МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка Android-приложения для сервиса PhotoPrism на Kotlin

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-362.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ Н.С. Силкина

Автор работы,
студент группы КЭ-404
_____ А.А. Машкин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-404
Машкину Анатолию Андреевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка Android-приложения для сервиса PhotoPrism на Kotlin.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. PhotoPrism. AI-Powered Photos App for the Decentralized Web. [Электронный ресурс] URL: https://github.com/photoprism/photoprism/ (дата обращения: 01.02.2024 г.).

3.2. Material Design. [Электронный ресурс] URL: https://m3.material.io/ (дата обращения: 01.02.2024 г.).

3.3. Android Developers. [Электронный ресурс] URL: https://developer.android.com/ (дата обращения: 01.02.2024 г.).

3.4. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – Санкт-Петербург: Питер, 2018. – 352 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.

4.2. Спроектировать мобильное приложение.

4.3. Реализовать мобильное приложение.

4.4. Провести тестирование разработанного приложения.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП Н.С. Силкина

Задание принял к исполнению А.А. Машкин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Мобильные приложения стали неотъемлемой частью нашей жизни, и их популярность только увеличивается с каждым годом. Они предоставляют пользователю широкий спектр возможностей, позволяя получать информацию, общаться, развлекаться, работать, покупать товары и услуги, делать фотографии и многое другое. Современный мир насыщен информацией, и пользователи все больше ориентируются на мобильные приложения, так как они обеспечивают более удобный и быстрый доступ к нужным данным, чем традиционные веб-сайты или десктопные приложения.

Одной из областей, в которых мобильные приложения особенно удобны, является просмотр фотографий. Фотографии давно перестали быть просто средством хранения воспоминаний, они стали средством общения и самовыражения. Пользователи хотят иметь быстрый и удобный доступ к своим фотографиям, а также возможность легко и быстро делиться ими с друзьями и близкими.

Мобильные приложения для просмотра фотографий решают эту проблему, предоставляя пользователю множество функций и возможностей. Они позволяют загружать фотографии на сервер, создавать альбомы, редактировать фотографии, просматривать их в высоком разрешении, а также делиться ими в социальных сетях и мессенджерах.

Таким образом, мобильные приложения для просмотра фотографий являются актуальными и необходимыми в настоящее время, так как они удовлетворяют потребности пользователей в быстром и удобном доступе к своим фотографиям, а также легко делиться ими с другими людьми. Кроме того, они обеспечивают удобное и безопасное хранение, что особенно важно для людей, которые хранят много ценных и важных фотографий.

Постановка задачи

Целью выпускной квалификационной работы является разработка Android-приложения для сервиса PhotoPrism [1] на Kotlin. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области;
2) спроектировать мобильное приложение;
3) реализовать мобильное приложение;
4) провести тестирование разработанного приложения.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 69 страниц, объем списка литературы – 22 источника.

В первой главе описывается предметная область разрабатываемого приложения, сравниваются существующие его аналоги, а также производится обзор и обоснование используемых технологий.

Вторая глава посвящена проектированию мобильного приложения – в ней описываются требования к системе, выбор архитектуры и проектирование базы данных.

В третьей главе приводится описание ключевых идей и компонентов, которые были применены и созданы соответственно во время реализации мобильного приложения.

В четвертой главе описывается проверка разработанного приложения на соответствие требованиям.

В приложении А содержится спецификация вариантов использования разрабатываемого мобильного приложения.

В приложении Б содержится диаграмма деятельности формы авторизации.

В приложении В содержится пример юнит-теста для класса, который группирует по дате элементы в галерее.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

PhotoPrism [1] – это приложение для организации централизованной домашней медиа-библиотеки с использованием собственных мощностей пользователя. Скриншот главного экрана веб-приложения представлен на рисунке 1. Под домашней медиа-библиотекой подразумевается хранимые на диске личные фотографии и видео пользователя.

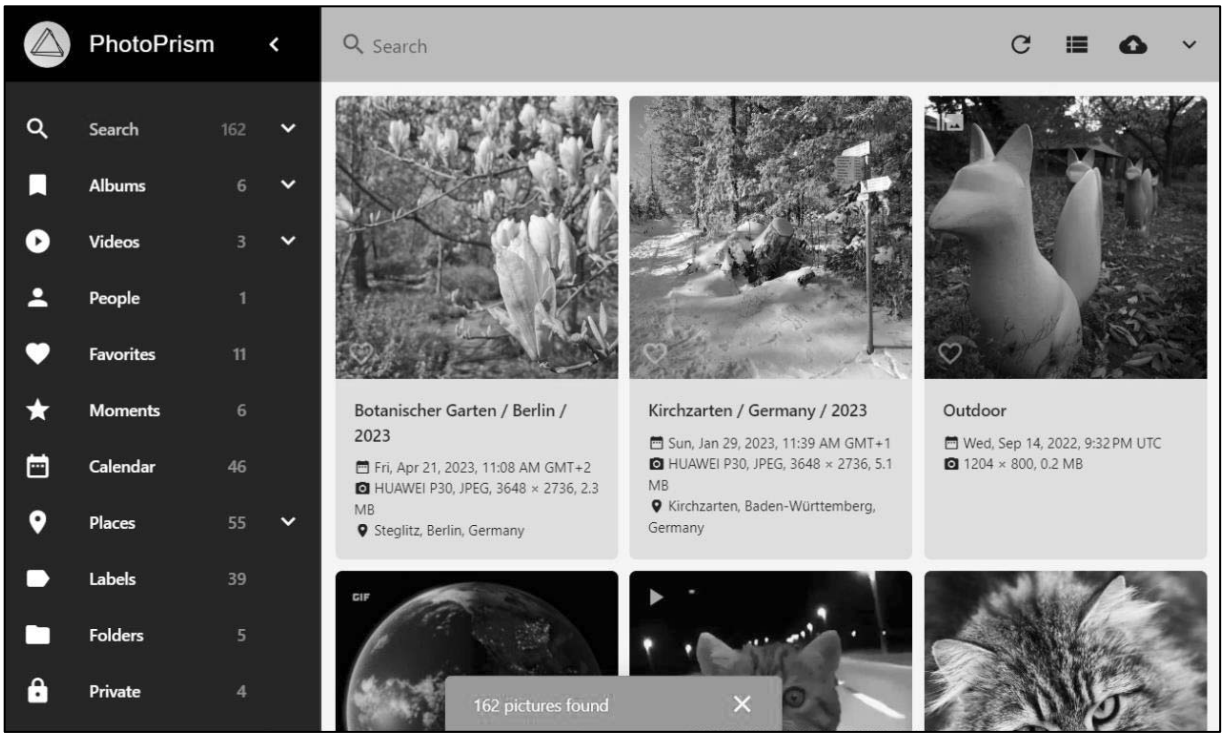

Рисунок 1 – Веб-интерфейс приложения PhotoPrism

PhotoPrism обладает большим множеством возможностей, в список которых входит:

1) просмотр всех фотографий и видео пользователя в различных форматах с автоматической конвертацией RAW файлов [2] и группировкой дубликатов;

2) поиск определенных медиафайлов с использованием гибких поисковых фильтров и ключевых слов;

3) автоматическое распознавание лиц пользователей сервиса на фотографиях;

4) автоматическая классификация фотографий на основании их содержания и метаданных;

5) просмотр медиафайлов (при наличии соответствующих метаданных) на интерактивной карте мира;

6) поддержка сетевых протоколов WebDAV и SMB [3].

Сервис распространяется в виде образа Docker [4], который может быть установлен на любую операционную систему, поддерживающую данную платформу виртуализации. В число таких операционных систем входят Windows, Linux и macOS.

Для успешной работы PhotoPrism система пользователя должна удовлетворять следующим требованиям:

1) не менее двух ядер процессора;

2) не менее 3 ГБ оперативной памяти;

3) файл подкачки объемом 4 ГБ;

4) 64-разрядная операционная система.

PhotoPrism разрабатывается компанией «PhotoPrism» (Германия) и имеет открытый исходный код, распространяемый под лицензией GNU General Public License v3.0, что позволяет свободно исследовать исходный код приложения и вносить любые изменения при необходимости.

Большинство функций PhotoPrism предоставляются бесплатно, но некоторые распространяются по подписочной модели. Так, например, подписка PhotoPrism Plus стоит 6 евро в месяц и дает доступ к таким функциям, как: управление ролями пользователей (в базовой версии доступны только две роли – admin и superadmin), просмотр фотографий в привязке к 3D картам и снимкам со спутника (в базовой версии только векторные карты) и расширенные настройки безопасности. Остальные функции бесплатны.

PhotoPrism построен на клиент-серверной архитектуре с применением протокола REST. Благодаря этому возможна разработка клиентов для данного сервиса сторонними разработчиками.

1.2. Сравнительный анализ аналогов

В ходе проведенного анализа было обнаружено несколько приложений, совестимых с PhotoPrism. Далее рассматриваются основные их достоинства и недостатки.

Приложение Flutter App for PhotoPrism [5] было разработано сообществом энтузиастов, однако репозиторий с исходным кодом проекта был переведен в режим «только чтение» 28.06.2023 г. На рисунке 2 представлен скриншот работы последней версии приложения – как можно видеть, галерея отображается некорректно.

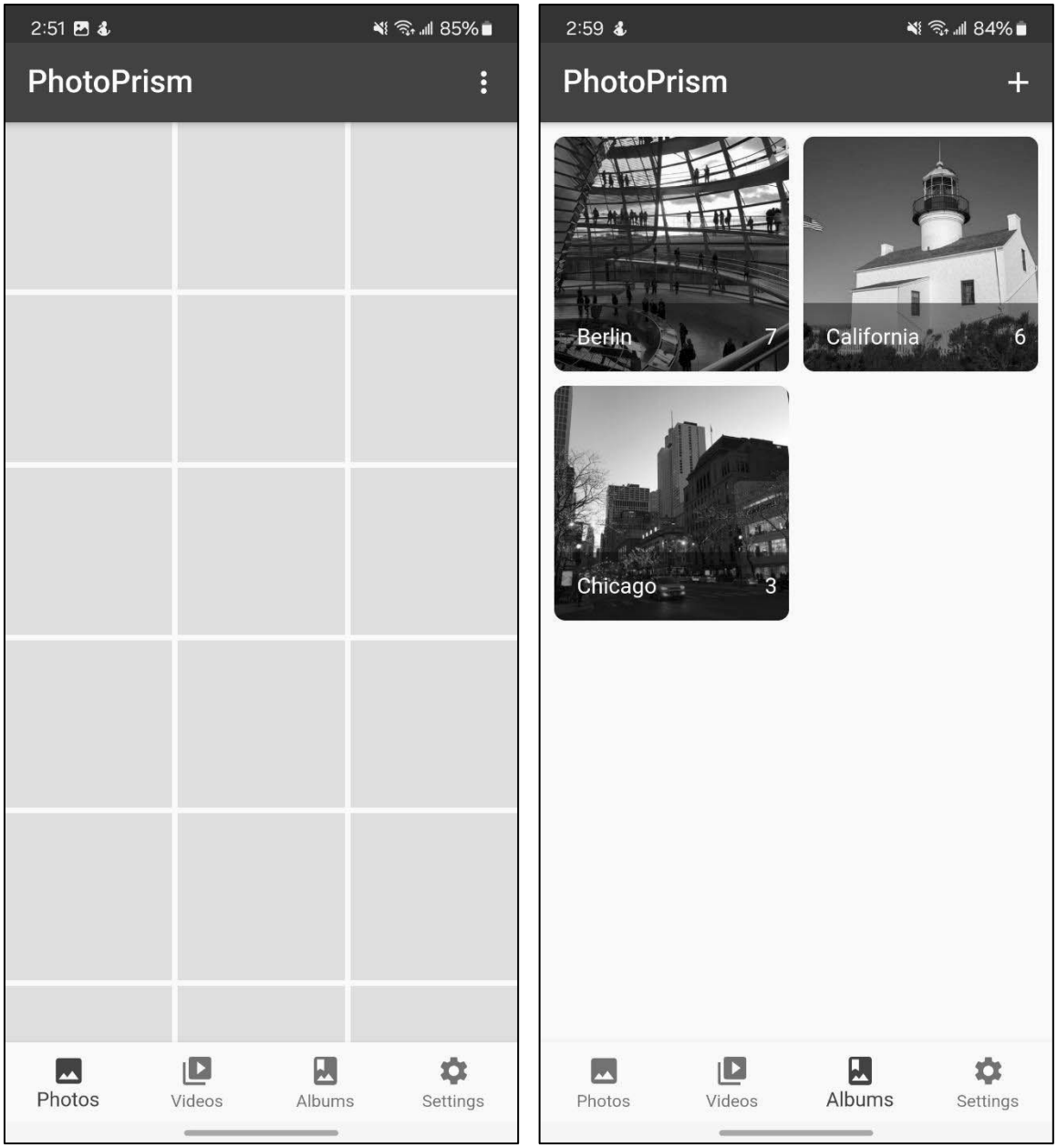

Рисунок 2 – Мобильное приложение Flutter App for PhotoPrism

Приложение MobilePrism [6] разрабатывается Адрианом Гермеком, а последняя версия была выпущена 26.10.2022 г. Скриншот мобильного приложения для Android представлен на рисунке 3.

Основные достоинства приложения MobilePrism:

1) позволяет просматривать полный перечень медиафайлов и содержимое альбомов;

2) предусмотрено кэширование медиафайлов, что позволяет просматривать медиа-библиотеку без доступа к интернету.

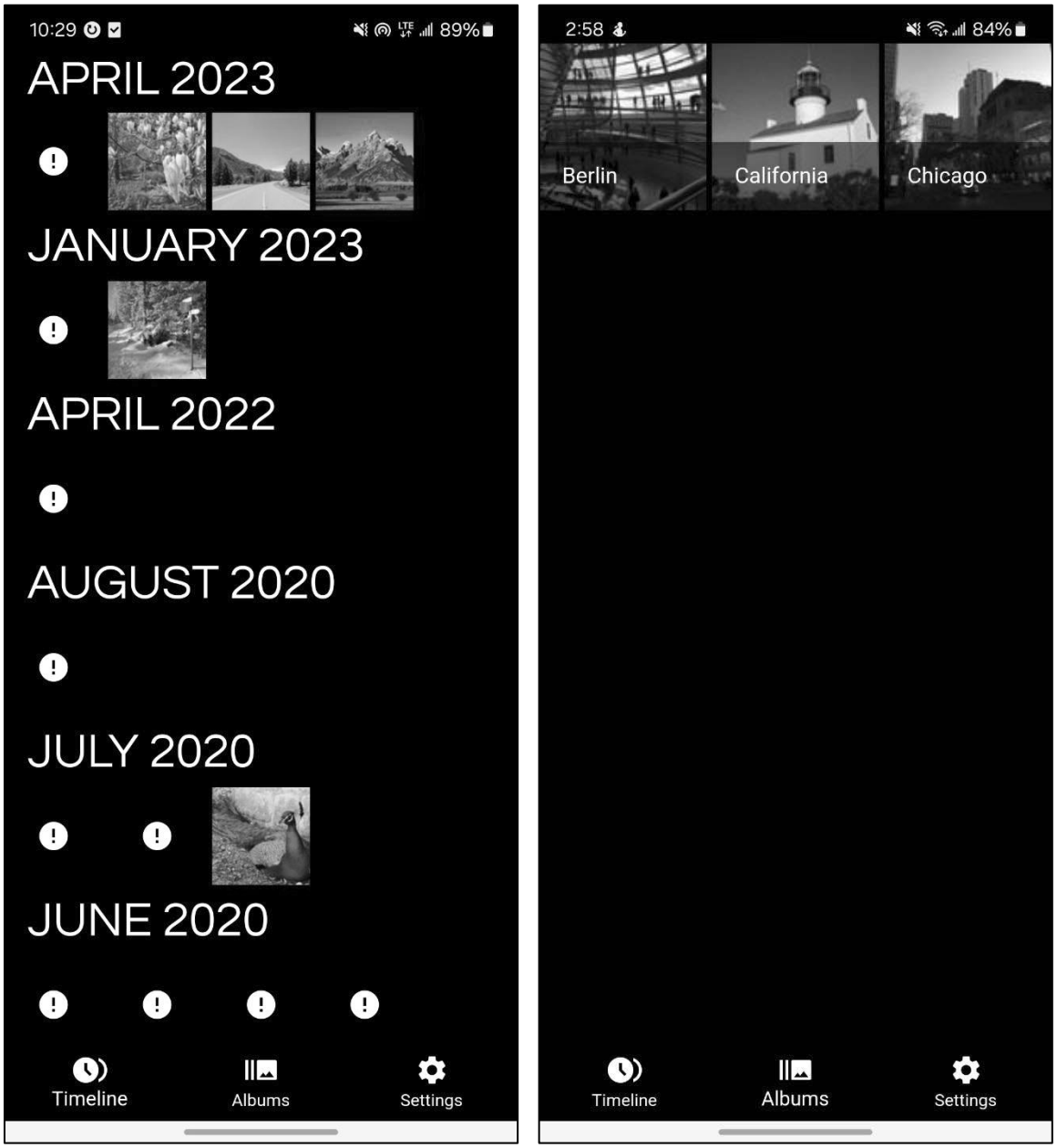

Рисунок 3 – Мобильное приложение MobilePrism

Основные недостатки приложения MobilePrism:

1) не поддерживает просмотр видео;
2) нет поиска с использованием фильтров и ключевых слов;
3) не позволяет «поделиться» медиафайлом с другими приложениями;
4) нет автоматического продления сессии пользователя.

Приложение PhotoPrism Gallery [7] разрабатывается Олегом Корецким. Скриншот мобильного приложения для Android представлен на рисунке 4.

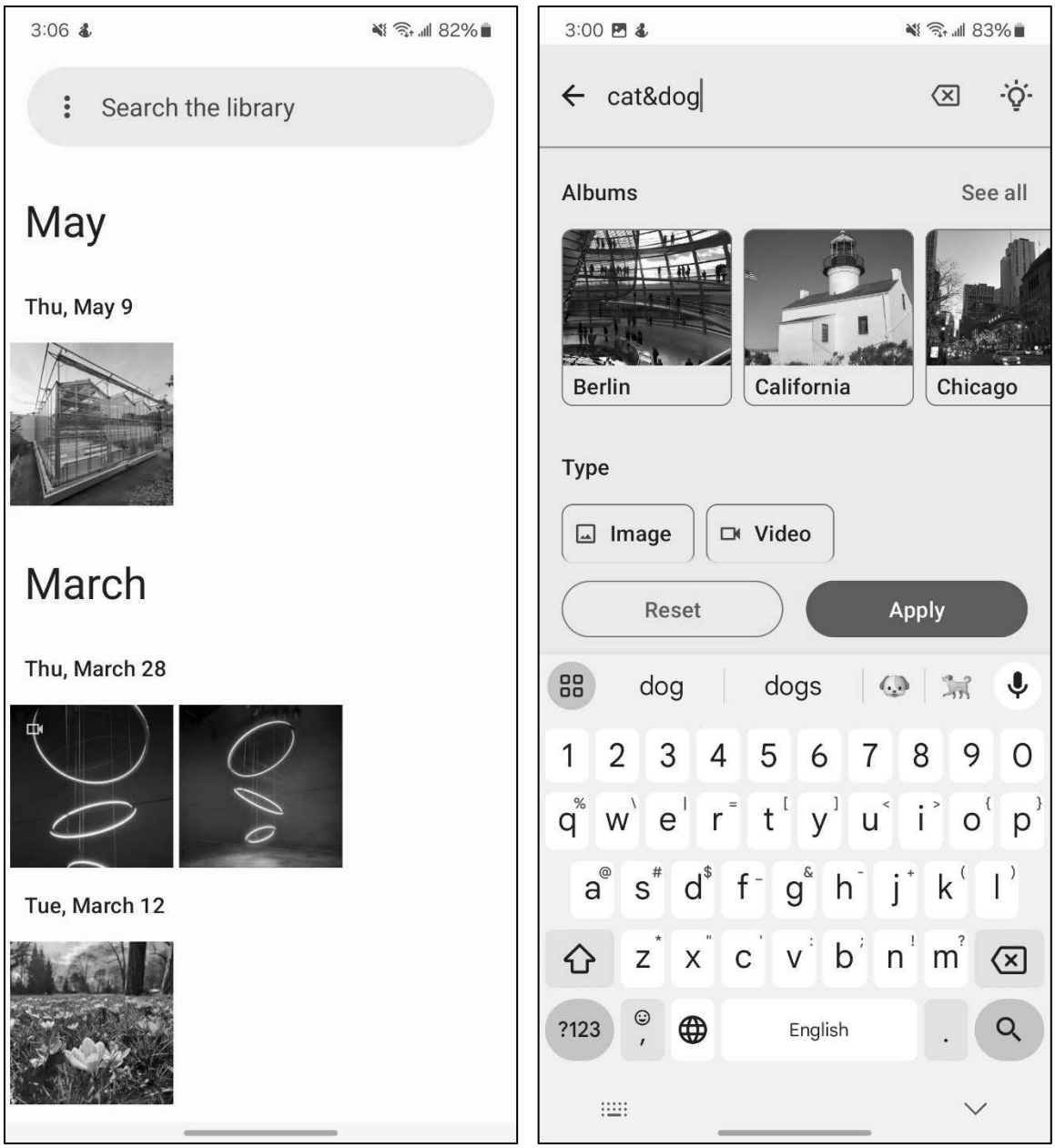

Рисунок 4 – Мобильное приложение PhotoPrism Gallery

Основные достоинства приложения PhotoPrism Gallery:

1) позволяет просматривать полный перечень медиафайлов и содержимое альбомов;

2) интегрируется с системой, что позволяет получить доступ к медиа-библиотеке из других приложений;

3) есть возможность производить поиск с использованием фильтров и ключевых слов;

4) есть автоматическое продление сессии пользователя;

5) поддерживает просмотр видео.

Основным недостатком приложения PhotoPrism Gallery является то, что в нем не предусмотрено кэширование списка медиафайлов поэтому оно не работает без доступа к интернету или в условиях недоступности сервера.

Анализ аналогичных проектов показал, что на данный момент существующие приложения не обладают всеми нужными функциями (таблица 1), необходимыми для удобного доступа к домашней медиа-библиотеке в различных условиях. Из чего можно сделать вывод о том, что разрабатываемое приложение должно, как минимум, поддерживать просмотр всех медиафайлов в библиотеке, перечня всех альбомов и их содержимого с возможностью воспроизведения видео, а также уметь производить гибкий поиск разных элементов. Помимо этого, приложение должно автоматически продлять сессию пользователя и поддерживать работу в оффлайн режиме.

Таблица 1 – Сравнение функционала рассмотренных приложений

Критерий сравнения	**Flutter App for PhotoPrism**	**MobilePrism**	**PhotoPrism Gallery**	**Разрабатываемое приложение**
Просмотр полного перечня медиафайлов в медиа-библиотеке	+	+	+	+
Просмотр содержимого альбомов	+	–	+	+
Кэширование медиафайлов и работа в оффлайн режиме	+	+	–	+

Окончание таблицы 1

Критерий сравнения	Flutter App for PhotoPrism	MobilePrism	PhotoPrism Gallery	Разрабатываемое приложение
Поиск медиафайлов с использованием фильтров и ключевых слов	–	–	+	+
Поддержка воспроизведения видео	–	–	+	+
Автоматическое продление сессии пользователя	–	–	+	+

1.3. Анализ технологий для реализации проекта

В настоящее время существуют несколько решений для разработки Android приложений.

Первым вариантом является использование нативных технологий для платформы Android, таких как язык программирования Kotlin [8] и Java API самой операционной системы [9], а также готовых архитектурных компонентов из официально поддерживаемого и рекомендуемого набора Android Jetpack [10], в число которых входят следующие библиотеки: Room для работы с базой данных SQLite, Paging 3 для создания эффективных прокручиваемых списков, Navigation Component для разработки навигации между экранами приложения, ViewModel для построения пользовательского интерфейса с применением архитектуры MVVM и т.д. Таким образом, данный подход позволяет не только достичь наибольшего уровня интеграции приложения с системными сервисами, но и благодаря готовым компонентам сделать приложение более надежным, оптимизированным и поддерживаемым в будущем.

Вторым вариантом может быть использование инструментов для создания кроссплатформенных мобильных приложений. Среди них находятся Flutter [11] и React Native [12], которые, согласно результатам опроса на портале для разработчиков StackOverflow в 2022 году [13], являются двумя наиболее популярными фреймворками для разработки мобильных

приложений с использованием Dart [14] и JavaScript соответственно. Основным преимуществом обеих технологий является единая кодовая база для нескольких платформ, включающих Android, iOS и Web. При этом Flutter обеспечивает хорошую производительность, но может столкнуться с проблемами интеграции с системными сервисами. React Native, с другой стороны, обеспечивает нативный интерфейс и широкую поддержку библиотек и инструментов, но может иметь проблемы с производительностью на сложных приложениях. Таким образом, при выборе данного подхода для разработки Android приложений следует учитывать конкретные потребности проекта и оценить преимущества и недостатки каждого фреймворка.

На основании всего вышесказанного можно сделать вывод о том, что нативная разработка обеспечивает более глубокую интеграцию с операционной системой Android по сравнению с кроссплатформенными фреймворками. Это обеспечивает более высокую производительность приложения, так как нативные компоненты могут взаимодействовать напрямую с операционной системой, минуя дополнительные абстракции, которые могут присутствовать в кроссплатформенных решениях. Поскольку разрабатываемое приложение будет загружать и обрабатывать больше количество графической информации, то было принято решение использовать первый вариант для разработки проекта, а именно – язык программирования Kotlin и Java API.

2. ПРОЕКТИРОВАНИЕ

2.1. Требования к системе

Функциональные требования [15] определяют, как приложение должно работать и какие функции оно должно предоставлять. Они описывают конкретные задачи и действия, которые пользователь должен быть способен выполнить в приложении.

В ходе анализа предметной области были выявлены следующие функциональные требования:

1) приложение должно поддерживать авторизацию в экземпляре PhotoPrism, расположенным на произвольном домене или доступным по IP адресу;

2) приложение должно предоставлять возможность отображения всех файлов в медиа-библиотеке в виде галереи с группировкой по дате;

3) приложение должно предоставлять возможность поиска файлов по всей медиа-библиотеке с помощью фильтров и ключевых слов;

4) приложение должно предоставлять возможность отображения списка альбомов;

5) приложение должно предоставлять возможность отображения всех файлов в альбоме в виде галереи;

6) приложение должно автоматически продлять сессию пользователя, если она более не действительна;

7) приложение должно иметь возможность просмотра файлов во весь экран с дополнительной информацией о них, в том числе поддерживать воспроизведение видео;

8) приложение должно уметь работать без наличия активного интернет-соединения или при недоступности экземпляра PhotoPrism.

Нефункциональные требования [15] определяют, как приложение должно работать в рамках своих функций. Они включают в себя условия, которые определяют ограничения на систему или ее компоненты, но не связаны с ее основной функциональностью.

В ходе анализа предметной области были определены следующие нефункциональные требования:

1) приложение должно быть написано с использованием языка программирования Kotlin;

2) приложение должно работать на устройствах с ОС Android версии 6 и выше;

3) приложение должно поддерживать обмен данными с сервером PhotoPrism через протокол HTTPS;

4) приложение должно иметь интерфейс, разработанный согласно дизайн системе Material 3;

5) приложение должно задействовать встроенные возможности ОС Android для защиты учетных данных пользователя.

На рисунке 5 представлена диаграмма вариантов использования [15], составленная на основе требований к разрабатываемой системе.

Были выявлены следующие актеры.

1. Неавторизованный пользователь. Он не имеет доступа к функциям приложения. Единственное доступный вариант использования – авторизация.

2. Авторизованный пользователь. Он имеет доступ ко всем функциям приложения, кроме авторизации.

Также были определены следующие варианты использования, спецификация которых приведена в приложении А.

1. Авторизация. Позволяет пользователю авторизоваться в системе.

2. Поиск медиафайлов с использованием фильтров. Позволяет отображать на экране только медиафайлы, удовлетворяющие заданным параметрам поиска.

3. Просмотр списка альбомов. В результате данного действия на экране пользователя отображается список раннее созданных им альбомов.

4. Поиск альбомов с использованием фильтров. В результате данного действия на экране пользователя отображается список альбомов, удовлетворяющих критериям поиска.

5. Просмотр содержимого альбома. В результате данного действия на экране пользователя отображаются медиафайлы, которые были ранее добавлены в просматриваемый альбом.

6. Просмотр полного перечня медиафайлов. В результате данного действия на экране пользователя отображаются все медиафайлы, которые присутствуют в его медиа-библиотеке.

7. Просмотр медиафайла в деталях. В результате данного действия выбранный медиафайл отображается на экране пользователя во весь размер, а также показывается дополнительная информация, такая как название, дата создания и т.д.

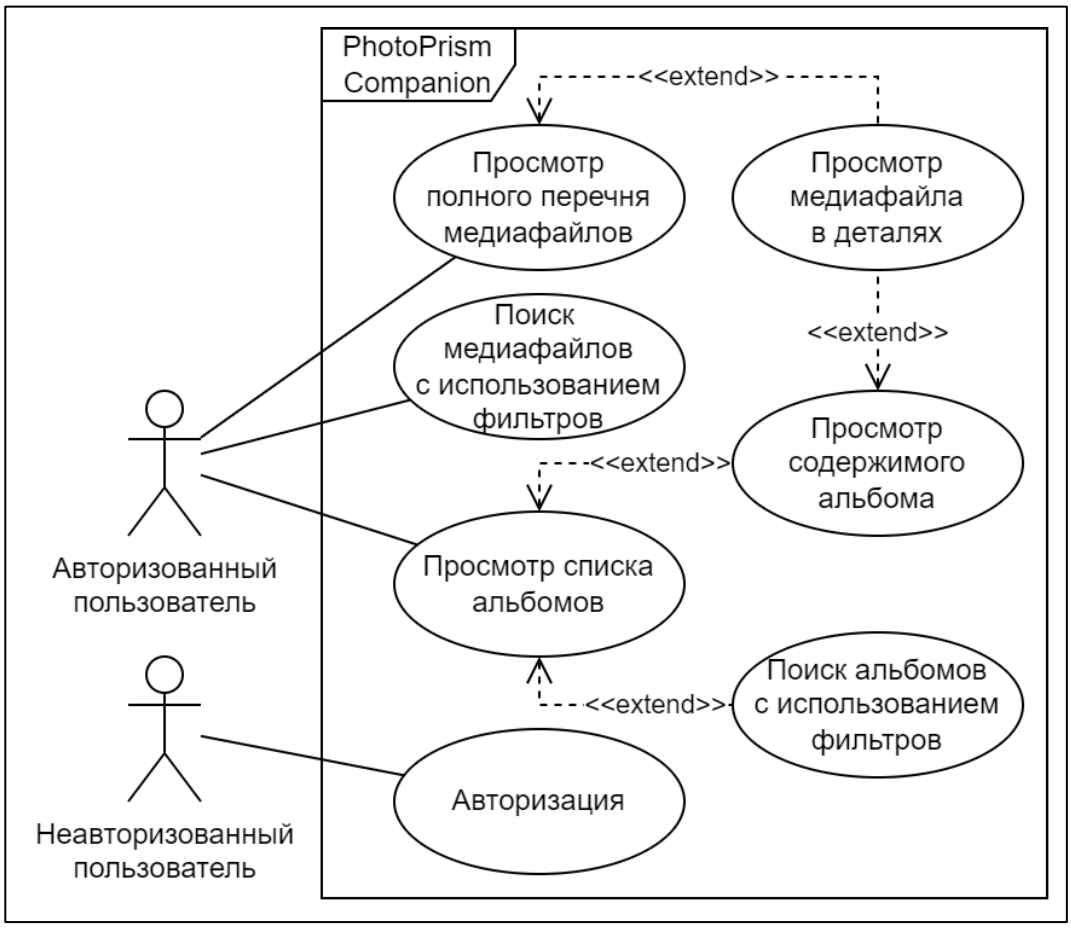

Рисунок 5 – Диаграмма вариантов использования

2.2. Архитектура приложения

За основу архитектуры разрабатываемого приложения взята концепция чистой архитектуры.

Чистая архитектура (Clean Architecture) [16] – это концепция проектирования программного обеспечения, которая ставит целью создать гибкую, расширяемую и тестируемую архитектуру, разделяющую бизнес-логику от инфраструктурных и технических деталей.

Использование чистой архитектуры позволяет упростить создание приложений, которые масштабируются и обслуживаются с минимальными изменениями в коде.

Основным принципом чистой архитектуры является разделение кода на независимые слои. Каждый слой решает конкретную задачу и зависит только от слоя, расположенного непосредственно под ним (схема зависимостей приведена на рисунке 6). Наиболее распространенной схемой, используемой в мобильной разработке, является трехслойная архитектура.

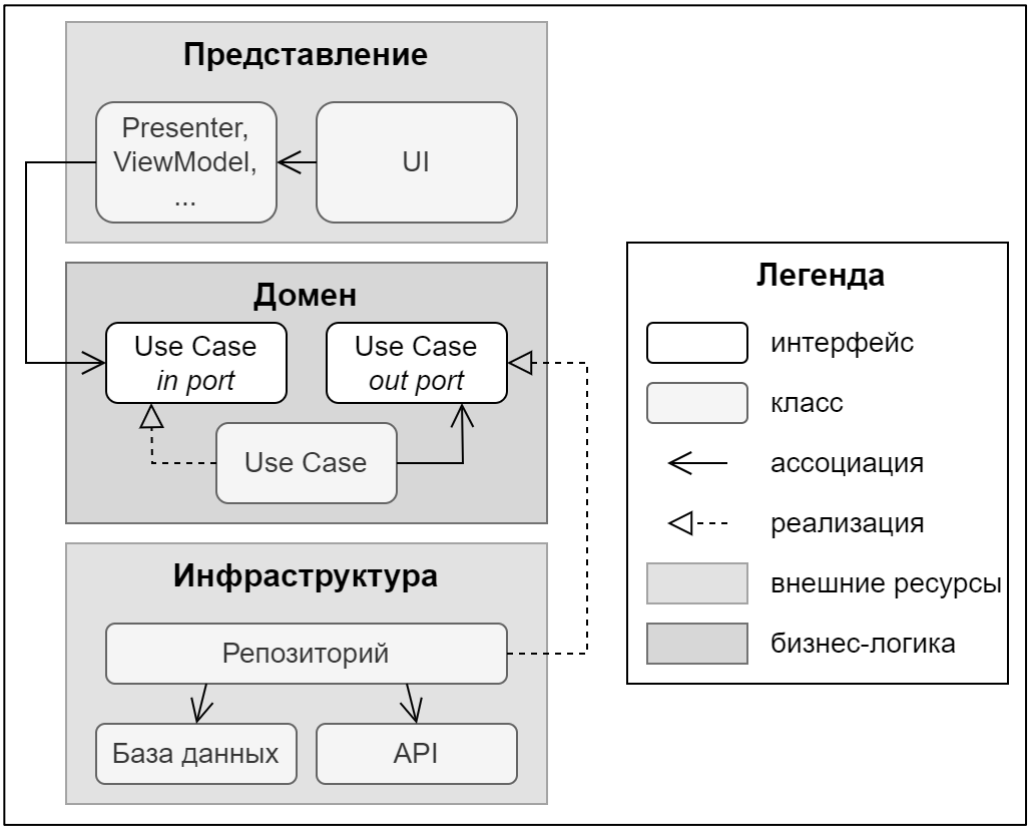

Рисунок 6 – Схема зависимостей в чистой архитектуре

Далее рассматривается каждый из трех слоев.

1. Представление отвечает за отображение пользовательского интерфейса. Он взаимодействует с пользователем, обрабатывает события, а также ввод и вывод данных. Этот слой не знает ничего о бизнес-логике, а только вызывает соответствующие методы интерфейса из слоя домена.

2. Доменный слой содержит бизнес-логику приложения. Он обрабатывает запросы из слоя представления и обращается к инфраструктурному слою через интерфейс. Доменный слой не знает ничего о том, как данные сохраняются или отображаются на экране, он просто выполняет свои задачи и возвращает результаты.

3. Инфраструктурный слой предоставляет реализацию доменных интерфейсов и решает вопросы, связанные с хранением данных, сетевыми запросами и т.д. Он не знает ничего о бизнес-логике, а только реализует интерфейсы доменного слоя и обеспечивает доступ к различным внешним системам и ресурсам.

Если при проектировании последних двух слоев сложностей не возникает, то слой представления можно организовать разными способами.

В первую очередь рассматривается официально рекомендуемый [17] для разработки Android приложений архитектурный паттерн MVVM (Model-View-ViewModel) [18]. Данный паттерн служит для более тщательного разделения ответственности между различными компонентами приложения и состоит из следующих элементов.

1. Модель (Model) представляет собой бизнес-логику и данные приложения. Она может включать запросы к базе данных, сетевые запросы и другие источники данных. В терминах чистой архитектуры состоит из доменного и инфраструктурного слоев.

2. Представление (View) отвечает за отображение пользовательского интерфейса и взаимодействие с пользователем. Оно получает данные для отображения от модели представления (ViewModel), при этом сообщая ей о действиях пользователях.

3. Модель представления (ViewModel) связывает представление и модель, обрабатывая действия пользователя и предоставляя данные для отображения. При этом ViewModel полностью не зависит от конкретной реализации представления, а передача данных для отображения зачастую осуществляется с использованием паттерна «наблюдатель».

К достоинствам данной модели можно отнести разделение ответственности слоя представления между дополнительными элементами, что способствует уменьшению связанности кода и упрощению дальнейшей поддержки. Также стоит отметить простоту понимания данного подхода, поскольку вся логика разделена, по сути, между представлением и его моделью.

К недостаткам данной модели можно отнести то, что она не регламентирует, как именно следует решать или минимизировать многие прикладные проблемы, возникающие в ходе проектирования и реализации слоя представления. К таким проблемам, в частности, относятся:

1) противоречивое состояние пользовательского интерфейса, возникающее вследствие ошибки программиста из-за большого количества точек выхода из модели представления;

2) сложность обработки смены конфигурации и восстановления процесса, когда система Android уничтожает все классы;

3) отсутствие разделения на зоны ответственности в рамках модели представления;

4) проблема «раздувания» модели представления по мере развития приложения.

Еще одним архитектурным паттерном, пришедшим из языка программирования Elm для разработки веб-приложений, является The Elm Architecture, TEA [19]. Идея подхода основана на том, чтобы разделить веб-

приложение на несколько ключевых элементов, описание которых приведено далее.

1. Модель (Model) представляет собой объект с состоянием приложения. Его особенность состоит в том, что он является «единственным источником истины» для представления, т.е. все ключевые для отображения данные содержатся в нем. При этом логики обработки данных в модели нет.

2. Представление (View) – это то, что пользователь видит и с чем взаимодействует. При этом модель является источником данных для отрисовки.

3. Обновление (Update) – это функция, которая принимает текущую модель и сообщение (например, действие пользователя), и возвращает новую модель.

4. Команда (Command). В Elm команды используются для выполнения асинхронных операций (например, отправки HTTP-запросов).

Схема организации этих компонентов приведена на рисунке 7.

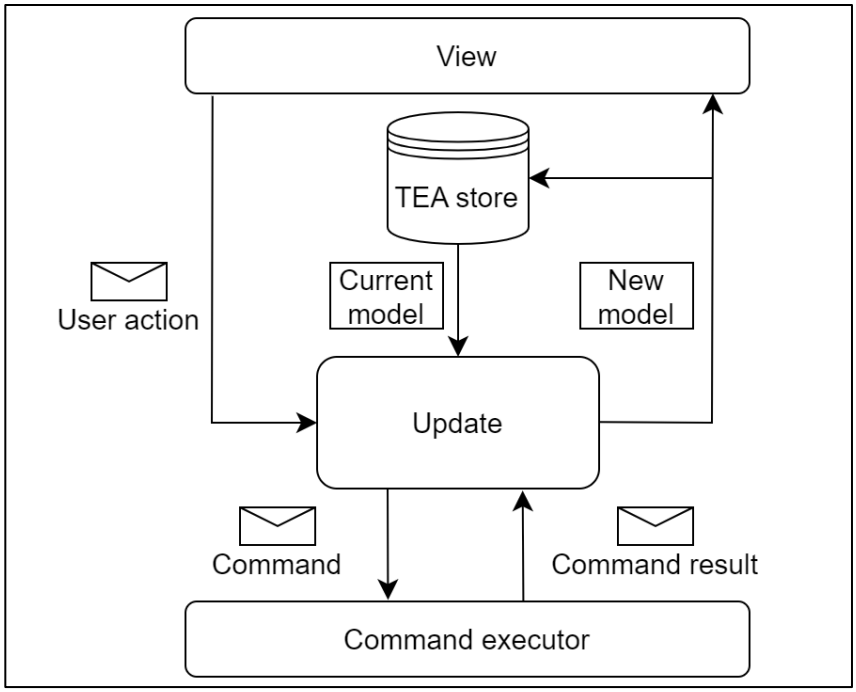

Рисунок 7 – Схема организации элементов TEA

К достоинствам данного подхода можно отнести явную регламентацию и разделение зон ответственности между элементами слоя представления. Благодаря этому решаются проблемы, обозначенные у MVVM ранее –

единое состояние не допускает противоречий в отображении пользовательского интерфейса, его легко сохранить во время смены конфигурации и восстановить позже. Также значительно повышается тестируемость приложения.

К недостаткам данного подхода можно отнести то, что его организация намного сложнее MVVM. Также он требует создания большого количества объектов даже для простых экранов.

Важно отметить, что при разработке мобильных приложений часто возникает потребность в выполнении однократных действий на уровне представления, например, показать всплывающее уведомление (toast) или осуществить навигацию на другой экран. К сожалению, паттерн TEA не регламентирует решение данных задач. Поэтому, согласно [20] в схему TEA был введен пятый элемент под названием «эффект».

Эффект (Effect) – это однократное действие, которое требуется выполнить на уровне представления. Обновленная схема с учетом нового элемента приведена на рисунке 8.

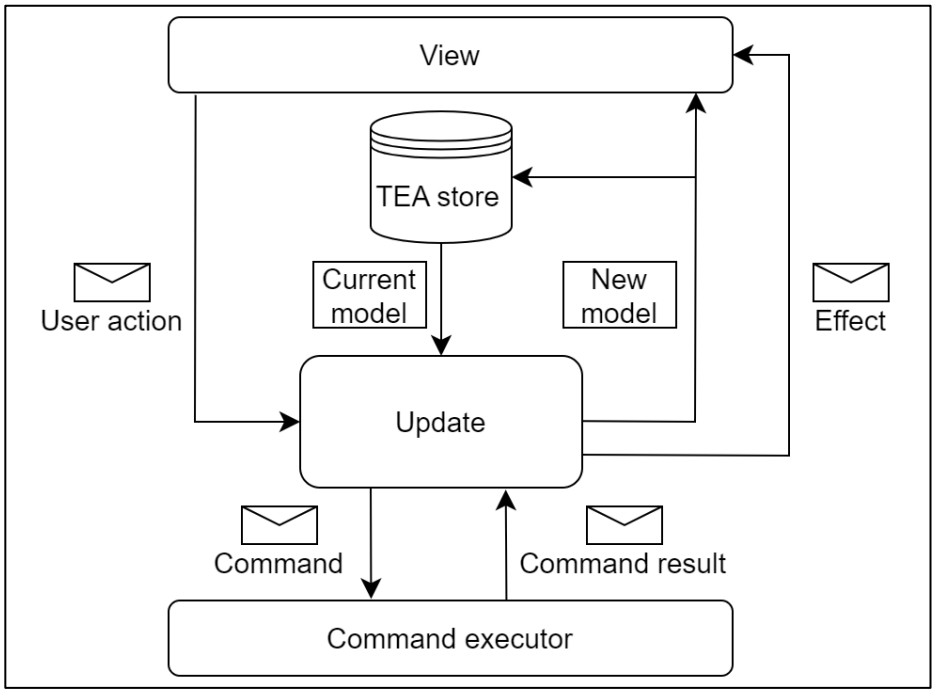

Рисунок 8 – Дополненная схема организации элементов TEA

Подводя итог, можно сказать, что у обоих подходов есть свои достоинства и недостатки, поэтому лучшим решением будет выбирать подход в зависимости от сложности того или иного экрана. Для простых экранов отлично подходит паттерн MVVM, в то время как для сложных экранов со множеством функциональных элементов – TEA.

2.3. Схема базы данных приложения

С целью хранения данных приложения была спроектирована схема базы данных. На рисунке 9 приведена часть таблиц, ответственных за хранение информации об альбомах.

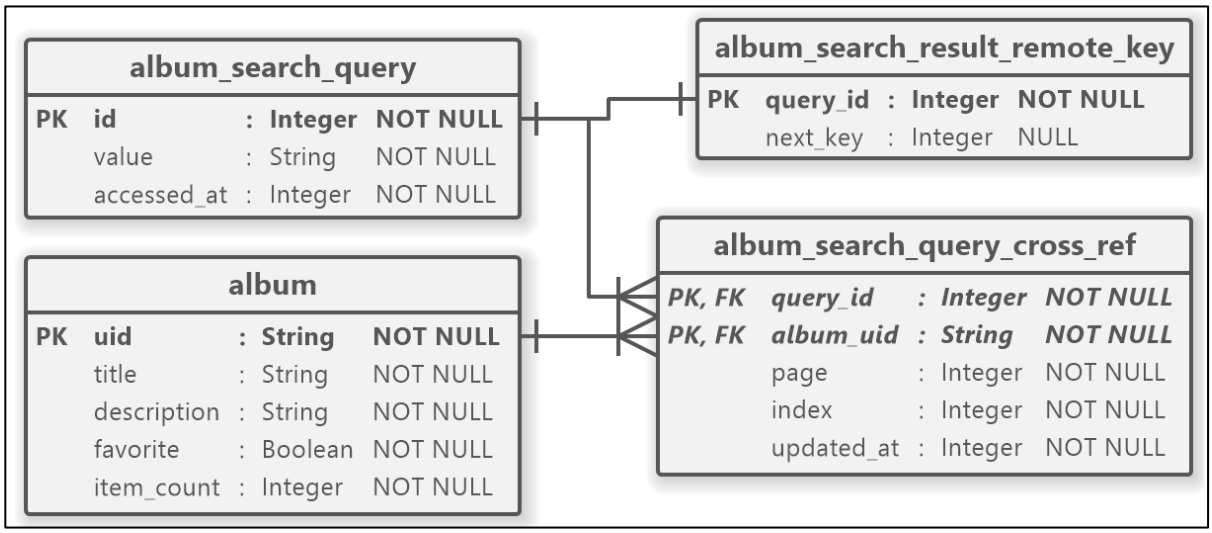

Рисунок 9 – Схема таблиц БД, ответственных за хранение альбомов

Таблица `album` служит для хранения информации о загруженных альбомах. Описание ее полей приведено в таблице 2.

Таблица 2 – Описание полей таблицы `album`

Название	Описание
uid	Уникальный идентификатор альбома
title	Название альбома
description	Описание альбома
favorite	Флаг, указывающий на то, находится ли альбом в разделе «избранное»
item_count	Количество элементов в альбоме

Таблица `album_search_query` служит для хранения информации о поисковых запросах, связанных с альбомами. Описание полей данной таблицы приведено в таблице 3.

Таблица 3 – Описание полей таблицы `album_search_query`

Название	Описание
id	Уникальный идентификатор поискового запроса
value	Поисковой запрос
accessed_at	Время последнего выполнения данного запроса в формате Unix Timestamp

Таблица `album_search_query_cross_ref` служит для выражения связи многие-ко-многим между таблицами `album_search_query` и `album`. Описание ее полей приведено в таблице 4.

Таблица 4 – Описание полей таблицы `album_search_query_cross_ref`

Название	Описание
query_id	Уникальный идентификатор поискового запроса
album_uid	Уникальный идентификатор альбома
page	Номер страницы поисковой выдачи, на которой находится данный альбом. Используется, чтобы сохранить порядок элементов поисковой выдачи при асинхронной загрузке страниц
index	Индекс альбома на странице. Используется, чтобы сохранить порядок элементов поисковой выдачи при асинхронной загрузке страниц
updated_at	Время последнего обновления данной записи базы данных

Таблица `album_search_result_remote_key` хранит необходимую информацию для организации постраничной загрузки результатов поиска. Описание ее полей приведено в таблице 5.

Таблица 5 – Описание полей таблицы `album_search_result_remote_key`

Название	Описание
query_id	Уникальный идентификатор поискового запроса
next_key	Индекс следующей страницы, которую нужно загрузить

На рисунке 10 приведена часть таблиц, ответственных за хранение информации о медиафайлах.

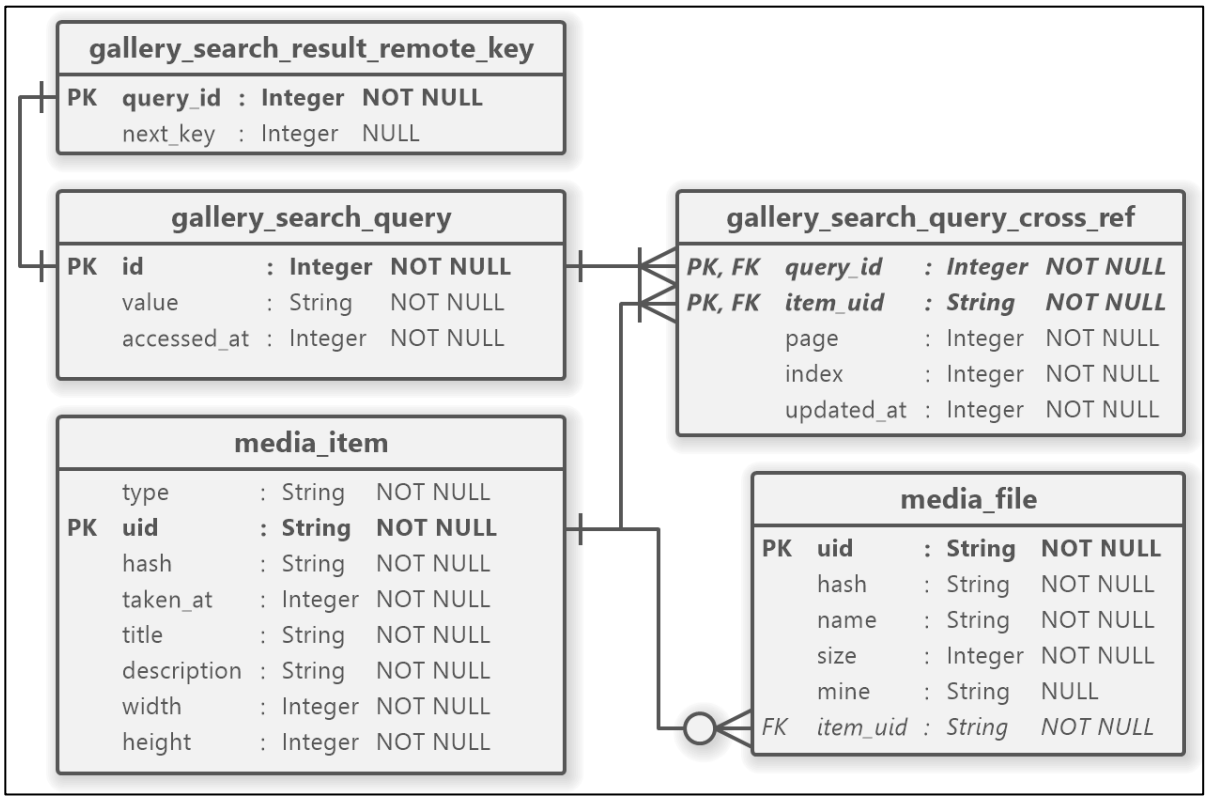

Рисунок 10 – Схема таблиц БД, ответственных за хранение медиафайлов

Таблица `gallery_search_query` служит для хранения информации о поисковых запросах, связанных с медиафайлами. Описание ее полей приведено в таблице 6.

Таблица 6 – Описание полей таблицы `gallery_search_query`

Название	Описание
id	Уникальный идентификатор поискового запроса
value	Поисковой запрос
accessed_at	Время последнего выполнения данного запроса в формате Unix Timestamp

Таблица `gallery_search_result_remote_key` хранит необходимую информацию для организации постраничной загрузки результатов поиска. Описание ее полей приведено в таблице 7.

Таблица 7 – Поля таблицы `gallery_search_result_remote_key`

Название	Описание
query_id	Уникальный идентификатор поискового запроса
next_key	Индекс следующей страницы, которую нужно загрузить

Таблица `media_item` служит для хранения информации о загруженных медиа-элементах – сущность, объединяющая относящиеся друг к другу медиафайлы, например, видео и его превью. Описание ее полей приведено в таблице 8.

Таблица 8 – Описание полей таблицы `media_item`

Название	Описание
type	Информация о типе медиа-элемента (фото, видео, gif и т.д.)
uid	Уникальный идентификатор медиа-элемента
hash	Хеш медиафайла, выступающего в роли превью
taken_at	Время в формате Unix Timestamp, когда был сделан данный медиафайл
title	Название медиа-элемента
description	Описание медиа-элемента
width	Ширина медиа-элемента
height	Высота медиа-элемента

Таблица `media_file` служит для хранения информации о медиафайлах, ассоциированных с конкретными медиа-элементами через связь многие-к-одному. Описание ее полей приведено в таблице 9.

Таблица 9 – Описание полей таблицы `media_file`

Название	Описание
uid	Уникальный идентификатор медиафайла
hash	Хеш медиафайла
name	Имя медиафайла в файловой системе
size	Размер медиафайла в байтах
mime	MIME-тип медиафайла
item_uid	Уникальный идентификатор родительского медиа-элемента

Таблица `gallery_search_query_cross_ref` служит для выражения связи многие-ко-многим между таблицами `gallery_search_query` и `media_item`. Описание ее полей приведено в таблице 10.

Таблица 10 – Описание полей таблицы `gallery_search_query_cross_ref`

Название	Описание
query_id	Уникальный идентификатор поискового запроса
item_uid	Уникальный идентификатор медиа-элемента
page	Номер страницы, на которой находится данный медиа-элемент
index	Порядковый номер альбома на странице
updated_at	Время последнего обновления данной записи базы данных

3. РЕАЛИЗАЦИЯ

3.1. Описание подхода

Для реализации мобильного приложения был выбран подход Single Activity, при котором на все приложение создается лишь одна активность (activity), обычно называемая `AppActivity`, а различные экраны создаются с использованием фрагментов. Большим преимуществом данного подхода является упрощение жизненного цикла приложения – он становится равным жизненному циклу `AppActivity`, что позволяет избежать многих проблем, связанных с неправильной реализаций восстановления приложения после смены конфигурации или завершения системой. Помимо этого, упрощается работа по созданию анимированного UI, поскольку платформа Android предлагает большой набор API для работы с фрагментами.

Также в приложении применено внедрение зависимостей (dependency injection), что позволяет уменьшить связность компонентов приложения. Для автоматизации реализации данного подхода был выбран Yatagan – библиотека от компании Яндекс, основанная на API библиотеки Dagger 2 от компании Google – одного из самых широко используемых решений для DI в сфере разработки Android приложений. Преимуществом данной библиотеки является то, что она имеет более содержательные сообщения об ошибках, нежели Dagger 2, что значительно ускоряет их поиск и устранение.

Yatagan позволяет хранить зависимости разных частей приложения в специальных компонентах, что дает возможность управлять временем жизни групп зависимостей (scope), привязывая его к времени жизни компонента. Библиотека также позволяет выстраивать иерархию компонентов, в рамках которой дочерним компонентам становятся доступны зависимости с верхних уровней иерархии. Иерархия компонентов разрабатываемого приложения приведена на рисунке 11, а описание всех компонентов – в таблице 11.

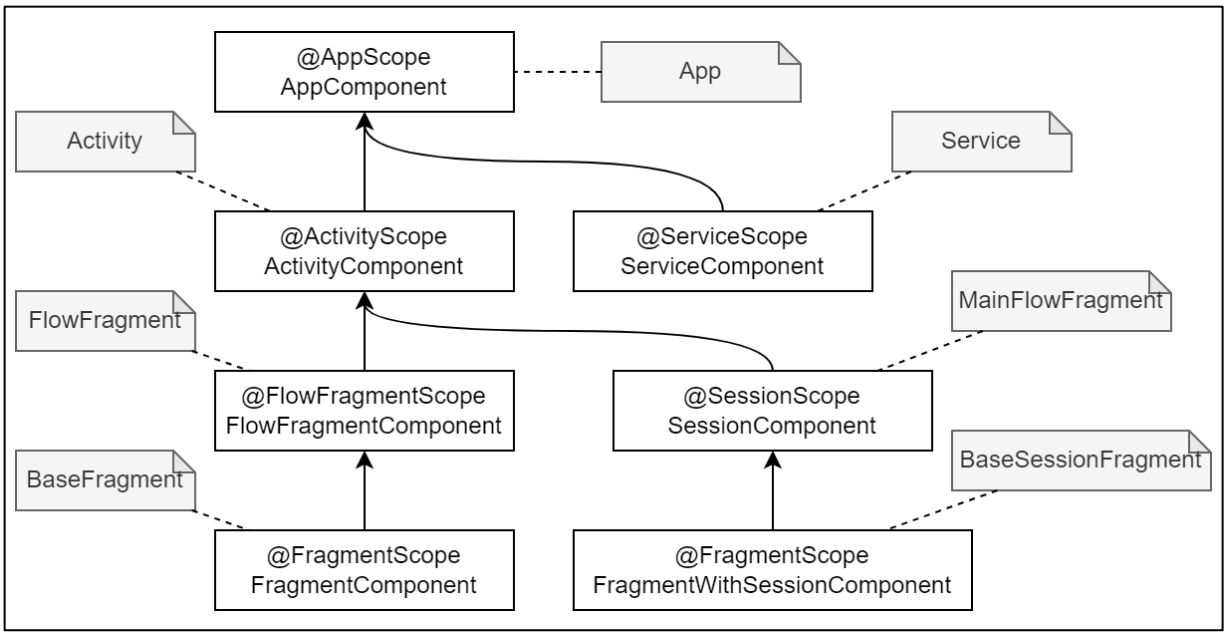

Рисунок 11 – Иерархия компонентов Yatagan

Таблица 11 – Описание компонентов DI

Компонент	**Описание**
AppComponent	Предназначен для хранения глобальных зависимостей приложения, таких как база данных, хранилище данных, http клиент, контекст приложения и так далее. Время жизни равно времени жизни приложения
ActivityComponent	Предназначен для хранения и предоставления контекста activity дочерним компонентам. Время жизни равно времени жизни activity
ServiceComponent	Предназначен для хранения и предоставления контекста сервиса (компонент Android) дочерним компонентам. Время жизни равно времени жизни сервиса
FlowFragmentComponent	Предназначен для хранения зависимостей одного маршрута навигации вне авторизованной зоны (например, экрана авторизации). Время жизни равно времени жизни фрагмента, ответственного за конкретный маршрут (FlowFragment)
FragmentComponent	Предназначен для хранения зависимостей фрагмента, используемого для создания экранов вне авторизованной зоны
SessionComponent	Предназначен для хранения зависимостей одного маршрута навигации в рамках авторизованной зоны. К таким зависимостям относится сессия пользователя, дополнительно настроенный http клиент и так далее
FragmentWithSessionComponent	Предназначен для хранения зависимостей фрагмента, используемого для создания экранов в рамках авторизованной зоны

При реализации проекта также использовались множество библиотек, которые значительно ускорили разработку приложения и позволили повысить качество конечного прототипа. Полный перечень библиотек приведен в таблице 12.

Таблица 12 – Перечь используемых библиотек

Название	**Описание**
Retrofit2	Библиотека для работы с сетевыми запросами в приложениях на Android. Она предоставляет удобный способ определения и отправки запросов на сервер, обработки ответов и работы с REST API.
Yatagan	Библиотека для внедрения зависимостей в приложениях на Android. Она предоставляет механизмы автоматического создания и связывания объектов, основываясь на их зависимостях.
Kotlin coroutines	Фреймворк для асинхронного и многопоточного программирования в языке Kotlin. Он предоставляет простой и лаконичный способ написания асинхронного кода в виде сопрограмм, которые позволяют писать последовательный код, выглядящий как синхронный.
Kotlin serialization	Библиотека для сериализации и десериализации данных в формат JSON или другие форматы. Она позволяет преобразовывать объекты Kotlin в JSON и наоборот, автоматически сопоставляя поля объекта с полями JSON.
Paging3	Библиотека, которая помогает загружать данные пакетами (страницами) из источников данных, таких как базы данных или сетевые запросы, и отображать их в пользовательском интерфейсе. Она предоставляет механизм автоматического управления постраничной загрузкой данных, предзагрузкой следующих страниц и обновлением списка при прокрутке.
Timber	Логгер для Android приложений
Room	Библиотека для работы с базами данных SQLite в приложениях на Android. Она предоставляет абстракцию над SQLite, позволяющую удобно определять структуру базы данных и выполнять запросы к ней.
KoTEA	Реализация паттерна TEA на Kotlin от компании Тинькофф
TiRecyler	Библиотека для удобной работы с RecyclerView от компании Тинькофф
Glide	Библиотека для загрузки и отображения изображений в Android приложениях. Она предоставляет эффективные механизмы кэширования и обработки изображений, поддерживает анимацию загрузки и многое другое.
Media3	Современная библиотека для работы с медиа в Android приложениях, разработанная командой Google. Применяется для встраивания и управления воспроизведением аудио и видео контента, обеспечивая плавное воспроизведение при различных условиях сети.
Navigation Component	Библиотека для удобного определения и управления навигационным графом приложения, переходами между фрагментами или активностями, анимациями и передачей параметров между экранами.
Kotest	Многофункциональный фреймворк для тестирования Kotlin приложений, предоставляющий богатый набор инструментов и возможностей.

3.2. Навигация

В приложении используется библиотека Navigation Component, которая позволяет удобно выстраивать навигацию в приложении с помощью навигационного графа.

Со временем развитие приложения приводит к появлению большого количества функций и увеличению количества вершин графа. В следствие чего поддержка существующих экранов и добавление новых становится достаточно трудоемкой задачей. С целью упрощения навигационного графа в разрабатываемом приложении был применен принцип декомпозиции, когда экраны, похожие по смыслу и относящиеся к различным областям приложения выносятся в отдельный навигационный граф. Такой граф называется маршрутом (flow).

В разрабатываемом приложении было выделено два различных маршрута: маршрут авторизации (`AuthFlow`) и основной маршрут, который доступен только после авторизации (`MainFlow`). Каждому из них соответствует свой файл с навигационным графом – `auth_flow.xml` и `main_flow.xml` соответственно. Чтобы связать эти маршруты был создан отдельный файл с навигационным графом, в котором определены только глобальные действия, позволяющие в любой момент запустить нужный маршрут.

В рамках библиотеки Navigation Component переключение между экранами подразумевает переключение фрагментов, но не навигационных графов. Поэтому, чтобы иметь возможность переключать маршруты, для каждого из них был создан отдельный так называемый навигационный фрагмент (`FlowFragment`), в котором содержится соответствующий навигационной контроллер (`NavController`) с графом (рисунок 12).

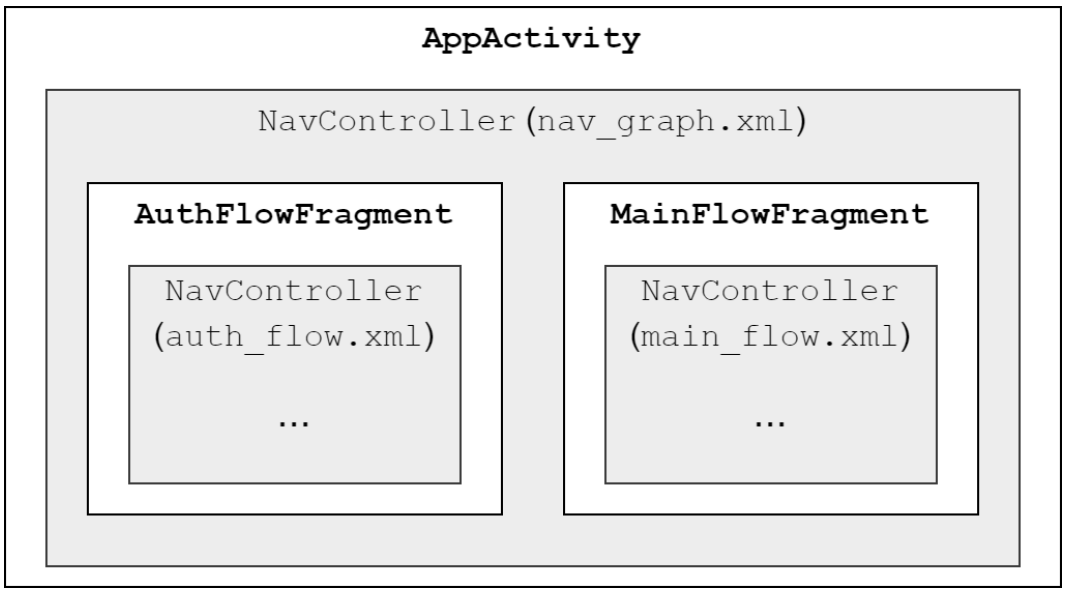

Рисунок 12 – Организация навигационных компонентов

Помимо навигации, данные фрагменты выполняют еще одну важную функцию – они позволяют управлять временем жизни группы зависимостей. Например, основной маршрут доступен только после авторизации. Соответственно, все относящиеся к данному маршруту экраны имеют зависимость в виде специального HTTP клиента, который к каждому запросу добавляет заголовок с сессионным ключом. Данный HTTP клиент существует до тех пор, пока пользователь находится в авторизованной зоне (т.е. на основном маршруте) и должен быть уничтожен сразу же, как только пользователь покинет данную зону. Навигационные фрагменты позволяют очень просто реализовать такое поведение – достаточно поместить в них соответствующий компонент Yatagan и определить область жизни зависимостей (scope).

Реализация навигационных компонентов однотипна, поэтому, чтобы избежать дублирования кода, она была вынесена в базовый абстрактный класс `BaseFlowFragment`, от которого наследуются все остальные навигационные фрагменты.

3.3. Структура проекта

Поскольку разрабатываемое приложение содержит большое количество различных компонентов, было принято решение использовать организацию проекта в виде пакетов. Итоговая структура приложения приведена на рисунке 13.

Рисунок 13 – Дерево пакетов

Рассмотрим каждый пакет подробнее.

1. Пакет `photoprism` является корневым и содержит классы и функции, которые отвечают за высокоуровневые функции приложения. В частности, он содержит класс `AppActivity`, роль которого описана в разделе 3.1.

2. Пакет `api` содержит все необходимое для выполнения сетевых запросов к серверу PhotoPrism – описание моделей, интерфейсы Retrofit и классы исключений.

3. Пакет `common` содержит сервисы, которые используются несколькими компонентами или модулями. Такими сервисами, например, являются фабрики URL-адресов, которые позволяют сгенерировать URL-адрес для получения миниатюр или скачивания медиафайлов.

4. Пакет `base` содержит базовые классы, в которые вынесен часто повторяющийся код.

5. Пакет `db` содержит файлы, необходимые для работы с базой данных SQLite посредством ОРМ библиотеки Room. Стоит отметить, что данный пакет содержит лишь объявление базы данных, а объекты доступа к данным (DAO) находятся непосредственно в пакетах модулей.

6. Пакет `di` содержит все файлы, связанные с инъекцией зависимостей: компоненты, модули и аннотации.

7. Пакет `features` содержит модули, ответственные за весь основной функционал приложения – авторизацию, отображение на экране списка медиафайлов и альбомов, просмотр медиафайлов в деталях.

8. Пакет `navigation` содержит все компоненты, которые относятся к навигации в приложении.

9. В пакет `util` вынесены удобные функции. Например, для проверки строки на то, что она является корректным URL-адресом.

На рисунке 14 представлено содержимое пакета `features`. Из рисунка видно, что каждый модуль состоит из описанных ранее архитектурных слоев, а структура пакетов выполнена в едином стиле, что облегчает навигацию по коду программы.

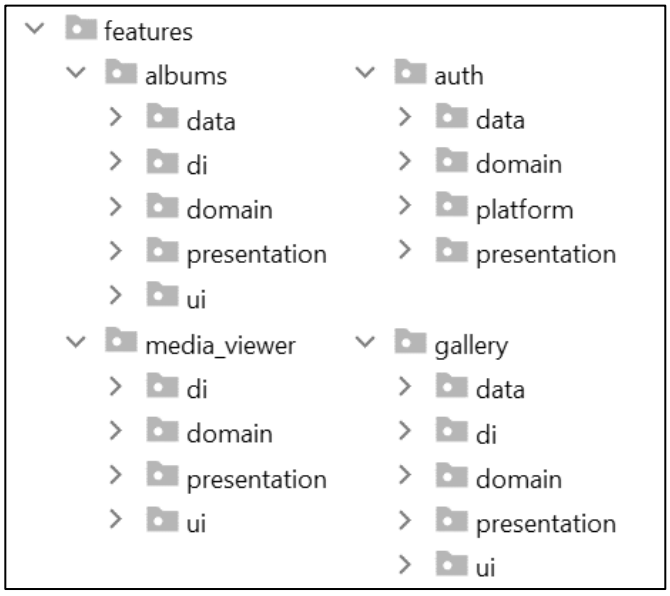

Рисунок 14 – Содержимое пакета `features`

3.4. Модуль «Авторизация»

Данный модуль содержит реализацию формы авторизации, представленной на рисунке 15.

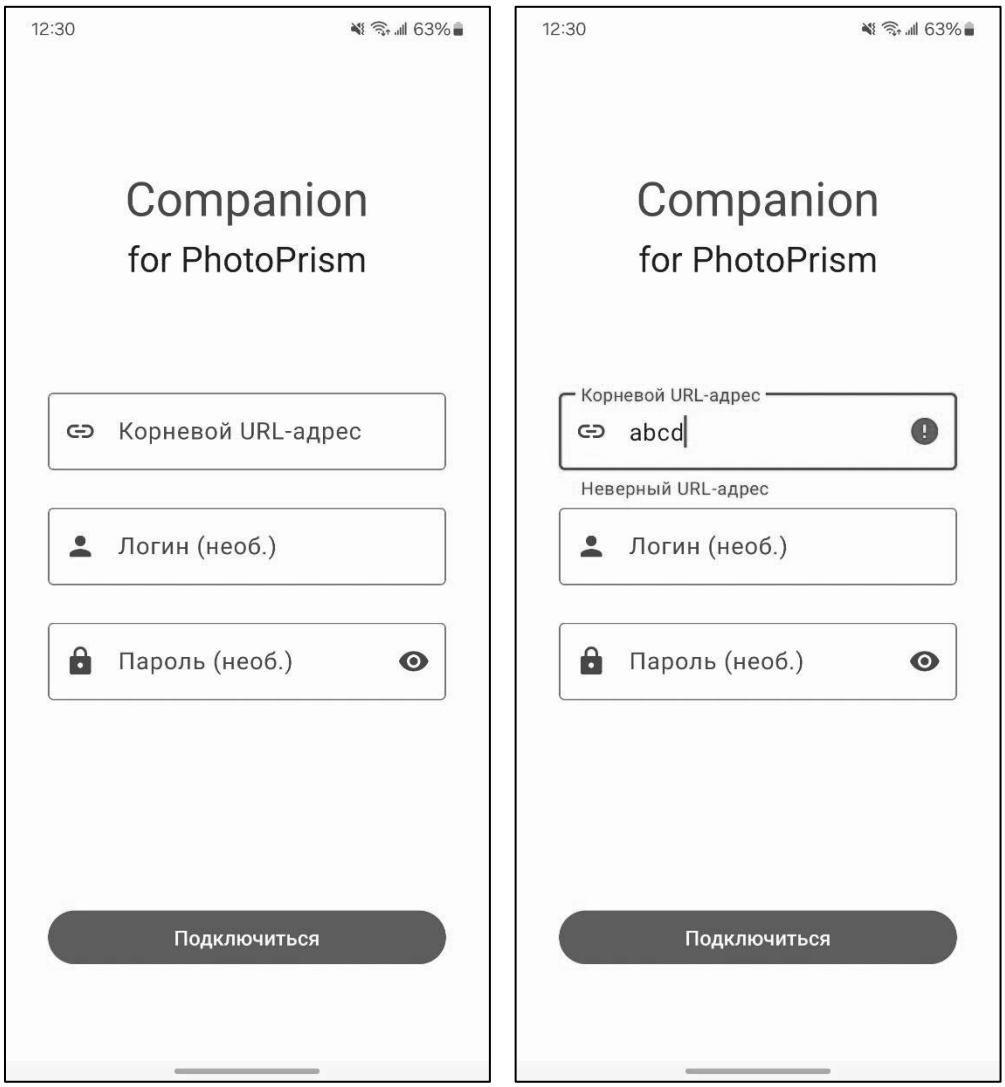

Рисунок 15 – Форма авторизации

Согласно принятой архитектуре, модуль «Авторизация» состоит из трех слоев – представления, доменного и инфраструктурного. Однако из-за специфики данного модуля появляется необходимость в еще одном слое – платформенном.

Платформенный слой содержит два класса – `AccountAuthenticator` и `AuthenticatorService`. Данные классы позволяют задействовать функционал Android для хранения учетных данных и автоматического продления сессии.

Доменный слой содержит описание моделей и бизнес-логику. К моделям относятся классы, представленные ниже.

1. `LibraryAccount` – это класс данных, который хранит основную информацию об аккаунте пользователя: корневой URL-адрес сервера PhotoPrism и имя пользователя.

2. `LibraryAccountCredentials` – это класс данных, который хранит учетные данные пользователя, необходимые для авторизации на сервере PhotoPrism: корневой URL-адрес сервера PhotoPrism, имя и пароль пользователя.

3. `LibraryAccountSession` – это класс данных, который хранит сессионные ключи авторизации: идентификатор сессии, токены для загрузки миниатюр и файлов.

4. `LibraryConnectParams` – это запечатанный (sealed) интерфейс, который определяет режим подключения к серверу PhotoPrism. Для подключения в публичном режиме требуется только корневой URL-адрес сервера PhotoPrism. Для подключения в частном режиме требуется корневой URL-адрес сервера PhotoPrism, логин и пароль пользователя. Диаграмма классов для данной модели приведена на рисунке 16.

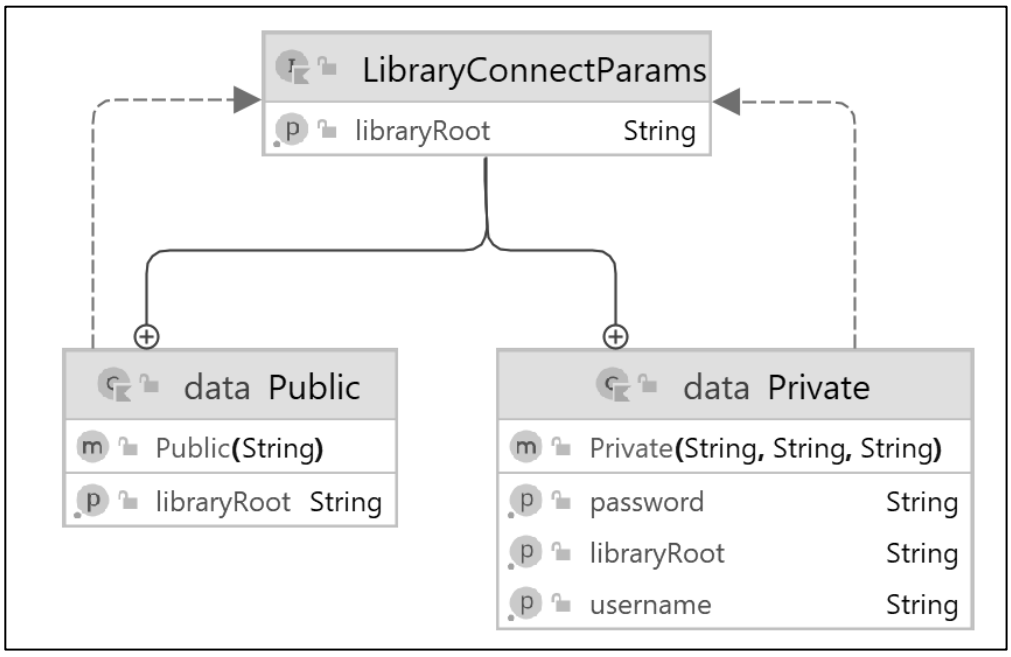

Рисунок 16 – Интерфейс `LibraryConnectParams`

К бизнес-логике данного модуля относятся следующие два класса.

1. `PhotoPrismAuthInterceptor` – это перехватчик запросов для HTTP-клиента OkHttp, который для каждого запроса добавляет заголовок, содержащий идентификатор сессии. В случае ошибки отправляет `AuthException` в `AuthExceptionHandler`.

2. `AuthExceptionHandler` – объект, реализующий паттерн «наблюдатель». При возникновении ошибки авторизации он оповещает всех подписчиков.

Помимо этого, к бизнес-логике относятся классы, реализующие различные варианты использования данного модуля другими компонентами приложения:

1) `AddAccountUseCase` – позволяет добавить новый аккаунт;

2) `ConnectToLibraryUseCase` – позволяет подключиться к серверу;

3) `GetActiveAccountUseCase` – позволяет получить активный на данный момент аккаунт;

4) `GetSessionUseCase` – позволяет получить данные сессии, относящиеся к определенному аккаунту;

5) `InvalidateSessionUseCase` – позволяет инвалидировать данные сессии и тем самым запустить процесс авторизации заново;

6) `SetActiveAccountUseCase` – позволяет установить активный аккаунт;

7) `SetSessionUseCase` – позволяет сохранить данные сессии в хранилище Android.

Каждый вариант использования обращается к одному из следующих репозиториев, реализация которых относится к инфраструктурному (data) слою:

1) `LibraryAccountRepository` – это репозиторий, который позволяет совершать различные действия с аккаунтами – добавлять новый аккаунт, а также получать и устанавливать активный аккаунт;

2) `LibrarySessionRepository` – это репозиторий, который позволяет совершать различные действия с сессиями авторизации, а именно – получать, добавлять и удалять.

К инфраструктурному слою также относятся вспомогательные классы, такие как `PhotoPrismAuthenticator` и `ActiveAccountDataStore`. Они позволяют авторизовываться на сервере PhotoPrism и сохранять информацию об активном аккаунте на диск соответственно.

Слой представления построен на базе паттерна MVVM и содержит классы, необходимые для функционирования формы авторизации.

К классам формы авторизации относятся:

1) `LibraryConnectFragment` – фрагмент, непосредственно отвечающий за отображение формы авторизации;

2) `LibraryConnectViewModel` – модель представления, которая хранит в себе состояние экрана и содержит логику валидации данных.

Основной задачей данной формы является проверка вводимых пользователем данных на корректность и отображение соответствующих ошибок. Диаграмма деятельности, демонстрирующая алгоритм работы формы, представлена в приложении Б.

3.5. Модуль «Галерея»

Данный модуль содержит реализацию экрана, который отображает полный перечень медиафайлов в виде галереи, группирует элементы по дате создания, а также позволяет выполнить поисковой запрос и отобразить его результаты. Скриншоты данного экрана, демонстрирующие обычное состояние и режим поиска, представлены на рисунке 17.

Согласно принятой архитектуре, модуль «Галерея» состоит из трех слоев – представления, доменного и инфраструктурного. Рассмотрим состав каждого из них по отдельности.

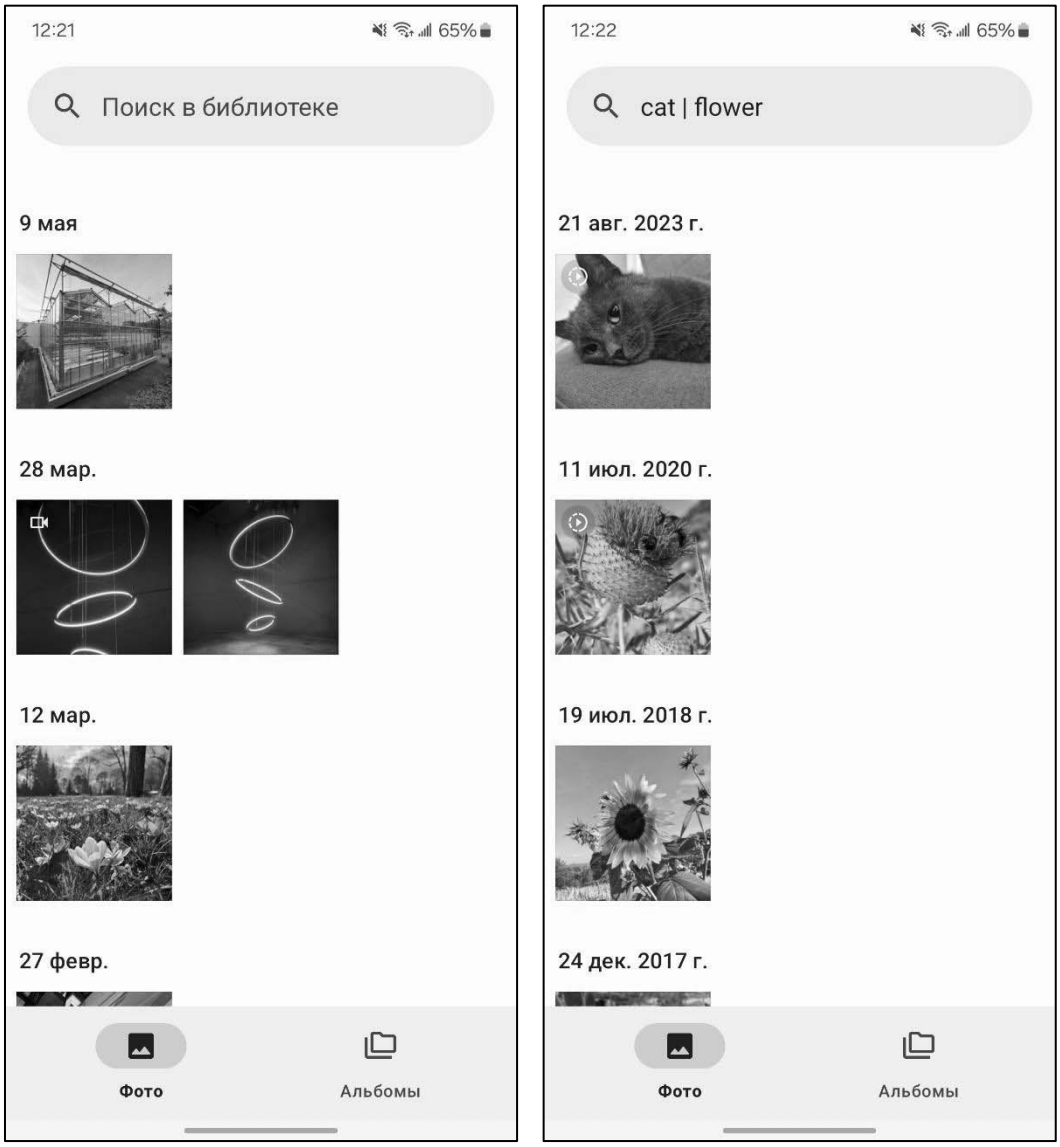

Рисунок 17 – Галерея медиафайлов

Доменный слой содержит описание моделей и бизнес-логику. К моделям относятся следующие классы.

1. `MediaItem` – это запечатанный (sealed) интерфейс, который предписывает всем наследникам иметь свойства, хранящие основную информацию о медиа-элементе: уникальный идентификатор, дату создания, заголовок, описание, хеш файла, выступающего в роли превью, высоту, ширину, список медиафайлов, относящихся к данному элементу, URL-адрес превью малого размера, URL-адрес превью среднего размера. Список классов, реализующих данный интерфейс, следующий: `Unknown`, `Animation`, `Live`,

`Video`, `Sidecar`, `Text`, `Other`, `Image`, `Raw`, `Vector`. Таким образом, каждый из наследников представляет определенный тип файла.

2. `MediaFile` – это класс данных, который хранит основную информацию о медиафайле: уникальный идентификатор, имя, хеш, размер в байтах, MIME-тип, URL-адрес миниатюры малого размера, URL-адрес миниатюры среднего размера, URL-адрес для загрузки файла.

3. `GallerySearchQuery` – это класс данных, который хранит запрос на поиск медиафайлов, а также дополнительные параметры сообщающие, например, о том, что нужно полностью очистить кэш поисковых запросов и создать его заново.

К бизнес-логике данного модуля относятся классы, реализующие различные варианты использования данного модуля другими компонентами приложения:

1) `GetSearchResultUseCase` – класс, который позволяет получить постранично результаты поиска в виде Kotlin Flow;

2) `GetThumbnailUrlUseCase` – класс, который позволяет сгенерировать URL-адрес миниатюры медиафайла.

Каждый вариант использования обращается к репозиторию `MediaRepository`, реализация которого относится к инфраструктурному (data) слою. Данный репозиторий позволяет выполнять поиск медиафайлов с кэшированием в локальной базе данных и получать количество результатов поиска.

Для эффективного использования ресурсов смартфона при загрузке длинных списков была задействована библиотека Paging 3. Она позволяет достаточно просто реализовать постраничную загрузку списков, предоставляя удобные для этого инструменты.

Как только адаптер списка достигает края кэша, компонент `Pager` сообщает компоненту `MediaRemoteMediator` о необходимости пополнить кэш данными из сети. `MediaRemoteMediator`, получив данный сигнал, загружает данные из сети и сохраняет в локальную базу данных и помечает

текущий экземпляр `PagingSource` как недействительный. Затем компонент `Pager` создает новый `PagingSource`, который подхватывает недавно загруженные данные из базы данных и предоставляет их компоненту `Pager`, который в свою очередь разбивает их на страницы, упаковывает в поток типа `Flow<PagingData>` и предоставляет пользователю класса репозитория (в данном случае `GetSearchResultUseCase`). Подробнее с принципом работы библиотеки можно ознакомится на странице документации [21].

Работа с базой данных организована посредством библиотеки Room на основании ранее разработанной схемы. Подробное описание реализации опускается, поскольку она является типичной для данной библиотеки и ORM в целом [22]. Следует отметить, что перед сохранением в базу данных поисковой запрос подвергается нормализации – из него убираются лишние пробелы и символы, поскольку они не влияют на результат поиска, но при этом вызывают дублирование записей.

Слой представления построен на базе паттерна TEA и реализует все классы, необходимые для функционирования экрана с перечнем медиафайлов в виде галереи.

Данный экран выполняет сразу две большие задачи, а именно – постраничное отображение списка и предоставление пользовательского интерфейса для задания параметров поиска. С целью облегчения разработки и дальнейшей поддержки он был разделен на соответствующие компоненты, каждый из которых построен с использованием паттерна TEA. Далее следует подробное рассмотрение этих компонентов.

Компонент Gallery

Компонент для постраничного отображения списка состоит из следующих классов и интерфейсов.

1. `GalleryStore` – центральный класс, который управляет состоянием компонента и обрабатывает события.

2. `GalleryEvent` – общий маркерный интерфейс для всех событий, связанных с постраничной загрузкой.

3. `GalleryUiEvent` – маркерный интерфейс для событий пользовательского интерфейса.

4. `GalleryCommandResult` – маркерный интерфейс для событий, представляющих собой результат асинхронных команд.

5. `GalleryNews` – маркерный интерфейс для событий, представляющих одноразовое действие.

6. `GalleryCommand` – маркерный интерфейс для событий, представляющих собой асинхронные команды.

7. `GalleryState` – класс данных для хранения состояния списка.

8. `GalleryUpdate` – класс для обновления состояния списка в зависимости от текущего события. Код класса приведен в листинге 1.

9. `GalleryUiState` – класс для хранения состояния списка в виде, непосредственно пригодном для отображения на экране.

10. `GalleryUiStateMapper` – класс для преобразования внутреннего состояния `GalleryState` в состояние пользовательского интерфейса.

11. `GalleryFragment` – это фрагмент, непосредственно отвечающий за отображение галереи медиафайлов. Он также ответственен за отображение интерфейса поиска.

12. `GalleryViewHolderFactory` – класс, необходимый для отображения списка медиафайлов в виде набора элементов типа `ViewTyped` (библиотека TiRecycler) с использованием RecyclerView (компонент Android).

Полный перечень событий, которые участвуют в управлении состоянием экрана в обычном режиме (когда пользовательский интерфейс для ввода поисковых параметров скрыт) приведен в таблице 13. Код класса, ответственного за обработку данных событий и изменение состояния, приведен в листинге 1.

Таблица 13 – Перечень событий компонента Gallery

Название события	Тип	Описание
`OnClickMediaItem`	`GalleryUiEvent`	Нажатие на элемент списка
`OnClickRestart`	`GalleryUiEvent`	Нажатие кнопки «Повторить» при ошибке загрузки
`OnLoadMore`	`GalleryUiEvent`	Сообщение о необходимости предпринять попытку загрузки следующей страницы
`OnPerformSearch`	`GalleryUiEvent`	Действие «инициализировать постраничный загрузчик с новыми параметрами»
`PerformSearchResult`	`GalleryCommandResult`	Сообщение о текущем состоянии постраничной загрузки
`PerformSearchError`	`GalleryCommandResult`	Сообщение об ошибке постраничной загрузки
`OpenPreview`	`GalleryNews`	Действие «открыть просмотр медиафайла»

Листинг 1 – Класс `GalleryUpdate`

```
class GalleryUpdate @Inject constructor() :
    DslUpdate<GalleryState, GalleryEvent, GalleryCommand, GalleryNews>() {

    override fun NextBuilder.update(event: GalleryEvent) = when(event) {
        is OnClickMediaItem -> news(OpenPreview(event.position))
        is PerformSearchResult -> state { copy(listState=event.listState) }
        is PerformSearchError -> Unit
        is OnClickRestart -> commands(Restart)
        is OnLoadMore -> commands(LoadMore(event.position))
        is OnPerformSearch -> handleOnPerformSearch(event.query)
    }

    private fun NextBuilder.handleOnPerformSearch(
        query: GallerySearchQuery
    ) {
        state { copy(listState = PagingState.initial()) }
        commands(PerformSearch(query))
    }
}
```

Для моделирования состояний постраничной загрузки списка была разработана универсальная запечатанная (sealed) иерархия классов, код которой приведен в листинге 2.

Листинг 2 – Иерархия классов `Paging`

```
sealed interface PagingState<out T : Any> {
    companion object {
        fun initial(): PagingState<Nothing> = EmptyProgress
    }

    data object Empty : PagingState<Nothing>
    data object EmptyProgress : PagingState<Nothing>
```

```
    data class EmptyError(val error: Throwable) : PagingState<Nothing>
    data class Data<out T : Any>(val data: List<T>) : PagingState<T>
    data class Refresh<out T : Any>(val data: List<T>) : PagingState<T>
    data class NewPageProgress<out T : Any>(
        val data: List<T>
    ) : PagingState<T>
    data class NewPageError<out T : Any>(
        val data: List<T>
    ) : PagingState<T>
    data class FullData<out T : Any>(val data: List<T>) : PagingState<T>
}

sealed interface PagingError {
    data class PageLoadError(val error: Throwable) : PagingError
}
```

Как видно из листинга, список может пребывать во многих состояниях, для каждого из которых предусмотрен соответствующий набор элементов списка типа `ViewTyped`.

Компонент GallerySearch

Компонент для отображения пользовательского интерфейса для задания параметров поиска состоит из следующих классов и интерфейсов.

1. `GallerySearchStore` – центральный класс, который управляет состоянием компонента и обрабатывает события.

2. `GallerySearchEvent` – общий маркерный интерфейс для всех событий, связанных с поиском медиафайлов.

3. `GallerySearchUiEvent` – маркерный интерфейс для событий пользовательского интерфейса.

4. `GallerySearchCommandResult` – маркерный интерфейс для событий, представляющих собой результат асинхронных команд.

5. `GallerySearchNews` – маркерный интерфейс для событий, представляющих одноразовое действие.

6. `GallerySearchCommand` – маркерный интерфейс для событий, представляющих собой асинхронные команды.

7. `GallerySearchState` – класс данных для хранения состояния поиска. Код класса приведен в листинге 3.

8. `GallerySearchUiState` – класс для хранения состояния поиска в виде, непосредственно пригодном для отображения на экране. Код класса приведен в листинге 4.

9. `GallerySearchUpdate` – класс для обновления состояния поиска в зависимости от текущего события.

10. `GallerySearchUiStateMapper` – класс для преобразования внутреннего состояния `GallerySearchState` в состояние пользовательского интерфейса.

Листинг 3 – Класс `GallerySearchState`

```
data class GallerySearchState(
    val album: Album? = null,
    val queryText: String = EMPTY_STRING,
    val currentQuery: GallerySearchQuery? = null
)
```

Листинг 4 – Класс `GallerySearchUiState`

```
class GallerySearchUiState(
    val applyBtnEnabled: Boolean,
    val searchBarHint: String,
    val searchBarText: String,
    val searchViewText: String
)
```

Полный перечень всех событий, которые участвуют в управлении экраном в режиме поиска, приведен в таблице 14. Код класса, ответственного за обработку данных событий, приведен в листинге 5. Отдельно стоит отметить, что событие `PerformSearch` не запускает процесс поиска, а лишь уведомляет компонент Gallery о таковом намерении пользователя.

Таблица 14 – Список событий компонента GallerySearch

Название события	Тип	Описание
`OnSearchTextChanged`	`GallerySearchUiEvent`	Изменение текста поискового запроса
`OnApplySearch`	`GallerySearchUiEvent`	Нажатие кнопки «Применить»
`OnResetSearch`	`GallerySearchUiEvent`	Нажатие кнопки «Сбросить»
`HideSearchView`	`GallerySearchNews`	Действие «скрыть диалог с параметрами поиска»
`PerformSearch`	`GallerySearchNews`	Намерение «запустить поиск с заданными параметрами»

Листинг 5 – Класс `GallerySearchUpdate`

```
class GallerySearchUpdate @Inject constructor(val albumUid: String)
: DslUpdate<GallerySearchState, GallerySearchEvent, GallerySearchCommand,
GallerySearchNews>() {
    override fun NextBuilder.update(event: GallerySearchEvent) {
        when(event) {
            is OnApplySearch -> handleOnApplySearch()
            is OnResetSearch -> handleOnResetSearch()
            is OnSearchTextChanged -> state {
                copy(queryText = event.text.trim())
            }
        }
    }

    private fun NextBuilder.handleOnApplySearch() {
        news(GallerySearchNews.HideSearchView)
        val query = GallerySearchQuery(
            value = state.queryText,
            albumUid = albumUid.ifBlank { null },
            config = Config(refresh = true)
        )
        if (query.value != state.currentQuery?.value) {
            state { copy(currentQuery = query) }
            news(GallerySearchNews.PerformSearch(query))
        }
    }

    private fun NextBuilder.handleOnResetSearch() {
        state { copy(queryText = FULL_GALLERY_QUERY) }
        news(GallerySearchNews.HideSearchView)
        val query = GallerySearchQuery(
            value = state.queryText,
            albumUid = albumUid.ifBlank { null },
        )
        if (query.value != state.currentQuery?.value) {
            state { copy(currentQuery = query) }
            news(GallerySearchNews.PerformSearch(query))
            config = Config(refresh = false)
        }
    }
}
```

3.6. Модуль «Просмотр»

Требования к данному модулю почти такие же, как и к модулю «Галерея» – постранично загружать список медиафайлов и отображать его на экране. Отличие заключается лишь в том, что файлы отображаются не виде списка, а на весь экран с возможностью пролистывания вправо и влево. Помимо этого, также показывается дополнительная информация о медиафайле и поддерживается воспроизведение видео. Скриншоты экрана представлены на рисунке 18.

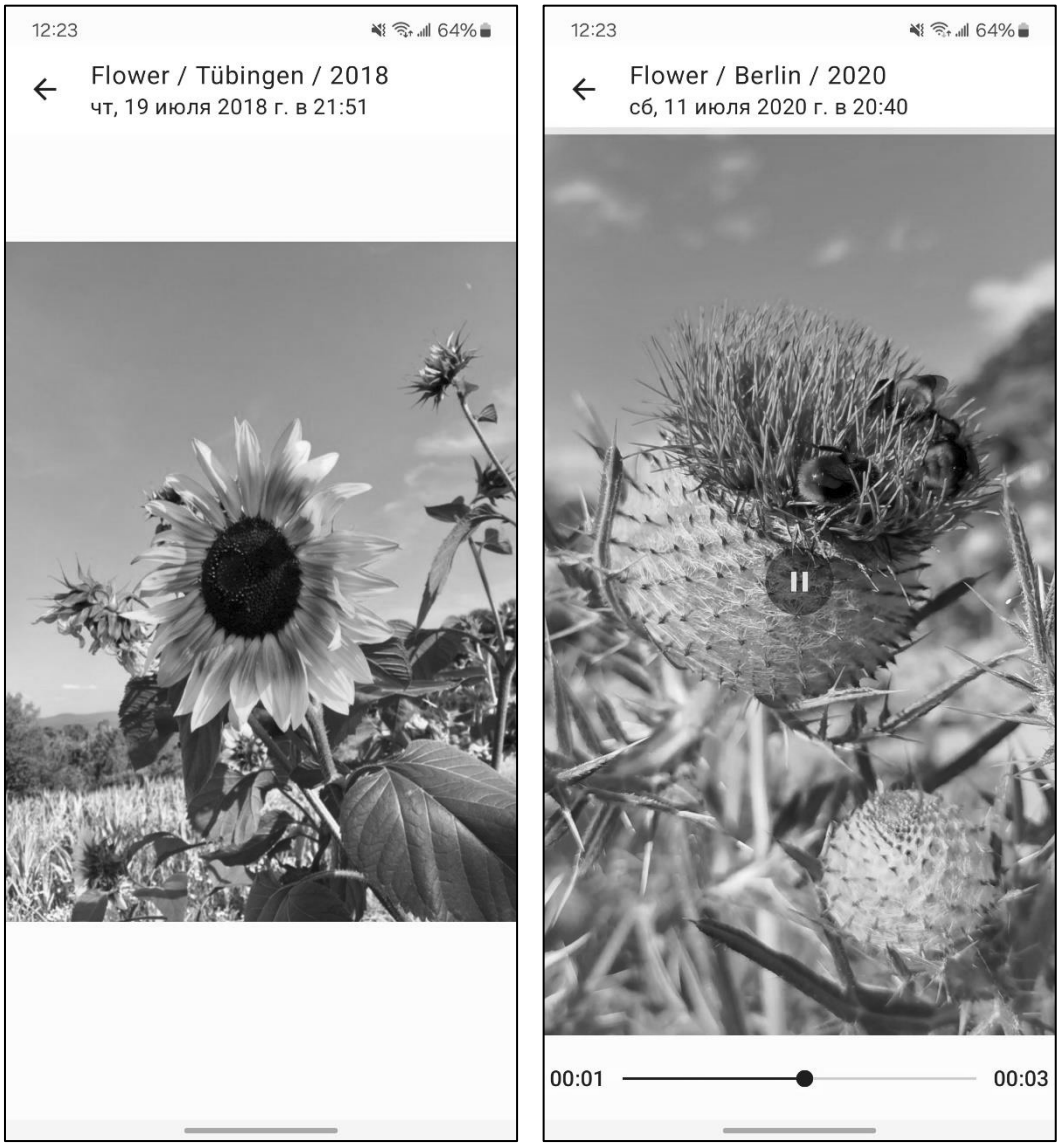

Рисунок 18 – Полноэкранный просмотр фото и видео

Среди основных классов, необходимых для реализации данного модуля, можно выделить следующие.

1. `PagerUiState` – класс для представления состояния списка медиафайлов в виде, непосредственно пригодном для отображения на экране просмотра медиафайла.

2. `PagerUiStateMapper` – класс для преобразования внутреннего состояния `GalleryState` в состояние пользовательского интерфейса (в `PagerUiState`).

3. `MediaViewerFragment` – это фрагмент, непосредственно отвечающий за отображение содержимого экрана.

4. `MediaViewerHolderFactory` – класс, необходимый для отображения списка медиафайлов в виде набора элементов типа `ViewTyped` (библиотека TiRecycler) с использованием `ViewPager2` (компонент Android).

5. `MediaViewerPreloadModelProvider` – класс, который отслеживает направление пролистывания списка и заранее формирует запросы на загрузку полноразмерных файлов из сети.

Благодаря использованию паттерна TEA для реализации модуля «Галерея» становиться возможным гибкое повторное использование его компонентов, отвечающих за постраничную загрузку списка (в частности, `GalleryStore` и `GalleryState`).

Однако в таком случае для реализации расширенного функционала (например, отображения дополнительной информации о файле) потребуются отдельные классы TEA, аналогичные таковым в других модулях. Также потребуется логика синхронизации двух TEA компонентов – `GalleryStore` и `MediaViewerStore` – она приведена в листинге 6.

Листинг 6 – Код для синхронизации `MediaViewerStore` с `GalleryStore`

```
// Данный фрагмент кода расположен в MediaViewerFragment в функции onCreate
// Здесь и далее component - это DI контейнер с
// зависимостями для MediaViewer
val pagerUiStateMapper = component.pagerUiStateMapper
// Инициализация компонента MediaViewerStore, который
// позволяет реализовать дополнительный функционал на экране просмотра
viewerStore.collectOnCreate(
    fragment = this,
    uiStateMapper = component.mediaViewerUiStateMapper,
    stateCollector = /* Ссылка на функцию для отображения состояния */
)
// Инициализация компонента GalleryStore, который
// повторно используется из модуля "Галерея"
galleryStore.collectOnCreate(
    lifecycleOwner = this,
    stateCollector = { // Синхронизация происходит здесь
        // При обновлении состояния GalleryState уведомляем
        // MediaViewerStore о новом списке элементов
        viewerStore.dispatch(OnItemsUpdate(it.listState.data))
        // Преобразуем состояние GalleryState в PagerUiState и
        // и вызываем функцию для отображения списка
        collectPagerState(
            // resourcesProvider - это компонент из библиотеки KoTEA
            pagerUiStateMapper.map(resourcesProvider, it)
        )
    }
)
```

3.7. Модуль «Альбомы»

Данный модуль содержит реализацию экрана, который отображает полный перечень альбомов, а также позволяет выполнить поисковой запрос и отобразить его результаты. Скриншоты данного экрана, демонстрирующие обычное состояние и режим поиска, представлены на рисунке 19.

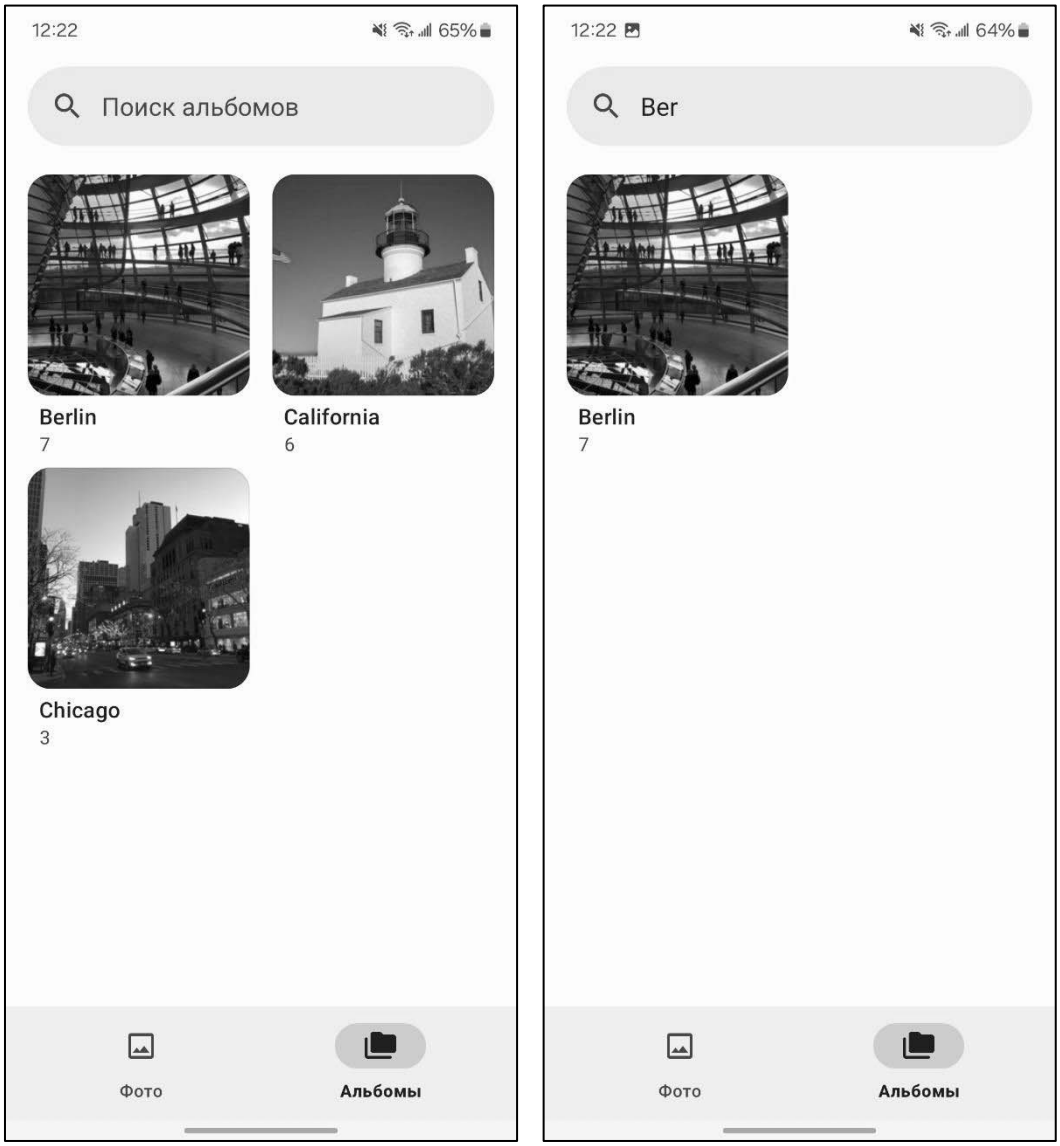

Рисунок 19 – Список альбомов

В соответствии с принятой архитектурой, модуль «Альбомы» состоит из трех слоев – представления, доменного и инфраструктурного. Далее следует рассмотрение состава каждого из этих слоев по отдельности.

Доменный слой содержит описание моделей и бизнес-логику. К моделям относятся следующие классы.

1. `Album` – это класс данных, который хранит основную информацию об альбоме: уникальный идентификатор, заголовок, описание, количество элементов и признак того, что альбом добавлен в «избранное».

2. `AlbumSearchQuery` – это класс данных, который хранит запрос на поиск альбомов, а также дополнительные параметры сообщающие, например, о том, что нужно полностью очистить кэш поисковых запросов и создать его заново.

К бизнес-логике данного модуля относятся классы, реализующие различные варианты использования данного модуля другими компонентами приложения:

1) `GetSearchResultUseCase` – класс, который позволяет получить постранично результаты поиска в виде Kotlin Flow;

2) `GetAlbumUseCase` – класс, который позволяет получить основную информацию об альбоме (экземпляр `Album`) по его уникальному идентификатору;

3) `GetThumbnailUrlUseCase` – класс, который позволяет сгенерировать URL-адрес миниатюры обложки альбома.

Каждый вариант использования обращается к репозиторию `AlbumRepository`, реализация которого относится к инфраструктурному слою. Данный репозиторий позволяет выполнять поиск альбомов с кэшированием в локальной базе данных и получать основную информацию об альбоме.

Для эффективного использования ресурсов смартфона при загрузке длинных списков была задействована библиотека Paging 3. Она позволяет достаточно просто реализовать постраничную загрузку списков, предоставляя удобные для этого инструменты.

Логика работы постраничной загрузки почти такая же, как и в модуле «Галерея» (раздел 3.5). Отличается лишь класс, ответственный за загрузку

данных из сети и сохранение их в базу данных – в данном модуле это `AlbumRemoteMediator`.

Работа с базой данных организована посредством библиотеки Room на основании ранее разработанной схемы. Реализация аналогична таковой у модуля «Галерея».

Слой представления построен на базе паттерна TEA и реализует все классы, необходимые для функционирования экрана с перечнем альбомов.

Данный экран выполняет сразу две задачи, а именно – постраничное отображение списка (компонент Albums) и предоставление пользовательского интерфейса для установки параметров поиска (компонент AlbumsSearch). С целью облегчения разработки и дальнейшей поддержки он, как и модуль «Галерея», был разделен на соответствующие компоненты, каждый из которых построен с использованием паттерна TEA. Рассмотрим их подробно.

Компонент Albums

Компонент для постраничного отображения списка состоит из следующих классов и интерфейсов.

1. `AlbumStore` – центральный класс, который управляет состоянием компонента и обрабатывает события.
2. `AlbumEvent` – общий маркерный интерфейс для всех событий, связанных с постраничной загрузкой.
3. `AlbumUiEvent` – маркерный интерфейс для событий пользовательского интерфейса.
4. `AlbumCommandResult` – маркерный интерфейс для событий, представляющих собой результат асинхронных команд.
5. `AlbumNews` – маркерный интерфейс для событий, представляющих единоразовое действие.
6. `AlbumCommand` – маркерный интерфейс для событий, представляющих собой асинхронные команды.
7. `AlbumState` – класс данных для хранения состояния списка.

8. `AlbumUpdate` – класс для обновления состояния списка в зависимости от текущего события. Код класса приведен в листинге 7.

9. `AlbumUiState` – класс для хранения состояния списка в виде, непосредственно пригодном для отображения на экране.

10. `AlbumUiStateMapper` – класс для преобразования внутреннего состояния `AlbumState` в состояние пользовательского интерфейса.

11. `AlbumFragment` – это фрагмент, непосредственно отвечающий за отображение списка альбомов. Он также ответственен за отображение интерфейса поиска.

12. `AlbumViewHolderFactory` – класс, необходимый для отображения списка альбомов в виде набора элементов типа `ViewTyped` (библиотека TiRecycler) с использованием RecyclerView (компонент Android).

Полный перечень всех событий, которые участвуют в управлении состоянием экрана в обычном режиме (когда пользовательский интерфейс для ввода поисковых параметров скрыт) приведен в таблице 15. Код класса, ответственного за обработку данных событий, приведен в листинге 7.

Таблица 15 – Перечень событий компонента Albums

Название события	**Тип**	**Описание**
`OnClickAlbumItem`	`AlbumUiEvent`	Нажатие на элемент списка
`OnClickRestart`	`AlbumUiEvent`	Нажатие кнопки «Повторить» при ошибке загрузки
`OnLoadMore`	`AlbumUiEvent`	Сообщение о необходимости предпринять попытку загрузки следующей страницы
`OnPerformSearch`	`AlbumUiEvent`	Действие «инициализировать постраничный загрузчик с новыми параметрами»
`PerformSearchResult`	`AlbumCommandResult`	Сообщение о текущем состоянии постраничной загрузки
`PerformSearchError`	`AlbumCommandResult`	Сообщение об ошибке постраничной загрузки
`OpenGalleryView`	`AlbumNews`	Действие «открыть просмотр содержимого альбома в виде галереи»

Листинг 7 – Класс `AlbumUpdate`

```
class AlbumUpdate @Inject constructor(
) : DslUpdate<AlbumState, AlbumEvent, AlbumCommand, AlbumNews>() {

    override fun NextBuilder.update(event: AlbumEvent) = when(event) {
        is OnClickAlbumItem -> news(OpenGalleryView(event.albumUid))
        is PerformSearchResult -> state {
            copy(listState = event.listState)
        }
        is PerformSearchError -> Unit
        is OnClickRestart -> commands(Restart)
        is OnLoadMore -> commands(LoadMore(event.position))
        is OnPerformSearch -> handleOnPerformSearch(event.query)
    }

    private fun NextBuilder.handleOnPerformSearch(
        query: AlbumSearchQuery
    ) {
        state { copy(listState = PagingState.initial()) }
        commands(PerformSearch(query))
    }
}
```

Для моделирования состояний постраничной загрузки списка используется универсальная запечатанная (sealed) иерархия классов, описание которой было приведено ранее в разделе 3.6.

Компонент AlbumsSearch

Компонент для отображения пользовательского интерфейса для задания параметров поиска состоит из следующих классов и интерфейсов.

1. `AlbumSearchEvent` – общий маркерный интерфейс для всех событий, связанных с поиском альбомов.

2. `AlbumSearchUiEvent` – маркерный интерфейс для событий пользовательского интерфейса.

3. `AlbumSearchCommandResult` – маркерный интерфейс для событий, представляющих собой результат асинхронных команд.

4. `AlbumSearchNews` – маркерный интерфейс для событий, представляющих единоразовое действие.

5. `AlbumSearchCommand` – маркерный интерфейс для событий, представляющих собой асинхронные команды.

6. `AlbumSearchState` – класс данных для хранения состояния поиска. Код класса приведен в листинге 8.

7. `AlbumSearchUiState` – класс для хранения состояния поиска в виде, непосредственно пригодном для отображения на экране. Код класса приведен в листинге 9.

8. `AlbumSearchUpdate` – класс для обновления состояния поиска в зависимости от текущего события.

9. `AlbumSearchUiStateMapper` – класс для преобразования внутреннего состояния `AlbumSearchState` в состояние пользовательского интерфейса.

Листинг 8 – Класс `AlbumSearchState`

```
data class AlbumSearchState(
    val album: Album? = null,
    val queryText: String = EMPTY_STRING,
    val currentQuery: AlbumSearchQuery? = null
)
```

Листинг 9 – Класс `AlbumSearchUiState`

```
class AlbumSearchUiState(
    val applyBtnEnabled: Boolean,
    val searchBarHint: String,
    val searchBarText: String,
    val searchViewText: String
)
```

Полный перечень всех событий, которые участвуют в управлении состоянием экрана в режиме поиска приведен в таблице 16. Код класса, ответственного за обработку данных событий и изменение состояния, приведен в листинге 10. Отдельно стоит отметить, что событие `PerformSearch` не запускает процесс поиска, а лишь уведомляет компонент Albums о таковом намерении пользователя (событие `OnPerformSearch`).

Таблица 16 – Список событий компонента AlbumsSearch

Название события	Тип	Описание
`OnSearchTextChanged`	`AlbumSearchUiEvent`	Изменение текста поискового запроса
`OnApplySearch`	`AlbumSearchUiEvent`	Нажатие кнопки «Применить»
`OnResetSearch`	`AlbumSearchUiEvent`	Нажатие кнопки «Сбросить»
`HideSearchView`	`AlbumSearchNews`	Действие «скрыть диалог с параметрами поиска»
`PerformSearch`	`AlbumSearchNews`	Действие «запустить поиск с заданными параметрами»

Листинг 10 – Класс `AlbumSearchUpdate`

```
class AlbumSearchUpdate @Inject constructor(
): DslUpdate<AlbumSearchState, AlbumSearchEvent, AlbumSearchCommand, Album-
SearchNews>() {

    override fun NextBuilder.update(event: AlbumSearchEvent) {
        when(event) {
            is OnApplySearch -> handleOnApplySearch()
            is OnResetSearch -> handleOnResetSearch()
            is OnSearchTextChanged -> state {
                copy(queryText = event.text.trim())
            }
        }
    }

    private fun NextBuilder.handleOnApplySearch() {
        news(AlbumSearchNews.HideSearchView)

        if (state.queryText != state.currentQuery?.value) {
            val query = AlbumSearchQuery(
                value = state.queryText,
                config = Config(
                    refresh = true
                )
            )

            state { copy(currentQuery = query) }
            news(AlbumSearchNews.PerformSearch(query))
        }
    }

    private fun NextBuilder.handleOnResetSearch() {
        state { copy(queryText = FULL_ALBUMS_QUERY) }
        news(AlbumSearchNews.HideSearchView)

        if (state.queryText != state.currentQuery?.value) {
            val query = AlbumSearchQuery(
                value = state.queryText,
                config = Config(
                    refresh = false
                )
            )

            state { copy(currentQuery = query) }
            news(AlbumSearchNews.PerformSearch(query))
        }
    }
}
```

4. ТЕСТИРОВАНИЕ

Для тестирования приложения применялось функциональное тестирование, то есть был создан набор тестов для проверки корректного выполнения функциональных требований. В таблицах 17–18 представлены наборы тестовых случаев для основных модулей – «Авторизация», «Галерея» и «Альбомы». Тестовые случаи для остальных модулей и режимов приведены далее.

Таблица 17 – Набор тестов модуля «Авторизация»

№	Тест	Действия	Результат
1.	Подключение к публичному серверу	1. Открыть экран авторизации. 2. В поле для ввода URL-адреса ввести адрес сервера, работающего в публичном режиме. 3. Нажать кнопку «Подключиться».	Приложение успешно подключилось к серверу. В настройках устройства отображается аккаунт с именем public@<домен>
2.	Подключение к частному серверу	1. Открыть экран авторизации. 2. В поле для ввода URL-адреса ввести адрес сервера, работающего в частном режиме. 3. В поля для ввода логина и пароля ввести корректные учетные данные. 4. Нажать кнопку «Подключиться».	Приложение успешно подключилось к серверу. В настройках устройства отображается аккаунт с именем <имя пользователя>@<домен>
3.	Валидация URL-адреса	1. Открыть экран авторизации. 2. В поле для ввода URL-адреса ввести некорректный URL-адрес, например «abc». 3. Нажать кнопку «Подключиться».	Поле для ввода URL-адреса сменило цвет на красный. Под полем появилась надпись «Неверный URL-адрес»
4.	Валидация пароля	1. Открыть экран авторизации. 2. В поле для ввода URL-адреса ввести корректный URL-адрес. 3. В поле для ввода имени пользователя ввести имя пользователя. 4. Поле для ввода пароля оставить пустым. 5. Нажать кнопку «Подключиться».	Поле для ввода пароля сменило цвет на красный. Под полем появилась надпись «Введите пароль тоже»
5.	Валидация имени пользователя	1. Открыть экран авторизации. 2. В поле для ввода URL-адреса ввести корректный URL-адрес. 3. Поле для ввода имени пользователя оставить пустым. 4. В поле для ввода пароля ввести любой пароль. 5. Нажать кнопку «Подключиться».	Поле для имени пользователя сменило цвет на красный. Под полем появилась надпись «Введите логин тоже»

Окончание таблицы 17

№	Тест	Действия	Результат
6.	Обработка ошибки подключения	1. Открыть экран авторизации. 2. В поле для ввода URL-адреса ввести корректный URL-адрес, на котором расположен не сервер PhotoPrism, например «https://google.com». 3. Нажать кнопку «Подключиться».	Поле для ввода URL-адреса сменило цвет на красный. Под полем появилась надпись «Не удалось подключиться»
7.	Обработка ошибки подключения	1. Открыть экран авторизации. 2. В поле для ввода URL-адреса ввести корректный URL-адрес, на котором расположен сервер PhotoPrism, работающий в частном режиме. 3. В поля для ввода учетных данных ввести некорректные данные. 4. Нажать кнопку «Подключиться».	Поля для ввода учетных данных сменили цвет на красный
8.	Автоматическое продление сессии	1. Открыть экран авторизации. 2. В поле для ввода URL-адреса ввести адрес сервера, работающего в частном режиме. 3. В поля для ввода логина и пароля ввести корректные учетные данные. 4. Нажать кнопку «Подключиться». 5. Через строку команд администратора на сервере завершить текущую сессию.	На экране пользователя на непродолжительное время появился индикатор загрузки, после чего работа приложения возобновилась без необходимости повторной авторизации

Таблица 18 – Набор тестов модулей «Галерея» и «Альбомы»

№	Тест	Действия	Результат
1.	Отображение полного перечня медиафайлов	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism.	На главном экране (раздел «Фото») отображается полный перечень медиафайлов в виде галереи
2.	Поиск медиафайлов	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism. 2. Находясь на экране «Фото» нажать на поле для ввода поискового запроса. 3. Ввести запрос «cats». 4. Нажать на кнопку «Применить».	На главном экране (раздел «Фото») теперь отображаются только те медиафайлы, на которых присутствуют кошки
3.	Отображение полного перечня альбомов	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism. 2. Используя нижний бар навигации перейти на экран «Альбомы».	На экране с альбомами (раздел «Альбомы») отображается полный перечень альбомов

Окончание таблицы 18

№	Тест	Действия	Результат
4.	Просмотр содержимого альбома	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism. 2. Используя нижний бар навигации перейти на экран «Альбомы». 3. Нажать на альбом, содержащий как минимум один элемент.	На экране отображаются медиафайлы в виде галереи, принадлежащие выбранному ранее альбому

В таблицах 19–20 представлен набор тестовых случаев модуля «Просмотр», а также поиска в различных режимах.

Таблица 19 – Набор тестов модуля «Просмотр»

№	Тест	Действия	Результат
1.	Отображение	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism. 2. Нажать на любой медиафайл в галерее.	Открылся экран просмотра, на котором отображается указанный медиафайл в большем разрешении с оригинальным соотношением сторон.
2.	Пролистывание	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism. 2. Нажать на любой медиафайл в галерее. 3. Смахнуть открывшийся медиафайл вправо или влево.	Медиафайл перелистнулся на соседний. Теперь на экране отображается другой медиафайл вместо предыдущего, причем тот, который следует за ним в порядке, заданным фильтром по умолчанию.
3.	Смена конфигурации	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism. 2. Нажать на любой медиафайл в галерее. 3. Сменить ориентацию устройства, вызвав тем самым поворот экрана и смену конфигурации.	Экран просмотра отображается в другой ориентации с тем же медиафайлом, который был на экране до смены ориентации.
4.	Приближение	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism. 2. Нажать на медиафайл в галерее, представляющий собой статичное изображение. 3. Выполнить жест pinch-to-zoom.	Медиафайл отображается на экране на экране в приближении, при этом функциональность пролистывания в стороны работает как положено.

Таблица 20 – Набор тестов поиска

№	Тест	Действия	Результат
1.	Постраничная загрузка	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism. 2. Нажать на первый медиафайл. 3. На экране предпросмотра пролистывать элементы вправо до конца.	Отображается последний элемент списка, т.е. во время пролистывания была выполнена дозагрузка страниц.
2.	Поиск	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism; 2. Находясь на экране «Фото» нажать на поле для ввода поискового запроса; 3. Ввести запрос «cats»; 4. Нажать на кнопку «Применить». 5. После загрузки результата нажать на любой медиафайл в галерее.	Открылся экран просмотра, на котором отображается указанный медиафайл в большем разрешении с оригинальным соотношением сторон.
3.	Поиск с пролистыванием	1. Авторизоваться в приложении с использованием публичной демоверсии PhotoPrism; 2. Находясь на экране «Фото» нажать на поле для ввода поискового запроса; 3. Ввести запрос «cats»; 4. Нажать на кнопку «Применить». 5. После загрузки результата нажать на любой медиафайл в галерее. 6. Смахнуть открывшийся медиафайл вправо или влево.	Медиафайл перелистнулся на соседний. Теперь на экране отображается другой медиафайл заместо предыдущего, причем тот, который следует за ним в порядке, заданным поисковым запросом.

Помимо этого, были реализованы юнит-тесты для класса `DateItemsGrouper`, задача которого группировать медиафайлы в списке по дате. Для этого были созданы два класса: `DateItemsGrouperTest` и `DateItemsGrouperTestData`. Первый класс содержит общую логику запуска теста, а второй – непосредственно тест кейсы. В приложении В приведена часть класса `DateItemsGrouperTestData` с примером юнит-теста для списка из одного элемента.

На рисунке 20 представлен результат запуска юнит-тестов. Как можно видеть, все 64 теста пройдены, а значит группировка элементов в списке по дате работает правильно.

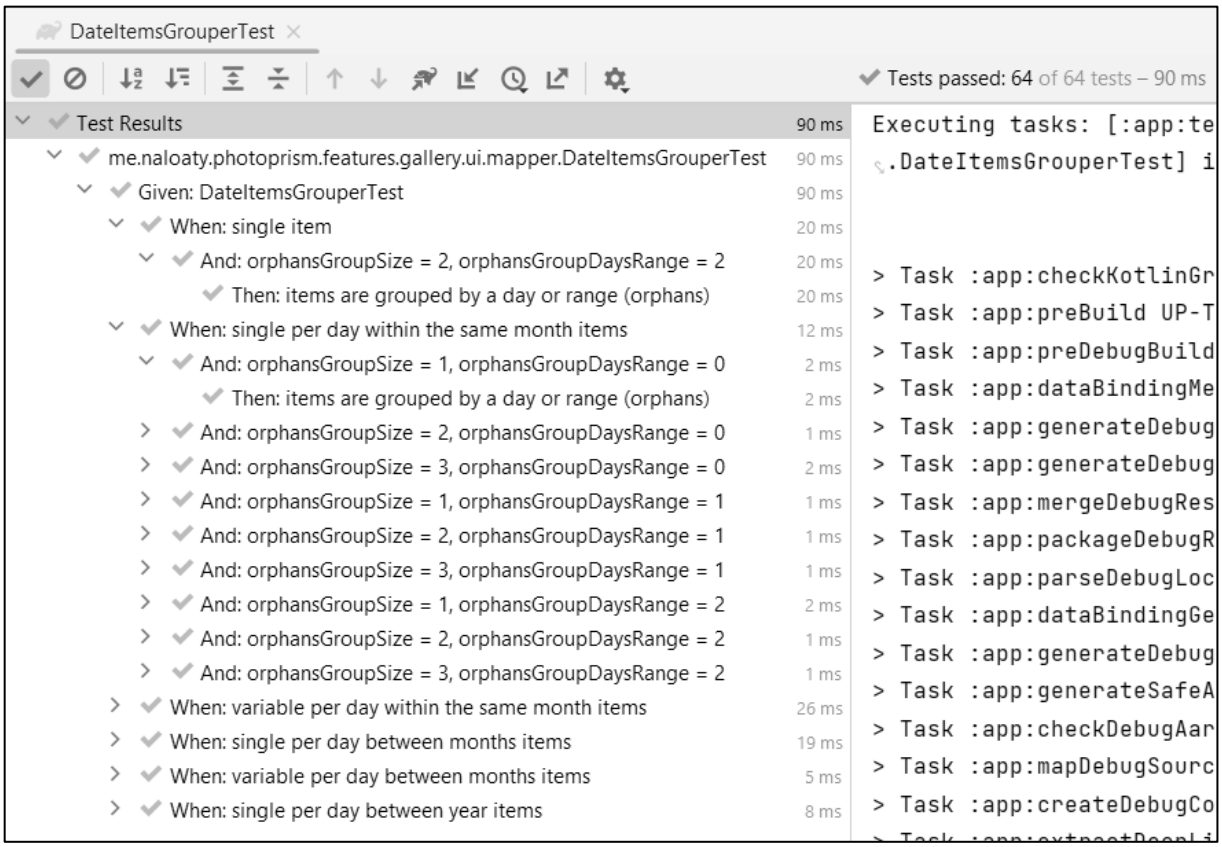

Рисунок 20 – Результат юнит-тестирования

Отдельно стоит отметить, что работа приложения была проверена на различных версиях операционной системы Android с использованием эмулятора Android Studio. Перечень конфигураций, на которых была проведена проверка, приведен в таблице 21. По результатам проверки проблемы в работе приложения на различных конфигурациях устройств не обнаружены.

Таблица 21 – Конфигурации виртуальных устройств

Разрешение экрана	**Плотность пикселей**	**Версия Android**	**Уровень API**
1080x2220	440dpi	8.0 (Google Play)	26
1080x2220	440dpi	9.0 (Google Play)	28
1440x3120	560dpi	10.0 (Google Play)	29
1080x2340	440dpi	11.0 (Google APIs)	30
1080x2340	440dpi	12.0 (Google APIs)	31
1080x2400	420dpi	13.0 (Google Play)	33

Таким образом, по результатам проведенного ручного и автоматизированного тестирования можно сделать вывод о том, что разработанная система соответствует требованиям.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано Android-приложение для сервиса PhotoPrism на Kotlin. В частности, были решены следующие задачи:

1) проведен анализ предметной области;
2) спроектировано мобильное приложение;
3) реализовано мобильное приложение;
4) проведено тестирование разработанного приложения.

В ходе работы были изучены различные подходы к разработке и проектированию приложений для платформы Android, освоено и применено на практике современное инструментальное программное обеспечение, изучен язык программирования Kotlin, задействованы различные прикладные библиотеки.

В дальнейшем работа будет направлена на доработку и расширение функционала, а именно:

1) интеграция с операционной системой с целью предоставления доступа к медиафайлам другим приложениям;
2) улучшение механизма кэширования и оффлайн поиска;
3) возможность быстрого переключения между несколькими аккаунтами и экземплярами PhotoPrism.

Помимо этого, планируется публикация приложения в магазины Google Play, RuStore и F-Droid.

ЛИТЕРАТУРА

1. Browse Your Life in Pictures – PhotoPrism. [Электронный ресурс] URL: https://www.photoprism.app/ (дата обращения: 19.03.2024 г.).

2. Катунин Г.П. Основы мультимедийных технологий: Учебное пособие для вузов. 3-е изд. // Санкт-Петербург: Лань, 2023. – 784 с.

3. Бураков Д.П. Основы хранения данных: учебное пособие. // Санкт-Петербург: ПГУПС, 2022. – 60 с.

4. Маркелов А.А. Введение в технологию контейнеров и Kubernetes. // Москва: ДМК Пресс, 2019. – 194 с.

5. Flutter App for PhotoPrism. [Электронный ресурс] URL: https://github.com/thielepaul/photoprism-mobile (дата обращения: 19.03.2024 г.).

6. MobilePrism – A Mobile App for PhotoPrism. [Электронный ресурс] URL: https://github.com/bleibdirtroy/MobilePrism (дата обращения: 19.03.2024 г.).

7. PhotoPrism Gallery. [Электронный ресурс] URL: https://github.com/Radiokot/photoprism-android-client (дата обращения: 19.03.2024 г.).

8. Android’s Kotlin-first approach | Android Developers. [Электронный ресурс] URL: https://developer.android.com/kotlin/first (дата обращения: 12.02.2024 г.).

9. Android Package Index | Android Developers. [Электронный ресурс] URL: https://developer.android.com/reference/packages (дата обращения: 12.02.2024 г.).

10. Explore the Jetpack libraries by type | Android Developers. [Электронный ресурс] URL: https://developer.android.com/jetpack/androidx/explorer (дата обращения: 26.02.2024 г.).

11. Flutter – Build apps for any screen. [Электронный ресурс] URL: https://flutter.dev/ (дата обращения: 12.02.2024 г.).

12. React Native – Learn once, write anywhere. [Электронный ресурс] URL: https://reactnative.dev/ (дата обращения: 12.02.2022 г.).

13. Stack Overflow Developer Survey 2022 | Stack Overflow. [Электронный ресурс] URL: https://survey.stackoverflow.co/2022/#most-popular-technologies-misc-tech (дата обращения: 26.02.2024 г.).

14. Dart programming language | Dart. [Электронный ресурс] URL: https://dart.dev/ (дата обращения: 26.02.2024 г.).

15. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. // Москва: ДМК Пресс, 2008. – 496 с.

16. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. // Санкт-Петербург: Питер, 2018. – 352 с.

17. ViewModel | Android Developers. [Электронный ресурс] URL: https://developer.android.com/topic/libraries/architecture/viewmodel (дата обращения: 12.03.2024 г.).

18. Крючкова Е.Н., Старолетов С.М. Объектно-ориентированное программирование: Архитектурное проектирование и паттерны программирования. // Барнаул: АлтГТУ, 2020. – 180 с.

19. The Elm Architecture | An Introduction to Elm. [Электронный ресурс] URL: https://guide.elm-lang.org/architecture/ (дата обращения: 20.02.2024 г.).

20. Разбираем ELM архитектуру в рамках мобильного приложения | Хабр. [Электронный ресурс] URL: https://habr.com/ru/companies/vivid_money/articles/550932/ (дата обращения: 05.04.2024 г.).

21. Paging library overview | Android Developers. [Электронный ресурс] URL: https://developer.android.com/topic/libraries/architecture/paging/v3-overview (дата обращения: 23.02.2024 г.).

22. Синицын И.В., Воронцов Ю.А., Михайлова Е.К. Встраиваемые системы управления базами данными для мобильных приложений: учебное пособие. // Москва: РТУ МИРЭА, 2022. – 529 с.

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–7.

Таблица 1 – Спецификация ВИ «Авторизация»

Прецедент: Авторизация
Краткое описание: Авторизация в приложении
Главные актеры: Неавторизованный пользователь
Второстепенные актеры: Авторизованный пользователь
Предусловия: Неавторизованный пользователь находится на экране авторизации.
Основной поток: 1. Прецедент начинается, когда неавторизованный пользователь заходит на экран авторизации. 2. Неавторизованный пользователь вводит URL-адрес сервера PhotoPrism, логин и пароль. 3. Неавторизованный пользователь нажимает на кнопку «Войти». 4. Приложение производит проверку полей. 5. Приложение производит авторизацию пользователя.
Постусловия: 1. Авторизованный пользователь попадает на главный экран.
Альтернативные потоки: I. Ошибка валидации. Если пользовать заполнил поля не корректно, то приложение сообщает об этом ему. Поток начинается заново с шага 2. II. Ошибка авторизации. В случае ошибки авторизации, приложение сообщает об этом пользователю и предлагает повторить попытку с шага 2.

Таблица 2 – Спецификация ВИ «Поиск медиафайлов с использованием фильтров»

Прецедент: Поиск медиафайлов с использованием фильтров
Краткое описание: Поиск в медиа-библиотеке медиафайлов согласно заданным критериям
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь нажал на строку поиска; 2. Пользователь ввел в поле для ввода поисковой запрос.

Продолжение таблицы 2 приложения А

Основной поток: 1. Прецедент начинается, когда пользователь нажимает кнопку «Применить». 2. Приложение выполняет поиск медиафайлов, соответствующих заданному ключевому слову. 3. Приложение выводит на экран найденные медиафайлы.
Постусловия: На экране отображаются результаты поиска.
Альтернативные потоки: I. Ошибка поиска. Если во время поиска происходит ошибка, то приложение отображает соответствующее сообщение. При этом приложение выводит на экран результаты поиска из кэша, если ранее производился аналогичный запрос. Пользователь может предпринять повторную попытку поиска, начиная с шага 1. II. Пустой результат поиска. Если соответствующие поисковому запросы медиафайлы не найдены, то приложение выводит на экран соответствующее сообщение. Пользователь может предпринять повторную попытку поиска, начиная с шага 1.

Таблица 3 – Спецификация ВИ «Просмотр списка альбомов»

Прецедент: Просмотр списка альбомов
Краткое описание: Просмотр списка альбомов
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь находится на экране для просмотра списка альбомов.
Основной поток: 1. Прецедент начинается, когда пользователь открывает экран для просмотра списка альбомов. 2. Приложение загружает список альбомов. 3. Приложение отображает на экране результат загрузки альбомов.
Постусловия: На экране отображается результат загрузки списка альбомов.
Альтернативные потоки: I. Ошибка загрузки. Если во время загрузки происходит ошибка, то приложение отображает соответствующее сообщение. При этом приложение выводит на экран список альбомов из кэша (если присутствует в нем). Пользователь может предпринять повторную попытку загрузки альбомов, начиная с шага 1. II. Пустой список альбомов. Если список альбомов оказался пустым, то приложение выводит на экран соответствующее сообщение. Пользователь может предпринять повторную попытку загрузки альбомов, начиная с шага 1.

Таблица 4 – Спецификация ВИ «Просмотр полного перечня медиафайлов»

Прецедент: Просмотр полного перечня медиафайлов
Краткое описание: Просмотр всех файлов, находящихся в медиа-библиотеке
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь находится на главном экране (экране для просмотра всей медиа-библиотеки)
Основной поток: 1. Прецедент начинается, когда пользователь переходит на главный экран. 2. Приложение загружает полный список медиафайлов из медиа-библиотеки. 3. Приложение отображает на экране загруженный список медиафайлов.
Постусловия: На экране отображается результат загрузки всей медиа-библиотеки.
Альтернативные потоки: I. Ошибка загрузки. Если во время загрузки списка медиафайлов происходит ошибка, то приложение отображает соответствующее сообщение. При этом приложение выводит на экран список медиафайлов из кэша (если присутствует в нем). Пользователь может предпринять повторную попытку загрузки альбомов, нажав на кнопку «retry». II. Пустой список медиафайлов. Если список медиафайлов оказался пустым, то приложение выводит на экран соответствующее сообщение. Пользователь может предпринять повторную попытку загрузки альбомов, нажав на кнопку «refresh».

Таблица 5 – Спецификация ВИ «Просмотр содержимого альбома»

Прецедент: Просмотр содержимого альбома
Краткое описание: Просмотр содержимого альбома
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь на экране для просмотра списка альбомов и нажал на один из альбомов. 2. После действия 1 пользователь находится на экране для просмотра содержимого альбома.
Основной поток: 1. Прецендент начинается, когда пользователь попадает на экран для просмотра содержимого альбома. 2. Приложение загружает список медиафайлов, принадлежащих данному альбому. 3. Приложение отображает на экране загруженный список медиафайлов.
Постусловия: На экране отображается результат загрузки содержимого альбома.

Продолжение таблицы 5 приложения А

Альтернативные потоки: I. Ошибка загрузки. Если во время загрузки содержимого альбома происходит ошибка, то приложение отображает соответствующее сообщение. При этом приложение выводит на экран содержимое альбома из кэша (если присутствует в нем). II. Пустой список медиафайлов. Если список медиафайлов оказался пустым, то приложение выводит на экран соответствующее сообщение. Пользователь может предпринять повторную попытку загрузки альбомов, нажав на кнопку «обновить».

Таблица 6 – Спецификация ВИ «Поиск альбомов с использованием фильтров»

Прецедент: Поиск альбомов с использованием фильтров
Краткое описание: Поиск в медиа-библиотеке альбомов согласно заданным критериям
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь нажал на строку поиска; 2. Пользователь ввел в поле для ввода поисковой запрос.
Основной поток: 1. Прецедент начинается, когда пользователь нажимает кнопку «Применить». 2. Приложение выполняет поиск альбомов, соответствующих заданному ключевому слову. 3. Приложение выводит на экран найденные альбомы.
Постусловия: На экране отображаются результаты поиска.
Альтернативные потоки: I. Ошибка поиска. Если во время поиска происходит ошибка, то приложение отображает соответствующее сообщение. При этом приложение выводит на экран результаты поиска из кэша, если ранее производился аналогичный запрос. Пользователь может предпринять повторную попытку поиска, начиная с шага 1. II. Пустой результат поиска. Если соответствующие поисковому запросы альбомы не найдены, то приложение выводит на экран соответствующее сообщение. Пользователь может предпринять повторную попытку поиска, начиная с шага 1.

Таблица 7 – Спецификация ВИ «Просмотр медиафайла в деталях»

Прецедент: Просмотр медиафайла в деталях
Краткое описание: Просмотр медиафайла на весь экран в высоком разрешении
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Нет
Предусловия: 1. Пользователь на главном экране или на экране для просмотра содержимого альбома и нажал на один из элементов в списке медиафайлов. 2. После действия 1 пользователь находится на экране для просмотра медиафайла в деталях.
Основной поток: 1. Прецендент начинается, когда пользователь попадает на экран для просмотра медиафайла в деталях. 2. Приложение загружает полноразмерную версию медиафайла. Во время загрузки на месте загружаемого медиафайла отображается миниатюра из кэша. 3. Приложение отображает на экране загруженную полноразмерную версию медиафайла.
Постусловия: На экране отображается результат загрузки полноразмерной версии медиафайла.
Альтернативные потоки: I. Ошибка загрузки. Если во время загрузки полноразмерной версии медиафайла происходит ошибка, то приложение отображает соответствующее сообщение. При этом приложение выводит на экран миниатюру из кэша (если присутствует в нем).

Приложение Б. Диаграмма деятельности формы авторизации

Диаграмма деятельности, описывающая процесс ввода и валидации данных формы авторизации, приведена на рисунке 1.

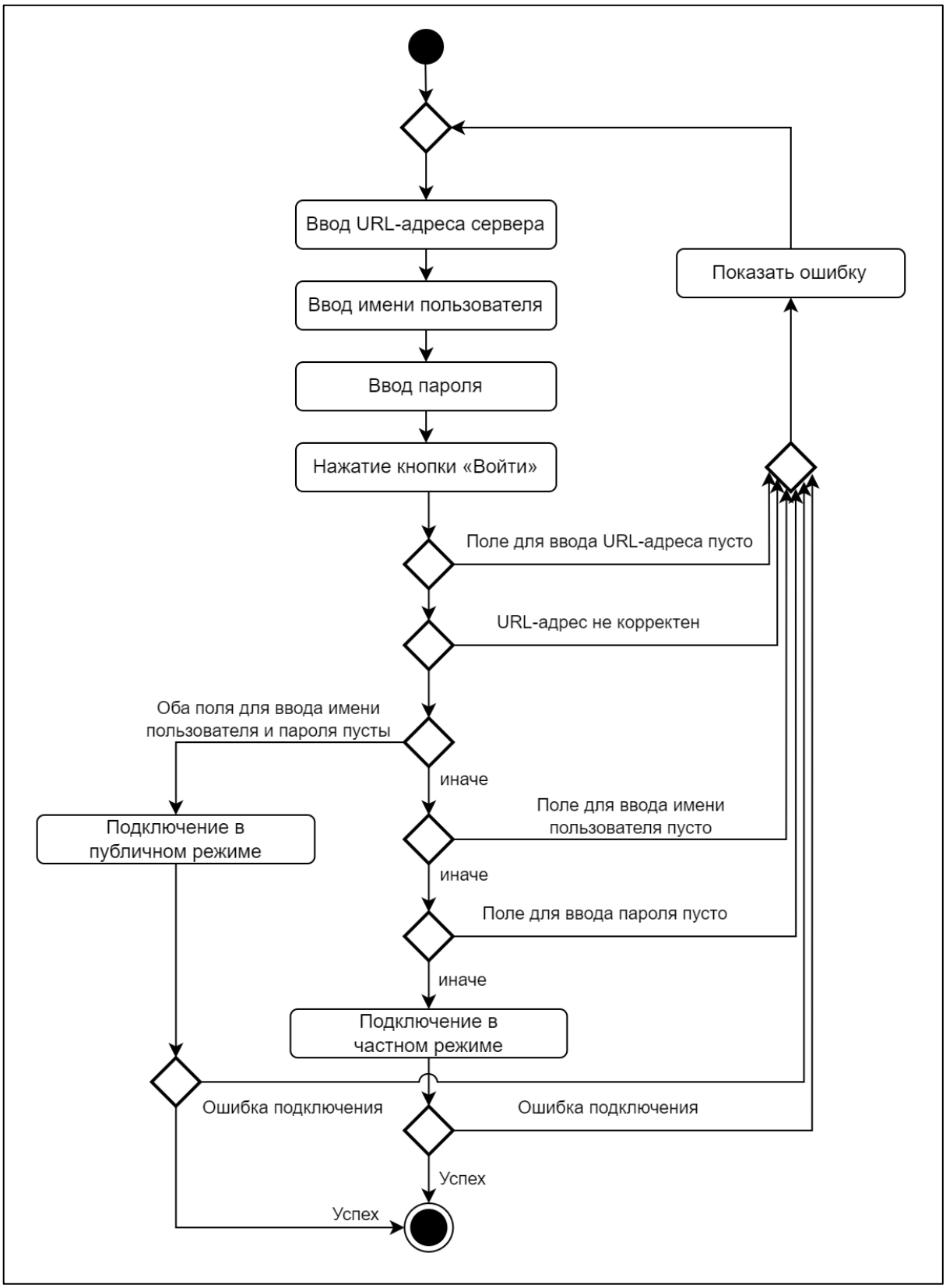

Рисунок 1 – Диаграмма деятельности формы авторизации

Приложение В. Пример юнит-теста

В листинге 1 приведен юнит-тест для класса `DateItemsGrouper` на примере списка из одного элемента.

Листинг 1 – Пример юнит-теста

```
class DateItemsGrouperTestData {
    var testGroups = listOf<Pair<String, List<TestCase>>>()
    val singleItem = listOf(
        date(year = 2024, month = 6, day = 1, hour = 10, minute = 10))
    // Результатом работы класса DateItemsGrouper является список вида:
    // [группа1S, группа1E/группа2S, ..., группа[N-1]E/группаNS, группа NE)
    // где:
    // группаNS – индекс первого элемента группы (включительно),
    // группаNE – индекс последнего элемента группы (не включительно).
    val singleItemTestCases = listOf(
        grouper(2, 2).test(singleItem to listOf(0, 1)),
    ) registerAs "Single item"
    ...
    // Вспомогательные функции для формирования тестовых данных
    private fun date(
        year: Int, month: Int, day: Int, hour: Int, minute: Int
    ) = Calendar.getInstance().apply {
        set(year, month - 1, day, hour, minute)
    }

    private fun grouper(
        orphansGroupSize: Int, orphansGroupDaysRange: Int
    ): TestCase.GrouperParams {
        return TestCase.GrouperParams(
            orphansGroupSize, orphansGroupDaysRange)
    }

    private fun TestCase.GrouperParams.test(
        testCase: Pair<List<Calendar>, List<Int>>
    ) = TestCase(this, testCase.first, testCase.second)


    private infix fun List<TestCase>.registerAs(
        name: String
    ): List<TestCase> {
        testGroups += name to this
        return this
    }

    class TestCase(
        val params: GrouperParams,
        val itemsToGroup: List<Calendar>,
        val expectedGroups: List<Int>,
    ) {
        data class GrouperParams(
            // Макс. размер группы, состоящей из одиночных элементов
            val orphansGroupSize: Int,
            // Временное окно, в рамках которого идет группировка
            val orphansGroupDaysRange: Int,
        )
    }
}
```