Building a page in React

React gives us new tools, but also new questions

- What to put into a Component?
- What to keep in state?
- Which Component(s) have state?
- How to organize CSS?

Building the Outline

This just one way (my way) of starting:

- Create Project
- Create top level "shell" components
    - Just have their name

App.jsx

```
import './App.css';

import Header from './Header';
import MainArea from './MainArea';
import Footer from './Footer';

function App() {
  return (
    <div className="app">
      <Header/>
      <MainArea/>
      <Footer/>
    </div>
  );
}

export default App;
```

Components are named semantically

- Not after elements
- Some Components match semantic elements
    - Ex: `Button` is a common Component name
- Components names like semantic class names
    - Ex: `Card`, `Panel`, `RegisterForm`, `CatList`

Clean up the example code

- Add missing semicolons
    - REQUIRED FOR THIS COURSE
    - Teams each have their own conventions
- Replace the contents of `App.jsx`
- Remove any unneeded imports in `App.jsx`
- Decide if fragment or not
    - I switched to `div` with `className="app"`
- Delete or replace contents of `App.css`
- Delete or replace contents of `index.css`

Create the "stubs"

- A **stub** is a partial function/component
- Exists to let the code run
- NOT FINAL
    - But does let you see the code works so far

Creating the Stubs

```
// src/Header.jsx
function Header() {
  return (
    <header>
      Header
    </header>
  );
}

export default Header;
```

- Repeat for the MainArea and Footer stubs
  - in `MainArea.jsx` and `Footer.jsx` files
  - using `<main>` and `<footer>` as base elements
  - using `MainArea` and `Footer` as text
- Can now see generated page

Why `<MainArea>` and not `<Main>`?

- Vite creates a `src/main.jsx`
- `Main.jsx` and `main.jsx` confuse humans
    - Also systems that "pretend" case-sensitivity
        - Like OS X :(
- We could rename `main.jsx`
    - and change `index.html` to match
    - But `MainArea.jsx` is small change too
    - You decide
- My `<MainArea>` still returns `<main>`
    - Still uses `className="main"`

Examining the Header Stub

- Compare Header stub to generated HTML

```
  return (
    <header>
      Header
    </header>
  );
```

```
<div class="app">
  <header>Header</header>
```

- JSX removes whitespace at start/end!
- Raw HTML does not
- 99.99% of the time this is Good
    - Can format for humans reading the code

Expanding the Header

```
import GlobalNav from './GlobalNav';

function Header() {
  return (
    <header className="header">
      <img
        src="http://placecats.com/100/100"
        className="header__logo"
        alt="Feline Overlords Logo"
      />
      <h1 className="header__title">
        I welcome our Feline Overlords
      </h1>
      <GlobalNav/>
    </header>
  );
}

export default Header;
```

Explaining the Expansion

- I added the core elements of the header
- The nav was going to be:
    - A chunk of HTML
    - A "piece" of the website that might move
        - Ex: outside of the header?
    - So I made it a separate Component
        - A stub for now
- I filled in classes on elements

Follow the work/confirm/repeat cycle

- Lets you fix problems quickly
    - Easier to find what the problem is
    - Fewer bugs that hide if a fix works
    - Fewer things depend on the code you change
- Mistakes are INEVITABLE
    - I'm not a lousy programmer, this is normal
    - Better programmers detect and fix quicker
        - Not superhuman accuracy
- A tight, rapid work/confirm cycle takes practice
    - Impulse to get it out of your head, then check

When do you start CSS?

- After all HTML is generated?
- As you go?

This is where you say "it depends", isn't it?

Yes

- If HTML is known
    - Probably better to finish HTML
    - Break down into Components
    - Then apply CSS once HTML is done
- If deciding both HTML and CSS by experiment
    - Better to write CSS as you go
- A dash of personal preference

Adding CSS

A LOT of "it depends"

- How are you naming classes?
    - Ex: semantic? BEM? Utility?
        - Components give new options!
    - This course requires kebab semantic/BEM
- How are you breaking down files?
    - `src/index.css` (autoloaded by Vite)
    - `.css` files per Component?
        - Must manually import
        - How to organize them?

Example CSS decisions

- BEM styles
    - names similar to component name
- `index.css`
    - page-wide defaults (if any)
    - body/html
    - `#root`
    - `::root` custom properties
- `COMPONENT.css` (per component)
    - Ex: `App.css`, `Header.css`
    - CSS used only in that component

CSS Decisions are not simple!

- Breaking up CSS can make it easier to find
- Breaking up CSS can make it HARDER to find
- Easier to accidentally reuse class names!
    - BEM helps, but doesn't solve
    - Chaos when same class name used with different definitions
- Will Components be used outside of this project?
    - That's when styled-components, CSS modules, etc are used

My example: Components+CSS only in this project

Example `index.css`

```css
*, *::before, *::after {
  box-sizing: border-box;
}

html, body {
  height: 100%;
  margin: 0;

  background-color: #C0FFEE;

  font-family: sans-serif;
}

#root {
  height: 100%;
}
```

Revising Header

```css
/* Header.css - remember to import in Header.jsx */
.header {
  display: grid;
  grid-template-areas:
    "header-logo header-title"
    "header-nav header-nav"
  ;
  grid-template-columns: auto 1fr;

  background-color: skyblue;
}

.header__logo {
  grid-area: header-logo;
}

.header__title {
  grid-area: header-title;

  text-align: center;
}
```

When CSS spans components

- We haven't written `<GlobalNav>` yet
    - Except for a stub
- It's part of `<Header>`
    - How do we style it?

Lots of options

- Put HTML and CSS in Header instead?
- Class in GlobalNav, reference in Header.css?
- Define classes in both GlobalNav AND Header?

Evaluating CSS Options

- Easy to get overwhelmed
    - This is a result of CSS choices
    - Ex: If our CSS wasn't tied to components
        - No problem with overlap!
- No best answer
    - "It depends" on so much!
- Can be bad answers though
    - All HTML and CSS in one file is poor choice

Option: Passing Class

- Header will set a `className` prop on `<GlobalNav>`
- `<GlobalNav>` will use passed prop
    - AND set its own class name
- Each "layer" of component
    - Manages the CSS of that layer
- Not a perfect option
    - There is no perfect option

Updating `Header.jsx`

```
function Header() {
  return (
    <header className="header">
      <img
        src="http://placecats.com/100/100"
        className="header__logo"
        alt="Feline Overlords Logo"
      />
      <h1 className="header__title">
        I welcome our Feline Overlords
      </h1>
      <GlobalNav className="header__nav"/> /* +className */
    </header>
  );
}
```

Updated `Header.css`

```css
.header {
  display: grid;
  grid-template-areas:
    "header-logo header-title"
    "header-nav header-nav"
  ;
  grid-template-columns: auto 1fr;

  background-color: skyblue;
}

/* other css here */

.header__nav {
  grid-area: header-nav;

  background-color: lime; /* to test */
}
```

Nothing Happened!

- Check generated HTML
    - No `header__nav` class!
- Components aren't elements!
    - `<GlobalNav>` *outputs* a `<nav>`
        - `<GlobalNav>` is NOT a `<nav>`
    - We need to use the passed prop
    - On an HTML element

Updating `<GlobalNav>` with `className` prop

```
function GlobalNav({ className }) {
  return (
    <nav className={className}>
      GlobalNav
    </nav>
  );
}
```

- Passed `className` now on `<nav>`
- But what if we also want our OWN class name?

Updating <GlobalNav> with own `className`

```
function GlobalNav({ className }) {
  return (
    <nav className={`global-nav ${className}`}>
      GlobalNav
    </nav>
  );
}
```

- That is a lot of layers!
- `{}` to replace with a JS value
- `` `` `` to have a string with values in it
- `${}` to replace with a value in `` `` ``

Passing props to use in HTML is common

- Very common to pass props for direct use
    - `className`
    - `disabled`
    - Event handlers (`onClick`, etc)
- Sometimes we add to them
    - Or "wrap" them
        - To add/alter behavior
    - Before use on an element

Expanding GlobalNav

- Fill in HTML
- Add CSS
- Functionality?

Updated `GlobalNav.jsx`

```
return (
  <nav className={`global-nav ${className}`}>
    <ul className="global-nav__list">
      <li className="global-nav__item">
        <a className="global-nav__link" href="/">
          Home
        </a>
      </li>
      <li className="global-nav__item">
        <a className="global-nav__link" href="/about.html">
          About
        </a>
      </li>
      <li className="global-nav__item">
        <a className="global-nav__link" href="/cats.html">
          Cats
        </a>
      </li>
    </ul>
  </nav>
);
```

Computers are good at repetitive and detailed

- Writing this HTML is repetitive and detailed
    - Can we make the computer do more?

Define the Data

```
const menu = [
  {
    name: "Home",
    path: "/",
  },
  {
    name: "About",
    path: "/about.html",
  },
  {
    name: "Cats",
    path: "/cats.html",
  },
];

function GlobalNav({ className }) {
```

Move Repetition to a loop

```
function GlobalNav({ className }) {
  const list = menu.map( item => {
    return (
      <li className="global-nav__item">
        <a className="global-nav__link" href={item.path}>
          {item.name}
        </a>
      </li>
    );
  });

  return (
    <nav className={`global-nav ${className}`}>
      <ul className="global-nav__list">
        { list }
      </ul>
    </nav>
  );
}
```

Why did we loop?

- Now it is easy to add/remove/change menu items
- Changing menu HTML is just one change
    - Will automatically apply to all items
    - No chance of missing one element

Are you watching the Browser Console?

- We have a warning about the `key` prop
  - Must set the `key` prop on items in an array
  - To a unique value

```
const list = menu.map( item => {
  return (
    <li className="global-nav__item" key={item.name}> /**/
      <a className="global-nav__link" href={item.path}>
        {item.name}
      </a>
    </li>
  );
});
```

- Resolve warnings ASAP!
- You have to watch for them

Where is the best place for the data?

- Right now we have `menu` defined in `GlobalNav.jsx`
    - Just kind of sitting there
    - Can't be reused (stuck in this file)
- In the JSX file is not a good place for data
    - Data is not Presentation
- Some Projects will have "set" data like this
- Other Projects will get data from service calls

Importing Data

- We will pull our data from a `.js` file
    - We are focusing on UI
- To move further as a Web Dev, learn services

```
// menu.js - Yes, .js, not .jsx
const menu = [
  // menu options here
];

export default menu;
```

```
// GlobalNav.jsx
import menu from './menu';
```

Create and import GlobalNav.css

```css
.global-nav__list {
  display: flex;
  flex-direction: row;
  justify-content: space-around;

  padding: 0;
  margin: 0;

  list-style: none;
}

.global-nav__link {
  display: inline-block;

  padding: 0.5rem 1rem;

  font-size: 1.3rem;
}

.global-nav__link:hover {
  background-color: dodgerblue;
  color: black;
}
```

Fill in Footer stub

- Give a class
- Add some content (lorem?)
    - With class names
- Add CSS

How do we have/change state?

- Ex: hamburger menu
- I won't show a complete hamburger menu
    - Too many students cut/paste/tweak
    - You need to learn the CONCEPTS
        - Handle completely different scenarios
            - Such as an accordion!
        - So don't Google/ChatGPT it either!
- But let's look at an INCOMPLETE version

Basic Plan

- Add a button to `<GlobalNav>`
    - Click to toggle showing menu
    - Requires **state** to track if showing
- React gives new option for "hiding"
    - Just don't output the HTML at all!
- We use dropdown for state examples
    - Not trying to convert JS version
    - You know the HTML/CSS already

Toggle State

- Anything that decides what to show is **state**
- Here only `<GlobalNav>` cares about this state
  - So created and used in `<GlobalNav>`
  - Else created in **nearest common ancestor**

```
const [showMenu, setShowMenu] = useState(false);
// Other stuff here
return (
  // more JSX

  <button
    onClick={ () => setShowMenu(!showMenu) }
  >Menu</button>

  // more JSX
);
```

Using State

- Plain JS without render/state loop:
    - Toggle class
    - Use class in CSS to show/hide element
- React:
    - Toggle state
    - Use state to output/not output HTML

```
const menuHtml = (
  <ul className="global-nav__list">
    { list }
  </ul>
);
{ showMenu && menuHtml }
```

Conditional Rendering

- A common pattern in React
    - Output/Not output HTML ("rendering")
    - Based on state (a "condition")
- CANNOT do `if()` INSIDE `{}`
    - Can do `&&`
    - Can do `? :`

```
{ condition && <SomeComponent/> }
{ condition ? <SomeComponent/> : <OtherComponent/> }
```

- Can do `if()` OUTSIDE of HTML-like JSX
- Can assign JSX to variables and output those

Conditional Rendering of Text

- Can use conditional logic in element contents
- Also in prop values

```
<button>{ showMenu ? "Close Menu" : "Open Menu"}</button>

<button aria-label={showMenu ? "Close Menu" : "Open Menu"}>
  ...
</button>

<button disabled={isValid}>Submit</button>
```

- Parts of these buttons (classes, types, etc) not shown to save space

Adaptive CSS in React

- `@media` queries in each `.css` as applicable
- If using this example
    - Each component has the adaptive differences
- React doesn't know your size by default!
    - Complex and usually not needed
- Use state for conditional rendering based on state
- Use CSS for styling changes based on viewport

Hamburger: Both apply

What to do when both state and `@media` involved?

- React State manages toggle
    - But toggles class, not existence
        - Change from the "normal" React way
- Styling changes `display:none`
    - Based on class and `@media` query

Otherwise it can break on resize

- How to hide by default when shrinking?
- How to add when hidden and expanding?