Code Fellows

Class 06
HTTP and REST

seattle-javascript-401n14

Lab 05 Review

Code Challenge 05 Review

Vocab Review!

What is a linked list?

What is a linked list?

A linked list is an ordered collection of data, where each item in the list contains a **reference** or **pointer** to the next item in the list. Unlike an array which has a set length and uses an index to navigate, linked lists have no set length and are only navigated by individual items pointing to the next item.

What is a circular linked list?

What is a circular linked list?

A circular linked list is a small modification on a standard linked list; instead of having the last node in the list point to `null`, we have the last node in the list point to the `head`. This creates a loop and makes certain traversals a little bit faster.

What is NoSQL?

What is NoSQL?

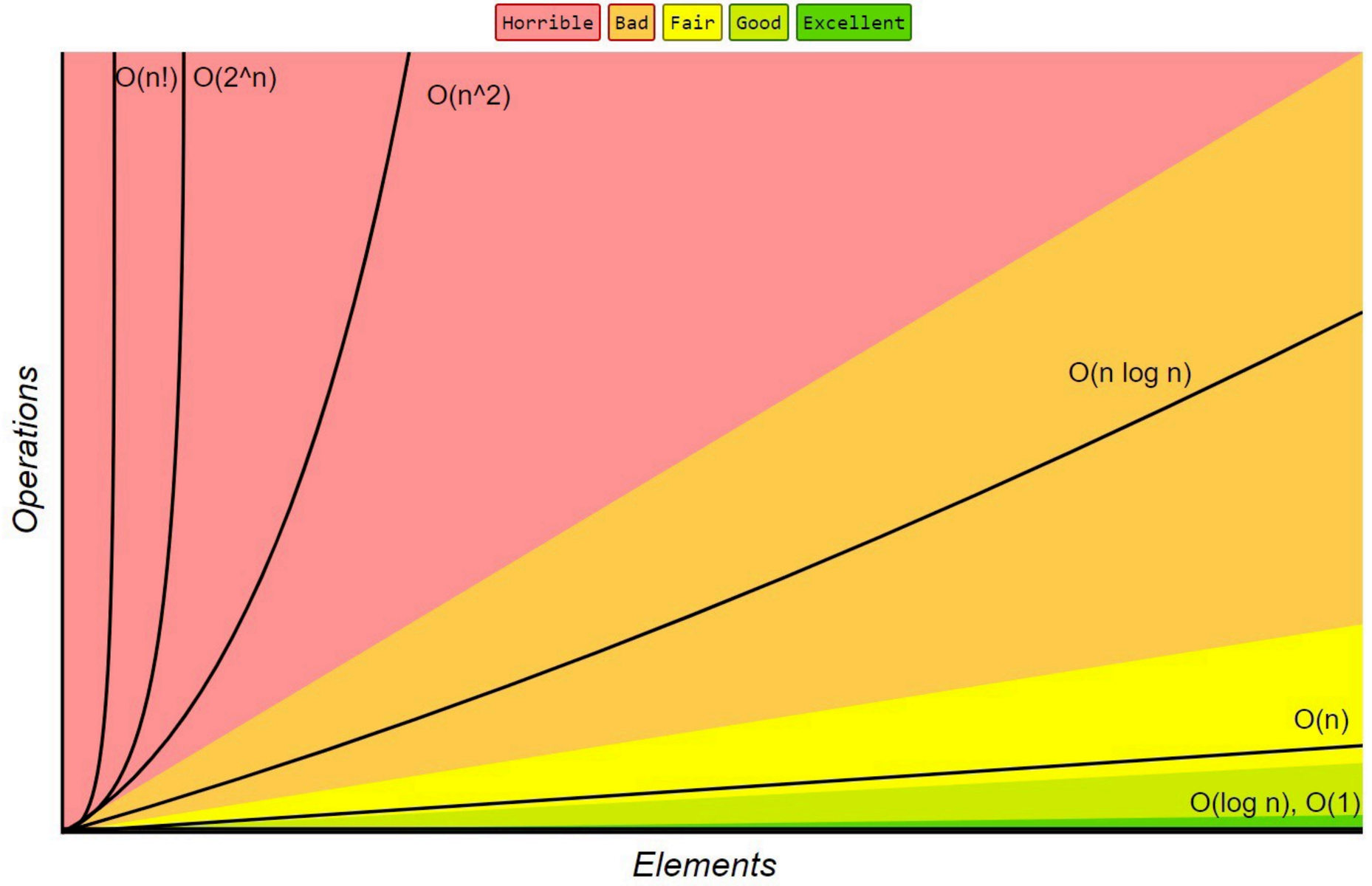
NoSQL stands for “non SQL” or “non relational”, and it refers to databases which model their data as a collection of documents (raw data files) instead of categorizing all data into strictly defined tables. NoSQL allows for more flexible and customizable data storage.

What is big-o?

What is big-o?

We use big-o notation to describe the **worst-case behavior** of a function. Big-o describes how the time and space of a function grows in relation to how the size of the input (`n`) grows. Standard growth rates are constant (`1`), logarithmic (`log n` or `n log n`), 1:1 (`n`), squared (n^2), exponential (2^n) and factorial (`n!`)

Big-O Complexity Chart



What is a server?

What is a server?

A server is an application whose purpose is to provide data or expose APIs to another application, called the **client**. A single server can have multiple clients, and a single client can use multiple servers. We created a MongoDB server in Lab 05.

What is HTTP?

What is HTTP?

HTTP stands for **HyperText Transfer Protocol**, and it is how data is shared across the internet. HTTP is not just the beginning of most web URLs, it is also a format for making data requests. For example, every website is just a path from where our browser then loads data onto our screens.

HTTP and HTTPS

- When we start our urls with **HTTP** and **HTTPS**, what are we actually saying?
- Similar to the terminal command `mongod --dbpath=""`, using `http` or `https` tells our browser to "open up a connection" to some location, defined by our **URL**
    - URL stands for **Uniform Resource Locator**
- `https` is a more secure connection than `http`

http://localhost:3000/path/to/something?name='Sonia'

https://www.facebook.com/soniaKandah?ref=bookmarks

Protocol URL/Host [Port] [Path] [Query]

HTTP and CRUD

- We've been "requesting data" in our labs using our `Model.js` and CRUD operations
    - We requested data from a file in Lab 04
    - We requested data from a MongoDB server in Lab 05
    - Now let's request data from a **web server**
- We can still use our `Model.js` and CRUD operations, they just map to something a little different!

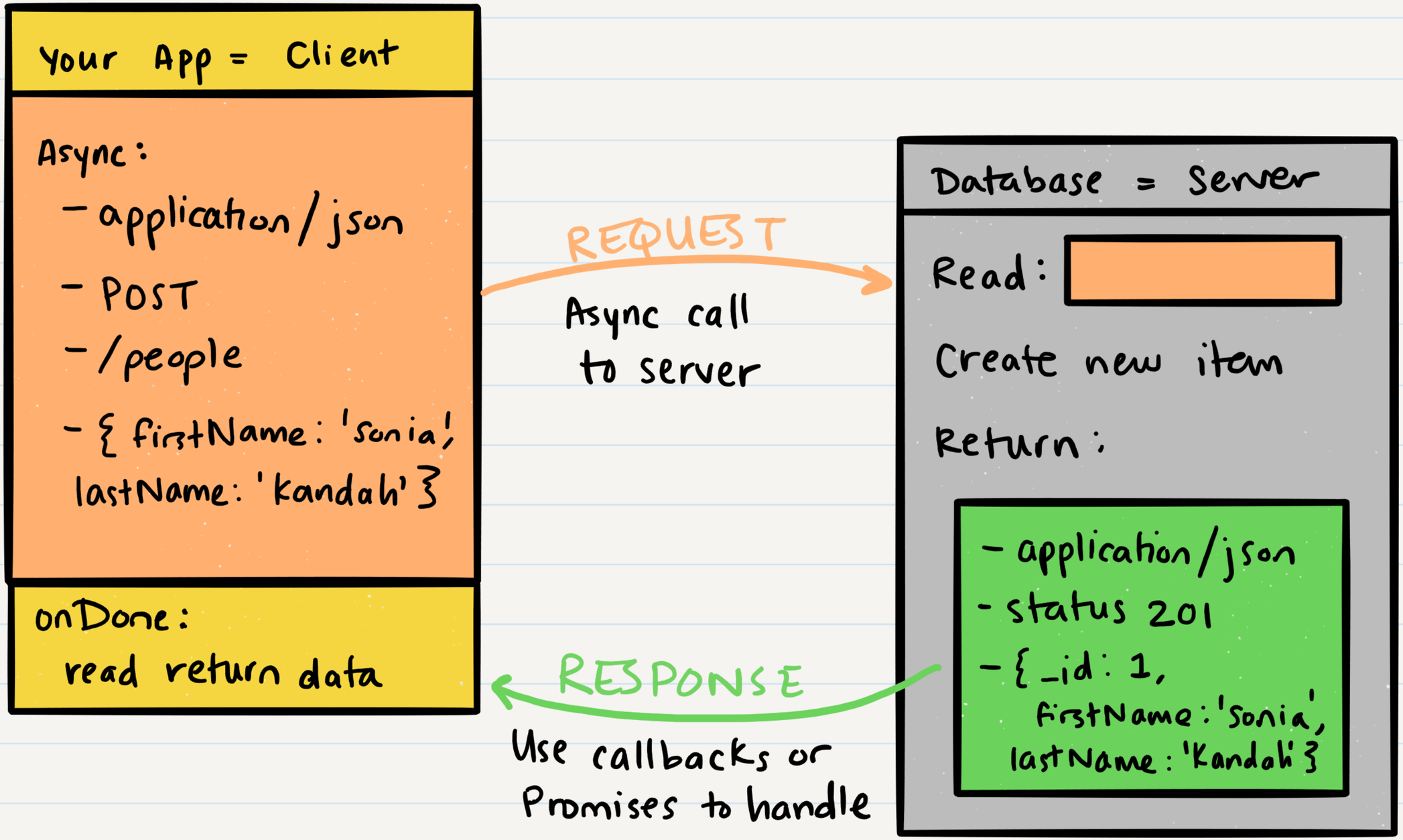
CRUD Operations	Using a file	Using MongoDB	Using HTTP
Create	`writeFile`	`.save()`	`POST`
Read / Get	`readFile`	`find()` `findOne()`	`GET`
Update	`writeFile`	`findByIdAndUpdate()` `updateOne()`	`PUT`
Delete	`writeFile`	`findByIdAndDelete()` `deleteOne()`	`DELETE`

More Acronyms: REST

- **REpresentational State Transfer** is a set of rules that you should follow to communicate with other systems on the internet. REST defines rules for both **clients** and **servers**
- Clients send **requests** to servers. These requests contain:
  - HTTP CRUD operation (`GET`, `PUT`, `POST`, `DELETE`)
  - A **header** which contains additional request data
  - A **path** to the thing on the server you want to access
  - A **body** containing any data the request needs

More Acronyms: REST

- Servers send **responses** to clients. These responses contain:
    - A response **type**, describing if it is sending a file, some JSON, an image, etc
    - A response **status code**, describing if the request was successful or returned an error
    - Any **data** that should be returned on a successful request


Your App = Client
Async:
- application/json
- POST
- /people
- { firstName: 'sonia',
lastName: 'kandah' }
onDone:
read return data
REQUEST
Async call
to server
RESPONSE
Use callbacks or
Promises to handle
Database = Server
Read:
Create new item
Return:
- application/json
- status 201
- { _id: 1,
firstName: 'sonia',
lastName: 'Kandah' }

```
soniakandah … class-06 lab _sol master http GET https://swapi.co/api/people/1/
HTTP/1.1 200 OK
Allow: GET, HEAD, OPTIONS
CF-Cache-Status: DYNAMIC
CF-RAY: 5265d7107992c991-SEA
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/json
Date: Wed, 16 Oct 2019 00:15:45 GMT
Etag: W/"145c70f4eca80b4752674d42e5bf1bcf"
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Set-Cookie: __cfduid=d1c1261246660256d820586ddf989b15a1571184944; expires=Thu, 15-Oct-20 00:15:44 GMT
re
Transfer-Encoding: chunked
Vary: Accept, Cookie
Via: 1.1 vegur
X-Frame-Options: SAMEORIGIN

{
    "birth_year": "19BBY",
    "created": "2014-12-09T13:50:51.644000Z",
    "edited": "2014-12-20T21:17:56.891000Z",
    "eye_color": "blue",
    "films": [
        "https://swapi.co/api/films/2/",
        "https://swapi.co/api/films/6/",
        "https://swapi.co/api/films/3/",
        "https://swapi.co/api/films/1/",
```

Demo

HTTPie - GET

You can make requests to REST APIs right from your terminal! We use HTTPie to make it quick and easy.

Let's test things out using a publicly available API, the Star Wars API.

Status	Meaning
200	Generic Success
201	Created Successfully
202	Accepted the request, but it might take a while to complete so here’s an empty return
204	Successfully completed, but not sending back any data

Status	Meaning
1xx	Generic Information
2xx	Success
3xx	Request needs updating / changing
4xx	Client Error (Bad Request)
5xx	Server Error

Requesting in Our Code

- We can write a request in our JavaScript applications using some helper modules
- We're going to use the module node-fetch, but there are other options out there:
  - Built-in (no package installation needed) option: `http` and `https` modules.

    However, it only uses callbacks, not Promises!
  - `request` module - improvement on https but still no Promises
  - `axios`, `SuperAgent` and `node-fetch` are all Promise based!

Easy to GET, Hard to POST

- We usually can’t PUT, POST or DELETE from databases that we don’t own

```
{
    "detail": "Method 'POST' not allowed."
}
```

- Let’s try to make our own database so we can mess around with it!

JSON-Server Makes it Easy

- The node package `json-server` makes it effortless to create a database running on a server. All you need is a `json` file with the data! It creates the `GET`, `PUT`, `POST`, `DELETE` endpoints for you!
- Install it **globally**:

  ```
  npm install -g json-server
  ```

- Connect it to a database file and run:

  ```
  json-server --watch=./data/db.json
  ```

```
soniakandah … class-06 demo json-server master json-server --watch ./data/db.json

\{^_^}/ hi!

Loading ./data/db.json
Done

Resources

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...
```

Demo

demo/json-server

Not only are we going to make our own server, we're also going to write a client application that will interact with our server!

Swagger - Generate API Docs

- Using a nifty tool called Swagger Inspector you can not only make API requests, but you can also generate documentation on that API
- For Lab 06, keep your Swagger Generated Docs in your `/docs/` folder

What's Next:

- Tomorrow: **Deploying MongoDB and JSON-Server to Heroku**
- Due by Midnight tomorrow: **Learning Journal 06**
- Due by Midnight Thursday: **Code Challenge 06**
- Due by 9am Saturday:
    - **Lab 06**
    - **Read: Class 07**
- Next Class:
    - **Class 07 - Express**
    - **Lightning Talk - Liskov Substitution Principle [??]**

Questions?