OAuth 2.0 Device Flow Grant

19 Apr 2016

When signing into apps and services on devices such as a Playstation or an Apple TV it can be immensely frustrating experience. Generally you will ordeal something similar to one of the following scenarios:

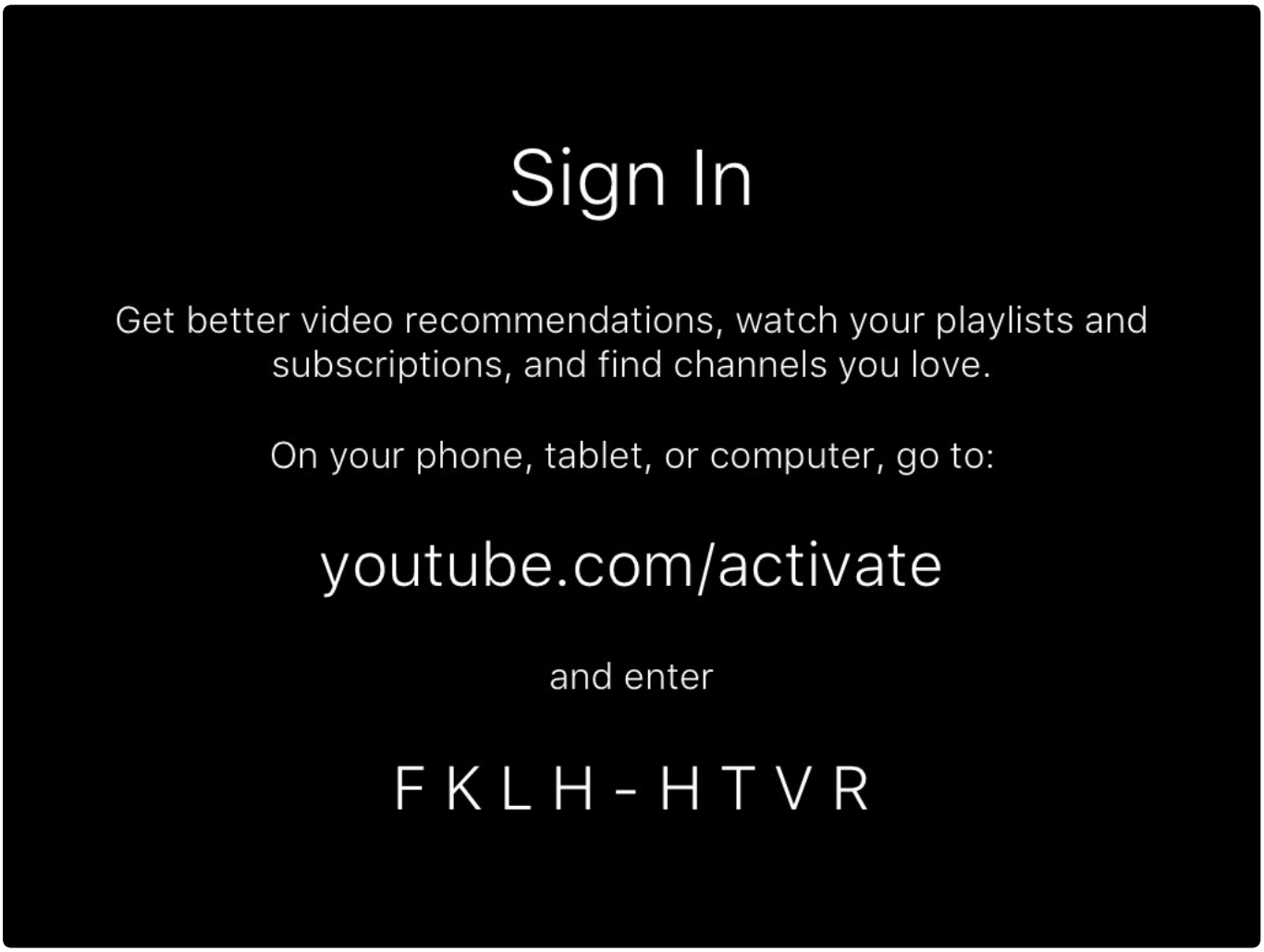
1. The utterly terrible experience whereby you don't have anything other than an onscreen keyboard and whatever pointing device/controller to enter your username and painfully complex 50 character password.
2. A slightly less terrible experience whereby you can pair a bluetooth

keyboard to enter your username and that crazy long password but you're balancing the keyboard on your knees whilst trying to copy the password out of 1password on your iPhone.

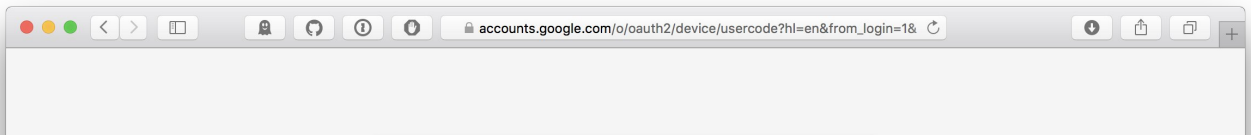
Of course if you use the same 8 character password everywhere then it's less of an issue but you can understand my predicament... ^_^

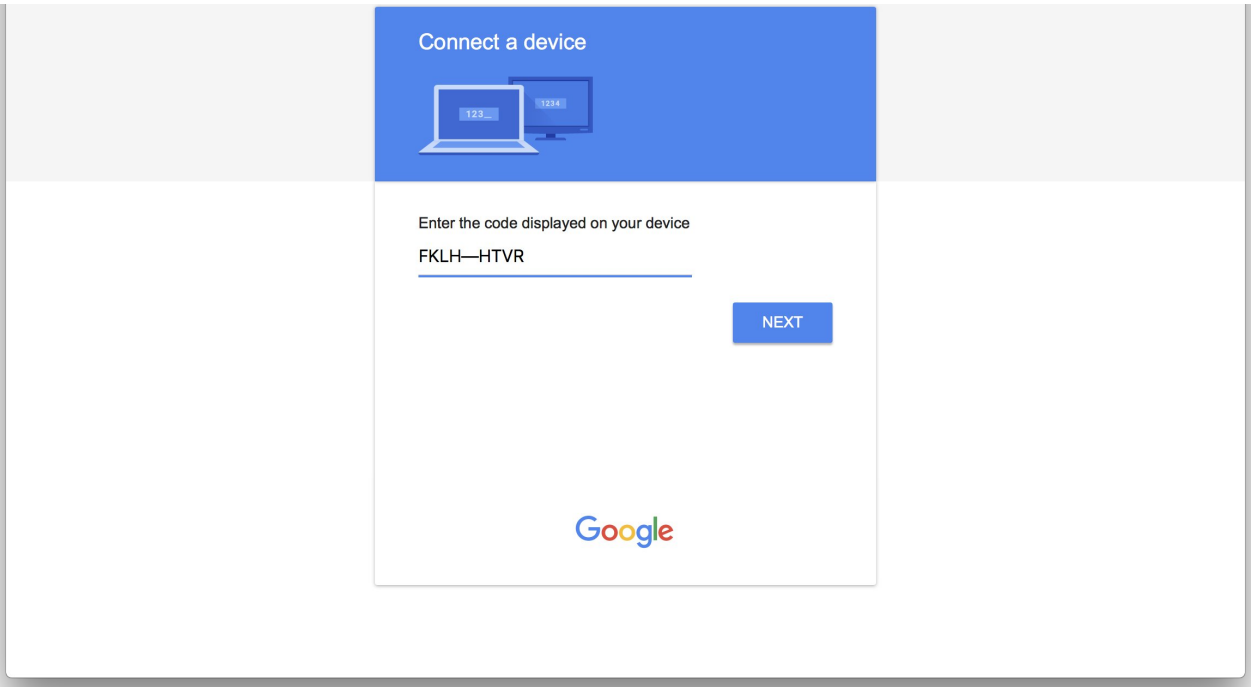
There are some apps however - such as Youtube for Apple TV - that have a much better end user experience.

When you sign into the YouTube app you're presented with this screen:

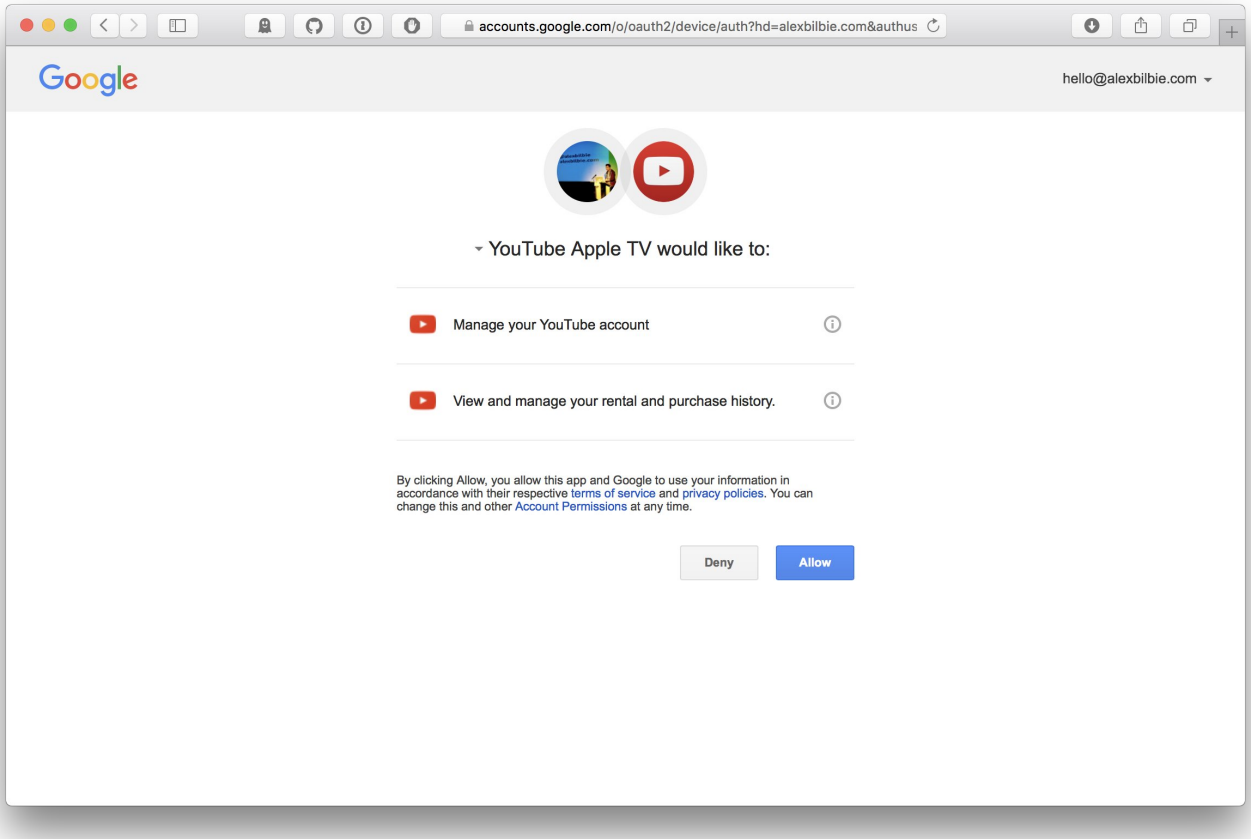


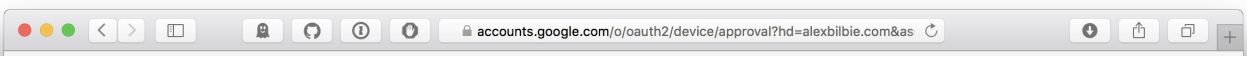
At the URL displayed on screen you see this (after signing into your Google account):

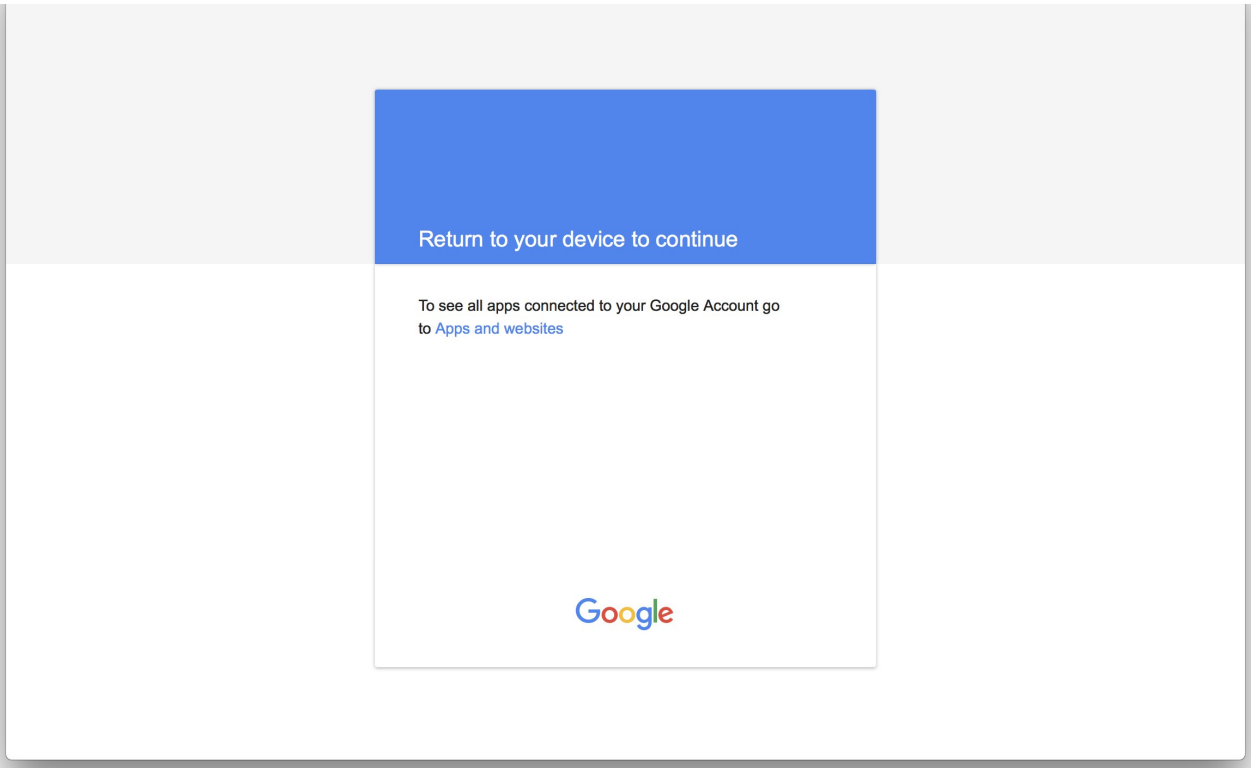




Having entered the code presented on the TV screen a standard OAuth authorisation dialog is shown:







A few seconds after click Allow the Youtube app had refreshed to show my account information.

Awesome!

This lovely UX features an implementation of the still very much in draft phase (at time of writing) OAuth 2.0 Device Flow Grant.

How it works

First an OAuth client (for example the Youtube app) makes a request to the authorization server:

```
POST /token HTTP/1.1
Host: authorization-server.com
Content-Type: application/x-www-form-urlencoded

response_type=device_code
&client_id=s6BhdRkqt3
```

The authorization server then responds with a JSON payload:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "verification_uri": "https://authorization-
server.com/authorize",
    "user_code": "94248",
    "device_code": "74tq5miHKB",
    "interval": 5
}
```

The `verification_uri` is the URL that the user should navigate to on another device.

The `user_code` is the code that the user should enter once they've authenticated with the authorization server.

Meanwhile the client should attempt to acquire an access token every few seconds (at a rate specified by `interval`) by POSTing to the access token endpoint on the authorization server:

```
POST /token HTTP/1.1
Host: authorization-server.com
Content-Type: application/x-www-form-urlencoded

grant_type=device_code
&client_id=s6BhdRkqt3
&code=74tq5miHKB
```

The value of `code` should be the `device_code` from the JSON response in the previous request.

The client should continue to request an access token repeatedly until the end-user grants or denies the request, or the verification code expires.

If the client is polling too frequently it will receive an error from the

authorisation server:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
        "error":"slow_down"
}
```

If the user hasn't yet authorised the client the error will be:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
        "error":"authorization_pending"
}
```

If the user authorises the client an access token will be returned:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "bearer",
    "expires_in": 3600,
    "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA"
}
```

If the user denies the client an error will be returned:

```
HTTP/1.1 400 Bad Request
```

```
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "error":"access_denied"
}
```

Conclusion

The device flow grant provides a really pleasant experience for devices that do not support an easy data-entry method. In addition it is really simple for developers to integrate with.

OAuth and API Consultation

If you'd like to hire me to help out with your OAuth or API implementation, or would like some advice or training for you and your team please email me at hello@glyndelabs.com.

Other posts you might like

Coding Solo episode 6 24 Nov 2017

Coding Solo episode 5 21 Sep 2017

Getting the AWS X-Ray deamon to run on Alpine Linux 31 Aug 2017