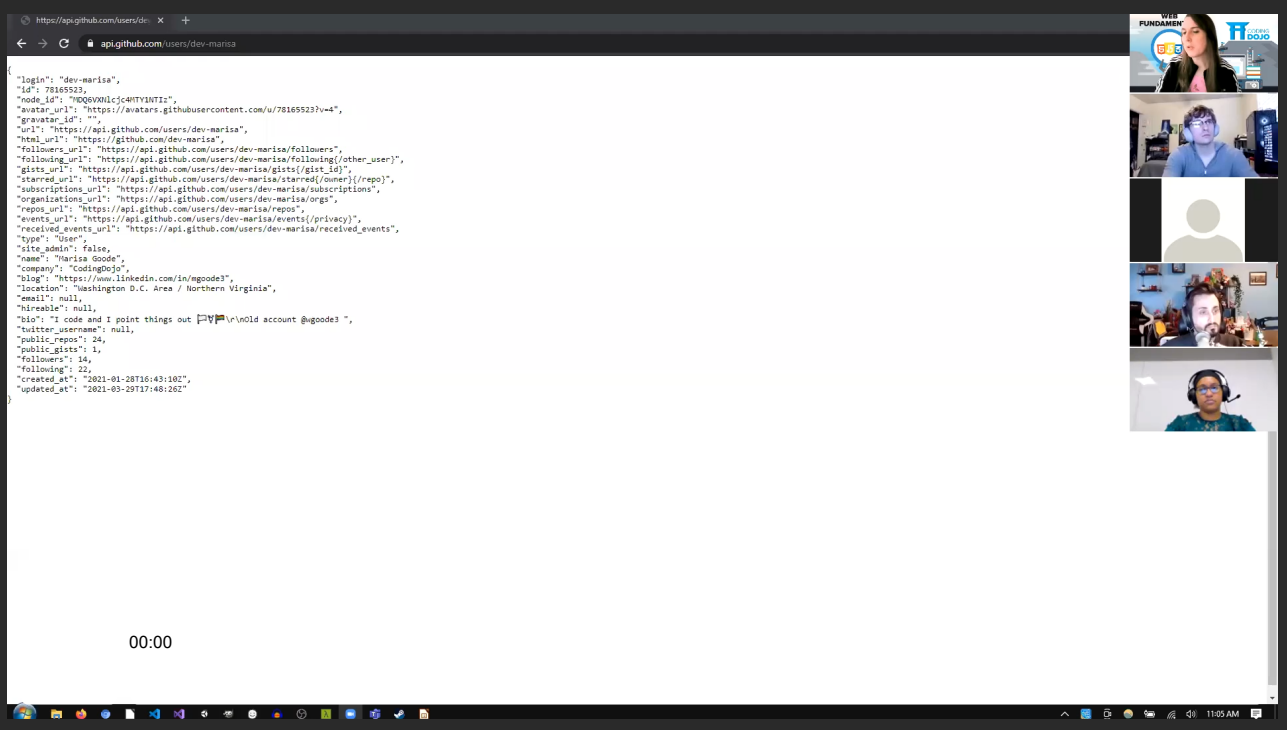
APIs and AJAX



APIs

An API (application programming interface) is a concept that will be important for us to understand. At the most basic level an API is the commands that we can use to make our application (our website) communicate with another application. We're mainly going to use this to retrieve information, but in the future we will be able to use an API to create, update, or delete data. For an example let's consider the GitHub API. If we navigate our browser to the following page `https://api.github.com/users/dev-marisa` we should see information that looks like the following.

JSON Raw Data Headers

Save Copy Pretty Print


```
{
  "login": "dev-marisa",
  "id": 78165523,
  "node_id": "MDQ6VXNlcjc4MTY1NTIz",
  "avatar_url": "https://avatars.githubusercontent.com/u/78165523?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/dev-marisa",
  "html_url": "https://github.com/dev-marisa",
  "followers_url": "https://api.github.com/users/dev-marisa/followers",
  "following_url": "https://api.github.com/users/dev-marisa/following{/other_user}",
  "gists_url": "https://api.github.com/users/dev-marisa/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/dev-marisa/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/dev-marisa/subscriptions",
  "organizations_url": "https://api.github.com/users/dev-marisa/orgs",
  "repos_url": "https://api.github.com/users/dev-marisa/repos",
  "events_url": "https://api.github.com/users/dev-marisa/events{/privacy}",
  "received_events_url": "https://api.github.com/users/dev-marisa/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Marisa Goode",
  "company": "CodingDojo",
  "blog": "https://www.linkedin.com/in/mgoode3",
  "location": "Washington D.C. Area / Northern Virginia",
  "email": null,
  "hireable": null,
  "bio": "Being cool with code 🏳️‍🌈\r\nOld account @wgoode3 ",
  "twitter_username": null,
  "public_repos": 38,
  "public_gists": 2,
  "followers": 33,
  "following": 50,
  "created_at": "2021-01-28T16:43:10Z",
  "updated_at": "2021-06-01T14:34:25Z"
}
```


If we look at the form the data is in it looks very similar to a **JavaScript** Object.

```
{
  "login": "dev-marisa",
  "id": 78165523,
  "node_id": "MDQ6VXNlcjc4MTY1NTIz",
  "avatar_url": "https://avatars.githubusercontent.com/u/78165523?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/dev-marisa",
  "html_url": "https://github.com/dev-marisa",
  // ... more lines left our for brevity
}
```

JSON

Often the form that we get our data back is in the form of **JSON** (JavaScript Object Notation). We can access the values inside of it using the keys just like with **JavaScript** Objects. To access the avatar image we can write code that will look like `data.avatar_url`. This could be used to set the `src` of an `img` tag.

`fetch`

The way we accessed the data by putting the url into our browser works to show the data, but if we wanted to bring it into our webpage we'll need to do this using code. The easiest way we can do this is using the built in function fetch. Fetch has good support across modern browsers. We can use the function to make what is known as an API call where we request the information from the url we provide. As this API call goes to a separate website the request must be sent and responded to before we can use the data. This is usually fairly quick, but we will need to write code to tell **JavaScript** to wait for the results.

```
async function getCoderData() {
    var response = await fetch("https://api.github.com/users/dev-marisa");
    var coderData = await response.json();
    console.log(coderData);
}

getCoderData();
```

In the above code we need to make to tell **JavaScript** in the browser to wait for the results to come back and to wait for the results to be converted to **JSON** format. We do that by using the `await` keyword. Any time we use the `await` keyword in a function in **JavaScript** we need to apply the `async` keyword to that function. We can still call it like a normal function though. If we're curious why we have to jump through so many hoops just to wait for the data in the API call, it's because **JavaScript** only runs in a single thread. If that thread is taken up by waiting for something like an API call it can slow down our website. To get around this **JavaScript** uses something called an event loop. This is a fairly advanced concept and something we will learn more about in later courses.

AJAX

When working with traditional servers, each interaction the user has with the website like navigating to a new page or submitting a form will result in a new page being sent and the webpage refreshing. AJAX (Asynchronous JavaScript and XML) gives us the ability to update data or submit data on a webpage without needing to refresh the page or redirect the user to another page. We haven't yet worked with a traditional back-end server, but when we do, being able to apply AJAX will allow us to use much of the **DOM manipulation** we have already learned and to offer our users a smoother experience using our webpage.