Using a REST API

REST APIs are extremely popular on the web and allow you to freely grab a site's data if that site has an available API over an HTTP connection.

If you've ever seen a music bot that accepts a YouTube query instead of just a video's URL, then you've seen a REST API in action. As a matter of fact, discord.js is made to use Discord's API. So, you've probably used an API yourself.

Making HTTP requests with Node

In these examples we are going to be using **node-fetch** which is a great library for making HTTP requests.

To install node-fetch, run the following command:

```sh
npm install node-fetch
```

Skeleton code

To start off, you're just going to be using this skeleton code:

```js
const { Client, MessageEmbed } = require('discord.js');

const client = new Client();
const prefix = '!';

client.once('ready', () => {
    console.log('Ready!');
});

client.on('message', async message => {
    if (!message.content.startsWith(prefix) || message.author.bot) return;
```

```
    const args = message.content.slice(prefix.length).trim().split(/ +/);
    const command = args.shift().toLowerCase();

    // ...
});

client.login('your-token-goes-here');
```

> **TIP**
>
> We're going to take advantage of **destructuring** in this tutorial to maintain readability.

Using node-fetch

node-fetch is a lightweight module that brings the **Fetch API** which is available in browsers to node. It is a promised based library. If you aren't already familiar with promises, you should read up on them **here**.

In this tutorial we'll be making a bot with 2 API-based commands. The first will be using **random.cat** and the other will use **Urban Dictionary**.

To require node-fetch, you'd do:

```js
const fetch = require('node-fetch');
```

Random Cat

Random cat's API is available at https://aws.random.cat/meow and returns a **JSON** response. To actually fetch data from the API, you're going to do the following:

```js
fetch('https://aws.random.cat/meow').then(response => response.json());
```

Now, of course it seems like this does nothing but what it's doing is launching a request to the random.cat server and random.cat is returning some JSON that contains a `file` property which is a string containing a link to a random cat. node-fetch returns a response object which

we can change into JSON with `response.json()`. Next, let's implement this into a command. The code should look similar to this:

```js
if (command === 'cat') {
    const { file } = await fetch('https://aws.random.cat/meow').then(response => respons

    message.channel.send(file);
}
```

So, here's what's happening in this code:

1. You're sending a `GET` request to random.cat.
2. random.cat sees your request and gets a random file from their database.
3. random.cat then sends that file's URL as a JSON object that contains a link to the image.
4. node-fetch receives the response and deserializes it with `response.json()`.
5. You then send the object's `file` property in Discord.

> **WARNING**
>
> The response will only be parsed if the server's `Content-Type` header includes `application/json`. In some cases you may have to apply the `.text()` method instead of `.json()` and `JSON.parse()` it yourself.

Urban Dictionary

Urban Dictionary's API is available at https://api.urbandictionary.com/v0/define, accepts a `term` parameter, and also returns a JSON response.

First, you're going to need to fetch data from the API. To do this, you'd do:

```js
const querystring = require('querystring');

if (command === 'urban') {
  if (!args.length) {
    return message.channel.send('You need to supply a search term!');
  }
```

```
    const query = querystring.stringify({ term: args.join(' ') });

  const { list } = await fetch(`https://api.urbandictionary.com/v0/define?${query}`).th
}
```

Here, we use Node's native **querystring module** to create a **query string** for the URL so that the Urban Dictionary server can parse it and know what to search for.

If you were to do `!urban hello world`, then the URL would become https://api.urbandictionary.com/v0/define?term=hello%20world since the string gets encoded.

You can get the respective properties from the returned JSON. If you were to view it in your browser, it usually looks like a bunch of mumbo jumbo. If it doesn't, great! If it does, then you should get a JSON formatter/viewer. If you're using Chrome, **JSON Formatter** is one of the more popular extensions. If you're not using Chrome, search for "JSON formatter/viewer <your browser>" and get one.

Now, if you look at the JSON, you can see that's a `list` property, which is an array of objects containing various definitions for the term (maximum 10). Something you always want to do when making API based commands is to handle there being no results. So, let's throw a random term in there (e.g. `njaksdcas`) and then look at the response. The `list` array should then be empty. Now you are ready to start writing!

As explained above we want to check if the API returned any answers for our query like so:

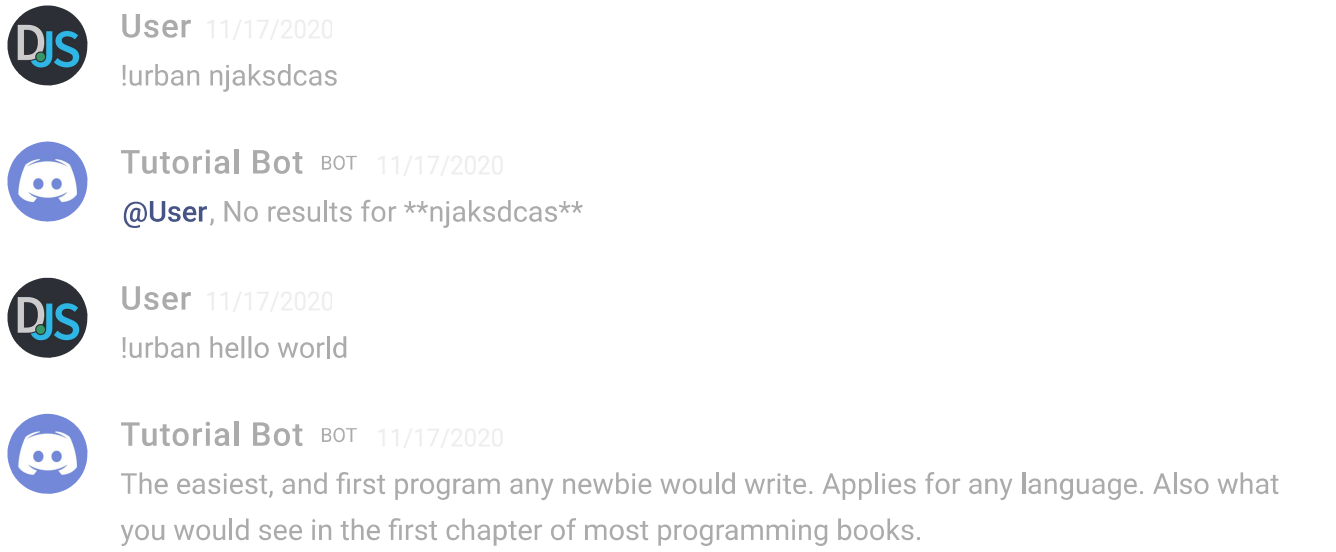
```js
if (!list.length) {
    return message.channel.send(`No results found for **${args.join(' ')}**.`);
}
```

After making sure that there are results, you will use those results. For now, let's simply send back the definition and nothing more. It's as simple as:

```js
message.channel.send(list[0].definition);
```

Here, you are simply getting the first object from the array of objects called `list` and grabbing its `definition` property.

If you've followed the tutorial, you should have something like this:



Now, let's just make this an **embed**.

We are also going to be defining a utility function at the top of our file so that our embed doesn't error when the field value is over 1024 characters. Here is a bit of code to do that:

```js
const trim = (str, max) => ((str.length > max) ? `${str.slice(0, max - 3)}...` : str);
```

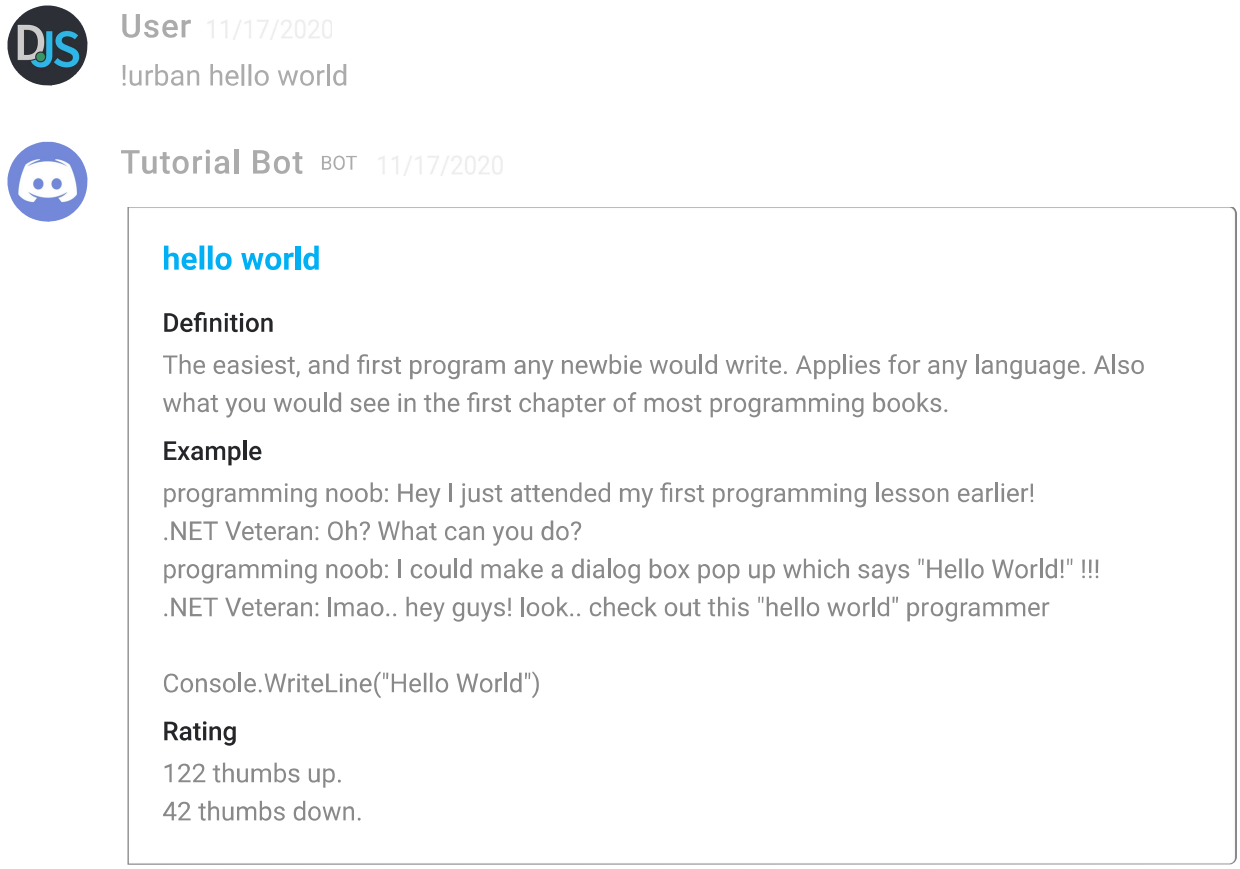
This is how we'll be structuring the embed:

```js
const [answer] = list;

const embed = new MessageEmbed()
    .setColor('#EFFF00')
    .setTitle(answer.word)
    .setURL(answer.permalink)
    .addFields(
        { name: 'Definition', value: trim(answer.definition, 1024) },
        { name: 'Example', value: trim(answer.example, 1024) },
        { name: 'Rating', value: `${answer.thumbs_up} thumbs up. ${answer.thumbs_down}
    );

message.channel.send(embed);
```

Now, if you do that same command again, you should get this:



Resulting code

If you want to compare your code to the code we've constructed so far, you can review it over on the GitHub repository **here**.

Edit this page

Last Updated: 10/22/2020, 4:30:58 PM

← Understanding async/await

Updating from v11 to v12 →