

# Consumptie van web-APIs op mobiele platformen

Practicum Internettechnologie

Academiejahr 2013–2014

## 1 Cliënten van web-APIs

In the 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 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. Begin alvast met deze API te bekijken, bijvoorbeeld via de browser en/of met `curl` of <http://onlinecurl.com/>.

## 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 toch liever op je eigen systeem installeert, raden we je aan om een kijkje te nemen op de ontwikkelaarswebsite van Android<sup>1</sup>. Daarna gaan we over tot het eigenlijke programmeren van een mobiele web-client. We kiezen voor Android omdat het een open-sourceplatform is (in tegenstelling tot bijvoorbeeld Apple's iOS). Het is een *native* besturingssysteem voor mobiele telefoons, tablets, koelkasten, camera's en meer, gebaseerd op de Linux-kernel. Bovendien maakt het gebruik van het Java-programmeerplatform. Toch is er bij Android helemaal geen sprake van de Java Virtuele Machine – Java-bytecode wordt dus niet uitgevoerd. In plaats daarvan worden Java-klassen gecompileerd naar uitvoerbare bestanden en uitgevoerd op Dalvik<sup>2</sup>. Sinds Android 4.4 is het mogelijk om gebruik te maken van de nieuwere runtime ART<sup>3</sup>. We veronderstellen dat je bekend bent met de Eclipse-omgeving. Indien je een heropfrissing nodig hebt, kan je best een tutorial volgen<sup>4</sup>.

<sup>1</sup> <http://developer.android.com/sdk/index.html>

<sup>2</sup> <http://source.android.com/devices/tech/dalvik/>

<sup>3</sup> <http://source.android.com/devices/tech/dalvik/art.html>


<sup>4</sup> Bijvoorbeeld: [http://portal.ou.nl/documents/informatica/snapshots/T42241\\_54.pdf](http://portal.ou.nl/documents/informatica/snapshots/T42241_54.pdf)

## 2.1 Kennismaken met de ontwikkelomgeving

Om de ontwikkelomgeving te leren kennen, maken we een eenvoudige toepassing. We beginnen met het aanmaken van een emulator, die een Android-toestel nabootst. Daarna hebben we het over de verschillende tools die aanwezig zijn om een standaard Android-toepassing aan te passen. Op het einde starten we het resultaat dan in de emulator.

### 2.1.1 Aanmaken Android-project en -emulator

Om toepassingen te testen heb je een emulator (of een fysieke Android-toestel) nodig. In het begin is het best om de emulator te gebruiken. Je kan meerdere emulators aanmaken, en bij elke emulator kan je de versie van het Android-besturingssysteem kiezen. Hierdoor kan je makkelijk toepassingen testen op de verschillende Android-versies. We gaan nu een emulator maken met de Android-versie “4.1 Jelly Bean”, momenteel de meestgebruikte versie<sup>5</sup>. Voor meer gedetailleerde instructies kan je terecht op de ontwikkelaarspagina van Android<sup>6</sup>.

1. Open de Eclipse-versie met de Android Development Toolkit (ADT). Deze is reeds aanwezig op Athena (of gebruik de versie die je zelf hebt geïnstalleerd).
2. Klik rechts onder “Welcome!” op de knop **New Android Application**.
3. Kies een naam voor het project (bv. “HelloWorld”) en stel deze in. Controleer de volgende instellingen:
  - *Minimum Required SDK*: API16 - Android 4.1 (Jelly Bean)
  - *Target SDK*: API 16 - Android 4.1 (Jelly Bean)
  - *Compile with*: Android 4.4 (KitKat)
  - *Theme*: Holo Light with Dark Action Bar
4. Klik **Next**.
5. Vink **Create a custom launcher icon** af, controleer het pad van je workspace, en verifieer dat **Mark this project as a library** afgevinkt staat.
6. Klik **Next**. Laat alle instellingen staan en klik vervolgens op **Finish**. Je project is nu aangemaakt.
7. Controleer dat je in de weergave “Java(default)” zit en klik op het icoontje  in de werkbalk.
8. Klik rechts op **New...** en neem de volgende instellingen over:
  - *AVD Name*: My\_Emulator
  - *Device*: 4.65"720p (720 x 1280: xhdpi)
  - *Target*: Android 4.4 - API Level 19
  - *CPU/ABI*: ARM
  - *Hardware keyboard present*: ✓
  - *Display a skin with hardware controls*: ✓
  - *Front Camera*: Emulated
  - *Back Camera*: Emulated
  - *Memory Options*: RAM: 768; VM Heap: 64
  - *Internal Storage*: 200 MiB
  - *SD Card*: (leeg)
  - *Snapshot*: (leeg)
  - *Use Host GPU*: ✓
9. Klik op **OK** om de emulator aan te maken en *sluit* de AVD Manager.

### 2.1.2 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: hij wordt automatisch opgestart. Het duurt typisch 2 à 3 minuten om de emulator op te starten.

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.

<sup>5</sup><http://developer.android.com/about/dashboards/>

<sup>6</sup><http://developer.android.com/training/basics/firstapp/creating-project.html>

## 2.2 Ontwerpen

Ontwerpen in Android gebeurt met behulp van layouts. Layouts worden bijgehouden in een bestand waarin je achter en/of onder elkaar opschrijft welke views er in de layout zitten. Vervolgens wordt de layout geladen op het scherm zodat de gebruiker een interface van een toepassing voor zich ziet. Voor je een layout kan maken moet je weten wat een view is. Indien je na het lezen van deze uitleg nog meer wil weten over layouts en views raden we je aan om te vertrekken van de Android Developer Tutorial<sup>7</sup>.

### 2.2.1 Views

Elk scherm in Android bestaat uit zogenaamde “views”. Een view is een rechthoekig gedeelte op het scherm waarin een bepaalde afbeelding is geladen. Doordat er een bepaalde afbeelding in een view is geladen, herkent de gebruiker een view als een knop, een stuk tekst. ... Views kunnen zelf weer views bevatten, een zogenaamde “ViewGroup”. Wanneer je in een views andere views stopt, kan er meteen ook aangeven hoe de views zich ten opzichte van elkaar moeten gedragen.

### 2.2.2 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). Omdat we de ADT hebben geïnstalleerd, kunnen layouts aangepast worden met de *Layout Editor* in Eclipse. 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 terugvinden:

- Ga in de **Package Explorer Window** naar de map **res/layout** en vind het bestand **activity\_main.xml**.
- Verwijder de tekst **Hello World** uit het scherm.
- Voeg een **Text Fields > Plain Text** view toe.
- Voeg een **Form Widgets > Button** view toe.
- Onthoud de **ID** van deze beide widgets, die je kan terugvinden in de **properties** rechts.

## 2.3 Programmeren

Een **activity** is een stukje Androidcode die activiteiten uitvoeren voor de gebruikers. Wanneer gebruikers op een van de knoppen drukt, roep je eigenlijk code aan die bij de knop hoort. Een gedetailleerde uitleg over hoe activiteiten in elkaar zitten vind je op de developersite van Android<sup>8</sup>.

De basis van de code voor de toepassing komt terecht in een activity, in het “Code Editor Window” van Eclipse. Voor eenvoudig toepassingen komen alle regels code in de activity. Toen we het Androidproject aangemaakt hebben, gaven we aan dat er al een activity aangemaakt mocht worden (vinkje bij “Create Activity”). Dit is de standaard “MainActivity”.

### 2.3.1 Views koppelen en interactief maken

Ga terug naar Eclipse en open **MainActivity.java**. Je ziet dat de klasse **MainActivity** welgeteld twee methodes heeft: **onCreate** en **onCreateOptionsMenu**. Deze methodes worden uitgevoerd wanneer deze activity wordt opgestart, respectievelijk het optie menu gecreëerd wordt. We willen hier nu de EditText en de Button gebruiken die we reeds in de layout plaatsten.

#### Listing 1: View-definitie voor gebruik in de activiteit

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

Bovenstaande regels kan je toevoegen in de MainActivity-klasse. Ontbrekende klassen mogen geïmporteerd worden. In de **onCreate** methode linken we onder de **setContent** instructie de layout met de activity.

<sup>7</sup> <http://developer.android.com/training/basics/firstapp/building-ui.html>

<sup>8</sup> <http://developer.android.com/guide/components/activities.html>

#### Listing 2: Een view aan een activiteit koppelen

```
editText = (EditText) findViewById(R.id.editText1); //zelfde ID als in de layout  
button = (Button) findViewById(R.id.btnCalculate); //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**, en klik in het overzicht op de **Button** en ga vervolgens naar het **Properties**-venster en zet **On click** op `doList`. Bewaar de aanpassingen. Nu moeten we de methode nog toevoegen in de klasse **MainActivity**. Maak na de methode `onCreate` en de nieuwe methode `doList` aan, in overeenstemming met de layout. Dit is een void-methode. Voeg onderstaande instructies toe in de methode:

#### Listing 3: Opvragen van tekstinput uit een inputveld.

```
String text = editText.getText().toString();  
Toast.makeText(this, "Je voerde in: " + text, Toast.LENGTH_LONG).show();
```

We gebruiken **Toast**, een zwart zwevend balkje onderin het scherm dat een tekstbericht toont en na een tijdje verdwijnt. De duur wordt aangegeven door **Toast.LENGTH\_LONG**.

## 2.4 Achtergrondactiviteiten

Er zijn twee manieren om activiteiten in de achtergrond uit te voeren: `AsyncTask`<sup>9</sup> en `Intent`<sup>10</sup>. Een `AsyncTask` wordt gebonden aan een activiteit. Een `Intent` start een achtergrondproces op, en de oorspronkelijke activiteit wordt op de hoogte gebracht wanneer er 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 (afhankelijk van de configuratie).

Om achtergrondactiviteit op te zetten die gebruik maakt van een web-API, moeten 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 "Permission" werd toegevoegd. Selecteer nu **android.permission.INTERNET**.

### 2.4.1 AsyncTask

Om een taak te bouwen, maken we in de **MainActivity.java** een nieuwe private klasse aan.

#### Listing 4: 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 de `doList`-methode na het inlezen van de tekst.

#### Listing 5: Start de asynchrone taak na het inlezen van de tekst

```
String text = editText1.getText().toString();  
new ListEventsTask().execute();
```

<sup>9</sup> <http://developer.android.com/reference/android/os/AsyncTask.html>

<sup>10</sup> <http://developer.android.com/training/run-background-service/index.html>

Daarnaast definiëren we ook een methode om de call REST-API uit te voeren. Om JSON te interpreteren, gebruiken we de JSON-Java-API<sup>11</sup>.

#### Listing 6: Asynchrone Task 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);
            InputStream bodyStream = response.getEntity().getContent();
            BufferedReader bodyReader = new BufferedReader(new InputStreamReader(bodyStream));
            StringBuilder body = new StringBuilder();
            String chunk;
            while ((chunk = bodyReader.readLine()) != null)
                body.append(chunk);
            JSONObject bodyObject = new JSONObject(body.toString());

            JSONArray events = (JSONArray)bodyObject.get("events");
            for (int i = 0; i < events.length(); i++) {
                JSONObject event = (JSONObject)events.get(i);
                String eventTitle = event.get("title");
                eventsText += (eventTitle == null ? "untitled" : 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**<sup>12</sup>.

### 2.4.2 Intents

Een **Intent** is een object dat je kan gebruiken om een actie op te vragen vanuit een andere component. Intents worden vooral gebruikt om activiteiten te starten, achtergrondactiviteiten aan te roepen en om berichten te broadcasten. We overlopen hier een scenario waarbij elk van deze aspecten gedemonstreerd worden. We werken met **expliciete intents**. Deze hebben als beperking dat ze enkel binnen in een toepassing kunnen gebruikt worden, maar hebben als groot voordeel dat ze eenvoudig te definiëren en makkelijk af te handelen zijn. **Impliciete** intents daarentegen definiëren een algemene actie, waardoor een geschikte component in een andere toepassing deze ook kan afhandelen<sup>13</sup>.

1. Voeg een nieuwe **Button** toe naast de bestaande en koppel deze in de **MainActivity** zoals voorheen.
2. Maak een nieuwe methode-actie en koppel deze aan de nieuwe button die je hebt aangemaakt. Deze moet opnieuw de tekst inlezen uit het veld **editText** en de input kort tonen. Extraheer hiervoor best de methode van de andere knop en hergebruik deze in de actie van de nieuwe. Je gebruikt hiervoor best de **Extract method...**-functie in Eclipse om een nieuwe private methode `readInputText` te genereren.

<sup>11</sup> <http://www.json.org/javadoc/org/json/JSONObject.html>

<sup>12</sup> <http://developer.android.com/guide/topics/ui/layout/listview.html>

<sup>13</sup> <http://developer.android.com/guide/components/intents-filters.html>

Listing 7: Methode om tekst uit het invoerveld te lezen

```
private String readInputText() {
    String text = editText.getText().toString();
    Toast.makeText(this, "Je voerde in: " + text, Toast.LENGTH_LONG).show();
    return text;
}
```

3. Vervolgens maak je een nieuwe klasse aan die overerft van **IntentService**<sup>14</sup>.

Listing 8: Klasse die IntentService uitbreidt.

```
public class DescriptionService extends IntentService {
    public DescriptionService() {
        super("DescriptionService");
    }

    @Override
    protected void onHandleIntent(Intent workIntent) {
        String dataString = workIntent.getStringExtra("text");
        Log.v("ready", dataString);
    }
}
```

4. Vervolgens registreren we de intent in het **AndroidManifest.xml**. Zorg ervoor dat je die opent in de XML-weergave en voeg daartoe volgende regels toe tussen de **application**-tags na de afsluitende **activity**-tag:

Listing 9: Registratie van de IntentService in het AndroidManifest

```
<service
    android:name=".DescriptionService"
    android:exported="false" />
```

Merk op dat **android:exported** op **false** staat: dit betekent dat deze service enkel toegankelijk is binnen deze toepassing. De naam komt overeen met de **IntentService** die we net hebben aangemaakt: **DescriptionService** (let op het extra punt voor de naam).

5. De methode die opgeroepen wordt door op de nieuwe knop drukken moet deze IntentService aanroepen met de ingevoerde tekst als extra parameter.

Listing 10: Oproepen van de IntentService vanuit de MainActivity

```
public void doDescribe(View view){
    String text = readInputText();
    if(text.length() > 0) {
        Intent mServiceIntent = new Intent(MainActivity.this, DescribeService.class);
        mServiceIntent.putExtra("text", text);
        this.startService(mServiceIntent);
    }
}
```

6. Start de toepassing en verifieer dat een druk op de knop ervoor zorgt dat de ingevoerde tekst verschijnt in **LogCat** in Eclipse.
7. In plaats van de tekst in Eclipse te tonen, willen we deze uiteraard terugsturen naar de activiteit om weergegeven op het scherm. Daarvoor moeten we in de MainActivity berichten kunnen ontvangen. De eenvoudigste manier hiervoor is een **BroadcastReceiver**, die luistert naar berichten die gericht zijn aan alle toepassingen. We zullen deze echter expliciet definiëren om enkel te reageren op de IntentService die we hebben aangemaakt. Volgende code komt terecht in de MainActivity-klasse:

Listing 11: Een BroadcastReceiver aanmaken

```
private BroadcastReceiver myReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
```

<sup>14</sup>Zie ook <http://developer.android.com/guide/components/services.html>

```

        Toast.makeText(MainActivity.this, "Je ontving: " + intent.getStringExtra("response"),
                        Toast.LENGTH_LONG).show();
    }
};

```

We moeten deze ontvanger ook registreren, zodat deze wel degelijk luistert naar bepaalde acties en intents. Voeg de code toe aan de onCreate-methode in MainActivity-klasse.

#### Listing 12: De BroadcastReceiver registreren

```
registerReceiver(myReceiver, new IntentFilter("describe.action"));
```

8. Als de activiteit niet meer bestaat, moet de listener gestopt worden.

#### Listing 13: De BroadcastReceiver verwijderen als MainActivity niet meer bestaat

```

@Override
public void onDestroy() {
    super.onDestroy();
    unregisterReceiver(myReceiver);
}

```

9. Zorg dat de IntentService deze extra parameters broadcast na het ontvangen van de user input.

#### Listing 14: Een nieuwe intent aanmaken

```

Intent intent = new Intent("describe.action");
intent.putExtra("response", dataString);
sendBroadcast(intent);

```

Merk hierbij op dat we een intentfilter koppelen aan een bepaalde actie. Deze actie zal uitgevoerd worden door de **IntentService**. We zien ook dat de ontvanger verwacht dat er extra tekst wordt meegestuurd met de intent in de parameter **response**.

10. Zorg er nu voor dat beide knoppen vergelijkbaar gedrag vertonen. Let hierbij op dat, als je code hergebruikt uit de AsyncTask, je deze best verpakt in een aparte methode voor hergebruik in een andere klasse. Verzorg tevens de package-hiërarchie<sup>15</sup>.

<sup>15</sup><http://www.therealjoshua.com/2011/11/android-architecture-part-2-packages/>

## 3 Opgave

### 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 de 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 dus niet blokkeren – gebruik dus AsyncTask en/of IntentService;

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.

### 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 vak *Multimedia* zag je hoe je via CSS ervoor kan zorgen dat HTML-pagina's correct weergeven op kleinere schermen. Wat zijn de redenen om al dan niet te kiezen voor platformspecifieke toepassingen? Vermeld deze in een verslag van een half of een volledig A4-blad.

### 3.3 Indienen

Je dient een zip-bestand in met naam `itech-p2-g<groep>.zip` dat je volledige project bevat. Daarin plaats je je verslag als `reflectie.pdf`. 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 objectgeëoriënteerde talen in het algemeen. Vermijd bijvoorbeeld dubbele code door aparte (ouder-)klassen te gebruiken.

### 3.4 Voorbeeldgebruik van de evenementen-API

Hoewel de API (zoals iedere REST-API) zichzelf zou moeten uitwijzen, geven we hieronder enkele voorbeeldjes met de curl-tool. Deze werken ook met <http://onlinecurl.com/>.

#### Listing 15: Voorbeeld-requests voor de evenementen-API

```
# Representaties 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
```