Cubstart Lecture 6

Node.js and Express

[start recording]

Administrivia

- HW 5: OpenWeatherMap is due on Friday
  - Please ask questions on the Ed Megathread!!!!!!
- HW 6 will be released soon
- Final project: maybe start thinking about partners?

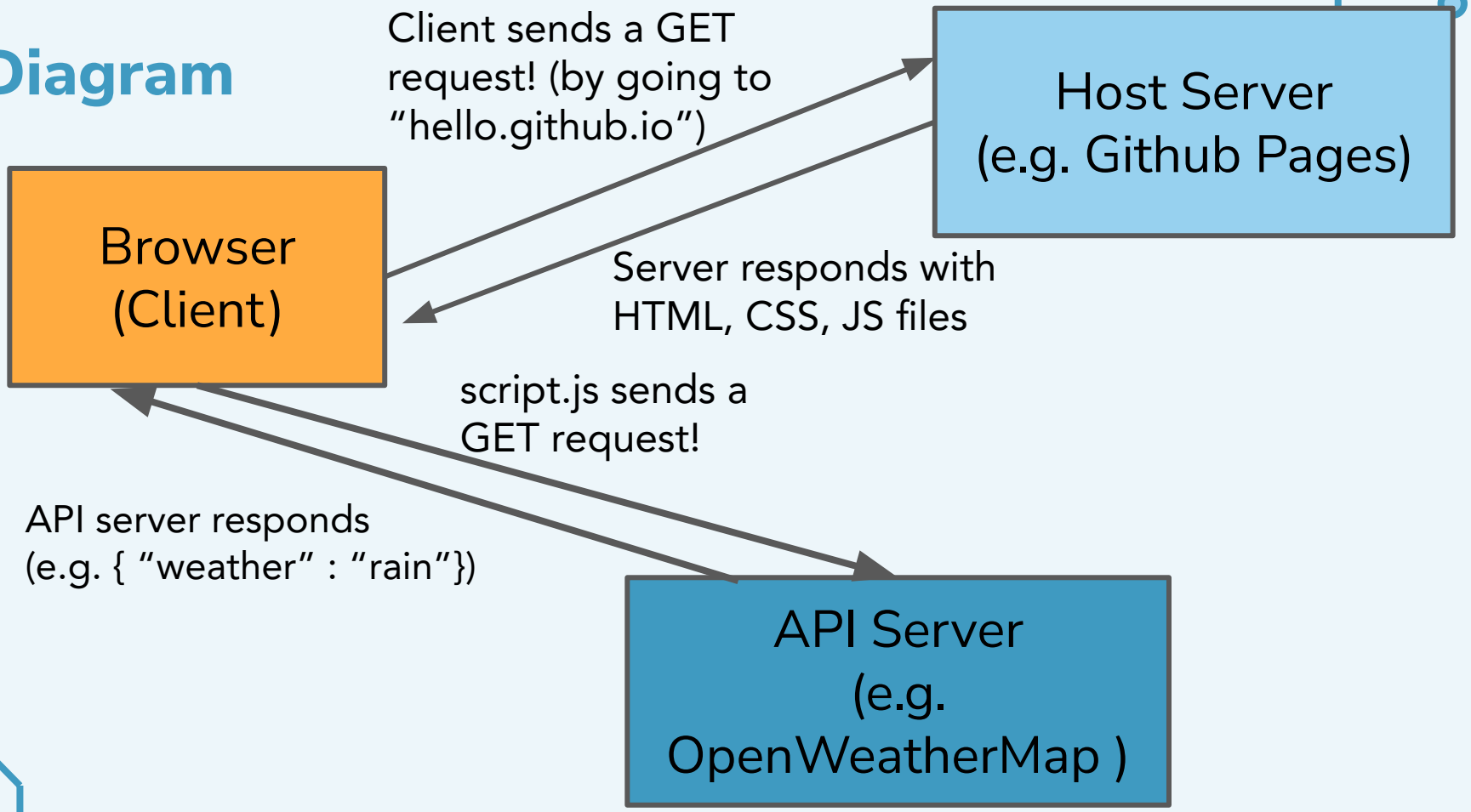
Recap

Recap: HTML, CSS, JS, APIs

- **HTML:** Website structure, content
- **CSS:** Website styling
- **JS:** DOM Manipulation
- **APIs:** Using pre-existing APIs and/or endpoints, retrieving and parsing data

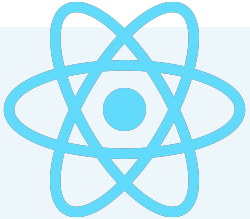
```
async function apiCall(url) {
  let response = await fetch(url)
  let data = await response.json()
  return data
}
```

Diagram


Client sends a GET request! (by going to "hello.github.io")
Host Server (e.g. Github Pages)
Browser (Client)
Server responds with HTML, CSS, JS files
script.js sends a GET request!
API server responds (e.g. { "weather" : "rain"})
API Server (e.g. OpenWeatherMap)

Frontend

The website/webapp that you see and interact with in your browser



React.js

What can you do with this?

- Simple weather app!
- COVID Data app?
- More stuff!


```
async function apiCall(url) {
  let response = await fetch(url)
  let data = await response.json()
  return data
}
```

Some limitations

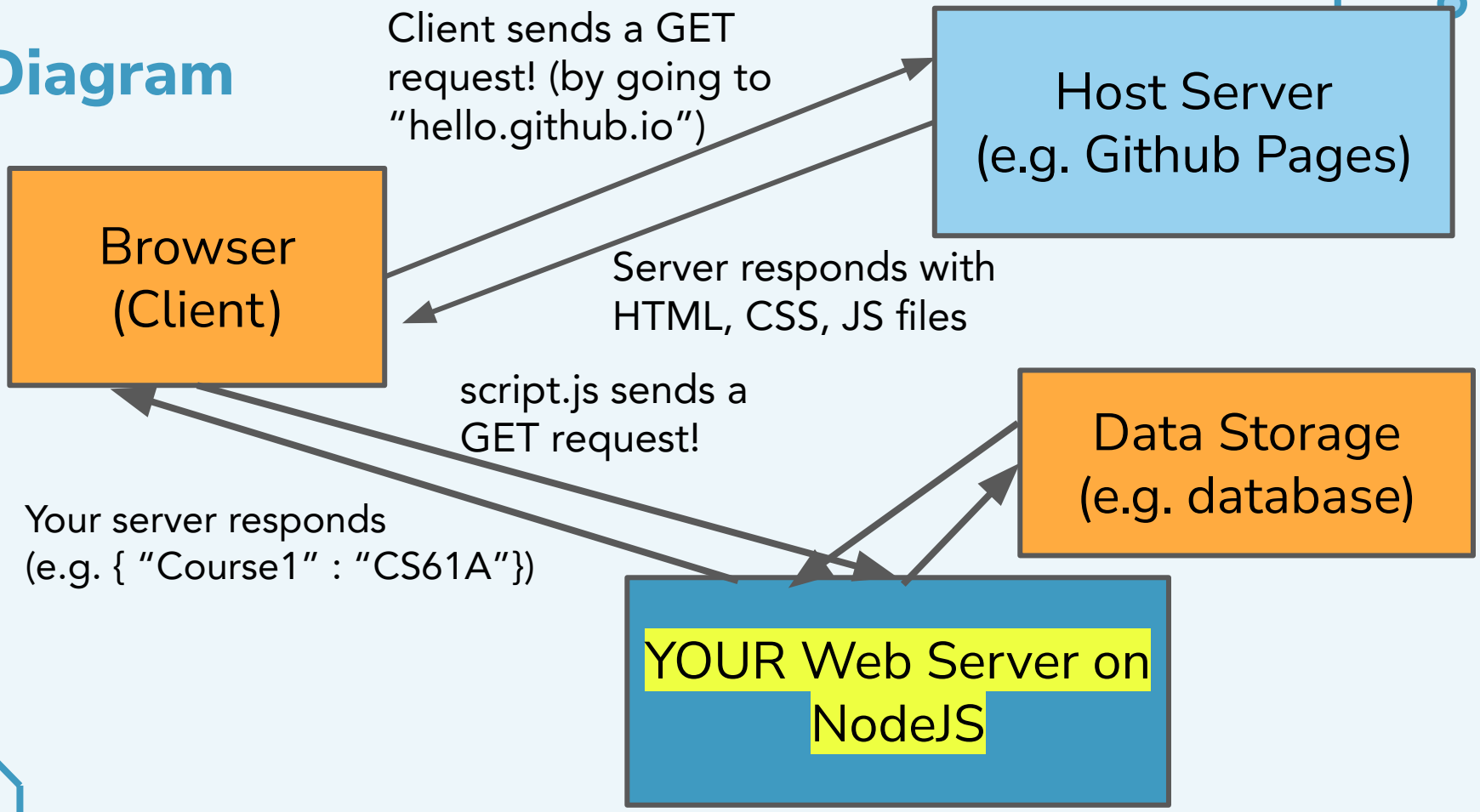
- Storing data (persisting data between sessions)
- User authentication and authorization
- Running everything on the client (browser)
- Many more...

Backend Development: Servers!
(i.e Making your own API)

Node.js

- Server environment
- Allows the programmer to run JavaScript on a web server

Diagram


Client sends a GET request! (by going to "hello.github.io")
Host Server (e.g. Github Pages)
Browser (Client)
Server responds with HTML, CSS, JS files
script.js sends a GET request!
Data Storage (e.g. database)
Your server responds (e.g. { "Course1" : "CS61A"})
YOUR Web Server on NodeJS

Node.js vs Browser

- Both powered by JavaScript
- Browser interacts with the document (DOM), and window
- Node.js is used to make web servers w/ a comprehensive API

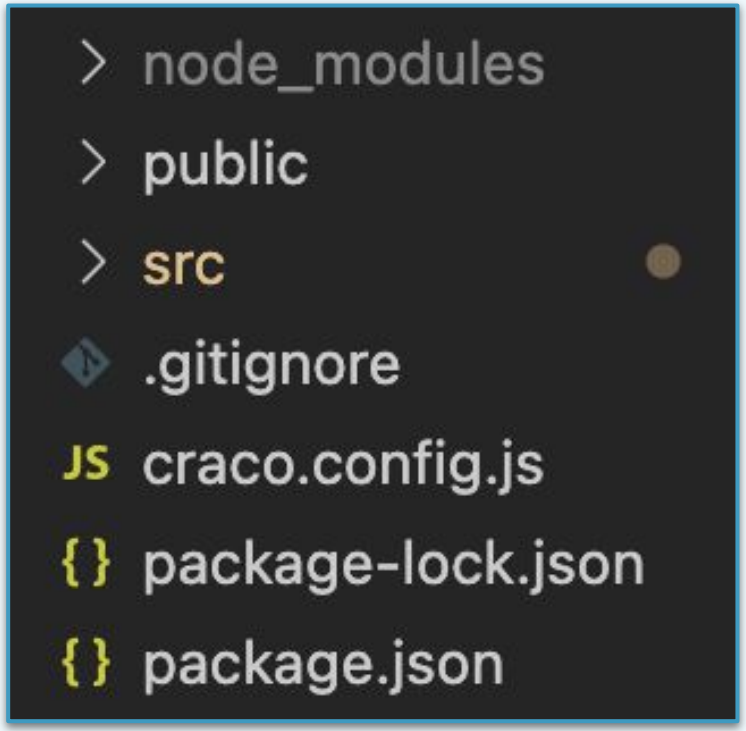
Modules

- Pieces of reusable code that you can include in your application without reinventing the wheel
- Examples of importing modules:
  - Data 8: import datascience (allows you to create tables, etc.)
  - Python: Import math, import pandas as pd, import numpy as np
  - Use functions from module: np.array(), math.maxint
- **require() over import**

```
const express = require('express');
```

npm - Node Package Manager

- Manage any dependencies (modules, libraries) within a project (frontend/backend)
- Lists all dependencies in a project in *`package.json`* and the actual code in a folder called *`node_modules`*
- **`npm init`**
  - `Creates package.json`
- **`npm install`**
  - `Installs module`
  - `Updates package.json`



Ok but how do we actually make a server?

Middleware

- Middleware literally means anything you put in the middle of one layer of the software and another
- Think of Node.js as the underlying mechanism to achieve web server functionality
- And middleware can be mounted on top of the web server and abstracts away the complex low-level APIs inherent to Node.js
- Anything that lives between the communication layer from your web client to the web server
- **Access to the request object (req) and the response object (res)**

Express

- Minimalistic middleware framework build atop Node.js
- APIs that abstract away Node.js low-level functionality for HTTP methods
- Makes it easy to create a robust API

Let’s make a Movie API!

Routing

- Need to define behavior for set endpoints for each HTTP method
- Express methods take care of this
- Routing syntax: **app.METHOD(PATH, HANDLER)**

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   res.send("Hello world!");
});
```

- app is an instance of express.
- METHOD is an HTTP request method, in lowercase.
- PATH is a path on the server.
- HANDLER is the function executed when the route is matched.

Routing

- Need to define behavior for set endpoints for each HTTP method
- Express methods take care of this
- Routing syntax: **app.METHOD(PATH, HANDLER)**

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   res.send("Hello world!");
});
```

- The **res.send()** function basically sends the HTTP response.

Running a Server Locally

.listen() function

- Creates and starts a server
- Listens for connections (requests) on a specified port

Port Number

Function that executes when the server starts running

```
app.listen(3000, () => {
        console.log("Listening on Port 3000”);
})
```

Running a Server Locally

Go to a browser and type in **http://localhost:3000/**. This is the “url” of your server. localhost represents your current computer that is running the server. When the page loads, it should display the “Hello world!”

```
app.get('/', function(req, res) {
    res.send("Hello world!")
})

app.listen(3000, function() {
    console.log("Server is listening on port 3000...");
})
```

Route Parameters

- Route parameters are segments of the URL that capture values inputted by the client
  - Parameter names follow a colon
  - Captured values are put in the **req.params** object

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

```
app.get('/users/:userId/books/:bookId', (req, res) => {
  res.send(req.params)
})
```

Express Response Methods

The **res** object has a lot of built-in methods!

res.send(“Hello World!”)	Sends the HTTP response → “Hello World!”
res.json({body goes here})	Converts body to JSON, sends JSON response
res.sendStatus(200)	Sets the response HTTP status code to the parameter and sends the status as the response → “OK!”

```
const express = require('express')
const app = express()
const port = 3000

app.get('/users', (req, res) => {
  res.send('Hello World!')
})

app.get('/users/:id', function(req, res){
   res.send('The id you specified is ' + req.params.id);
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Demo from lecture:

EXPLORER

LECTU...

- node_modules
- index.js
- package-lock.json
- package.json

index.js

index.js > ...

```javascript
const express = require('express')
const app = express()

app.get('/', (req, res) => {
    res.send("This is a basic API call")
})

app.get('/movie/:title/director/:name', (req, res) => {
    res.send("You have retrieved the movie "+req.params.title+" and director "+req.params.name)
})

app.listen(3000, () => {
    console.log("Listening on Port 3000")
})
```

Example request URL: http://localhost:3000/movie/jaws/director/spielberg

Complex routing

- How do we “modularize” our routing?
  - Example: Separate routes for users, posts, comments
    - /api/users, /api/users/:id, /api/users/random
    - /api/posts, /api/posts/:id, /api/posts/top
    - /api/comments, /api/comments/:id

index.js

```javascript
const express = require("express");
const app = express();
const userRoute = require("./routes/users");

...
app.use("/api/users", userRoute);
...
```

users.js

```javascript
const express = require("express")
const router = express.Router()

router.get("/:id", (req, res) => {
    res.send("You've accessed users!") // actually "/api/users/:id"
})

module.exports = router;
```

[end recording]

Attendance: Lecture 6

https://forms.gle/LAAZ28LAEzEcpfP59

Secret word: