Recap

Claire & Daniel

Agenda

- Announcements
  - Join us live at weblab.to/zoom
  - Piazza - 1 minute response time
  - Milestone 0 feedback out, Milestone 1 due 11:59 pm TODAY
  - Virtual OH after class at weblab.to/q
- Content Recap
- Kahoot

What have we learned so far?

The first 4 days...

- Three new languages
- An entire front-end library
- Using an API
- Servers

It's okay to be confused!

You may think you are completely lost…

You may think you are completely lost…

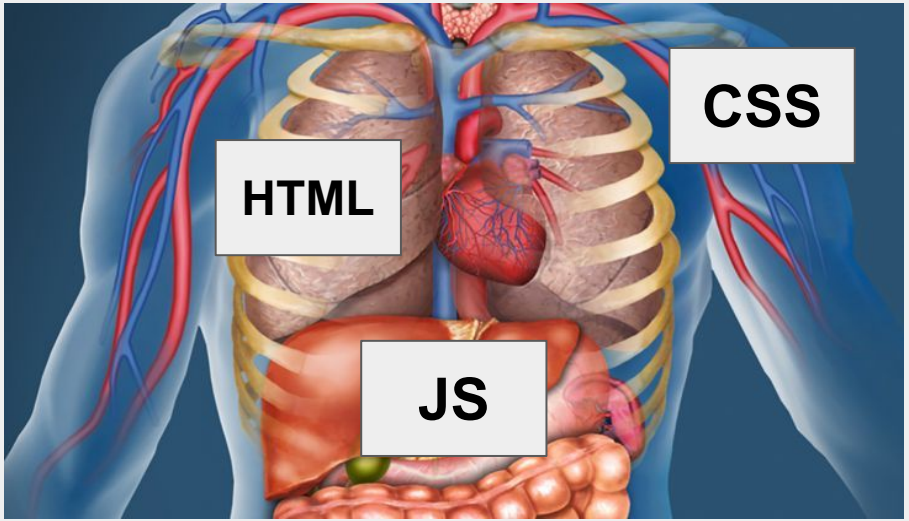
- Websites use the **j** _ _ _ _ _ _ _ _ _ programming language to handle the internal logic of the site and to update variables
- We divide our app into React **c** _ _ _ _ _ _ _ _ _ to organize the structure of the app
- Each of these components is a function with **p** _ _ _ _ as input, can store internal updatable private info as **s** _ _ _ _ variables, and returns HTML-like code
- Props pass **d** _ _ _ , from parent components to **c** _ _ _ _ components
- Websites have a frontend and a **b** _ _ _ _ _ _ , where the server lives to handle API calls

Recap: Web Development Languages

HTML → Organization & Content

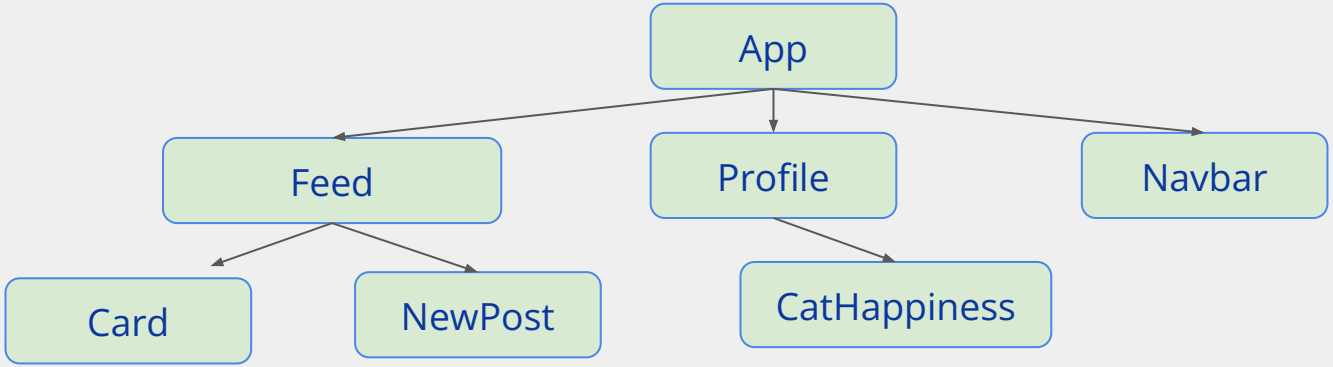
CSS → Layout & Styling

Javascript → Interaction



Recap: React

- Allows you to **structure** your web development code.



Recap: React

```
1  import React, {useState, useEffect} from "react"
2
3  // functional component Example
4  const Example = (props) => {
5      const [state1, setState1] = useState(1);
6
7      useEffect(() => {
8          // Get data we need from server here
9      }, [])
10
11     return (
12         <div>
13             <div>HTML-like code</div>
14             <div>{state1} and {props.dataFromParent}</div>
15         </div>
16     );
17 };
18
19 export default Example;
```

Define component (remember it's a function)

Other components can see this component

Recap: React

```jsx
import React, {useState, useEffect} from "react"

// functional component Example
const Example = (props) => {
    const [state1, setState1] = useState(1);

    useEffect(() => {
        // Get data we need from server here
    }, [])

    return (
        <div>
            <div>HTML-like code</div>
            <div>{state1} and {props.dataFromParent}</div>
        </div>
    );
};

export default Example;
```

Recap: React

```jsx
import React, {useState, useEffect} from "react"

// functional component Example
const Example = (props) => {
    const [state1, setState1] = useState(1);

    useEffect(() => {
        // Get data we need from server here
    }, [])

    return (
        <div>
            <div>HTML-like code</div>
            <div>{state1} and {props.dataFromParent}</div>
        </div>
    );
};

export default Example;
```

Recap: React

```
1  import React, {useState, useEffect} from "react"
2
3  // functional component Example
4  const Example = (props) => {
5      const [state1, setState1] = useState(1);
6
7      useEffect(() => {
8          // Get data we need from server here
9      }, [])
10
11     return (
12         <div>
13             <div>HTML-like code</div>
14             <div>{state1} and {props.dataFromParent}</div>
15         </div>
16     );
17 };
18
19 export default Example;
```

The code in `useEffect` runs after the **first appearance** of an instance of the component.

Recap: React

```jsx
import React, {useState, useEffect} from "react"

// functional component Example
const Example = (props) => {
    const [state1, setState1] = useState(1);

    useEffect(() => {
        // Get data we need from server here
    }, [])

    return (
        <div>
            <div>HTML-like code</div>
            <div>{state1} and {props.dataFromParent}</div>
        </div>
    );
};

export default Example;
```

The code in `useEffect` runs after the **first render**.

Make your API calls here

Recap: React

```
1  import React, {useState, useEffect} from "react"
2
3  // functional component Example
4  const Example = (props) => {
5      const [state1, setState1] = useState(1);
6
7      useEffect(() => {
8          // Get data we need from server here
9      }, [])
10
11     return (
12         <div>
13             <div>HTML-like code</div>
14             <div>{state1} and {props.dataFromParent}</div>
15         </div>
16     );
17 };
18
19 export default Example;
```

The return statement is where you write the **HTML** for the component.

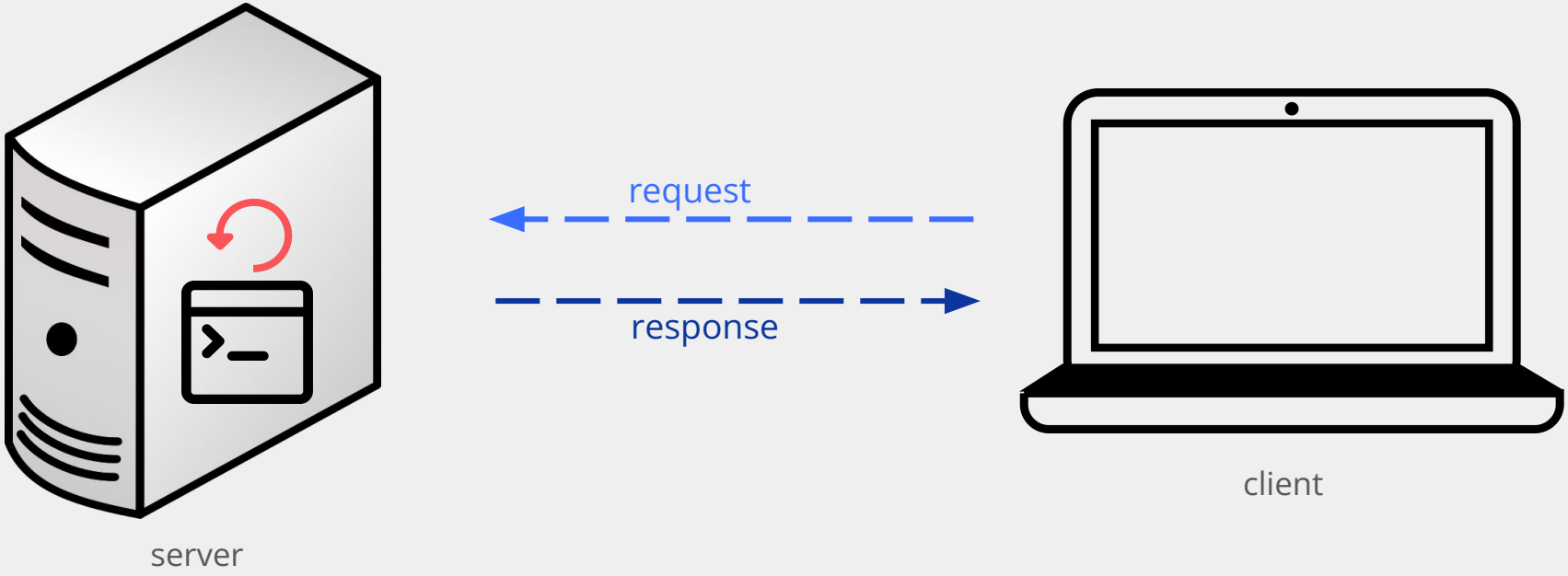
Recap: React

```jsx
import React, {useState, useEffect} from "react"

// functional component Example
const Example = (props) => {
    const [state1, setState1] = useState(1);

    useEffect(() => {
        // Get data we need from server here
    }, [])

    return (
        <div>
            <div>HTML-like code</div>
            <div>{state1} and {props.dataFromParent}</div>
        </div>
    );
};

export default Example;
```

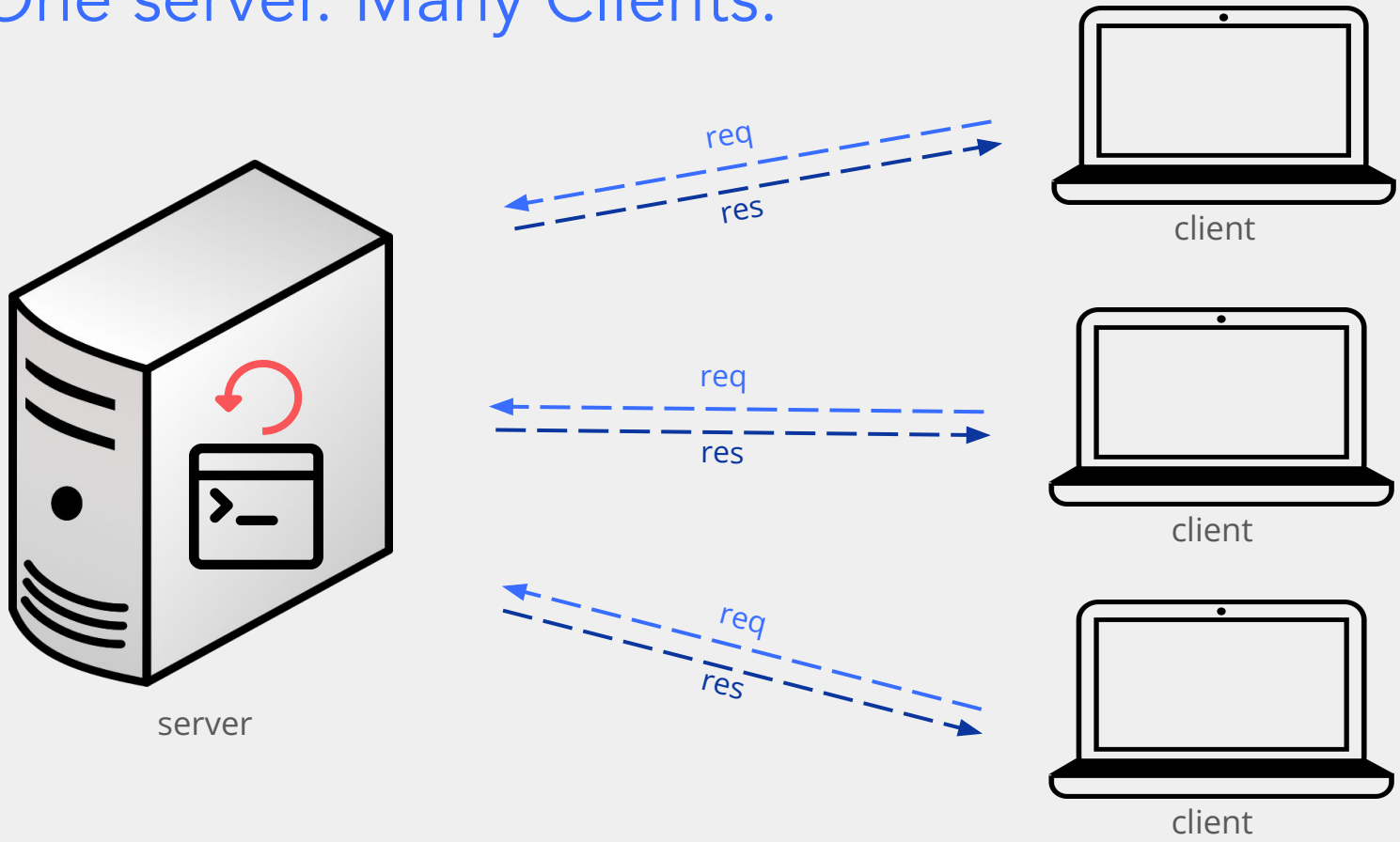
The return statement is where you write the **HTML** for the component.

You can **mix** HTML & Javascript to make your code cleaner!

Recap: what is a server?



One server. Many Clients.



What is the need for a server?

- File access
- Centralization
- Security

Recap: HTTP

- HTTP is a way to **communicate** on the internet
  - GET Request (ask for information)
  - POST Request (add new information)
  - You commonly use HTTP to interact with an API

Recap: APIs

- An API is just a **set of rules**.
- They describe how you **access data** from a server.
- API endpoints are the various functions you can call

Recap: Asynchronous Programming

- Javascript is an **asynchronous** language.
    - We cannot wait for an API request to complete before executing more code!

```
let petName = "unknown";
get('/pet/0').then((pet) => {
    petName = pet.name;
    //assume pet.name = "Labrador"
});
console.log(petName);
```

What does this print out?

Promises

- It is a guarantee that the unresolved return value will eventually become something (either a valid response or an error)
- Async functions, like get(), return a promise
- In order to act on that promise, we use the .then() syntax

```
let petName = "unknown";
get('/pet/0').then((pet) => {
    petName = pet.name;
    //assume pet.name = "labrador"
});
console.log(petName);
```

Promises

```
1  const five = 5;
2  get("/api/addOne", {input: 5})
3
4
5
6
7
8
```

Promises

```
1  const five = 5;
2  get("/api/addOne", {input: 5}).then((six) => {
3      return six + 1;
4  
5  
6  
7  
8  
```

Promises

```
1  const five = 5;
2  get("/api/addOne", {input: 5}).then((six) => {
3      return six + 1;
4  }).then(((seven) => {
5      console.log(seven); // will print 7
6  
7  
8  
```

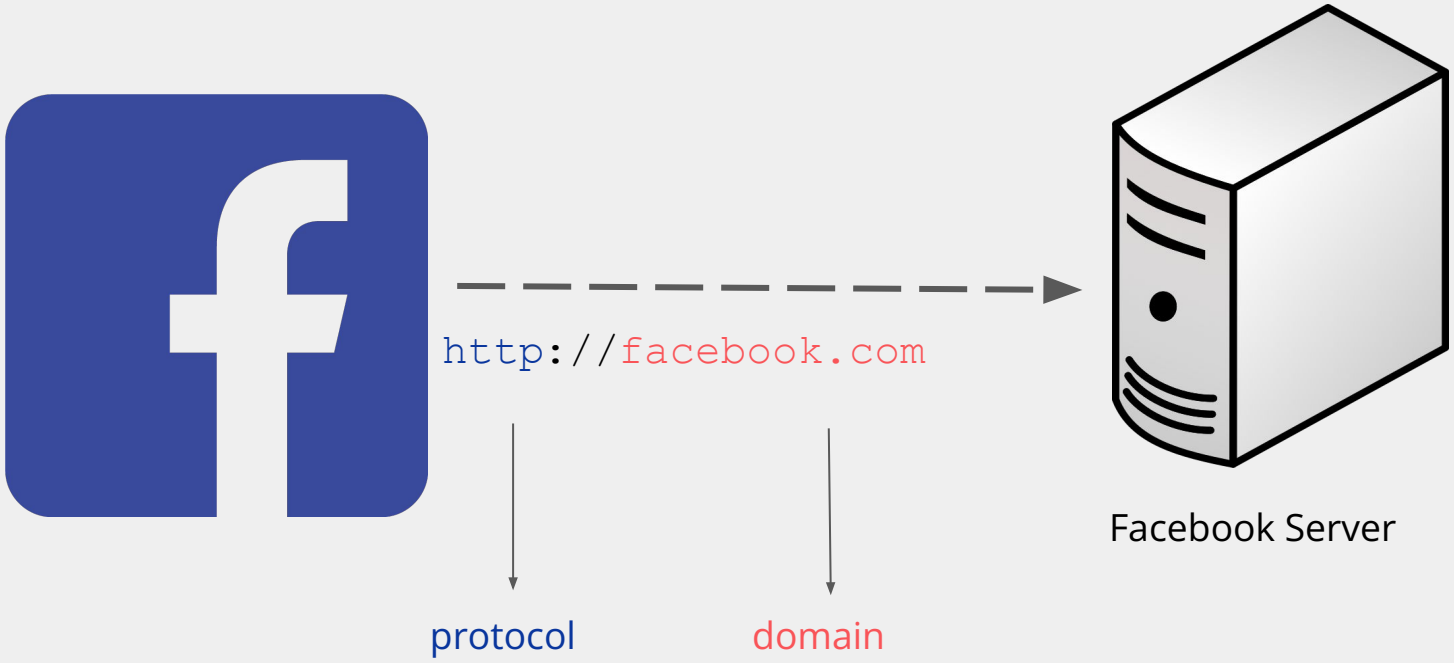
Promises

```
1  const five = 5;
2  get("/api/addOne", {input: 5}).then((six) => {
3      return six + 1;
4  }).then(((seven) => {
5      console.log(seven); // will print 7
6  })).catch((error) => {
7      // handle the error
8  })
```

Coming Up!

- Kahoot Review
- Databases
- Authentication
- Sockets
- Deployment
- & more advanced topics!

Connecting to a Server



Processes & Ports

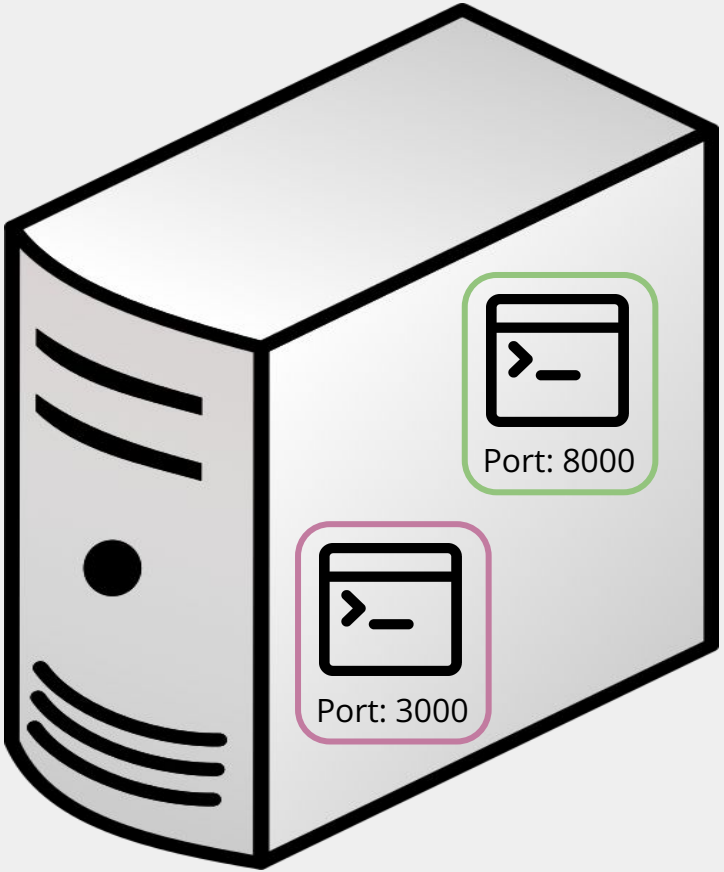
A server **binds** to a **port** on a computer.

`http://example.com:8000`

`http://example.com:3000`

`protocol://domain:port`

Why do we not need to specify ports on most websites?



Two servers on one computer

Using your own machine as a server

- Every computer can run server code!

- Your own computer has a special domain: **localhost**
  - http://localhost:3000 → connects to a server on port 3000.

Extras

npm start

- Starts your server
- Relaunches after code changes

npm run hotloader

- Recompiles when React code changes
- Forwards requests for /api/* to localhost:3000

npx webpack

- Creates bundle.js
- Need to run after every code change
- Not fun while developing :(

server.js

```
// load the compiled react files, which will serve /index.html and /bundl
const reactPath = path.resolve(__dirname, "..", "client", "dist");
app.use(express.static(reactPath));

// for all other routes, render index.html and let react router handle it
app.get("*", (req, res) => {
  res.sendFile(path.join(reactPath, "index.html"));
});
```