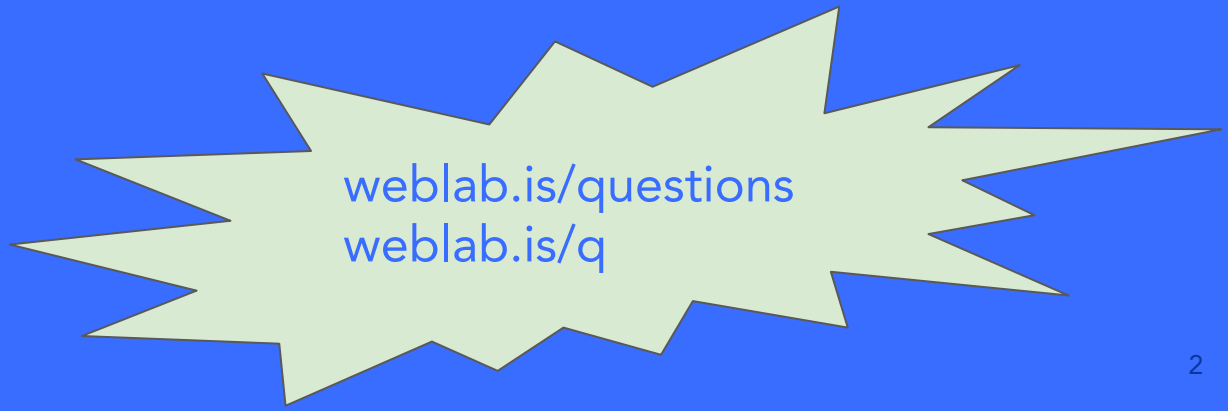
# Reminders 🧐💭

- Milestone 0 is due tonight!!! 🌕
- Recordings are up at weblab.is/recordings 📷!
  - Certain lectures too blurry to rewatch? Tell us on weblab.is/milkandcookies if you would benefit from re-recording 🥺📷
- As always,
  - weblab.is/q
  - weblab.is/questions
  - weblab.is/home

# Workshop 2: Catbook in React

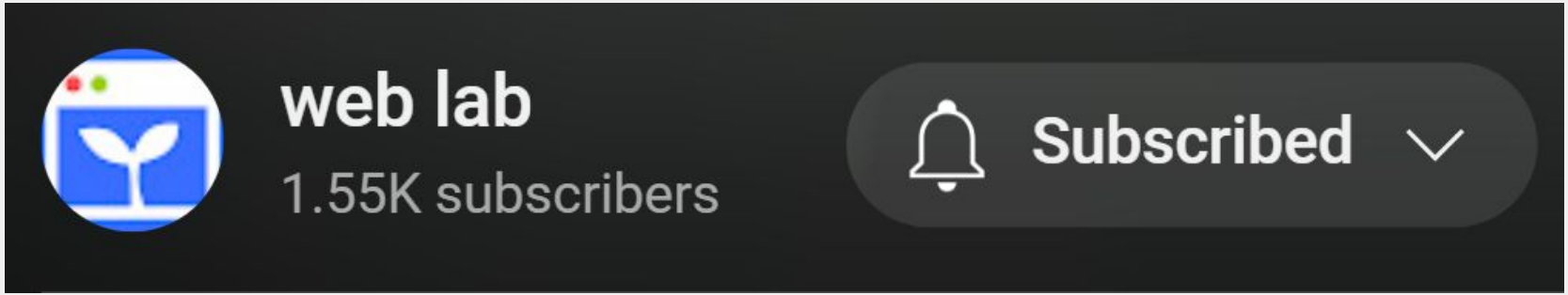
Enrique Casillas + Kenneth Choi



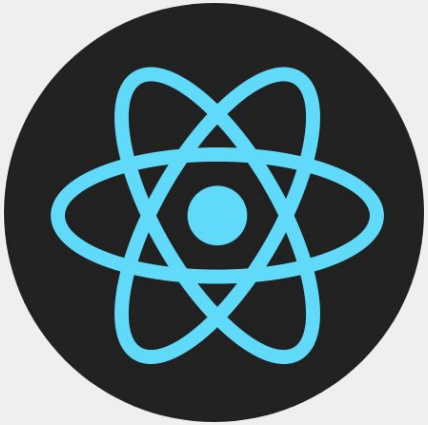
## Watch time from subscribers

Watch time · Since uploaded (lifetime)

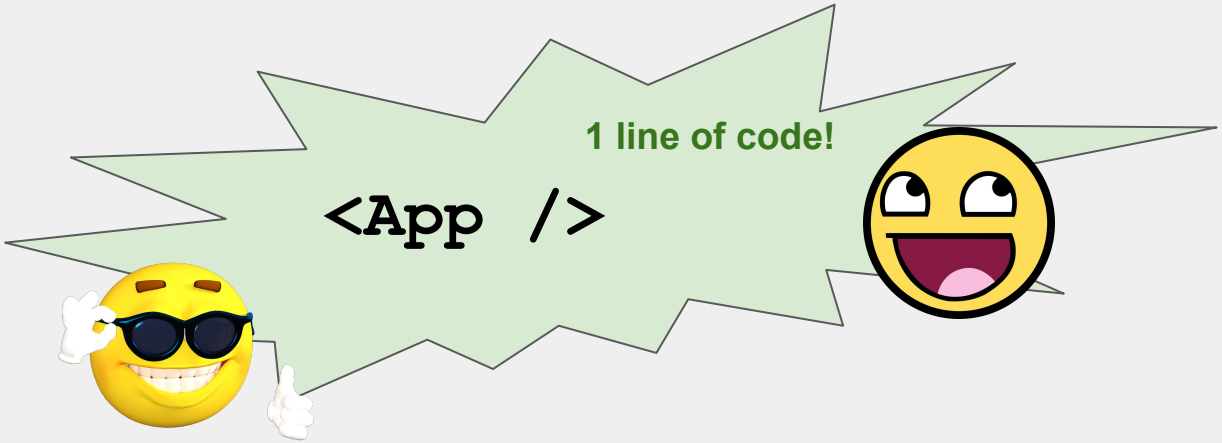
| | |
|---|---|
| Not subscribed | 85.2% |
| Subscribed | 14.8% |



# React Recap

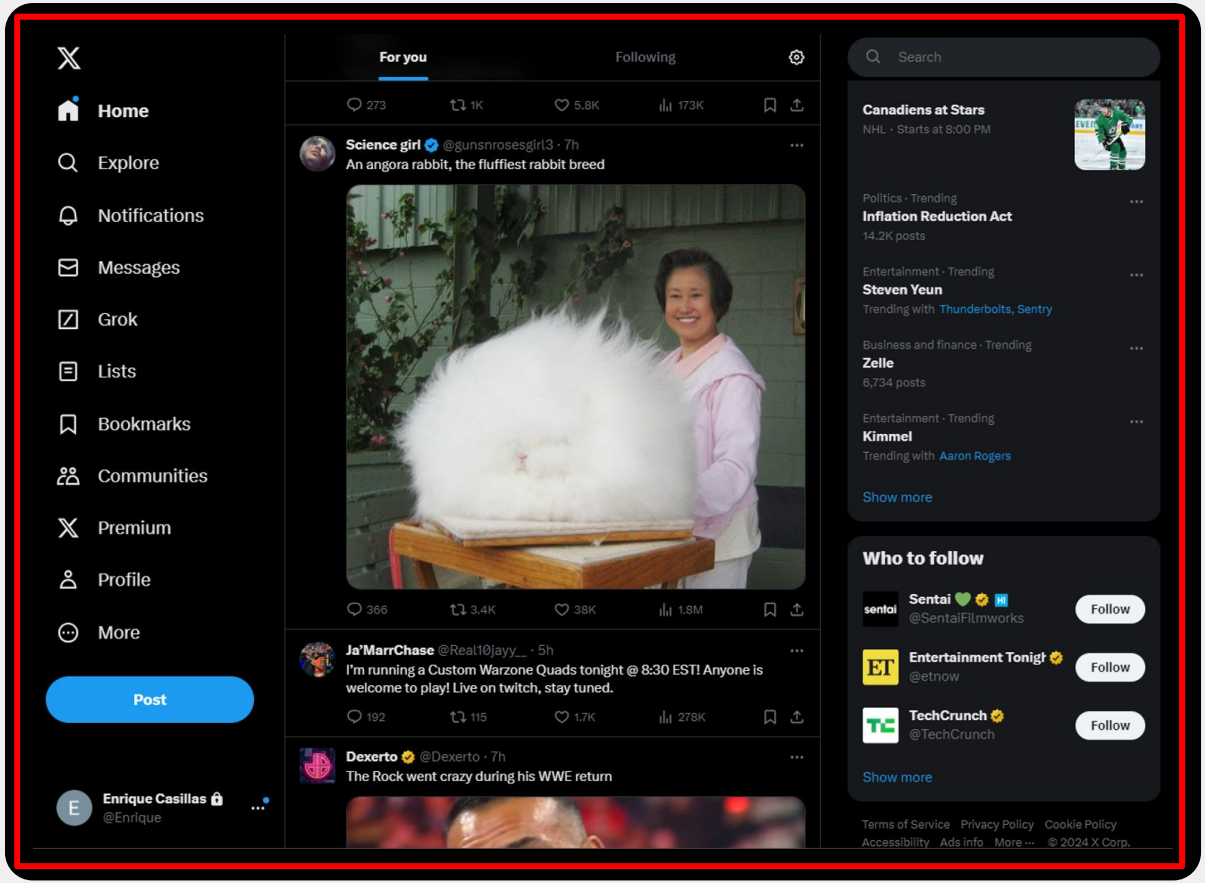


# How to write any website



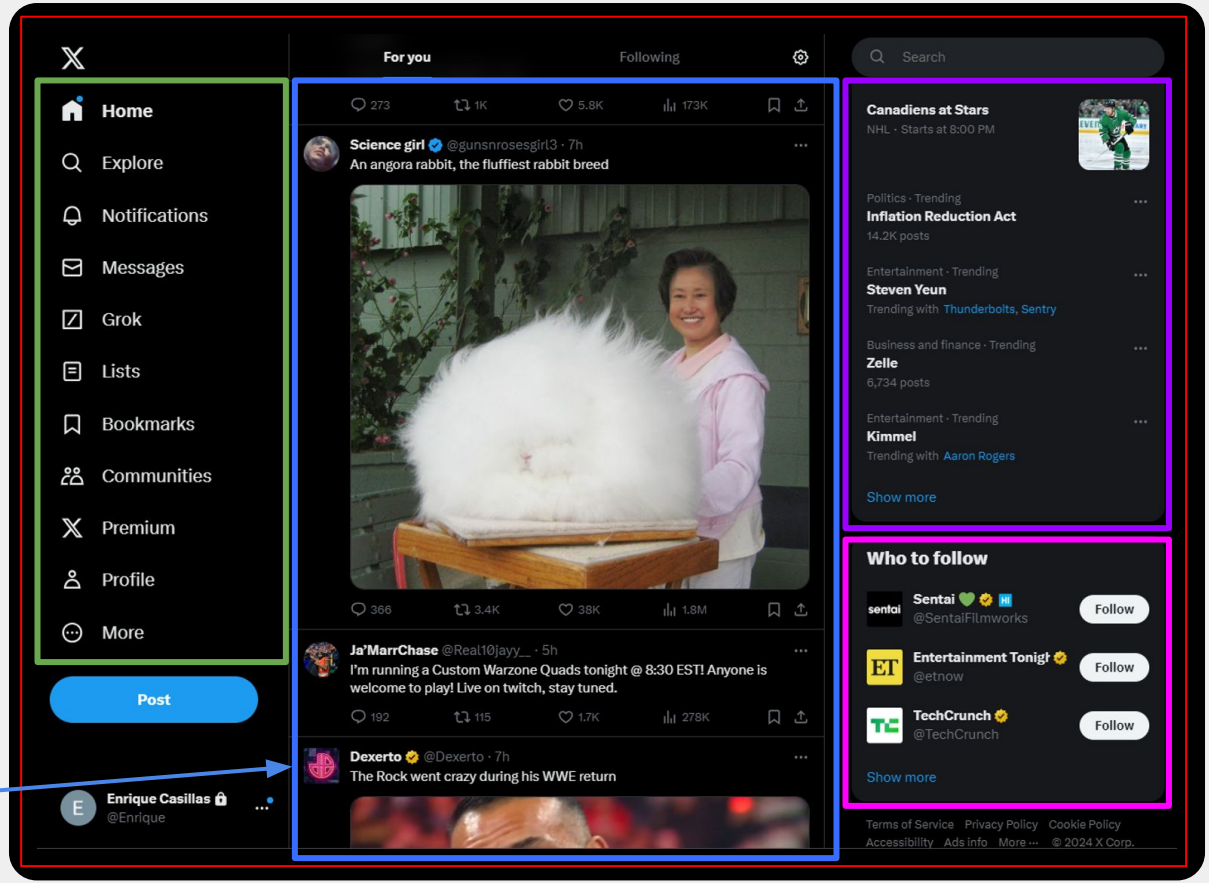
# Recap: Break down a website into components

The **root** component (**<App/>**)



# Recap: Break down a website into components

<Navbar/>

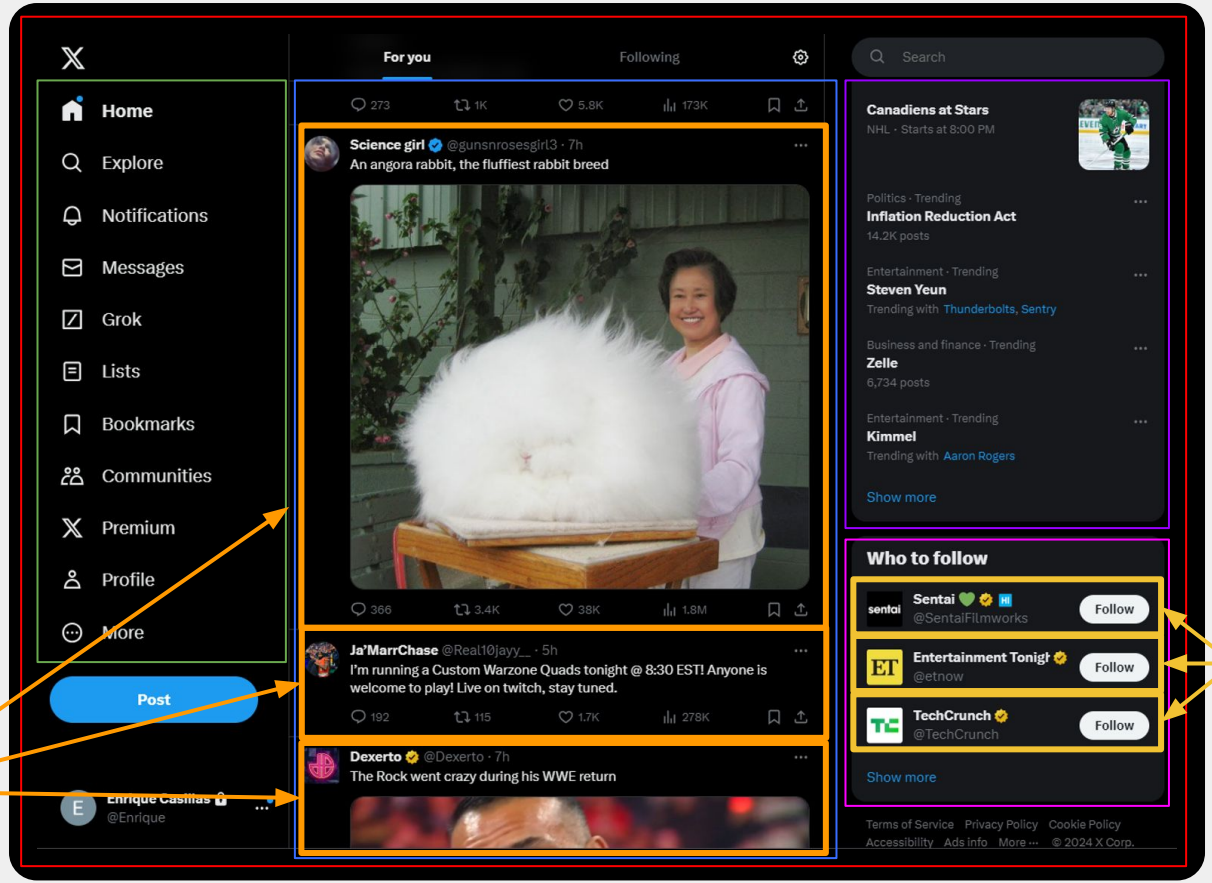



<Trending/>

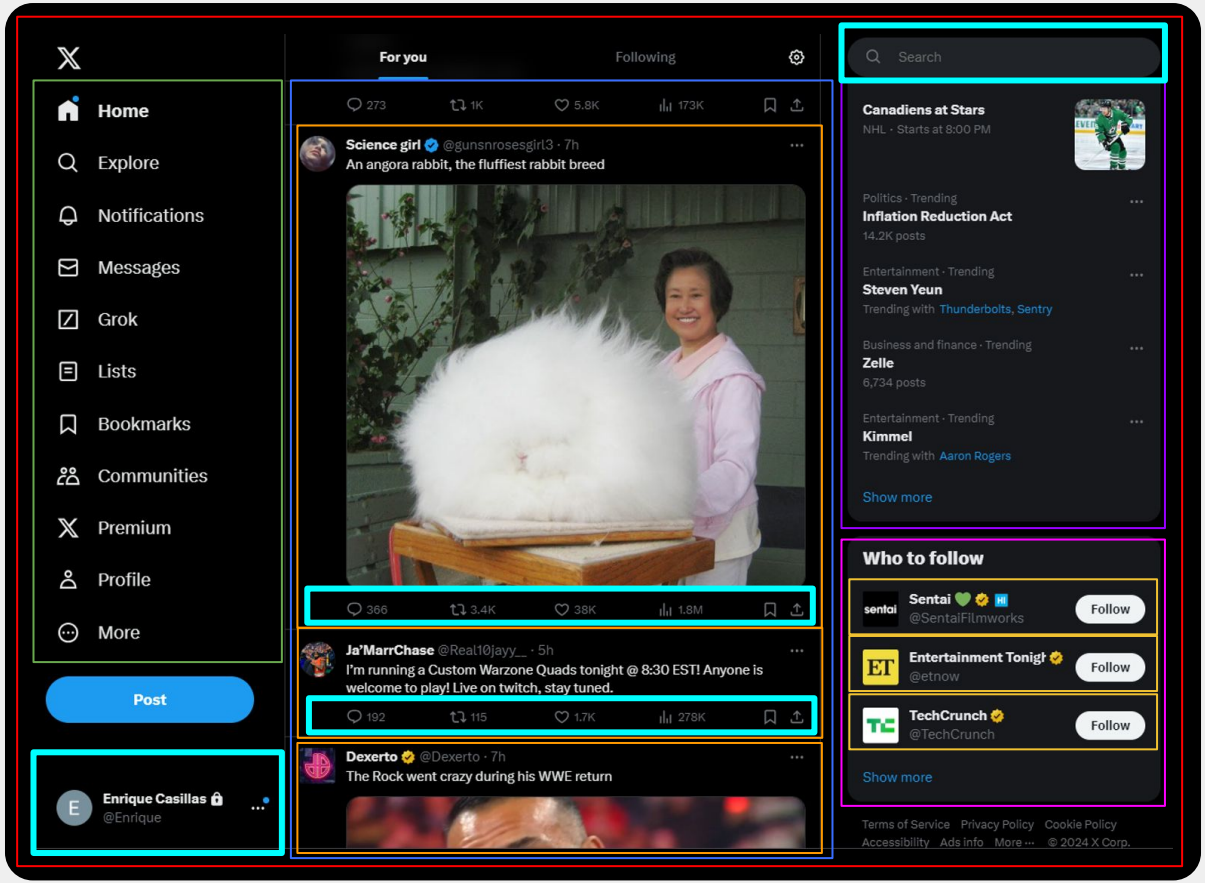
<Suggestions/>

<Feed/>

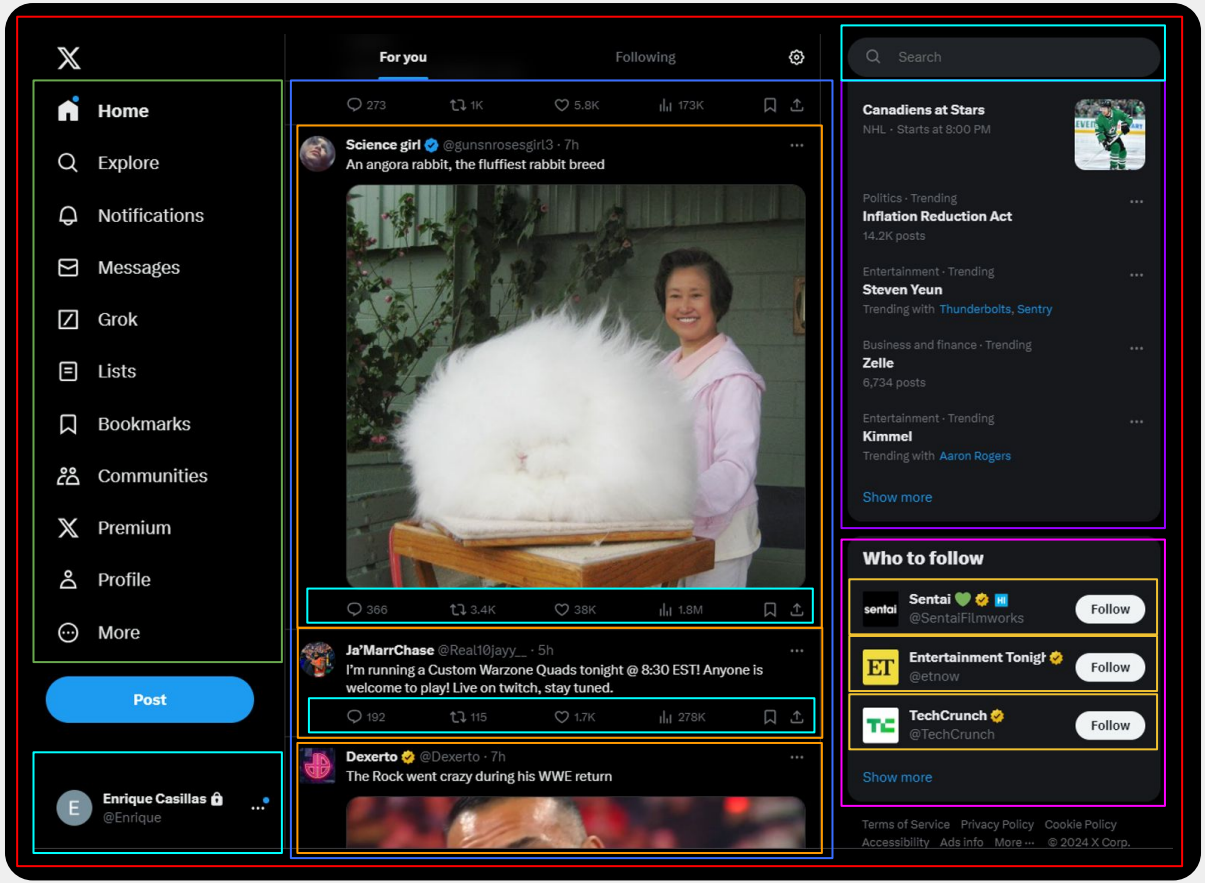
# Recap: Break down a website into components



# Recap: Break down a website into components

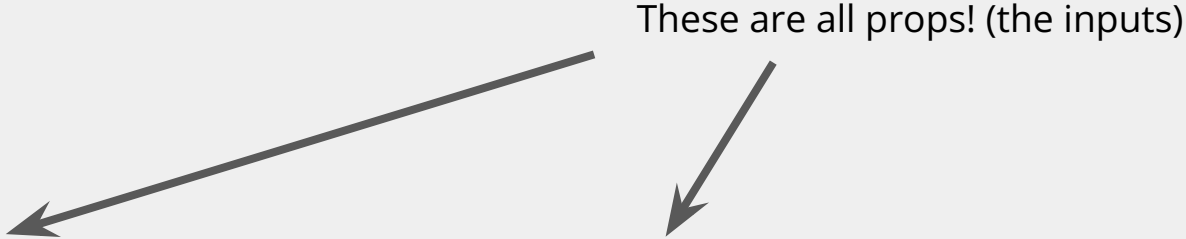


# Recap: Break down a website into components



# Recap: Props

## Information passed from a parent to a child component




```
<Post name="Kenneth" text="Welcome to web.lab!" />
```

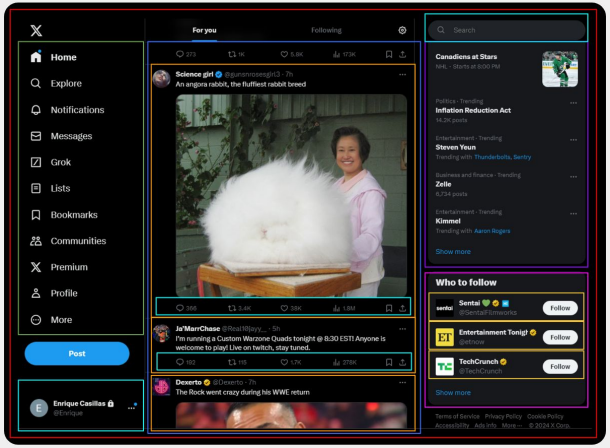
here, props = {name: "Kenneth", text: "Welcome to web.lab!"}

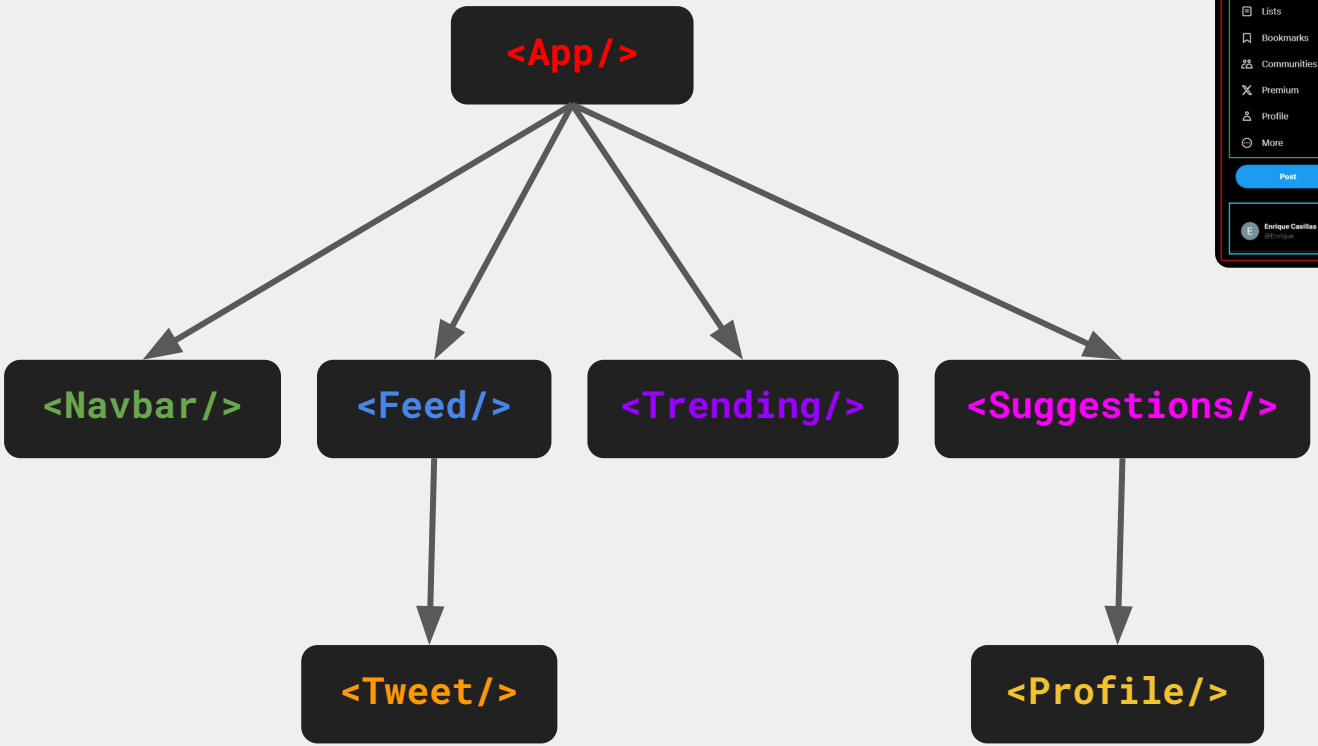
# Recap: State

**Updatable** pieces of information maintained by a component

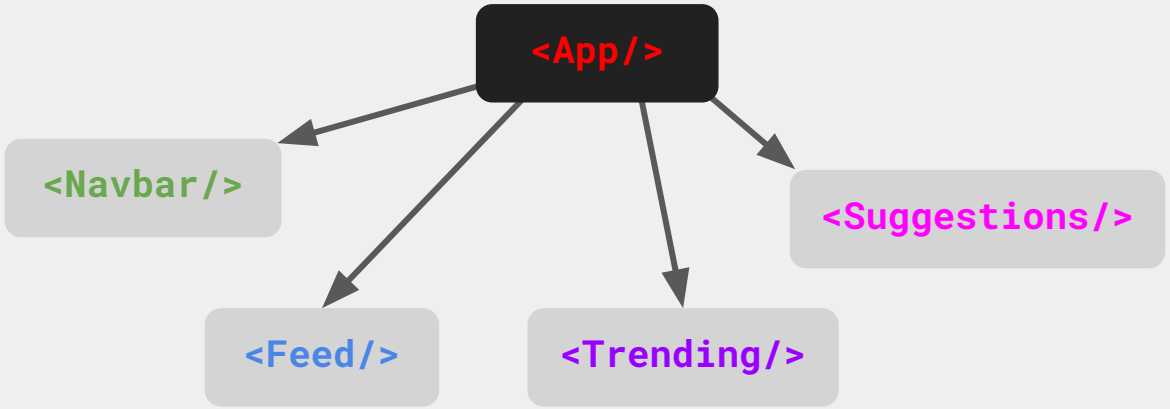
```
const [status, setStatus] = useState("busy");
const [isOnline, setIsOnline] = useState(false);
```

# Recap: Creating the component tree



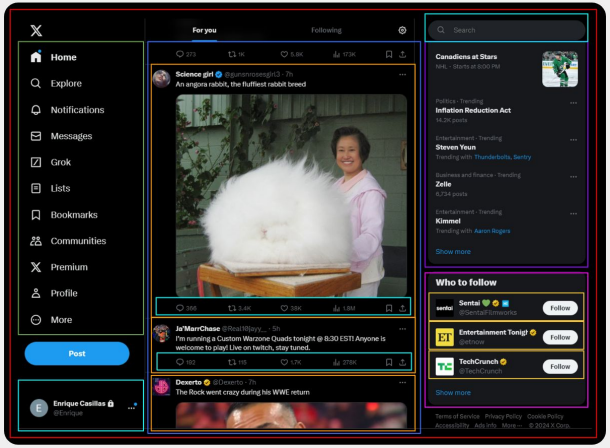



# Recap: Creating the component tree




State: None

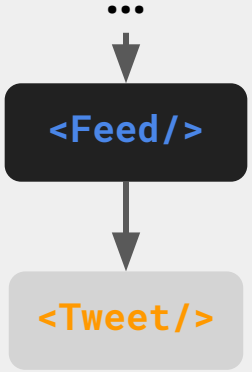
Props: None

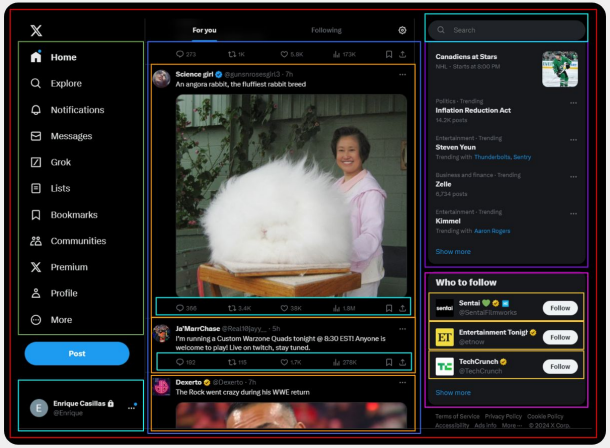



## App (App.js)

```
const App = () => {
  return (
    <div>
      <Navbar />
      <Feed />
      <div>
        <Trending />
        <Suggestions />
      </div>
    </div>
  )
}
```

# Recap: **Creating the component tree**



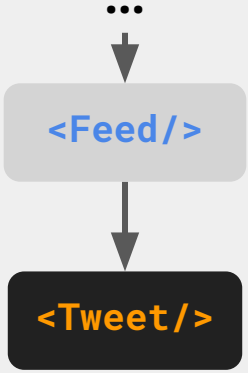

**Feed** (Feed.js)

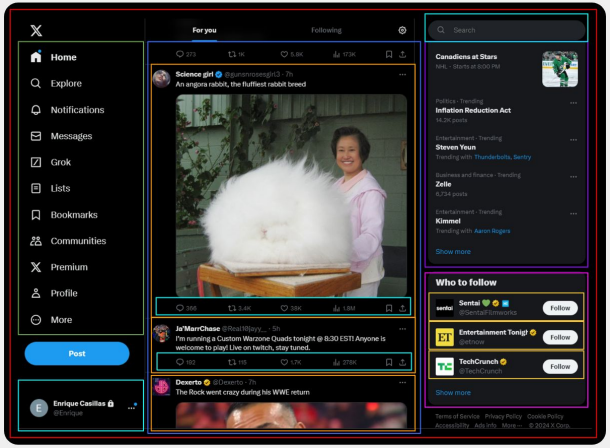
```
const Feed = () => {
  const [tweetData, setTweetData] = useState([]);

  const loadTweets= (() => {
     // Load tweets from API...
     setTweetData(loadedTweets);
  })();

  return (
    <div>
      <Tweet username={tweetData[0].username} ... />
      <Tweet username={tweetData[1].username} ... />
      <Tweet username={tweetData[2].username} ... />
       ...
    </div>
  )
}
```

State: All tweet data in the feed

Props: None

# Recap: **Creating the component tree**



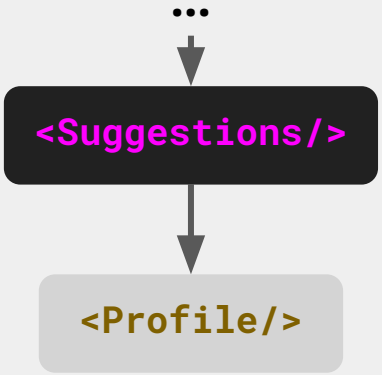

## **Tweet** (Tweet.js)

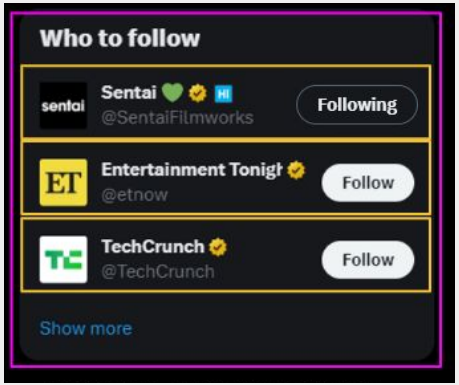
```
const Tweet = (props) => {
  return (
    <div>
      <div className="tweet-header">
        <img src={props.profilePicture} />
        <span>{props.username}</span>
        <span>{props.datePosted}</span>
      </div>
      <p>{props.content}</p>
      ... // Replies, retweet, etc.
    </div>
  )
}
```

State: None

Props: Individual tweet data

# Exercise: Creating Suggestions




**Suggestions** (Suggestions.js)

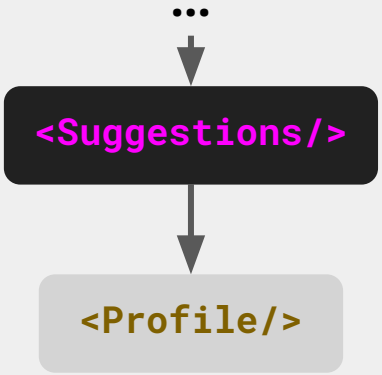
```
const Suggestions = (?) => {

  ???

  return (
    <div>

      ???

    </div>
  )
}
```

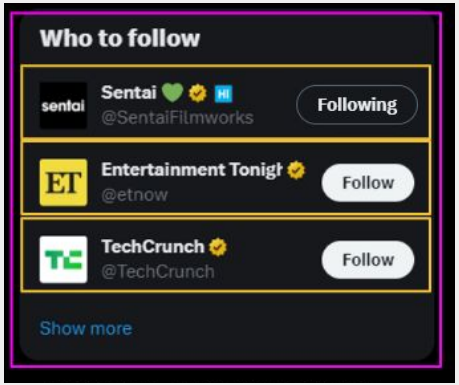
*Hint: Similar to Feed

State: ?

Props: ?

# Exercise: **Creating Suggestions**



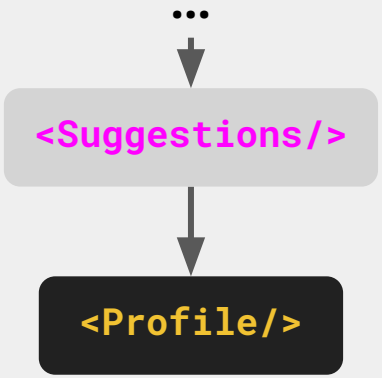

**Suggestions** (Suggestions.js)

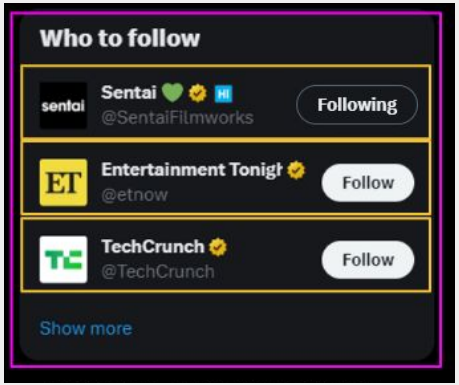
```
const Suggestions = () => {
  const [profileData, setProfileData] = useState([]);

  const loadSuggestions = (() => {
     // Load suggested profiles from API...
     setProfileData(loadedProfiles);
  })();

  return (
    <div>
      <Profile username={profileData[0].username} ... />
      <Profile username={profileData[1].username} ... />
      <Profile username={profileData[2].username} ... />
       ...
    </div>
  )
}
```

State: All post data in the feed

Props: None

# Exercise: **Creating Suggestions**



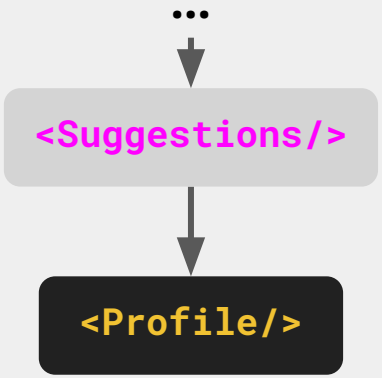

State: ?

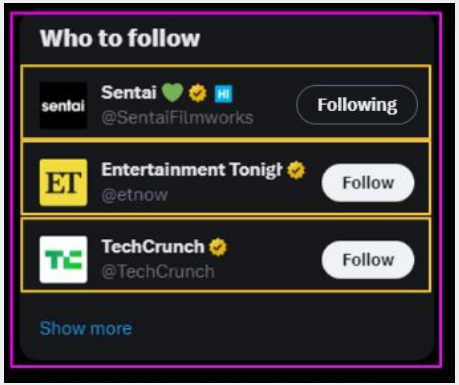
Props: ?

**Profile** (Profile.js)

```
const Profile = (?) => {
  ???

  return (
    <div>

      ???

    </div>
  )
}
```

# Exercise: **Creating Suggestions**




State: None

Props: Individual profile data

## **Profile** (Profile.js)

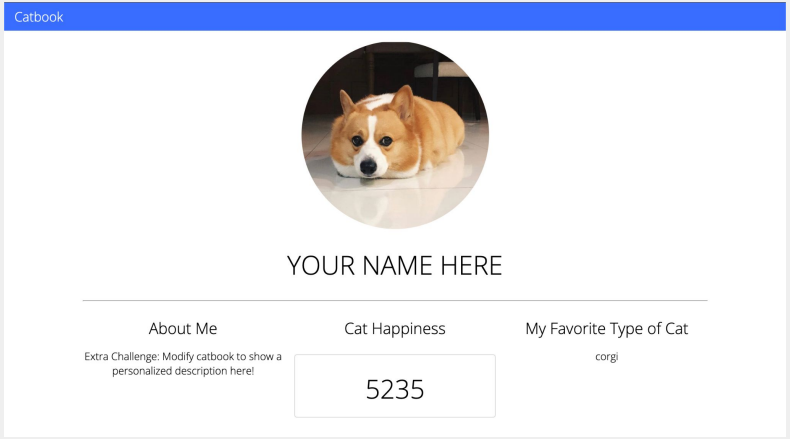
```
const Profile = (props) => {
  return (
    <div>
      <img src={props.profilePicture} />
      <div>
        <span>{props.username}</span>
        <span>{props.handle}</span>
      </div>
      <button>Follow</button>
    </div>
  )
}
```

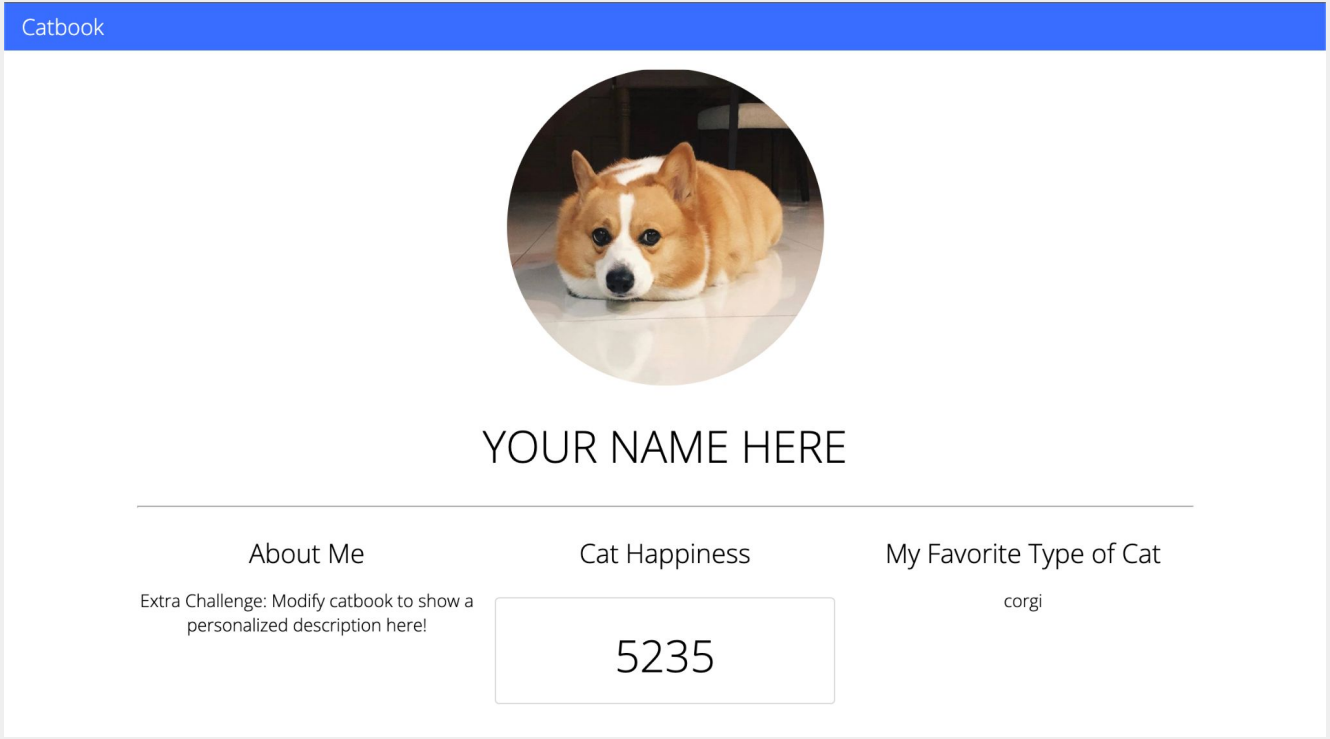
# Recap

- A **React component** lets you break down a chunk of your UI into a reusable and independent piece of code.
- A component can be represented as a piece of HTML code, other React components, or both.
- It can receive and maintain its own information
- React uses a **component tree structure** to pass information
- Each component can take in **props** (inputs), and manages its owned contained **state** (mutable data)

**Check out our recap guide at**
**http://weblab.is/react-guide**

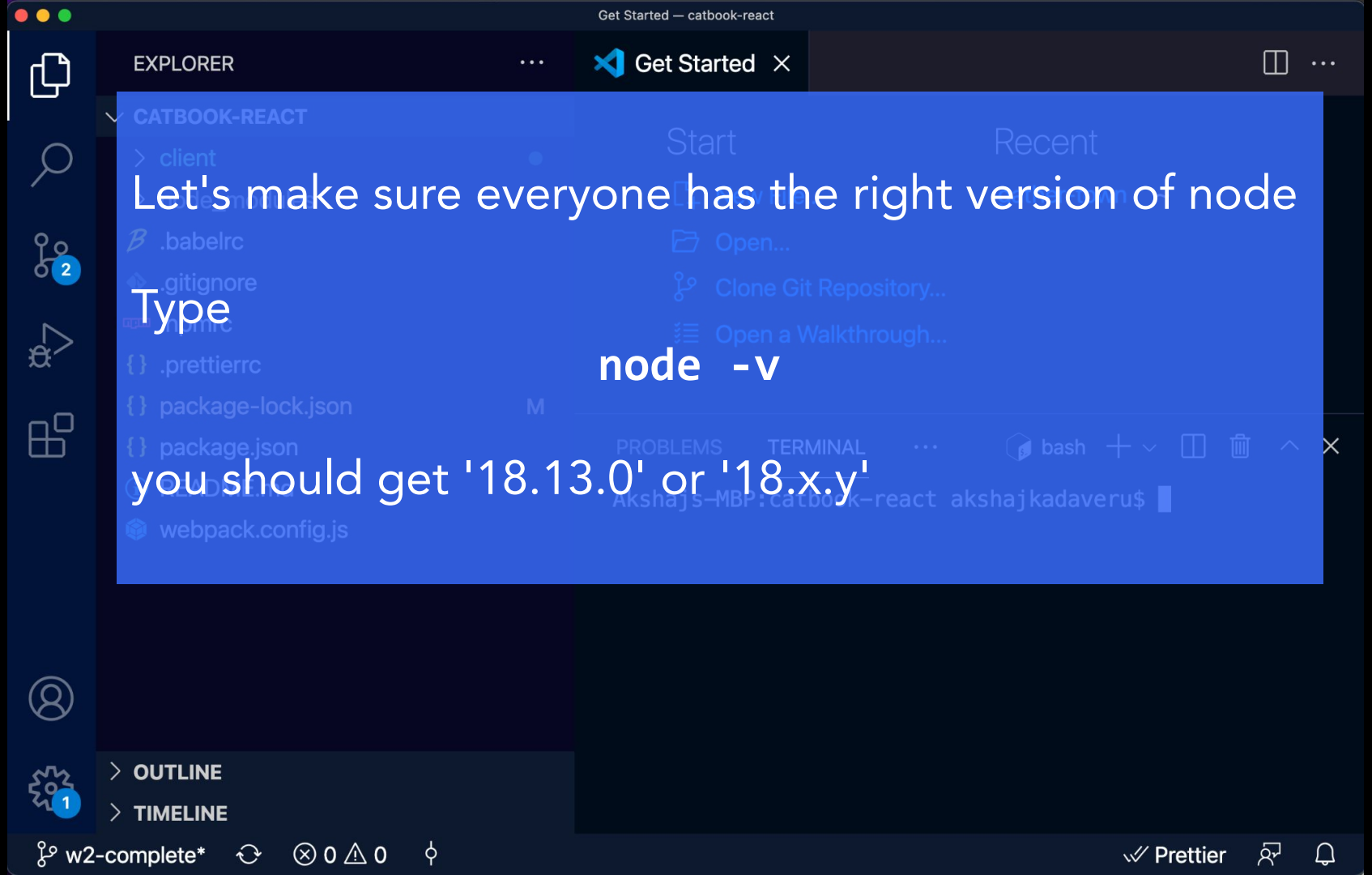
# Workshop 2: Catbook React

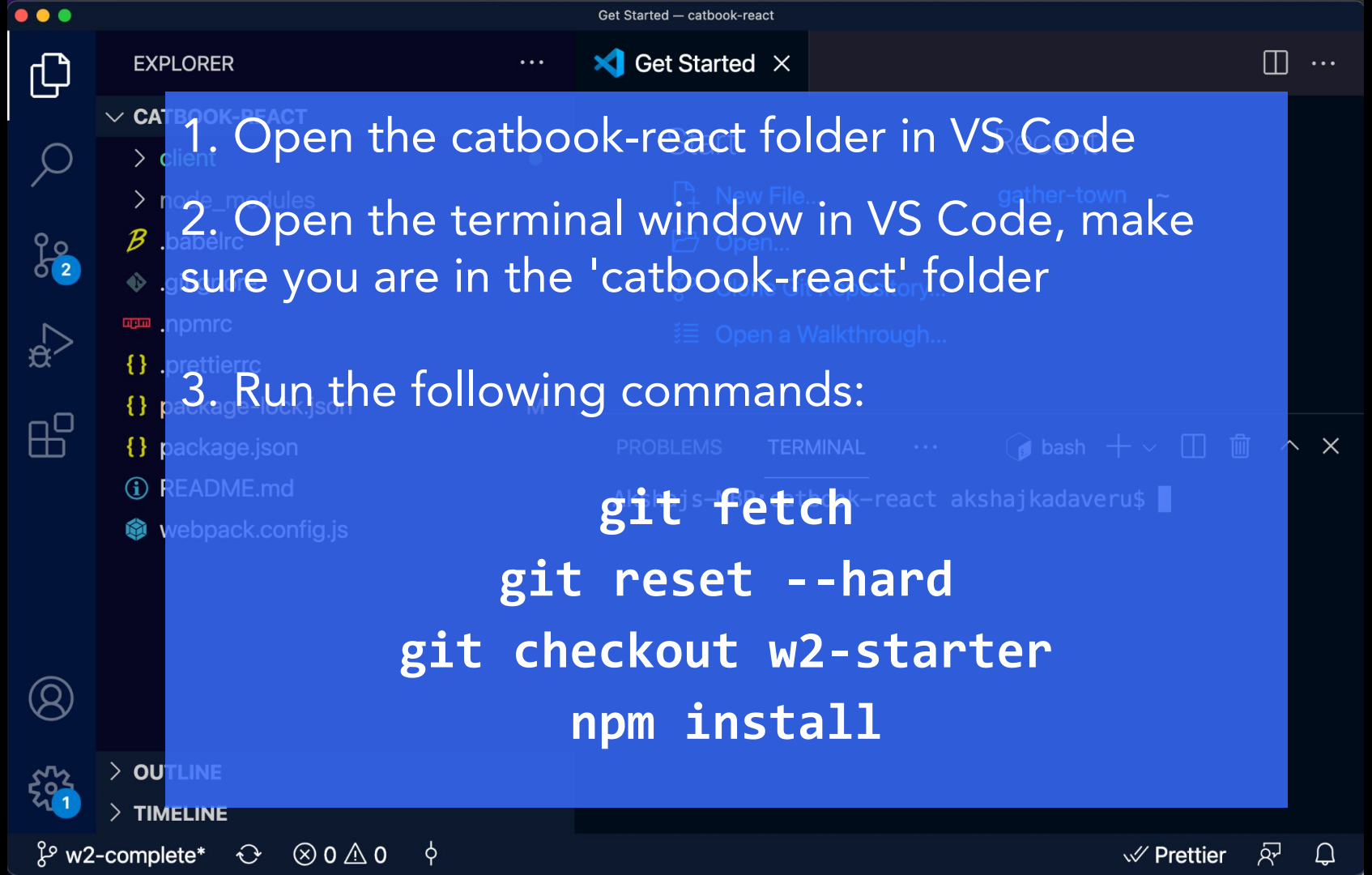



Catbook
YOUR NAME HERE
About Me
Extra Challenge: Modify catbook to show a personalized description here!
Cat Happiness
5235
My Favorite Type of Cat
corgi

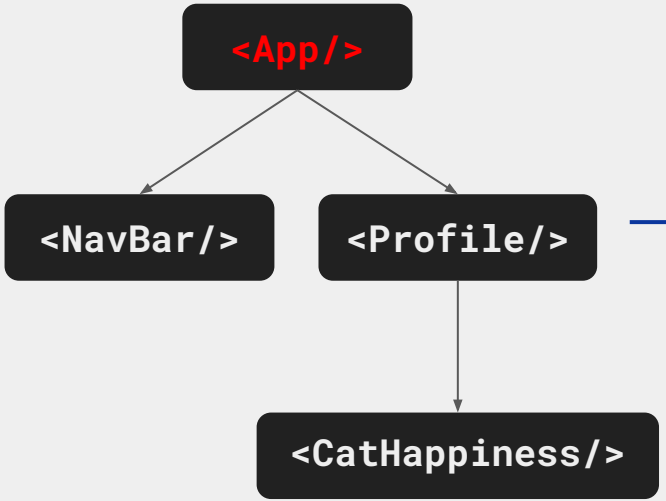
# Demo

# Setup and Starter Code

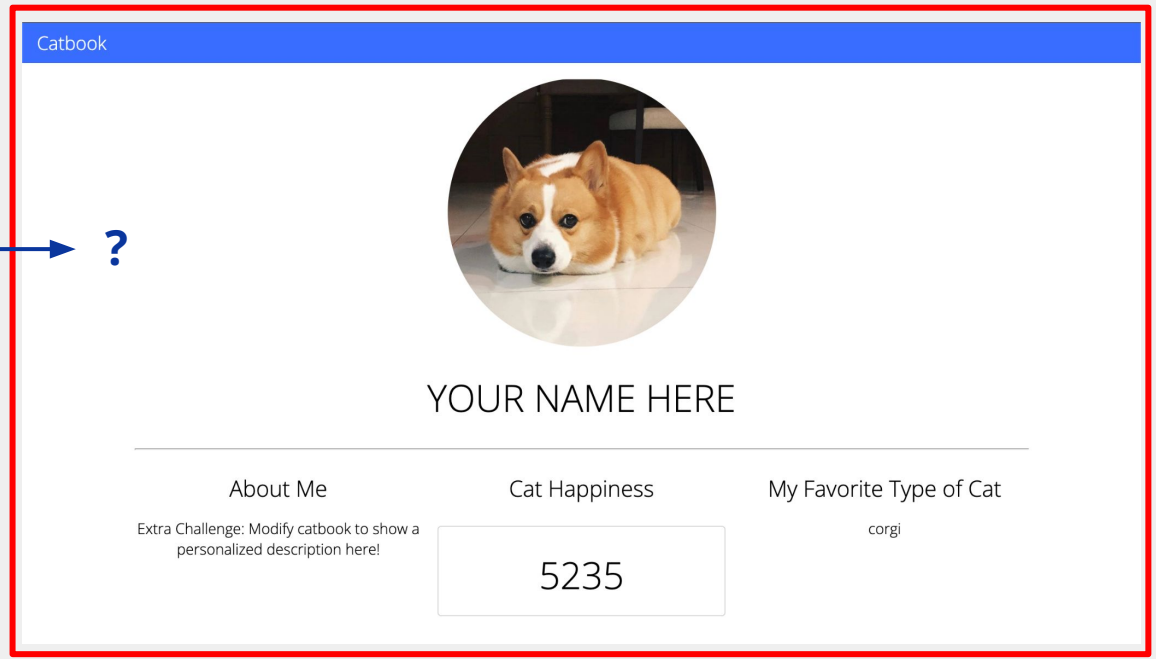

Get Started — catbook-react
EXPLORER
Get Started
CATBOOK-REACT
client
.babelrc
.gitignore
.prettierrc
package-lock.json
package.json
webpack.config.js
Start
Recent
Open...
Clone Git Repository...
Open a Walkthrough...
PROBLEMS
TERMINAL
bash
Akshajs-MBP:catbook-react akshajkadaveru$
OUTLINE
TIMELINE
w2-complete*
0 0
Prettier
Let's make sure everyone has the right version of node
Type
node -v
you should get '18.13.0' or '18.x.y'


Get Started — catbook-react
EXPLORER
Get Started
CATBOOK-REACT
client
node_modules
.babelrc
.npmrc
.prettierrc
package.json
README.md
webpack.config.js
OUTLINE
TIMELINE
1. Open the catbook-react folder in VS Code
2. Open the terminal window in VS Code, make sure you are in the 'catbook-react' folder
3. Run the following commands:
git fetch
git reset --hard
git checkout w2-starter
npm install
PROBLEMS
TERMINAL
bash
akshajkadaveru$
w2-complete*
0 0
Prettier

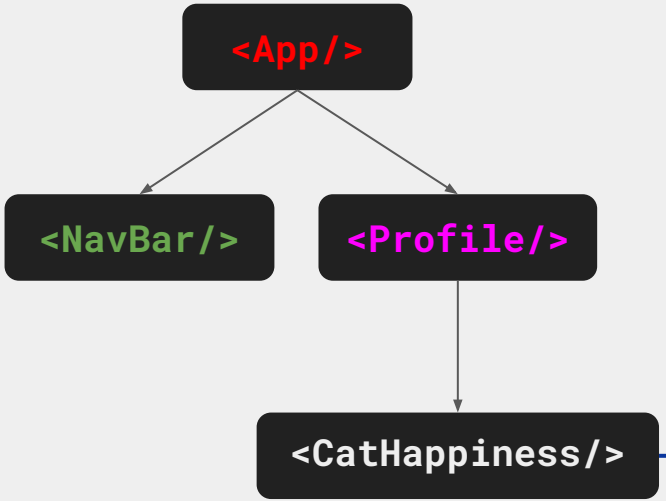
# First: the component tree for Catbook!



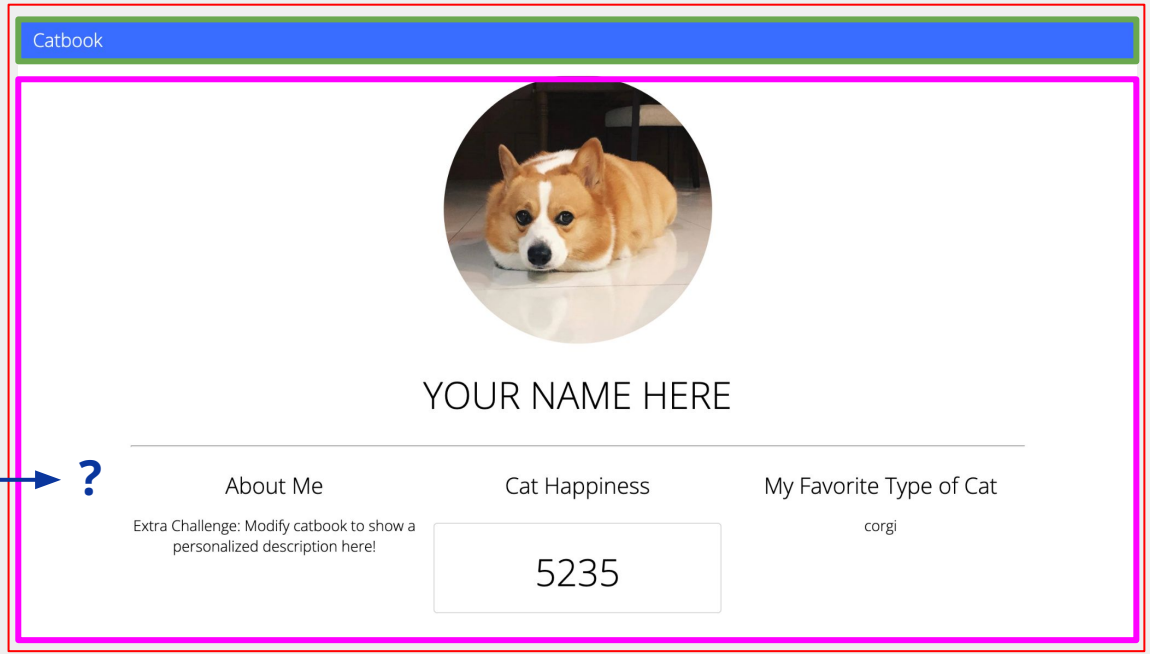

?



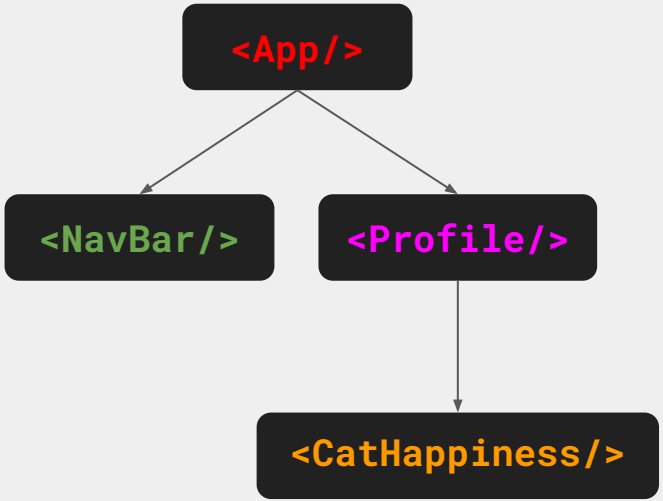
# First: the component tree for Catbook!

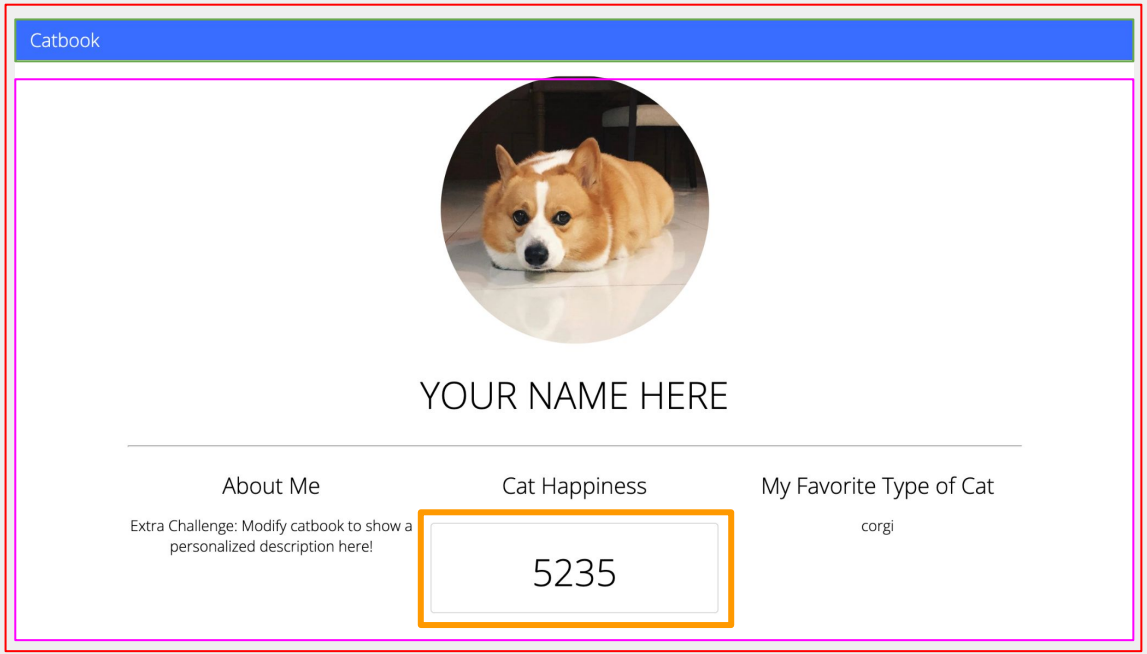



?

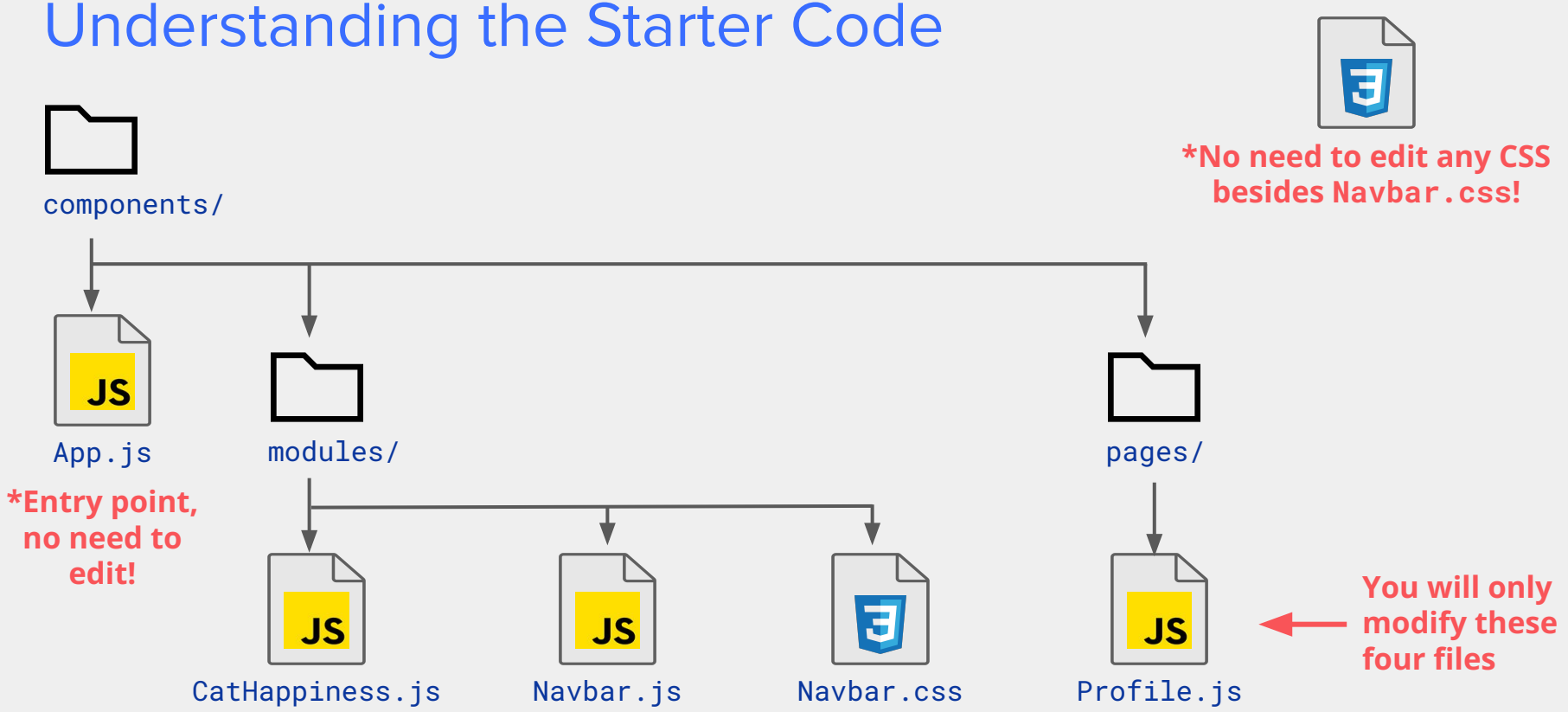


# First: the component tree for Catbook!



# Understanding the Starter Code



# Notes about **Element Classes**

- We use '**`className`**' instead of **`class`** in React
- We put the React component name in front of our class names (e.g. `className="App-container"`)
  - why? so that we don't have the same class name in different components
  - CSS always applies to the **entire webpage**, so must include className to make it specific
    - for example if we set `p {color: red}` in one component it actually applies to **all paragraphs** on the whole webpage

# Running your Code

```
npm run hotloader
```

Navigate to **localhost:5050** and see the page update with your live changes!

# Step 0:
# **Implement the Navbar**

```
git reset --hard
git checkout w2-starter
```

# **Step 0:** Implement React Navbar

```
git reset --hard
git checkout w2-starter
```

You've implemented Navbar using Vanilla HTML- let's do it with React!

Navbar.js


Navbar.css

**Tasks:**

A. Implement **`return()`** in **Navbar.js** with HTML code
B. Implement **Navbar.css** ... go wild! Try to make the NavBar look like the catbook navbar, but feel free to add your own twist!

Catbook

<div **className**="style-name">

```
git checkout w2-starter
```

# **Step 0a:** React Navbar JS

*// NavBar.js*

```
import React from "react";

import "./NavBar.css";

/**
 * The navigation bar at the top of all pages. Takes no props.
 */
const NavBar = () => {
  return (
    <nav className="NavBar-container">
      <div className="NavBar-title">Catbook</div>
    </nav>
  );
};

export default NavBar;
```

```
git checkout w2-starter
```

# **Step 0b:** React Navbar CSS

```css
.NavBar-container {
  padding: 8px 16px;
  background-color: #396dff;
}

.NavBar-title {
    color: white;
    font-size: 20px;
}
```

# Step 1:
# Adding CatHappiness

```
git reset --hard
git checkout w2-step1
```

# Let's get on the same page

Save or close out of all of your 'unsaved' files:

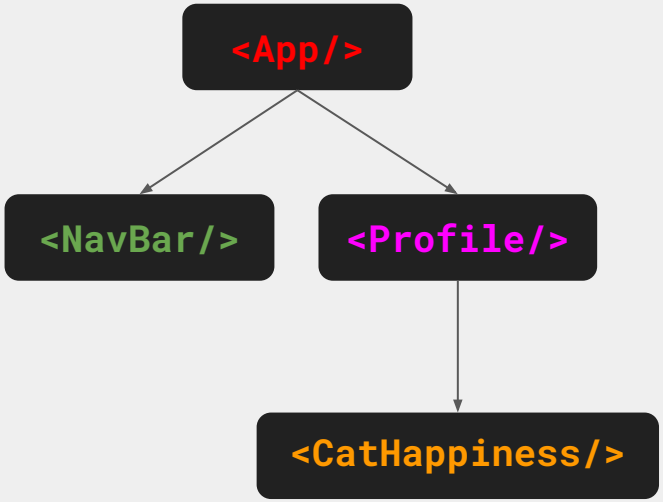

```
git reset --hard
git checkout w2-step1
```

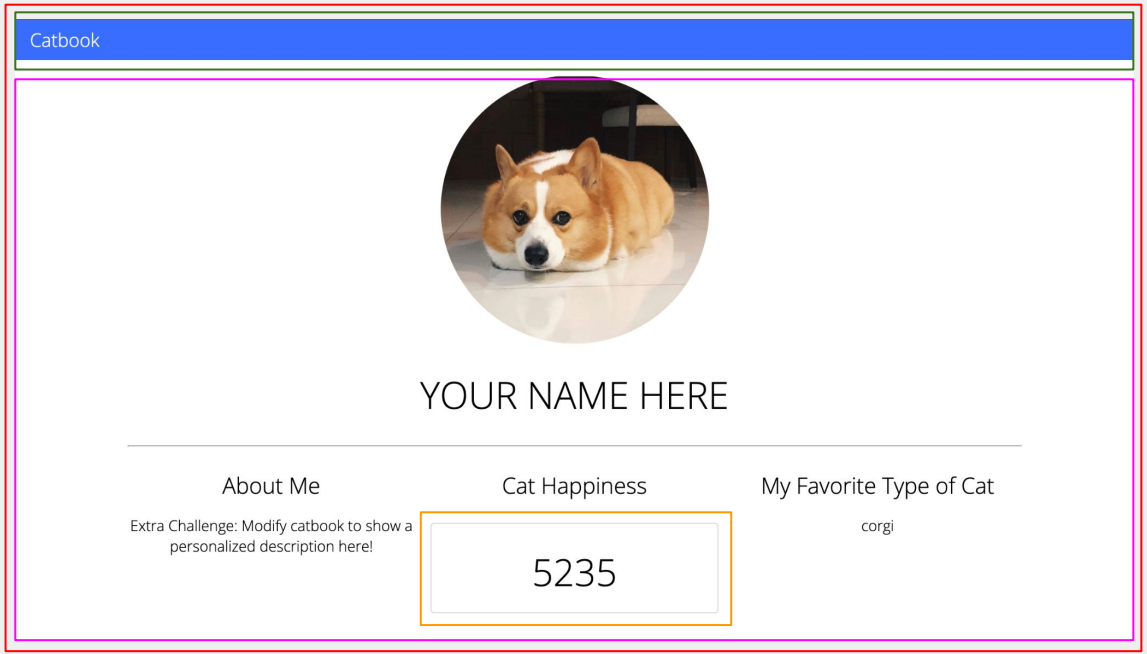
*If it doesn't let you checkout and says 'Please commit your changes or stash them', then 'git stash' should do the trick and you should be able to checkout

# Which component should store the **'catHappiness'** state?

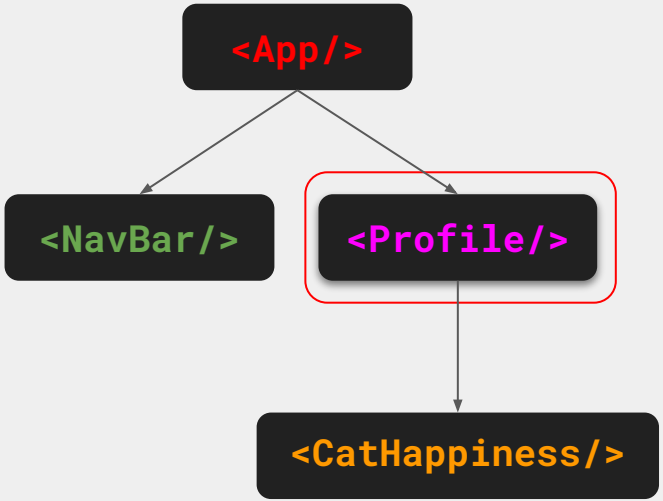
Cat Happiness

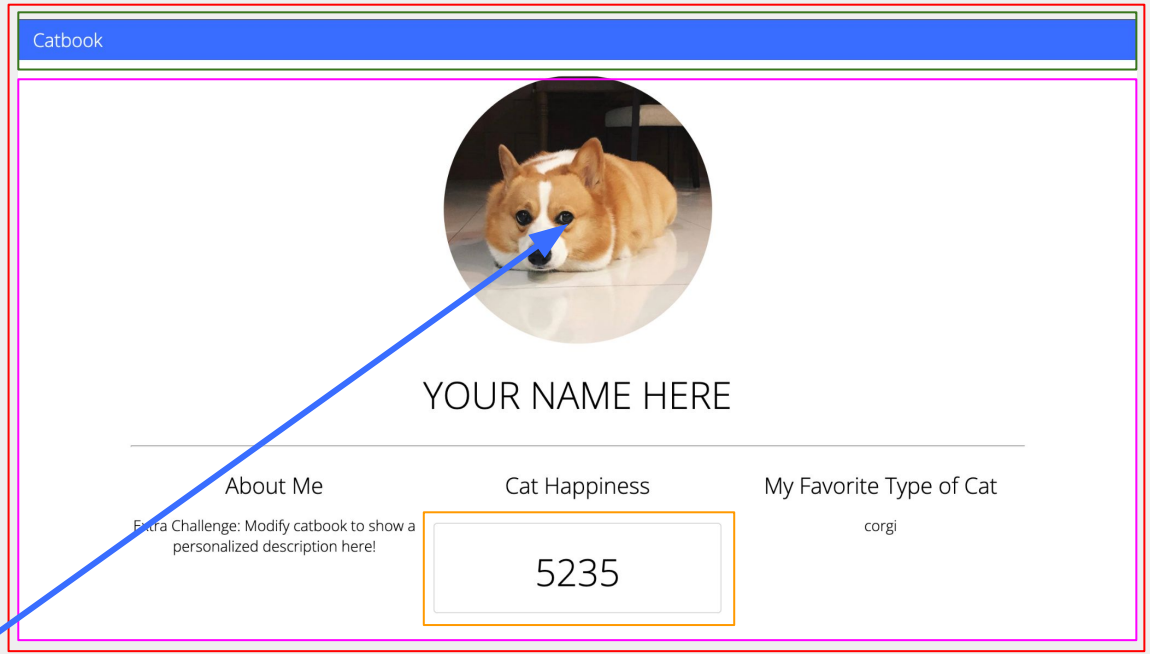
16



**Why Profile?** We want to update **'catHappiness'** when the cat is clicked




Clicking on the cat needs to update catHappiness, and the cat is in Profile!

<App/>

<NavBar/>

<Profile/>

**Props**

None

**State**

catHappiness

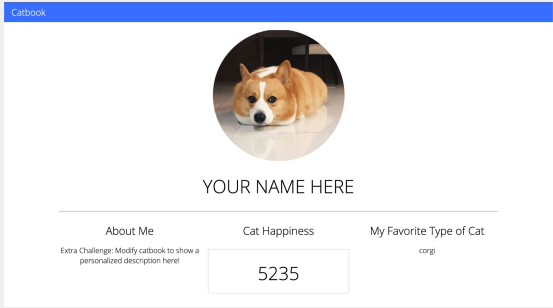
<CatHappiness/>

**Props**

catHappiness

**State**

None



# How do we add state to a component?

# How do we add state to a component?

- `const [something, setSomething] = useState(defaultValue)`

... we use **`useState`**!!

# **Step 1:** Adding CatHappiness

```
git reset --hard
git checkout w2-step1
```


Profile.js


CatHappiness.js

**Tasks:**

A. Add the **catHappiness** state to Profile.js

B. Import the **<CatHappiness/>** Component in Profile.js

C. Add the **<CatHappiness/>** component to Profile.js

D. Display the **catHappiness** state in the **<CatHappiness/>** component using props

```
git checkout w2-step1
```

# **Step 1a:** Add the **'catHappiness'** state to Profile

*In **`Profile.js`**.*

*Also don't forget to import useState*: `import React, { useState } from "react"`

Use the React Guide (http://weblab.is/react-guide) if you're stuck!

```
git checkout w2-step1
```

# **Step 1a:** Add the **'catHappiness'** state to Profile

*// Profile.js  (also don't forget to import useState)*

```
const Profile = () => {
  const [catHappiness, setCatHappiness] = useState(0);
```

```
git checkout w2-step1
```

# **Step 1b:** Import the CatHappiness Component

*// Profile.js*

```
git checkout w2-step1
```

# **Step 1b:** Import the CatHappiness Component

*// Profile.js*

```
import React, { useState } from "react";
import CatHappiness from "../modules/CatHappiness.js";
import "../../utilities.css";
import "./Profile.css";
```

```
git checkout w2-step1
```

# **Step 1c:** Add the CatHappiness component

- Add in the CatHappiness component to Profile.js (in the TODO STEP 1c area), as well as a header in front of it to look like:


- Also, pass in catHappiness as a prop! Don't forget, the syntax will look like:

```
<Component propName={propValue} />
```

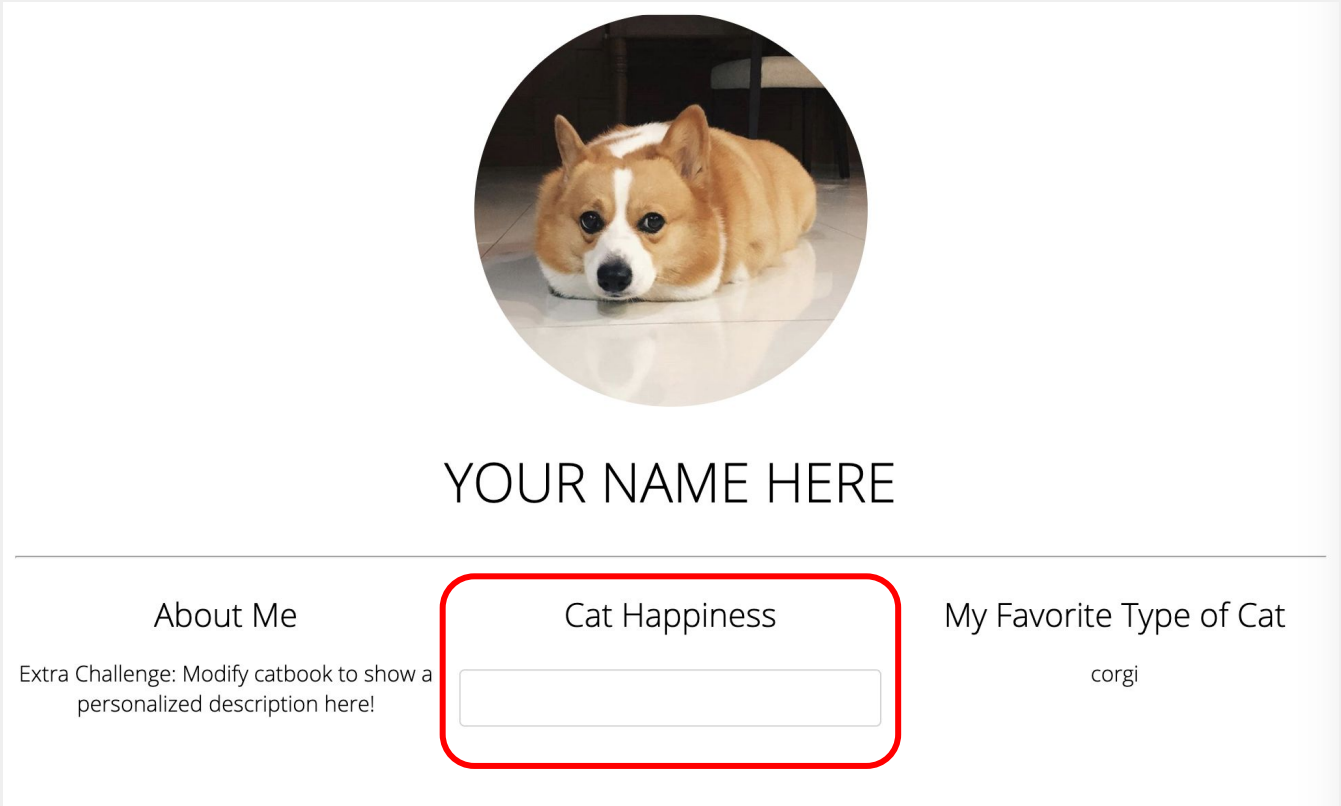
```
git checkout w2-step1
```

# **Step 1c:** Add the CatHappiness component

```
<div className="Profile-subContainer u-textCenter">
  <h4 className="Profile-subTitle">About Me</h4>
  <div id="profile-description">
    Extra Challenge: Modify catbook to show a personalized description here!
  </div>
</div>
<div className="Profile-subContainer u-textCenter">
  <h4 className="Profile-subTitle">Cat Happiness</h4>
  <CatHappiness catHappiness={catHappiness} />
</div>
<div className="Profile-subContainer u-textCenter">
  <h4 className="Profile-subTitle">My Favorite Type of Cat</h4>
  <div id="favorite-cat">corgi</div>
</div>
```

```
git checkout w2-step1
```

# Are we done?



```
git checkout w2-step1
```

# **Step 1d:** Display the incoming catHappiness Prop

*// CatHappiness.js*

```
const CatHappiness = (          ) => {
  return (
    <div className="CatHappiness-container">
      <div className="CatHappiness-story">
        <p className="CatHappiness-storyContent">{                    }</p>
      </div>
    </div>
  );
};
```

```
git checkout w2-step1
```

# **Step 1d:** Display the incoming catHappiness Prop

*// CatHappiness.js*

```
const CatHappiness = (props) => {
  return (
    <div className="CatHappiness-container">
      <div className="CatHappiness-story">
        <p className="CatHappiness-storyContent">{props.catHappiness}</p>
      </div>
    </div>
  );
};
```

# Step 2:
# Updating CatHappiness

```
git reset --hard
git checkout w2-step2
```

# Let's get on the same page

Save or close out of all of your 'unsaved' files:

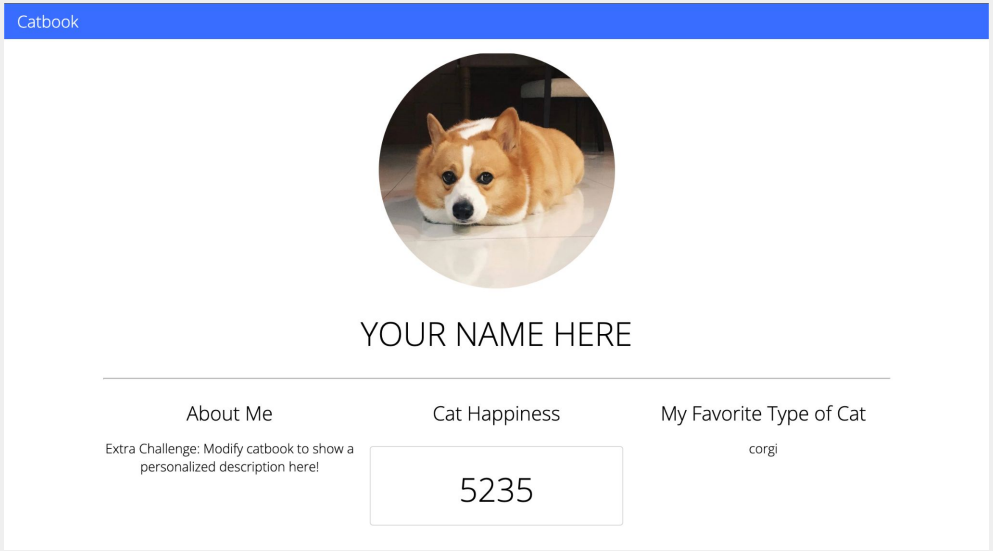

```
git reset --hard
git checkout w2-step2
```

*If it doesn't let you checkout and says 'Please commit your changes or stash them', then 'git stash' should do the trick and you should be able to checkout

```
git reset --hard
git checkout w2-step2
```

# **Step 2:** Update catHappiness State

Now we need to change the catHappiness when we click!



```
git checkout w2-step2
```

# **Step 2:** Updating catHappiness

Profile.js

**Tasks:**

A. (Profile.js line 10) Implement **`incrementCatHappiness`**
B. (Profile.js line 15) Call **`incrementCatHappiness`** whenever the profile picture is clicked

**HINT**: All divs have an `'onClick'` prop that takes a function. Whenever a div is clicked, it runs its onClick function.

# Which of these will work?

A.

```
<div
  className="Profile-avatarContainer"
  onClick={() => {
    incrementCatHappiness();
  }}
>
```

B.

```
<div
  className="Profile-avatarContainer"
  onClick={() => {
    setCatHappiness(catHappiness + 1);
  }}
>
```

C.

```
<div
  className="Profile-avatarContainer"
  onClick={incrementCatHappiness}
>
```

D.

```
<div
  className="Profile-avatarContainer"
  onClick={incrementCatHappiness()}
>
```

A.

```
<div
  className="Profile-avatarContainer"
  onClick={() => {
    incrementCatHappiness();
  }}
>
```

B.

```
<div
  className="Profile-avatarContainer"
  onClick={() => {
    setCatHappiness(catHappiness + 1);
  }}
>
```

C.

```
<div
  className="Profile-avatarContainer"
  onClick={incrementCatHappiness}
>
```

D.

```
<div
  className="Profile-avatarContainer"
  onClick={incrementCatHappiness()}
>
```

# Which of these will work?

A.

```
<div
  className="Profile-avatarContainer"
  onClick={() => {
    incrementCatHappiness();
  }}
>
```

Works, just not the most readable. Also unnecessary since we aren't doing anything else inside this function.

B.

```
<div
  className="Profile-avatarContainer"
  onClick={incrementCatHappiness}
>
```

Works and super clean code!! Recommended implementation!

C.

```
<div
  className="Profile-avatarContainer"
  onClick={() => {
    setCatHappiness(catHappiness + 1);
  }}
>
```

Also pretty good

D.

```
<div
  className="Profile-avatarContainer"
  onClick={incrementCatHappiness()}
>
```

Doesn't work since it will execute the function when the div element is created, not when it's clicked on.

# Finished App

```
git reset --hard
git checkout w2-complete
```

Navigate to **localhost:5050** and change the cat happiness by clicking the profile picture!

# Recap: Writing Components

- We divide our app into c _ _ _ _ _ _ _ _ s, and put one in each file
- Each component is a function with p _ _ _s as the input, and returns JSX (HTML-like code)
- Each component can store internal updatable private info as 's _ _ _ _' variables

- **( )** allows us to enter an JSX environment
- Inside the JSX environment, **{}** allows us to create a mini j _ _ _ _ _ _ _ _ t environment

# Recap: Writing Components

- We pass in props with
  `<Post text="I love weblab" />`
- We read in those props with
  `props.text`
- We declare state variables with
  `const [something, setSomething] = useState(initialValue)`
- React uses **className** instead of **class** for css styles

weblab.is/react-guide

# Later Today: React Lifecycle and Hooks

There’s more than props and state!

But first, APIs and Promises