МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Практическое занятие № 7/4ч.

Разработка мобильных компонент анализа безопасности информационно-аналитических систем

(наименование дисциплины (модуля) в соответствии с учебным планом)

Уровень: бакалавриат
(бакалавриат, магистратура, специалитет)

Форма обучения: очная
(очная, очно-заочная, заочная)

Направление(-я) подготовки: 10.05.04 Информационно-аналитические системы безопасности
(код(-ы) и наименование(-я))

Институт: комплексной безопасности и специального приборостроения ИКБСП
(полное и краткое наименование)

Кафедра: КБ-4 «Прикладные информационные технологии»
(полное и краткое наименование кафедры, реализующей дисциплину (модуль))

Используются в данной редакции с учебного года 2021/22
(учебный год цифрами)

Проверено и согласовано «____» _____20___г. _____ _____
(подпись директора Института/Филиала с расшифровкой)

Москва 2021 г.

МИРЭА

ОГЛАВЛЕНИЕ

МИРЭА

1 СЕТЕВОЕ ВЗАИМОДЕЙСТВИЕ В OS ANDROID

При реализации сетевой части приложения требуется учитывать следующие особенности:

1. Количество трафика. Пользователь не всегда имеет Wi-Fi-соединение, а мобильный интернет имеет ограничения.
2. Лимит заряда устройства.
3. Безопасность. Требуется защита передаваемой информации от клиента к серверу и обратно.

Существуют несколько типов реализации сетевого взаимодействия. Первый подход реализован на основе сокетов (работа непосредственно с Socket API, рисунок 1.1). Он часто используется в приложениях, где важна скорость доставки сообщения, важен порядок доставки сообщений и необходимость поддержки стабильного соединения с сервером. Такой способ зачастую реализуется в мессенджерах и играх.

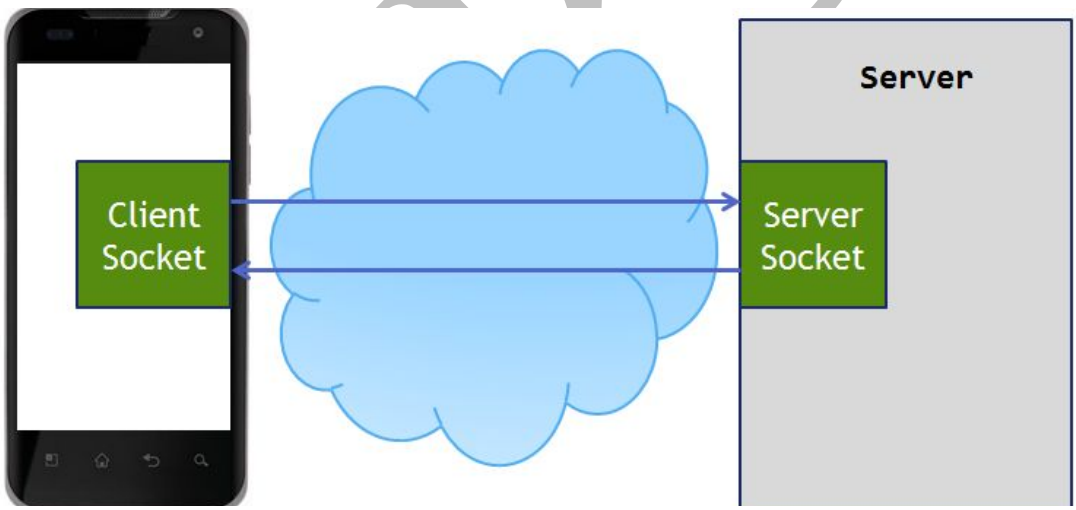


Рисунок 1.1 - Socket API

Второй подход — это частые опросы (polling): клиент посылает запрос на сервер и сообщает: «Верните новые данные»; сервер отвечает на запрос клиента и отдает все, что у него накопилось к этому моменту (рисунок 1.2). Минус такого подхода заключается в том, что клиент не знает, появились ли свежие данные на сервере, т.е. присутствует избыточный трафик, в первую очередь из-за частых установок соединений с сервером.

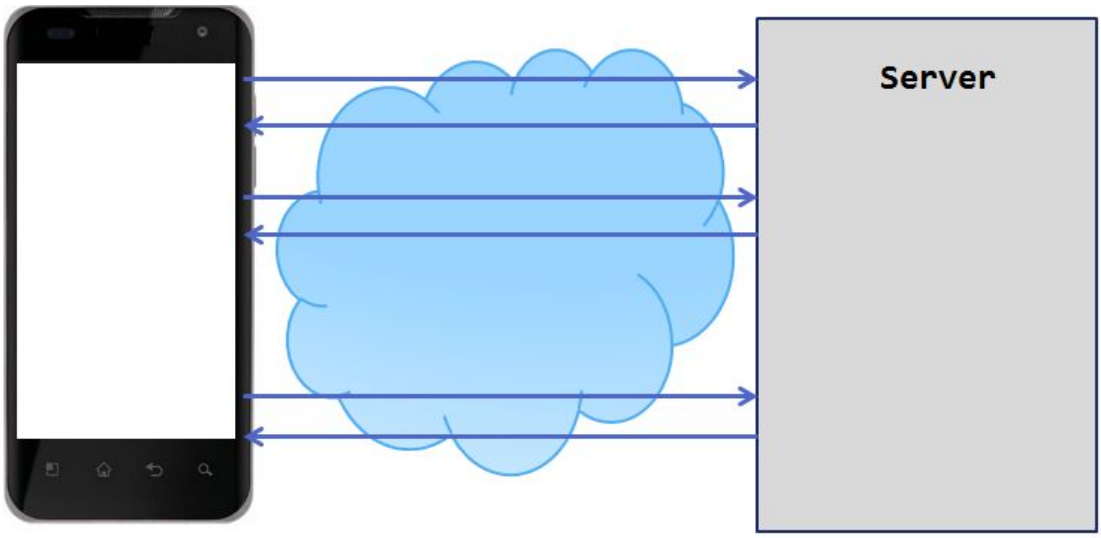


Рисунок 1.2 - Pooling

Третий подход — длинные опросы (long polling) — заключается в том, что клиент посылает «ожидающий» запрос на сервер. Сервер смотрит, есть ли свежие данные для клиента, если их нет, то он держит соединение с клиентом до тех пор, пока эти данные не появятся (рисунок 1.3). Как только данные появились, он «пушит» их обратно клиенту. Клиент, получив данные от сервера, тут же посылает следующий «ожидающий» запрос и т.д.

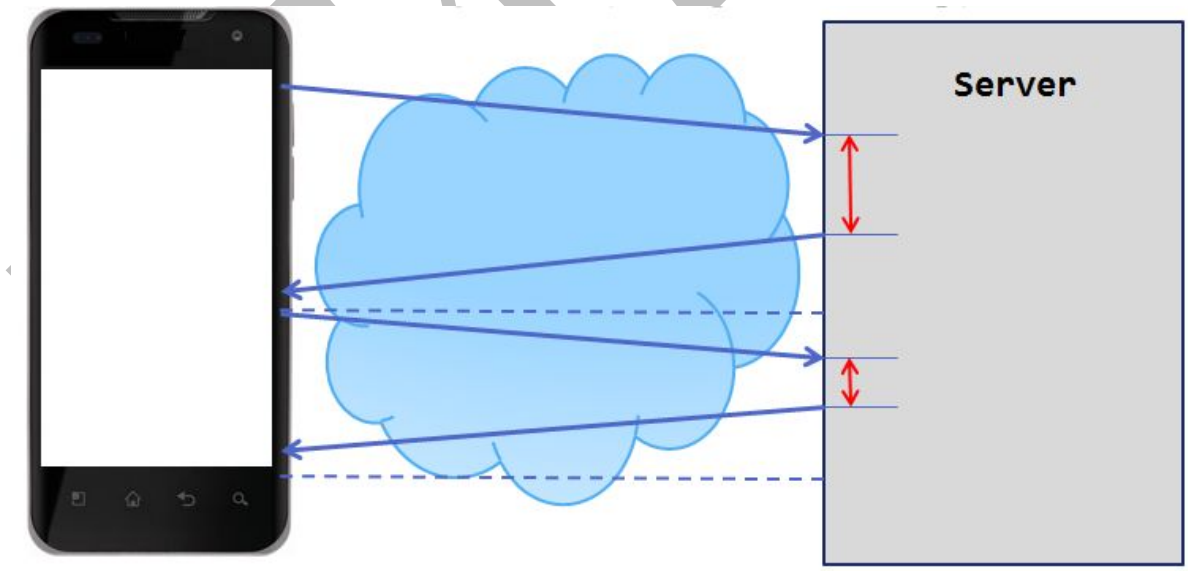


Рисунок 1.3 - Long polling

Реализация данного подхода достаточно сложна на мобильном клиенте в первую очередь из-за нестабильности мобильного соединения. При этом подходе трафика расходуется меньше, чем при обычном polling’е, т.к. сокращается

количество установок соединений с сервером.

Механизм long polling, или пуш-уведомлений (push notifications), реализован в самой платформе Android (Сервис Firebase). Для большинства задач будет лучше использовать его. Тем самым разрывается связь непосредственно между сервером и клиентом за счет того, что сервер работает с сервисом Firebase и отправляет свежие данные всегда на этот сервис, а он уже в свою очередь реализует всю логику доставки этих данных до клиентского приложения. Плюсы данного подхода заключаются в том, что устраняется необходимость частых установок соединения с сервером за счет того, что разработчик точно знает, что данные появились, и об этом вас оповещает сервис.

Из этих четырех подходов наиболее популярными при разработке бизнес-приложений являются пуш-уведомления и частые опросы. При реализации данных подходов приходится устанавливать соединение с сервером и передавать данные. Далее речь пойдет об инструментах, которые есть в наличии у разработчика для работы по HTTP/HTTPS-протоколам

1.1 HttpUrlConnection и HttpClient

Существует два класса для работы с протоколами HTTP/HTTPS. Первый – это java.net.HttpURLConnection, второй – org.apache.http.client.HttpClient. Обе эти библиотеки включены в Android SDK.

Класс java.net.HttpURLConnection является подклассом java.net.URLConnection и позволяет реализовать работу по отправке и получении данных из сети по протоколу HTTP. Данные могут быть любого типа и длины. Родительский класс URLConnection был спроектирован для работы не только по HTTP-протоколу, а еще по таким, как file, mailto, ftp и т.п.

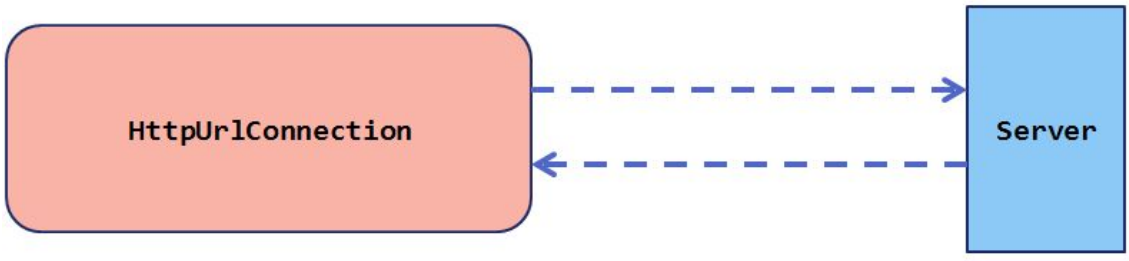


Рисунок 1.4 - Схема взаимодействия сервера с HttpUrlConnection

МИРЭА

HttpClient спроектирован более объектно-ориентированно. В нем есть четкое разделение абстракций. В самом простом случае разработчик использует пять разных интерфейсов: HttpRequest, HttpResponse, HttpEntity и HttpContext (рисунок 1.5). Данный клиент намного тяжеловеснее HttpUrlConnection.

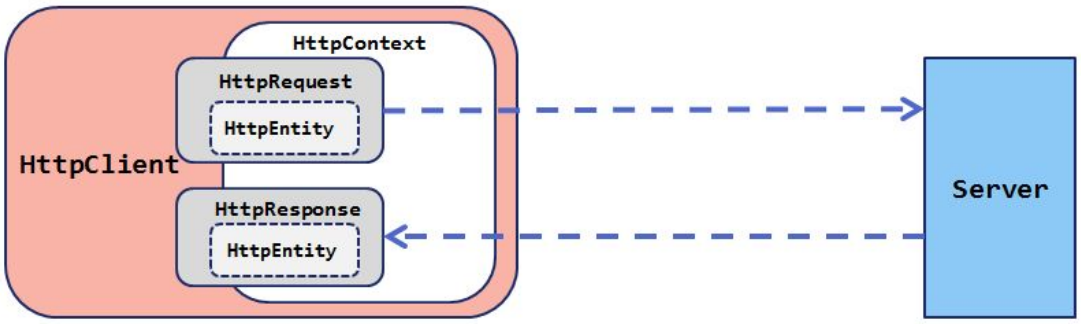


Рисунок 1.5 - Схема взаимодействия сервера с HttpClient

Как правило, на всё приложение существует один экземпляр класса HttpClient. Это обусловлено его тяжеловесностью. Использование отдельного экземпляра на каждый запрос является расточительством ресурсов. Возможно, к примеру, хранить экземпляр HTTP-клиента в наследнике класса Application (рисунок 1.6).

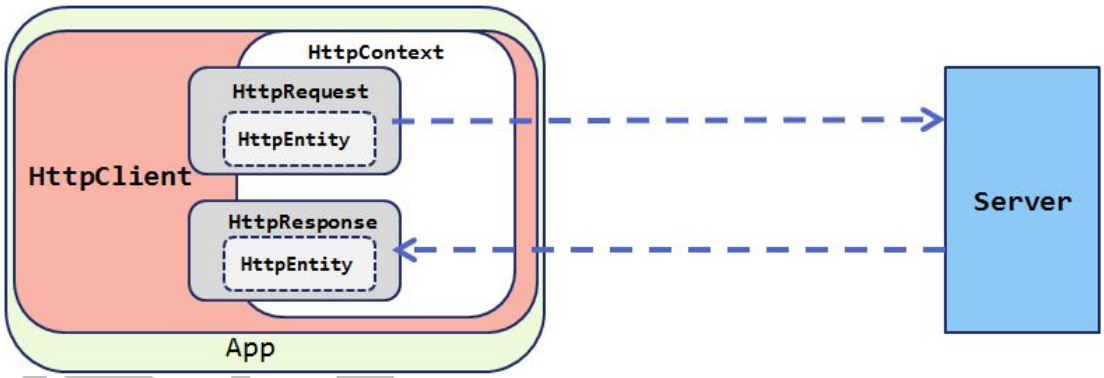


Рисунок 1.6 - Схема взаимодействия сервера с HttpClient

В случае HttpUrlConnection следует создавать на каждый запрос новый экземпляр клиента (рисунок 1.7).

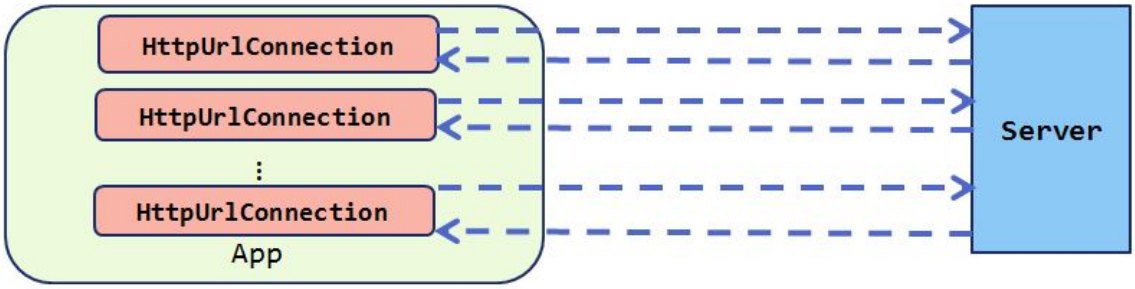


Рисунок 1.7 - Схема взаимодействия сервера с HttpClient

Во время работы стандартного сетевого приложения схема обмена данными представлена на рисунке 1.8.

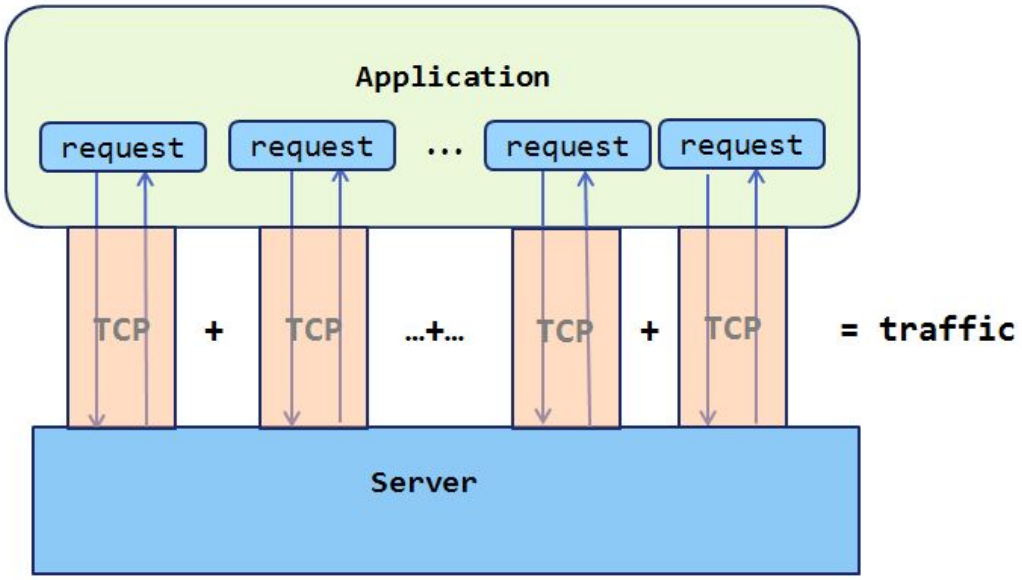


Рисунок 1.8 - Типовая схема функционирования клиент-серверного приложения

Количество и частота запросов зависят от функционала UI – интерфейса приложения. Каждый запрос устанавливает TCP-соединение с сервером. В данном случае трафик, который будет потрачен, будет равняться сумме установок соединений и сумме переданных данных. Понизить расход трафика в данном случае возможно за счет использования долгоживущего соединения (keep alive).

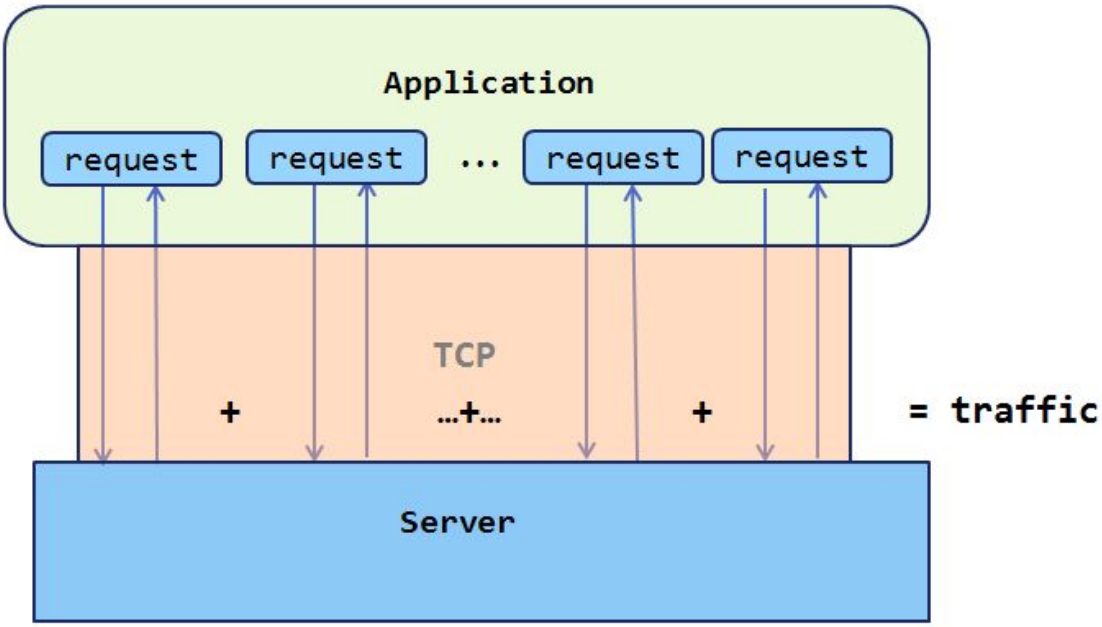


Рисунок 1.9 – Типовая схема функционирования клиент-серверного приложения с использование параметра keep-alive

Использование параметра keep alive-соединения заключается в использовании одного и то же TCP-соединения для отправки и приема HTTP-запросов. Главные преимущества — снижение трафика и времени выполнения запроса. Далее приведён пример создания 10 тестовых запросов с различным значением параметра keepAlive. Данные представлены в таблице ниже.

Номер запроса	Время(ms) KeepAlive = false	Время(ms) KeepAlive = true
1	2098	2023
2	2157	1604
3	2037	1698
4	2096	1774
5	1944	1173
6	2055	1573
7	1865	1683
8	2119	1670
9	1986	1666
10	1965	1541
	≈2032,2	≈1700,5

При использовании keep alive видно, что среднее время выполнения запроса составляет примерно две секунды. В случае с включенным keep alive это время снижается до 1,7 секунды, что на 16% быстрее. Это обуславливается в первую очередь тем, что устраняется необходимость частой установки соединения с сервером. При использовании защищенного HTTPS-соединения разница будет заметнее, т.к. процедура SSL Handshake гораздо затратней процедуры TCP Handshake.

Важным параметром keep alive-соединения является keep alive duration. Данный параметр устанавливает временной интервал существования соединения. Если приходит несколько HTTP-запросов в пределах этого интервала, то будет использоваться уже установленное TCP-соединение.

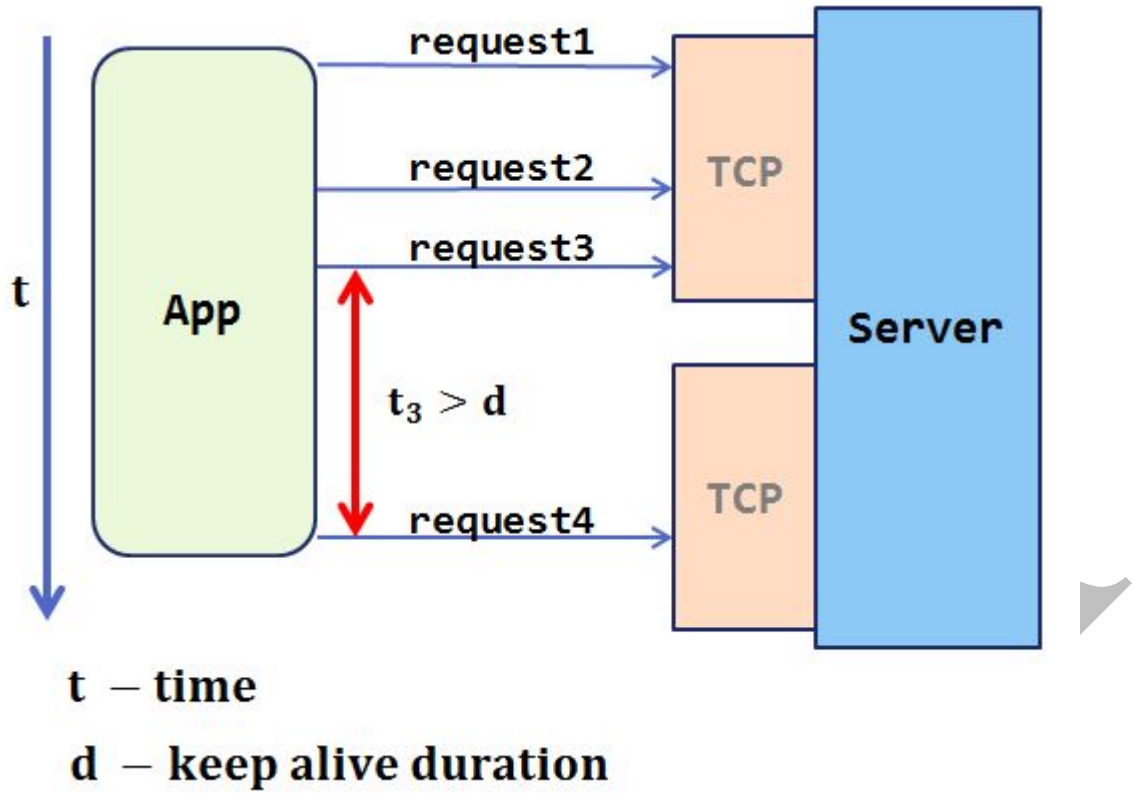


Рисунок 1.10 - Схема функционирования клиент-серверного приложения при установленном параметре keep alive duration

Из рисунка 1.10 представлен пример, когда время между четвертым и третьим запросом превысило keep alive duration, поэтому создается новое TCP-соединение с сервером.

1.2 Формат обмена данными

В настоящее время в качестве формата данных, предназначенные для взаимодействия в интернете, широко используются две технологии: XML и JSON. В какой-то степени они являются конкурентами, хотя и созданы для разных задач. В мобильной разработке, как правило, применяется формат JSON.

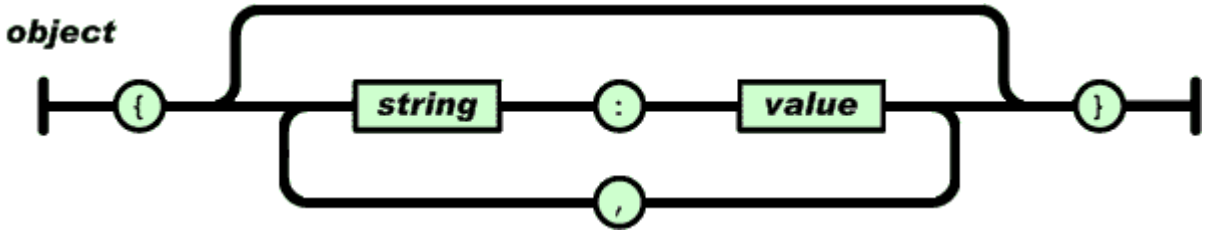
JSON:

- формат обмена данными;
- средство кодирования объектов JavaScript в виде строк;
- более экономичный, чем XML, формат с точки зрения размера данных.

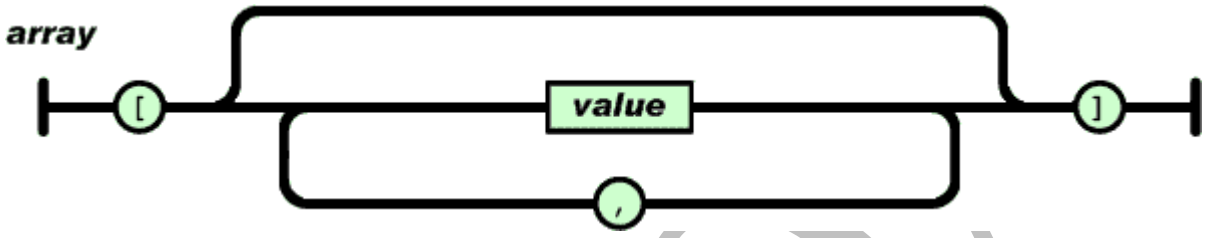
Структуру данных в JSON возможно возможно представить как комбинацию:

- JSON-объектов: {string:value, ... } (фигурные скобки). Представлен как

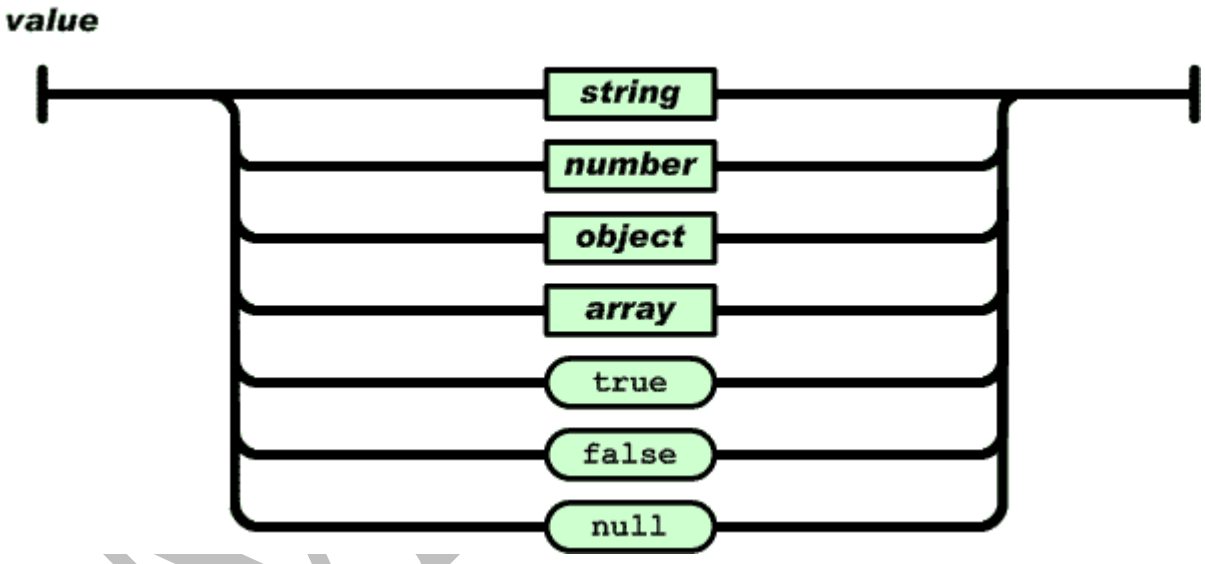
JSONObject;



– JSON-массивов: [value,value .....] (квадратные скобки). Представлен как JSONArray;



– значений: string || number || object || array || true || false, || null. Представлены как JSONStringer и др.;



В Android есть существуют классы для работы с JSON в пакете json.org: JSONArray, JSONObject, JSONStringer, JSONTokenizer, JSONException.

Далее приведено несколько примеров:

```
{
    "name": "mirea",
    "course": 4
}
```

Пример с массивом:

```
{
"name": "mirea",
"course": 4,
"address":
    {
    "postal": 111111,
    "country": "Russia",
    "city": "Moscow",
    "building": "Stromynka,20"
    },
"phoneNumbers":
    [{
    "type": "Ю3",
    "number": 111
    },
    {
    "type": "Стромынка",
    "number": 222
    },
    {
    "type": "Виртуальность",
    "number": 333
    }]
}
```

Объект address содержит несколько полей со своими значениями, а поле phoneNumbers является массивом.

Создание объектов производится следующим образом:

```
JSONObject jsonObject = new JSONObject();
```

Присвоить объекту значение:

```
jsonObject.put("id", 10);
jsonObject.put("name", "mirea");
```

Создать массив:

```
JSONArray jsonArrayMessages = new JSONArray();
JSONObject jsonMessage = new JSONObject();
jsonMessage.put("id_department", 1);
jsonMessage.put("value", "ИКБСП");
jsonArrayMessages.put(jsonMessage);
```

Добавить массив в JSONObject:

```
jsonObject.put("departments", jsonArrayMessages);
```

Создать JSONObject из строки:

```
JSONObject jsonObject = new JSONObject(result);
```

Получить нужную строку из объекта JSON:

```
String name = jsonObject.getString("name");
```

Получить нужный массив:

```
JSONArray jArray = jsonObject.getJSONArray("departments");
```

Получить нужный элемент из массива:

```
JSONObject msg = jArray.getJSONObject(1);
int id_message = msg.getInt("id_department");
```

1.3 Задание. Socket

Требуется создать новый проект *ru.mirea.«фамилия».practice7*. Название модуля SocketConnection.

Для работы с сокетами необходимо создать экземпляр класса Socket, с указанием ip-адреса и порта, а потом считать данные от сервера. После получения данных требуется закрыть сокет. Если будет указан несуществующий адрес, то вернётся исключение UnknownHostException. В интернете существуют веб-серверы, к которым возможно подключиться через сокеты.

Например, сервера времени позволяют узнать время. Сервис, который работает на сервере, возвращает ответ в виде двух строк, где первая строка пустая, а вторая содержит время. Требуется подключиться к сервису и узнать время.

Далее создаётся отдельный класс с методом для чтения поступающих данных.

```
public class SocketUtils {
    /**
     * BufferedReader для получения входящих данных
     */
    public static BufferedReader getReader(Socket s) throws IOException {
        return (new BufferedReader(new InputStreamReader(s.getInputStream())));
    }

    /**
     * Makes a PrintWriter to send outgoing data. This PrintWriter will
     * automatically flush stream when println is called.
     * В примере не используется
     */
    public static PrintWriter getWriter(Socket s) throws IOException {
        // Second argument of true means autoflush.
        return (new PrintWriter(s.getOutputStream(), true));
    }
}
```

Требуется создать экран с одним текстовым полем и кнопкой:

Рисунок 1.11 - Внешний вид приложения Socket

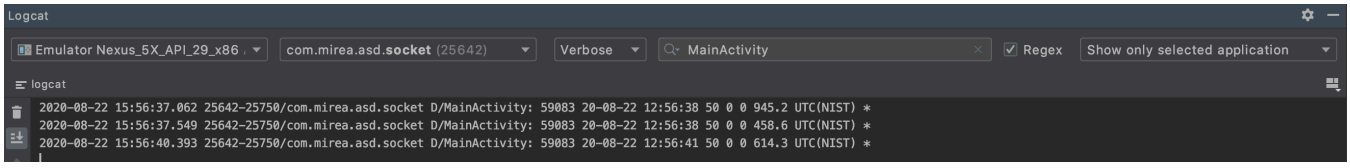
Сервер находится по адресу **time-a.nist.gov** и использует порт 13. Требуется создать константы для этих данных. Также необходимо разрешение на использование интернета в манифесте.

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

В данном примере для выполнения задачи в другом потоке будет использоваться класс AsyncTask. С API 30 данный класс является устаревшим. Более подробно возможно прочитать о механизме возможно по ссылке https://developer.android.com/reference/android/os/AsyncTask?hl=ru. В данном механизме основным методом является doInBackground – в котором выполняются основные вычисления и который возвращает результат вычислений в метод onPostExecute.

```
public class MainActivity extends AppCompatActivity {
    private String TAG = MainActivity.class.getSimpleName();
    private TextView mTextView;
    private String host = "time-a.nist.gov"; // или time-a.nist.gov or time-b.nist.gov
    private int port = 13;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView = findViewById(R.id.textView);
    }

    public void onClick(View view) {
        GetTimeTask timeTask = new GetTimeTask();
        timeTask.execute();
    }


    private class GetTimeTask extends AsyncTask<Void, Void, String> {

        @Override
        protected String doInBackground(Void... params) {
            String timeResult = "";
            try {
                Socket socket = new Socket(host, port);
                BufferedReader reader = SocketUtils.getReader(socket);
                reader.readLine(); // игнорируем первую строку
                timeResult = reader.readLine(); // считываем вторую строку
                Log.d(TAG,timeResult);
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return timeResult;
        }

        @Override
        protected void onPostExecute(String result) {
            super.onPostExecute(result);
            mTextView.setText(result);
        }
    }
}
```

Запустите приложение и нажмите на кнопку. В результате отобразится точное время. В логах отобразится полученная информация с сервера.



1.4 Задание. HttpURLConnection

Создать новый модуль. Название модуля HttpURLConnection. Требуется определить внешний ip адрес устройства.

Класс java.net.HttpURLConnection является подклассом java.net.URLConnection и позволяет реализовать работу по отправке и получении

данных из сети по протоколу HTTP. Данные могут быть любого типа и длины. Данный класс следует использовать для отправки и получения потоковых данных, размеры которых нельзя заранее определить. Используя данный класс, не требуется думать о сокетах и реализовывать собственные способы общения между клиентом и сервером.

Алгоритм использования механизма:

– получить объект HttpURLConnection через вызов URL.openConnection() и привести результат к HttpURLConnection;

– подготовить необходимый запрос. Основное в запросе - сетевой адрес. Также в запросе возможно указать различные метаданные: учётные данные, тип контента, cookies сессии и т.п.;

– опционально загрузить тело запроса. В этом случае используется метод setDoOutput(true). Передача данных, записанных в поток, возвращается через метод getOutputStream();

– прочитать ответ. Заголовок ответа обычно включает метаданные, такие как тип и длина контента, даты изменения, куки сессии. Прочитать данные из потока можно через метод getInputStream(). Если у ответа нет тела, то метод возвращает пустой поток.

– разорвать соединение. После прочтения ответа от сервера HttpURLConnection следует закрыть через вызов метода disconnect(). Тем самым освобождаются ресурсы, занимаемые соединением.

По умолчанию HttpURLConnection использует метод GET. Для использования POST требуется вызвать setDoOutput(true) и отправить данные через openOutputStream(). Другие HTTP-методы (OPTIONS, HEAD, PUT, DELETE and TRACE) настраиваются через метод setRequestMethod(String).

Для работы через прокси-сервер используется URL.openConnection(Proxy) при создании соединения. Каждый экземпляр HttpURLConnection может использоваться только для одной пары запроса/ответа. Операции с соединениями следует проводить в отдельном потоке.

Требуется привести файл разметки activity_main.xml к следующему виду:

МИРЭА

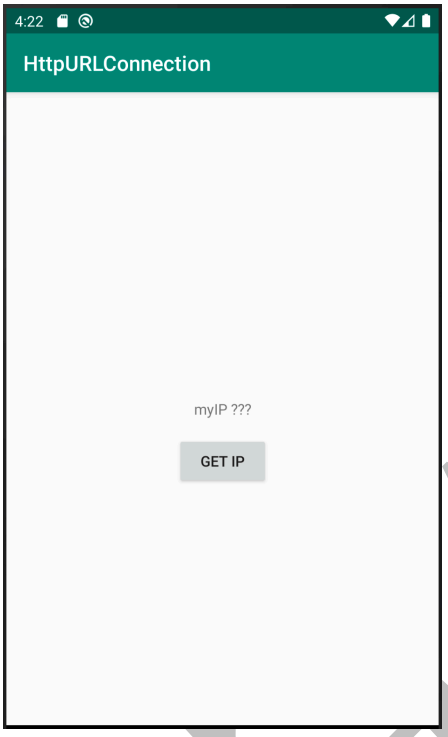


Рисунок 1.12 Внешний вид приложения HttpURLConnection

Необходимо установить требуемые разрешения для работы с интернетом и сетью в манифест-файле:

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

Прежде всего требуется: наличие подключения к интернету и обеспечить загрузку страницы в отдельном потоке.

Проверка подключения к интернету.

```
public void onClick(View view) {
    ConnectivityManager connectivityManager =
            (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkinfo = null;
    if (connectivityManager != null) {
        networkinfo = connectivityManager.getActiveNetworkInfo();
    }

    if (networkinfo != null && networkinfo.isConnected()) {
        new DownloadPageTask().execute(url); // запускаем в новом потоке
    } else {
        Toast.makeText(this, "Нет интернета", Toast.LENGTH_SHORT).show();
    }
}
```

Если подключение к интернету существует, то запускается новый поток AsyncTask (из предыдущего примера) и передаётся ему нужный адрес. Данный класс требуется разместить внутри MainActivity:

МИРЭА

```
private class DownloadPageTask extends AsyncTask<String, Void, String> {
        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            resultTextView.setText("Загружаем...");
        }
        @Override
        protected String doInBackground(String... urls) {
            try {
                return downloadIpInfo(urls[0]);
            } catch (IOException e) {
                e.printStackTrace();
                return "error";
            }
        }
        @Override
        protected void onPostExecute(String result) {
            resultTextView.setText(result);
            Log.d(MainActivity.class.getSimpleName(), result);
            try {
                JSONObject responseJson = new JSONObject(result);
                String ip = responseJson.getString("ip");
                Log.d(MainActivity.class.getSimpleName(), ip);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            super.onPostExecute(result);
        }
    }
    private String downloadIpInfo(String address) throws IOException {
        InputStream inputStream = null;
        String data = "";
        try {
            URL url = new URL(address);
            HttpURLConnection connection = (HttpURLConnection) url
                    .openConnection();
            connection.setReadTimeout(100000);
            connection.setConnectTimeout(100000);
            connection.setRequestMethod("GET");
            connection.setInstanceFollowRedirects(true);
            connection.setUseCaches(false);
            connection.setDoInput(true);
            int responseCode = connection.getResponseCode();

            if (responseCode == HttpURLConnection.HTTP_OK) { // 200 OK
                inputStream = connection.getInputStream();
                ByteArrayOutputStream bos = new ByteArrayOutputStream();
                int read = 0;
                while ((read = inputStream.read()) != -1) {
                    bos.write(read);
                }
                byte[] result = bos.toByteArray();
                bos.close();
                data = new String(result);
            } else {
                data = connection.getResponseMessage() + " . Error Code : " + responseCode;
            }
            connection.disconnect();
            //return data;
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
        }
        return data;
    }
}
```

Код соединения с сервером вынесен в отдельный метод *downloadIpInfo*,

который возвращает текст с указанной страницы. Вначале указывается нужный адрес для загрузки в переменной типа URL. Данный адрес передаётся в метод *doInBackground(...)*, а затем в метод *downloadOneUrl*. Внутри метода создаётся объект HttpURLConnection через вызов *URL.openConnection()*. После этого производится настройка различных параметров запроса и выполняется запрос к серверу.

Далее идёт проверка кода ответа от сервера. Если страница существует и сервер готов отдать её содержание по запросу, то возвращается код 200 (константа *HttpURLConnection.HTTP_OK*). Если страницы не существует, то возвращается код 404. В данном случае нет смысла получать содержимое страницы. Можно вывести собственное сообщение об ошибке, используя полученный код через переменную responseCode или использовать встроенный метод getResponseMessage(), который возвращает стандартные тексты ошибок (в примере я скомбинировал оба варианта). Если проверка прошла успешно, то возможно считывать данные с сервера. Есть различные способы чтения потоков. В данном примере используется класс ByteArrayOutputStream. Итоговый текст попадает в переменную data и выводится в TextView для просмотра. В методе *onPostExecute* представлен пример, как из ответа формируется JSON объект и извлекается значение ip адреса.

Результатом работы приложения будет информация о адресе с которого выходит в интернет пользователь.

Требуется изменить внешний вид экрана для вывода в отдельные текстовые поля значение полей: город, регион и т.д. В примере был представлена механизм извлечения значения ip – адреса и объекта JSON.

МИРЭА

2 СТОРОННИЕ БИБЛИОТЕКИ

2.1 Retrofit

Retrofit – это библиотека для android, предназначенная для организации сетевого взаимодействия приложения с внешними ресурсами. Авторами библиотеки Retrofit являются разработчики из компании Square. Библиотека предназначена для получения и разбора различного вида структурированных данных от вебсервисов, использующие REST. В Retrofit для (де)сериализации данных используются конверторы, которые необходимо указывать вручную. Поддерживаемый список конвертеров представлен ниже:

- Gson: com.squareup.retrofit2:converter-gson;
- Jackson: com.squareup.retrofit2:converter-jackson;
- Moshi: com.squareup.retrofit2:converter-moshi;
- Protobuf: com.squareup.retrofit2:converter-protobuf;
- Wire: com.squareup.retrofit2:converter-wire;
- Simple XML: com.squareup.retrofit2:converter-simplexml;
- JAXB: com.squareup.retrofit2:converter-jaxb;
- Scalars (primitives, boxed, and String): com.squareup.retrofit2:converter-scalars.

Для выполнения HTTP запросов Retrofit используется OkHttp библиотеку.

2.2 FIREBASE.

На конференции Google I/O 2015 была представлена облачная база данных на основе NoSQL с горящим названием Firebase. Год спустя в мае 2016 на этой же конференции было объявлено о революционном изменении. Гугл показал новые возможности - теперь это уже целая платформа для построения Android-, iOS- и мобильных веб-приложений, а не просто база данных в облаке.

База данных позволяет работать с данными, которые хранятся как JSON, синхронизируются в реальном времени и доступны при отсутствии интернета. Firebase поддерживает аутентификацию по связке электронная почта+пароль, Facebook, Twitter, GitHub, Google и другие аутентификационные системы. Кроме базы данных Firebase предлагает хостинг статичных файлов для веб-сайта.

МИРЭА

Firebase содержит следующий компоненты:

	**База данных в реальном времени** Храните и синхронизируйте данные
	**Аутентификация** Аутентификация пользователей через облачный сервис
	**Облачное хранилище** Хранение и получение файлов с облака
	**Тестовая лаборатория для Android** Тестирование приложений на устройствах в Google
	**Отчеты о сбоях** Обнаружение багов и их исправление. Сбор и отправка важной информации, которая может помочь в поиске проблем iOS/Android-приложений после релиза.
	**Облачные функции** Мобильный бэкенд
	**Хостинг** Обеспечивает быструю доставку ресурсов для приложений и безопасностью
	**Отслеживание быстродействия** Изучение быстродействия приложения
	**Google Analytics** Бесплатная и неограниченная аналитика для приложения
	**AdMob** Настройка в приложении рекламы
	**Облачные сообщения** Сервис для доставки push-уведомлений из облака на устройства
	**Удаленное конфигурирование** Позволяет подстраивать и обновлять элементы приложения на лету без необходимости обновлять пакет приложения и ожидания, пока он станет доступен в магазине приложений, а затем обновится у пользователей. Возможность включение и выключение определённых элементов приложения, распространение обновлений на конкретные Аудитории пользователей.
	**App Indexing** Привлечение поискового трафика в приложении
	**AdWords** Целевые рекламные кампании

https://firebase.google.com/- стартовая страница, в которой находится

МИРЭА

документация. В ней возможно увидеть ссылку на консоль, в которой находятся проекты для взаимодействия с данными.

В консоли имеется возможность создавать новые проекты, просматривать данные пользователей, управлять файлами, работать с базой данных.

Большая часть продуктов, включая Analytics, Crash Reporting, Remote Config и Dynamic Links — полностью бесплатны и не имеют каких-либо ограничений. Платные же сервисы — Test Lab, Storage, хостинг — имеет гибкую ценовую сетку. Бесплатный тариф SPARK с некоторыми ограничениями, отлично подойдёт для первых шагов, прототипа приложения, курсовой или дипломной работы, начала стартапа. Фиксированная ставка на тарифе FLAME для тех, кому нужна предсказуемая ежемесячная стоимость на ранних этапах развития приложения. Конфигурируемый тариф BLAZE для самых крупных клиентов.

Далее требуется аутентифицироваться в сервисе (требуется учетная запись в сервисах google).

2.3 Firebase Authentication

Firebase имеет несколько способов для проведения аутентификации: по электронному адресу и паролю, учётным записям Facebook, Twitter, GitHub, анонимно и другие возможности.

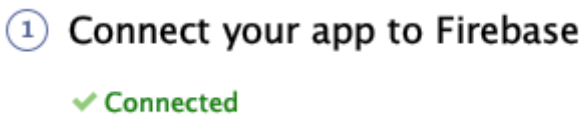
2.4 Задание

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Название приложения FirebaseAuth.

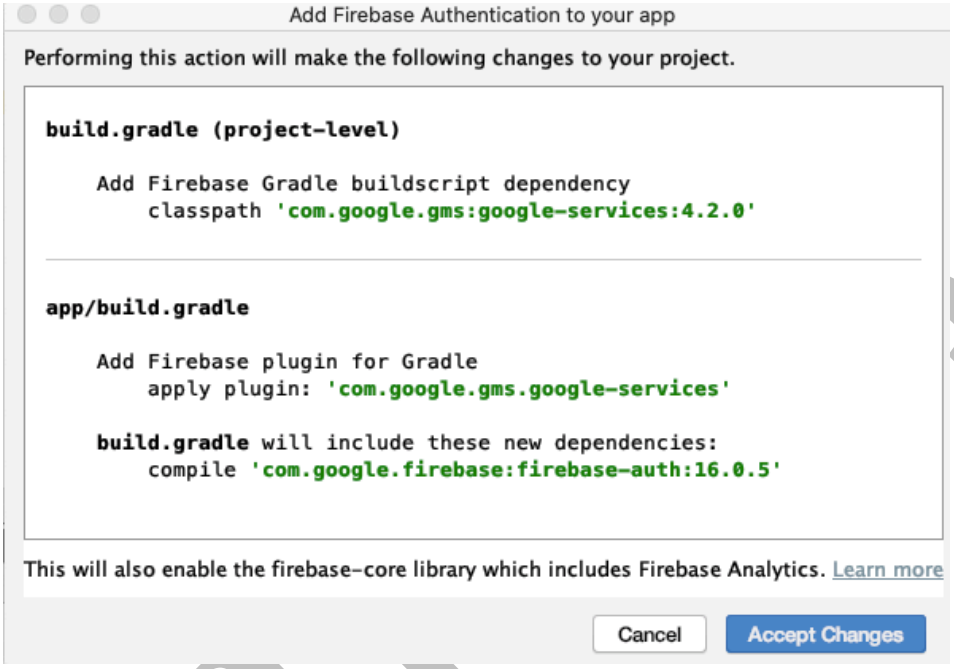
Сервис аутентификации - идентификация пользователей необходима в большинстве приложений. Это позволяет разделять доступ, надежно хранить личные данные пользователей в облаке и обеспечить персонализированный контент на всех устройствах пользователя.

Требуется зарегистрироваться в системе (необходим Google-аккаунт). В AndroidStudio выбрать Tools > Firebase > Authentication > «Email and password authentication» > «Connect to Firebase» > ввести NameProject > Соединение. После этого первый шаг должен быть отмечен следующим образом:

МИРЭА

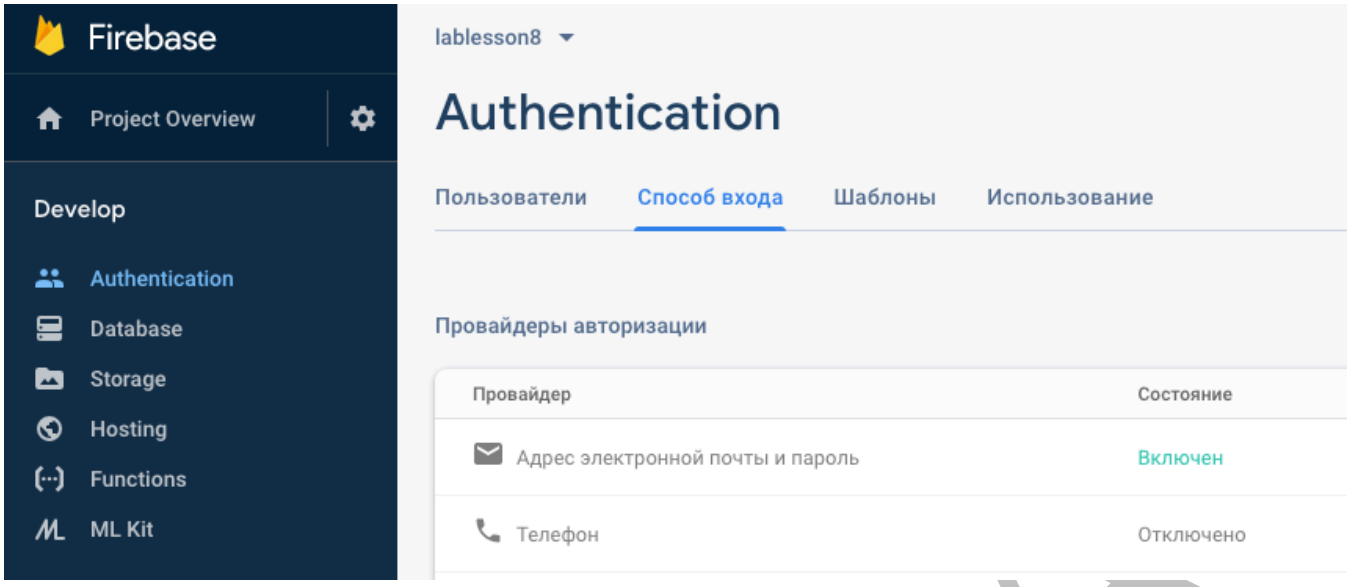


Далее требуется добавить зависимости в проект «Add Firebase Authentication to your app»



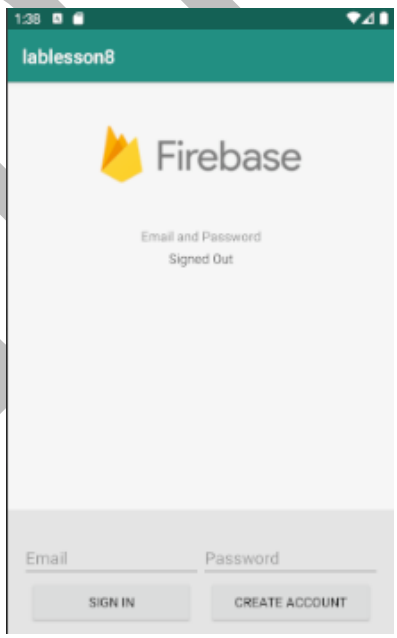
Подключено только ядро библиотеки Firebase, существуют и другие модули библиотеки, которые нужно подключать в случае необходимости. Далее осуществляется переход в консоль.

Сервис аутентификации тесно интегрируется с другими сервисами Firebase, использует отраслевые стандарты, такие как OAuth 2.0 и OpenID Connect, так что он может быть легко интегрирован с бэкэндом. Далее рассматривается метод аутентификации пользователей с помощью адреса электронной почты и пароля. Firebase Authentication SDK предоставляет методы для создания и управления пользователями, которые используют адреса электронной почты и пароли для входа в систему. Требуется установить в консоле поддерживаемый вид аутентификации.



В процессе регистрации пользователей в приложении информация о них будет появляться на вкладке «Пользователи». Здесь возможно управлять пользователями — например, добавить пользователя в базу приложения, а также отключить или удалить пользователя.

Создайте экран аутентификации, позволяющий производить аутентификацию пользователей и создание аккаунтов:



В файл res>values>strings.xml добавить следующее содержимое:

МИРЭА

```
<resources>
    <string name="app_name">lablesson8</string>

    <string name="label_emailpassword">Email/Password Authentication</string>
    <string name="desc_emailpassword">Use an email and password to authenticate with
Firebase.</string>
    <string name="hint_user_id">User ID</string>
    <string name="sign_in">Sign In</string>
    <string name="create_account">Create Account</string>
    <string name="sign_out">Sign Out</string>
    <string name="verify_email">Verify Email</string>
    <string name="signed_in">Signed In</string>
    <string name="signed_out">Signed Out</string>
    <string name="auth_failed">Authentication failed</string>
    <string name="firebase_status_fmt">Firebase UID: %s</string>
    <string name="firebase_user_management">Firebase User Management</string>
    <string name="emailpassword_status_fmt">Email User: %1$s (verified: %2$b)</string>
    <string name="emailpassword_title_text">Email and Password</string>
    <string name="error_sign_in_failed">Sign in failed, see logs for details.</string>
</resources>
```

В классе MainActivity требуется объявить элементы пользовательского интерфейса, а также объекта класса FirebaseAuth и его слушателя AuthStateListener. Класс FirebaseAuth — это точка входа в Firebase Authentication SDK. А интерфейс FirebaseAuth.AuthStateListener вызывается, когда происходит изменение в состоянии аутентификации.

В методе onCreate инициализируются все элементы экрана и присваиваются слушатели кнопкам. В методах onStart() проверяется пользователь аутентифицирован и вызывается метод обновления экрана.

МИРЭА

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private static final String TAG = MainActivity.class.getSimpleName();
    private TextView mStatusTextView;
    private TextView mDetailTextView;
    private EditText mEmailField;
    private EditText mPasswordField;
    // START declare_auth
    private FirebaseAuth mAuth;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Views
        mStatusTextView = findViewById(R.id.status);
        mDetailTextView = findViewById(R.id.detail);
        mEmailField = findViewById(R.id.fieldEmail);
        mPasswordField = findViewById(R.id.fieldPassword);
        // Buttons
        findViewById(R.id.emailSignInButton).setOnClickListener(this);
        findViewById(R.id.emailCreateAccountButton).setOnClickListener(this);
        findViewById(R.id.signOutButton).setOnClickListener(this);
        findViewById(R.id.verifyEmailButton).setOnClickListener(this);
        // [START initialize_auth]
        mAuth = FirebaseAuth.getInstance();
        // [END initialize_auth]
    }
    // [START on_start_check_user]
    @Override
    public void onStart() {
        super.onStart();
        // Check if user is signed in (non-null) and update UI accordingly.
        FirebaseUser currentUser = mAuth.getCurrentUser();
        updateUI(currentUser);
    }
```

Экземпляр класса FirebaseUser предоставляет сведения о профиле пользователя в базе данных пользователей проекта Firebase. Он также содержит вспомогательные методы для изменения или получения информации о профиле, а также для управления состоянием аутентификации пользователя. Экземпляр данного класса передаётся в метод updateUI, принимающий экземпляр текущего пользователя. В этом методе происходит вывод информации о текущем пользователе в текстовые поля на экране приложения, а также регулируется видимость элементов пользовательского интерфейса в зависимости от того, авторизован пользователь или нет.

```
private void updateUI(FirebaseUser user) {
    if (user != null) {
        mStatusTextView.setText(getString(R.string.emailpassword_status_fmt,
                user.getEmail(), user.isEmailVerified()));
        mDetailTextView.setText(getString(R.string.firebase_status_fmt, user.getUid()));

        findViewById(R.id.emailPasswordButtons).setVisibility(View.GONE);
        findViewById(R.id.emailPasswordFields).setVisibility(View.GONE);
        findViewById(R.id.signedInButtons).setVisibility(View.VISIBLE);

        findViewById(R.id.verifyEmailButton).setEnabled(!user.isEmailVerified());
    } else {
        mStatusTextView.setText(R.string.signed_out);
        mDetailTextView.setText(null);

        findViewById(R.id.emailPasswordButtons).setVisibility(View.VISIBLE);
        findViewById(R.id.emailPasswordFields).setVisibility(View.VISIBLE);
        findViewById(R.id.signedInButtons).setVisibility(View.GONE);
    }
}
```

Перед отправкой данных на сервер требуется реализовать проверку значений, которые ввёл пользователь:

```
private boolean validateForm() {
    boolean valid = true;

    String email = mEmailField.getText().toString();
    if (TextUtils.isEmpty(email)) {
        mEmailField.setError("Required.");
        valid = false;
    } else {
        mEmailField.setError(null);
    }

    String password = mPasswordField.getText().toString();
    if (TextUtils.isEmpty(password)) {
        mPasswordField.setError("Required.");
        valid = false;
    } else {
        mPasswordField.setError(null);
    }

    return valid;
}
```

В методе createAccount производится создание новой учетной записи пользователя, связанную с указанным адресом электронной почты и паролем. В базе данных проекта Firebase email служит уникальным идентификатором, а также используется для отправки письма сброса пароля.

```
private void createAccount(String email, String password) {
    Log.d(TAG, "createAccount:" + email);
    if (!validateForm()) {
        return;
    }
    // [START create_user_with_email]
    mAuth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        // Sign in success, update UI with the signed-in user's information
                        Log.d(TAG, "createUserWithEmail:success");
                        FirebaseUser user = mAuth.getCurrentUser();
                        updateUI(user);
                    } else {
                        // If sign in fails, display a message to the user.
                        Log.w(TAG, "createUserWithEmail:failure", task.getException());
                        Toast.makeText(MainActivity.this, "Authentication failed.",
                                Toast.LENGTH_SHORT).show();
                        updateUI(null);
                    }
                }
            });
    // [END create_user_with_email]
}
```

В случае успешного создания учетной записи пользователя, будет автоматически выполнен вход данного пользователя в приложении.

Создание учетной записи пользователя может завершиться ошибкой, если учетная запись уже существует, email некорректный или пароль недостаточно сильный. Вот список исключений, которые могут быть выброшены при этом:

– FirebaseAuthWeakPasswordException – пароль не является достаточно сильным;

– FirebaseAuthInvalidCredentialsException – email адрес имеет неправильный формат;

– FirebaseAuthUserCollisionException – уже существует учетная запись с таким email.

После создания учётной записи возможно пройти аутентификацию. Для этого требуется:

```
private void signIn(String email, String password) {
    Log.d(TAG, "signIn:" + email);
    if (!validateForm()) {
        return;
    }
    // [START sign_in_with_email]
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        // Sign in success, update UI with the signed-in user's information
                        Log.d(TAG, "signInWithEmail:success");
                        FirebaseUser user = mAuth.getCurrentUser();
                        updateUI(user);
                    } else {
                        // If sign in fails, display a message to the user.
                        Log.w(TAG, "signInWithEmail:failure", task.getException());
                        Toast.makeText(MainActivity.this, "Authentication failed.",
                                Toast.LENGTH_SHORT).show();
                        updateUI(null);
                    }

                    // [START_EXCLUDE]
                    if (!task.isSuccessful()) {
                        mStatusTextView.setText(R.string.auth_failed);
                    }
                    // [END_EXCLUDE]
                }
            });
    // [END sign_in_with_email]
}
```

Метод signIn принимает в качестве аргументов email и пароль, проверяет их и затем выполняет попытку авторизации пользователя с помощью метода signInWithEmailAndPassword. Возможные исключения:

– FirebaseAuthInvalidUserException возникает, если учетная запись пользователя с таким email не существует или отключена.

– FirebaseAuthInvalidCredentialsException возникает, если введен неправильный пароль.

Следующий метод — signOut — выполняется при нажатии кнопки выхода и вызывает одноименный метод класса FirebaseAuth. Выход из аккаунта текущего пользователя и удаление его из кэша.

```
private void signOut() {
    mAuth.signOut();
    updateUI(null);
}
```

Осталось обработать нажатия кнопок на экране и вызвать соответствующие методы:

```
@Override
public void onClick(View v) {
    int i = v.getId();
    if (i == R.id.emailCreateAccountButton) {
        createAccount(mEmailField.getText().toString(), mPasswordField.getText().toString());
    } else if (i == R.id.emailSignInButton) {
        signIn(mEmailField.getText().toString(), mPasswordField.getText().toString());
    } else if (i == R.id.signOutButton) {
        signOut();
    } else if (i == R.id.verifyEmailButton) {
        sendEmailVerification();
    }
}
```

Требуется запустить проект и создать пользователя. Затем перейти в консоль и проверить наличие созданного пользователя.

МИРЭА

3 КОНТРОЛЬНОЕ ЗАДАНИЕ

В проекте MireaProject. Добавить экран входа в приложение связанный с Firebase. Для этого требуется создать activity и в manifest указать.

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

После успешной авторизации произвести переход на главный экран. Добавить фрагмент, отображающий любую информацию из сетевого ресурса. Возможно использование различных способов сетевого взаимодействия (напр. Retrofit).

МИРЭА