

Building an OAuth Flow in a Node.js CLI

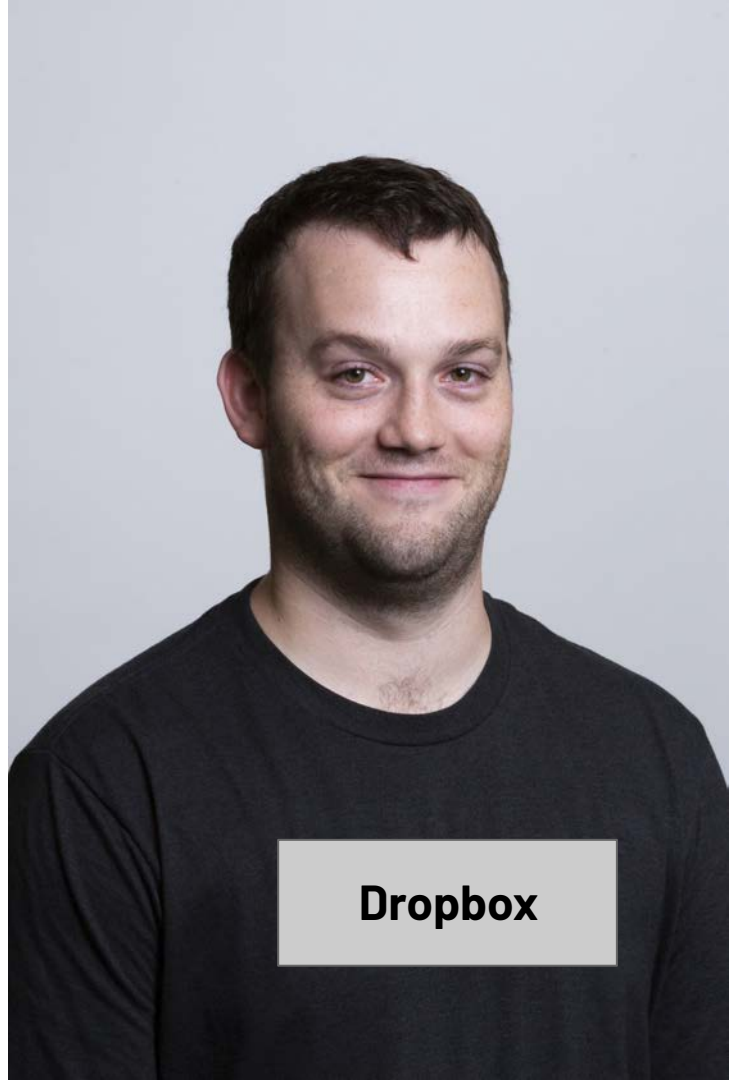
May 14th, 2019

Hi!

I'm **Taylor Krusen**.

I'm here to talk about
building an **OAuth Flow**
in a **Node CLI**.

I'm on Twitter:
[@TaylorKrusen](#)



Overview of talk

Discuss OAuth as a concept.

Cover 3-legged OAuth Flow.

Tinker with OAuth Flow implementations.



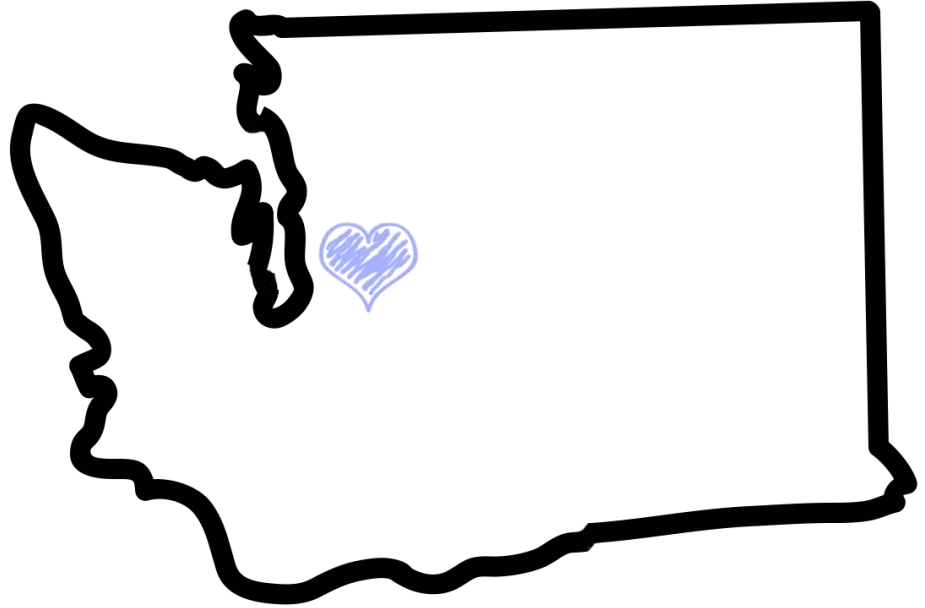
I'm from the USA. I live in a state named **Washington**.



... **Seattle**, Washington
to be precise.

We're known for **coffee**,
music, and a lack of
sunshine.

We have lots of **tech**
companies.



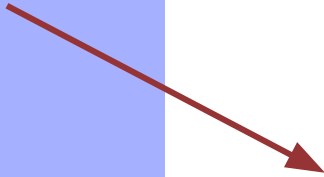
Life is good.



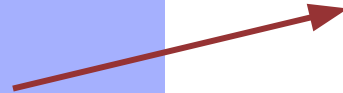
THE API ECONOMY

The Need

Bank Info

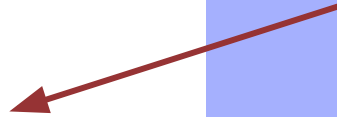


Social Info



Your Amazing App

Chat client



File storage





Recognize these?



HeartMonitor by **Me** would like the ability to access and write the following data in your Fitbit account for 1 week.

- ✓ sleep
- ✓ activity and exercise
- ✓ location and GPS
- ✓ food and water logs ⓘ
- ✓ heart rate
- ✓ profile ⓘ
- ✓ weight ⓘ
- ✓ Fitbit devices and settings
- ✓ friends ⓘ

Deny

Allow

Data shared with HeartMonitor will be governed by Me's privacy policy and terms of service. You can revoke this consent at any time in your Fitbit [account settings](#). More information about these permissions can be found [here](#).

Authorize Twitpic to use your account?

This application **will be able to**:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Sign In

Cancel



Twitpic
By Twitpic Inc
twitpic.com

Share photos on Twitter with Twitpic



OAuth 2.0

Industry-standard protocol for **authorization**.

- Provides **delegated access** to data between apps
- **Decouples** authentication and authorization
- Supports many **different use cases**
 - ◆ Web applications
 - ◆ Mobile applications
 - ◆ Consoles & IoT devices
 - ◆ Server-to-server applications

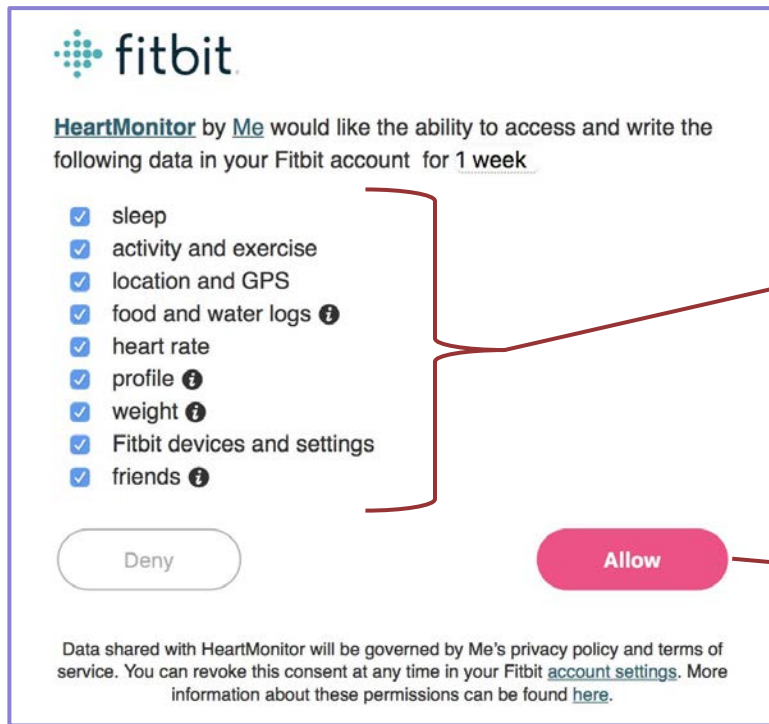
Authentication vs **authorization**



OAuth 2.0

- Scopes and consent
- Actors
- Clients
- Tokens
- Authorization server
- Flows

Scopes and Consent



The image shows a Fitbit authorization dialog. At the top is the Fitbit logo. Below it, text states: "HeartMonitor by Me would like the ability to access and write the following data in your Fitbit account for 1 week". A list of permissions follows, each with a checked checkbox: sleep, activity and exercise, location and GPS, food and water logs (with an info icon), heart rate, profile (with an info icon), weight (with an info icon), Fitbit devices and settings, and friends (with an info icon). A red bracket groups these permissions, with an arrow pointing to the "Scopes" section on the right. Below the list are two buttons: "Deny" and "Allow". A red arrow points from the "Allow" button to the text "Capture users consent" on the right. At the bottom, a line of small text reads: "Data shared with HeartMonitor will be governed by Me's privacy policy and terms of service. You can revoke this consent at any time in your Fitbit account settings. More information about these permissions can be found here."

fitbit

HeartMonitor by Me would like the ability to access and write the following data in your Fitbit account for 1 week

- ☒ sleep
- ☒ activity and exercise
- ☒ location and GPS
- ☒ food and water logs ⓘ
- ☒ heart rate
- ☒ profile ⓘ
- ☒ weight ⓘ
- ☒ Fitbit devices and settings
- ☒ friends ⓘ

Deny Allow

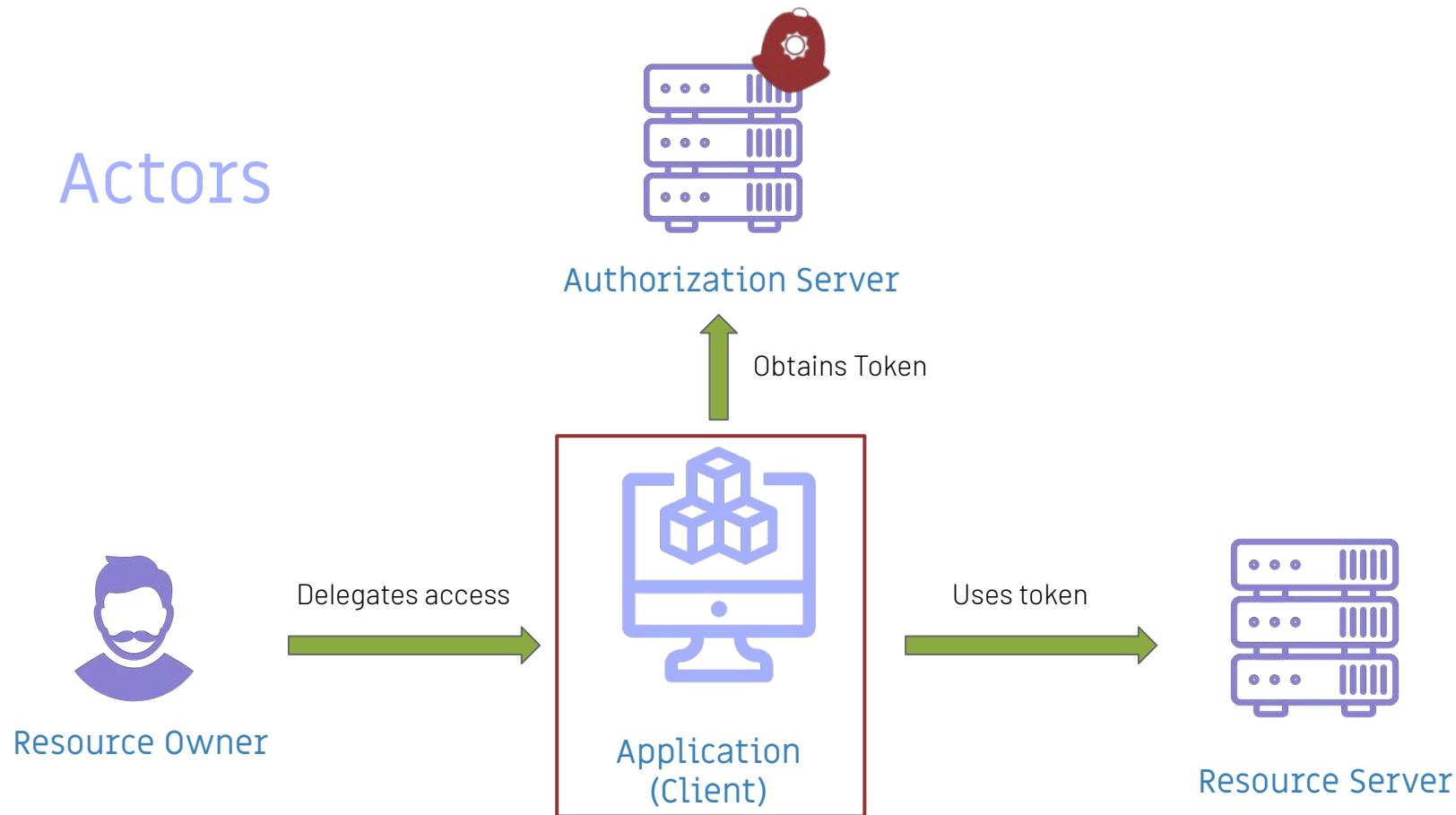
Data shared with HeartMonitor will be governed by Me's privacy policy and terms of service. You can revoke this consent at any time in your Fitbit [account settings](#). More information about these permissions can be found [here](#).

Scopes

- Bundles of **permissions** asked for by client when requesting access token
- Decouples authorization policy from enforcement

Capture users consent

Actors

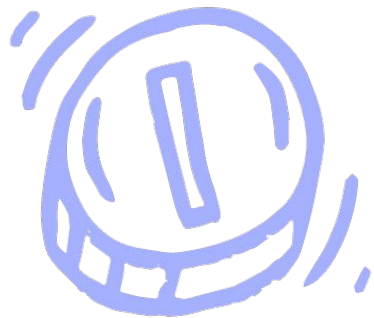


Tokens



Access Token

- Token used by **Client** to access **Resource Server** (API)

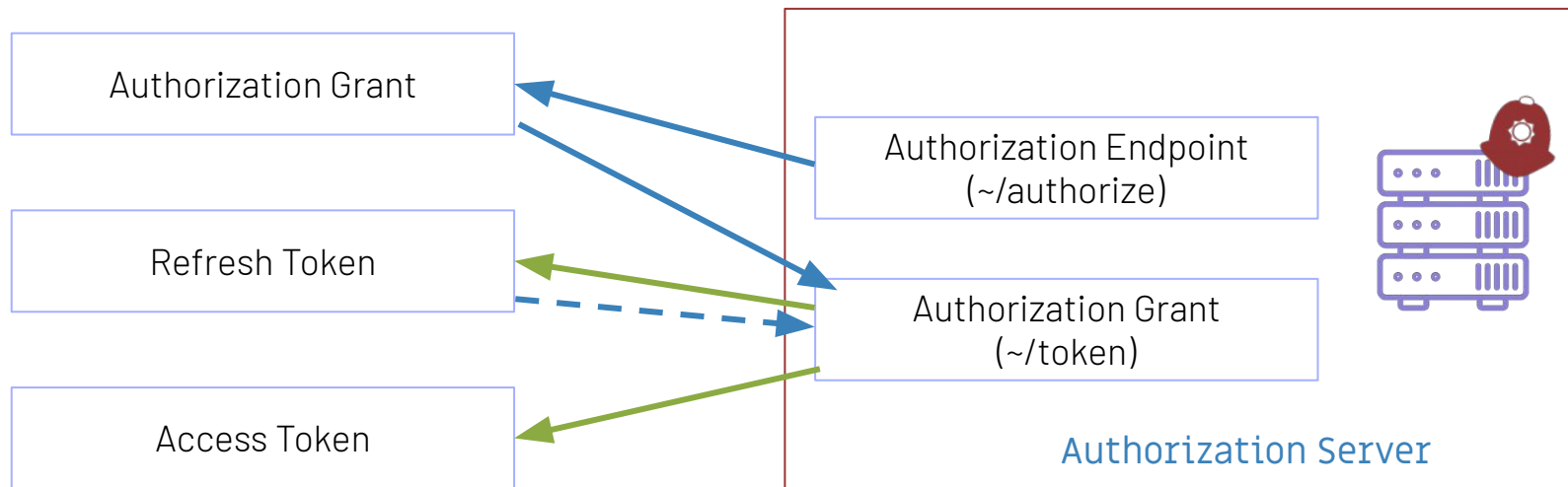


Refresh Token

(optional)

- Long-lived token for getting new access tokens from **Authorization Server**

Authorization Server



Flows

Implicit (2 Legged)

Best for browser-only
Public Clients

Auth Code (3 Legged)

Most secure.
Commonly used for
public APIs.

Device (Non-Standard)

Used for devices with
no access to browsers

Resource Owner Password

Legacy grant for native
UN / PW apps

Client Credential (2 Legged)

Used for Confidential
Clients (server-only)

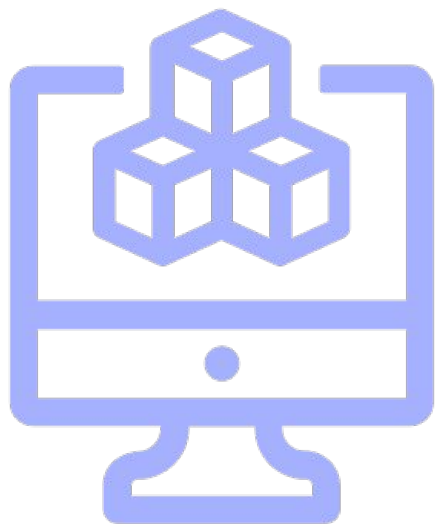
Assertion (2 Legged)

Authorization Server
can trust 3rd-party
such as SAML

3 Legged OAuth Flow (Auth Code)

- Uses **front channel flow** to get an **authorization code**
- Uses **back channel flow** to exchange authorization code for **access token** (optional refresh token)
- Most secure flow
- Assumes **Resource Owner** and **Client** are on separate devices

3 Legged OAuth Flow (Authorization Code)



Application
(Client)

1. Authorization Request

2. Authorization Grant

3. Authorization Grant

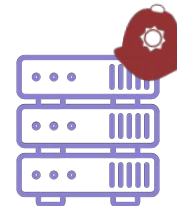
5. Access Token

5. Access Token

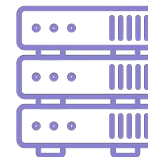
6. Protected Resource



Resource Owner



Authorization Server



Resource Server

Leg 1: Requesting Authorization

Request

GET



`https://www.dropbox.com/oauth2/authorize?
response_type=code&
client_id=9xhtame8mbpu3gz&
redirect_uri=http://localhost:3000/auth&
state=somestring`

Response



`http://localhost:3000/auth?
code=adfmWK8oHPAAAAAAAAAAAA6uvVPpX2xbiky2Q_VpxJKun&
state=somestring`

Leg 2: Requesting an Access Token

Request

GET



```
https://api.dropboxapi.com/oauth2/token?  
code=adfmWK8oHPAAAAAAAAAAAA7amOPmBYOPYiW3zooj_egxQ&  
grant_type=authorization_code&  
redirect_uri=http://localhost:3000/auth&  
client_id=9xhtame8mbpu3gz&  
client_secret=dei95eqke4bxnf7
```

Response



```
{  
  access_token: 'adfmWK8oHPAAAAAAAAAAAA7mxfnXIh_V47gTJ84g4mnDjgnaBD0fdlyNZK6AUhcFcr',  
  token_type: 'bearer',  
  uid: '2128328608',  
  account_id: 'dbid:AABIN7OVjwJNfLISv0DTKdzl1k_DwARnENY'  
}
```


Leg 3:

Requesting a Protected Resource



```
curl -X POST https://api.dropboxapi.com/2/users/get_current_account \  
  --header "Authorization: Bearer adfmWK8oHPAAAAAAAAA7mxfnXlh_V47gTJ84g4mnDjgnaBD0fdlyNZK6AUhcFcr"
```

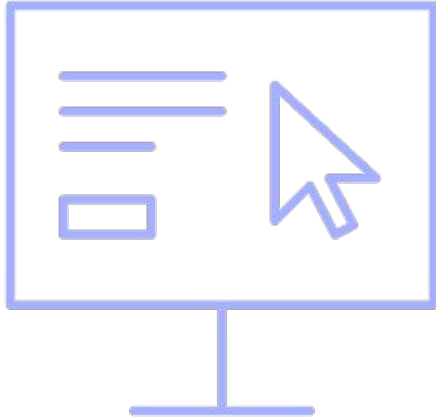
We did it!

A white silhouette of a person's head and shoulders is positioned in the bottom right corner of the blue background. A large thought bubble is connected to the head by two smaller circles. The thought bubble contains the text "...but how do we do that from a CLI?".

"...but how do we
do that from a
CLI?"

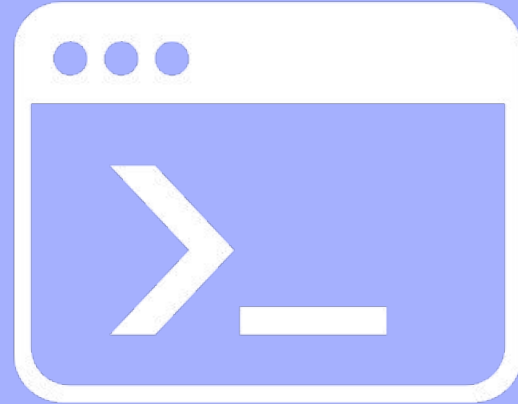
GUI

Graphical User Interface



CLI

Command-line Interface



Node CLI Frameworks

- Oclif
- Vorpai
- Commander
 - Smartsheet CLI
 - <https://github.com/smartsheet-samples/smartsheet-cli>
- Readline

Using Node Readline

```
const readline = require('readline');

let rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
  prompt: `~>`
});

rl.prompt();
```

```
let commands = {
  'auth': async () => {
    const authUrl = await dbx.getAuthenticationUrl(OAUTH_REDIRECT_URL, '', 'code');
    await getToken(authUrl);
  },
  'delete': () => {
    deleteToken();
  },
  'list': () => {
    listFolderContents();
  },
  'whoami': () => {
    getUserInfo();
  },
  'close': () => {
    rl.close();
  }
};

rl.on('line', (input) => {
  input = input.toLowerCase();
  if (input in commands) {
    commands[input]();
  }
});
```

Token STORAGE

```
function storeToken(token) {  
  return fs.ensureDir(constants.APP_DIR)  
    .then(() => {  
      const userAuthInfo = JSON.stringify(token);  
      const newFile = path.join(constants.APP_DIR, constants.TOKEN_FILE)  
      fs.writeFile(newFile, userAuthInfo, 'utf8', (err) => {  
        if (err) { return err; }  
      });  
    })  
    .catch((err) => {  
      console.error(err)  
    });  
}
```

- Saves to `Users/taylork/.dbx-cli/token.json`
- Probably not safe for production

Use Stored Token

```
fs.readFile(tokenFile, 'utf8', (err, token) => {  
  if (err !== null || token === '') {  
    // logic to get new token  
  } else {  
    // Found existing token  
    let parsedToken = JSON.parse(token);  
    dbx.setAccessToken(parsedToken);  
    dbx.usersGetCurrentAccount()  
    .catch(console.log)  
    console.log('Saved token is valid. Saved to DBX client.')  
  }  
})
```

These errors are a great way to test different retry logic.

Get New Token

```
fs.readFile(tokenFile, 'utf8', (err, token) => {
  if (err !== null || token === '') {
    const server = http.createServer( async (req, res) => {
      const parsedUrl = url.parse(req.url, true);
      const queryAsObject = parsedUrl.query;
      const authCode = queryAsObject.code;

      try {
        const newAccessToken = await dbx.getAccessTokenFromCode(OAUTH_REDIRECT_URL, authCode);
        await storeToken(newAccessToken);
        dbx.setAccessToken(newAccessToken)
      } catch (err) {
        console.log(err);
      }
      res.writeHead(302, {
        'Location': 'https://taylorkrusen.github.io/oauth_splash/',
      });
      res.end();

      req.connection.end();
      req.connection.destroy();
      server.close();
    }).listen(REDIRECT_PORT);

    opn(authUrl);
  } else {
    // logic to use stored token
  }
})
```

DEMO



THANKS!

Any questions?

I'm on Twitter:
[@TaylorKrusen](https://twitter.com/TaylorKrusen)

