Санкт–Петербургский государственный университет
Факультет математики и компьютерных наук

Вадим Маратович Салаватов

Выпускная квалификационная работа

Реализация мультиплатформенного доступа к файловым хранилищам на языке Kotlin

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2018 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Современное программирование»

Научный руководитель:
профессор, д.ф.-м.н. А. С. Куликов

Рецензент:
девелопер-адвокат ООО «ИнтеллиДжей Лабс»
А. А. Архипов

Санкт-Петербург
2022 г.

Содержание

Введение

(TODO: Введение широко представляет предметную область работы, указывает на место работы в научном или технологическом контексте.)

О чем можно сказать, наверное: котлин, мультиплатформа, single code base, time to market, преимущества (время, деньги, ошибки, ...), разные платформы и их ограничения....

Актуальность работы. [9, 32, 27, 20, 26]

Структура работы. **(TODO: где что в какой главе)**

Постановка задачи

Цель работы состоит в разработке мультиплатформенной библиотеки на языке программирования Kotlin, позволяющей разработчику описывать логику работы с различными файловыми хранилищами в общем модуле мультиплатформенного проекта. Библиотека должна предоставлять интерфейс (виртуальной) файловой системы с иерархической структурой папок и файлов и позволять записывать и читать файлы как массивы байт. Библиотека должна быть достаточно гибкой, чтобы разработчик мог самостоятельно дополнить её функциональность (например, поддержать на базе библиотеки новое облачное хранилище), а также иметь возможность расширения на уровне предоставляемых интерфейсов, когда от целевых хранилищ требуется поддержка особых возможностей (например, наличие у файлов атрибутов прав на чтение/запись).

От конечного продукта ожидается как минимум:

- поддержка трех платформ: JVM (для приложений, работающих в среде операционных систем Windows, Linux, и т.д.), JS (браузерные приложения), Android (мобильные приложения);
- поддержка как минимум одного облачного хранилища, доступного со всех поддерживаемых платформ.

1. Обзор

1.1. Предметная область

Пояснение терминов
Котлин, Мультиплатформа, JS browser/node.js ...

1.2. Существующие решения

Для поиска существующих решений были сделаны следующие запросы в поисковую систему Google:

- «kotlin multiplatform io»
- «kotlin multiplatform filesystem»
- «kotlin multiplatform file»
- «kotlin multiplatform storage»

Аналогичные запросы были сделаны и на русском языке, но никаких дополнительных результатов это не принесло.

Также был произведен поиск по релевантным теме вопросам на портале StackOverflow[34]. Несколько релевантных вопросов нашлось[9, 32], но дополнительных результатов они не принесли. Дополнительно был произведен поиск по репозиториям на портале GitHub[12]. Кроме решений, перечисленных ниже, был обнаружен курируемый сообществом репозиторий со списком Kotlin Multiplatform библиотек[19].

Список полученных релевантных решений можно разделить на три категории.

1.2.1 Решения, предоставляющие интерфейс виртуальных файловых систем

- **kile**[14]

  Последнее изменение кода — июль 2020 г., сайт с документацией недоступен. Из кода ясно, что (возможно) поддерживаются платформы JVM,

JS (Node.js). Из особенностей: нет методов для чтения и записи файлов; конкретные хранилища подключаются через адаптеры; есть адаптер для FTP[10]-сессии, некоторые методы не реализованы, что-то потенциально работающее написано только для платформы JVM; есть адаптер для локальной файловой системы (аналогично реализация есть только для JVM).

- **files**[4]

  Последнее изменение кода — ноябрь 2020 г., в репозитории есть минимальный сопровождающий текст. Из кода ясно, что (возможно) поддерживаются платформы JVM, Android, JS (browser). Предоставляется мультиплатформенный интерфейс файла, но только с возможностью чтения. Судя по коду, данное решение позволяет запрашивать у пользователя загрузку файла в веб-браузере.

- **kotlinx-fs**[21]

  Последнее изменение кода — февраль 2019 г., в репозитории есть минимальный сопровождающий текст. Заявлена поддержка JVM, JS (Node.js) и POSIX-совместимых операционных систем (Mac OS X и Linux) для нативных Kotlin-приложений. Предоставляется интерфейс локальной файловой системы (проверка существования пути, чтение атрибутов путей, создание файлов и папок, в том числе временных, перемещение и удаление путей, чтение и запись файлов) и его реализации на перечисленных платформах.

- **supernatural-fs**[35]

  Последнее изменение кода — декабрь 2020 г., есть минимальная документация. Предоставляется мультиплатформенный интерфейс локальной файловой системы, доступный на платформах Android, iOS, JS (Node.js), JVM.

- **korio**[16]

  Последнее изменение — август 2021 г., есть документация. Поддерживает очень много платформ: помимо Android, JVM, JS (browser), ещё и

iOS, Mac OS X, Linux x64, watchOS, tvOS и другие. Библиотека является частью большого проекта (множества библиотек) Korlibs[17], берущего начало в 2017 году и направленного на создание мультиплатформенного движка для видеоигр и сопутствующих мультиплатформенных библиотек. В связи с этим, данные библиотеки сильно завязаны друг на друга, и, вероятно, разрабатывались для написания приложений именно в среде этого набора библиотек.

Библиотека представляет свои реализации TCP- и HTTP-клиентов, примитивы для работы с потоковой обработкой данных (`AsyncStream`), собственную сериализацию/десериализацию форматов JSON, YAML, XML.

Есть абстракция виртуальной файловой системы `Vfs`, элементы иерархии — `VfsFile`, то есть нет различия на уровне типов между папками и файлами, но есть методы для определения этого (`isDirectory()`), поддерживаются атрибуты файлов, наблюдение за событиями файловой системы (`watch(path: String, handler: (FileEvent) -> Unit)`). Есть методы для чтения и записи файлов как целиком, так и в потоковом режиме (используя `AsyncStream`).

Среди доступных реализаций `Vfs` можно отметить `ZipVfs` — виртуальную файловую систему (ВФС) для чтения zip-архивов, `UrlVfs` — ВФС для чтения и записи ресурсов, расположенных в сети Интернет, по протоколу HTTP. Чтение доступно в потоковом режиме, запись — только целиком; реализация предполагает, что данные можно записать, отправив PUT-запрос с содержимым на удаленный сервер. Есть также `LocalVfs` для работы с локальным файловым хранилищем с несколькими реализациями для разных платформ.

1.2.2 Key-value решения

Здесь перечислены найденные продукты с открытым исходным кодом, которые представляют собой мультиплатформенные библиотеки, позволяющие персистентно сохранять пары ключ-значение в локальном хранилище. Они не решают поставленную задачу или ее части, однако в некоторых слу-

чаях их функциональности может оказаться достаточно, поэтому они стоят упоминания.

Многие из этих проектов в качестве хранилищ используют `SharedPreferences` для Android, `window.localStorage` для JS (browser), `NSUserDefaults` для iOS.

- **multiplatform-settings**[29]

  Проект активно развивается, доступна документация. Помимо платформ JVM, JS (browser) и Android поддержаны также iOS, macOS, watchOS, tvOS и Windows. Предоставляет интерфейс для сохранения пар ключ-значение в локальном хранилище.

- **KVault**[25]

  Проект активно развивается, есть минимальная документация. Поддержаны только платформы iOS и Android, однако реализована поддержка шифрования данных.

- **Kissme**[15]

  Последнее изменение — январь 2020 г., есть минимальная документация. Аналогично предыдущему пункту, поддержаны платформы iOS и Android, реализовано шифрование данных.

- **multiplatform-preferences**[28]

  Проект заархивирован, последнее изменение — февраль 2020 г., есть минимальная документация. Поддержаны платформы iOS и Android.

- **Kotlin Data Storage**[18]

  Последнее изменение — октябрь 2021 г., есть минимальная документация. Поддержаны платформы JVM, JS (Node.js и browser), Android. Решение позволяет сохранять сериализуемые данные в локальных хранилищах (в т.ч. в файлы на платформе JVM). Благодаря использованию делегации свойств[7], работа с данными происходит в виде чтения и записи значений переменных.

- **asoft-storage**[5]

  Последнее изменение кода — апрель 2020 г., нет документации и примеров. Из кода ясно, что (возможно) поддерживаются платформы JVM, Android, JS (browser). Не реализована логика работы на JVM.

- **kached**[13]

  Последнее изменение кода — октябрь 2020 г., есть небольшой сопровождающий текст. Из кода ясно, что (возможно) поддерживаются платформы JVM, Android, iOS. Для JVM реализовано хранилище на базе файловой системы. Из особенностей: поддерживается шифрование значений, хранение данных в Amazon S3[1] (но доступно только на платформе JVM).

1.2.3 Решения для мультиплатформенного ввода-вывода (IO)

Здесь перечислены решения, предоставляющие возможности для мультиплатформенного ввода-вывода данных. На их базе гипотетически можно построить решение главной задачи.

- **korio**[16]

  См. раздел 1.2.1.

- **okio**[31]

  Проект активно развивается, доступна документация. **(TODO: описать FileSystem и возможности IO, sink, source)**

- **kotlinx-io**[22]

  Официальная библиотека для мультиплатформенного ввода-вывода. Последнее изменение кода — май 2020 г., в вопросе об актуальности библиотеки[23] был дан ответ, что в разработке новая версия этой библиотеки. По всей видимости, проект стал частью Ktor[24] и развивается внутри него.

- **tinlok**[36]

  (TODO:)

2. Презентация решения

В данной главе дан высокоуровневый обзор реализованной библиотеки `multifs`[30], а также обоснования принятых архитектурных решений.

2.1. Архитектура библиотеки

Ввиду того, что одна из главных задач заключается в организации поддержки файловых хранилищ на платформе JS (browser), то есть в веб-браузерах, возникает вопрос, какие файловые хранилища вообще можно на этой платформе поддержать? Поскольку в современных веб-браузерах много внимания уделяется безопасности, в них отсутствует доступ к файловой системе операционной системы, на которой они запущены. Для сохранения данных между сеансами можно пользоваться свойством `localStorage`[37] — оно позволяет сохранять данные в виде пар ключ-значение. Гипотетически можно эмулировать файловую систему через эту структуру данных. Однако объем `localStorage` часто ограничен веб-браузерами и невелик: например, веб-браузер Firefox версии 98.0 ограничивает объем `localStorage` примерно десятью мегабайтами памяти. Если представить, что мы пишем, скажем, онлайн-редактор документов в формате docx и pdf, поддерживающее к тому же картинки, то сохранить много документов в таком хранилище просто не выйдет.

Необходим доступ к «настоящему» файловому хранилищу, и в таком случае можно предложить следующие альтернативы локальной файловой системе, доступные из веб-браузеров:

- облачные хранилища, предоставляющие HTTP REST[33] API интерфейс для доступа к данным (Google Drive, Yandex.Disk, Dropbox, Amazon S3, и т.д.);
- подключение к удаленным файловым системам посредством FTP[10], как это сделано в проекте kile[14], или другим протоколам передачи файлов (SFTP, WebDAV, и т.д.).

Поскольку целью работы является также создание как можно более

гибкой библиотеки в плане расширения множества поддерживаемых файловых хранилищ, имеет смысл выяснение функциональности и особенностей, которые имеют современные файловые хранилища.

2.1.1 Обзор функциональности и особенностей файловых хранилищ

Для изучения выбраны несколько наиболее популярных онлайн-сервисов для хранения данных с публичным API: Google Drive, Яндекс.Диск, Dropbox, Amazon S3. Здесь перечисляются наиболее интересная (по мнению автора) функциональность и особенности, которые могут пригодится при решении поставленной задачи, более подробная информация содержится в официальной документации сервисов.

Google Drive[11]. Файлы и папки не отличаются с точки зрения представления (то есть всё является файлом), различаются полем MimeType (у папок он равен `application/vnd.google-apps.folder`). Каждый файл имеет уникальный идентификатор (ID), при этом не запрещается иметь несколько файлов с одинаковым именем в одной папке. Файлы могут помимо своего содержимого хранить произвольные свойства (пары ключ-значение), есть атрибуты времени создания, модификации и доступа, атрибуты прав доступа. Файлы могут принадлежать одному из трёх изолированных пространств (spaces): пространство диска (drive space) — основное пространство, в котором пользователь хранит свои файлы; пространство приложения (app data folder space) — пространство, доступное только приложению и не доступное пользователю; пространство фотографий (photos space). Среди методов доступны создание файлов, скачивание, загрузка (в том числе больших файлов по частям), копирование (не папок), получение и изменение метаданных отдельно от содержимого (можно таким образом перемещать файлы), получение списка файлов с указанием фильтров (например, по id родителя).

Яндекс.Диск[3]. Строгая иерархия каталогов и файлов, имена файлов уникальны и чувствительны к регистру. Доступна папка приложения, однако здесь пользователь имеет к ней прямой доступ. Файлы могут хранить произвольные

атрибуты (пары ключ-значение). Присутствуют аналогичные методы для выполнения операций, перечисленных выше. Стоит отметить, что загрузка файлов на сервер по частям недоступна, а также доступно копирование папок. Файлы имеют одного владельца, но можно делать файлы полностью публичными.

Dropbox[8]. Иерархия папок и файлов, имеющих уникальный идентификатор. Можно адресовать узлы как по пути от корня, так и по идентификаторам. Имена не чувствительны к регистру. Есть папки приложений. Поддерживаются произвольные атрибуты (пары ключ-значение). Доступна аналогичная функциональность как и у Яндекс.Диска, но доступна загрузка по частям. Есть система контроля доступа к файлам с разной гранулярностью (пользователи и группы).

Amazon S3[2]. Данные хранятся в объектах (objects), расположенных в контейнерах (buckets). Объекты представляют собой содержимое файла и метаданные. Формально контейнер является плоской структурой (т.е. он хранит все объекты без иерархии), но можно организовать иерархию файлов и папок на логическом уровне, используя разделители (/) и общие префиксы для файлов, находящихся в одной директории. Есть система управления доступом к объектам, загрузка по частям, версионирование, произвольные атрибуты, копирование и другие методы, перечисленные выше.

Отдельно стоит отметить, что файловые системы наиболее популярных операционных систем для персональных компьютеров (Windows — NTFS, FAT; дистрибутивы Linux — ext4, btrfs, другие; Mac OS X — APFS, HFS+) тоже имеют свои особенности. Например, файловая система HFS+ по умолчанию настроена так, что имена не чувствительны к регистру. В Windows по умолчанию операции с файловой системой осуществляются в не чувствительном к регистру режиме[6]. В дистрибутивах Linux файловые системы чувствительны к регистру. Данную особенность нужно учитывать при разработке приложений для этих платформ.

2.1.2 Исследование вариантов организации архитектуры (TODO: поменять?)

Самое близкое к решению целевой задачи из уже существующего (в концептуальном смысле) — это упомянутый в обзоре проект korio[16]. В нем представлен базовый интерфейс виртуальной файловой системы как абстракции, которую должны реализовать все наследники на конкретных платформах. Однако этот интерфейс сильно перегружен: например, там есть методы для наблюдения за событиями файловой системы на определенном файле, но (в нашем случае) такую функциональность поддерживают не все облачные хранилища (например, Яндекс.Диск не поддерживает), также некоторые абстрактные методы имеют базовые реализации, которые ничего не делают (например, метод `touch`). При попытке поддержать конкретное файловое хранилище, может возникнуть ситуация, когда реализация части функциональности не будет возможна ввиду отсутствия такой функциональности у целевого хранилища. В результате, когда пользователь-разработчик будет пытаться использовать определенную функциональность виртуальной файловой системы в общем модуле мультиплатформенного проекта, у него не будет уверенности, что она вообще реализована.

Каких дополнительных свойств хочется добиться от конечной библиотеки:

- базовый интерфейс виртуальной файловой системы дает гарантию поддержки минимального набора операций, с помощью которого можно выполнять любые базовые операции над файлами (создание, удаление файлов и папок; запись, чтение, перемещение, копирование файлов) — это позволит быстро обеспечивать базовую поддержку любых хранилищ, которая нужна пользователю;

- поддержка особых возможностей обеспечивается через реализацию дополнительных интерфейсов-расширений виртуальной файловой системы.

Понятно, что для приложения может быть недостаточно лишь базовых операций для работы с файлами. Нужен инструмент для определения контракта по

необходимой функциональности конкретных файловых хранилищ, которые будут использоваться на целевых платформах. Для решения этой проблемы предлагается использовать мощь системы типов языка Kotlin, об этом будет рассказано позже.

Ввиду особенностей различных файловых хранилищ, перечисленных в разделе 2.1.1, предлагается отказаться от адресации узлов по полным путям и ввести на уровне типов отдельно папку и файл как узлы иерархии.

(TODO: что-то ещё?) (TODO: про симлинки?)

2.1.3 VFS

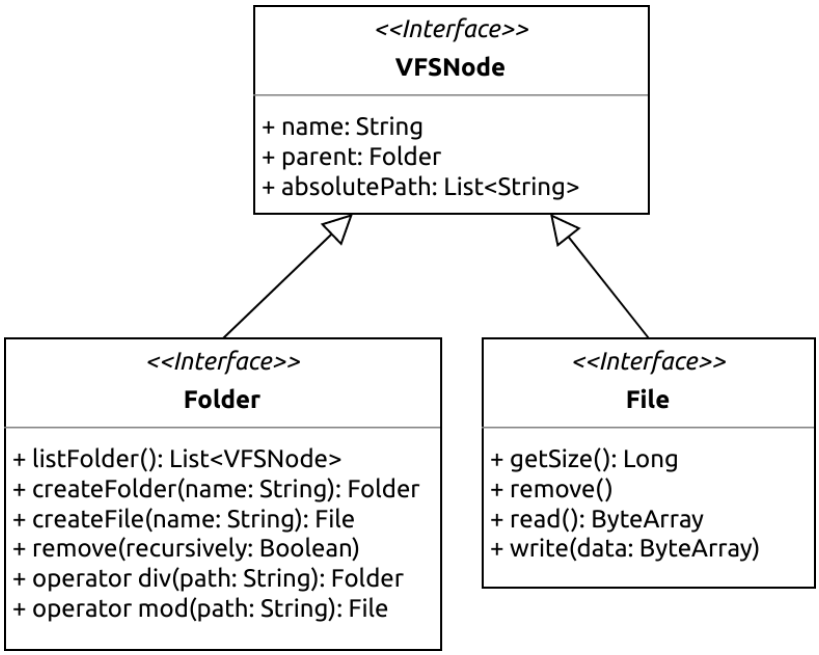

Рис. 1: Интерфейсы узлов виртуальной файловой системы.

Ссылаемся на график 1.

Ссылаемся на график 2.

2.1.4 Инварианты и гарантии

(TODO: ссылка на родителя)

(TODO: exception'ы и их ловля, метки)

(TODO: если папка не пуста, удалить ее нельзя без recursively)

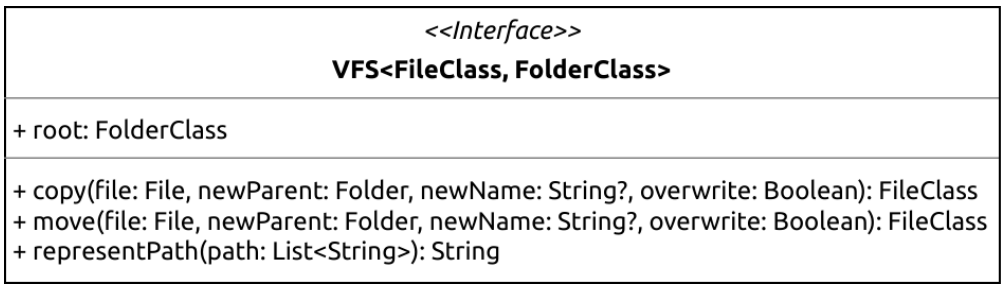

Рис. 2: Интерфейс виртуальной файловой системы.

(TODO: нет ограничения на имена файлов папок и тд, за этим должен следить пользователь-разраб)

2.1.5 Преимущества и недостатки выбранного подхода

(TODO: удобная и понятная навигация с выводом типов)

(TODO: интерфейс предоставляет только ту функциональность, которая на самом деле доступна)

(TODO: довольно емкий платформо-специфичный код (определение бекендов))

(TODO: не очень удобно записывать ограничения на типы, но вот если котлин сделает типы-пересечения, то станет норм)

(TODO: без тайпкастов не обойтись)

2.2. Хранилища, поддержанные библиотекой

2.2.1 `SystemFS`

Ссылаемся на график 3. **(TODO: обертка над `java.nio.Path`)**

2.2.2 `GoogleDriveFS`

(TODO: GoogleDriveAPI и сам GoogleDriveFS в общем коде, на платформах только Authorizer'ы) Ссылаемся на график 4.

2.2.3 `SqliteFS`

(TODO: адаптер к Sqlite на Андроиде)

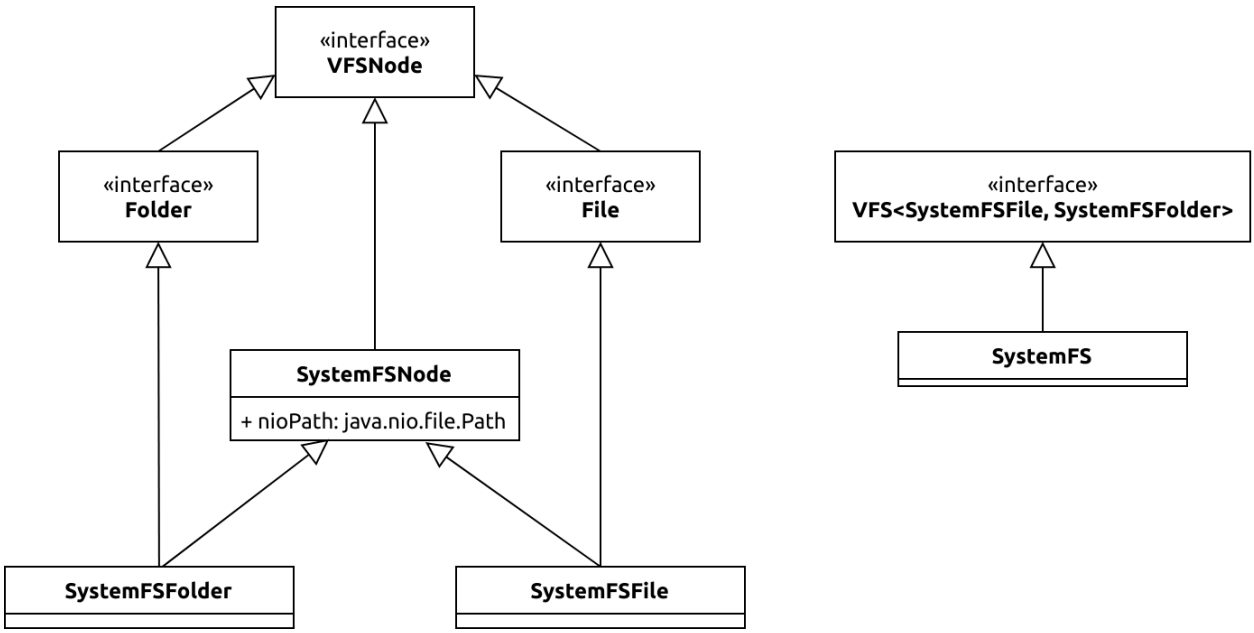

Рис. 3: **(TODO: SystemFS)**.

2.3. Используемые зависимости

2.3.1 Ktor

(TODO: нужен как мультиплатформенный HTTP клиент + из IO взял ByteChannel)

2.3.2 Зависимости на платформе Android

- activity-ktx **(TODO: предоставляет расширения java классов для языка котлин??)**
- play-services-auth **(TODO: тут лежит SignInIntent чтобы авторизацию производить)**

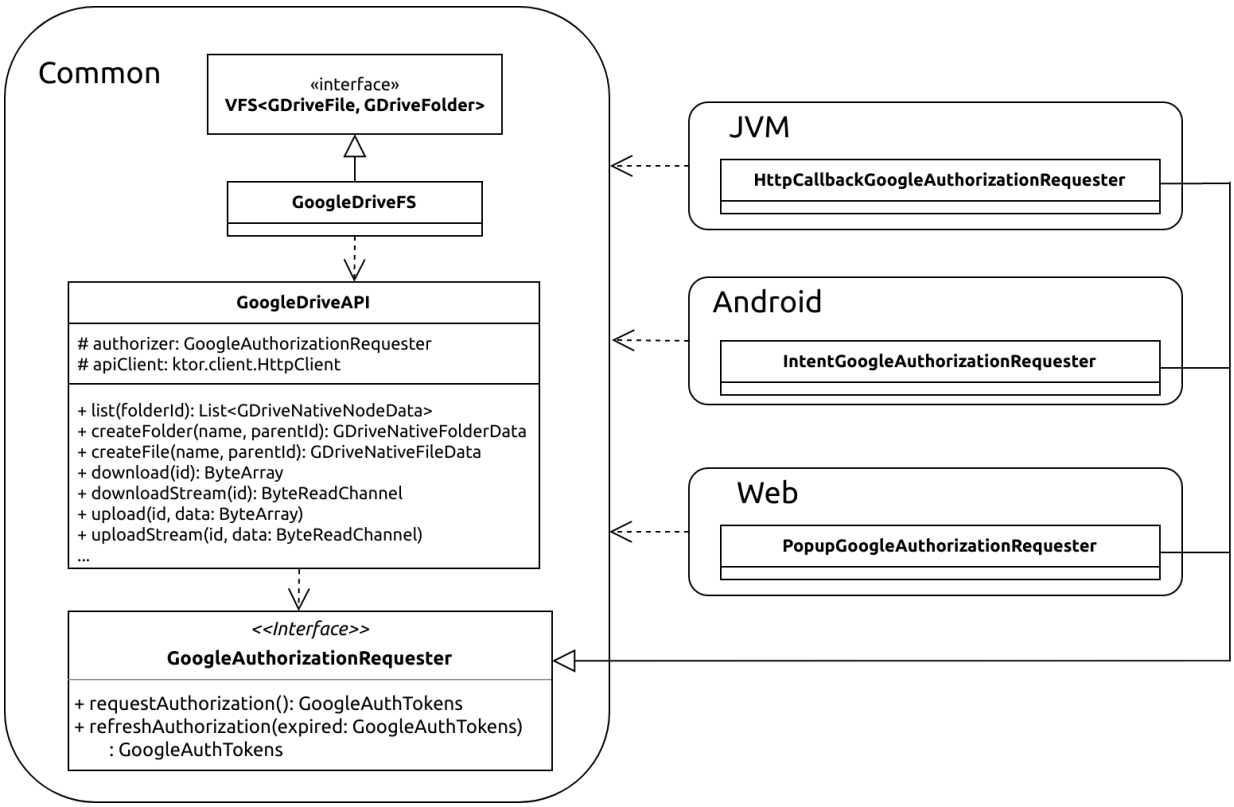

Рис. 4: Архитектура реализации поддержки файлового хранилища Google Drive.

3. Детали реализации

3.1. Структура проекта

(TODO: сказать про каждый модуль)

(TODO: commonJvmAndroid)

3.2. Сборка проекта

(TODO: про gradle)

(TODO: про maven и локальный репозиторий)

3.3. `SystemFS`

(TODO: реализует также `StreamingIO`)

3.4. `GoogleDriveFS`

3.4.1 `GoogleDriveAPI`

(TODO: обзор реализованных методов)

(TODO: в том числе StreamingIO)

3.4.2 `GoogleAuthorizationRequester` и его реализации

(TODO: Как на десктопе)

(TODO: как на андроиде)

(TODO: как в вебе)

3.5. `SqliteFS`

(TODO: как устроено, что ему нужно)

(TODO: запросы)

3.6. Тестирование

(TODO: тесты для systemfs)

(TODO: еще руками что-то потыкал через реализованные приложения (в т.ч. StreamingIO гугла))

4. Примеры использования библиотеки

4.1. Multieditor

4.1.1 Подключение библиотеки

(TODO: собираем либу, гредл команда, чтобы в локальный репозиторий артефакты попали)

(TODO: что дописать в гредл)

4.1.2 Реализация бизнес-логики в общем модуле приложения

(TODO: ну тут понятно, что написано в `AppState.kt`)

4.1.3 Определение списка доступных хранилищ на каждой из целевых платформ

(TODO: тут что написано на каждой платформе и как)

4.1.4 Реализация клиентских приложений на разных платформах

(TODO: Общие слова про компоуз)

(TODO: Клиент для платформ JVM и Android.)

(TODO: Клиент на платформе JS (browser).)

4.2. gdrive-cli

4.2.1 Написание логики приложения с учетом нескольких типовых ограничений на функциональность VFS

(TODO: тут описать как писать логику, если нужна расширенная функциональность от вфс)

(TODO: как написан runCli)

(TODO: заодно как пользоваться StreamingIO)

5. Результаты

(TODO: написана либа, поставленная задача выполнена, на ее базе написаны примеры)

(TODO: есть куда расти в плане поддержки платформ, бекендов, фичей вфс)

Заключение

(TODO: Заключение должно подводить итоги работы и содержать информацию о полученных в рамках работы результатах.)

Ссылки

[1] *Amazon S3. Cloud Object Storage*. URL: https://aws.amazon.com/s3/. (дата обращения 04.05.2022).

[2] *Amazon Simple Storage Service Documentation*. URL: https://docs.aws.amazon.com/s3/index.html. (дата обращения 11.05.2022).

[3] *API Яндекс.Диска*. URL: https://yandex.ru/dev/disk/api/concepts/about.html. (дата обращения 11.05.2022).

[4] *aSoft-Ltd/files*. URL: https://github.com/aSoft-Ltd/files. (дата обращения 04.05.2022).

[5] *asoft-storage*. URL: https://github.com/andylamax/asoft-storage. (дата обращения 04.05.2022).

[6] *Case Sensitivity | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/windows/wsl/case-sensitivity. (дата обращения 11.05.2022).

[7] *Delegated properties | Kotlin*. URL: https://kotlinlang.org/docs/delegated-properties.html. (дата обращения 04.05.2022).

[8] *Dropbox API v2*. URL: https://www.dropbox.com/developers/documentation/http/documentation. (дата обращения 11.05.2022).

[9] *File IO with Kotlin multiplatform - Stack Overflow*. URL: https://stackoverflow.com/q/68191209. (дата обращения 04.05.2022).

[10] *File Transfer Protocol - Wikipedia*. URL: https://en.wikipedia.org/wiki/File_Transfer_Protocol. (дата обращения 04.05.2022).

[11] *Files and folders overview | Drive API | Google Developers*. URL: https://developers.google.com/drive/api/guides/about-files. (дата обращения 11.05.2022).

[12] *GitHub*. URL: https://github.com. (дата обращения 04.05.2022).

[13] *kached*. URL: https://github.com/faogustavo/kached. (дата обращения 04.05.2022).

[14] *kile*. URL: https://github.com/realad/kile. (дата обращения 04.05.2022).

[15] *Kissme*. URL: https://github.com/netguru/Kissme. (дата обращения 04.05.2022).

[16] *korio*. URL: https://github.com/korlibs/korio. (дата обращения 04.05.2022).

[17] *Korlibs*. URL: https://docs.korge.org/. (дата обращения 04.05.2022).

[18] *Kotlin Data Storage*. URL: https://github.com/kotlingang/kds. (дата обращения 04.05.2022).

[19] *Kotlin Multiplatform Libraries*. URL: https://github.com/AAkira/Kotlin-Multiplatform-Libraries. (дата обращения 04.05.2022).

[20] *Kotlin/native: library for file io? - r/Kotlin (reddit)*. URL: https://www.reddit.com/r/Kotlin/comments/s3wzmt/kotlinnative_library_for_file_io/. (дата обращения 04.05.2022).

[21] *kotlinx-fs*. URL: https://github.com/qwwdfsad/kotlinx-fs. (дата обращения 04.05.2022).

[22] *kotlinx-io*. URL: https://github.com/Kotlin/kotlinx-io. (дата обращения 04.05.2022).

[23] *kotlinx-io. Is this library active/supported?* URL: https://github.com/Kotlin/kotlinx-io/issues/54. (дата обращения 04.05.2022).

[24] *Ktor Framework*. URL: https://ktor.io/. (дата обращения 04.05.2022).

[25] *KVault*. URL: https://github.com/Liftric/KVault. (дата обращения 04.05.2022).

[26] *Looking for a library or an alternative for basic multiplatform File I/O - Kotlin Discussions*. URL: https://discuss.kotlinlang.org/t/looking-for-a-library-or-an-alternative-for-basic-multiplatform-file-i-o/21472. (дата обращения 04.05.2022).

[27] *Multiplatform File I/O - r/Kotlin (reddit)*. URL: https://www.reddit.com/r/Kotlin/comments/g0hjsw/multiplatform_file_io/. (дата обращения 04.05.2022).

[28] *Multiplatform-Preferences*. URL: https://github.com/florent37/Multiplatform-Preferences. (дата обращения 04.05.2022).

[29] *multiplatform-settings*. URL: https://github.com/russhwolf/multiplatform-settings. (дата обращения 04.05.2022).

[30] *mutifs*. URL: https://github.com/vsalavatov/multifs. (дата обращения 04.05.2022).

[31] *okio*. URL: https://github.com/square/okio. (дата обращения 04.05.2022).

[32] *Read/Write file in kotlin native - IOS side - Stack Overflow*. URL: https://stackoverflow.com/q/61078285. (дата обращения 04.05.2022).

[33] *Representational state transfer - Wikipedia*. URL: https://en.wikipedia.org/wiki/Representational_state_transfer. (дата обращения 11.05.2022).

[34] *StackOverflow*. URL: https://stackoverflow.com. (дата обращения 04.05.2022).

[35] *supernatural-fs*. URL: https://github.com/suparngp/kotlin-multiplatform-projects. (дата обращения 04.05.2022).

[36] *Tinlok*. URL: https://github.com/Fuyukai/Tinlok. (дата обращения 04.05.2022).

[37] *Window.localStorage - Интерфейсы Web API | MDN*. URL: https://developer.mozilla.org/ru/docs/Web/API/Window/localStorage. (дата обращения 10.05.2022).