COMPSCI 326 - Web Programming
CRUD

join on the Slack #q-and-a channel as well as Zoom
Remember, you can ask questions of your teammates on your group Slack!

please **turn on your webcam** if you can

mute at all times *when you aren't asking a question*
(https://docs.google.com/document/d/1yNqVyX8DmsCmlyCIsdScb7VGw5UxCaZRXjRgMaDQF5o/edit?usp=sharing)

Today: CRUD (Server-Side)

Last time: fetch + REST APIs
Today: implementing the server side of APIs, CRUD, ACID

Your web app is going to interact with a server to manage state. To do this, you'll use fetch on the browser to talk to the server, which will provide a REST API interface.

REST APIs

As mentioned before, there are lots of free REST APIs out there.

These APIs are typically "RESTful". REST = "Representational state transfer". This means:

- There is some base URL, like https://api.github.com/
- You access it over HTTP via methods we've discussed, like GET and POST
- You can use this to read data, either through special URLs, or via GET or POST, and to create / update data (also via POST, but can be others)

Examples:

(these are referred to as **endpoints**)

- https://api.github.com/repos/jvilk/browserfs
  - jvilk = username
  - browserfs = repository
- https://api.github.com/repos/jvilk/browserfs/stargazers
  - lists the people who have "starred" the repo

Notice response is in JSON.

What do we need to do to implement an API?

- The server must first parse the URL (more on that later)
- The server needs a way to implement the basic operations of the API

CRUD

The basic operations that RESTful APIs implement generally fall into four categories, with the acronym CRUD:

Create - make a new thing
Read - read its current value
Update - update its value
Delete - delete the thing

For example, pretend we want to provide an API for the Registry of Motor Vehicles (=DMV for non-Mass residents).

We might want to be able to create a new driver's license for Juan Tanamera:

`/create/Juan+Tanamera/state/MA/age/20/certified/passenger`

We can update his certification:

`/update/Juan+Tanamera/certified/trucks`

Read his age:

`/read/Juan+Tanamera/age`

And, if Juan moves to another state, delete his record:

`/delete/Juan+Tanamera`

We could just maintain this in memory as an object, and update it as needed:

```
const info = {};

function create(name, certification, age) {
   info[name] = { 'certified' : certification, 'age' : age };
}
```

etc.

- This is easy, but it's not ideal...

Transient vs. Persistent State

This is not a great strategy because if the server reboots for any reason (e.g., software upgrade, power outage, etc.), all of the data will be lost! That is, the state is **transient**.

Persistence in the browser

One alternative is to do what you did with `localStorage` for the game state: on every change, write everything to `localStorage`. When you reboot, load everything from `localStorage`. In combination with other browser storage systems like `cookies` and `sessionStorage`, this is how you make the state **persistent** on the client.

However, as we discussed before (e.g. news site example, famed meal plan case, etc.) nothing on the browser is safe from user tampering. Thus, this methodology is not sufficient for most applications in reality.

Persistence on the server

You can do the same thing on the server side, just by storing something into a file (e.g., writing and reading a `.json` file). (This is what you will do for the exercise & homework this week.)

This *does* provide persistence.

BUT is not a great strategy for three reasons:

1. writing a file is kind of low-level (you have to convert back and forth between the internal representation and JSON, which is slow and clunky)
2. you can't easily control permissions (e.g., you might want only certain people to access certain records, like their own user names)

3. it could be super slow to have to write *the whole state* (potentially enormous) when *very little has actually changed*
    a. imagine you have a megabyte of state, and everytime you change a few characters, you have to write a megabyte to a file -- this will slow things down **a lot**
4. it doesn't have a lot of other good properties we really want
    a. consider what happens if you have two browsers connected to the same server, so there are concurrent updates being made to the file
        i. *at best*, it's easy for one set of changes to get lost (because the whole file is overwritten)

Exercise:

COMPSCI 326 F20 - 13. CRUD Exercise