0.9f)
                        )
                    }
                }
            }
            Spacer(modifier = Modifier.height(8.dp))
            OutlinedTextField(
                value = description,
                onValueChange = {},
                readOnly = true,
                label = { Text(stringResource(R.string.description)) },
                colors = ExposedDropdownMenuDefaults.textFieldColors(),
                modifier = Modifier
                    .fillMaxWidth()
            )
            if (day != null) {
                Spacer(modifier = Modifier.height(8.dp))
                OutlinedTextField(
                    value = day.format(DateTimeFormatter.ofPattern("dd.
MM.yyyy")),
                    onValueChange = {},
                    readOnly = true,
                    label = { Text(stringResource(R.string.date)) },
                    colors = ExposedDropdownMenuDefaults.
textFieldColors(),
                    modifier = Modifier
                        .fillMaxWidth()
                )
            }
        }
    }
  }
}
//ABOUT DIALOG
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AboutDialog(
  onDismiss: () -> Unit
) {
```

```
    BasicAlertDialog(
        onDismissRequest = onDismiss
    ) {
        Surface(
            shape = RoundedCornerShape(16.dp),
            tonalElevation = 8.dp,
            color = MaterialTheme.colorScheme.surface,
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp)
                .size(150.dp),
        ) {
            Box(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(16.dp)
            ) {
                IconButton(
                    modifier = Modifier
                        .align(Alignment.TopStart),
                    onClick = { onDismiss() }
                ) {
                    Icon(
                        Icons.Default.Close,
                        contentDescription = "Close",
                        tint = MaterialTheme.colorScheme.onSurface
                    )
                }
                Column(
                    modifier = Modifier
                        .align(Alignment.Center),
                ) {
                    Text(
                        modifier = Modifier
                            .padding(8.dp),
                        text = stringResource(R.string.arrow_do),
                        style = MaterialTheme.typography.titleLarge,
                        color = MaterialTheme.colorScheme.onSurface
                    )
                    Text(
                        modifier = Modifier
                            .padding(8.dp),
                        text = "version: 1.0.1",
                        style = MaterialTheme.typography.titleMedium,
                        color = MaterialTheme.colorScheme.onSurface
                    )
                }
            }
        }
    }
}
package com.pavlopodoliaka.arrowdo.ui.components
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
```

```
import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import com.pavlopodoliaka.arrowdo.data.Task
@Composable
fun TaskCard(
  task: Task,
  onClick: (Task) -> Unit = {},
  onDelete: (Task) -> Unit = {}
) {
  Card(
    modifier = Modifier
      .fillMaxWidth()
      .height(60.dp)
      .clickable { onClick(task) },
    elevation = CardDefaults.cardElevation(defaultElevation = 6.dp),
    colors = CardDefaults.cardColors(containerColor = MaterialTheme.
colorScheme.inversePrimary)
  ) {
    Column(
      modifier = Modifier
        .fillMaxSize(),
      verticalArrangement = Arrangement.Center
    ) {
      Row (
        modifier = Modifier
          .fillMaxHeight(),
        verticalAlignment = Alignment.CenterVertically
      ){
        Box(
          modifier = Modifier
            .fillMaxHeight()
            .weight(1f)
            .background(Color(task.color)),
        ) {
          Text(
            modifier = Modifier
              .align(Alignment.Center),
            text = "${task.time}",
            textAlign = TextAlign.Center,
            style = MaterialTheme.typography.titleMedium,
            color = MaterialTheme.colorScheme.onSurface
          )
        }
        Text(
          modifier = Modifier
            .weight(4f)
            .padding(horizontal = 4.dp),
          text = task.title,
```

```
                textAlign = TextAlign.Start,
                overflow = TextOverflow.Ellipsis,
                style = MaterialTheme.typography.titleMedium,
                color = MaterialTheme.colorScheme.onSurface
            )
            IconButton(
                modifier = Modifier
                    .weight(1f),
                onClick = {
                    onDelete(task)
                },
            ) {
                Icon(
                    imageVector = Icons.Default.Delete,
                    contentDescription = "DeleteTaskCard",
                    tint = MaterialTheme.colorScheme.onSurface
                )
            }
        }
    }
}
package com.pavlopodoliaka.arrowdo.ui.components
import androidx.annotation.StringRes
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.compose.animation.slideInHorizontally
import androidx.compose.animation.slideOutHorizontally
import androidx.compose.foundation.background
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.navigation.NavHostController
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.navArgument
import com.pavlopodoliaka.arrowdo.R
import com.pavlopodoliaka.arrowdo.ui.screens.HomeScreen
import com.pavlopodoliaka.arrowdo.ui.screens.MonthScreen
import com.pavlopodoliaka.arrowdo.ui.screens.SingleDayScreen
import com.pavlopodoliaka.arrowdo.ui.screens.WeekScreen
import com.pavlopodoliaka.arrowdo.viewmodel.TaskViewModel
import java.time.LocalDate
enum class Destination(
    val route: String,
    @StringRes val labelId: Int,
    val iconId: Int,
    val contentDescription: String,
) {
    WEEK("week", R.string.left_screen_label, R.drawable.list_alt, "Week
Schedule"),
    HOME("home", R.string.main_screen_label, R.drawable.home, "Today Tasks"),
    MONTH("month", R.string.right_screen_label, R.drawable.calendar_month, "
Month"),
    DAY("day/{date}", R.string.single_day_label, R.drawable.view_day, "Tasks of
Single Day")
}
@Composable
fun AppNavHost(
    navController: NavHostController,
    startDestination: Destination,
```

```
    viewModel: TaskViewModel,
    modifier: Modifier = Modifier,
) {
    NavHost(
        navController,
        startDestination = startDestination.route,
        modifier = modifier
            .background(MaterialTheme.colorScheme.surfaceBright)
    ) {
        Destination.entries
            .filter { it != Destination.DAY }
            .forEach { destination ->
                composable(
                    route = destination.route,
                    enterTransition = {
                        val initialRoute = initialState.destination.route
                        val targetRoute = targetState.destination.route
                        when {
                            initialRoute == Destination.HOME.route &&
                                    targetRoute == Destination.MONTH.route ||
                                    initialRoute == Destination.WEEK.route &&
                                    targetRoute == Destination.MONTH.route
                                -> slideInHorizontally { it } + fadeIn()
                            initialRoute == Destination.HOME.route &&
                                    targetRoute == Destination.WEEK.route ||
                                    initialRoute == Destination.MONTH.route &&
                                    targetRoute == Destination.WEEK.route
                                -> slideInHorizontally { -it } + fadeIn()
                            else -> null
                        }
                    },
                    exitTransition = {
                        val initialRoute = initialState.destination.route
                        val targetRoute = targetState.destination.route
                        when {
                            initialRoute == Destination.HOME.route &&
                                    targetRoute == Destination.MONTH.route ||
                                    initialRoute == Destination.WEEK.route &&
                                    targetRoute == Destination.MONTH.route
                                -> slideOutHorizontally { -it } + fadeOut()
                            initialRoute == Destination.HOME.route &&
                                    targetRoute == Destination.WEEK.route ||
                                    initialRoute == Destination.MONTH.route &&
                                    targetRoute == Destination.WEEK.route
                                -> slideOutHorizontally { it } + fadeOut()
                            else -> null
                        }
                    },
                    popEnterTransition = {
                        val initialRoute = initialState.destination.route
                        val targetRoute = targetState.destination.route
                        when {
                            initialRoute == Destination.MONTH.route &&
                                    targetRoute == Destination.HOME.route ||
                                    initialRoute == Destination.MONTH.route &&
                                    targetRoute == Destination.WEEK.route
                                -> slideInHorizontally { -it } + fadeIn()
                            initialRoute == Destination.WEEK.route &&
                                    targetRoute == Destination.HOME.route ||
                                    initialRoute == Destination.WEEK.route &&
                                    targetRoute == Destination.MONTH.route
                                -> slideInHorizontally { it } + fadeIn()
```

```
                    else -> null
                }
            },
            popExitTransition = {
                val initialRoute = initialState.destination.route
                val targetRoute = targetState.destination.route
                when {
                    initialRoute == Destination.MONTH.route &&
                            targetRoute == Destination.HOME.route ||
                            initialRoute == Destination.MONTH.route &&
                            targetRoute == Destination.WEEK.route
                        -> slideOutHorizontally { it } + fadeOut()
                    initialRoute == Destination.WEEK.route &&
                            targetRoute == Destination.HOME.route ||
                            initialRoute == Destination.WEEK.route &&
                            targetRoute == Destination.MONTH.route
                        -> slideOutHorizontally { -it } + fadeOut()
                    else -> null
                }
            }
        ) {
            when (destination) {
                Destination.WEEK -> WeekScreen(viewModel)
                Destination.HOME -> HomeScreen(viewModel)
                Destination.MONTH -> MonthScreen(navController =
navController)
                Destination.DAY -> {}
            }
        }
    }
    composable(
        route = "day/{date}",
        arguments = listOf(navArgument("date") { type = NavType.StringType
 })
    ) { backStackEntry ->
        val dateString = backStackEntry.arguments?.getString("date")
        val date = dateString?.let { LocalDate.parse(it) }
        if (date != null) {
            SingleDayScreen(viewModel = viewModel, date = date)
        }
    }
  }
 }
package com.pavlopodoliaka.arrowdo.ui.components
 import androidx.compose.material.icons.Icons
 import androidx.compose.material.icons.filled.Add
 import androidx.compose.material3.FloatingActionButton
 import androidx.compose.material3.Icon
 import androidx.compose.material3.MaterialTheme
 import androidx.compose.runtime.Composable
 @Composable
 fun ArrowDoFAB(
   currentRoute: String?,
   onFabClick: () -> Unit
 ) {
   if (
       currentRoute == Destination.HOME.route ||
       currentRoute == Destination.WEEK.route ||
       currentRoute == Destination.DAY.route
   ) {
       FloatingActionButton(
           onClick = onFabClick,
```

```
            containerColor = MaterialTheme.colorScheme.primaryContainer,
            contentColor = MaterialTheme.colorScheme.onPrimaryContainer
        ) {
            Icon(Icons.Default.Add, contentDescription = "Add")
        }
    }
}
package com.pavlopodoliaka.arrowdo.ui.components
import androidx.compose.foundation.layout.width
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Close
import androidx.compose.material.icons.filled.Info
import androidx.compose.material3.DrawerState
import androidx.compose.material3.HorizontalDivider
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.ModalDrawerSheet
import androidx.compose.material3.ModalNavigationDrawer
import androidx.compose.material3.NavigationDrawerItem
import androidx.compose.material3.NavigationDrawerItemDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import com.pavlopodoliaka.arrowdo.R
import kotlinx.coroutines.launch
@Composable
fun ArrowDoDrawer(
    drawerState: DrawerState,
    content: @Composable () -> Unit,
    onAboutClick: () -> Unit
) {
    val scope = rememberCoroutineScope()
    ModalNavigationDrawer(
        drawerState = drawerState,
        drawerContent = {
            ModalDrawerSheet(
                drawerContainerColor = MaterialTheme.colorScheme.surfaceBright,
                drawerContentColor = MaterialTheme.colorScheme.onSurface,
                modifier = Modifier.width(320.dp),
                ) {
                IconButton(
                    onClick = {
                        scope.launch {
                            drawerState.close()
                        }
                    }
                ) {
                    Icon(
                        Icons.Default.Close,
                        contentDescription = "Close",
                    )
                }
                HorizontalDivider()
                NavigationDrawerItem(
                    icon = { Icon(Icons.Default.Info, contentDescription = "
About") },
                    label = { Text(text = stringResource(R.string.about)) },
                    selected = false,
```

```
                onClick = {
                    scope.launch {
                        drawerState.close()
                    }
                    onAboutClick()
                },
                colors = NavigationDrawerItemDefaults.colors(
                    unselectedIconColor = MaterialTheme.colorScheme.
onSurface,
                    unselectedTextColor = MaterialTheme.colorScheme.
onSurface,
                )
            )
        }
    },
    gesturesEnabled = true,
    content = content
  )
}
package com.pavlopodoliaka.arrowdo.ui.components
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.AccountCircle
import androidx.compose.material.icons.filled.Menu
import androidx.compose.material3.CenterAlignedTopAppBar
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextOverflow
import com.pavlopodoliaka.arrowdo.R
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ArrowDoTopBar(
  onMenuClick: () -> Unit,
) {
  CenterAlignedTopAppBar(
    colors = TopAppBarDefaults.topAppBarColors(
      containerColor = MaterialTheme.colorScheme.surface,
      titleContentColor = MaterialTheme.colorScheme.onSurface,
      navigationIconContentColor = MaterialTheme.colorScheme.onSurface,
      actionIconContentColor = MaterialTheme.colorScheme.onSurface,
    ),
    title = {
      Text(
        text = stringResource(R.string.app_name),
        maxLines = 1,
        overflow = TextOverflow.Ellipsis
      )
    },
    navigationIcon = {
      IconButton(onClick = onMenuClick) {
        Icon(
          imageVector = Icons.Filled.Menu,
          contentDescription = "Menu"
        )
      }
    },
    actions = {
```

```
            IconButton(onClick = { }) {
                Icon(
                    imageVector = Icons.Filled.AccountCircle,
                    contentDescription = "User"
                )
            }
        }
    )
}
package com.pavlopodoliaka.arrowdo.ui.components
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.NavigationBar
import androidx.compose.material3.NavigationBarDefaults
import androidx.compose.material3.NavigationBarItem
import androidx.compose.material3.NavigationBarItemDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.navigation.NavHostController
import androidx.navigation.compose.currentBackStackEntryAsState
@Composable
fun ArrowDoNavigation(
    navController: NavHostController,
) {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route
    NavigationBar(
        containerColor = MaterialTheme.colorScheme.surface,
        contentColor = MaterialTheme.colorScheme.onSurface,
        windowInsets = NavigationBarDefaults.windowInsets,
    ) {
        Destination.entries
            .filter { it != Destination.DAY }
            .forEach { destination ->
                NavigationBarItem(
                    selected = destination.route == currentRoute,
                    onClick = {
                        if (currentRoute != destination.route) {
                            navController.navigate(destination.route) {
                                popUpTo(navController.graph.startDestinationId
) {
                                    saveState = true
                                }
                                launchSingleTop = true
                                restoreState = true
                            }
                        }
                    },
                    icon = {
                        Icon(
                            painter = painterResource(id = destination.iconId),
                            contentDescription = destination.contentDescription
,
                        )
                    },
                    label = {
                        Text(
                            stringResource(destination.labelId),
                            color = MaterialTheme.colorScheme.onSurface
```

```
                    )
                },
                colors = NavigationBarItemDefaults.colors(
                    selectedIconColor = MaterialTheme.colorScheme.secondary
,
                    unselectedIconColor = MaterialTheme.colorScheme.
onSurface,
                    indicatorColor = MaterialTheme.colorScheme.
secondaryContainer
                )
            )
        }
    }
}
package com.pavlopodoliaka.arrowdo.data
import androidx.room.Entity
import androidx.room.PrimaryKey
import java.time.DayOfWeek
import java.time.LocalDate
import java.time.LocalTime
@Entity(tableName = "tasks")
data class Task(
@PrimaryKey(autoGenerate = true) val id: Int = 0,
val day: LocalDate? = null,
val time: LocalTime,
val title: String,
val description: String,
val color: Int,
val dayType: DayOfWeek? = null,
val scheduled: Boolean = false,
)
package com.pavlopodoliaka.arrowdo.data
import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import kotlinx.coroutines.flow.Flow
import java.time.LocalDate
@Dao
interface TaskDao {
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insert(task: Task)
@Delete
suspend fun delete(task: Task)
@Query("SELECT * FROM tasks ORDER BY time ASC")
fun getAllTasks(): Flow<List<Task>>
@Query("SELECT * FROM tasks ORDER BY time ASC")
fun getAllTasksForBoot(): List<Task>
@Query("SELECT * FROM tasks WHERE day = :date ORDER BY time ASC")
fun getDayTasks(date: LocalDate): Flow<List<Task>>
}
package com.pavlopodoliaka.arrowdo.data
import androidx.room.TypeConverter
import java.time.DayOfWeek
import java.time.LocalDate
import java.time.LocalTime
import java.time.format.DateTimeFormatter
class Converter {
    @TypeConverter
    fun fromLocalDate(value: String?): LocalDate? {
        return value?.let { LocalDate.parse(it) }
```

```
    }
    @TypeConverter
    fun toLocalDate(date: LocalDate?): String? {
        return date?.format (DateTimeFormatter.ISO_LOCAL_DATE)
    }
    @TypeConverter
    fun fromLocalTime(value: String?): LocalTime? {
        return value?.let { LocalTime.parse(it) }
    }
    @TypeConverter
    fun toLocalTime(time: LocalTime?): String? {
        return time?.format (DateTimeFormatter.ISO_LOCAL_TIME)
    }
    @TypeConverter
    fun fromDayOfWeek(day: DayOfWeek?): String? {
        return day?.name
    }
    @TypeConverter
    fun toDayOfWeek(value: String?): DayOfWeek? {
        return value?.let { DayOfWeek.valueOf(it) }
    }
}
package com.pavlopodoliaka.arrowdo.data
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.TypeConverters
@Database(entities = [Task::class], version = 1, exportSchema = false)
@TypeConverters(Converter::class)
abstract class TaskDatabase : RoomDatabase() {
    abstract fun taskDao(): TaskDao
    companion object {
        @Volatile
        private var INSTANCE: TaskDatabase? = null
        fun getDatabase(context: Context): TaskDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    TaskDatabase::class.java,
                    "task_database"
                )
                    .fallbackToDestructiveMigration(true)
                    .build()
                INSTANCE = instance
                instance
            }
        }
    }
}
package com.pavlopodoliaka.arrowdo.data
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map
import java.time.DayOfWeek
import java.time.LocalDate
class TaskRepository(private val taskDao: TaskDao) {
    val allTasks: Flow<List<Task>> = taskDao.getAllTasks()
    fun getTasksForDay(date: LocalDate): Flow<List<Task>> {
        return taskDao.getDayTasks(date)
    }
    fun getTasksForToday(): Flow<List<Task>> {
        val today = LocalDate.now()
```

```
        val todayDayType = today.dayOfWeek
        return taskDao.getAllTasks().map { tasks ->
            tasks.filter { task ->
                (!task.scheduled && task.day == today) ||
                        (task.scheduled && task.dayType == todayDayType)
            }
        }
    }
    fun getTasksForTodayBoot(): List<Task> {
        val today = LocalDate.now()
        val todayDayType = today.dayOfWeek
        val allTasks = taskDao.getAllTasksForBoot()
        return allTasks.filter { task ->
            (!task.scheduled && task.day == today) ||
                    (task.scheduled && task.dayType == todayDayType)
        }
    }
    fun getTasksForDayType(dayOfWeek: DayOfWeek): Flow<List<Task>> {
        return taskDao.getAllTasks().map { tasks ->
            tasks.filter { task -> task.scheduled && task.dayType == dayOfWeek
}
        }
    }
    suspend fun addTask(task: Task) {
        taskDao.insert(task)
    }
    suspend fun deleteTask(task: Task) {
        taskDao.delete(task)
    }
}
package com.pavlopodoliaka.arrowdo.utils
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.util.Log
import com.pavlopodoliaka.arrowdo.data.TaskDatabase
import com.pavlopodoliaka.arrowdo.data.TaskRepository
import com.pavlopodoliaka.arrowdo.viewmodel.TaskViewModel
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
class BootReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent?) {
        CoroutineScope(Dispatchers.IO).launch {
            try {
                val database = TaskDatabase.getDatabase(context)
                val repository = TaskRepository(database.taskDao())
                val viewModel = TaskViewModel(repository)
                val todayTasks = viewModel.getTasksForTodayBoot()
                NotificationUtils.scheduleTodayNotifications(context,
todayTasks)
            } catch (e: Exception) {
                Log.e("BootReceiver", "Error scheduling", e)
            }
        }
    }
}
package com.pavlopodoliaka.arrowdo.utils
import android.app.AlarmManager
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
```

```
import android.os.Build
import com.pavlopodoliaka.arrowdo.data.Task
import java.time.LocalDate
import java.time.LocalDateTime
import java.time.ZoneId
object NotificationUtils {
  fun scheduleTodayNotifications(context: Context, tasks: List<Task>) {
    val alarmManager = context.getSystemService(Context.ALARM_SERVICE) as
AlarmManager
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S &&
      !alarmManager.canScheduleExactAlarms()
    ) {
      return
    }
    tasks.forEach { task ->
      val now = LocalDateTime.now()
      val taskDateTime = LocalDateTime.of(LocalDate.now(), task.time)
      if (taskDateTime.isAfter(now)) {
        val intent = Intent(context, TaskNotificationReceiver::class.
java).apply {
          putExtra("title", task.title)
          putExtra("description", task.description)
        }
        val pendingIntent = PendingIntent.getBroadcast(
          context,
          task.id,
          intent,
          PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.
FLAG_IMMUTABLE
        )
        alarmManager.cancel(pendingIntent)
        alarmManager.setExactAndAllowWhileIdle(
          AlarmManager.RTC_WAKEUP,
          taskDateTime.atZone(ZoneId.systemDefault()).toInstant().
toEpochMilli(),
          pendingIntent
        )
      }
    }
  }
}
package com.pavlopodoliaka.arrowdo.utils
import android.Manifest
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.graphics.Color
import android.media.AudioAttributes
import android.media.RingtoneManager
import android.os.Build
import androidx.core.app.ActivityCompat
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import com.pavlopodoliaka.arrowdo.R
import kotlin.random.Random
class TaskNotificationReceiver : BroadcastReceiver() {
  override fun onReceive(context: Context, intent: Intent) {
    createNotificationChannel(context)
    val title = intent.getStringExtra("title") ?: "Task Reminder"
```

```
        val description = intent.getStringExtra("description") ?: ""
        val notification = NotificationCompat.Builder(context, "task_channel_id
 ")
            .setSmallIcon(R.drawable.notification)
            .setContentTitle(title)
            .setContentText(description)
            .setPriority(NotificationCompat.PRIORITY_HIGH)
            .setCategory(NotificationCompat.CATEGORY_REMINDER)
            .setAutoCancel(true)
            .setGroupSummary(false)
            .build()
        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.TIRAMISU ||
            ActivityCompat.checkSelfPermission(
                context,
                Manifest.permission.POST_NOTIFICATIONS
            ) == PackageManager.PERMISSION_GRANTED
        ) {
            NotificationManagerCompat.from(context).notify(Random.nextInt(),
notification)
        }
    }
    private fun createNotificationChannel(context: Context) {
        val channel = NotificationChannel(
            "task_channel_id",
            "Task Reminders",
            NotificationManager.IMPORTANCE_HIGH
        ).apply {
            description = "Time-sensitive task reminders"
            enableLights(true)
            lightColor = Color.BLUE
            enableVibration(true)
            setSound(
                RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION
 ),
                AudioAttributes.Builder()
                    .setUsage(AudioAttributes.USAGE_NOTIFICATION)
                    .build()
}
 }
 )
 val manager = context.getSystemService(NotificationManager::class.java)
 manager.createNotificationChannel(channel)
 }
package com.pavlopodoliaka.arrowdo.viewmodel
 import androidx.compose.runtime.State
 import androidx.compose.runtime.mutableStateOf
 import androidx.lifecycle.ViewModel
 import androidx.lifecycle.viewModelScope
 import com.pavlopodoliaka.arrowdo.data.Task
 import com.pavlopodoliaka.arrowdo.data.TaskRepository
 import kotlinx.coroutines.flow.Flow
 import kotlinx.coroutines.flow.SharingStarted
 import kotlinx.coroutines.flow.StateFlow
 import kotlinx.coroutines.flow.stateIn
 import kotlinx.coroutines.launch
 import java.time.DayOfWeek
 import java.time.LocalDate
 open class TaskViewModel(private val repository: TaskRepository) : ViewModel
 () {
    val tasks: StateFlow<List<Task>> = repository.allTasks
        .stateIn(viewModelScope, SharingStarted.Lazily, emptyList())
    private val _selectedDate = mutableStateOf<LocalDate?>(null)
```

```
    val selectedDate: State<LocalDate?> = _selectedDate
    fun addTask(task: Task) {
        viewModelScope.launch {
            repository.addTask(task)
        }
    }
    fun deleteTask(task: Task) {
        viewModelScope.launch {
            repository.deleteTask(task)
        }
    }
    fun getTasksForDay(date: LocalDate): Flow<List<Task>> {
        return repository.getTasksForDay(date)
    }
    fun getTasksForToday(): Flow<List<Task>> {
        return repository.getTasksForToday()
    }
    fun getTasksForTodayBoot(): List<Task> {
        return repository.getTasksForTodayBoot()
    }
    fun getTasksForDayType(dayOfWeek: DayOfWeek): Flow<List<Task>> {
        return repository.getTasksForDayType(dayOfWeek)
    }
    fun setSelectedDate(date: LocalDate) {
        _selectedDate.value = date
    }
}
```

```
package com.pavlopodoliaka.arrowdo.viewmodel
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.pavlopodoliaka.arrowdo.ui.screens.Day
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.SharingStarted
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.flow.stateIn
import kotlinx.coroutines.flow.update
import java.time.LocalDate
import java.time.YearMonth
class CalendarViewModel : ViewModel() {
    private val _selectedMonth = MutableStateFlow(YearMonth.now())
    val selectedMonth: StateFlow<YearMonth> = _selectedMonth.asStateFlow()
    val days: StateFlow<List<Day>> = selectedMonth.map { month ->
        val  today = LocalDate.now()
        val firstDay = month.atDay(1)
        val daysInMonth = month.lengthOfMonth()
        val startDayOfWeek = firstDay.dayOfWeek.value - 1
        buildList {
            repeat(startDayOfWeek) {
                add(Day(null))
            }
            for (day in 1..daysInMonth) {
                val date = month.atDay(day)
                add(Day(date = date, isToday = date == today))
            }
            while (size % 7 != 0) {
                add(Day(null))
            }
        }
    }.stateIn(viewModelScope, SharingStarted.Eagerly, emptyList())
    fun prevMonth() {
```

```
            _selectedMonth.update { if(it > YearMonth.of(2020, 1)) it.minusMonths(1
        ) else it }
          }
          fun nextMonth() {
            _selectedMonth.update { if(it < YearMonth.now().plusYears(5)) it.
        plusMonths(1) else it}
          }
        }
        package com.pavlopodoliaka.arrowdo.viewmodel
        import androidx.lifecycle.ViewModel
        import androidx.lifecycle.ViewModelProvider
        import com.pavlopodoliaka.arrowdo.data.TaskRepository
        class TaskViewModelFactory(private val repository: TaskRepository) :
        ViewModelProvider.Factory {
          override fun <T : ViewModel> create(modelClass: Class<T>): T {
            if (modelClass.isAssignableFrom(TaskViewModel::class.java)) {
              @Suppress("UNCHECKED_CAST")
              return TaskViewModel(repository) as T
            }
            throw IllegalArgumentException("Unknown ViewModel class")
          }
}
        <?xml version="1.0" encoding="utf-8"?>
        <resources>
        <color name="purple_200">#FFBB86FC</color>
        <color name="purple_500">#FF6200EE</color>
        <color name="purple_700">#FF3700B3</color>
        <color name="teal_200">#FF03DAC5</color>
        <color name="teal_700">#FF018786</color>
        <color name="black">#FF000000</color>
        <color name="white">#FFFFFFFF</color>
        <color name="splash">#FFF9FAEF</color>
        </resources>

        <?xml version="1.0" encoding="utf-8"?>
        <resources>
        <style name="Theme.ArrowDoSplashScreen" parent="Theme.SplashScreen">
        <item name="windowSplashScreenBackground">@color/splash</item>
        <item name="windowSplashScreenAnimatedIcon">@drawable/
        arrow_do_with_title</item>
        <item name="postSplashScreenTheme">@style/Theme.ArrowDo</item>
        </style>
        <style name="Theme.ArrowDo" parent="android:Theme.Material.Light.
        NoActionBar" />
        </resources>

        <resources>
          <string name="app_name">Arrow Do</string>
          <string name="left_screen_label">Schedule</string>
          <string name="main_screen_label">Today</string>
          <string name="right_screen_label">Month</string>
          <string name="single_day_label">Single Day</string>
          <string name="cancel_button">Cancel</string>
          <string name="save_button">Save</string>
          <string name="task_title">Task Title</string>
          <string name="title_warning">Title can\'t be empty!</string>
          <string name="time_warning">Time can\'t be empty!</string>
          <string name="time">Time</string>
          <string name="description">Description</string>
          <string name="ok">OK</string>
          <string name="day_type_label">Day of the Week</string>
          <string name="color_choice">Choose Task Color</string>
```

```
    <string name="settings">Settings</string>
    <string name="about">About</string>
    <string name="home_hint">Tap the + button to add a task</string>
    <string name="week_hint">There are no tasks for this day yet</string>
    <string name="day_warning">Day of the Week can\'t be empty!</string>
    <string name="date">Date</string>
    <string name="arrow_do">Arrow Do</string>
    <string name="toast_exact_alarm">To deliver reminders on time, please allow
  exact alarms for Arrow Do</string>
  </resources>
 <?xml version="1.0" encoding="utf-8"?>
  <resources>
  <color name="ic_launcher_background">#FFFFFF</color>
</resources>
```