Week 6: CRUD

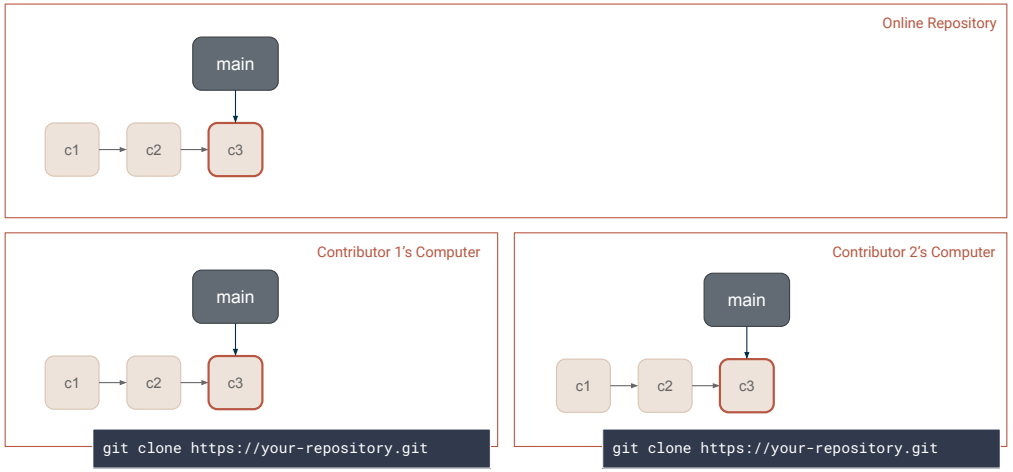
Agenda

- Questions?
- Potential Git Workflow
- Additional Resources
- Next Week: React

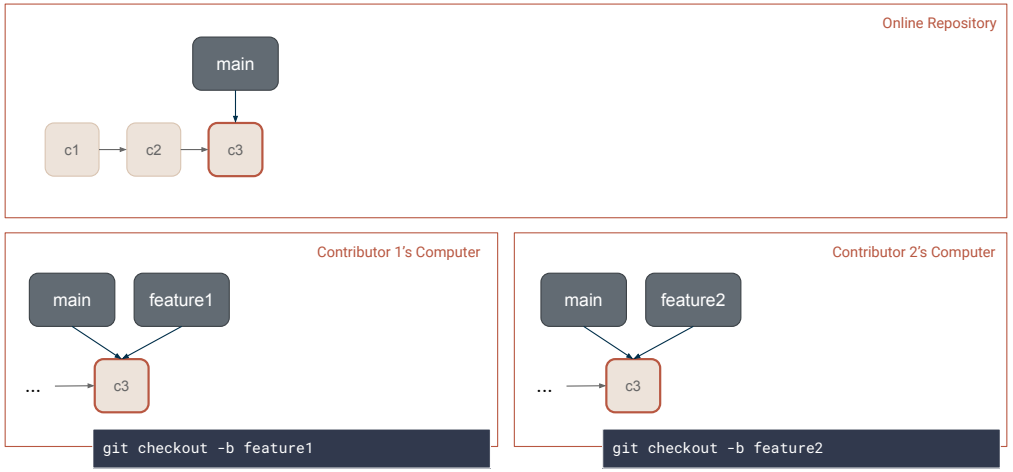
Using Git with Branches

Potential Git Workflow

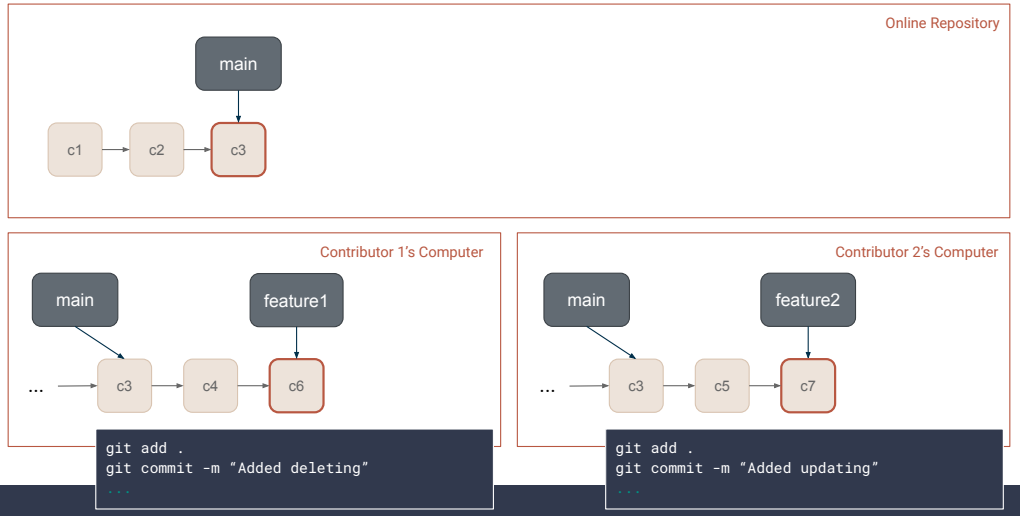
1. Clone the repository to your computer:
   `git clone https://your-repository.git`
2. Create a branch for the feature you're going to add (and name it after that feature):
   `git checkout -b my-branch-name`
3. Write your code!
4. At any time you can check the status of your branch:
   `git status`
5. While you're working on it, add and commit frequently whenever you get a piece working:
   `git add .`
   `git commit -m "Added something"`
5. **Optional:** You can push your branch to the online repository if you want it backed up:
   `git push origin my-branch-name`
6. When you're completely finished with that feature, merge it back into the main branch:
   `git checkout main`
   `git pull origin main`
   `git merge my-branch-name`
7. If you have any merge conflicts, you'll need to go to those files and fix them and then add and commit:
   `git add .`
   `git commit -m "Merged feature in"`
8. And push your newly merged code to the online repository
   `git push`



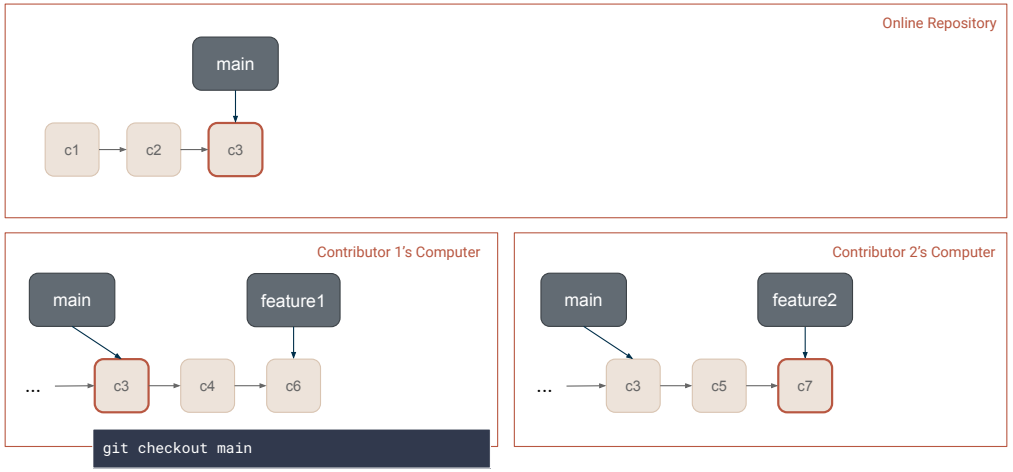
Potential Git Workflow



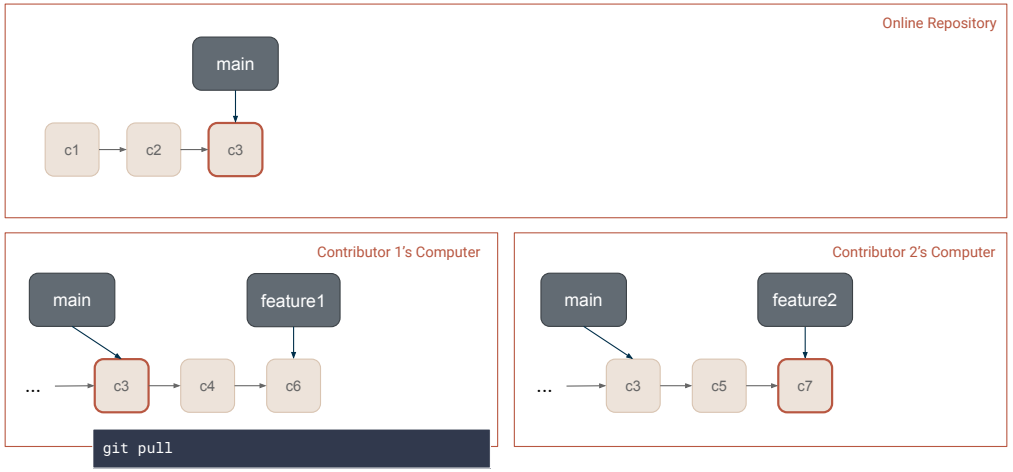
Potential Git Workflow



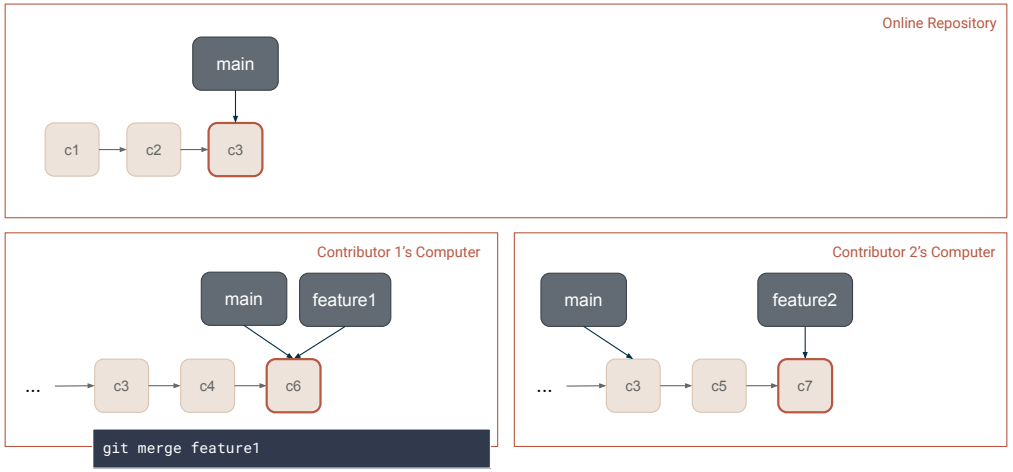
Potential Git Workflow



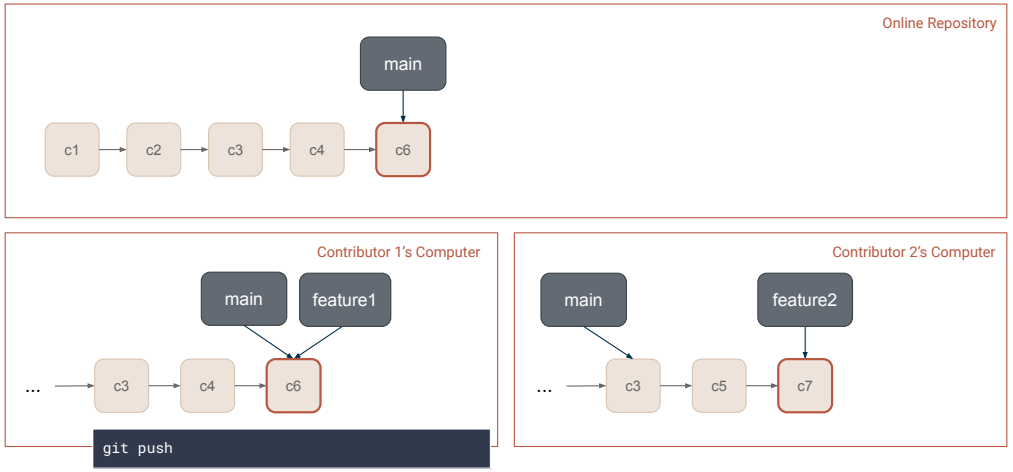
Potential Git Workflow



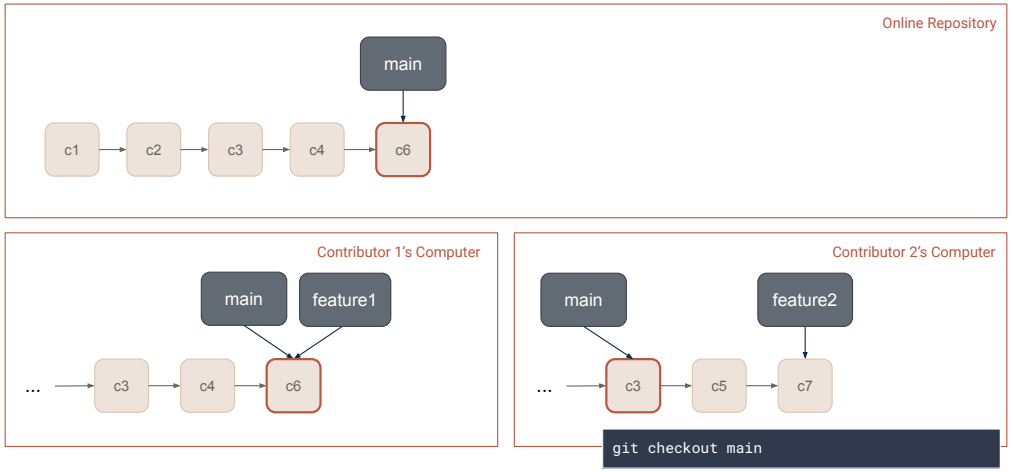
Potential Git Workflow



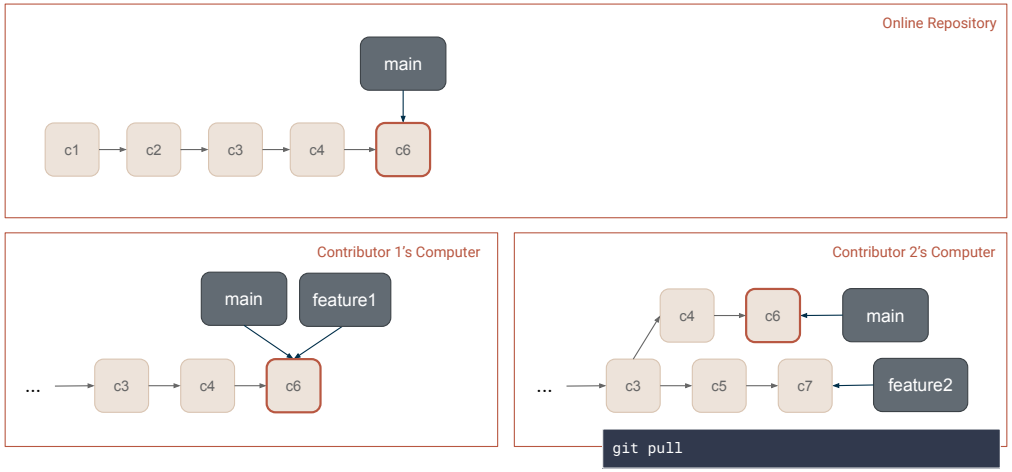
Potential Git Workflow



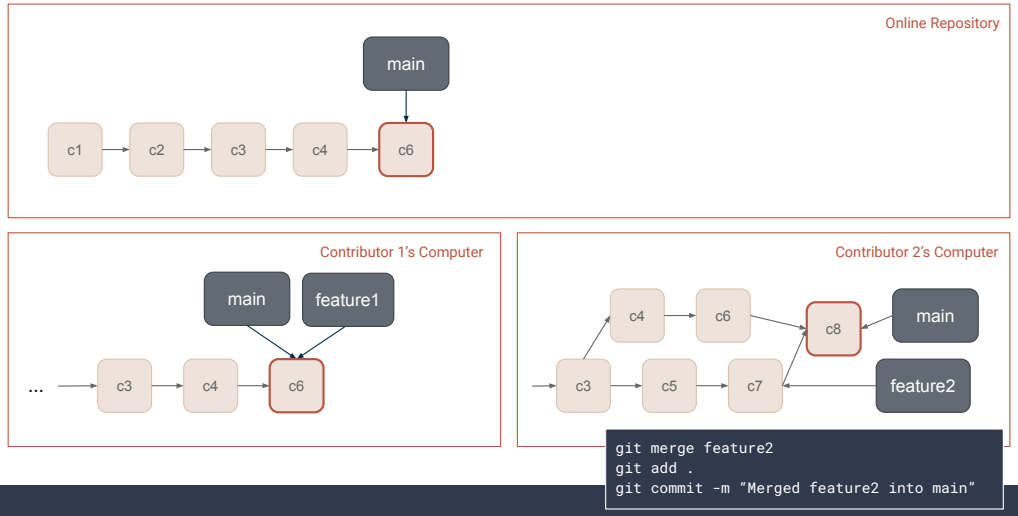
Potential Git Workflow



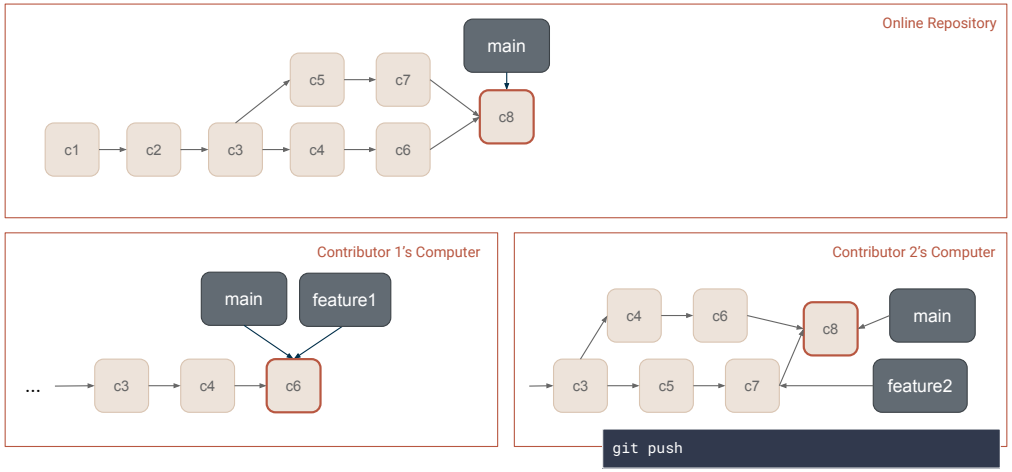
Potential Git Workflow



Potential Git Workflow

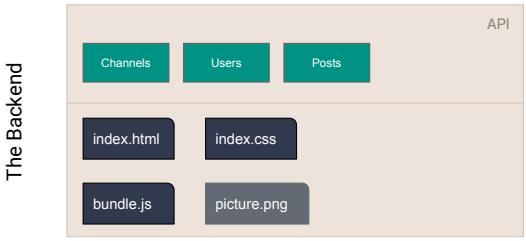


Potential Git Workflow

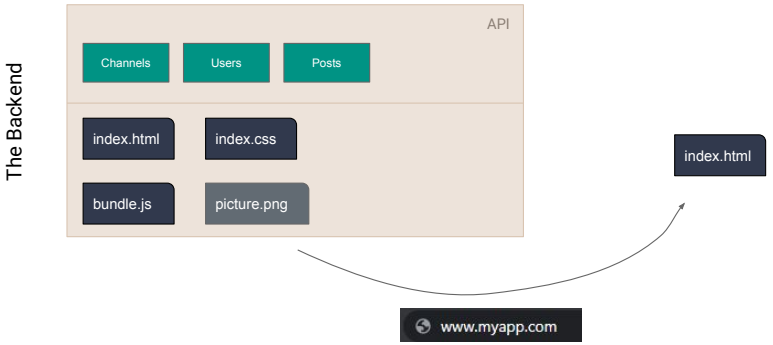


Potential Git Workflow

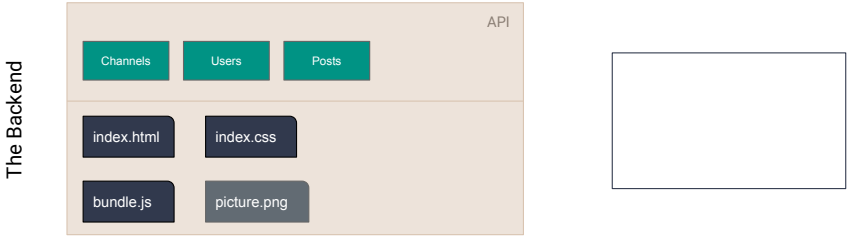
AJAX with a REST API



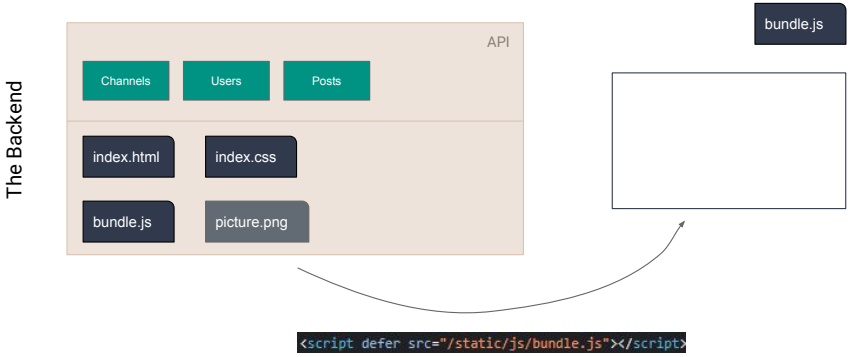
Single Page Application



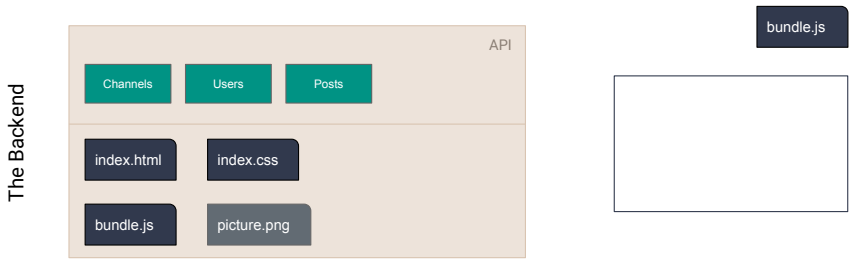
Single Page Application

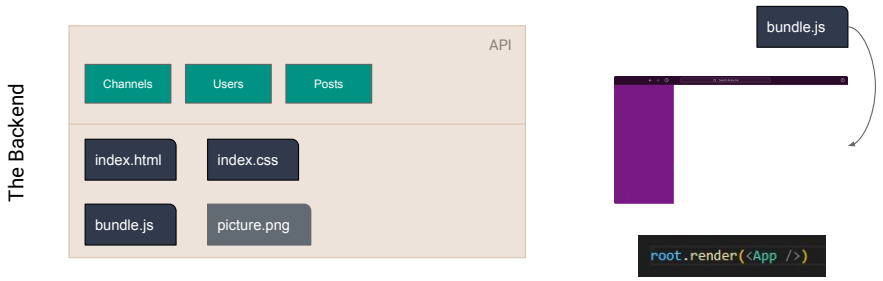


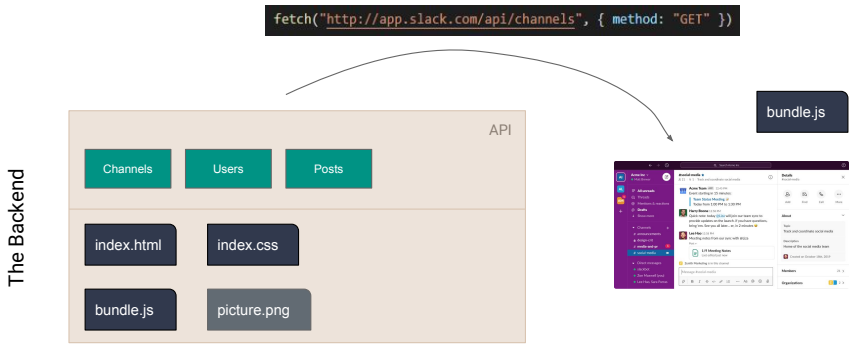
Single Page Application



Single Page Application







Simple Definitions

Asynchronous: code that can run in the background while other code is also running

HTTP: a way to communicate across the internet, used by browsers and fetch, short for Hypertext Transfer Protocol

API: a chunk of a backend that handles requests to get or change data, short for Application Programming Interface

REST: A very popular philosophy for how to build a backend API, short for Representational State Transfer

JSON: a syntax for data, based on Javascript's objects and arrays, short for Javascript Object Notation

AJAX: a way to make a HTTP request (to an API) asynchronously with Javascript, short for Asynchronous Javascript and XML (which is a dumb name because nobody uses XML anymore, we use JSON)

CORS: communicating with a backend at a different location on the internet, short for Cross-Origin Resource Sharing

Metaphor

Instead of making an Asynchronous HTTP request using AJAX to a REST API that accepts CORS requests and getting a response with JSON...

You're multitasking by texting under the table to your mom (respectfully) in another state and she sends you internet slang back because she's cool like that.

Disclaimer: It's not a perfect metaphor, but I still like it

Asynchronous = multitasking

HTTP = texting

API = your mom

REST = how you should talk to your mom

JSON: internet slang

AJAX: texting under the table

CORS: talking to someone in another state

REST & Fetch

Endpoint: `"http://something.com/api/unicorns"`

Action	Method	Fetch
Create	POST	`fetch("http://something.com/api/unicorns", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify(unicornToCreate) })`
Read	GET	`fetch("http://something.com/api/unicorns") fetch("http://something.com/api/unicorns/1234")`
Update	PUT	`fetch("http://something.com/api/unicorns/1234", { method: "PUT", headers: { "Content-Type": "application/json" }, body: JSON.stringify(updatedUnicornData) })`
Delete	DELETE	`fetch("http://something.com/api/unicorns/1234", { method: "DELETE" })`

Async & Await

Await

Tells Javascript you want to wait for the function to fully finish and give the result, not a Promise

Oversimplified:

`await` = I'll wait

Example:

```
const response = await fetchData()
```

Async

Tells Javascript this function takes a long time and you probably don't want to wait for the result

Oversimplified:

`aync` = It's gonna to be a awhile

Example:

```
const fetchData = async () => {
 // returned value is put in a Promise
}
```

Steps of Working with Fetch

1. Make the request and get a response. You'll need to give fetch the right parameters for the type of request you're making.

```
const response = await fetch('whatever')
```

2. Optional: Handle any errors. You can check the response to see if the server had an issue. You can also wrap everything in a try-catch to handle if there's an issue sending the request (like lost internet).

```
if(!response.ok) {
    // handle that it didn't work
}
```

3. Optional: Parse the data from the response. Not all responses will have data. For example, the response from a delete request often doesn't have any data. Sometimes the response has data but you don't need it, and then you wouldn't need to parse it. Once you parse the data, you may need to get certain properties on it or process it.

```
const data = await response.json()
```

Additional Resources

Git and Github Workflow for Beginners

https://www.codingmadeclear.com/git-github-workflow-for-beginners/

Git and Github Cheatsheet for Beginners

https://www.codingmadeclear.com/git-github-cheat-sheet-for-beginners/

Git: The Simple Guide

https://rogerdudler.github.io/git-guide/

Git & Github for Beginners

https://www.youtube.com/watch?v=SWYqp7iY_Tc

Merging a Branch Into Main

https://stackabuse.com/git-merge-branch-into-master