Critter API docs

The Critter API is a REST API for a Twitter clone. It offers a limited subset of the kinds of things you can do on Twitter.

There is no user authentication. Instead, a hardcoded user is always logged in, `treasurymog`. You cannot log out, or switch users.

Some aspects of this API design are a little strange, and this is by design; often, APIs are a bit quirky, and learning how to use documentation to work with those quirks is a valuable skill 😃

> **NOTE**: Certain endpoints will fail about 5% of the time, with an unknown server error. Make sure you're handling these failures on the client!

Setup and ports

Run `yarn install` to install dependencies, and then `yarn start:server` to start the server.

The server runs on port **31415**, so you can access the server at `http://localhost:31415/`. As a test, you can go to `http://localhost:31415/api/me/profile` on your browser. You should see a JSON object:

```
{
  "profile": {
    "handle": "treasurymog",
    "displayName": "Gladstone, Esq.",
    "avatarSrc": "/assets/treasurymog-avatar.jpg",
    "bannerSrc": "/assets/treasurymog-banner.jpeg",
    "location": "Whitehall, London",
    "joined": "2016-10-12T12:00",
    "bio": "I live and work at the Treasury as a mouser but I also have a paw in the finances. Here to help lighten up the political world. Unofficial.",
    "numFollowing": 2,
    "numFollowers": 2,
    "numLikes": 1,
    "isFollowingYou": false,
    "isBeingFollowedByYou": false
  }
}
```

You will need to add a `proxy` to the server in the `package.json` of the `client`. This will allow you to use relative paths in your `fetch` requests to the server.

Add this line at the top of the `package.json` of the `client`, directly below the `"name"` key:

```
"proxy": "http://localhost:31415"
```

> NOTE: Once you've added the proxy, you need to restart your React server (if it was running)!

Endpoints

Endpoints are grouped in 3 categories:

- **profile** - relating to users and profiles
- **tweet** - single tweet
- **feeds** - groups of tweets

Profile Endpoints

These endpoints control user-specific things: getting profile data and following./unfollowing.

GET /api/me/profile

Get the profile for the currently-logged-in user.

Should come in this structure:

```
{
  "profile": {
    "handle": "treasurymog",
    "displayName": "Gladstone, Esq.",
    "avatarSrc": "/assets/treasurymog-avatar.jpg",
    "bannerSrc": "/assets/treasurymog-banner.jpeg",
    "location": "Whitehall, London",
    "joined": "2016-10-12T12:00",
    "bio": "I live and work at the Treasury as a mouser but I also have a paw in the finances. Here to help lighten up the political world. Unofficial.",
    "numFollowing": 2,
    "numFollowers": 2,
    "numLikes": 1,
    "isFollowingYou": false,
    "isBeingFollowedByYou": false
  }
}
```

GET /api/:handle/profile

Fetch the profile information for a specific user. Returns data in the same shape as `/api/me/profile`.

If the user handle supplied does not exist, it returns a 404 error of `user-not-found`.

GET /api/:handle/following - **(THIS IS A STRETCH GOAL)**

Returns an array of user profiles that the specified user is following.

```
{
  "following": [
    /* User profile 1, same shape as above endpoints */
    /* User profile 2, same shape as above endpoints */
    /* User profile 3, same shape as above endpoints */
  ]
}
```

GET /api/:handle/followers - **(THIS IS A STRETCH GOAL)**

Same as /api/:handle/following, but shows the user's followers (people who follow the user, instead of people that the user follows).

```
{
  "followers": [
    /* User profile 1, same shape as above endpoints */
    /* User profile 2, same shape as above endpoints */
    /* User profile 3, same shape as above endpoints */
  ]
}
```

PUT /api/:handle/follow - **(THIS IS A STRETCH GOAL)**

Follow the specified user, for the currently-logged-in user.

If you are *already following this user*, you'll get a "409 Conflict" error; you want to use the /unfollow endpoint instead.

If all goes well, you should receive the following response:

```
{
  "success": true
}
```

PUT /api/:handle/unfollow - **(THIS IS A STRETCH GOAL)**

Stop following the specified user, for the currently-logged-in user.

If you are *not following this user*, you'll get a "409 Conflict" error; you want to use the /follow endpoint instead.

If all goes well, you should receive the following response:

```
{
  "success": true
}
```

Tweet Endpoints

GET /api/tweet/:tweetId

Returns data about the specified tweet.

Example:

```
{
  "tweet": {
    "id": "1212689921057665024",
    "author": {
      "handle": "diplomog",
      "displayName": "Palmerston",
      "avatarSrc": "/assets/diplomog-avatar.jpg",
      "bannerSrc": "/assets/diplomog-banner.jpeg",
      "location": "Whitehall",
      "url": "http://fco.gov.uk",
      "joined": "2016-02-02T12:00",
      "bio": "Best friends with @treasurymog.",
      "numFollowing": 3,
      "numFollowers": 8,
      "numLikes": 2,
      "isFollowingYou": true,
      "isBeingFollowedByYou": true
    },
    "retweetFrom": /* possibly another profile, like 'diplomog' just above */,
    "timestamp": "2020-01-12T09:14:00+00:00",
    "isLiked": true,
    "isRetweeted": true,
    "numLikes": 1,
    "numRetweets": 0,
    "status": "Ok people #backtowork you go. Cats...just carry on lounging around
as usual.",
    "media": [
      {
        "type": "img",
        "url": "/assets/ENRXDPKWwAEJqFu.jpeg"
      }
    ]
  }
}
```

Some things worth highlighting:

- `status` is the text content of the tweet (what the person said)
- `isLiked` and `isRetweeted` convey whether the currently logged-in user (you!) has already liked or retweeted the tweet.

- Some data is omitted, like the full list of users who have liked/retweeted the tweet.
- `retweetFrom` might be undefined, if this tweet isn't a retweet of another tweet.
- `media` is an array of assets, though only images are supported, and no tweet has multiple images.

> **IMPORTANT:** this is the 'standard' tweet data format. When an API endpoint returns a tweet, it's probably going to come in this format.

POST /api/tweet

Create a new tweet from the current user. This is the endpoint you'll want to hit when you compose a new tweet and click "tweet" (or "meow").

You'll need to send a JSON body in the following format:

```
{
  "status": "your text here"
}
```

> **Important**: The API does not support uploading media. So you can only create new text tweets, not images. Sorry about that!

The endpoint will return your new tweet, in the standard tweet format (including the auto-generated unique ID).

PUT /api/tweet/:tweetId/like - **(THIS IS A STRETCH GOAL)**

Mark a tweet as "liked" from the current user, or remove an existing "like" status.

Important: You'll need to specify whether you're liking the current tweet or not, with a JSON body:

```
// Like a tweet that you don't already like:
{ like: true }

// Remove the "like" from a tweet you already like:
{ like: false }
```

If everything goes well, you'll get a response that looks like this:

```
{
  "success": true
}
```

You'll get an error if you try to like a tweet that is already liked, or unlike a tweet that is not liked.

PUT /api/tweet/:tweetId/retweet - **(THIS IS A STRETCH GOAL)**

A "retweet" takes someone else's tweet and copies it to your own timeline. It's a way of sharing a piece of content with your followers.

It works very similar to liking a tweet; you need to make a PUT with a JSON body that indicates whether you intend to retweet, or remove an existing retweet:

```
// Retweet a tweet :
{ retweet: true }

// Remove the "retweet":
{ retweet: false }
```

If everything goes well, you'll get a response that looks like this:

```
{
  "success": true
}
```

Feed Endpoints

All feed endpoints return data in the following structure:

```
{
  "tweetsById": {
    "ghi": /* Tweet with ID 'ghi' */,
    "def": /* Tweet with ID 'def' */,
    "abc": /* Tweet with ID 'abc' */,
  },
  "tweetIds": [
    "abc",
    "def",
    "ghi"
  ]
}
```

`tweetsById` is an object where the keys are IDs, and the values are the tweet objects themselves. We also return `tweetIds`, which tells the user which order the tweets should be shown in.

Tweets take the following shape:

```
{
  "id": "1212689921057665024",
  "author": {
    "handle": "diplomog",
    "displayName": "Palmerston",
    "avatarSrc": "/assets/diplomog-avatar.jpg",
    "bannerSrc": "/assets/diplomog-banner.jpeg",
    "location": "Whitehall",
    "url": "http://fco.gov.uk",
    "joined": "2016-02-02T12:00",
    "bio": "Best friends with @treasurymog.",
    "numFollowing": 3,
    "numFollowers": 8,
    "numLikes": 2,
    "isFollowingYou": true,
    "isBeingFollowedByYou": true
  },
  "retweetFrom": /* possibly another profile, like 'diplomog' just above */,
  "timestamp": "2020-01-12T09:14:00+00:00",
  "isLiked": true,
  "isRetweeted": true,
  "numLikes": 1,
  "numRetweets": 0,
  "status": "Ok people #backtowork you go. Cats...just carry on lounging around as
usual.",
  "media": [
    {
      "type": "img",
      "url": "/assets/ENRXDPKWwAEJqFu.jpeg"
    }
  ]
}
```

GET /api/me/home-feed

Get all the tweets from all the users that the current user is following.

> **HINT**: Make sure to read the section above to figure out how to render the tweets in the correct order!

GET /api/:handle/feed

Returns a list of tweets authored by the user specified with the :handle param. Includes any retweets that user has made.
