HTTP Client

Kaum eine App kommt ohne den Zugriff auf weitere Webservices aus. Ein zentrales Element der mobilen Softwareentwicklung ist daher der Zugriff auf Webservices. Das stellt Android die Klasse `HttpURLConnection` zur Verfügung, mit der via `GET` und `POST` Requests auf URLs zugegriffen und das Ergebnis verarbeitet werden kann.

Der Rückgabetyp ist `InputStream` und kann damit mit Java Boardmitteln genauso wie eine Textdatei gelesen werden.

Da wir nun auf das Internet zugreifen wollen, müssen wir zu allererst unserer App die entsprechende Berechtigung verschaffen. Dazu genügt ein Eintrag in der Manifest-Datei, da die Permission `Internet` zu den Standard-Permissions und nicht zu den gefährlichen Permissions zählt. Eine zusätzliche Abfrage der Permission im Code ist daher nicht erforderlich.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.eaustria.http_demo">
    <uses-permission android:name="android.permission.INTERNET" />
    <application>
        ...
    </application>

</manifest>
```

Änderungen ab Android API Level 28

Seit Android 9 (API Level 28) ist der Zugriff auf nicht verschlüsselte Webseite standardmäßig nicht mehr erlaubt.

Man erhält die Fehlermeldung:

```
com.android.okhttp.HttpHandler$CleartextURLFilter.checkURLPermitted(HttpHandler.java:115)
```

Um den Zugriff auf `http` ohne SSL-Verschlüsselung zu erlauben, muss man im Manifest einen entsprechenden Eintrag hinzufügen:

```
android:usesCleartextTraffic="true"
```

Das Manifest File sieht dann folgendermaßen aus:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.eaustria.bazar">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
```

```
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="true">
        <activity android:name=".MainActivity">
            ...
    </application>
</manifest>
```

REST-Webservice Architektur

Die Idee hinter REST *(Representational State Transfer)* ist die Verwendung von Standard HTML-Requests:

Methode	Beschreibung	SQL-Pendant	Idempotent
GET	Klassische Abfrage	SELECT	x
POST	Anlegen eines neuen Datensatzes	INSERT	
PUT	Modifikation bestehender Daten	UPDATE	X
DELETE	Löscht Datensatz	DELETE	x

Idempotent: (lat.: *idem*: selbst, *potentia*: Vermögen, Kraft, Wirksamkeit) bedeutet in der Informatik, dass ein mehrfacher Methodenaufruf die gleichen Ergebnisse liefert. Führt man die Abfrage also mehrfach durch, so erhält man immer die gleichen Ergebnisse.

Natürlich kann sich jedoch der Entwickler eines Webservices für eine andere Verwendung der Request-Typen entscheiden. Man könnte natürlich auch mit `GET` Datensätze löschen. Dies wäre jedoch ein etwas *schräges* Verhalten eines Webservices. Ich könnte mir keinen Grund denken, warum ich ein Webservice *schräg* implementieren würde.

Ablauf für den Abruf eines Webserices in Android:

1. **Eigenen Thread starten**: In aktuellen Androidversionen müssen sämtliche "Internet-Tätigkeiten" in einem eigenen Thread ausgeführt werden, da diese länger dauern und somit das UI blockieren könnten.
2. **Connection auf Webservice aufbauen**
3. **Responsecode prüfen**
4. **InputStream auswerten**
5. **Erhaltene Daten verarbeiten**

aufgrund der Forderung von Android nach Nebenläufigkeit beim Zugriff auf Webservices bietet sich der Einsatz vom AsyncTask-Framework an.

GET-Abruf

Beim GET-Request werden die Parameter des Aufrufs direkt an die URL angehängt! Wir wollen uns alle Repositories eines bestimmten Users auf github

anzeigen lassen.

```
public void showGitHubRepos(View view) {
    GitHubServerTask task = new GitHubServerTask();
    task.execute(mUserName.getText().toString());
}
```

Dafür verwenden wir die API von github (siehe: https://developer.github.com/v3/repos/#list-user-repositories) und führen einen GET-Request auf folgende URL durch: `https://api.github.com/users/[USERNAME]/repos` - anstelle von [USERNAME] fügen wir einen Benutzernamen (zB. unseren eigenen) als Aufrufparameter ein. Wir können aber die öffentlichen Repositories von jedem beliebigen Benutzer abfragen.

The Klasse `HttpURLConnection` führt die konkrete Abfrage durch. Die Anfrageparameter werden als Teil der URL mitübergeben.

Sobald die Methode `getResponseCode` aufgerufen wird, wird die Abfrage durchgeführt.

Der Responsecode beinhaltet die Antwort vom Server. Dieser Code entspricht den HTML Codes, die auch als Konstante zur Verfügunge stehen.

Als Antwort erhält man einen InputStream, von dem die Daten gelesen werden können. Fast alle Webservices bieten die Antwort im JSON Format an, weshalb in der Regel danach das Parsen der JSON Antwort erforderlich ist (im Demobeispiel erfolgt dies in der Methode `outputRepoNames`).

```
private class GitHubServerTask extends AsyncTask<String, Integer, String> {

    private final String TAG = GitHubServerTask.class.getSimpleName();

    @Override
    protected String doInBackground(String... strings) {
        Log.d(TAG, "entered doInBackground");
        String username = strings[0];
        Log.d(TAG, "url: "+URL+username+"/repos");
        String sJson = "";
        try {
            HttpURLConnection connection =
                    (HttpURLConnection) new URL(URL+username+"/repos").openConnection();
            connection.setRequestMethod("GET");
            connection.setRequestProperty("Content-Type", "application/json");
            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                BufferedReader reader = new BufferedReader(
                        new InputStreamReader(connection.getInputStream()));
                sJson = readResponseStream(reader);
            }
```

```
        } catch (IOException e) {
            Log.d(TAG, e.getLocalizedMessage());
        }
        return sJson;
    }

    private String readResponseStream(BufferedReader reader) throws IOException {
        Log.d(TAG, "entered readResponseStreaulat");
        StringBuilder stringBuilder = new StringBuilder();
        String line = "";
        while ( (line=reader.readLine()) != null) {
            stringBuilder.append(line);
        }
        return stringBuilder.toString();
    }

    @Override
    protected void onPostExecute(String s) {
        Log.d(TAG, "entered onPostExecute");
        outputRepoNames(s, mRepoList);
        super.onPostExecute(s);
    }

    private void outputRepoNames(String s, EditText mRepoList) {
        Log.d(TAG, "entered outputRepoNames");
        Log.d(TAG, s);
        StringBuilder stringBuilder = new StringBuilder();
        try {
            JSONArray array = new JSONArray(s);
            for (int i=0 ; i < array.length() ; i++) {
                JSONObject jsonObject = array.getJSONObject(i);
                stringBuilder.append(jsonObject.optString("full_name"));
                stringBuilder.append(("\n\n"));
            }
        } catch (JSONException e) {
            Log.d(TAG, e.getLocalizedMessage());
        }
        mRepoList.setText(stringBuilder.toString());
    }
}
```

POST Aufruf

Der POST-Aufruf erfolgt ähnlich wie GET. Der einzige Unterschied besteht darin, dass die Aufruf-Parameter nicht an die URL angehängt werden, sondern in ein PostParameter Objekt verpackt werden.

```
private byte[] getPostBytes() throws JSONException {
    JSONObject postParams = new JSONObject();
    postParams.put("paramA", "first param value");
    postParams.put("paramB", 123456);
    String body = postParams.toString();
    Log.d(TAG, "body: " + body);
    return body.getBytes();
}
```

Beim Aufbau der Verbindung zum Webservice müssen diese Post-Parameter nun in der Form des Bytes Arrays mitübergeben werden:

```
private HttpURLConnection getPostConnection( byte[] data, String url)
    throws IOException {

    HttpURLConnection connection =
        (HttpURLConnection) new URL(url).openConnection();
    connection.setDoOutput(true);
    connection.setRequestMethod("POST");
    connection.setRequestProperty("Content-Type", "application/json");
    connection.setFixedLengthStreamingMode(data.length);
    connection.getOutputStream().write(data);
    connection.getOutputStream.flush();
    return connection;
}
```

Der Aufruf erfolg nun genauso wie bei GET in einem eigenen Thread.

PUT/DELETE Requests

PUT und DELETE Aufrufe erfolgen genauso wie get, nur dass die entsprechende Request Methode gesetzt werden muss:

```
connection.setRequestMethod("PUT")
```