Using JS

Only experience can teach

- All the options available
- How to break down a problem

But…

- Some best practices can save you a lot of time

State

An application has "state"

- Frontend/backend different + separate states
- The current values for all things that can change

A chat application

- Are you logged in?
- As who?
- Are there messages?
- What are they?
- Are you typing a message?

How to store state

Store your state in variables/object

- These live only until a new page load
- Use these to update the screen as needed

DO NOT read the HTML (DOM) to recapture the state

- Industry did that
- Apps worked great…until they got big enough
- Then changes got too complex (too coupled)

How NOT to store state

Example: You show a list of users on the screen.

To get the list of users, should you read the DOM?

No. Why?

- The screen is the visual output
- Alter the display = change how to read DOM for list
- As display gets complex , so does state interaction
    - Does not "scale"
    - New changes = increasingly difficult

Model-View-Controller (MVC)

We do this frontend as well as backend!

- Frameworks exist, but we look at concepts

MVC is common best-practice pattern:

- Something manages your data (model)
- Something the flow of the application (controller)
- Something translates the data to output (view)

We will change a lot, but that breakdown will remain

Client side storage

Sometimes you want to store information outside of the page *on the browser*

- Cookies
- localStorage
- IndexedDB

BE CAREFUL

- Limited security
- Users will change browsers/machines
- Can get changed/deleted by user/browser
- Not all clients are browsers

Cookies

"Cookies" are just an HTTP header

- Special is how browsers treat them
- Browser sends cookies along with each request

Cookies are text-based key/values pairs

- limited to a URL and descendant paths
- might have expiration date
- might (should) require HTTPS
- might not be accessible to JS
- shared between tabs

When to use cookies

Most Common:

- Store a random key that is IS also server-side
    - a "session" identifier
- request with key lets server read extra data
- Depends on that random number staying secret
- Cookie/session should NOT hold **application** state
    - because user might be using multiple tabs
    - each page/tab has its own application state
    - session data is useful regardless of state

When not to use cookies

DO NOT use cookies to store:

- Sensitive data (CC numbers, passwords)
- Personal data (addresses, etc)
- Application state
- Big data
- Data hard to represent in short bits of text

Local Storage

localStorage and sessionStorage

- key/value
- client-side only (not sent to server)
- JS only (no JS, no using localStorage)
- Store bigger values than cookies
- localStorage is shared between tabs
    - sessionStorage is NOT
- localStorage does not expire
    - sessionStorage lasts until browser quits
- Still domain-limited
    - Not path limited

When to use localStorage

- Store JS-applicable preferences
- When data too awkward for cookies
- When user switching devices isn't a problem
- To keep tabs in sync with choices

Rarely want sessionStorage

- Lack of tab-sharing causes confusion

When NOT to use localStorage

- Cookie security restrictions still apply
  - Sensitive data (CC numbers, passwords)
  - Personal data (addresses, etc)
- If the data is needed without JS

IndexedDB

Browser-side object-based DB

- NOT relational, NOT table-based

Asynchronous

- Like a click handler: response will happen later

JS-only

Stores larger data, non-expiring

- Browser can limit and/or delete without warning

When to use IndexedDB

Fairly few cases

Transactions

Larger data, but unreliable storage

Non-trivial to use

When NOT to use IndexedDB

- Cookie security restrictions still apply
    - Sensitive data (CC numbers, passwords)
    - Personal data (addresses, etc)
- If the data is needed without JS
- If you don't want the complexity

Client-side Storage for Assignments

- Cookies for session id only
- No localStorage/sessionStorage
- No IndexedDB

What is a Polyfill?

Polyfills add newer functionality to older JS

Example:

- `forEach()` is a method on Arrays
- takes a callback, calls that callback with each element in turn

You can write this in JS versions prior to it being standard

How do Polyfills work?

- Check to see if the feature exists
- If not, add the new function to the prototype

Why all methods in MDN refer to `Foo.prototype.`

- The **only** time you modify native prototypes
- Someone else has done this for you

JS Tools

JS ecosystem has many tools beyond the engine

- linters
- minifiers
- bundlers
- transpilers

Linters

Linters (not JS-specific): programs to check syntax

- For purely stylistic preferences
- For patterns that are technically correct
    - but tend to lead to errors

Formatting is long debated

Linters can help find unintended errors

`eslint` is the most common JS linter

- Many IDEs have linting built-in

Prettier

- Newer tool (JS only?)
- Auto-formats code to a common style
- Popular among those that don't want to argue

Minifiers

- Removes unneeded whitespace
- Replaces variable names with short ones
    - where possible

Reduces file size of JS/CSS/HTML

Makes them harder to read/debug

Is NOT security

Smaller size CAN matter

Bundlers

Frontend struggles to handle multiple JS files well

"bundlers" convert multiple files into one

- Some use NodeJS `require()` syntax
- Others use the newer standard `import` command

Some common bundlers:

- Webpack
- Rollup
- Parcel

Browersify - an example bundler

```
// Commands
mkdir b-ify
cd b-ify
npm init -y
npm install browserify
```

```
// foo.js
const bar = require('./bar');
console.log(`The other file says ${ bar() } successfully`);
```

```
// bar.js
module.exports = function() {
  return `"I like cats"`;
};
```

```
// Commands
browserify foo.js -o bundle.js
```

```
// index.html
<script src="bundle.js"></script>
```

Transpilers

Transpilers are "**trans**forming com**pilers**"

- input (something)
- output JS

Examples:

- Input typescript, output JS
- Input clojurescript, output JS
- Input modern JS, output older JS
- Input **future** JS, output modern JS

Example: See Babel at **https://babeljs.io/**

Hot reloading

During Front end development, it is common to have a setup that will reload your changes easily

- great during development
- not great for when the product is shipped

In This Course

- We will start without tools
    - Your IDE might have linting
    - We will add some tools over time
- Tools make things easy
    - But understand the concepts without them
    - You aren't lost if they aren't working
    - And many tools aren't noticed
        - Others definitely ARE

BUT: You may think WebDev is annoying