Функциональные характеристики и информация необходимая для установки, эксплуатации КОМПОНЕНТ «NPP - SDK»

Комплект средств разработки (библиотек) для встраивания в мобильные приложения банка или организации, позволяющие расширить функциональность возможностями сервиса NPP (Далее - nopaper[1]).

1.1. Комплект средств для встраивания имеет следующий функционал:

1. Авторизация. Авторизация пользователя по логину и паролю.
2. Верификация. Возможна очная верификация пользователя или посредством liveness проверки пользователя и фото паспорта. После верификации будет сохранен ключ на устройстве. Можно будет его активировать, получив и приняв акт признания ключа.
3. Получение списка всех документов.
4. Получение детальной информации о документе и список файлов документа.
5. Скачивание файлов документа.
6. Подпись документа. При пройденной верификации (и активном ключе на устройстве) пользователь может подписывать документы.
7. Получение информации об ЭП. Информация о всех выпущенных сертификатах ЭП пользователю.
8. Завершение сеанса работы. Возможность закончить текущий сеанс осуществив выход пользователя.

1.2. Сценарии встройки NPP - SDK в приложение банка:

Сценарий 1.

Приложение банка (Рис.1)

[1] «nopaper» - программное обеспечение, приложение для операционных систем iOS и Android, обеспечивающее осуществление электронного документооборота между пользователями и подписание электронных документов с помощью электронной подписи

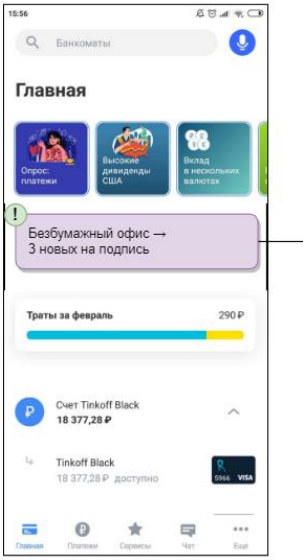

Рисунок 1

SDK открывается поверх приложения банка. Можно вернуться «Назад» и sdk закроется. (Рис.2)

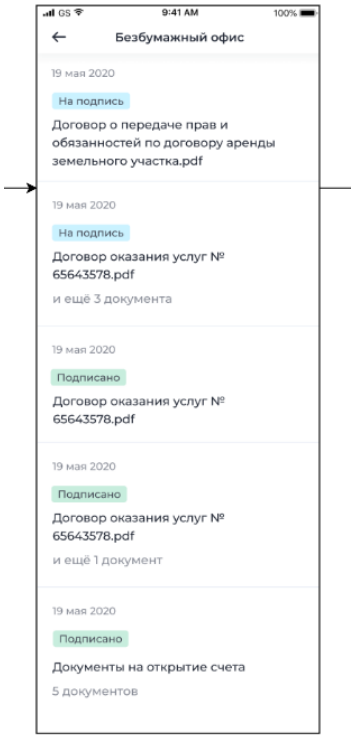

Рисунок 2

Далее, стандартная работа приложения NPP: навигация, функционал внутри приложения. (Рис.3)

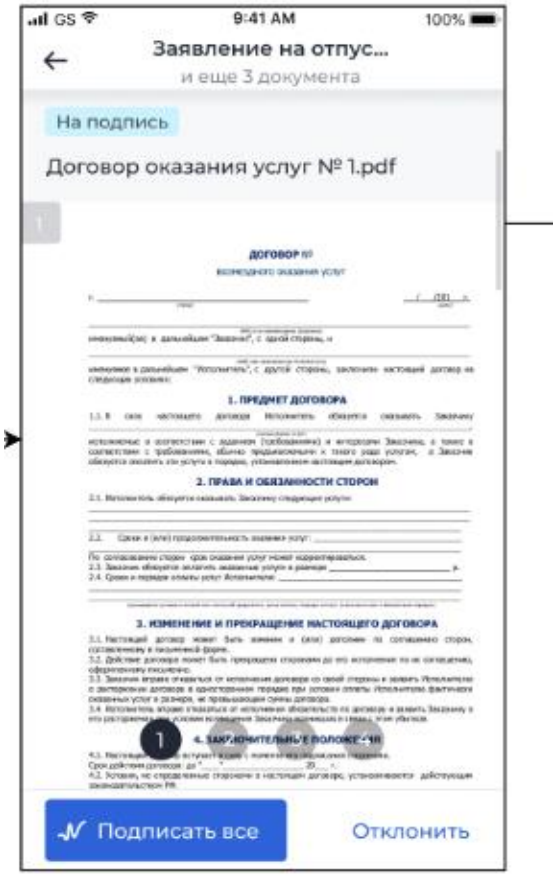

Рисунок 3

Стандартная работа приложения. (Рис.4)

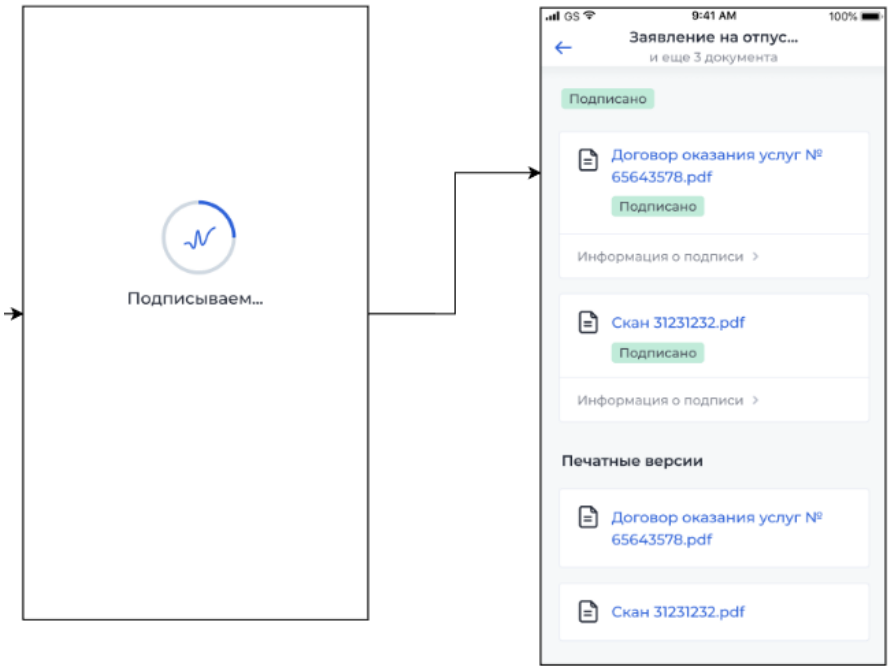

Рисунок 4

Сценарий 2.

Приложение банка. (Рис.5)

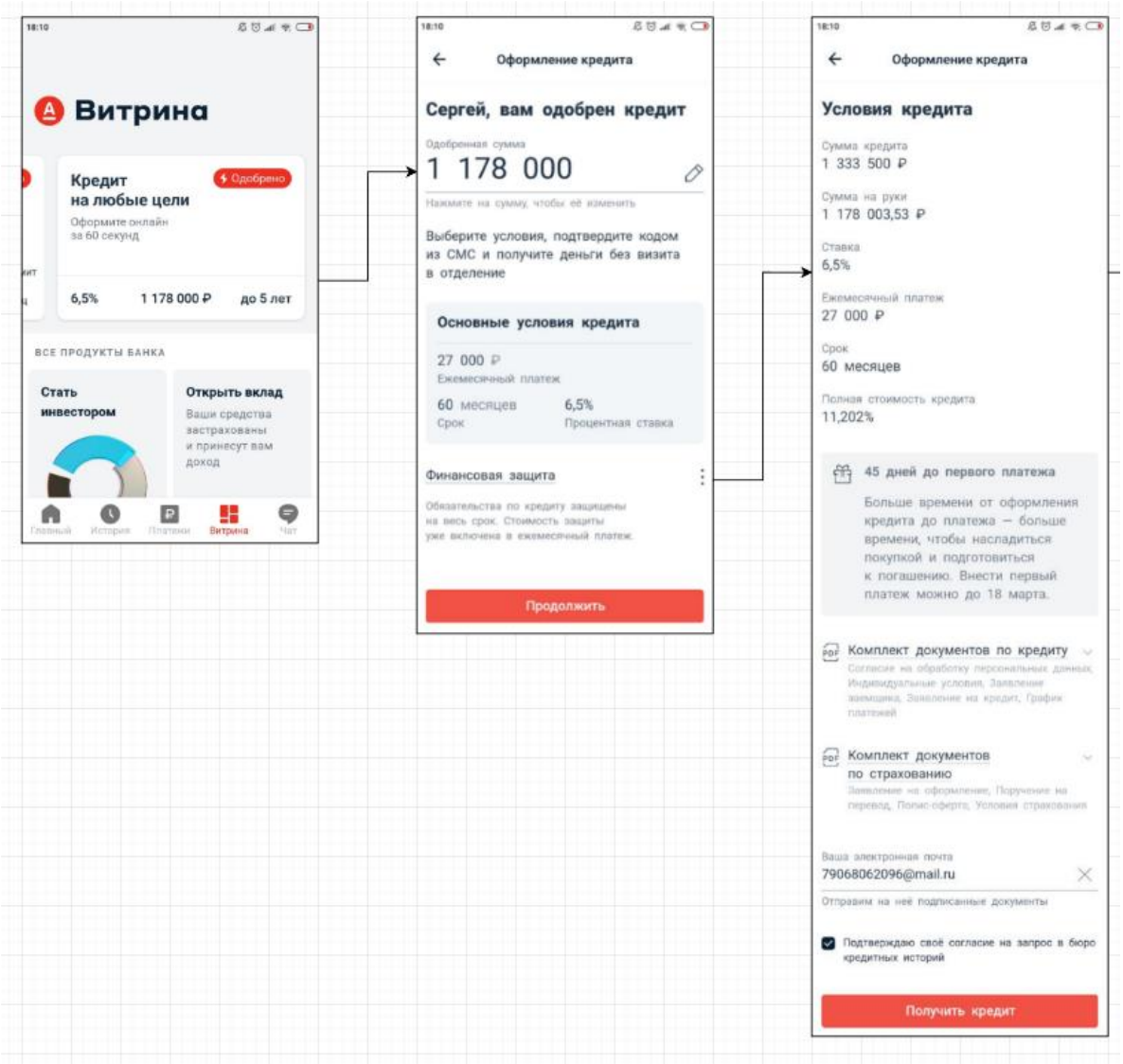

Рисунок 5

ДБО банка создаёт заявку на подпись в NPP, прикрепляет файлы к заявке, отправляет на подпись клиенту; открывает страницу SDK для просмотра файлов и подписи. И SDK-NPP, открывается поверх приложения банка. Можно вернуться «Назад» — NPP закроется. (Рис.6)

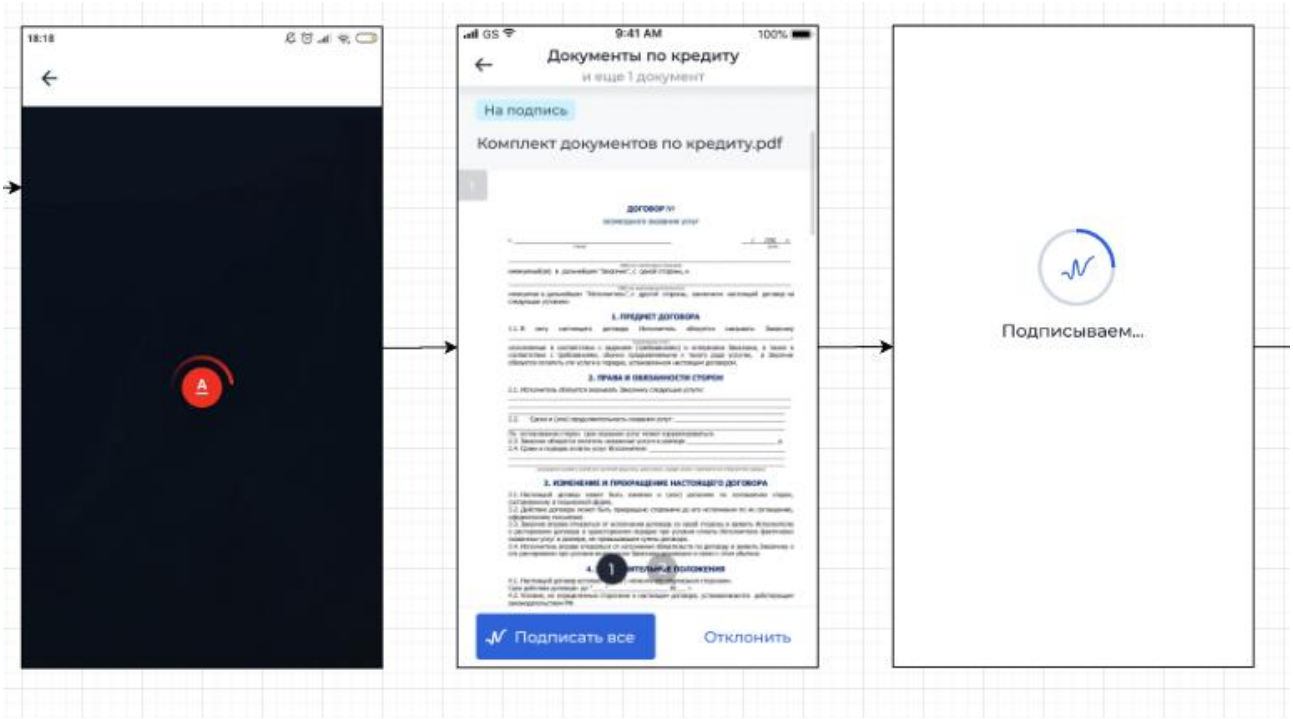

Рисунок 6

ДБО банка забирает подписанные файлы из NPP, отображает клиенту в приложении. Далее, приложение банка отображает клиенту историю документооборота. (Рис.7)

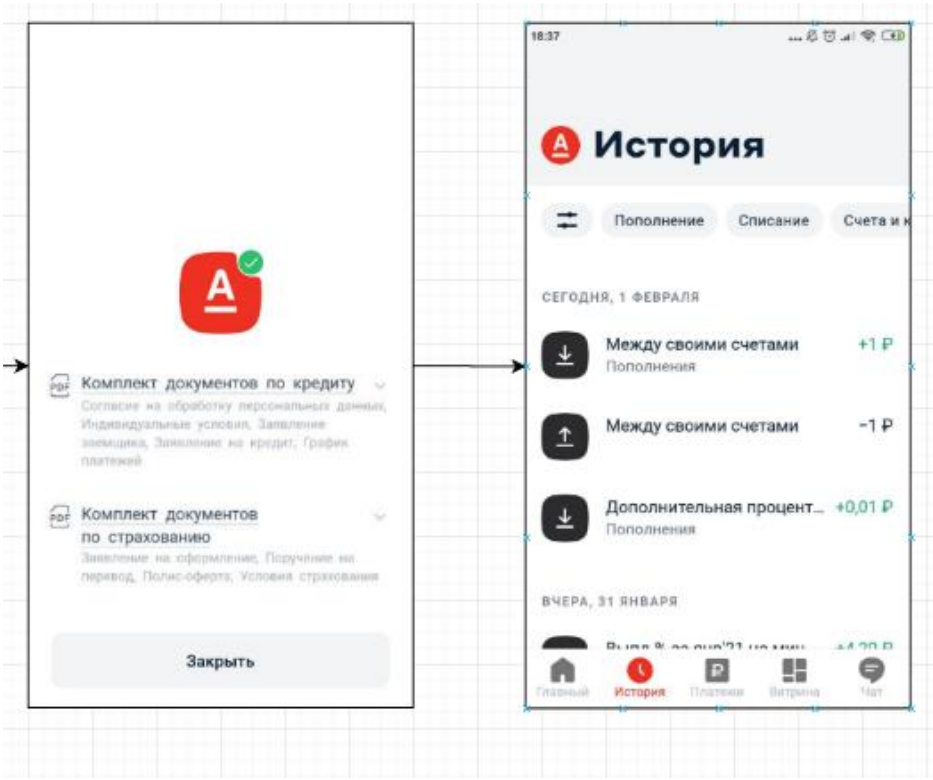
Комплект документов по кредиту
Комплект документов по страхованию
Закрыть
18:37
История
Пополнение
Списание
СЕГОДНЯ, 1 ФЕВРАЛЯ
Между своими счетами
Пополнения
+1 ₽
Между своими счетами
−1 ₽
Дополнительная процент...
Пополнения
+0,01 ₽
ВЧЕРА, 31 ЯНВАРЯ
Главный
История
Платежи
Витрина
Чат

Рисунок 7

2. Архитектура SDK

Общая архитектура на диаграмме (список методов только для примера, реальная реализация будет отличаться) (Рис.8):

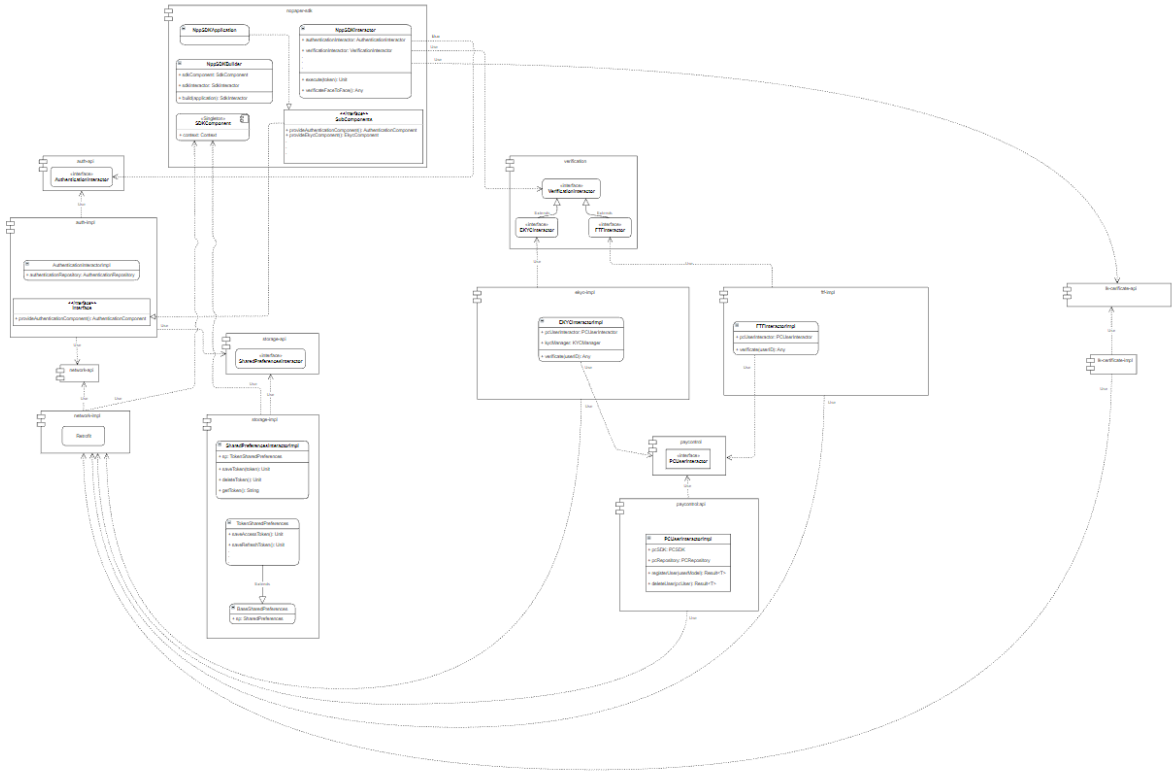

Рисунок 8

2.1. Состав:

1. nopaper-sdk – главный модуль SDK
2. манифест – xml-файл с объявленными android-компонентами
3. dagger – библиотека DI
4. dagger-компонент и dagger-модуль – классы, предоставляющие зависимости в компонент SDK (Не путать понятие dagger-модуля и компонент SDK. Компонент SDK - отдельная часть всего приложения. Dagger-модуль содержит методы типа

provideNppSDKInteractor(nppSDKInteractorImpl: NppSDKInteractorImpl): NppSDKInteractor для получения нужных зависимостей)

Общий сценарий SDK с ui такой (Activity в данном примере принадлежит родительскому приложению):

Application.onCreate() → NppSDKInitializer.create() → NppSDKBuilder.build() → Activity.onStart() → Activity.onResume() → NppSDKActivity.onCreate() → nav_graph_main → ...

Без ui такой:

Application.onCreate() → NppSDKInitializer.create() → NppSDKBuilder.build() → Activity.onStart() → Activity.onResume() → NppSDKInteractor.auth() → какое-то отображение → NppSDKInteractor.verificate() → какое-то отображение → NppSDKInteractor.getDocuments() → ...

Подробное описание сценария:

В манифесте nopaper-sdk объявлен класс NppSDKInitializer(). У него есть метод:

```
override fun create(context: Context): Boolean {
    NppSDKBuilder.build(context)
    return true
}
```

Он вызывается при каждом старте родительского приложения и в нем вызывается метод build у класса NppSDKBuilder.

Метод build сначала проверяет, не проинициализировано ли свойство sdkInteractor, и если нет, то получает имплементацию SdkInteractor через dagger-компонент и возвращает ее. Теперь родительское приложение может получить это свойство, так как оно статическое.

После инициализации SDK управление вернется родительскому Application, которое запускает отображение какой-нибудь Activity. Внутри этой активити, когда это нужно, можно создать интент на запуск стартовой активити SDK вот так:

```
Intent sdkIntent = new Intent(this, NppSDKActivity.class);
sdkIntent.putExtra("oneTimeLink", "https://onetimelink");
startActivity(sdkIntent);
```

На этом этапе проиведется запуск активити SDK, корневым элементом которой является граф навигации nav_graph_main. И там уже будет идти цепочка вызовов фрагментов из вложенных графов. Каждый модуль SDK, содержащий ui, будет иметь свой граф. Он и будет вложенным в nav_graph_main. Подробнее про библиотеку навигации можно почитать здесь: https://developer.android.com/guide/navigation/navigation-getting-started

2.2. Список основных модулей:

Каждый модуль содержит следующие пакеты: di и domain, а также data и ui, если предполагается работа с сетью, БД и пользовательским интерфейсом, соответственно.

- :nopaper-sdk

Главный модуль, зависит от всех остальных модулей. Содержит abanking-settings.json, MainActivity, корневой граф, ссылающийся на остальные, SdkComponent и SdkModule. Интерактор верхнего уровня SdkInteractor. Методы этого интерактора доступны родительскому приложению. Чтобы получить SdkInteractor необходимо вызвать метод build(application: Application) у класса SdkBuilder. Он создаст компонент, от которого можно получать зависимости, и вернет SdkInteractor.

Главный модуль предоставляет АПИ для обращения к внутренним модулям, управления и получения состояния встраимого приложения.

В данный момент есть разделение на две реализации этого интерфейса (для очной верификации и для ekyc) через buildFlavors. В зависимости от выбранной сборки будет использоваться нужная реализация.

- :auth:impl

Модуль, отвечающий за авторизацию. Зависит от модуля network-api

- :storage

Модуль, отвечающий за работу с локальным хранилищем. На диаграмме показан SharedPreferencesInteractor, с помощью которого сохраняются, удаляются и получаются access и refresh токены.

- :verification

Модуль с интерфейсами. Пока что это VerificationInteractor, EkycInteractor и FTFInteractor

Суть в том, что на внешнем слое мы будем просить у даггера объект VerificationInteractor. И в зависимости от buildVariant, с которым мы будем делать сборку, даггер предоставит нам либо EkycInteractorImpl, либо FTFInteractorImpl. А с помощью дженериков можем менять типы аргументов и возвращаемого значения. Для примера везде Any (аналог Object из Java)

- :ekyc-impl

Модуль, отвечающий за прохождение eKYC-верификации. Зависит от network-api и от PCSDK.

- :ftf-impl

Модуль, отвещающий за прохождение очной верификации. Зависит от network-api и от PCSDK.

- :paycontrol:impl

Модуль, отвещающий за создание и удаление пользователя на сервере PC, сохранение ключей в хранилище.

- :nopaper-lk:certificate:impl

Модуль для работы с сертификатами через lk-api

- :nopaper-lk:documents:impl

Модуль для работы с документами через lk-api

- :nopaper-lk:profile:impl

Модуль для работы с профилем через lk-api

- :network

Модуль, предоставляющий объект Retrofit, необходимый для общения с сервером.

- :utils

Утилитарный модуль

Все зависимости, необходимые только вложенным модулям, предоставляются там же, то есть в главном модуле мы получаем только интерфейсы-интеракторы и взаимодействуем через них.

3. Стек:

- Kotlin
- Gradle DSL (для настройки сборки)
- Android Jetpack Navigation (для навигации по экранам)
- Android Architecture Components (различные зависимости для удобной разработки android)
- Dagger (для внедрения зависимостей)
- Retrofit (для запросов на сервер)
- GSON (парсинга Json)
- Coroutines (для многопоточности)
- Timber (для логов)