**Лабораторная работа 4**

**Работа с сетью**

Дополните голосового помощника возможностью поддержать разговор о погоде. Научите его получать сведения о погоде из сети с использованием api.

Воспользуйтесь сервисом weatherstack.com, в нем есть бесплатная возможность получать погоду. Но нужно зарегистрироваться. При регистрации выберите свободный доступ на https://weatherstack.com/product и получите бесплатный ключ для использования api.

После регистрации скопируйте себе API Access Key.

Познакомьтесь с документацией <https://weatherstack.com/documentation>.

Выполните в строке адреса браузера следующий запрос

http://api.weatherstack.com/current?access\_key=YOUR\_ACCESS\_KEY&query=Киров

Ответ на запрос придет в формате json:

{  
 **"request"**: {  
 **"type"**: **"City"**,  
 **"query"**: **"\u041a\u0438\u0440\u043e\u0432,\u0420\u043e\u0441\u0441\u0438\u044f"**,  
 **"language"**: **"en"**,  
 **"unit"**: **"m"** },  
 **"location"**: {  
 **"name"**: **"\u041a\u0438\u0440\u043e\u0432"**,  
 **"country"**: **"\u0420\u043e\u0441\u0441\u0438\u044f"**,  
 **"region"**: **"Kirov"**,  
 **"lat"**: **"58.597"**,  
 **"lon"**: **"49.658"**,  
 **"timezone\_id"**: **"Europe\/Kirov"**,  
 **"localtime"**: **"2020-02-16 23:31"**,  
 **"localtime\_epoch"**: 1581895860,  
 **"utc\_offset"**: **"3.0"** },  
 **"current"**: {  
 **"observation\_time"**: **"08:31 PM"**,  
 **"temperature"**: -2,  
 **"weather\_code"**: 332,  
 **"weather\_icons"**: [  
 **"https:\/\/assets.weatherstack.com\/images\/wsymbols01\_png\_64\/wsymbol\_0036\_cloudy\_with\_heavy\_snow\_night.png"** ],  
 **"weather\_descriptions"**: [  
 **"Moderate snow"** ],  
 **"wind\_speed"**: 19,  
 **"wind\_degree"**: 216,  
 **"wind\_dir"**: **"SW"**,  
 **"pressure"**: 1024,  
 **"precip"**: 0.1,  
 **"humidity"**: 92,  
 **"cloudcover"**: 100,  
 **"feelslike"**: -8,  
 **"uv\_index"**: 1,  
 **"visibility"**: 5,  
 **"is\_day"**: **"no"** }  
}

Ответ состоит из нескольких частей. В часть **location** содержится информация о месте, для которого выполняется запрос о погоде и времени, в которое выполняется запрос.

В разделе **current** содержится информация о погоде: температура – **temperature**, направление – **wind\_dir** и скорость ветра – **wind\_speed**, даже адрес соответствующей погоде картинки **weather\_icons**, текстовое описание погоды – **weather\_descriptions**.

Как научить помощника отвечать на вопрос: *погода в городе Киров?* Нужно разобрать эту фразу и «достать» из нее название города.

В методе getAnswer класса AI создайте шаблон для разбора строки:

Pattern cityPattern = Pattern.*compile*(**"погода в городе (\\p{L}+)"**, Pattern.***CASE\_INSENSITIVE***);  
Matcher matcher = cityPattern.matcher(question);  
**if** (matcher.find()){  
 String cityName = matcher.group(1);  
 answers.add(**"Не знаю я, какая там погода у вас в городе "**+cityName);  
}

**Запуск!** Запустите и протестируйте приложение.

Сейчас нужно решить задачу получения данных из сети. Для этого добавьте разрешение на использование интернета в файл AndroidManifest:

<**uses-permission android:name="android.permission.INTERNET"**/>

Внутри тега application добавьте атрибут usesCleartextTraffic. Он указывает на то, что приложение обращается к серверу по незащищенному протоколу http:

**android:usesCleartextTraffic="true"**

Для выполнения запросов к api используется библиотека **Retrifit2.**

Это сторонняя библиотека, ее нужно добавить в файл gradle. В проекте два файла build.gradle. Выберите файл, как на рисунке 1, (Module:app). В это файл пропишите зависимости:

implementation **'com.squareup.retrofit2:retrofit:2.7.1'**implementation **'com.squareup.retrofit2:converter-gson:2.7.1'**

и нажмите Sync now

Справочный материал по работе с библиотекой <https://square.github.io/retrofit/>, <https://github.com/square/retrofit>. Там же можно найти номер последней версии библиотеки. Подождите, пока gradle синхронизирует зависимости и добавить библиотеки в проект.

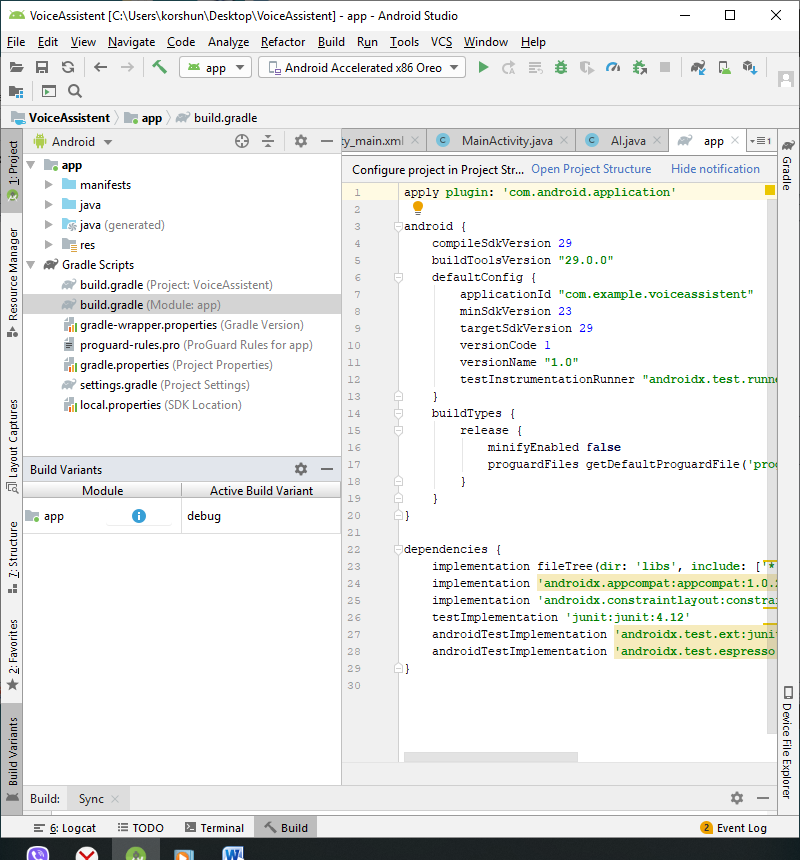


Рисунок 1. build.gradle (Module: app)

Создайте класс Forecast. Он будет моделью данных о погоде. Данные о погоде находятся в разделе current.

Из всего ответа на запрос о погоде будем брать только значение температуры (**temperature**) и описание погоды (**weather\_descriptions**). Заметим, что одно из полей является списком.

**public class** Forecast **implements** Serializable {  
  
 @SerializedName(**"current"**)  
 @Expose  
 **public** Weather **current**;  
  
 **public class** Weather{  
 @SerializedName(**"temperature"**)  
 @Expose  
 **public** Integer **temperature**;  
  
 @SerializedName(**"weather\_descriptions"**)  
 @Expose  
 **public** List<String> **weather\_descriptions**;  
 }  
}

При описании данных используется аннотация @SerializedName(строковый\_параметр), параметром которой является имя данных в json-файле.

Создайте интерфейс ForecastApi. В нем опишите следующий метод:

@GET(**"/current?access\_key=YOUR\_ACCESS\_KEY"**)  
Call<Forecast> getCurrentWeather(@Query(**"query"**) String city);

Метод принимает параметр город, для которого нужно выполнять прогноз погоды. Название города уходит в get-запрос. Имя соответствующего параметра в get-запросе query, поэтому это имя нужно указать в аннотации к параметру. Метод возвращает объект Call< Forecast >. На место YOUR\_ACCESS\_KEY запишите свой ключ.

Создайте класс ForecastService. В нем опишите единственный метод:

**public static** ForecastApi getApi() {  
 Retrofit retrofit = **new** Retrofit.Builder()  
 .baseUrl(**"http://api.weatherstack.com"**) *//Базовая часть адреса* .addConverterFactory(GsonConverterFactory.*create*().*create*()) *//Конвертер, необходимый для преобразования JSON'а в объекты* .build();  
  
 **return** retrofit.create(ForecastApi.**class**); *//Создание объекта, при помощи которого будут выполняться запросы*}

Здесь создается объект класса Retrofit через builder. При создании ему нужно указать URL ресурса, на который будут отправляться запросы, определить формат, который будет приходить в ответе, и указать способ (библиотеку) разбора полученного ответа.

Создайте класс ForecastToString. В нем опишите метод получения погоды:

**public static void** getForecast(String city){  
  
}

В нем нужно получить объект класса ForecastApi и с помощью него выполнить запрос погоды в выбранном городе:

ForecastApi api = ForecastService.*getApi*();  
Call<Forecast> call = api.getCurrentWeather(city);

Результат работы вернется в переменную типа Call<Forecast>.

Retrofit делает запросы не в основном потоке работы приложения, а в некотором фоновом режиме (background). Делать запрос в основном потоке работы приложения нельзя, иначе приложение может зависнуть.

Весь запрос к сети на получение текущей погоды сконфигурирован и содержится в переменной call. Осталось только выполнить этот запрос. Начните набирать следующие строки, студия сама подскажет, какой класс нужно выбрать и сама вставить все необходимые строки кода:

call.enqueue(**new** Callback<Forecast>() {  
 @Override  
 **public void** onResponse(Call<Forecast> call, Response<Forecast> response){  
  
 }  
  
 @Override  
 **public void** onFailure(Call<Forecast> call, Throwable t) {  
  
 }  
});

Когда запрос выполнен, будет получен Callback. Осталось обработать методы этого объекта.

В методе обработки ошибок отправьте сообщение в LOG о полученной ошибке:

Log.*w*(**"WEATHER"**,t.getMessage());

Если запрос успешно выполнен, нужно получить тело ответа:

Forecast result = response.body();

Объект result и будут содержать ответ о погоде. Сохраните этот ответ в какую-то строковую переменную:

**if** (result!=**null**) {  
 String answer = **"сейчас где-то "** + result.**current**.**temperature** + **" градуса "** + **" и "** + result.**current**.**weather\_descriptions**.get(0);  
}

Строка ответа получена, но куда ее девать. Метод onResponse ничего не возвращает, так как запрос выполняется асинхронно.

Как передать полученную строку в графический интерфейс?

Для этого моно использовать Consumer. Добавьте в метод getForecast второй параметр:

**final** Consumer<String> callback

В итоге получится метод:

**public static void** getForecast(String city, **final** Consumer<String> callback)

Передача строки в теле метода onResponse будет выглядеть следующим образом:

callback.accept(answer);

В случае равенства переменной result значению null выполните:

callback.accept(**"Не могу узнать погоду"**);

Вызов метода получения погоды нужно выполнить в кассе AI. После получения cityName из вопроса начните набирать:

ForecastToString.*getForecast*(cityName, **new** Consumer<String>()

Студия сама запишет все правильно.

ForecastToString.*getForecast*(cityName, **new** Consumer<String>() {  
 @Override  
 **public void** accept(String s) {  
  
 }  
});

В тело метода accept запишите команду добавления weatherString к списку ответов.

Далее нужно «вытащить» массив ответов из класса AI. Если выполнять запросы к сети, массив ответов нельзя вернуть мгновенно. Поэтому добавьте Consumer к параметрам метода getAnswer точно так же, как это было сделано в методе getForecast. Сам метод getAnswer сейчас не должен ничего возвращать, измените тип возвращаемого объекта. Вместо строки return запишите:

callback.accept(String.*join*(**", "**, answers));

Заметим, если при создании приложения минимальный уровень API меньше 26, то метод join работать не будет. Нужно будет по-другому получить строку из списка строк.

Эту же строку нужно вставить в метод accept, после того, как ответ добавлен в список.

Осталось обновить метод onSend класса MainActivity. Нужно изменить вызов метода getAnswer: к тексту вопроса добавьте Consumer:

AI.*getAnswer*(text, **new** Consumer<String>() {  
 @Override  
 **public void** accept(String answer) {  
  
 }  
});

Всю логику вывода ответа ассистента на экран нужно поместить в метод accept.

**Запуск**. Запустите и протестируйте приложение.

**Задание**.

1. Сейчас AI выводит строки о погоде не грамотно, например, Там сейчас, где-то -1 градуса. Добавьте в класс AI метод, который будет по данному числу возвращать правильное окончание слова градус.
2. Есть совершенно бестолковый сервис <https://htmlweb.ru>, который по введенному числу возвращает его перевод в строку. Например, 34 – тридцать четыре. Кроме того он добавляет рубли и копейки. Научите помощника получать строковую запись числа.