|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ИВАНОВСКОЙ ОБЛАСТИ  Областное государственное бюджетное профессиональное образовательное учреждение  «Ивановский промышленно-экономический колледж» | | | | | |
|  | | | | | |
| **КУРСОВОЙ ПРОЕКТ** | | | | | |
|  | | | | | |
| **Разработка программного обеспечения мобильного приложения для ведения дневника и блога** | | | | | |
| **ИВПЭК. 09.02.07. 22** | | | | | |
|  | | | | | |
|  | | **Специальность:** | | **09.02.07 Информационные системы и программирование,**  **базовая подготовка** | |
|  | | | | | |
|  | | | | | |
| Руководитель курсового проекта | | |  | | Р.В. Вяткин |
| Выполнил обучающийся группы 407а | | |  | | Д.Г. Карташов |
|  | | |  | |  |
|  | | | | | |
|  | | | | | |
| Курсовой проект выполнен и защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |
|  | | | | | |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г. | | | | | |
|  |  | | | | |
|  |  | | | |  |
|  |  | | | |  |
|  | Иваново 2025 | | | |  |
|  | | | | | |
|  | | | | | |
|  | | | | | |

Содержание

[Введение 3](#_Toc193795878)

[1 Концептуальное проектирование 5](#_Toc193795879)

[1.1 Теоретический вопрос 5](#_Toc193795880)

[1.2 Спецификация требований программного обеспечения 8](#_Toc193795881)

[1.3 Требование к программному средству 8](#_Toc193795882)

[2. Техно-рабочий проект 12](#_Toc193795883)

[2.1 Алгоритм решения поставленной задачи программного продукта 12](#_Toc193795884)

[2.2 Обоснование выбора средств разработки 14](#_Toc193795885)

[2.3 Разработка макета приложения 18](#_Toc193795886)

[2.4 Описание разработки приложения 20](#_Toc193795887)

[2.5 Тестирование приложения 22](#_Toc193795888)

[3. Рабочая документация 24](#_Toc193795889)

[3.1 Руководство пользователя 24](#_Toc193795890)

[3.2 Руководство программиста 39](#_Toc193795891)

[Заключение 44](#_Toc193795892)

[Список использованной литературы 45](#_Toc193795893)

[Приложение 46](#_Toc193795894)

Введение

В современную эпоху цифровизации ведение личных записей и публикация контента становится неотъемлемой частью повседневной жизни. Мобильные устройства являются основным инструментом для создания и потребления контента, что создает потребность в удобных и функциональных приложениях для ведения блогов и дневников.Актуальность разработки мобильного приложения DayBlog обусловлена растущим спросом на персонализированные инструменты для ведения записей с возможностью их публикации. Существующие решения часто либо слишком сложны для повседневного использования, либо не предоставляют необходимой функциональности для управления личным контентом.Цель работы: разработка Android-приложения для ведения личного дневника и блога с возможностью публикации записей, поддержкой мультимедийного контента и современным пользовательским интерфейсом.

Задачи проекта:

1. Анализ существующих решений и определение ключевых функциональных требований.
2. Проектирование архитектуры приложения с использованием современных технологий Android.
3. Разработка системы хранения и управления записями с использованием Room Database.
4. Реализация пользовательского интерфейса на базе Jetpack Compose.
5. Внедрение функционала работы с изображениями и медиафайлами.
6. Создание системы публикации и управления контентом.
7. Тестирование и оптимизация производительности приложения.

Объект исследования: процесс разработки мобильного приложения для ведения личного дневника и блога.Предмет исследования: методы и технологии создания Android-приложения с использованием современного стека технологий: Kotlin, Jetpack Compose, Room Database, Material Design 3.Практическая значимость работы заключается в создании удобного инструмента для ведения личных записей с возможностью их публикации, который может быть использован широким кругом пользователей. Приложение предоставляет функционал для создания, редактирования и организации записей, поддерживает работу с медиафайлами и имеет интуитивно понятный интерфейс. Результатом работы является полнофункциональное Android-приложение DayBlog, реализующее все поставленные задачи и соответствующее современным стандартам разработки мобильных приложений.

1 Концептуальное проектирование

1.1 Теоретический вопрос

В рамках концептуального проектирования мобильного приложения DayBlog необходимо рассмотреть теоретические аспекты разработки современных мобильных приложений для ведения дневников и блогов. На основе анализа кодовой базы проекта можно выделить следующие ключевые теоретические направления:

1. Архитектурные принципы.

Приложение DayBlog построено на основе современных архитектурных паттернов:

* MVVM (Model-View-ViewModel) архитектура, реализованная через BlogViewModel, HomeViewModel и другие компоненты;
* Clean Architecture с четким разделением на слои данных, домена и представления;
* Dependency Injection для управления зависимостями компонентов;
* Repository Pattern для абстракции источников данных.

1. Технологический стек.

Проект использует современный стек технологий Android-разработки:

* Kotlin как основной язык программирования;
* Jetpack Compose для построения пользовательского интерфейса;
* Room Database для локального хранения данных;
* Coroutines и Flow для асинхронных операций;
* Material Design 3 для обеспечения современного визуального стиля.

1. Управление данными.

Система управления данными включает:

* локальное хранение через Room Database с использованием сущностей BlogEntryEntity и DiaryEntryEntity;
* валидация данных на уровне моделей (BlogEntry.validate());
* поддержка различных типов контента (текст, изображения);
* система тегов для категоризации записей;
* механизмы публикации и управления видимостью контента.

1. Пользовательский интерфейс.

Теоретические аспекты UI/UX включают:

* декларативный подход к построению интерфейса с Jetpack Compose;
* компонентный подход с переиспользуемыми элементами (BlogEntryCard, Modal);
* адаптивный дизайн с поддержкой различных размеров экрана;
* система навигации на основе NavController;
* анимации и визуальная обратная связь.

1. Безопасность и производительность.

Теоретические основы безопасности и оптимизации:

* управление правами доступа к файловой системе через Android Permissions;
* оптимизация работы с изображениями через библиотеку Coil;
* эффективное управление памятью при работе с медиафайлами;
* кэширование и оптимизация запросов к базе данных;
* валидация пользовательского ввода.

1. Функциональные возможности.

Теоретическое обоснование функционала:

* создание и редактирование записей с поддержкой форматирования;
* система тегов для организации контента;
* управление медиафайлами с поддержкой различных источников;
* механизм публикации с контролем видимости;
* поиск и фильтрация контента.

1. Масштабируемость и поддержка.

Теоретические аспекты развития приложения:

* модульная архитектура для простоты расширения функционала;
* версионирование базы данных для обновлений схемы;
* поддержка различных версий Android API;
* механизмы миграции данных;
* cистема логирования и отладки.

Данные теоретические аспекты формируют фундаментальную базу для разработки приложения DayBlog, обеспечивая его надежность, производительность и удобство использования. Понимание этих концепций позволяет создать качественный продукт, соответствующий современным требованиям к мобильным приложениям.Теоретическая база проекта основана на актуальных практиках разработки Android-приложений и учитывает специфику предметной области ведения дневников и блогов. Это обеспечивает создание эффективного решения, отвечающего потребностям пользователей и техническим требованиям платформы.

1.2 Спецификация требований программного обеспечения

Анализ осуществимости ИС – разработка приложения осуществима и не требует крупных инновационных технологий. Аналоги проекта:

Таблица 1 – Аналоги проекта

|  |  |  |  |
| --- | --- | --- | --- |
| Приложение | Преимущества | Недостатки | Вывод |
| WordPress | - Большое количество тем и плагинов.  - Поддежка SEO. | - Сложна для новичков.  - Требуется хостинг. | - Разработать продвинутый редактор. |
| Wix | - Удобный интерфейс.  - Быстрая настройка. | - Ограничение для SEO.  - Менее гибкий чем другие. | - Создать современный пользовательский интерфейс. - Улучшить производительность. |
| Blogger | - Простой в использовании.  - Интеграция с Google. - Бесплатный хостинг. | - Ограничение функций.  - Ограничение на дизайн и настройки. | - Улучшить индивидуальность. |

1.3 Требование к программному средству

Управление записями:

* создание новых записей с заголовком (3-100 символов) и содержанием (50-20000 символов);
* редактирование существующих записей;
* удаление записей с подтверждением;
* просмотр детальной информации о записи;
* поддержка тегов (до 5 тегов на запись);
* автоматическое сохранение даты и времени создания.

Работа с медиафайлами:

* прикрепление изображений к записям;
* поддержка форматов изображений: JPEG, PNG;
* предварительный просмотр изображений;
* оптимизация размера изображений;
* управление разрешениями для доступа к медиафайлам.

Система публикации:

* переключение статуса публикации записи;
* разделение на личные и публичные записи;
* управление видимостью контента;
* возможность отмены публикации.

Поиск и навигация:

* полнотекстовый поиск по заголовкам и содержимому;
* фильтрация по датам;
* фильтрация по тегам;
* сортировка записей по дате;
* навигация между разделами приложения.

Нефункциональные требования

Производительность:

* время запуска приложения не более 3 секунд;
* время отклика интерфейса не более 0.1 секунды;
* плавная анимация (60 fps);
* эффективное использование памяти при работе с изображениями;
* оптимизированные запросы к базе данных.

Надежность:

* сохранение данных при внезапном закрытии приложения;
* валидация пользовательского ввода;
* обработка ошибок при работе с файловой системой;
* корректная работа с различными размерами экрана;
* поддержка оффлайн-режима.

Безопасность:

* безопасное хранение локальных данных;
* контроль доступа к файловой системе;
* защита от SQL-инъекций при работе с базой данных;
* безопасная обработка пользовательского ввода;
* защита приватных записей.

Удобство использования:

* интуитивно понятный интерфейс;
* поддержка жестов навигации;
* визуальная обратная связь при действиях;
* доступность всех функций в пределах 3 нажатий;
* поддержка светлой и темной темы.

Масштабируемость:

* модульная архитектура для простоты расширения;
* возможность добавления новых типов контента;
* поддержка будущих версий Android;
* возможность локализации интерфейса;
* расширяемая система хранения данных;
* технические требования.

Системные требования:

* минимальная версия Android: API 24 (Android 7.0);
* целевая версия Android: API 35;
* поддержка различных разрешений экрана;
* минимальные требования к памяти: 2 ГБ RAM;
* требуемое место на устройстве: 50 МБ.

Зависимости:

* Kotlin версии 2.0.0;
* Jetpack Compose BOM версии 2024.04.01;
* Room Database версии 2.6.1;
* Coil для работы с изображениями версии 2.5.0;
* Material Design 3 компоненты.

Требования к разработке:

* использование MVVM архитектуры;
* следование принципам Clean Architecture;
* покрытие кода тестами;
* документирование кода;
* соблюдение Android coding guidelines.

Данная спецификация требований обеспечивает основу для разработки надежного, производительного и удобного мобильного приложения, соответствующего современным стандартам разработки под Android.

# 2 Техно-рабочий проект

2.1 Алгоритм решения поставленной задачи программного продукта

На основе анализа кодовой базы проекта DayBlog, алгоритм решения задачи включает следующие ключевые компоненты и процессы:

* инициализация приложения;
* запуск DayBlogApplication;
* инициализация базы данных Room;
* настройка внедрения зависимостей;
* конфигурация основных компонентов;
* структура навигации;
* главная активность (MainActivity);
* система навигации через NavController;
* нижняя панель навигации (BottomNav);
* управление стеком переходов между экранами;
* управление данными;

Создание записи:

* получение данных из формы;
* валидация входных данных;
* создание объекта BlogEntry;
* сохранение в базу данных;
* обновление списка записей.

Редактирование записи:

* загрузка существующей записи;
* внесение изменений;
* валидация изменений;
* обновление в базе данных;
* обновление UI;
* работа с медиафайлами.

Алгоритм обработки изображений:

* проверка необходимых разрешений;
* использование системного выбора файлов;
* оптимизация через библиотеку Coil;
* сохранение ссылки в базе данных;
* система публикации.

Процесс публикации записи:

* проверка валидности записи;
* изменение статуса публикации;
* обновление UI для отображения статуса;
* синхронизация изменений;
* поиск и фильтрация.

Алгоритм поиска:

* полнотекстовый поиск по базе;
* фильтрация по датам и тегам;
* сортировка результатов;
* обновление списка записей;
* управление состоянием.

Процесс обработки состояний:

* использование StateFlow;
* обработка загрузки данных;
* управление ошибками;
* обновление интерфейса;
* кэширование и оптимизация.

Процесс кэширования:

* кэширование изображени;
* оптимизация запросов к базе;
* управление памятью;
* асинхронные операции;
* обработка ошибок.

Алгоритм обработки ошибок:

* валидация пользовательского ввода;
* обработка исключений;
* информирование пользователя;
* восстановление состояния.

Данный алгоритм обеспечивает эффективное решение поставленной задачи, учитывая все требования к функциональности, производительности и удобству использования приложения. Модульная структура и четкое разделение ответственности позволяют поддерживать и расширять функционал приложения в будущем.

2.2 Обоснование выбора средств разработки

Выбор средств разработки обоснован следующими факторами:

1. Язык программирования Kotlin (версия 2.0.0).

Обоснование выбора:

* официальный язык для Android-разработки;
* полная совместимость с Java и Android SDK;
* современные языковые конструкции и безопасность типов;
* поддержка корутин для асинхронного программирования;
* лаконичный и выразительный синтаксис.

1. Jetpack Compose (версия 2024.04.01).

Преимущества использования:

* декларативный подход к построению UI;
* улучшенная производительность отрисовки;
* встроенная поддержка анимаций и жестов;
* упрощенное создание адаптивных интерфейсов;
* богатая экосистема готовых компонентов.

1. Room Database (версия 2.6.1).

Причины выбора:

* абстракция над SQLite с типобезопасностью;
* встроенная поддержка Kotlin Coroutines;
* автоматическая генерация кода;
* эффективное кэширование и управление данными;
* поддержка сложных запросов и миграций.

1. Material Design 3.

Обоснование применения:

* современные принципы дизайна от Google;
* готовые компоненты высокого качества;
* адаптивность и масштабируемость;
* поддержка тем и стилизации;
* согласованность с платформой Android.

1. Coil (версия 2.5.0).

Преимущества для работы с изображениями:

* оптимизированная загрузка и кэширование;
* встроенная поддержка Compose;
* эффективное управление памятью;
* гибкая система трансформаций;
* малый размер библиотеки;

1. Android Architecture Components.

Обоснование использования:

* ViewModel для управления UI-данными;
* LiveData/StateFlow для реактивного программирования;
* Navigation для управления переходами;
* Lifecycle для управления жизненным циклом;
* WorkManager для фоновых задач.

1. Инструменты разработки.

Выбор инструментария:

* Android Studio для разработки;
* Gradle для сборки проекта;
* Git для версионного контроля;
* JUnit для модульного тестирования;
* Espresso для UI-тестирования.

1. Дополнительные библиотеки.

Обоснование включения:

* KSP для генерации кода;
* Navigation Compose для навигации;
* Material Icons Extended для иконок;
* AndroidX Core для базового функционала;
* Lifecycle компоненты для управления состоянием.

Технические преимущества выбранного стека:

1. Производительность:

* оптимизированная компиляция Kotlin;
* эффективная работа Compose;
* быстрые операции с базой данных;
* оптимальное использование памяти.

1. Надежность:

* строгая типизация Kotlin;
* встроенная обработка ошибок;
* стабильные релизы библиотек;
* широкое тестовое покрытие.

1. Масштабируемость:

* модульная архитектура;
* возможность расширения функционала;
* поддержка новых версий Android;
* гибкость настройки компонентов.

1. Поддержка:

* активное сообщество разработчиков;
* регулярные обновления библиотек;
* обширная документация;
* готовые решения типовых задач.

Выбранные средства разработки обеспечивают оптимальный баланс между функциональностью, производительностью и удобством разработки. Современный стек технологий позволяет создать качественное приложение, соответствующее всем поставленным требованиям и готовое к дальнейшему развитию.

2.3 Разработка макета приложения

Начали с разработки начального экрана, которое включает в себя кран со списком, разделенным на категории и ФАБ кнопку для вызова экрана (всплывающего окна) пример представлен на Рисунок – 1. Список имеет стиль в линию у которого с левой стороны превьюшка, далее идет название и иконка.

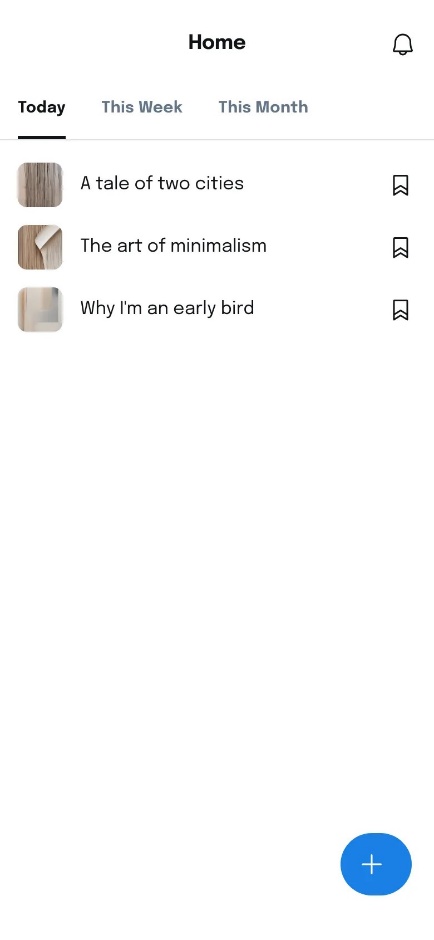


Рисунок - Макет главного экрана

Далее идет экран с блогом, на текущем месте идут как обычно карточки новостей(статей), их можно посмотреть на Рисунок – 2.

В каждой карточке включает заголовок, акцентного цвета, и подзаголовок, второго цвета из палитры.

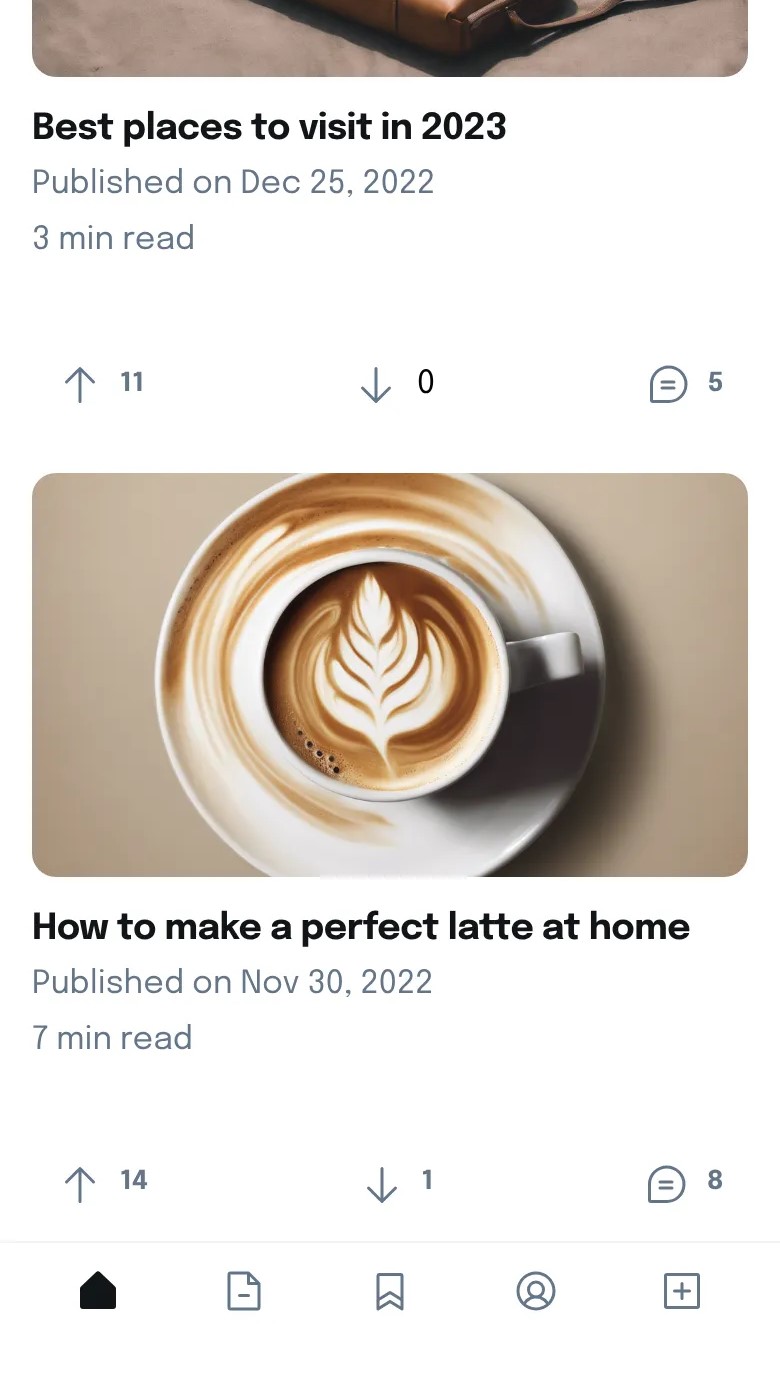


Рисунок - Пример экрана блога

Экран дневника — это приватный экран, который включает заметки пользователя включающих заголовок и метку времени изображенной на Рисунок – 3, также включает ФАБ иконку для создания ещё одной записи.

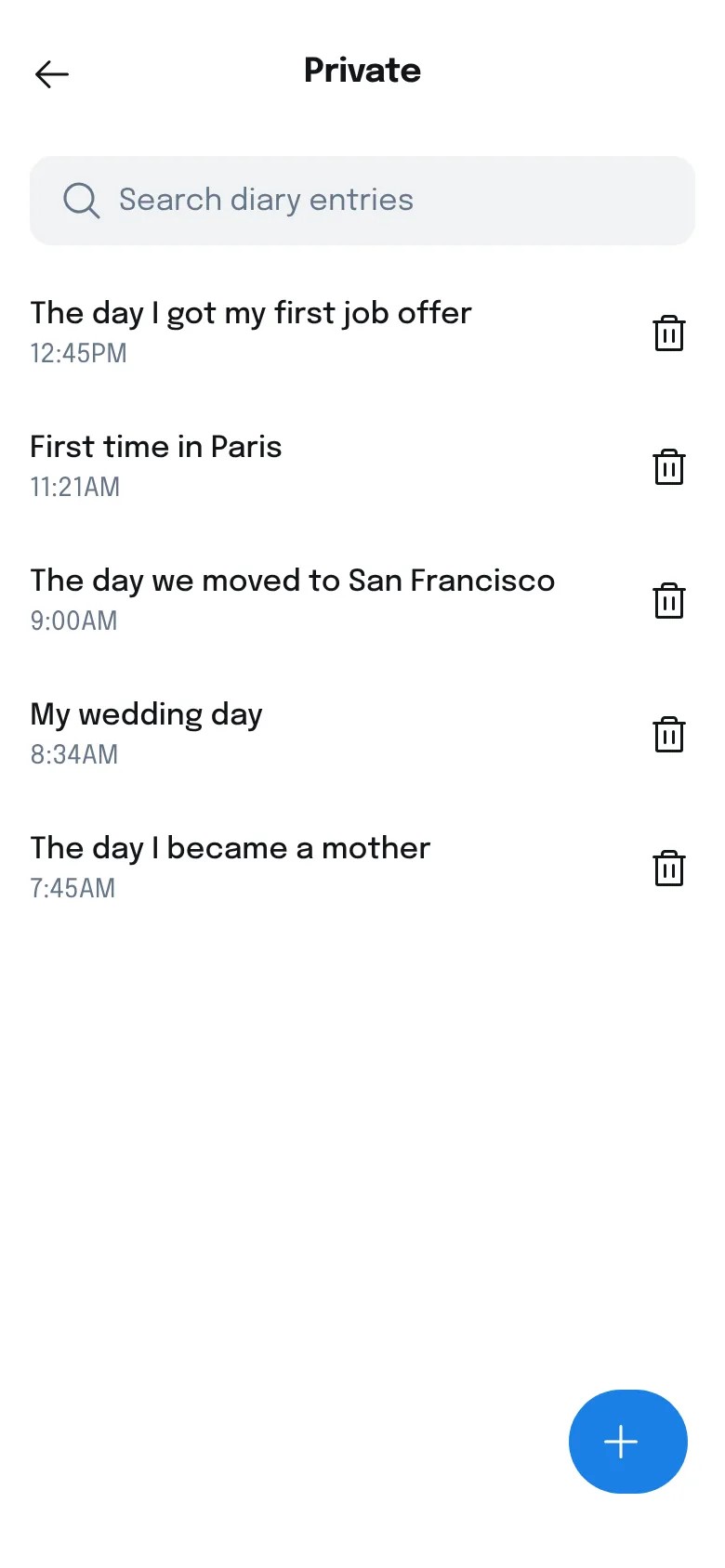


Рисунок - Пример экрана дневника

2.4 Описание разработки приложения

Архитектура приложения

В процессе разработки мобильного приложения для ведения дневника и блога была применена архитектура MVVM (Model-View-ViewModel). Данная архитектура была выбрана благодаря её преимуществам в разделении ответственности между компонентами и обеспечении тестируемости кода. Приложение разработано с использованием современного инструментария Android, включая Jetpack Compose для построения пользовательского интерфейса.Структура проекта организована по принципу разделения на модули в соответствии с их функциональным назначением:

* модуль data содержит классы для работы с данными;
* модуль domain включает бизнес-логику приложения;
* модуль ui отвечает за представление информации пользователю;
* модуль di реализует внедрение зависимостей.

Реализация слоя данных

В качестве основы для хранения данных была выбрана локальная база данных Room. Разработаны две основные сущности: DiaryEntry и BlogEntry, представляющие записи дневника и блога соответственно. Для каждой сущности определены ограничения валидации, такие как минимальная и максимальная длина заголовка и содержимого.Репозитории обеспечивают абстракцию доступа к данным и инкапсулируют логику взаимодействия с локальным хранилищем. Для каждого типа данных создан отдельный репозиторий: DiaryRepository и BlogRepository. Это позволило изолировать бизнес-логику от деталей реализации хранения данных.

Разработка пользовательского интерфейса

Пользовательский интерфейс приложения разработан с использованием декларативного подхода Jetpack Compose. Основные экраны приложения включают:

* HomeScreen - главный экран с обзором последних записей;
* DiaryScreen - экран со списком записей дневника;
* BlogScreen - экран со списком записей блога;
* ProfileScreen - экран профиля пользователя;
* CreateDiaryScreen - экран создания записи дневника;
* CreateBlogScreen - экран создания записи блога;
* DiaryDetailScreen - экран детального просмотра записи дневника;
* BlogDetailScreen - экран детального просмотра записи блога.

Для навигации между экранами использована библиотека Navigation Compose, позволяющая организовать перемещение между различными частями приложения. Реализована нижняя навигационная панель (BottomNav) для быстрого доступа к основным разделам.

Реализация бизнес-логики

Бизнес-логика приложения инкапсулирована в ViewModels, которые обеспечивают связь между пользовательским интерфейсом и данными. Для каждого основного экрана создана соответствующая ViewModel:

* HomeViewModel;
* DiaryViewModel;
* BlogViewModel;
* ProfileViewModel.

ViewModels отвечают за обработку пользовательских действий, валидацию вводимых данных и обновление состояния UI. Применение паттерна Factory позволило организовать создание ViewModels с необходимыми зависимостями.

Оптимизация производительности

В ходе разработки были применены различные техники оптимизации производительности:

* использование корутин для асинхронных операций;
* применение LazyColumn для эффективного отображения списков;
* кэширование данных для уменьшения количества обращений к базе данных;
* оптимизация запросов к базе данных.

Данные меры позволили обеспечить плавную работу приложения даже при большом количестве записей в дневнике и блоге.

2.5 Тестирование приложения

В процессе разработки было проведено модульное тестирование компонентов приложения. Тесты размещены в директориях test и androidTest. Особое внимание уделено тестированию бизнес-логики и валидации данных.Для проверки корректности работы пользовательского интерфейса применялось ручное тестирование на различных устройствах и версиях Android. Выявленные в ходе тестирования ошибки были исправлены, что позволило повысить стабильность работы приложения.

# 3 Рабочая документация

3.1 Руководство пользователя

Диаграмма вариантов использования (Рисунок - 7) для приложения "Paint Sort: Coloring Puzzle" представляет собой визуальное описание взаимодействия пользователя с системой. Она включает ключевые функции приложения и их взаимосвязи. Основным актором в данной диаграмме является Пользователь, который взаимодействует с игрой для выполнения различных задач [5].

Основные функции приложения представлены следующими вариантами использования:

1) Просмотр главного экрана – Пользователь может просматривать главный экран приложения, который является отправной точкой для всех функций.

2) Работа с дневником – Пользователь может создавать, просматривать, удалять и искать записи в личном дневнике.

3) Работа с блогом – Пользователь может управлять публичными записями блога, включая создание, просмотр, удаление и настройку видимости записей.

4) Управление профилем – Пользователь может просматривать свой профиль и связанную статистику.

Дополнительные функции включают:

1. поиск по записям – Пользователь может искать как в дневнике, так и в блоге.
2. работа с изображениями – Пользователь может добавлять изображения к записям дневника и блога.
3. управление тегами – Пользователь может добавлять теги к записям блога и фильтровать записи по ним.
4. Связи между вариантами использования включают:

* Include – Некоторые функции, такие как "Добавление изображений" и "Добавление тегов", являются включаемыми при создании записей;
* Extend – Функция "Фильтрация по тегам" расширяет возможности просмотра списка записей блога;
* просмотр статистики – Является включаемой функцией при просмотре профиля пользователя.

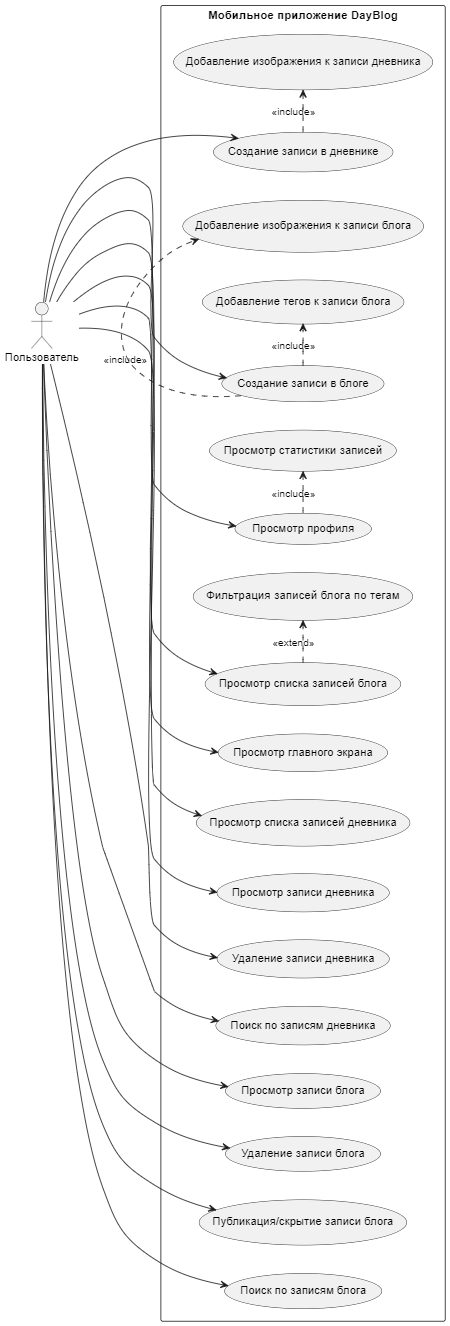


Рисунок - Диаграмма использования

Диаграмма последовательностей (Рисунок 5) для создания записи в дневнике описывает взаимодействие между пользователем и различными компонентами системы в процессе создания новой записи. Основная цель диаграммы — показать последовательность действий при создании записи в дневнике и обмен сообщениями между компонентами системы.

В диаграмме присутствуют следующие акторы и объекты (Таблица 2):

Таблица 2 – акторов и обьектов

|  |  |  |
| --- | --- | --- |
| Элемент | Тип | Назначение |
| Пользователь | Актор | Представляет конечного пользователя, который взаимодействует с приложением. |
| DiaryScreen | Объект | Основной экран дневника, отображающий список записей. |
| CreateDiaryScreen | Объект | Интерфейс создания новой записи в дневнике. |
| DiaryViewModel | Объект | Управляет бизнес-логикой и состоянием пользовательского интерфейса. |
| DiaryRepository | Объект | Обеспечивает доступ к данным и их сохранение в базе данных. |
| DiaryDao | Объект | Предоставляет методы для непосредственной работы с базой данных. |

Диаграмма отражает следующий процесс:

1. Инициация создания:

* пользователь нажимает кнопку "+" на экране дневника;
* DiaryScreen передает команду в ViewModel для отображения модального окна;
* система отображает экран создания новой записи (CreateDiaryScreen).

1. Ввод информации:

* пользователь вводит заголовок записи в дневнике;
* пользователь вводит основное содержание записи;
* при желании пользователь может добавить изображение через ImagePicker;
* система обеспечивает интерфейс для ввода всех необходимых данных.

1. Процесс сохранения:

* пользователь нажимает кнопку "Save" для сохранения записи;
* ViewModel создает новый объект DiaryEntry;
* система выполняет валидацию введенных данных.

4) Обработка в базе данных:

* Repository преобразует объект DiaryEntry в формат для базы данных;
* DiaryDao выполняет SQL-запрос для вставки новой записи;
* база данных сохраняет информацию и возвращает ID новой записи.

1. Обновление интерфейса:

* система закрывает экран создания записи;
* DiaryScreen запрашивает обновленный список записей;
* Repository получает все записи из базы данных;
* пользователю отображается обновленный список с новой записью.

1. Обработка ошибок:

* в случае неуспешной валидации система отображает сообщение об ошибке;
* пользователь может исправить данные и повторить попытку сохранения.

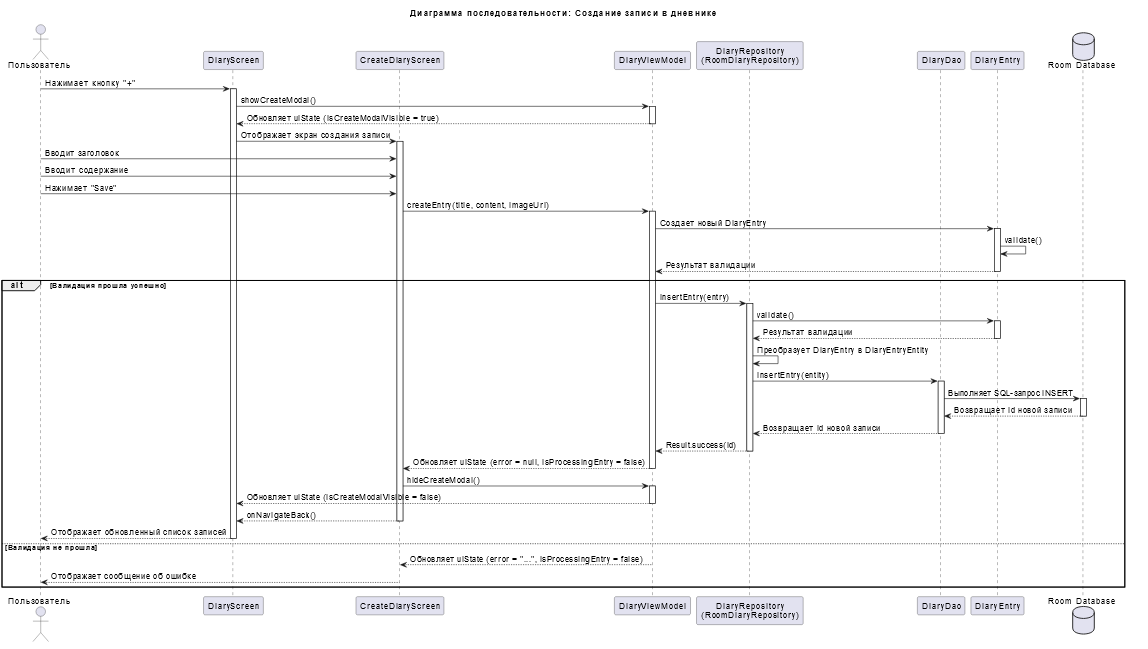


Рисунок - Диаграмма последовательностей

Диаграмма кооперации (Рисунок 6) для системы ведения дневника представляет собой визуализацию взаимодействия между различными объектами системы в рамках процесса создания записей. Она демонстрирует, как объекты взаимодействуют друг с другом для выполнения задачи создания и сохранения записей в дневнике.

Диаграмма включает следующие объекты:

1. Редактор записи – Основной интерфейс, где пользователь создает и редактирует записи дневника.
2. Система управления – Координирует процесс создания и сохранения записей.
3. Валидатор – Проверяет корректность введенных данных.
4. Хранилище данных – Управляет сохранением записей в базе данных.

Последовательность взаимодействий отражает процесс создания записи:

Этап 1: Инициация создания записи:

* Шаг 1: пользователь начинает создание новой записи через интерфейс редактора;
* Шаг 2: редактор записи инициализирует новую сессию создания записи.

Этап 2: Ввод и валидация данных:

* Шаг 3: пользователь вводит содержимое записи;
* Шаг 4: система проводит валидацию введенных данных;
* Шаг 5: валидатор проверяет корректность формата и содержания.

Этап 3: Обработка данных:

* Шаг 6: система управления обрабатывает валидированные данные;
* Шаг 7: происходит подготовка данных к сохранению.

Этап 4: Сохранение записи:

* Шаг 8: система передает данные в хранилище;
* Шаг 9: хранилище данных сохраняет запись в базе данных;
* Шаг 10: система подтверждает успешное сохранение.

Этап 5: Завершение процесса:

* Шаг 11: пользователь получает подтверждение о создании записи;
* Шаг 12: система возвращается в исходное состояние готовности к созданию новой записи.

Последовательность взаимодействий отражает процесс выполнения программы:



Рисунок – Диаграмма кооперации

Диаграмма развертывания (Рисунок - 7) для приложения "DayBlog" представляет собой блок-схему процесса создания и сохранения записей в дневнике. Она демонстрирует последовательность действий пользователя — от открытия приложения до сохранения записи. Диаграмма построена по многоуровневой структуре, где каждый последующий этап логически связан с предыдущим [1].

1. Начальный этап: «Пользователь открывает главный экран приложения»:

* DiaryScreen является основной точкой входа для взаимодействия с приложением;
* пользователь начинает работу с просмотра существующих записей и возможности создания новых.

1. Действие: «Пользователь нажимает кнопку создания новой записи»:

* система отображает специальный экран CreateDiaryScreen;
* этот экран предоставляет интерфейс для создания новой записи в дневнике.

1. Действие: «Пользователь вводит содержимое записи»:

* пользователь заполняет необходимые поля (заголовок, текст);
* система проверяет корректность введенных данных через DiaryViewModel.

1. Действие: «Система обрабатывает данные»:

* DiaryViewModel обрабатывает введенную информацию;
* DiaryRepository подготавливает данные для сохранения.

1. Действие: «Система сохраняет запись в базе данных»:

* DiaryDao выполняет необходимые операции с базой данных;
* Room Database обеспечивает надежное хранение информации.

1. Действие: «Система подтверждает сохранение»:

* база данных отправляет подтверждение успешного сохранения;
* система обновляет интерфейс для отображения новой записи.

1. Действие: «Система возвращается на главный экран»:

* CreateDiaryScreen закрывается;
* DiaryScreen обновляется с новой записью.

1. Завершающий этап: «Пользователь видит обновленный список записей»:

* новая запись появляется в списке на главном экране;
* система готова к созданию следующих записей.

Каждый этап поддерживается соответствующими компонентами системы:

* уровень представления (DiaryScreen, CreateDiaryScreen);
* уровень бизнес-логики (DiaryViewModel);
* уровень данных (DiaryRepository, DiaryDao);
* уровень хранения (Room Database).

Эта структура обеспечивает надежное и последовательное создание записей в дневнике, с соответствующей валидацией данных и обработкой ошибок на каждом этапе процесса.[1] Структура и взаимодействие компонентов определены в соответствии с принципами Clean Architecture и MVVM паттерном.

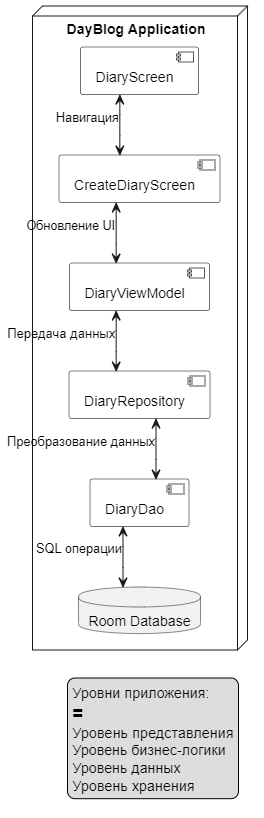


Рисунок - Диаграмма развертывания

Диаграмма деятельности (Рисунок - 8) описывает последовательность действий пользователя в приложении DayBlog, начиная с открытия приложения и заканчивая созданием записей в дневнике. Диаграмма отображает логику взаимодействия пользователя с системой, включая просмотр записей, создание новых записей и различные варианты работы с контентом [1].

Основные этапы диаграммы:

1. Открытие приложения:

* пользователь запускает DayBlog;
* отображается главный экран (DiaryScreen) со списком записей.

1. Выбор действия на главном экране:

* пользователь может просматривать существующие записи;
* создавать новые записи через кнопку "+";
* осуществлять поиск по записям;
* переходить к настройкам профиля.

1. Создание новой записи:

* пользователь нажимает кнопку создания;
* система отображает CreateDiaryScreen;
* пользователь вводит заголовок и содержание;
* опционально добавляет изображение.

1. Процесс сохранения:

* DiaryViewModel валидирует введенные данные;
* создается объект DiaryEntry;
* Repository преобразует данные для сохранения;
* DiaryDao выполняет операции с базой данных.

1. Проверка сохранения:

* если сохранение успешно, запись добавляется в базу данных;
* система возвращается на главный экран;
* отображается обновленный список записей;
* если возникла ошибка, показывается сообщение об ошибке.

1. Дополнительные действия:

* просмотр статистики записей;
* управление публикацией записей в блоге;
* работа с тегами и категориями;
* поиск и фильтрация записей.

Система поддерживает следующие основные варианты использования:

* управление записями дневника (создание, просмотр, удаление);
* работа с блогом (публикация, управление тегами);
* управление профилем и настройками;
* работа с медиа-контентом (изображения).

Взаимодействие компонентов системы происходит через четко определенные интерфейсы:

* DiaryScreen ↔ DiaryViewModel: управление отображением;
* DiaryViewModel ↔ Repository: бизнес-логика;
* Repository ↔ DiaryDao: доступ к данным;
* DiaryDao ↔ Room Database: хранение информации.

[1] Структура взаимодействия основана на принципах Clean Architecture и MVVM паттерне, что обеспечивает надежность и масштабируемость приложения.

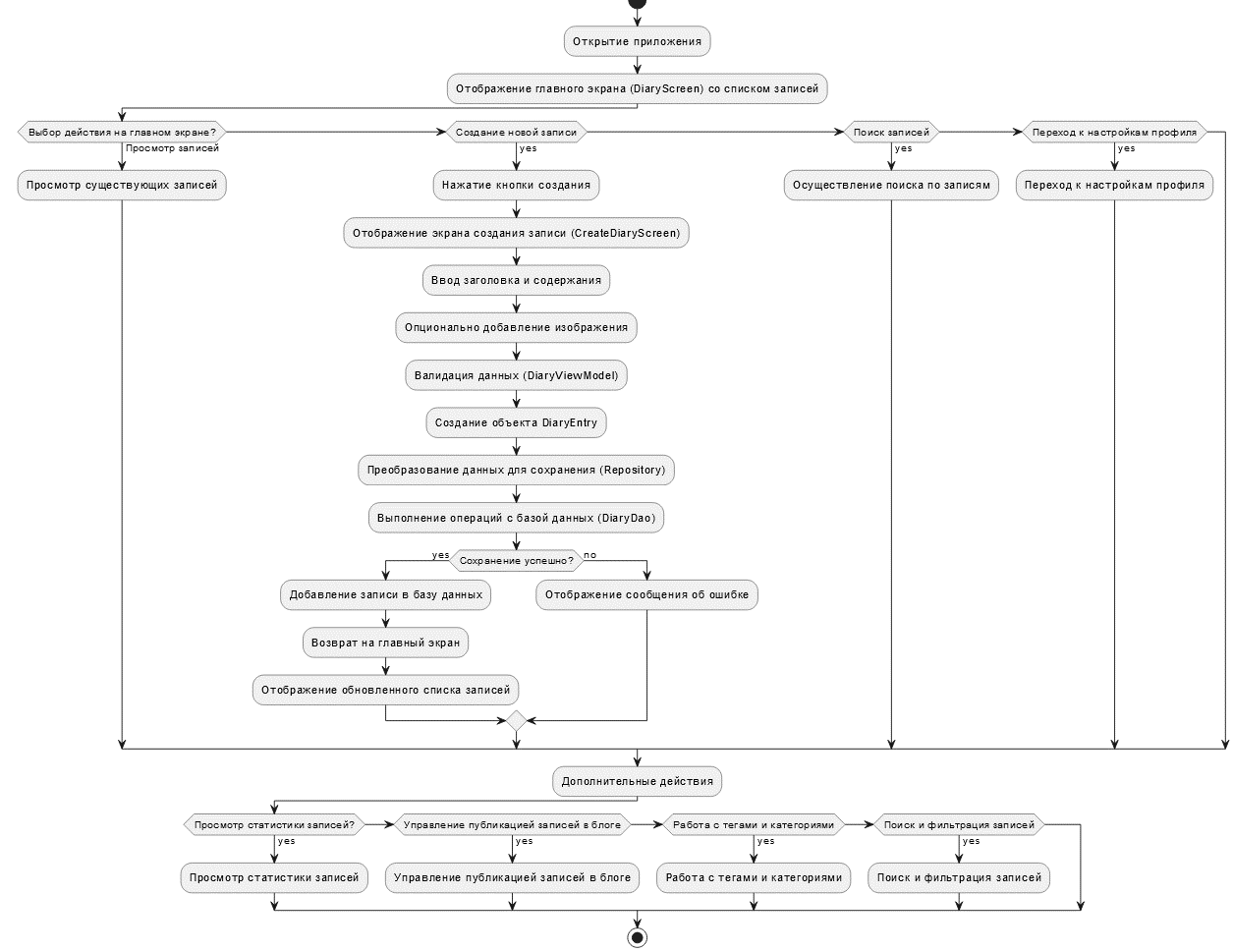


Рисунок - Диаграмма деятельности

Диаграмма состояний (Рисунок - 9) моделирует поведение приложения DayBlog в терминах различных состояний системы и переходов между ними. Она демонстрирует, как пользователь взаимодействует с приложением и как система реагирует на его действия [1].

Основные компоненты диаграммы:

1. Состояния:

* Ожидание действия - Приложение ожидает действий пользователя, например, создания новой записи или просмотра существующих;
* Создание записи - Состояние, в котором пользователь создает новую запись в дневнике;
* Просмотр записей - Состояние, в котором пользователь просматривает список записей;
* Настройки - Состояние, в котором пользователь может управлять настройками приложения;
* Выход - Состояние, в котором пользователь завершает работу с приложением.

1. Переходы:

* Начать создание - Пользователь нажимает кнопку создания новой записи, что приводит к переходу в состояние "Создание записи";
* Сохранить запись - Пользователь сохраняет созданную запись, что возвращает систему в состояние "Ожидание действия";
* Просмотреть записи - Пользователь выбирает просмотр записей, что приводит к переходу в состояние "Просмотр записей";
* Управление настройками - Пользователь изменяет настройки приложения, что приводит к переходу в состояние "Настройки";
* Завершить работу - Пользователь завершает работу с приложением, что приводит к переходу в состояние "Выход".

1. Начальное и конечное состояния:

* Начальное состояние - Приложение запущено и ожидает выбора пользователя;
* Конечное состояние - Пользователь завершает работу с приложением.

Переход между состояниями:

* Запуск приложения → Ожидание действия - Приложение загружено, и пользователь видит главный экран (DiaryScreen);
* Ожидание действия → Начать создание - Пользователь нажимает кнопку "+";
* Начать создание → Создание записи - Система отображает экран создания записи (CreateDiaryScreen);
* Создание записи → Сохранить запись - Пользователь нажимает "Сохранить";
* Сохранить запись → Ожидание действия - Система возвращается на главный экран;
* Ожидание действия → Просмотреть записи - Пользователь выбирает просмотр записей;
* Просмотреть записи → Ожидание действия - Пользователь возвращается на главный экран;
* Ожидание действия → Управление настройками - Пользователь выбирает настройки;
* Управление настройками → Ожидание действия - Пользователь возвращается на главный экран;
* Ожидание действия → Завершить работу - Пользователь выбирает выход из приложения.

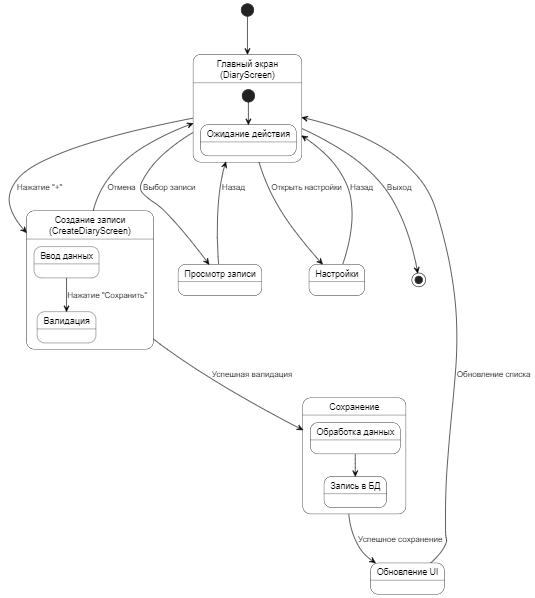


Рисунок - Диаграмма состояний

3.2 Руководство программиста

Диаграмма классов (Рисунок - 10) иллюстрирует архитектуру классов приложения DayBlog, включая основные компоненты, такие как модели данных, репозитории, слои доступа к данным и управление пользовательским интерфейсом. Она показывает взаимосвязи между классами и их функциональность.Основные классы и их описание:

1. DiaryEntry (Запись в дневнике):

Атрибуты:

* id: Int – Уникальный идентификатор записи;
* title: String – Заголовок записи;
* content: String – Содержимое записи;
* dateTime: LocalDateTime – Дата и время создания записи;
* imageUri: String? – URI изображения, прикрепленного к записи (опционально).

Методы:

* isValid(): Boolean – Проверяет, валидна ли запись;
* validate(): List<String> – Возвращает список ошибок валидации.

1. BlogEntry (Запись в блоге):

Атрибуты:

* id: Int – Уникальный идентификатор записи;
* title: String – Заголовок записи;
* content: String – Содержимое записи;
* dateTime: LocalDateTime – Дата и время создания записи;
* imageUri: String? – URI изображения, прикрепленного к записи (опционально);
* tags: List<String> – Список тегов, связанных с записью;
* isPublished: Boolean – Флаг публикации записи.

Методы:

* isValid(): Boolean – Проверяет, валидна ли запись;
* validate(): List<String> – Возвращает список ошибок валидации.

1. DiaryViewModel (Модель представления дневника):

Атрибуты:

* repository: DiaryRepository – Репозиторий для доступа к данным дневника;
* uiState: StateFlow<DiaryUiState> – Поток состояния пользовательского интерфейса.

Методы:

* createEntry(title: String, content: String, imageUri: String?) – Создает новую запись;
* deleteEntry(id: Int) – Удаляет запись по идентификатору;
* searchEntries(query: String) – Ищет записи по запросу.

1. RoomDiaryRepository (Репозиторий дневника на основе Room):

Атрибуты:

* diaryDao: DiaryDao – Объект доступа к данным для записей дневника.

Методы:

* getAllEntries(): Flow<List<DiaryEntry>> – Получает все записи дневника;
* insertEntry(entry: DiaryEntry): Result<Long> – Вставляет новую запись;
* deleteEntry(id: Int): Result<Int> – Удаляет запись по идентификатору.

1. DiaryDao (Объект доступа к данным дневника):

Методы:

* getAllEntries(): Flow<List<DiaryEntryEntity>> – Получает все записи из базы данных;
* insertEntry(entry: DiaryEntryEntity): Long – Вставляет новую запись в базу данных;
* deleteEntry(id: Int): Int – Удаляет запись по идентификатору;
* searchEntries(query: String): Flow<List<DiaryEntryEntity>> – Ищет записи по запросу.

1. AppDatabase (База данных приложения):

Методы:

* diaryDao(): DiaryDao – Предоставляет объект доступа к данным дневника;
* blogDao(): BlogDao – Предоставляет объект доступа к данным блога;
* getDatabase(context: Context): AppDatabase – Получает экземпляр базы данных.

1. DiaryScreen (Экран дневника):

Атрибуты:

* viewModel: DiaryViewModel – Модель представления для управления данными.

Методы:

* DiaryScreen(viewModel: DiaryViewModel, onNavigateToDetail: (Int) -> Unit) – Отображает список записей и кнопку создания.

CreateDiaryScreen (Экран создания записи):

Атрибуты:

* viewModel: DiaryViewModel – Модель представления для управления данными;
* title: String – Заголовок новой записи;
* content: String – Содержимое новой записи;
* imageUri: Uri? – URI выбранного изображения.

Методы:

* saveEntry() – Сохраняет новую запись.

Взаимосвязи между классами:

* DiaryScreen взаимодействует с DiaryViewModel для отображения и управления записями дневника;
* CreateDiaryScreen использует DiaryViewModel для создания новых записей;
* DiaryViewModel обращается к RoomDiaryRepository для выполнения операций с данными;
* RoomDiaryRepository использует DiaryDao для доступа к локальной базе данных;
* DiaryDao взаимодействует с AppDatabase для выполнения SQL-операций;
* DiaryEntryEntity преобразуется в DiaryEntry и обратно при операциях с базой данных;
* BlogEntryEntity преобразуется в BlogEntry и обратно при операциях с базой данных.

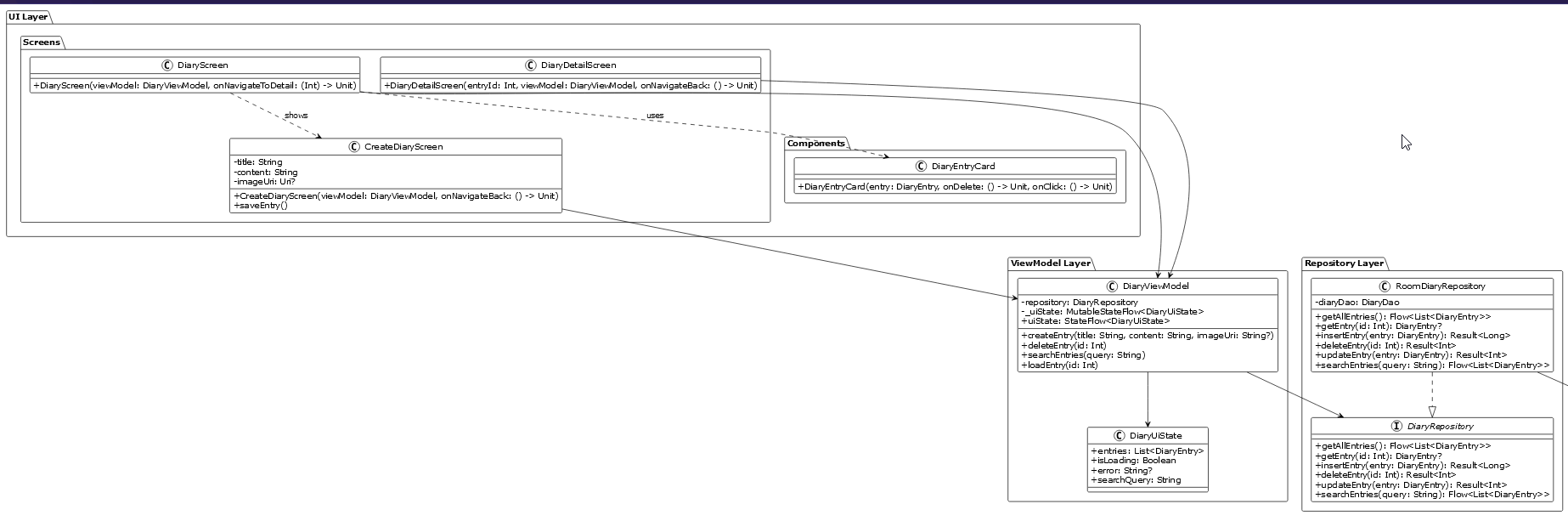


Рисунок - Диаграмма классов

# Заключение

Курсовой проект посвящен разработке мобильного приложения для ведения дневника и блога на платформе Android. Основные компоненты системы включают:

1. Архитектура MVVM: В проекте реализована архитектура Model-View-ViewModel, обеспечивающая разделение ответственности между компонентами и повышающая тестируемость кода.
2. Пользовательский интерфейс: Современный декларативный пользовательский интерфейс разработан с использованием Jetpack Compose, что обеспечивает гибкость и возможность создания анимированных компонентов.
3. Локальное хранилище данных: Система хранения данных реализована с использованием библиотеки Room, обеспечивающей типобезопасный доступ к локальной базе данных SQLite.
4. Функциональность дневника: Позволяет пользователям создавать, просматривать и удалять личные записи с возможностью добавления изображений и поиска по содержимому.
5. Управление блогом: Система обеспечивает функциональность для создания публикуемых записей с поддержкой тегов, управления статусом публикации и категоризации контента.
6. Навигация между экранами: Реализована с помощью Navigation Compose, обеспечивающего удобное перемещение между различными частями приложения.

В результате курсового проекта было создано полнофункциональное мобильное приложение, которое позволяет эффективно вести личный дневник и публичный блог на устройствах Android, обеспечивая интуитивно понятный интерфейс и широкие возможности для пользователей.

# Список использованной литературы

1. https://developer.android.com/jetpack/compose - Официальная документация Jetpack Compose [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
2. https://developer.android.com/topic/libraries/architecture/room - Документация по Room [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
3. https://developer.android.com/topic/libraries/architecture/viewmodel - Руководство по использованию ViewModel [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
4. https://developer.android.com/kotlin/coroutines - Документация по корутинам Kotlin [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
5. https://developer.android.com/guide/navigation/navigation-getting-started - Руководство по навигации в Android приложениях [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
6. https://kotlinlang.org/docs/flow.html - Документация по Kotlin Flow [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
7. https://material.io/design - Material Design Guidelines [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
8. https://developer.android.com/training/data-storage/room/relationships - Room Database Relationships [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
9. https://developer.android.com/jetpack/compose/state - Managing State in Compose [Электронный ресурс], 2025 (Дата обращения 4.03.2025).
10. https://coil-kt.github.io/coil/ - Документация библиотеки Coil [Электронный ресурс], 2025 (Дата обращения 4.03.2025).

Приложение

*// DiaryEntry.kt - Data model for Diary entries*

package com.den5.dayblog.data.model

import java.time.LocalDateTime

data class DiaryEntry(

    val id: Int = 0,

    val title: String,

    val content: String,

    val dateTime: LocalDateTime = LocalDateTime.now(),

    val imageUri: String? = null

) {

    companion object {

        const val MAX\_TITLE\_LENGTH = 100

        const val MIN\_TITLE\_LENGTH = 3

        const val MAX\_CONTENT\_LENGTH = 10000

        const val MIN\_CONTENT\_LENGTH = 10

    }

    fun isValid(): Boolean {

        return title.length in MIN\_TITLE\_LENGTH..MAX\_TITLE\_LENGTH &&

                content.length in MIN\_CONTENT\_LENGTH..MAX\_CONTENT\_LENGTH

    }

    fun validate(): List<String> {

        val errors = mutableListOf<String>()

        if (title.length < MIN\_TITLE\_LENGTH) {

            errors.add("Title must be at least $MIN\_TITLE\_LENGTH characters")

        }

        if (title.length > MAX\_TITLE\_LENGTH) {

            errors.add("Title cannot exceed $MAX\_TITLE\_LENGTH characters")

        }

        if (content.length < MIN\_CONTENT\_LENGTH) {

            errors.add("Content must be at least $MIN\_CONTENT\_LENGTH characters")

        }

        if (content.length > MAX\_CONTENT\_LENGTH) {

            errors.add("Content cannot exceed $MAX\_CONTENT\_LENGTH characters")

        }

        return errors

    }

}

*// DiaryRepository.kt - Repository interface for Diary operations*

package com.den5.dayblog.data.repository

import com.den5.dayblog.data.model.DiaryEntry

import kotlinx.coroutines.flow.Flow

import java.time.LocalDateTime

interface DiaryRepository {

    fun getAllEntries(): Flow<List<DiaryEntry>>

    suspend fun getEntry(id: Int): DiaryEntry?

    suspend fun insertEntry(entry: DiaryEntry): Result<Long>

    suspend fun deleteEntry(id: Int): Result<Int>

    suspend fun updateEntry(entry: DiaryEntry): Result<Int>

    fun searchEntries(query: String): Flow<List<DiaryEntry>>

    fun getEntriesByDateRange(startDate: LocalDateTime, endDate: LocalDateTime): Flow<List<DiaryEntry>>

    fun getEntryCount(): Flow<Int>

    fun getEntriesWithImages(): Flow<List<DiaryEntry>>

}

*// RoomDiaryRepository.kt - Implementation of DiaryRepository using Room*

package com.den5.dayblog.data.repository

import com.den5.dayblog.data.local.DiaryDao

import com.den5.dayblog.data.local.DiaryEntryEntity

import com.den5.dayblog.data.model.DiaryEntry

import kotlinx.coroutines.flow.Flow

import kotlinx.coroutines.flow.catch

import kotlinx.coroutines.flow.map

import java.util.NoSuchElementException

class RoomDiaryRepository(

    private val diaryDao: DiaryDao

) : DiaryRepository {

    override fun getAllEntries(): Flow<List<DiaryEntry>> {

        return diaryDao.getAllEntries()

            .map { entities -> entities.map { it.toDiaryEntry() } }

            .catch { e ->

                emit(emptyList())