**Курсовая работа «Rick and Morty»**

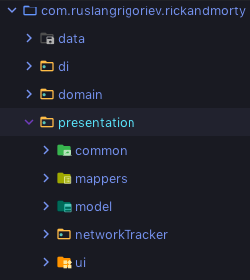
Основной ресурс: [Rick and Morty API](https://rickandmortyapi.com/documentation).

1. **Общее описание**

* Приложение имеет три основные вкладки с информацией о персонажах, локациях и эпизодах сериала.
* Из каждой вкладки можно перейти на страницу с деталями выбранного элемента
* На каждой вкладке есть поиск по странице и фильтрация по параметрам.
* Все вкладки поддерживают Pull-to-Refresh.
* В момент загрузки отображается прогресс-индикатор.
* На всех экранах кроме главное есть навигация назад и отображается стрелка.
* Имеет возможность работать без интернета и поддерживает кэширование.

1. **Структура приложения**

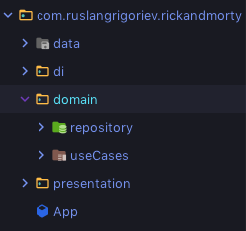
Приложение поделено на Data, Domain и Presentation слои которые находятся в соответствующих папках.

****

**Presentation:**

Тут находится все что касается интерфейса приложения.

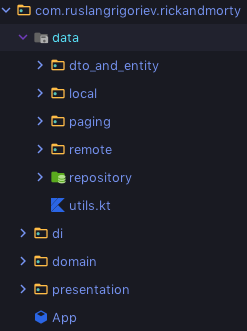
* common – общие для всех экранов классы
* mappers – мапперы для классов моделей отображения.
* model – модели для отображения на экранах деталей
* networkTracker – классы для отслеживания состояния интернет подключения
* ui – активити, фрагменты, viewmodels, адаптеры и холдеры.



**Domain:**

Тут описана бизнес-логика приложения

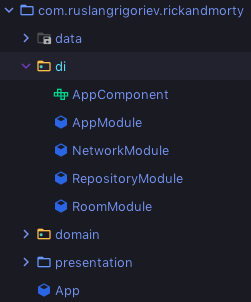
* repository – интерфейсы репозиториев для экранов
* useCases – классы для взаимодействия с репозиториями.
* Пакета models нет, т.к. данные из data слоя передаются в presentation без изменений.

****

**Data:**

Тут находится вся логика получения данных из локального и удаленного источников.

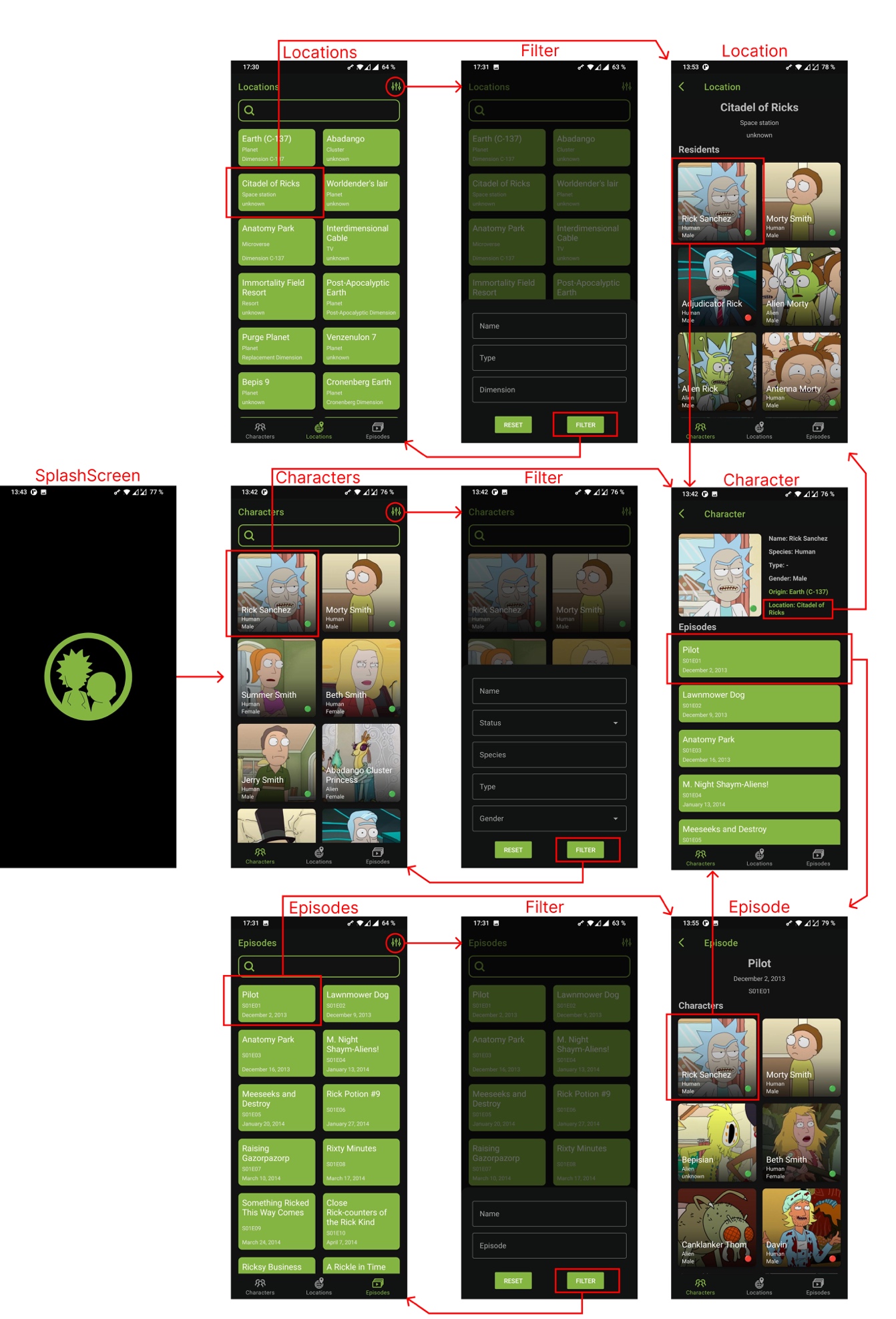
* dto\_and\_entity – data классы для конвертации полученных json ответов API сервиса (dto). Они же используются как сущности для хранения в БД (entity)
* local – БД, DAO интерфейсы и конвертеры
* paging – PagingSource классы для пагинации данных из API.
* remote – API интерфейсы для всех экранов.
* repository – реализация интерфейсов репозиториев для экранов.

**DI:**

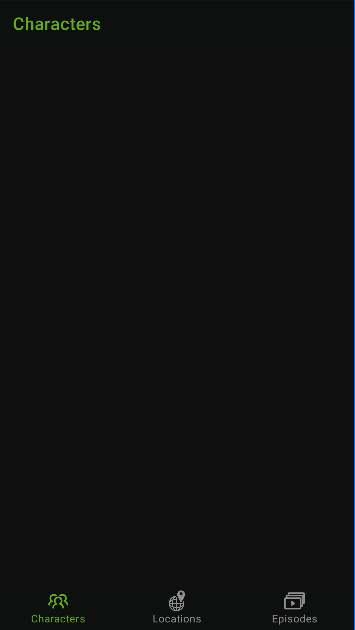
Тут находится все что касается внедрения зависимостей. Модули БД, API, репозитория, приложения и компонент для их предоставления.

В классе App инициализируется Dagger.

1. **Схема навигации в приложении**

****

1. **Интерфейс**

****

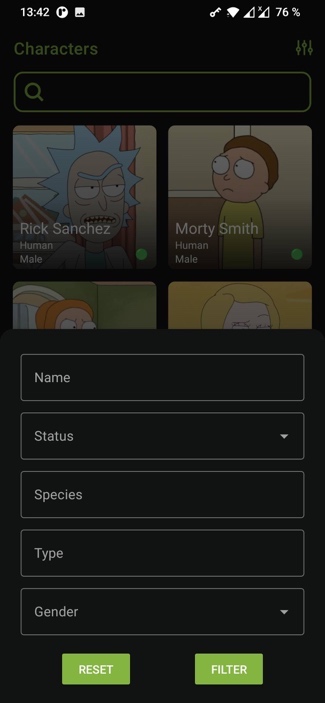
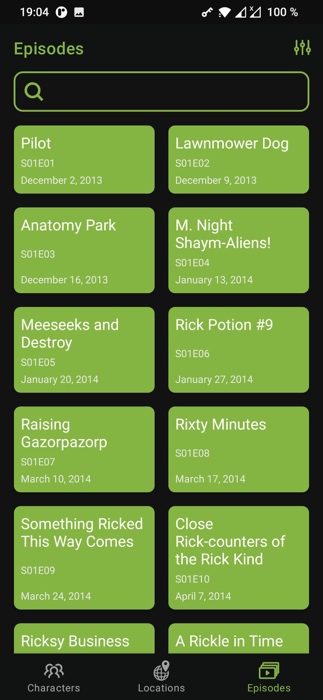
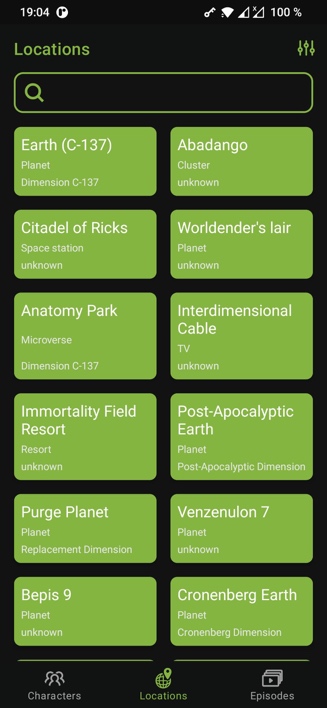
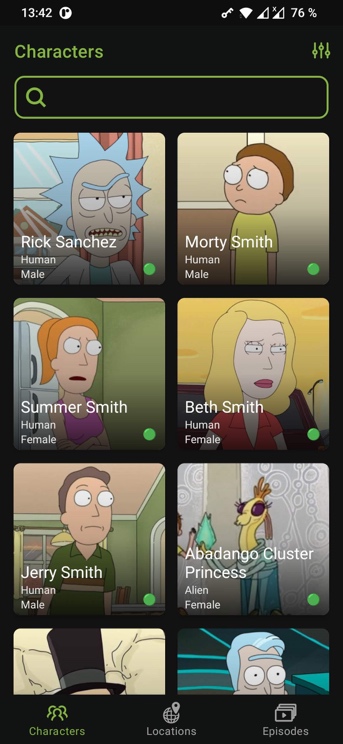
**SplashScreen**

Приложение стартует с отображения SplashScreen. Для этого используется SplashScreen API. В файле res/values/themes.xml создана тема с описанием splash экрана и указана тема для самого приложения.

В MainActivity методом *startSplashScreen* запускается splash экран на заданное время.

**Основной экран (MainActivity)**

Интерфейс построен на Single Activity Architecture. MainActivity содержит контейнер для фрагментов, BottomNavigationView и Toolbar. Меню навигации содержит три пункта: Characters, Locations, Episodes (файл res/menu/bottom\_menu\_nav.xml). При выборе пункта меню происходит переключение на соответствующий фрагмент с помощью интерфейса FragmentNavigator реализованного в MainActivity. Стартовым пунктом установлен Characters.

****

**Основной фрагмент (CharactersFragment, LocationsFragment, EpisodesFragment).**

Все основные фрагменты работают по одинаковому принципу. Отличаются только параметрами для фильтрации данных.

* **Toolbar:**

При создании фрагмента в Toolbar устанавливается title c названием экрана и меню для фильтрации (файл res/menu/menu\_filter.xml). Отключается кнопка «Назад».

* **Данные:**

Данные получаются из Viemodel иотображаются в RecyclerView. Пагинацию дынных осуществляет PagingAdapter (метод *collectData).*

При нажатии на элемент списка происходит смена фрагмента на экран с деталями (метод *onListItemClick*), передается ID элемента.

* **Индикация загрузки:**

В методе *initRecyclerView* адаптеру устанавливается *addLoadStateListener* в котором устанавливается индикация загрузки SwipeRefreshLayout и отображение TextView, когда результат пустой.

* **Фильтрация:**

При нажатии на меню фильтрации открывается BottomSheetDialogFragment

(метод *showFilter*) с полями для ввода или выпадающими списками возможных вариантов параметров (файл res/values/strings.xml). Кнопка Reset очищает поля фильтра.

При применении фильтра дынные сохраняются в поле filter фрагмента для дальнейшего использования и делается [запрос](#Запрос1) данных с фильтром. Очищается поле поиска по странице.

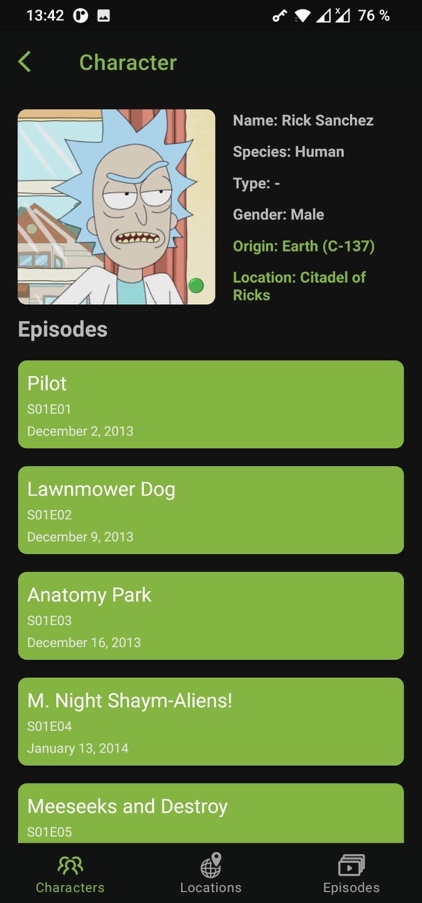
* **Поиск:**

В методе *initSearch* устанавливается *OnQueryTextListener* для SearchView.

В нем сохраняется набираемый текст в поле searchQuery и делается [запрос](#Запрос1) данных. Полученные данные фильтруются в методе *collectData.*

* **Обновление:**

Для реализации Swipe-to-Refresh используется SwipeRefreshLayout в верстке экрана. В методе *initSwipeToRefresh* устанавливается *setOnRefreshListener* в котором очищается фильтр и сохраненный searchQuery, делается [запрос](#Запрос1) данных.



**Детали персонажа (CharacterDetailsFragment).**

ID персонажа передается в аргументах при создании фрагмента.

* **Toolbar:**

При создании фрагмента в Toolbar устанавливается title c названием экрана и включается кнопка «Назад».

* **Данные:**

В методе *fetchData* проверяется загружены ли уже данные (что бы повторно не загружать при возвращении фрагмента из бэкстека) и делается [запрос](#Запрос2) данных. Фрагмент подписывается на получение данных из ViewModel и отображает их в методе *bindUi*.

Поля экрана Location и Origin делаются кликабельными и выделяются цветом если не «unknown». При нажатии открывается фрагмент LocationDetailsFragment, которому в аргументы передается ID (метод *onLocationClick*).

Список эпизодов отображается в RecyclerView, используется тот же макет элемента, что и на основном экране эпизодов (файл res/layout/item\_episode.xml).

При нажатии на эпизод происходит смена фрагмента на EpisodeDetailsFragment (метод *onListItemClick*), в аргументы передается ID элемента.

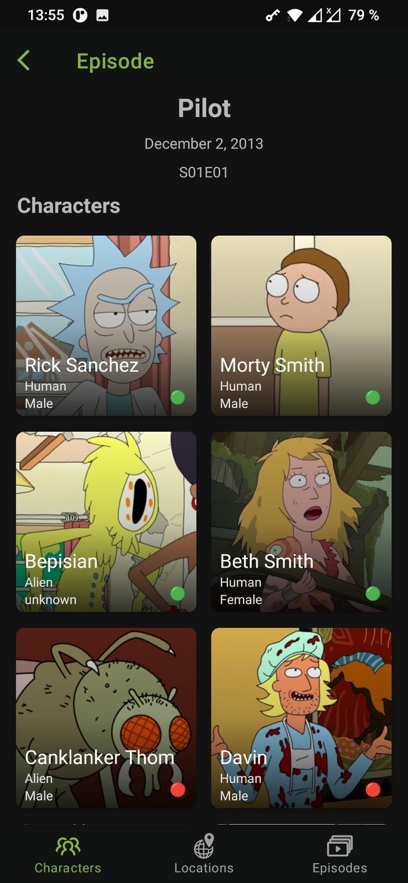
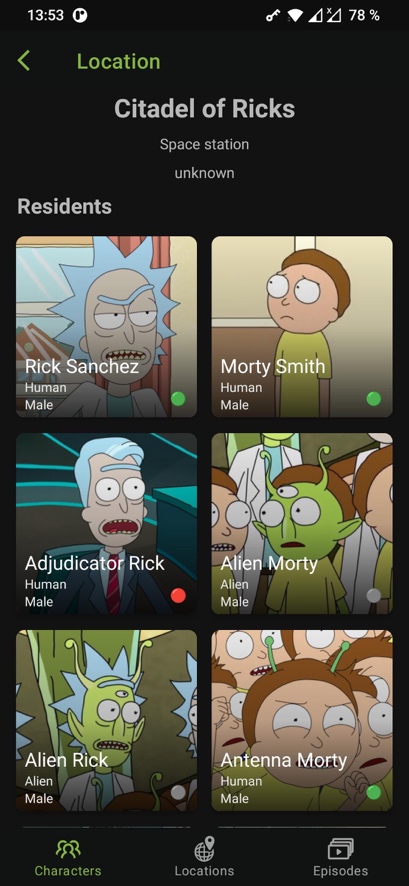
* **Индикация загрузки:**

ViewModel хранит LiveData-поле *isLoading* и обновляет его в зависимости от состояния [запроса](#Запрос2). Фрагмент подписывается на него в методе *fetchData* и

устанавливает видимость прогресс-бара и индикатора загрузки SwipeRefreshLayout.

* **Обновление:**

Для реализации Swipe-to-Refresh используется SwipeRefreshLayout в верстке экрана. В методе *initSwipeToRefresh* устанавливается *setOnRefreshListener* в котором делается [запрос](#Запрос2) данных.

****

**Детали локации, детали эпизода. (LocationDetailsFragment, EpisodeDetailsFragment).**

Дынные фрагменты работают по одинаковом принципу.

ID персонажа передается в аргументах при создании фрагмента.

* **Toolbar:**

При создании фрагмента в Toolbar устанавливается title c названием экрана и включается кнопка «Назад».

* **Данные:**

В методе *fetchData* проверяется загружены ли уже данные (что бы повторно не загружать при возвращении фрагмента из бэкстека) и делается запрос данных. Фрагмент подписывается на получение данных из ViewModel и отображает их в методе *bindUi*.

Список персонажей отображается в RecyclerView, используется тот же макет элемента, что и на основном экране эпизодов

(файл res/layout/item\_character.xml).

При нажатии на элемент списка происходит смена фрагмента на CharacterDetailsFragment (метод *onListItemClick*), в аргументы передается ID элемента.

* **Индикация загрузки:**

ViewModel хранит LiveData-поле *isLoading* и обновляет его в зависимости от состояния запроса. Фрагмент подписывается на него в методе *fetchData* и

устанавливает видимость прогрессбара и индикатора загрузки SwipeRefreshLayout.

* **Обновление:**

Для реализации Swipe-to-Refresh используется SwipeRefreshLayout в верстке экрана. В методе *initSwipeToRefresh* устанавливается *setOnRefreshListener* в котором делается запрос данных.

1. **Получение данных**

Все репозитории работают по одинаковому принципу. Далее будет описан вариант запросов из экранов Characters и CharacterDetails.

В приложении два источника данных: API и локальная база данных.

Все полученные данные из API сохраняются в БД для возможности работы при отсутствии подключения к интернету. CharactersRepositoryImpl определяет источник в зависимости от установленного флага isNetworkAvailable

(ext. метод getRemoteOrCachedData).

Состояние подключения отслеживается с помощью класса NetworkStatusTracker

(файл rickandmorty/presentation/networkTracker/NetworkStatusTracker.kt).

MainActivity подписывается на свойство networkState MainViewModel в методе

*observeConnection* и устанавливает во всех репозиториях флаг isNetworkAvailable.

Все методы репозитория при наличии интернета обращаются за данными к API либо к БД при его отсутствии. При появлении подключения после отсутствия, БД очищается методом *clearCache.*

**Запрос данных для экрана Characters:**

* CharactersViewModel вызывает через UseCase у репозитория метод *getCharacters.*
* API запрос *-* [*https://rickandmortyapi.com/api/character*](https://rickandmortyapi.com/api/character)

может вызываться с параметрами для фильтрации и без

([*https://rickandmortyapi.com/documentation/#filter-characters*](https://rickandmortyapi.com/documentation/#filter-characters)).

* Для реализации пагинации в методе *getCharacters()* используется CharactersPagingSource для запросов к API. Из БД же получается сразу PagingSource. Полученные PagingSource передаются в Pager.
* Во ViewModel возвращается Flow<PagingData<Character> и на него подписывается фрагмент.

**Запрос данных для экрана CharacterDetails:**

* CharacterDetailsViewModel вызывает через UseCase у репозитория метод *getCharacterById(id)*
* API запрос *-* [*https://rickandmortyapi.com/api/character/{id}*](https://rickandmortyapi.com/api/character/1)
* Метод возвращает объект Character
* Во viewmodel собирается список ID эпизодов у полученного персонажа – ids
* Viewmodel вызывает через UseCase у репозитория метод *getCharacterEpisodes(ids).*
* API запрос *-* [*https://rickandmortyapi.com/api/episode/{ids}*](https://rickandmortyapi.com/api/episode/%7bids%7d)
* Метод возвращает List<Episode>
* С помощью CharacterMapper мапится Character и List<Episode> в CharacterModel
* Обновляется значение LiveData data на которую подписан фрагмент.