БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет КСИС

Специальность ИиТП

Индивидуальная практическая работа №2

по дисциплине

«Модели данных и системы управления базами данных. Часть 2»

Кэширование с помощью БД в android-приложениях

Выполнил студент: Драгун О.В.

группа 893551

Зачетная книжка № 2520050

Минск 2022

Оглавление

[Теория 3](#_Toc98146043)

[Практика 4](#_Toc98146044)

[UI 4](#_Toc98146045)

[Забор данных из сети 4](#_Toc98146046)

[Сохранение данных в БД 5](#_Toc98146047)

[Отображение данных 5](#_Toc98146048)

[Вывод 7](#_Toc98146049)

[Ссылки 8](#_Toc98146050)

# Теория

В данной работе я освещу тему, отсутствующую в вариантах к выполнению, однако вписывающуюся в рамки предмета. Это кэширование с помощью БД в android-приложениях.

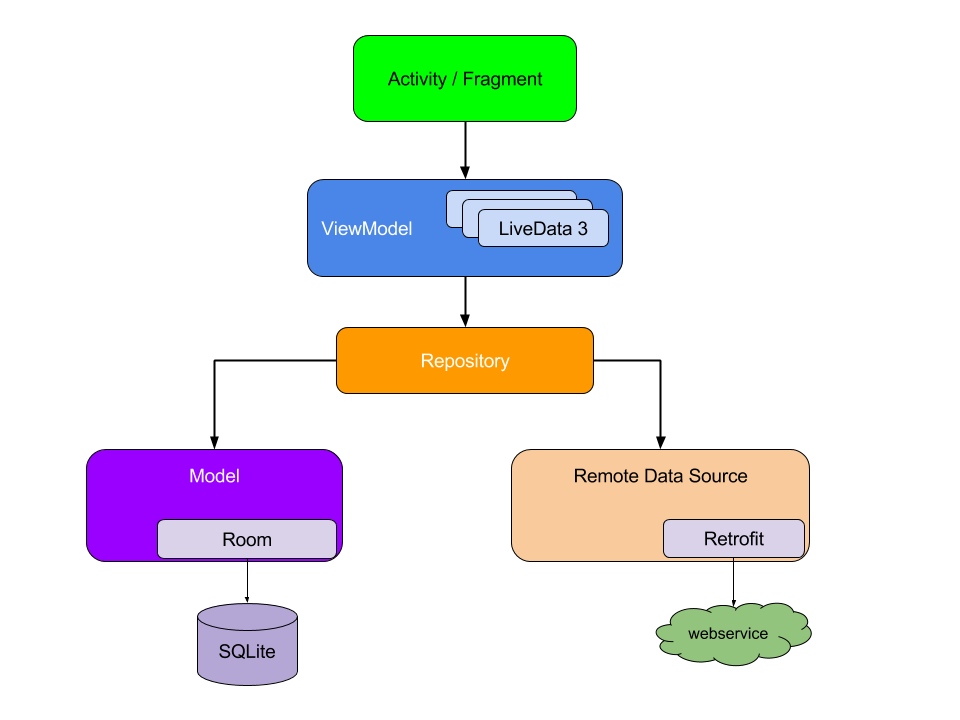


Рис 1. Архитектура приложения

Не будем останавливаться подробно на каждом слое, скажем лишь про основные

1. Webservice

Тут я взял две открытые апи сервиса <https://openweathermap.org/>

Первое для получения подробного прогноза погоды для текущего местоположения пользователя

Второе для получения менее подробного прогноза по названию города

1. SQLite

В основе лежит БД SQLite

1. Room

Самая популярная обертка над SQLite для android на данный момент.

# Практика

## UI

Изображение выглядит как стол

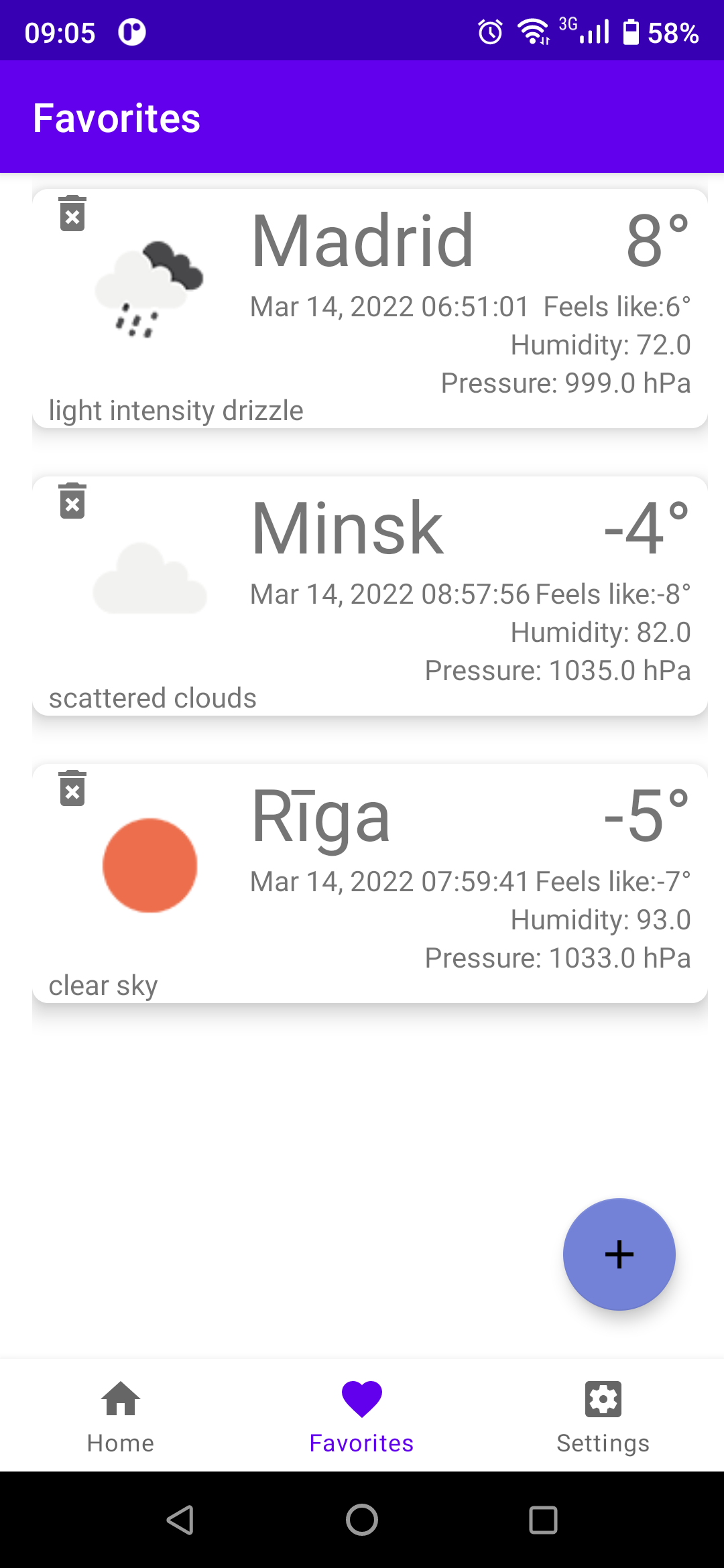
Автоматически созданное описание 

Рис2. Текущий прогноз для апи1, прогнозы по апи2

## Забор данных из сети

Производим по 2 функциям

@GET("data/2.5/onecall")  
suspend fun getByCoordinates(  
 @Query("lat") lat: Double,  
 @Query("lon") lon: Double,  
 @Query("appid") appId: String = APP\_ID,  
 @Query("units") units: String = "metric"  
): ForecastOnecall  
  
*/\*\*  
 \** ***@param*** *city might be City, City + state code, City + state + country divided by comma  
 \* more: https://openweathermap.org/current#name  
 \*/*@GET("data/2.5/weather")  
suspend fun getByCityName(  
 @Query("q") city: String,  
 @Query("appid") appId: String = APP\_ID,  
 @Query("units") units: String = "metric"  
): ForecastByCity

Далее данные попадают в репозиторий, откуда мы сохраняем их в БД.

## Сохранение данных в БД

interface ForecastDao {  
 */\*\*  
 \* Onecall methods\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*/* //it is always single instance because update recreate it  
 @Query("select \* from databaseforecastonecall")  
 fun getOnecall(): LiveData<DatabaseForecastOnecall>  
  
 //transaction make sure BOTH operations done  
 @Transaction  
 fun updateData(forecastOnecall: DatabaseForecastOnecall) {  
 deleteAllForecastOnecall()  
 insertAll(forecastOnecall)  
 }  
  
 //don't call this from repo  
 @Insert  
 fun insertAll(forecastOnecall: DatabaseForecastOnecall)  
  
 //don't call this from repo  
 @Query("DELETE FROM databaseforecastonecall")  
 fun deleteAllForecastOnecall()  
  
 */\*\*  
 \* Cities methods\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*/* @Query("select \* from databaseforecastcity")  
 fun getAllCities(): LiveData<List<DatabaseForecastCity>>  
  
 @Query("select cityId FROM databaseforecastcity")  
 fun getCitiesNames(): List<String>  
  
 @Insert(onConflict = OnConflictStrategy.*REPLACE*)  
 fun insertAllCities(list: List<DatabaseForecastCity>)  
  
 @Insert(onConflict = OnConflictStrategy.*REPLACE*)  
 fun insertCity(city: DatabaseForecastCity)  
  
 @Query("DELETE FROM databaseforecastcity WHERE cityId = :cityID")  
 fun deleteCity(cityID: String)  
}

## Отображение данных

Со стороны презентера мы наблюдаем за базой и подписываемся на изменения в ней. Далее через трансформации пускаем на UI

*/\*\*  
 \* Observable list of favorite cities. Show this in your UI  
 \*/*val cities: LiveData<List<ForecastByCity>> =  
 Transformations.map(forecastDao.getAllCities()) **{** list **->** list?.*asDomainModel*()?.*sortedBy* **{ it**.name **}  
 }**

*/\*\*  
 \* Observable object of home weather. Show this in your UI  
 \*/*val forecastOnecall: LiveData<ForecastOnecall> =  
 Transformations.map(forecastDao.getOnecall()) **{  
 it**?.*asDomainModel*()  
 **}**

# Вывод

В реализованном приложении мы познакомились со способом кэширования через абстракцию над SQLite. Подобный подход применим и к production-решениям большего размера. Один раз загрузить данные и далее показывать их в offline -самый лучший вариант для пользователя любого приложения

# Ссылки

1. <https://medium.com/@nyavorskii/%D0%B7%D0%BD%D0%B0%D0%BA%D0%BE%D0%BC%D1%81%D1%82%D0%B2%D0%BE-%D1%81-android-architecture-components-%D0%B8-mvvm-%D0%BF%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%D0%B4-29654672f4ab>
2. <https://proandroiddev.com/enabling-cache-offline-support-on-android-using-room-4b82ae0c9c88>
3. <https://openweathermap.org/>