Herbert Braun, Noud van Kruysbergen

MyTube

Een YouTube-player op je eigen website zetten

Google maakt het makkelijk om video's op een website in te bedden. Wil je meer dan de standaard player, dan vind je bij Google een programmeerinterface waarmee je een individuele videoplayer in elkaar kunt zetten die helemaal bij de stijl van je website past.

Als je bij de broncode van je website kunt komen, kun je daar ook een YouTube-player inbouwen. YouTube biedt voor een stuk of tien webdiensten als Facebook, Tumblr en Pinterest kant-en-klare copy/paste-oplossingen. Daar zit ook een HTML-fragment bij voor een eigen website. Die werkt met een iframe:

```
<iframe width="560" height="315"
    src="https://www.youtube.com/embed/a_426RiwST8"
                                frameborder="0" allowfullscreen>
</iframe>
```

Dit frame wordt gevuld met de gebruikelijke HTML5- of Flash-player. YouTube geeft de voorkeur aan de HTML5-player en grijpt alleen terug op Flash als iemand je webpagina oproept met een verouderde browser. De attributen width, height, frameborder en allowfullscreen zijn standaard voor iframes. De video wordt aangepast aan de grootte van het frame, maar behoudt zijn proporties. Met allowfullscreen heb je de mogelijkheid om de player het hele scherm te laten vullen.

Inbedden

Met een aantal url-parameters kun je zelfs die simpele iframe-oplossing aan verschillende omstandigheden aanpassen, bijvoorbeeld:

```
https://www.youtube.com/embed/a_426RiwST8?
        autoplay=1&loop=1&start=21&end=25&fs=0&
    showinfo=0&hl=nl&modestbranding=1&theme=light
```

Met autoplay=1 start de video automatisch, met loop wordt hij herhaald. Met start en end stel je in vanaf welke seconde tot welke seconde de video moet draaien. Normaal toont de player de titel van de video die je aan het bekijken bent als je daar met de muis overheen beweegt, maar showinfo=0 verhindert dat. Door fs=0 toe te voegen verdwijnt de fullscreen-knop, maar dat kun je ook doen door allowfullscreen bij <iframe> weg te halen. Met modestbranding schakel je het YouTube-logo op de knoppenbalk uit. Dat komt dan alleen rechtsonder in de video te staan als je er met de muis overheen beweegt en als je de video pauzeert. Bij de Flash-player biedt YouTube naast het standaard-theme dark ook de variant light.

Met de playlist-parameter kun je meerdere video's na elkaar afspelen:

```
https://www.youtube.com/embed/a_426RiwST8?
                playlist=id0fpT4VMzo,sUTK_TGoS40
```

De player speelt dan drie video's af, te beginnen met a_426RiwST8. Voor playlists die bij YouTube staan heb je daarentegen de parameter list nodig. Daarmee kun je favorieten, positief beoordeelde video's, eigen lijsten (listType=playlist), kanalen (listType=user_uploads) en zoekresultaten (listType=search) laten afspelen:

```
https://www.youtube.com/embed/?
                list=LLVJ...&listType=playlist
```

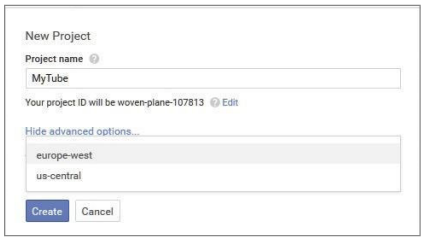
Een video-ID hoef je in dit geval niet mee te geven. Vervang in de HTML-code alle ampersands tussen de parameters door & om geen HTML-fouten te veroorzaken.

API-keuze

Voor veel doeleinden zijn deze aanpassingsmogelijkheden al voldoende. Bij een professioneel vormgegeven website wil je een player echter een eigen uiterlijk geven – een soort skin met aangepaste bedieningselementen. Een praktijkscenario is bijvoorbeeld om de video's van een YouTube-playlist overzichtelijk op een webpagina op te sommen en ze dan in een zelf vormgegeven player af te spelen.

YouTube heeft daar meerdere programmeerinterfaces voor. Met de YouTube Data API kun je video's vinden via zoektermen of playlists, maar ook video's uploaden en verwijderen en playlists bewerken. Met de iframe-player-API kun je het YouTube-iframe aansturen.

Stap één op de weg naar een individuele speler is YouTube met de Data API te bevragen. Omdat met die API ook data geschre-



Bij het registreren van de API kun je de locatie van de app-engine opgeven waar de player zijn data vandaan moet halen.

ven en verwijderd kunnen worden, moet de applicatie zich eerst legitimeren – ook als dat wat overdreven lijkt omdat je alleen een openbare lijst wilt openen.

De daarvoor benodigde toegangsgegevens krijg je door je aan te melden bij de Google Developer-console (zie de link aan het eind van dit artikel). Daar moet je een nieuw project aanmaken ('Create an empty project'). Let er bij de 'advanced options' op dat je als 'App Engine location' kiest voor 'europe-west' – niet uit privacy-overwegingen, maar omdat de data dan niet zo ver hoeven te reizen. Klik vervolgens op 'Create'.

Als dat gebeurd is, ga je naar 'APIs & auth / APIs' om de YouTube Data API te activeren. Voor ons voorbeeld is openbare API-toegang via de browser genoeg. Het gedoe met OAuth 2.0 om te authenticeren is alleen bedoeld voor applicaties die in naam van een aangemelde gebruiker YouTube-bestanden bewerken. Kies bij 'Credentials' dan ook voor 'Add credentials / API key' en maak een 'Server key' aan. Geef daarbij aan vanaf welke IP-adressen de sleutel gebruikt mag worden, en voeg daar tijdens het testen ook '127.0.0.1' aan toe.

De basis van een YouTube-applicatie kan er bijvoorbeeld zo uitzien:

```
<ol id="playlist"></ol>
```

```
<script type="text/javascript">
    var youtubeURL =
        'https://www.googleapis.com/youtube/v3/',
        youtubeAPIKey = '...',
        videolist = '...',
        data = [],
        ajax = function(param, token) {...},
        onPlaylistLoaded = function() {...};
    ajax(...);
</script>
```

Een -lijst dient als container voor de op te roepen data. YouTubes Data-API haalt telkens maar 50 datarecords op – en ook alleen maar als je de url-parameter maxResults gebruikt. Als de lijst langer is, moet je dat vaker doen. Daarom moet je het opvragen van de data (ajax) in een functie zetten – en om het overzichtelijk te houden de uitvoer van de data (onPlaylistLoaded) meteen ook maar.

REST-resources

De Data-API is een REST-interface – alle requests hebben het schema https://www.googleapis.com/youtube/v3/RESOURCE, gevolgd door een paar url-parameters, waaronder key met de API-sleutel. Het responsformaat is JSON.

Google maakt toegang tot meer dan tien resources mogelijk, waaronder commentaren (Comments), videodetails (Videos) en ondertitels (Captions). Voor het voorbeeld gaat het om de items op een playlist (PlaylistItems). Met de HTTP-methoden kun je items toevoegen (POST), veranderen (PUT) en verwijderen (DELETE), maar in dit geval volstaat lezen (GET). Dat betekent dat je de urls om te testen ook eenvoudig op de adresbalk van een browser kunt intypen als je geen domeinbeperking ingesteld hebt.

Naast key en de playlistId is ook de url-parameter part verplicht. Die bepaalt wat er precies van de server gehaald moet worden. Met snippet zijn dat in principe de titel, de beschrijving en een thumbnail.

De ajax()-functie maakt van het eerste argument de request-url. Als er als tweede argument een token meegegeven werd, wordt die met de parameter pageToken aan de url gehangen:

```
ajax = function(param, token) {
    var req = new XMLHttpRequest(),
    url = youtubeURL + param + '&key=' +
        youtubeAPIKey;
    if (token) url += '&pageToken=' + token;
    req.open('GET', url, true);
    req.onload = function() {...};
    req.onerror = function() {/*foutmelding*/};
    req.send();
}
ajax('playlistItems?part=snippet&playlistId=' +
    videolist + '&maxResults=50');
```

Datavracht

Als alles goed gaat, komt het antwoord met de statuscode 200 of 300. In dat geval hoef je de responseText alleen naar de JSON-parser van de JavaScript-engine door te sturen om de gewenste data eruit te kunnen halen:

```
req.onload = function() {
    if (req.status >= 200 && req.status < 400) {
        var newdata = JSON.parse(req.responseText);
        data.push.apply(data, newdata.items);
        if (data.length < newdata.pageInfo.
                totalResults && newdata.nextPageToken) {
            ajax(param, newdata.nextPageToken);
        } else {
            onPlaylistLoaded();
        }
    } else {/*foutmelding*/}
}
```

Het JSON-resultaat newdata is een object, de afzonderlijke playlist-items staan in de array newdata.items. Met data.push.apply() hang je de inhoud daarvan aan de eerder geïnitialiseerde array data. Dan wordt getest of het request alle datarecords gevonden heeft. Het aantal resultaten staat in newdata.pageInfo.totalResults.

Als er nog datarecords ontbreken (en er dus meer dan 50 zijn), moet de request bovendien een nextPageToken mee terugleveren. Die gebruik je bij een nieuwe ajax()-aanroep om de volgende rits datarecords binnen te halen. Als alles daarentegen compleet is, neemt onPlaylistLoaded() het over. Die functie zorgt voor een simpele HTML-uitvoer.

```
onPlaylistLoaded = function() {
    var olHtml = '';
    data.forEach(function(el, nr) {
        olHtml += '<li id="yt + ' nr + '">';
        olHtml += '<img src="' +
            el.snippet.thumbnails.default.url +
            '"alt="Thumbnail"/>';
        olHtml += '<div><h3>' + el.snippet.title +
            '</h3>';
        olHtml += '<p>' + el.snippet.description +
            '</p>';
        olHtml += '</div></li>';
    });
    document.getElementById('playlist').
        innerHTML = olHtml;
};
```


```
45
  contentDetails
    caption  "false"
    definition  "sd"
    dimension  "2d"
    durSec  301
    duration  "PT5M1S"
    licensedContent  false
    __proto__
  etag  ""xmg9xJZuZD438sF4hb-VcBBREXc/z1TU5ldTkgHPQ5HvyvH9hbb0Qjw""
  id  "Va2UrymZwLE"
  kind  "youtube#video"
  snippet
    channelId  "UCVJvSL8kGTW3wcYEmBopOuw"
    channelTitle  "Herbert Braun"
    description  "UK - 1971"
    playlistId  "PL7_a5-g3-dZkMxXjNZjjwMYqNwdIDha3g"
    position  39
    publishedAt  "2015-05-21T15:56:39.000Z"
    resourceId
      kind  "youtube#video"
      videoId  "Va2UrymZwLE"
      __proto__
    thumbnails
      default
      high
      medium
      standard
      __proto__
    title  "Zakarrias - Never Reachin'"
    __proto__
  __proto__
46
```


De Data-API van YouTube heeft alle informatie over iedere video, maar haalt die pas na aandringen tevoorschijn.

De belangrijkste data van ieder item staan in dit snippet-object. YouTube levert thumbnails in drie varianten, de hier gekozen default-

grootte is normaal gesproken 120 bij 90 pixels.

Voor het afspelen heb je later de YouTube-ID van een video nodig. Die staat in snippet.resourceId.videoId. Je kunt die waarde het beste niet rechtstreeks meegeven, maar via het doorlopende datarecordnummer dat je in de ID van het -element zet – dan kan de player makkelijk bij de tot nu toe verzamelde data komen.

De code die je bij de link aan het eind van dit artikel kunt downloaden, bevat nog wat extra's die met de playlistItems-resource niet meteen te realiseren zijn. Daar kun je bijvoorbeeld niet mee achterhalen hoe lang een video is en of een bepaalde video hier wellicht geblokkeerd wordt.

Om die twee dingen wel te achterhalen moet je de videos-resource met de parameter part=contentDetails oproepen, maar die ondersteunt weer geen playlists als zoekcriterium.

Je moet dan met playlistItems de ID's achterhalen en die in een tweede Ajax-doorloop als kommagescheiden lijst naar YouTube sturen – opnieuw in pakketten van 50, bij grotere aantallen antwoordt de API met een foutmelding. Daarna moet je de antwoorddata samenvoegen. Dat is net zo'n gedoe als het klinkt – met een beter doordachte API zou dat makkelijker moeten kunnen.

Als beloning krijg je een visualisatie van de playlist, waar met CSS nog wat aan bijgeschaafd kan worden – en vooral de mogelijkheid om de afzonderlijke video's ook af te spelen.

Filmdoek

Met een klik op een video op de lijst moet een box met de player openen. De eerste stap daartoe is het aanklikbaar maken van de <h3> en de preview. Voeg de volgende regels toe aan onPlaylistLoaded():

```
var clickable = document.querySelectorAll('li h3, li img');
for (var i = 0; i < clickable.length; i++) {
    clickable[i].addEventListener('click', function(ev) {
        var thisEl = ev.target;
        while (!thisEl.id || thisEl.id.indexOf('yt') < 0)
            thisEl = thisEl.parentNode;
        playMyVideo(parseInt(thisEl.id.substr(2)));
    });
}
```

Na het opbouwen van de lijst toont deze code de titels en de previews in de lijst met een onClick-event. De while-loop werkt zich via parentNode net zo lang door de DOM-tree omhoog tot hij een element met de bijpassende ID vindt. Daaruit extraheert hij de getalwaarde (pure getallen zijn geen geldige HTML-ID) en geeft die door aan de functie playMyVideo().

Met een eenvoudige versie van playMyVideo() kun je eerst controleren of dat werkt:

```
var playMyVideo = function(nr) {
    console.log(data[nr]);
};
```

De simpelste oplossing voor de player is om de code statisch in HTML te schrijven en die alleen zichtbaar te maken als dat nodig is. Dat kan er ongeveer zo uitzien:

```
<div id="playerBox">
    <h2 id="movieTitle"></h2>
    <div id="player"></div>
    <div>
        <span id="controlPlay">Play/Pauze</span>
        <span id="progressbar">
            <span id="progressbarInner"></span>
        </span>
    </div>
    <p id="movieDesc"></p>
    <span id="playerClose">Player sluiten</span>
</div>
```

De video zelf zit in – daarover later meer. De titel en beschrijving moeten als metadata verschijnen. Een Play/Pauze-knop start of stopt de video, een voortgangsbalk laat zien hoeveel tijd er verstreken is. De playerClose-knop laat de hele box weer verdwijnen.

Filmposter

Bij het laden maakt de stylesheet de #playerBox onzichtbaar, anders moet hij als gefixeerde box in de rechterbovenhoek van de webpagina te zien zijn:

```
#playerBox {
    position: fixed;
    display: none;
    top: 0;
    right: 0;
    margin: 0;
    width: 640px;
    background-color: #ddd;
}
```

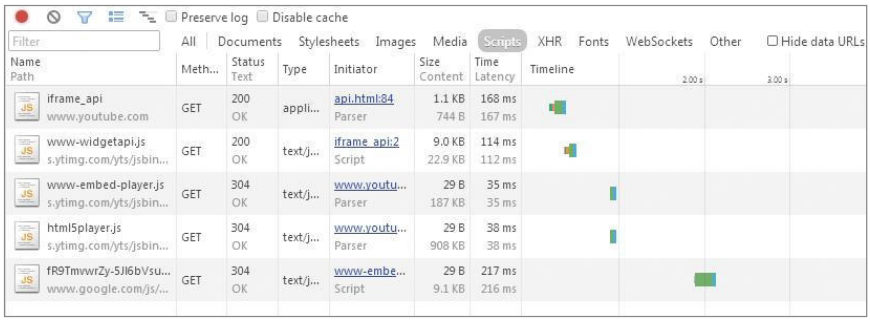
Een paar eenvoudige commando's vullen de voorbereide container met de content uit data[nr] en maken de playerBox zichtbaar:

```
var playerBox = document.getElementById('playerBox'),
playMyVideo = function(nr) {
    document.getElementById('movieTitle').
        textContent = data[nr].snippet.title;
    document.getElementById('movieDesc').
        textContent = data[nr].snippet.description;
    playerBox.style.display = 'block';
};
```

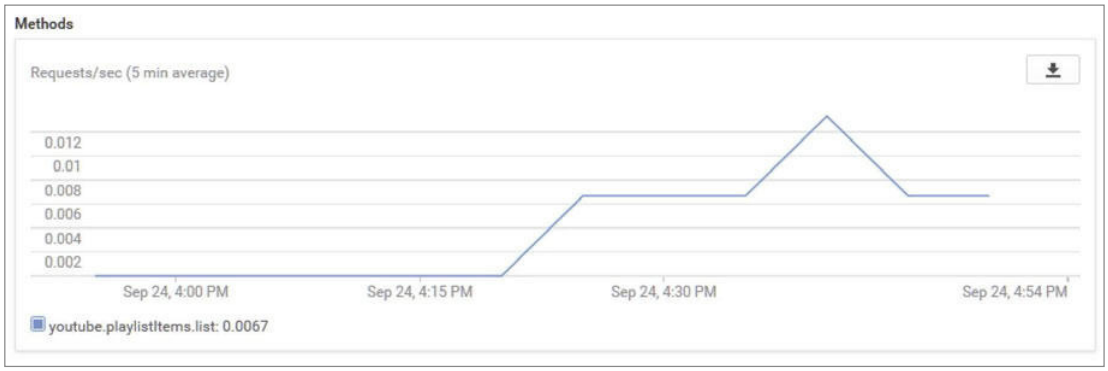
IFrame-API

Nu wordt het tijd voor de IFrame Player API om de video af te spelen. Deze hoeft niet geregistreerd te worden bij Google om hem te kunnen gebruiken. Je hoeft alleen een klein script toe te voegen:

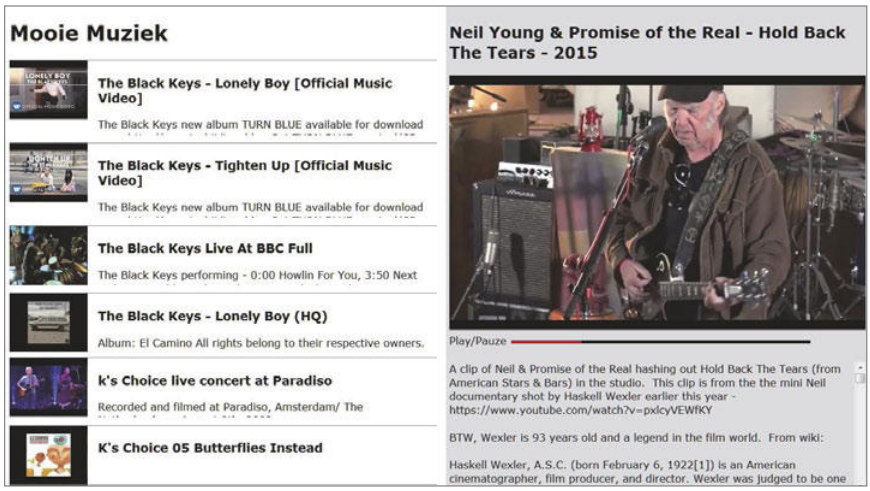
```
<script type="text/javascript" src="https://www.
                youtube.com/iframe_api"></script>
```



De IFrame Player-API downloadt meer dan een megabyte aan scripts voordat de eerste video gaat lopen.



Googles API-tools maken inzichtelijk hoe populair je eigen videoweb-site is.



De kant-en-klare voorbeeldplayer heeft verschillende sorteeropties en een interactieve slider en kan playlists afspelen.

Daarmee haalt de browser een paar honderd bytes aan code van de server. Die dient er alleen voor om andere scripts van s.ytimg.com te downloaden. Daar zit alles in: de totale grootte van de code overstijgt een megabyte – en die is dan ook nog geminimaliseerd. Het is moeilijk te begrijpen waarom Google webpagina's dergelijke JavaScript-monsters laat laden terwijl de API niet bijzonder complex is.

Projector opstellen

Als de API-scripts geladen zijn, begint de functie onYouTubeIframeAPIReady():

```
var onYouTubeIframeAPIReady = function() {
    player = new YT.Player('player', {
        width: 640,
        height: 390,
        playerVars: {
            controls: 0,
            showinfo: 0,
            modestbranding: 1
        },
        events: {
            'onStateChange': onStateChange,
            'onError': onError
        }
    });
};
```

Daarbij maakt new YT.Player() een instantie van de YouTube-player aan die je van nu af aan als player kunt aanspreken. Het eerste argument is de ID van het HTML-element waar de video zich bevindt. Bij het initialiseren vervangt de API dat door een iframe. Je kunt die echter ook rechtstreeks in HTML zetten.

Het tweede argument bevat een object met de instellingen. De hier gebruikte waarden voor width en height komen overeen met de standaard instellingen. Met een extra videoID zou de player meteen beginnen met het laden van een video. In dit geval moet je echter een video selecteren uit de via de Data-API opgevraagde playlist.

In de playerVars herken je de url-parameters van de ingebedde player. Hier zorg je ervoor dat YouTube behalve de video zelf zo min mogelijk laat zien.

Afspelen

De functie registreert bovendien twee gebeurtenissen: onStateChange() wordt geactiveerd als de afspeelstatus veranderd wordt, bijvoorbeeld van Play naar Pauze, en onError() wordt actief als YouTube de parameters niet begrijpt of een video bijvoorbeeld verwijderd is. Om ervoor te zorgen dat het script geen fouten produceert, maak je wat provisorische oplossingen voor de geregistreerde event-handlers:

```
var onStateChange = function(ev) {
    console.log(ev.data);
},
onError = function(ev) {
    console.log(ev);
};
```

Om de eerste video te kunnen afspelen, ontbreekt nog een regel, die je aan playMyVideo() toevoegt:

```
player.loadVideoById(data[nr].snippet.resourceId.videoId);
```

Met deze iFrame-API-methode laadt en start de player de video waarvan de ID in het datarecord data[nr] staat. De functie loadVideoById() voegt de twee functies cueVideoById (bufferen) en playVideo() (afspelen) samen.

Afstandsbediening

De player wordt bediend met een paar knoppen – #playerClose beëindigt bijvoorbeeld de video en sluit de videobox:

```
document.getElementById('playerClose').
    addEventListener('click', function() {
        player.stopVideo();
        playerBox.style.display = 'none';
});
```

Een klik op de Play-knop moet de weergave pauzeren of weer voortzetten:

```
document.getElementById('controlPlay').
    addEventListener('click', function() {
        if (player.getPlayerState() === 1)
            player.pauseVideo();
        else
            player.playVideo();
});
```

Met getPlayerState() achterhaal je of de player aan het afspelen is – daar staat de waarde 1 voor –, pauzeert, buffert of gestopt is. Iedere verandering van de playerstatus roept de eerder gedefinieerde onStateChange()-functie op. Overigens heeft een muisklik op de video hetzelfde effect als de zelfgemaakte Pauze-knop.

De meeste playerfuncties zijn vergelijkbaar simpel aan te sturen. Bij de link aan het eind van dit artikel staat ook een variant van het script die een playlist afspeelt. Die playlist geef je in de vorm van een array mee aan cuePlaylist() of loadPlaylist(). Met nextVideo() en previousVideo() kun je door de lijst heen en weer springen.

De IFrame-API heeft met setVolume(), mute() en unMute() meerdere mogelijkheden om het volume te beïnvloeden. Met setSize() verandert de grootte van de video en met seekTo() kun je naar een bepaald tijdpunt in seconden van de video springen.

Voortgang

Wat de player nog mist, is een voortgangsbalk. In HTML zit die als #progressbar. De stylesheet maakt daar een vijf pixels dikke zwarte horizontale streep van:

```
#progressbar {
    display: inline-block;
    position: relative;
    width: calc(100% - 180px);
    height: 5px;
    margin: 12px 2px;
    background-color: black;
    vertical-align: middle;
}
```

Op deze balk moet tijdens het afspelen een rode streep komen te staan die aan het begin nog breedte nul heeft:

```
#progressbarInner {
    display: block;
    width: 0;
    height: 3px;
    margin: 1px 0;
    background-color: red;
    text-align: left
}
```

Het script moet de rode streep continu langer maken zolang de player aan het afspelen is. Je kunt daar de bediening van de Play/Pauze-knop voor in de gaten houden, maar dat zal tot fouten leiden als de kijker de video zelf stopt door er rechtstreeks op te klikken. Maar we hebben de functie onStateChange() nog: als de data-eigenschap van de meegeleverde gebeurtenis de waarde 1 heeft, is de player nog aan het spelen, anders niet.

De standaardfunctie setInterval() zorgt ervoor dat de voortgangsbalk steeds bijgewerkt wordt. Een zinvol interval is bijvoorbeeld 300 milliseconden. De API-functies getCurrentTime() (de verstreken tijd) en getDuration(), die de totale duur van de video teruglevert, leveren de benodigde gegevens – beide in seconden. Dan hoef je alleen nog maar het quotiënt van die twee met de breedte van de voortgangsbalk te vermenigvuldigen. Die breedte achterhaal je met progressbar.getBoundingClientRect().width. In JavaScript ziet dat er zo uit:

```
var progressbarInterval,
onStateChange = function(ev) {
    clearInterval(progressbarInterval);
    if (ev.data === 1) {
        progressbarInterval = setInterval(function() {
            progressbarInner.style.width =
        (player.getCurrentTime() / player.getDuration()) *
        progressbar.getBoundingClientRect().width + 'px';
        }, 300);
    }
};
```

De clearInterval() aan het begin schakelt de andere playerstatus uit en zorgt dat de rode balk niet verder gaat. Hij is echter ook voor de Play-status nodig, anders zou ieder eenmaal aangemaakt interval eindeloos doorlopen. Dan zou de voortgangsbalk schijnbaar willekeurig heen en weer springen. De code die bij de link aan het eind van dit artikel staat heeft ook de mogelijkheid om met een slider naar een willekeurige plek in de video springen.

API-perikelen

De IFrame-API ontsluit op een relatief simpele manier de weg naar een eigen YouTube-player die je geheel naar eigen inzichten kunt vormgeven. Naast de monstrueuze grootte van de te downloaden scriptbestanden zijn er maar een paar kleine problemen en beperkingen.

Ondertiteling is bijvoorbeeld irritant geregeld. Of je zelfbouw-player die weergeeft of niet, ligt eraan of de gebruiker bij zijn laatste YouTube-bezoek ondertiteling wilde zien of niet. De documentatie zegt niets over ondertitels – die stel je namelijk alleen met de playerparameters in.

Met het meesturen van cc_load_policy: 1 krijg je toegang tot de ondertitelmodule. Daar zijn er twee van: een voor de HTML5-player (captions) en een voor de oude Flash-variant (cc). Met player.unloadModule("captions"); zijn bijvoorbeeld de gedownloade HTML5-ondertitels uit te schakelen. De verwarrende werking op dit gebied wordt zeker niet duidelijker door de incomplete documentatie.

Merkwaardig genoeg biedt de API geen mogelijkheid om een 'chromeless' player (zoals in het voorbeeld zonder knoppen) fullscreen te krijgen. Dat is raar, omdat de player dat wel kan: je kunt een video fullscreen krijgen door erop te dubbelklikken.

Die dubbelklik via scripts imiteren lijkt niet mogelijk te zijn. Een workaround is de fullscreen-API die in moderne browsers zit, maar die heeft nog de producentprefix (bijvoorbeeld webkitRequestFullScreen) – en met een goede reden, omdat de standaard nog niet definitief is. Daarbij moet ook de grootte van de video veranderd worden.

YouTubes tools voor een doe-het-zelf player zijn dus niet perfect, zoals bij de Data-API al bleek. Google heeft onlangs het overzicht over de afzonderlijke API's verbeterd, zodat de drempel in ieder geval al lager is. Los van die detailproblemen ziet een website er met een volledig zelf vormgegeven videoplayer in ieder geval een stuk beter uit, en YouTube biedt daar alle tools voor. (nkr)

www.ct.nl/softlink/1512137