Syntax Institut

Modul 3 – Android App Entwicklung mit Kotlin

API Calls

Gliederung

- RESTful Service
- URI, URL und HTTP
- retrofit
- Beispiel

Quelle: https://www.booknerds.de/2017/01/the-it-crowd-version-1-0-5-0-die-endgueltige-vollversion-tv-serie-dvd/

RESTful Service

Representational State Transfer

gängige Architektur für Webserver

Webservices in REST Architektur nennt man auch

RESTful Services

Serverresourcen werden über eine URI/URL

angesprochen

Kommunikation läuft über HTTP Protokoll

Quelle: https://www.opc-router.de/was-ist-rest/

URI, URL, HTTP

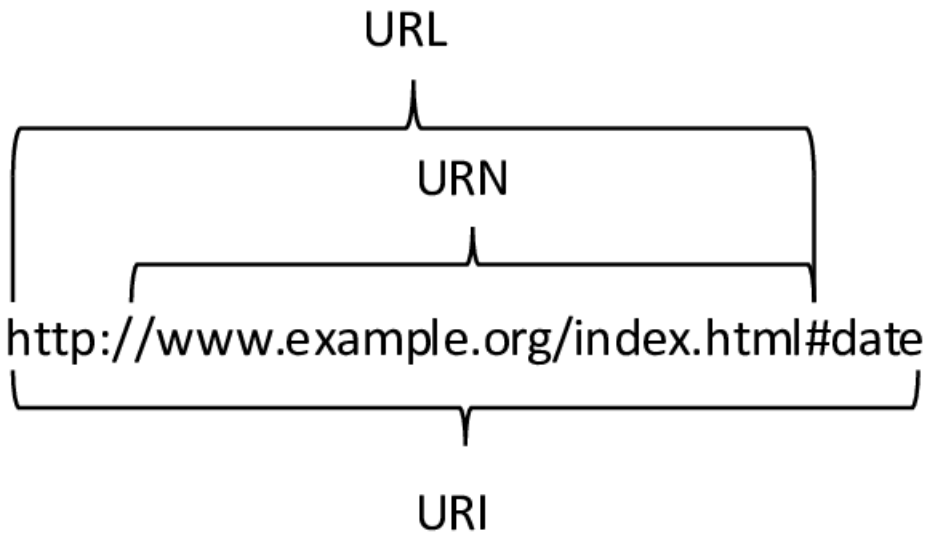
Uniform Resource Identifier

Uniform Resource Locator

Uniform Resource Name

Übliche HTTP Operationen:

- GET – um Daten vom Server abzufragen
- POST/PUT - um Daten am Server abzulegen oder zu verändern
- DELETE – um Daten am Server zu löschen



Quelle: https://www.researchgate.net/figure/The-illustration-of-the-URL-URN-and-URI-26_fig4_346585530

JSON

GET

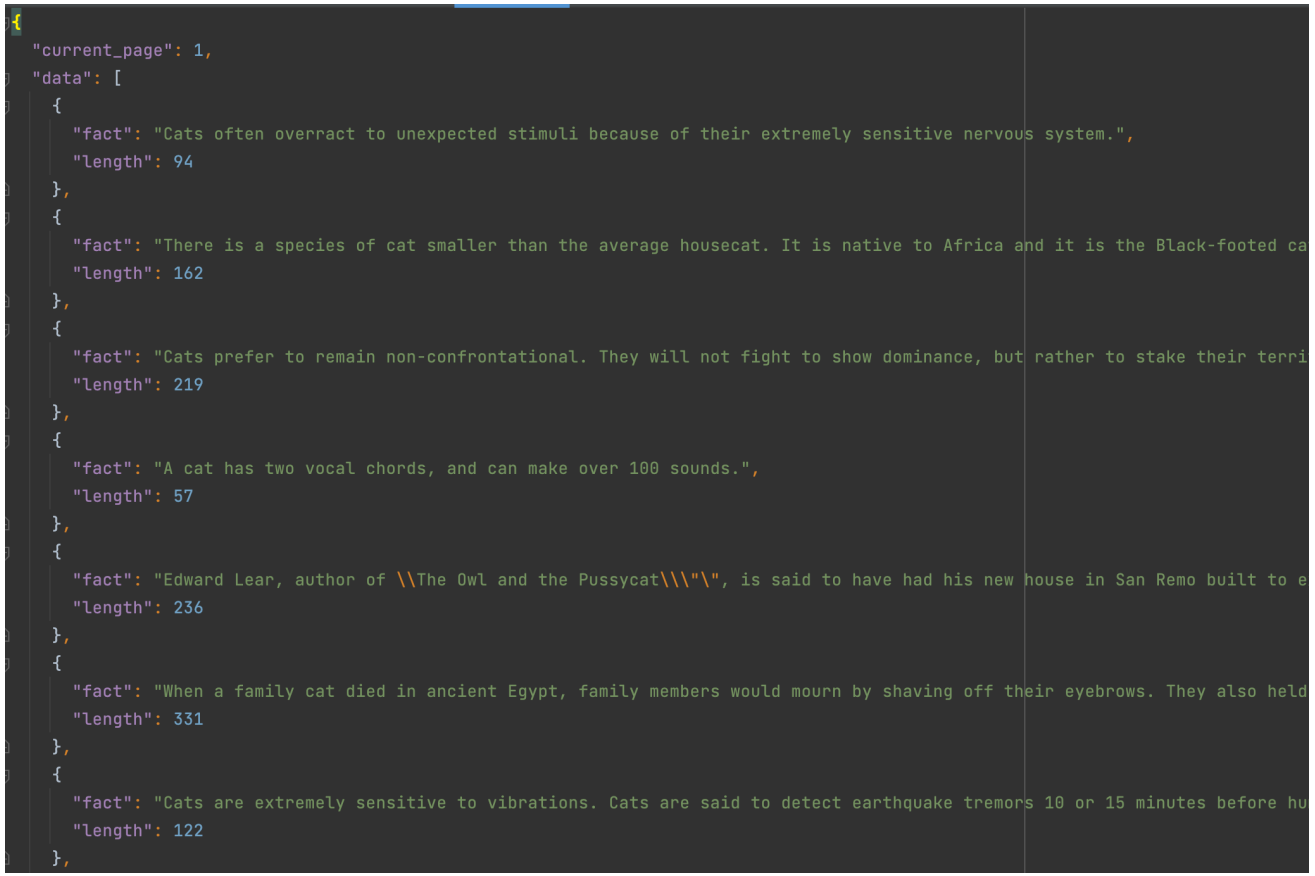
https://catfact.ninja/facts

Stellt eine Liste an Katzenfakten als JSON bereit

JSON

Java**S**cript **O**bject **N**otation

Gängiges Format um Objekte für Datenaustausch aufzubereiten

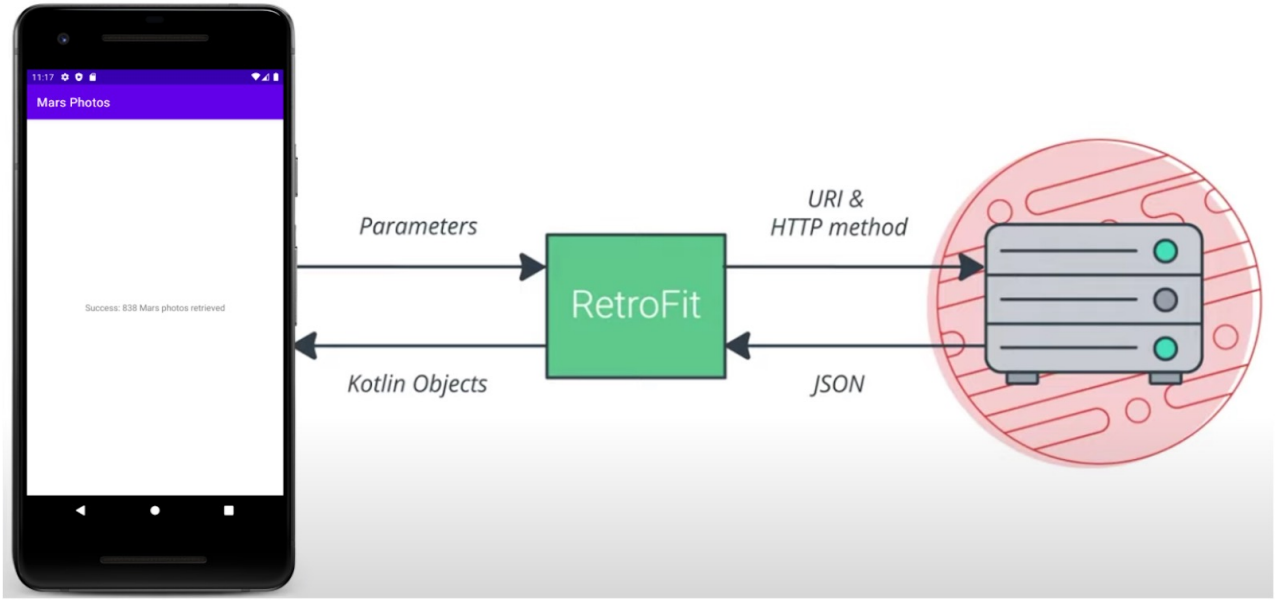


retrofit

https://square.github.io/retrofit/

Ist ein HTTP Client für Android

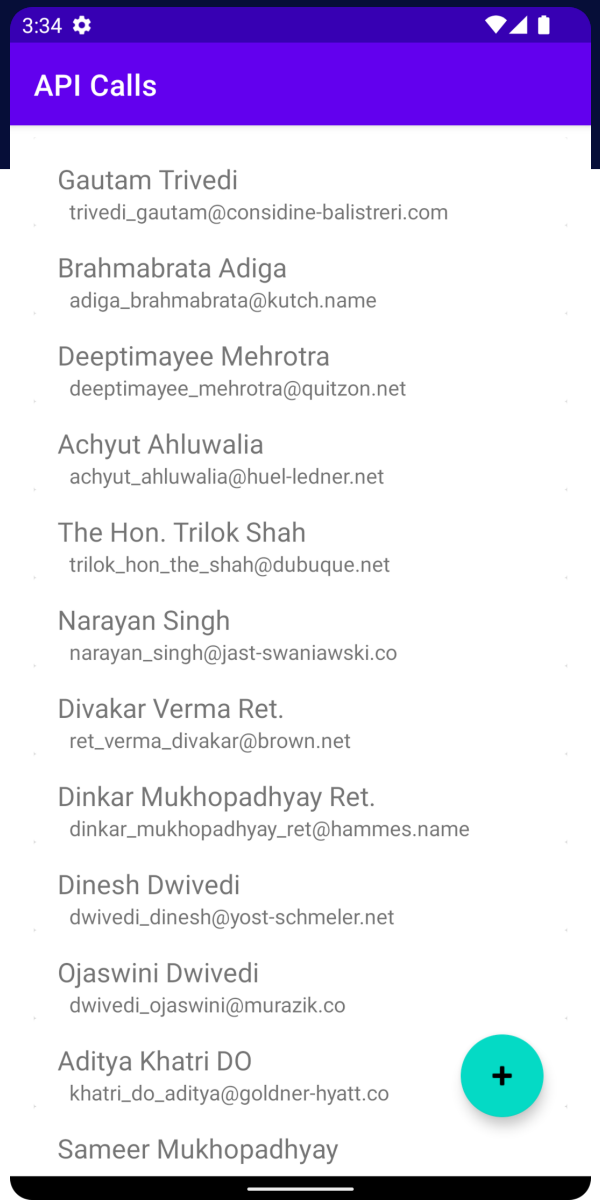
Ein **RetrofitService** übernimmt die
Kommunikation mit dem Server
und übersetzt die Antwort in Kotlin Objekte



Quelle: https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet/img/a8f10b735ad998ac.png

Beispiel: User List

- ein **Repository** kümmert sich um die Organisation der Daten
- ein **Retrofitservice** kommuniziert mit dem Server
- mittels **GET** wird eine Liste an Usern geladen
- mittels **POST** wird ein neuer User angelegt



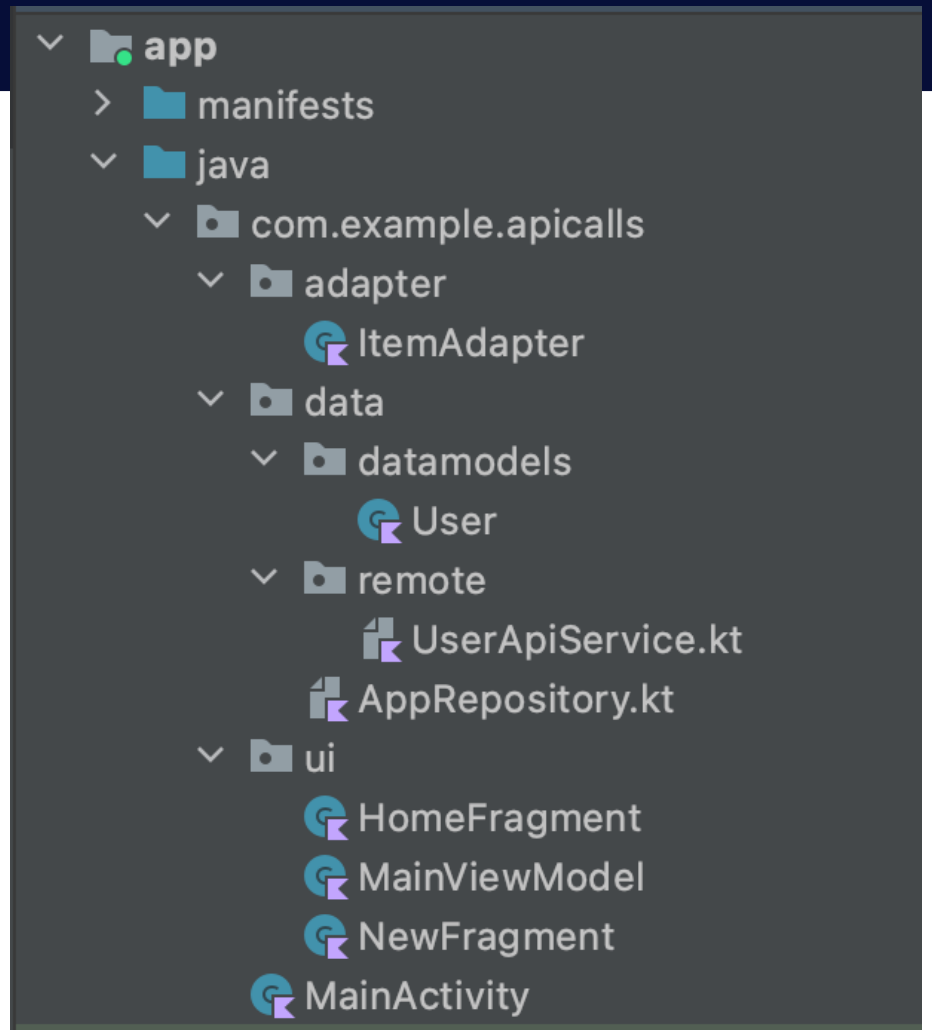
Struktur

gewohnte **MVVM** Architektur

Neu ist das **remote** Package mit

UserApiService.kt

hier wird die gesamte Kommunikation mit dem Server stattfinden und Daten fürs Repository vorbereitet



AndroidManifest.xml

Folgende Zeile erlaubt der App sich mit dem **Internet** zu verbinden

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.apicalls">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="API Calls"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.APICalls">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

build.gradle (app Module)

Im GradleScript werden neben den Dependencies für Navigation, LiveData und ViewModel

auch jene von **retrofit** und **moshi** eingefügt

moshi ist ein Konverter um Serverantworten in KotlinObjekte zu übersetzen

```
dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.4.1'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1'
    implementation 'androidx.fragment:fragment-ktx:1.4.1'


    // Navigation
    implementation "androidx.navigation:navigation-fragment-ktx:2.4.2"
    implementation "androidx.navigation:navigation-ui-ktx:2.4.2"

    //Retrofit
    implementation "com.squareup.retrofit2:retrofit:2.9.0"
    implementation "com.squareup.retrofit2:converter-moshi:2.9.0"
    implementation "com.squareup.moshi:moshi-kotlin:1.13.0"
}
```

MainViewModel

eine Liste an User wird vom **Repository** bereitgestellt

weiters gibt es eine Funktion um neue **Daten** zu **laden** und eine um einen **neuen User hinzuzufügen**

beide Funktionen werden innerhalb **Coroutines** abgerufen um das UI nicht zu blockieren

```
class MainViewModel : ViewModel() {

    private val repository = AppRepository(UserApi)

    val users = repository.users

    fun loadData() {
        viewModelScope.launch { this: CoroutineScope
            repository.getUsers()
        }
    }

    fun addNewUser(user: User) {
        viewModelScope.launch { this: CoroutineScope
            repository.addUser(user)
        }
    }
}
```

AppRepository

es wird versucht (**try**) eine Liste an User über **retrofit** zu laden und als **LiveData** bereitzustellen

weiters gibt es eine **suspend** Funktion welche über **retrofit** versucht einen neuen User anzulegen

```kotlin
const val TAG = "Repository"

class AppRepository(private val api: UserApi) {

    private val _users = MutableLiveData<List<User>>()
    val users: LiveData<List<User>>
        get() = _users

    suspend fun getUsers() {
        try {
            _users.value = api.retrofitService.getUsers()
        } catch (e: Exception) {
            Log.e(TAG, msg: "Error loading Data from API: $e")
        }
    }

    suspend fun addUser(user: User) {
        try {
            val call = api.retrofitService.createUser(user)
        } catch (e: Exception) {
            Log.e(TAG, msg: "Error putting Data on Server: $e")
        }
    }
}
```

UserApiService

manche APIs verlangen einen Zugangsschlüssel (API_TOKEN) um Daten abzufragen oder zu hinterlegen

hierfür wird ein **client** angelegt, welcher bei jedem Request den Schlüssel mitsendet

um Antworten direkt zu übersetzen wird ein **moshi** angelegt

danach wird ein **retrofit** gebaut welcher client, moshi und BASE_URL verwendet

```
const val BASE_URL = "https://gorest.co.in/public/v2/"
const val API_TOKEN = "mysupersecretpaikey"

val client: OkHttpClient = OkHttpClient.Builder().addInterceptor { chain ->
    val newRequest: Request = chain.request().newBuilder()
        .addHeader( name: "Authorization",  value: "Bearer $API_TOKEN")
        .build()
    chain.proceed(newRequest)  ^addInterceptor
}.build()

private val moshi = Moshi.Builder()
    .add(KotlinJsonAdapterFactory())
    .build()

private val retrofit = Retrofit.Builder()
    .client(client)
    .addConverterFactory(MoshiConverterFactory.create(moshi))
    .baseUrl(BASE_URL)
    .build()

interface UserApiService {

    @GET( value: "users")
    suspend fun getUsers(): List<User>

    @POST( value: "users")
    suspend fun createUser(@Body user: User)
}

object UserApi {
    val retrofitService: UserApiService by lazy { retrofit.create(UserApiService::class.java) }
}
```

UserApiService

das Interface **UserApiService** bestimmt wie mit dem Server kommuniziert wird

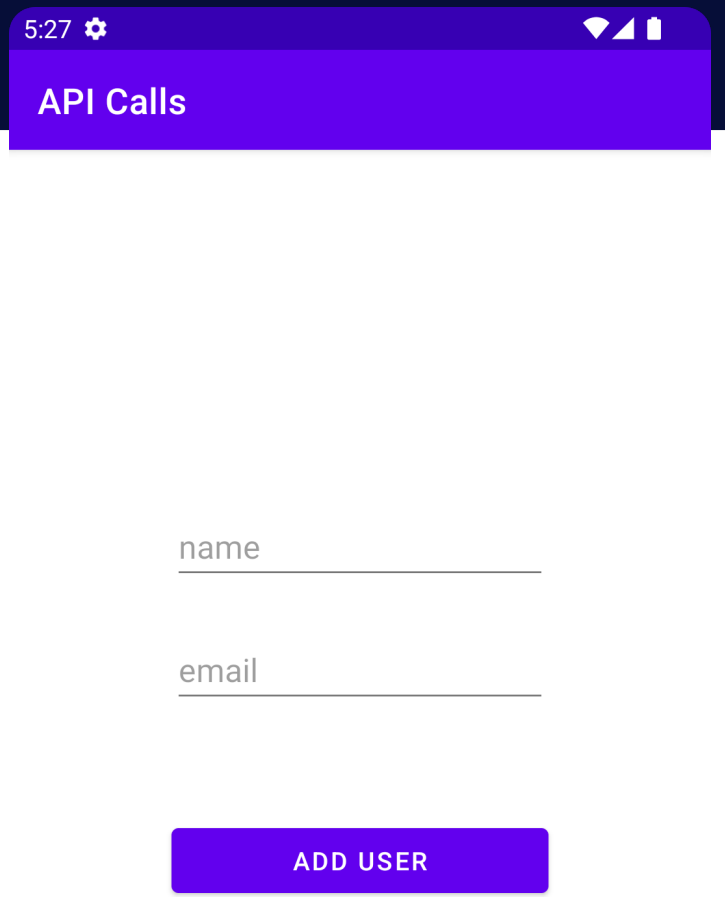
- ein **GET-Request** am Endpunkt users welcher eine Userliste zurückliefert
- ein **POST-Request** am Endpunkt users welcher einen zu speichernden User mitbringt

UserApi dient als Zugangspunkt für den Rest der App und stellt das Interface als **retrofitservice** zur Verfügung

```
const val BASE_URL = "https://gorest.co.in/public/v2/"
const val API_TOKEN = "mysupersecretpaikey"

val client: OkHttpClient = OkHttpClient.Builder().addInterceptor { chain ->
    val newRequest: Request = chain.request().newBuilder()
        .addHeader( name: "Authorization", value: "Bearer $API_TOKEN")
        .build()
    chain.proceed(newRequest)  ^addInterceptor
}.build()

private val moshi = Moshi.Builder()
    .add(KotlinJsonAdapterFactory())
    .build()

private val retrofit = Retrofit.Builder()
    .client(client)
    .addConverterFactory(MoshiConverterFactory.create(moshi))
    .baseUrl(BASE_URL)
    .build()

interface UserApiService {

    @GET( value: "users")
    suspend fun getUsers(): List<User>

    @POST( value: "users")
    suspend fun createUser(@Body user: User)
}

object UserApi {
    val retrofitService: UserApiService by lazy { retrofit.create(UserApiService::class.java) }
}
```

Fertig

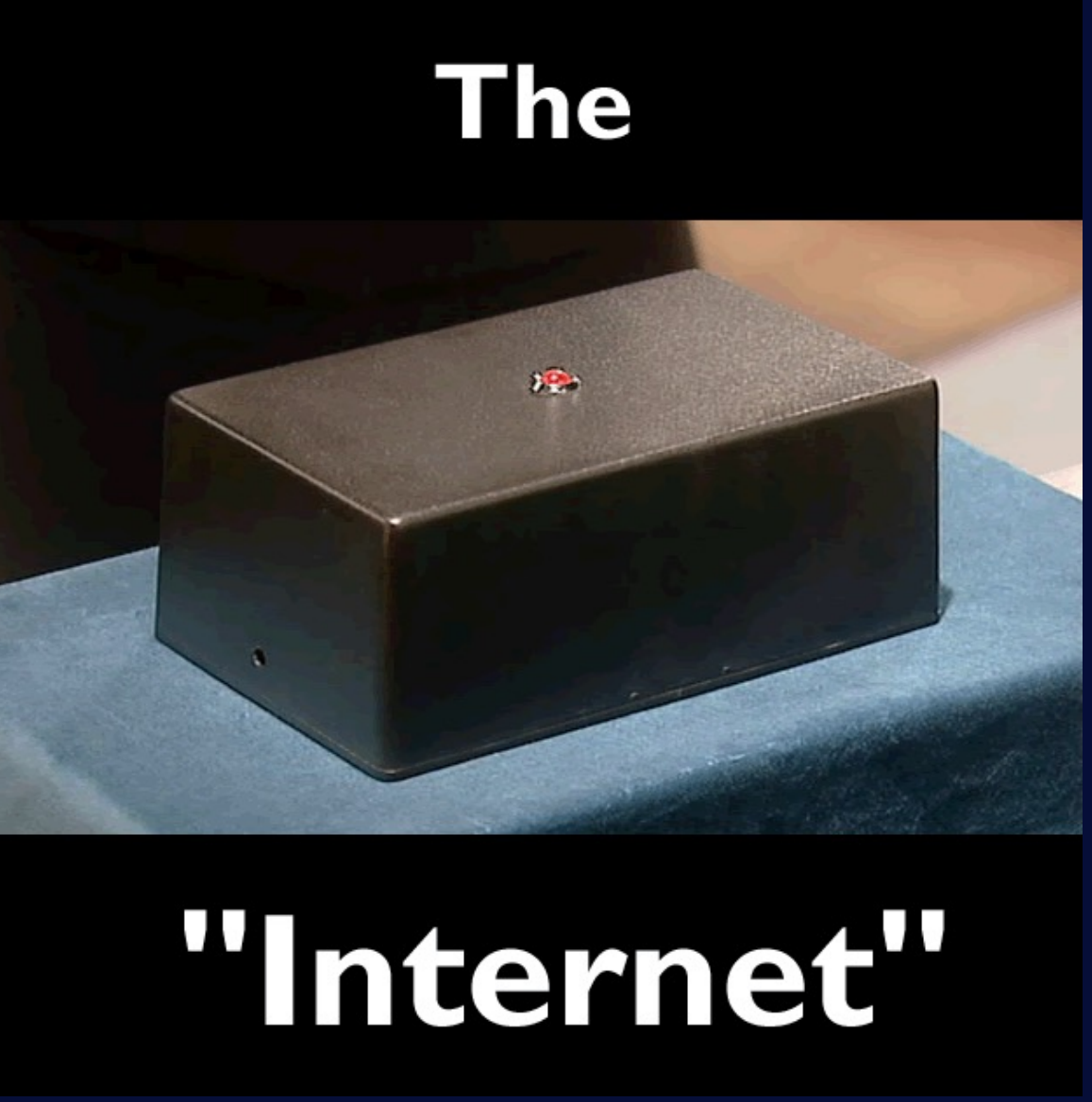
unsere App kann jetzt die aktuelle **Userliste** vom Server **laden** und einzelne **User hinzufügen**



Activities Lifecycle

Wiederholung - Was haben wir heute gelernt?

1	RESTful Service
2	URI, URL und HTTP
3	retrofit



Quelle: https://i.pinimg.com/originals/03/5b/48/035b486b37463ddd99945c891eb7f439.gif

Syntax Institut

Viel Spaß!