

Android Web Services

pe intelesul tuturor

Introducere

Un serviciu web este un standard folosit pentru schimbul de informatii intre aplicatii sau sisteme eterogene.

Multe aplicatii, fie ele desktop sau web, sunt scrise in limbaje diferite de programare. Deseori este imposibil ca doua sisteme sa comunice direct. Solutiile interoperabilitatii sunt serviciile web. Prin intermediul acestora putem face schimb de informatii, de exemplu, intre o aplicatie scrisa in C# si o aplicatie mobila, android, scrisa in Java.

Introducere

O mare parte din aplicatiile mobile prezente in piata, folosesc servicii web. Multe companii au deja produse care sunt furnizate fie sub forma unei aplicatii web sau desktop si vor sa le puna la dispozitia utilizatorilor si sub forma unei aplicatii mobile. Pentru a realiza aceasta se folosesc serviciile web.

Pentru aceasta fac public un API care poate fi folosit de catre aplicatia mobila pentru a avea acces la informatiile care sunt deja prezentate in cadrul celorlalte produse.

Plan de bataie

Pentru a exemplifica mai bine modul de lucru in cazul serviciilor web vom crea o aplicatie simpla care isi propune urmatoarele:

- Inregistrarea unui vot pe baza nr. matricol

- Prezentarea rezultatelor votului

Dilema care-i chinuie pe multi in aceasta perioada este:



Plan de bataie

Asadar vom crea o aplicatie care va oferi posibilitatea sa aflam odata pentru totdeauna cine este mai tare: Batman sau Superman?

Pentru a putea stoca informatiile de la toti utilizatorii centralizat si apoi sa prezentam rezultatul final ne propunem sa cream un API care va pune la dispozitie 2 metode:

POST /vote {matricol, vot}

GET /results

*Un numar matricol poate vota o singura data

Justice API REST

API-ul este creat folosind framework-ul Laravel (PHP)

POST /vote {"matricol": "", "option": "1/2"}

```
$matricol    = $request->input('matricol');
$vote_option = $request->input('option');

// validate input
// ...

// validation OK => insert vote into database
DB::table('votes')->insert([
    'matricol' => $matricol,
    'vote' => $vote_option,
]);

return response()->json([
    'message' => 'Votul dvs. a fost inregistrat cu succes!'
], 200);
```


GET /results

```
$votes = DB::table('votes')
    ->select(DB::raw('count(*) as vote_count, vote'))
    ->groupBy('vote')
    ->pluck('vote_count', 'vote');

return response()->json([
    1 => isset($votes[1]) ? intval($votes[1]) : 0,
    2 => isset($votes[2]) ? intval($votes[2]) : 0,
]);
```

Justice API REST

API-ul va stoca fiecare vot intr-o baza de date MySql, care contine o singura tabela:

| TABLES | | Field | Type | Length |
|---|----------|---------|------|--------|
|  votes | matricol | VARCHAR | ↕ | 255 |
| | vote | INT | ↕ | 11 |

Aplicatia mobila

In continuare vom crea o aplicatie mobila care va “consuma” acest API.

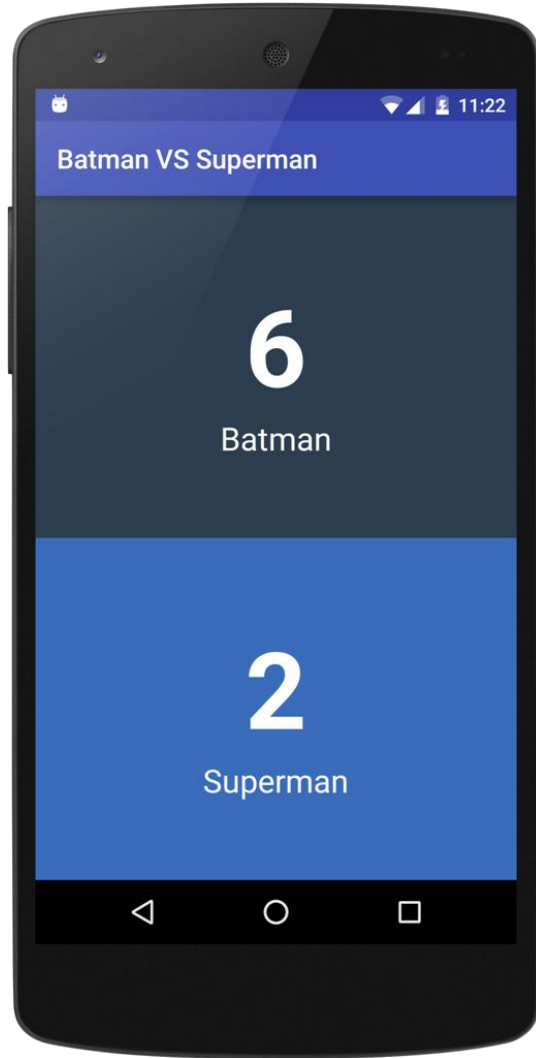
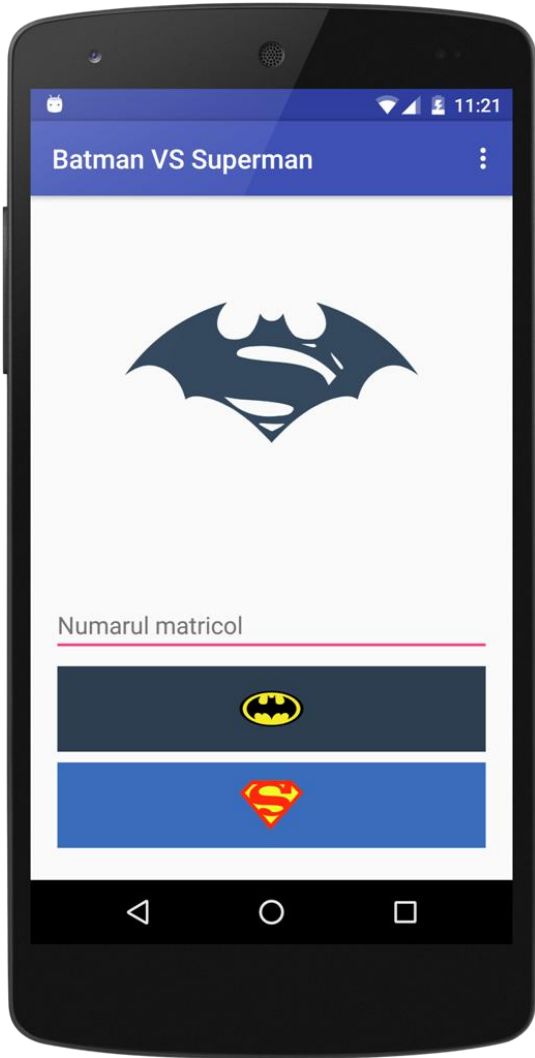
Aplicatia mobila va trebui sa puna la dispozitie:

- un camp de text pentru preluarea numarului matricol

- 2 butoane pentru votare

- Optiune pentru vizualizarea rezultatelor votului

Avand deja cunostintele necesare pentru a realizarea acestei interfete, rezultatul final ar trebui sa fie:



Aplicatia mobila

Pentru a accesa API-ul in cadrul aplicatiei mobile, vom folosi o librerie care ne va usura interactiunea cu acesta. **Retrofit** este cel mai popular client HTTP type-safe pentru Android, foarte usor de folosit si performant.

Pentru a folosi aceasta librerie, trebuie sa o adaugam in cadrul proiectului nostru. Modificam fisierul **build.gradle** si adaugam urmatoarele linii (dupa care facem sync):

```
compile 'com.squareup.retrofit2:retrofit:2.0.0-beta4'  
compile 'com.squareup.retrofit2:converter-gson:2.0.0-beta4'
```

retrofit: este librerie principala

converter-gson: este o librerie care converteste obiectele de tip JSON in obiecte Java si viceversa. Aceasta librerie este folosita de retrofit pentru a converti raspunsurile API-ului in obiecte in cadrul aplicatiei noastre.

Crearea Endpoint-urilor

```
public interface ApiService {  
    @GET("results")  
    Call<HashMap<Integer, Integer>> results();  
  
    @POST("vote")  
    Call<HashMap<String, String>> vote(@Body Map<String, String> params);  
}
```

În continuare vom defini metodele API care vor putea fi accesate. Pentru a face aceasta vom crea o interfata care va defini metodele disponibile.

Folosind adnotari, specificăm tipul requestului: GET / POST iar ca parametru vom specifica URL-ul specific metodei. În cadrul retrofit vom defini un URL de baza, care va fi concatenat pentru fiecare metoda, ex:

“http://batmanvsuperman/api/” + “results”

Crearea Endpoint-urilor

```
public interface ApiService {  
    @GET("results")  
    Call<HashMap<Integer, Integer>> results();  
  
    @POST("vote")  
    Call<HashMap<String, String>> vote(@Body Map<String, String> params);  
}
```

Pentru fiecare metoda va trebui sa specificam ce tip de raspuns vom primi si sub ce forma vom trimite parametrii catre functie.

Spre exemplu, in cazul nostru, metoda **/vote** va trimite parametrii sub forma unui `HashMap<String, String>` si va primi ca raspuns acelasi tip de obiect. (recomandare noastra ar fi sa definim cate un obiect pentru fiecare tip de raspuns)

Configurare Retrofit

Pentru a folosi aceasta librerie propunem urmatorul mod de lucru:

Vom crea o noua clasa: RestClient, in care vom crea obiectul retrofit caruia ii vom spune sa foloseasca metodele din interfata ApiService si GSON pentru interpretarea raspunsurilor

Cand aplicatia porneste vom instantia RestClient, astfel il vom putea folosi foarte usor din orice activitate

```
public class RestClient {
    public static final String BASE_URL = "URL/";
    private ApiService apiService;
    public static Retrofit retrofit;

    public RestClient() {
        retrofit = new Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        apiService = retrofit.create(ApiService.class);
    }

    public ApiService getApiService() {
        return apiService;
    }
}
```

Configurare Retrofit

Pentru a crea o instanță a `RestClient` atunci când porneste aplicația, procedăm astfel:

creăm o nouă clasă `App.java` care extinde `Application`

dorim ca această clasă să ruleze când aplicația porneste, pentru aceasta adăugăm în `AndroidManifest.xml` atributul `android:name`

În metoda `onCreate` vom crea un nou obiect `RestClient`

```
<application
    android:name=".app.App"
    ...
```

```
public class App extends Application {
    public static String TAG = "### B VS S ###";

    private static RestClient restClient;

    public static RestClient getRestClient() {
        return restClient;
    }

    @Override
    public void onCreate() {
        super.onCreate();

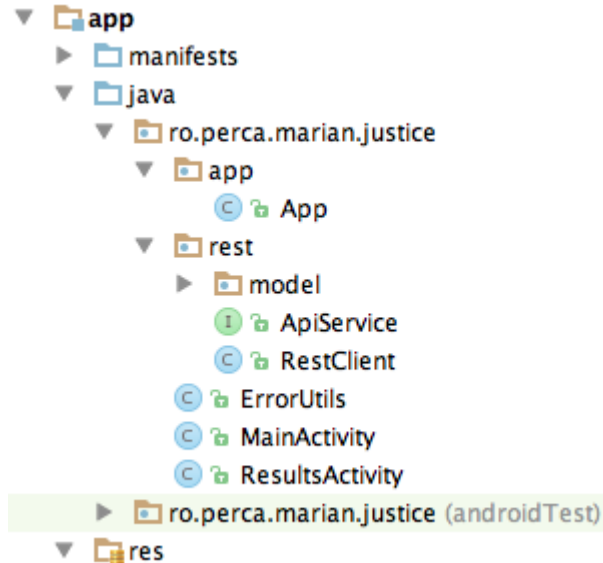
        restClient = new RestClient();
    }
}
```

Mod de organizare

Pentru structurarea aplicatiei propunem structura din imaginea alaturata.

Deoarece pentru a ne conecta la API avem nevoie de o conexiune la internet, trebuie sa declaram aceasta permisiune in AndroidManifest.xml

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



Apelarea API-ului

Mergem in activitatea MainActivity si vom scrie codul pentru inregistrarea unui vot.
Vom prelua numarul matricol si votul, dupa care le vom stoca intr-un obiect HashMap

```
HashMap<String, String> apiParams = new HashMap<>();
apiParams.put("matricol", mMatricol.getText().toString());
apiParams.put("option", String.valueOf(vote));

// call API with selected vote
Call<HashMap<String, String>> voteApiCall = App.getRestClient().getApiService().vote(apiParams);
voteApiCall.enqueue(new Callback<HashMap<String, String>>() {

    @Override
    public void onResponse(Call<HashMap<String, String>> call, Response<HashMap<String, String>> response) {
        // verificam daca raspunsul are codul 2xx sau avem o eroare 4xx - 5xx
    }

    @Override
    public void onFailure(Call<HashMap<String, String>> call, Throwable t) {
        // aceasta metoda este apelata daca API-ul nu poate fi accesat (ex: nu avem conexiune la internet)
    }
});
```

Apelarea API-ului

Pentru a nu bloca thread-ul principal (UI thread) cat timp apleam si asteptam un raspuns de la API, vom inregistra un observer, prin metoda: **enqueue**

Callback-ul care se va executa cand primim raspunsul are 2 metode, care se vor apela in functie de tipul raspunsului: onResponse() si onFailure()

Daca API call-ul reuseste, se va apela onResponse(), unde va trebuie sa verifica ce fel de raspuns avem: pentru succes vom avea codul 2xx iar pentru eroare: 4xx - 5xx

```
@Override
public void onResponse(Call<HashMap<String, String>> call, Response<HashMap<String, String>> response) {
    if (response.isSuccess()) {
        // afisam un mesaj daca raspunsul are codul 2xx
    } else {
        ApiError error = ErrorUtils.parseError(response);
        displayMessage(error.message(), "error"); // afisam mesaj de eroare
    }
}
```

Resurse

Proiect android: <https://github.com/marianperca/ama-justice>

Serviciu web: <https://bitbucket.org/marianperca/justice/>

Retrofit: <http://square.github.io/retrofit/>

Aplicatia poate fi descarcata de pe Google Play:

AMA Justice

<https://play.google.com/store/apps/details?id=ro.perca.marian.justice>