ECS 162
WEB PROGRAMMING

Announcements

- My lab hours after class in 67 Kemper
- Sorry we missed 9AM lab hour today

- Flashcards 1 due Thurs
  - Just card creation page, stores cards in DB
  - One Node+Express server, started by "node [server name].js"
  - HTML should contain a single div, React fills in DOM

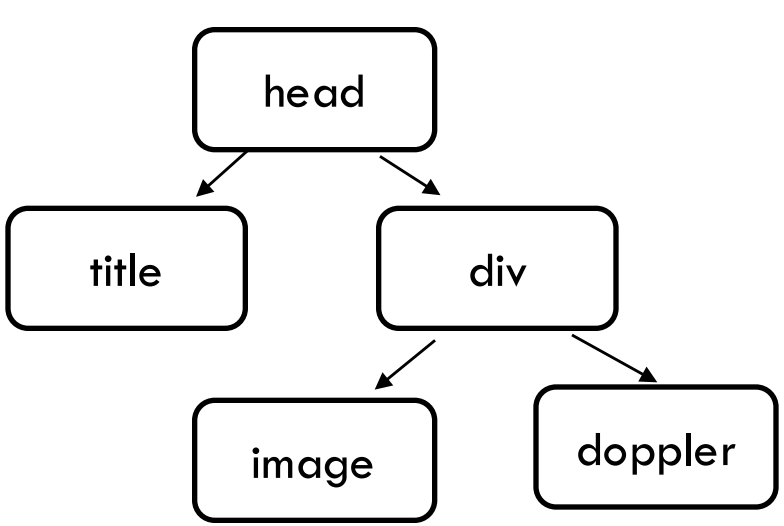
React benefits

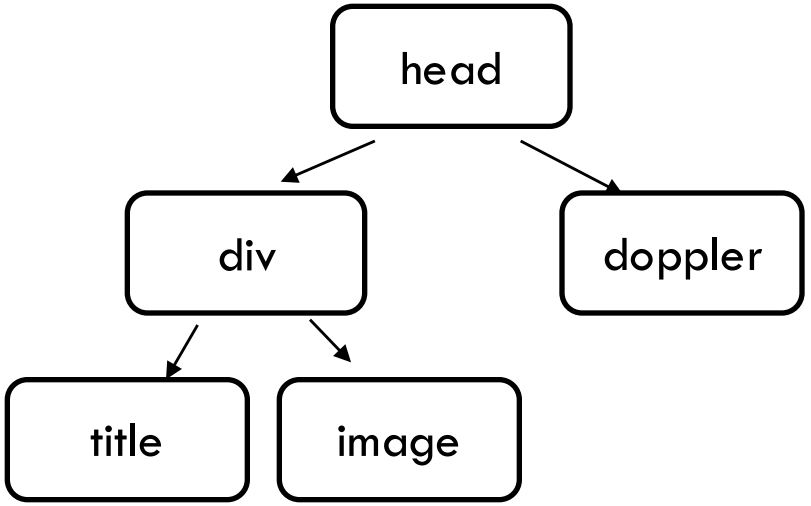
- Problems React solves
  - Modularity and reuse of components
  - Flexibility of DOM

  - How to achieve this?

Different DOM structures

- Need a component with an if statement. If narrow window, produces one DOM subtree, if wide another.





- Check viewport width in .js rather than using a media query.

Topics for today

- "this" in classes
- modules
- redirection

Behavior of “this”

```
class Weather {
    constructor (temp) {
        this.fahrenheit = temp;
        }
    celsius () {
         return (this.fahrenheit-32)*5/9;
         }
}
```

Detail: behavior of “this”

```
let davis = new Weather(71);
console.log(davis.celsius());

let newTemp = davis.celsius;
console.log(newTemp());
```

- Second one crashes! “this” is undefined in newTemp.

Similar problem

```
let sanFrancisco = new Weather(64);
sanFrancisco.newTemp = davis.celsius;

console.log(sanFrancisco.newTemp());
```

- Here “this” refers to the sanFrancisco object, even though it is a method of the davis object.
- How do we fix this?

Behavior of “this”

```
let sanFrancisco = new Weather(64);
sanFrancisco.newTemp = davis.celsius;

console.log(sanFrancisco.newTemp());
```

- Here “this” refers to the sanFrancisco object, even though it is a method of the davis object.
- How do we fix this?
- The answer ***used to be*** “closure”

Closure solution

```
class Weather {
    constructor (temp) {
        this.fahrenheit = temp;
        let that = this;
        this.celsius = function () {
            console.log(that);
            return ((that.fahrenheit-32)*5/9).toFixed(1);
        }
    }
```

- "That" is a local variable of the constructor
- The method is defined in the constructor
- It keeps "that" in its closure

ES5 solution: bind

```
class Weather {
    constructor (temp) {
        this.fahrenheit = temp;
        this.celsius = this.celsius.bind(this);
    }
    celsius () {
            console.log(this);
            return ((this.fahrenheit-32)*5/9).toFixed(1);
        }
}
```

ES6 - define method with arrow fn

```
class Weather {
    constructor (temp) {
        this.fahrenheit = temp;
        this.celsius = () => {
            return ((this.fahrenheit-32)*5/9).toFixed(1);
        }
    }
}
```

Semantics of arrow functions

- A function defined with a statement or expression, even a method, uses the value of “this” in the context in which it is run.
- An arrow function uses the value from the context in which it was defined.
- This is the semantic distinction between arrow functions and function expressions.

In React classes

```
class CreateCardMain extends React.Component {
  constructor(props) {
      super(props);
      this.state = { opinion: "Life is a bowl of cherries" }
      this.checkReturn = this.checkReturn.bind(this);
        ...
```

- When method checkReturn gets called, it needs to be able to use “this” to change state in object.

Modules

- We've been using modules for libraries in Node, including express, fs, sqlite3, request.
- Modules are also useful for breaking code up into several files (eg. server, API request, database).
- As usual, there are multiple ways to do it.
- "require" is built into Node
- "import" is built into the browser
- Let's start with "import", in the browser.

New file Ajax.js

```
export function sendTranslate(callback,phrase){
    let url = "translate?english="+phrase;
    let xhr = new XMLHttpRequest();
    xhr.open("GET",url);
        ...
```

- Contains functions that send and receive AJAX requests.
- Functions that need to be called from outside the module are labeled with “export”.
- Similarly data that needs to be seen from outside.

In .jsx file

```
import { sendTranslate } from './ajax.js';
```

- Brings in any of the exported functions from the module.
- Our module and the function it is called from both should be in /public (or a child).
- Both need to be downloaded to the broswer.

In .html file

```
<script src="lango.js" type="module"></script>
<script src="ajax.js" type="module"></script>
```

- Both need to be labled type=“module” (I am not sure why).

Browser software ecosystem

- Everything used on the browser has to be downloaded; nothing is installed.
- Scripts might be downloaded from many places.
- Until recently, ”import” and “export” were not well-supported by browsers. So everything existed in one big namespace. What is the problem with this?

Browser software ecosystem

- Everything used on the browser has to be downloaded; nothing is installed.
- Scripts might be downloaded from many places.
- Until recently, ”import” and “export” were not well-supported by browsers. So everything existed in one big namespace. What is the problem with this?

Two modules might use the same function or variable name, causing crashes or other bugs.

- Also, using lots of modules, either your own or imported, gets complicated.

Bundlers

- A bundler takes multiple modules (your own or included), handles compiling and linking to produce a single .js file for your app.
- Configuring the bundler on the server is kind of like making a makefile for a C program.
- Other possible features:
  - linter – checks for possible bugs
  - source-map – connect .js to original .jsx files for the debugger
  - minification – shrink .js file down
  - etc, etc…

Bundlers

- The Webpack bundler is widely used with React
- Webpack also includes a server (do not use in this assignment).
- Newer options coming up, eg. Parcel.

- On this project, just putting the pieces together is easier learning a bundler.
- On projects with multiple front-end programmers and hundreds of modules, bundlers are very important.

Modules on the server

- On server, we can install modules, instead of downloading them, so no bundling into one .js file.
- Handling namespaces, keeping track of updating modules, etc. still important.

- We have been including modules installed with npm using “require”. This has been around forever.
- There are multiple ways to get “import” working on the server, but I’m going to stick with require.

Our own server-side module

- In useAPI.js, at the bottom, export what needs to be visible to other files:

```
exports.functionName = functionName;
```

- In langoServer.js, import using “require”:

```
const api = require('./useAPI');
```

- Calling it api (or whatever) helps with namespace issues.

Our own server-side module

- Use via “api” variable:

```
api.issueRequest(q.english, handleAPIresponse);
```

- Have as many files as you want. One for API, one for database, one maybe for login?
- Only API module has to require “request”, only DB module has to require “sqlite3”, etc.

Redirects

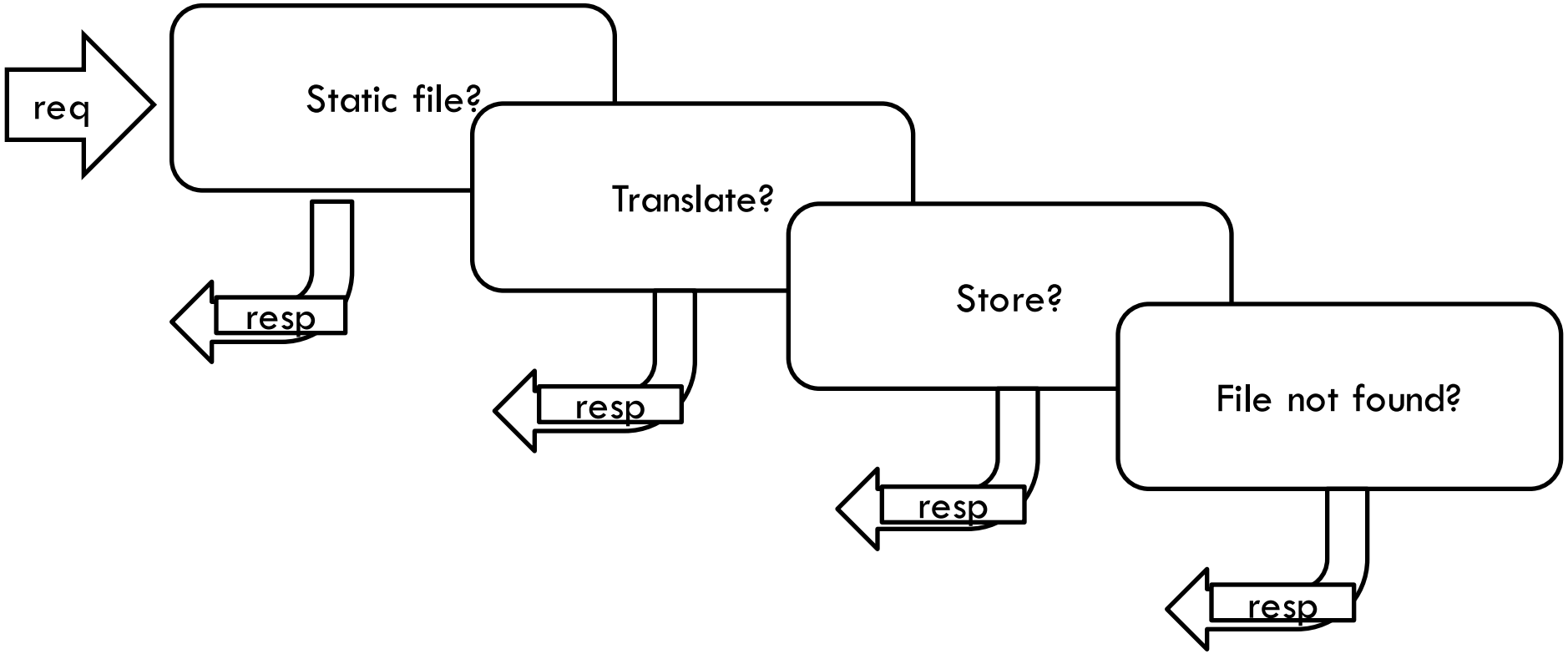
- You get to this app (and most apps) by typing a single URL, which brings in an HTML page, which brings in everything else...
- An app might need several HTML pages, but with React this is often unnecessary – much faster to rebuild the DOM than to download a new HTML page and then rebuild the DOM.
- React-router actually ***simulates*** using multiple pages, really stays in browser and redraws UI.

Redirects

- Sometimes we really need to go to another Web page, particularly somebody else's Web page.
- To use "login with Google", we will need to go to some pages at Google, and then come back.
- We want the card creation and card review pages visible only if the user is logged in.

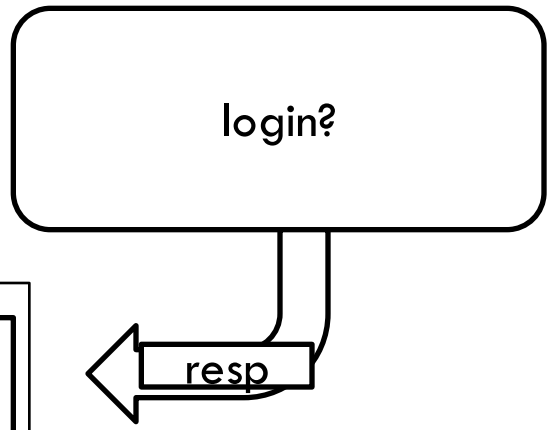
- We do these redirects in the server, using express.

Server Pipeline



Redirect

- New redirect pipeline stage sends HTTP response with new URL to go to.



Return code: 302
Redirect address:
https://accounts.google.com/o/oauth2/v2/auth?response_type=code&redirect_uri=http%3A%2F%2Fserver162.site%3A30057%2Fauth%2Fredirect&scope=profile&client_id=472036695689-s9n5kubr2kuqftbvk0ujl67i324njo3p.apps.googleusercontent.com

Browser

- Gets redirect HTTP response, without going into our code immediately sends new HTTP request to the specified address at Google.
- This kicks off the login process. More Wds.