Web-APIs consumeren op mobiele platformen

Practicum Internettechnologie

Academiejaar 2014–2015

1 Cliënten van web-APIs

In het vorige practicum maakte je kennis met REST-APIs. Een centraal principe van REST dat de server slechts één API aanbiedt die door verschillende types cliënten kan gebruikt worden. Dit heb je in het vorige practicum ook gemerkt, aangezien de evenemententoepassing die je bouwde al gebruikt werd door twee types cliënten:

- een **webbrowser** die HTML consumeert, bediend door mensen;
- een **script** dat JSON consumeert, automatisch geactiveerd door de browser.

In dit practicum gaan we een stap verder en ontwikkelen we een **platform-specifieke cliënt** voor Android-toestellen. Opnieuw maken we gebruik van dezelfde API, en meer bepaald de JSON-representaties daarvan.

Dit practicum bevat dus twee uitdagingen. Langs de ene kant is er de technische uitdaging om een toepassing voor Android te ontwikkelen. Sectie 2 beschrijft een voorbeeldtoepassing om je op weg te helpen. Langs de andere kant is er de uitdaging om een cliënt voor een REST-API te schrijven. Hierbij is het belangrijk dat je de REST-principes in het achterhoofd houdt, in het bijzonder dat "de toepassing wordt bestuurd via hypermedia". Dit betekent dat er *geen* hard-gecodeerde URLs in je broncode mogen staan, met uitzondering van één begin-URL. De volledige communicatie tussen jouw cliënt en de server moet gebeuren via hypermedia, dus door het volgen van links en formulieren.

Om er zeker van te zijn dat de API die je gebruikt de REST-principes correct implementeert, gebruiken we voor dit practicum een voorbeeldimplementatie van de evenemententoepassing. Deze toepassing is beschikbaar op *http://events.restdesc.org/*. Zoals je daar kan zien, worden enkel de JSON-representaties aangeboden. De API is echter volledig functioneel en je kan deze bekijken door de URLs die je tegenkomt in de adresbalk van je browser te plakken. De Android-toepassing die je ontwikkelt zal ook gebruik maken van deze URLs. Je kan evenementen, personen en berichten aanmaken, bewerken en verwijderen. Aangezien deze API voor iedereen openstaat, resetten we de inhoud iedere dag. Gewijzigde data wordt dus maximaal 24 uur bewaard. Experimenteer alvast met deze API, bijvoorbeeld via de browser en/of met `curl` of *http://onlinecurl.com/*. Hoewel de API (zoals iedere REST-API) zichzelf zou moeten uitwijzen, geven we hieronder enkele voorbeeldjes met de `curl`-tool.

Listing 1: Voorbeeld-requests voor de evenementen-API

```
# Een representatie van een persoon ophalen
curl -H "Accept: application/json" http://events.restdesc.org/people/1

# De naam van een persoon wijzigen
curl -X PATCH -H "Accept: application/json" -H "Content-Type: application/json" \
     -d '{"person":{"name":"Johan Joris"}}' http://events.restdesc.org/people/1

# Een persoon aanmaken
curl -X POST -H "Accept: application/json" -H "Content-Type: application/json" \
     -d '{"person":{"name":"Mia Maris", "email": "mia@maris.be", "birth_date": "1980-03-24"}}' \
     http://events.restdesc.org/people

# De aanwezigheid van een persoon bevestigen
curl -X POST -H "Accept: application/json" -H "Content-Type: application/json" \
     -d '{"confirmation":{"person":{"url":"http://events.restdesc.org/people/3"}, "going": true}}' \
     http://events.restdesc.org/events/1/confirmations

# Een persoon verwijderen
curl -X DELETE -H "Accept: application/json" http://events.restdesc.org/people/3
```

2 Android-toepassingen ontwikkelen

Voor we van start gaan met de ontwikkeling van een mobiele web-API, leggen we uit welke programma's nodig zijn om Android-apps te kunnen aanmaken en hoe je deze kan installeren. Alle tools zijn ter beschikking op Athena. Indien je liever de ontwikkelomgeving op je eigen systeem installeert, raden we je aan om een kijkje te nemen op de ontwikkelaarswebsite van Android[1]. Daarna gaan we over tot het eigenlijke programmeren van een mobiele webclient.

We kiezen voor Android omdat het een open-sourceplatform is (in tegenstelling tot bijvoorbeeld Apple's iOS). Het is een besturingssysteem voor mobiele telefoons, tablets, koelkasten, camera's en meer, gebaseerd op de Linux-kernel. Bovendien maakt het gebruik van het Java-programmeerplatform, hoewel er bij Android geen Java Virtuele Machine is. In plaats daarvan worden Java-klassen gecompileerd naar uitvoerbare bestanden en uitgevoerd op Dalvik[2]. Sinds Android 4.4 is het mogelijk om gebruik te maken van de nieuwere runtime ART[3].

2.1 Kennismaken met de ontwikkelomgeving

Om de ontwikkelomgeving te leren kennen, maken we een eenvoudige toepassing. We maken eerst een emulator aan, die een Android-toestel nabootst. Daarna hebben we het over de verschillende tools die aanwezig zijn om een standaard Android-toepassing aan te passen. Ten slotte starten we het resultaat in de emulator.

2.1.1 Nieuw project aanmaken

1. Start de Android Developer Tools op Athena.
2. Klik op de knop **New Android Application**.
3. Controleer de volgende instellingen:
   - *Minimum Required SDK*: API16 – Android 4.4 (KitKat)
   - *Target SDK*: API 16 – Android 4.4 (KitKat)
   - *Compile with*: Android 4.4 (KitKat)
4. Klik **Next**.
5. Vink **Create a custom launcher icon** af en verifieer dat **Mark this project as a library** afgevinkt staat.
6. Klik **Next**.
7. Laat alle instellingen staan en klik vervolgens op **Finish** om het project aan te maken.

2.1.2 Emulator aanmaken

Met een emulator kan je toepassingen testen. Je kan meerdere emulators aanmaken, waardoor je makkelijk kan testen op de verschillende Android-versies. Voor meer gedetailleerde instructies kan je terecht op de ontwikkelaarspagina van Android[4].

1. Ga naar **Window**, **Android Virtual Device Manager**.
2. Klik rechts op **New...** en neem de volgende instellingen over:
   - *Device*: 4.65"720p (720 x 1280: xhdpi)
   - *Target*: Android 4.4 - API Level 19
   - *CPU/ABI*: ARM
   - *Memory Options*: RAM: 768; VM Heap: 64
   - *Internal Storage*: 200 MiB
   - *Use Host GPU*: ✓
3. *Klik* op **OK** om de emulator aan te maken en *sluit* de Android Virtual Device Manager.

[1] *http://developer.android.com/sdk/index.html*
[2] *http://source.android.com/devices/tech/dalvik/*
[3] *http://source.android.com/devices/tech/dalvik/art.html*
[4] *http://developer.android.com/training/basics/firstapp/creating-project.html*

2.1.3 Toepassing opstarten

Nu we een Androidproject en een emulator hebben aangemaakt, kunnen we de toepassing opstarten. Je hoeft hiervoor niet apart de emulator op te starten als je hem al gesloten had: deze wordt automatisch opgestart. Het duurt typisch 2 à 3 minuten om de emulator op te starten, en daarna kan je hem laten openstaan.

1. Open de Java broncode-**MainActivity.java**
2. Klik op **Run** (groene "play"-icoon) en vervolgens **Android Application** en **OK**.
3. Verifieer dat je "Hello World" te zien krijgt.
4. Pas in **activity_main.xml** de tekst "Hello World" aan en bekijk het resultaat in de emulator.

2.2 Ontwerpen

2.2.1 Activity

In Android is een *activity* een specifieke taak die een gebruiker kan doen. Dit kan bijvoorbeeld zijn: een lijst van elementen weergeven, of een individueel element bewerken. De visuele structuur van een activity wordt bepaald door een *layout*, die verschillende *view*-elementen combineert. Een gedetailleerde uitleg over hoe activities in elkaar zitten, vind je op de developersite van Android[5].

2.2.2 View

Elk scherm in Android bestaat uit zogenaamde *views*. Een view is een rechthoekig gedeelte op het scherm met een bepaalde visualisatie, waardoor de gebruiker een view herkent als een knop, een stuk tekst, enzoverder. Een *ViewGroup* is een view die andere view kan bevatten. Wanneer je in een view andere views stopt, kan je aangeven hoe de views zich ten opzichte van elkaar gedragen (gelijkaardig aan Java Swing voor desktop-interfaces).

2.2.3 Layout

In Android is een layout een XML-bestand dat beschrijft welke views er ingesteld zijn en hoe views weergegeven worden. Door de eigenschappen van een view aan te passen, kan je wijzigen hoe de views eruit zien (zoals je in het vorige gedeelte de tekst "Hello world" hebt aangepast). Alle layouts die je voor je toepassing maakt, worden opgeslagen in de hoofdmap **res** ("resources"). Standaard-layouts worden opgeslagen in de map **res/layout**. Bij de creatie van de toepassing hadden we aangevinkt om een *activity* aan te maken. Deze activity bevatte reeds een layout, die we nu kunnen aanpassen:

1. Ga in de **Package Explorer Window** naar de map **res/layout** en vind het bestand **activity_main.xml**.
2. Verwijder de tekst **Hello World** uit het scherm.
3. Voeg een **Text Fields > Plain Text** view toe.
4. Voeg een **Form Widgets > Button** view toe.
5. Onthoud de **ID** van beide widgets, die je kan zien en aanpassen in de **properties** rechts.

2.3 Programmeren

De basis van de code voor de toepassing komt terecht in een activity, in het **Code Editor Window** van Eclipse. Wanneer een gebruiker op een van de knoppen drukt, roep dit de code aan die bij de knop hoort. Voor eenvoudig toepassingen komen alle regels code in de activity. Toen we het Android-project aangemaakt hebben, gaven we aan dat er al een activity aangemaakt mocht worden (vinkje bij **Create Activity**). Dit is `MainActivity`.

2.3.1 Views koppelen en interactief maken

Ga naar Eclipse en open **MainActivity.java**. Je ziet dat de klasse **MainActivity** twee methodes heeft: **onCreate** en **onCreateOptionsMenu**. Deze methodes worden uitgevoerd wanneer deze activity opgestart wordt, respectievelijk het optiemenu aangemaakt wordt. We willen hier nu het tekstvak en de knop gebruiken die we reeds in de layout plaatsten.

[5] *http://developer.android.com/guide/components/activities.html*

Listing 2: View-definitie voor gebruik in de activiteit

```
private EditText editText;
private Button button;
```

Bovenstaande regels kan je toevoegen in de MainActivity-klasse. Ontbrekende klassen mag je importeren. In de **onCreate** methode linken we na de **setContentView**-instructie de view-elementen met de activity.

Listing 3: Een view aan een activiteit koppelen

```
editText = (EditText)findViewById(R.id.editText1); // zelfde ID als in de layout
button = (Button)findViewById(R.id.button1);       // zelfde ID als in de layout
```

2.3.2 Een actie toevoegen aan een button

Open **activity_main.xml** in de **Graphical Layout Editor Tab**, selecteer in het overzicht de knop die je maakte, ga vervolgens naar het **Properties**-venster en zet **On click** op `handleMyButton`. Een alternatieve methode is om in de XML-weergave van **activity_main.xml** het attribuut `android:onClick="handleMyButton"` toe te voegen aan de `<Button>`-tag. Bewaar je aanpassingen. Nu moeten we deze methode nog toevoegen in de klasse **MainActivity**. Maak na de methode `onCreate` en de nieuwe methode `handleMyButton` aan:

Listing 4: Opvragen van tekstinput uit een inputveld.

```
public void handleMyButton(View view) {
    String userText = editText.getText().toString();
    Toast.makeText(this, "Je voerde in: " + userText, Toast.LENGTH_LONG).show();
}
```

We gebruiken **Toast**, een zwart zwevend balkje onderin het scherm dat een tekstbericht toont en na een aangegeven tijdstip (`Toast.LENGTH_LONG`) verdwijnt.

2.3.3 Naar een andere activiteit overschakelen

Om naar een andere activiteit over te schakelen (bijvoorbeeld van een overzichtslijst naar een detailpagina) gebruik je een **Intent**[6]. Hieraan kan je parameters hangen via de `putExtra`-methode. Hieronder geven we de ingevoerde tekst mee naar de volgende activiteit.

Listing 5: Overschakelen naar OtherActivity met parameters.

```
Intent startIntent = new Intent(MainActivity.this, DetailedPersonActivity.class);
startIntent.putExtra("usertext", editText.getText().toString());
startActivity(startIntent);
```

In deze andere activiteit kan je dan de parameters van de intent ophalen, bijvoorbeeld met `getStringExtra`.

Listing 6: Parameters ophalen uit de intent waarmee de activiteit gestart werd.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent startIntent = getIntent();
    String userText = getStringExtra("usertext");
    Toast.makeText(this, "Je voerde in: " + userText, Toast.LENGTH_LONG).show();
}
```

2.4 Achtergrondactiviteiten voor web-APIs

Er zijn twee manieren om activiteiten in de achtergrond uit te voeren: `AsyncTask`[7] en `Intent`. Een `AsyncTask` wordt gebonden aan een activiteit. Een `Intent` start een achtergrondproces op, en de oorspronkelijke activiteit

[6] *http://developer.android.com/training/run-background-service/index.html*
[7] *http://developer.android.com/reference/android/os/AsyncTask.html*

wordt op de hoogte gebracht wanneer resultaten ter beschikking zijn. Als een gebruiker van activiteit wisselt, zal de `AsyncTask` worden stopgezet, terwijl dit voor een `Intent` niet noodzakelijk het geval is. Voor dit practicum zal `AsyncTask` echter volstaan.

Om een achtergrondactiviteit op te zetten die gebruik maakt van een web-API, moet onze toepassing toelating vragen aan het besturingssysteem. Dit vragen we aan in het bestand **AndroidManifest.xml**. Ga daarvoor naar het bestand en open het in de **Permission Editor**-weergave. Klik op de knop **Add...**, selecteer **Uses Permission** en klik op **OK**. Er nieuwe "Uses Permission" werd toegevoegd. Stel de naam in op `android.permission.INTERNET`.

Om een **AsyncTask** te bouwen, maken we in de **MainActivity**-code een nieuwe private klasse aan.

Listing 7: Lege klasse voor een asynchrone taak

```
private class ListEventsTask extends AsyncTask<Void, Void, Void> {
    @Override
    protected Void doInBackground(Void... args) { return null; }

    @Override
    protected void onPostExecute(Void arg) { }
}
```

We roepen de `execute`-methode uit deze klasse aan vanuit `handleMyButton` na het inlezen van de tekst.

Listing 8: Start de asynchrone taak na het inlezen van de tekst

```
new ListEventsTask().execute();
```

Daarnaast definiëren we ook een methode om een resource op te vragen en te parsen (met de JSON-Java-API[8]).

Listing 9: AsyncTask beschrijving

```
private class ListEventsTask extends AsyncTask<Void, Void, Void> {
    private String eventsText = "";

    @Override
    protected Void doInBackground(Void... args) {
        DefaultHttpClient client = new DefaultHttpClient();
        HttpGet request = new HttpGet("http://events.restdesc.org/events");
        request.setHeader("Accept", "application/json");

        try {
            HttpResponse response = client.execute(request);
            String responseJson = EntityUtils.toString(response.getEntity());
            JSONObject responseObject = new JSONObject(responseJson);

            JSONArray events = (JSONArray)responseObject.get("events");
            for (int i = 0; i < events.length(); i++) {
                JSONObject event = (JSONObject)events.get(i);
                String eventTitle = event.getString("title");
                eventsText += eventTitle + "\n";
            }
        }
        catch (Exception e) {
            eventsText = "De evenementen konden niet opgehaald worden.";
            request.abort();
        }
        return null;
    }

    @Override
    protected void onPostExecute(Void arg) {
        Toast.makeText(MainActivity.this, eventsText, Toast.LENGTH_LONG).show();
    }
}
```

Pas de layout aan om het resultaat beter weer te geven, bijvoorbeeld met **MultiLine Text** of **ListView**.

[8] *http://www.json.org/javadoc/org/json/JSONObject.html*

3 Opgave

Voor dit practicum dien je opgaven 1 en 2 hieronder elk in als volgt.

- Maak een **zip-bestand** aan met twee elementen:
  1. een map `project` met je *volledige* project (broncode *en* gecompileerde bestanden);
  2. een bestand `reflectie.pdf` met je reflectie.
- Geef dit zip-bestand de naam `itech-p2-g<groep>.zip`.

Hou ook steeds in het achterhoofd dat scores doorgaans omgekeerd evenredig zijn met de benodigde verbetertijd. Besteed dus voldoende aandacht aan duidelijke code en commentaar, en respecteer de bovenstaande afspraken. Je oplossingen zullen getest worden op Athena; zorg er dus zeker voor dat alles daar werkt.

3.1 Implementatie

De bedoeling is om een Android-toepassing te ontwikkelen die toelaat om de API *http://events.restdesc.org/* te bedienen. De functionaliteit van deze toepassing is identiek aan te toepassing uit het vorige practicum, met het verschil dat het hier om een platform-specifieke toepassing gaat en dat de onderliggende functionaliteit wordt aangeboden door een externe API. Concreet verwachten we dat een gebruiker deze handelingen kan uitvoeren:

- **personen** weergeven, aanmaken, bewerken en verwijderen met naam, e-mailadres en geboortedatum;
- **evenementen** weergeven, aanmaken, bewerken en verwijderen met naam, beschrijving en tijdstippen;
- **boodschappen** weergeven, aanmaken, bewerken en verwijderen met tekst, persoon en evenement;
- **aanwezigheden** op een evenement weergeven en wijzigen.

Hierbij hou je rekening met volgende niet-functionele vereisten:

- **hypermedia** drijft de applicatie: slechts één URL mag hardgecodeerd zijn;
- de interface is **responsief** en mag niet blokkeren – gebruik dus `AsyncTask`;
- zorg voor een zinvolle **klassenhiërarchie** die functioneel verdeeld is over verschillende packages.

De nadruk in deze opgave ligt *niet* op de gebruikersinterface maar des te meer op de communicatie met de API. Daarnaast mag je veronderstellen dat de toepassing constant een internetverbinding ter beschikking heeft.

Zorg voor duidelijke code die commentaar bevat waar gepast. Hanteer een heldere programmeerstijl en gebruik de *best practices* die je leerde voor programmeren in Java en objectgeörienteerde talen in het algemeen. Vermijd bijvoorbeeld dubbele code door aparte (ouder-)klassen te gebruiken.

3.2 Reflectie

Naast de implementatie van de Android-toepassing, bespreek je ook kritisch de voor- en nadelen van HTML-toepassingen versus native toepassingen. In het vorige practicum heb je reeds een webapplicatie geïmplementeerd in HTML, en in het vak *Multimedia* zag je hoe je via CSS ervoor kan zorgen dat HTML-pagina's correct weergeven op kleinere schermen. Een alternatief zou dus zijn om de mobiele versie te implementeren door aanpassingen aan de HTML en CSS van je oplossing voor Practicum 1. Wat zijn de redenen om al dan niet te kiezen voor platformspecifieke toepassingen, zoals deze die je voor Android implementeerde? Nuanceer de argumenten en beschrijf je mening in een reflectie van een half of een volledig A4-blad.