WEEK 7

Introduction to Working with Data on the Web

DATA VISUALIZATION

Digital storytelling at the confluence of science, art, and technology

CHECK IN

How is everything going? What challenges are you having?

- Due next week: Interactive Website
  - Use HTML/CSS/Javascript, RShiny, Observable, or Jupyter Notebooks to create an interactive web interface where anyone can interact with your data. Use this as an opportunity to submit something you would like feedback on for your final project.
- Due in two weeks: Final Projects
  - Fill out google form with title and abstract for website
  - Create and host a webpage which describes your process (including the code you used to make it), displays your data visualization, and guides the reader through it.

OPEN STUDIOS : MEDIA LAB VIZ WALL

- Library will host our final visualizations in a virtual gallery and when we return to campus- on this large visualization wall.
- Library has offered to support us in developing our final projects into an exhibit. If you need software or equipment, please tell me so I can connect you with someone to help.

AGENDA FOR TODAY

Learning about how to manage data online:

Working with Data on the Web

Guest presentation by Chris Lowrie

Data on the web

Alternatives for hosting and accessing data online

WHAT IS A WEBSITE?

A website is really just a folder of HTML files with instructions for what graphics and text the browser should display on the screen.

A server “serves” those files to anyone browsing your website.

WAYS TO HOST AND REFERENCE DATA?

1) Small datasets: write directly into your code as objects like Jasmine showed us

```
let portfolio = [
    {artist_src: "https://news.ucsc.edu/2019/07/images/kendall-bar-jessica.jpg", artist: "Jessica Kendall-Bar", project: "Elephant Seal Animation"},
    {artist_src: "https://news.ucsc.edu/2019/07/images/kendall-bar-jessica.jpg", artist: "Jessica Kendall-Bar", project: "Elephant Seal Animation"},
    {artist_src: "https://news.ucsc.edu/2019/07/images/kendall-bar-jessica.jpg", artist: "Jessica Kendall-Bar", project: "Elephant Seal Animation"},
    {artist_src: "https://news.ucsc.edu/2019/07/images/kendall-bar-jessica.jpg", artist: "Jessica Kendall-Bar", project: "Elephant Seal Animation"},
  ]
```

WAYS TO HOST AND REFERENCE DATA?

2) Larger datasets: host source .CSV files on your site's file server (may run into space limitations)

- Github example (individual files limited to 100MB)
- Netlify example (storage limited to 100GB- not bad!)

Or…..

```
// Now fetching Elephant Seal Census data from a CSV
// Data cleaned from https://datadryad.org/stash/dataset/doi:10.7291/D1PP47

async function getData() {
    const response = await fetch("/data/Elephant Seal Census Data_smaller.csv");
    const census_data = await response.text();
    // console.log(census_data); // Preview data
    // Now parsing the data with separators
    const table = census_data.split('\n').slice(1);
    // Using \n to show us where each new line is and taking out first row
    // then write .forEach loop
    table.forEach(row => {
        const columns = row.split(',');
        const Census_ID = columns[0]; // Naming columns
        const Observer = columns[1];
        const agesexclass = columns[9];
        const population = columns[11];
        xlabels.push(Census_ID);
        pop_estimate.push(population);
        console.log(Census_ID,Observer);
    });
};

getData()
    .then(response => {
    console.log('yay data') // Celebrating successful data input
}).catch(error => { // Catching errors
    console.log("Error with data input! See below.") // Printing message to show error
    console.error(error); // Showing error message
});
```

Fetching csv stored locally in relative file path

Parsing CSV manually (there are also library parse functions)

DATA FROM AN API

API: Application Programming Interface

- A part of a program's server which receives requests and sends responses

- As a user, you can request information and data from these APIs, but you have to be using very specific syntax and follow certain rules
  - APIs may limit the rate or quantity of requests allowed

- Example: a weather service may have an API to provide weather data to its clients.

MAKE A REQUEST

```
fetch('https://example.com', {
  credentials: 'include'
});
```

You request data from the API's URL, and once you get a response, you can display or manipulate that response.

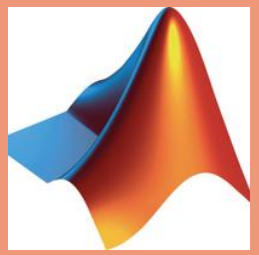
You can use Javascript's "async" and "await" functions to handle these requests and responses (see right).

```
// Example POST method implementation:
async function postData(url = '', data = {}) {
    // Default options are marked with *
    const response = await fetch(url, {
      method: 'POST', // *GET, POST, PUT, DELETE, etc.
      mode: 'cors', // no-cors, *cors, same-origin
      cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
      credentials: 'same-origin', // include, *same-origin, omit
      headers: {
        'Content-Type': 'application/json'
        // 'Content-Type': 'application/x-www-form-urlencoded',
      },
      redirect: 'follow', // manual, *follow, error
      referrerPolicy: 'no-referrer', // no-referrer, *no-referrer-when-
downgrade, origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-
when-cross-origin, unsafe-url
      body: JSON.stringify(data) // body data type must match "Content-Type" header
    });
    return response.json(); // parses JSON response into native JavaScript objects
  }

  postData('https://example.com/answer', { answer: 42 })
    .then(data => {
      console.log(data); // JSON data parsed by `data.json()` call
    });
```

WHAT SHOULD WE USE?

Exploring workflows in R, Javascript, and Python.

MATLAB (not free): High-level multi-paradigm programming language and interactive environment for numerical computation, visualization, and programming



R Studio

Install packages with RSTUDIO CONSOLE

EDIT & RUN CODE:

RSTUDIO

PUBLISH & SHARE CODE

RMARKDOWN

LET PEOPLE INTERACT:

RSHINY

JS

Observable: Include libraries by:

```
d3=require('d3');
```

HTML: Include libraries inline

```
<script src="link"></script>
```

EDIT & RUN CODE:

VISUAL STUDIO CODE

with HTML, CSS, JAVASCRIPT

PUBLISH & SHARE CODE

OBSERVABLE

LET PEOPLE INTERACT:

PUBLISH WEBPAGE

Install packages through package manager like ANACONDA or in your terminal:

```
conda install pip
```

EDIT & RUN CODE:

SPYDER

through ANACONDA

PUBLISH & SHARE CODE

JUPYTER NOTEBOOK

LET PEOPLE INTERACT:

PUBLISH WEBPAGE

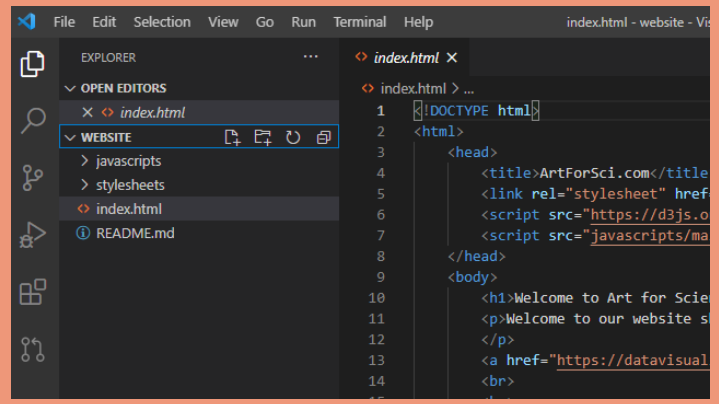
OUR WEB WORKFLOW

Local → Remote

VISUAL STUDIO CODE

Create HTML, CSS, & Javascript files.

Preview changes in browser with Live Server Extension.

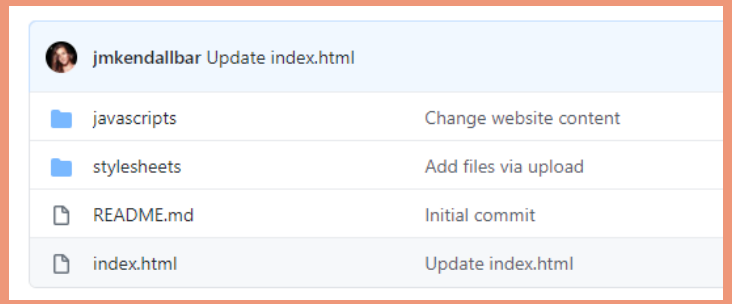


GITHUB

Make a GitHub account.

Create a repository called "Website"

Drop your website files in there.



NETLIFY

Deploy your website from your GitHub repository.

`https://www.netlify.com/`

GOOGLE DOMAINS

Buy your own domain and link it!

`https://domains.google.com/`

API USAGE AND DATA MANAGEMENT

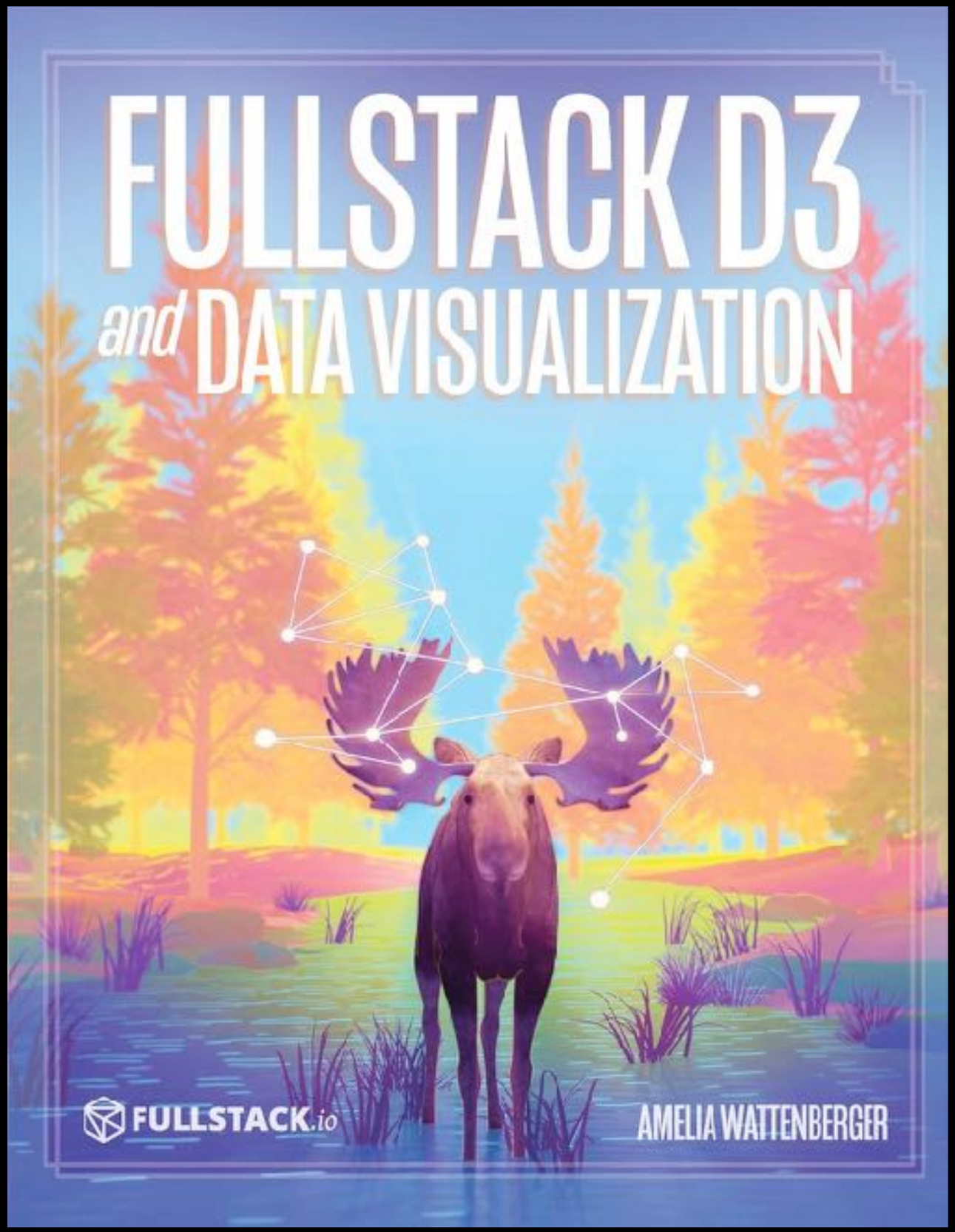
ABOUT ME

- GIS, Computational Mathematics, and Climate Science and Adaptation
- Previous work:
    - Apple Maps, GIS Lead and Project Manager,
    - Columbia/NASA working with flash flood data.

TOOLS I USE

- Python: all things GIS + numerical analysis + plotting and mapping
- Javascript: Node, Express, React, D3
- R: data mining, geostatistics
- SQL: PostGRES, DB administration, a various “Big Data” things

FULLSTACK D3 and DATA VISUALIZATION



FULLSTACK.io

AMELIA WATTENBERGER

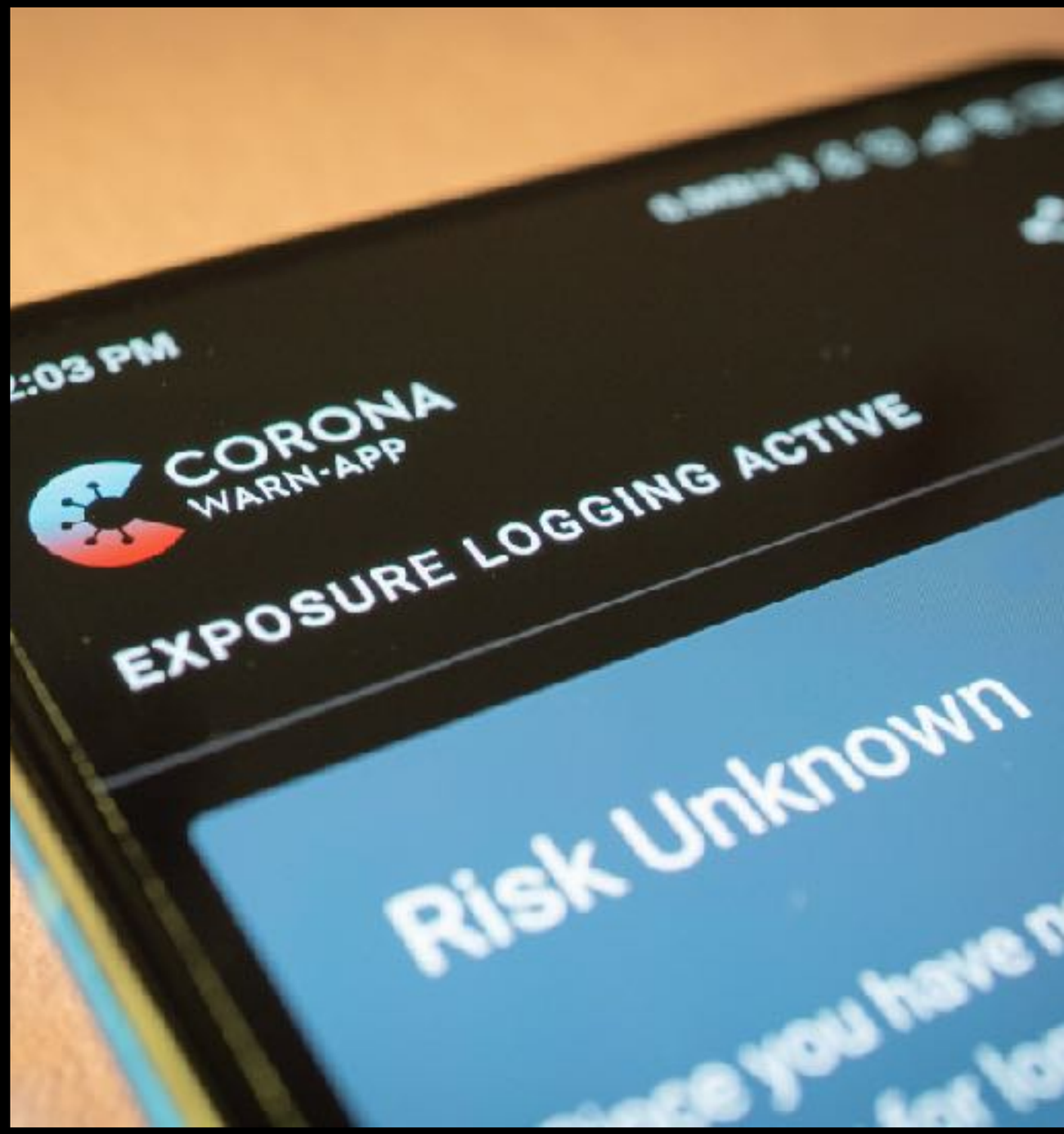
React IN ACTION

Mark Tielens Thomas

MANNING

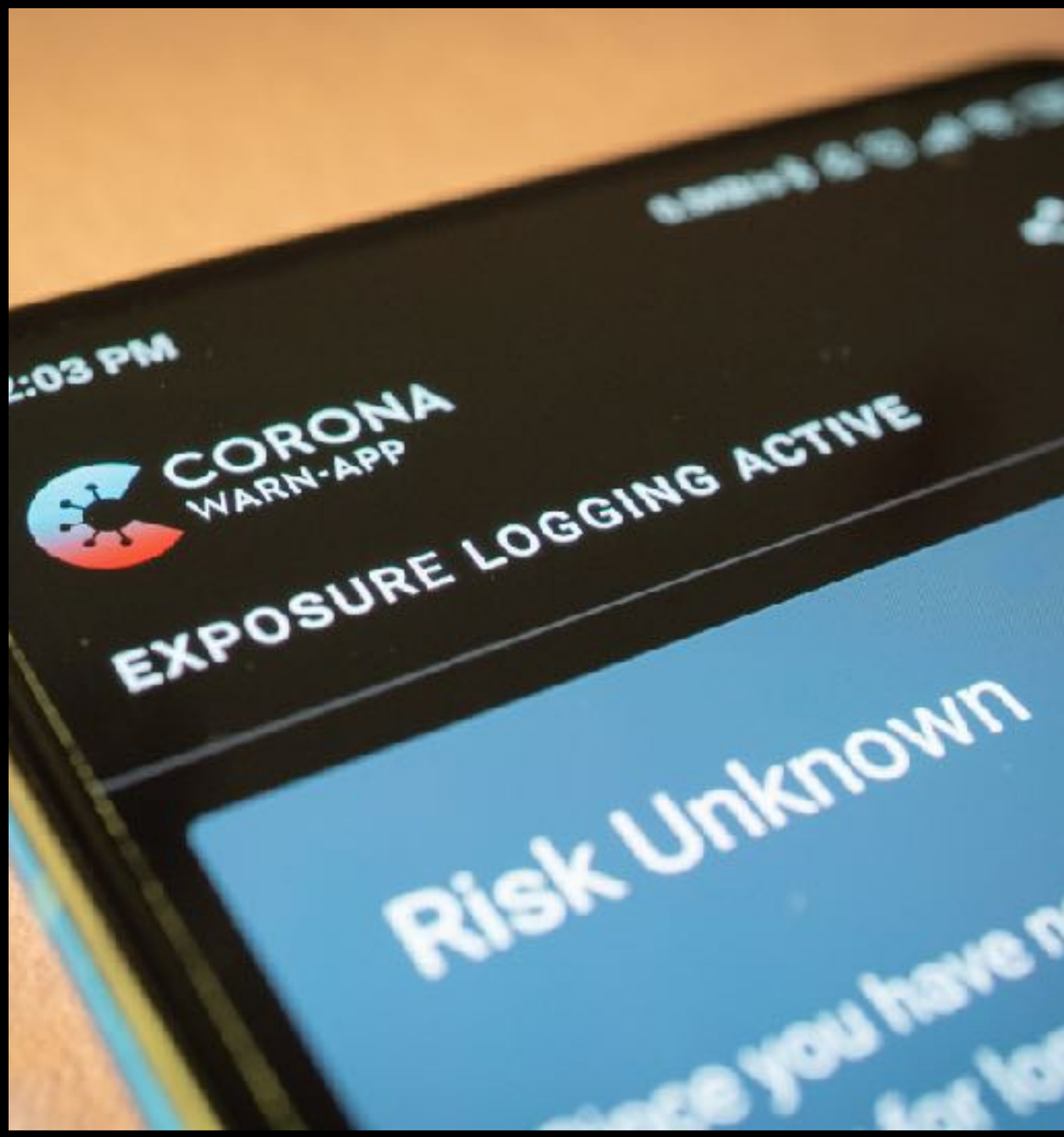
WHAT IS AN API?

- “Application Programming Interface”
- “a computing interface that defines interactions between multiple software intermediaries”
- Basically, it’s the way for people to interact with data, and for code running from different programs and machines to interact.



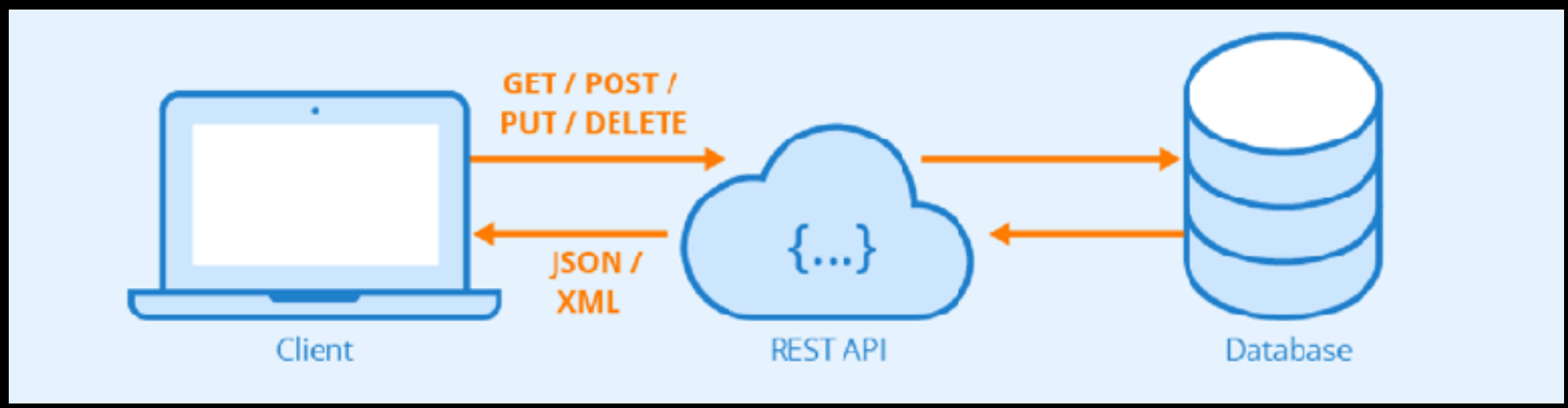
HOW DOES SOMETHING LIKE THE CORONA APP WORK?

- Everybody has an ID
- When two people get close, each phone sends the other its ID
- IDs are stored for two-ish weeks
- When someone gets sick, their ID is tagged as “sick”
- Every other phone that stores that ID learns this, and notifies accordingly



WHAT IS A REST API?

- A way for clients to communicate with servers using HTTP(S)
- HTTP is a set of criteria and protocols that servers abide by so that programs can interface with them
- REST is a type of Web API with specific design elements, but it's also by far the most common Web API and is somewhat used interchangeably



FRONTEND

- Browser
- HTML, CSS, JS (React)

- OR -

LOCAL DATA ANALYSIS

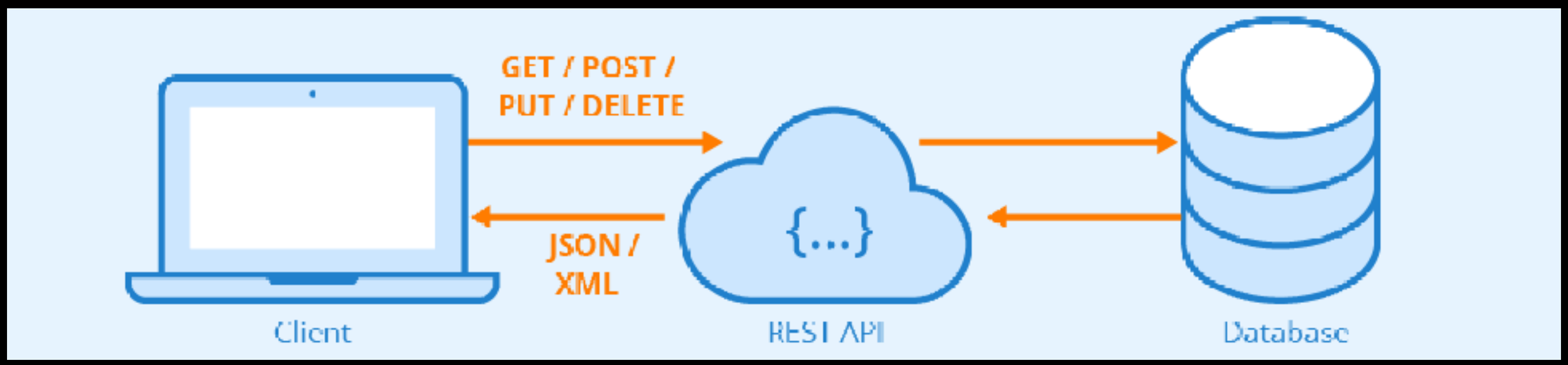
- Python: Requests
- Node

SERVER, "MIDDLEWARE"

- Node: Express
- Python: Flask & Django

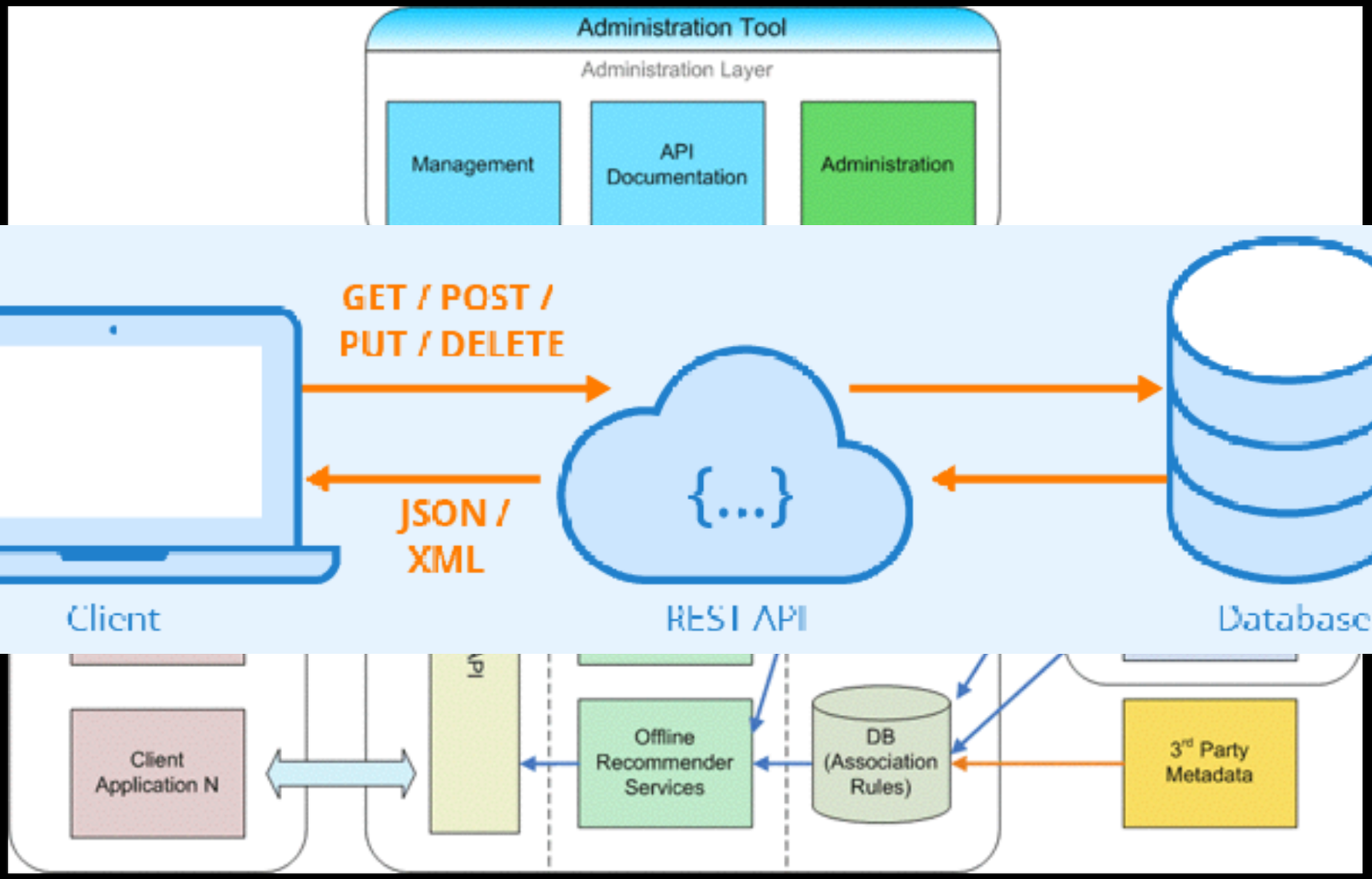
DATABASE

- Google Cloud & AWS
- RDBMs: PostGRES, SQLite, MySQL
- Graph DBs



ARCHITECTURE

- Architecture refers to the overall layout of an application, and the flow of data through that application



TWO BASIC USE CASES FOR LEARNING APIS

1. Consuming / visiting an API to get data

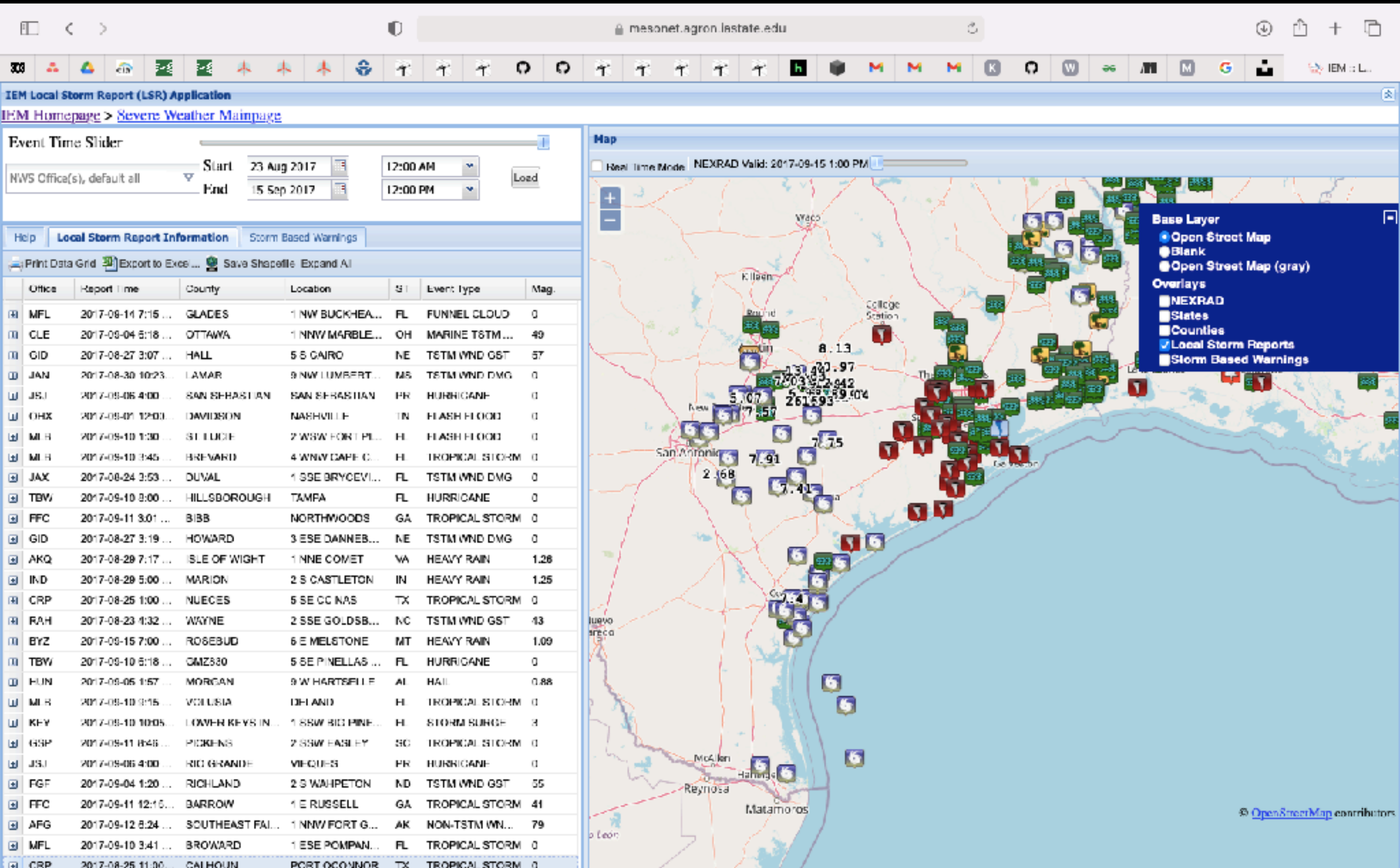
2. Creating an API with endpoints to distribute data, including to a web application

- Because browsers are the world's way to get things onto your local computer, a lot of file system access is tightly managed. Endpoints help with this.

DATA WRANGLING EXAMPLE

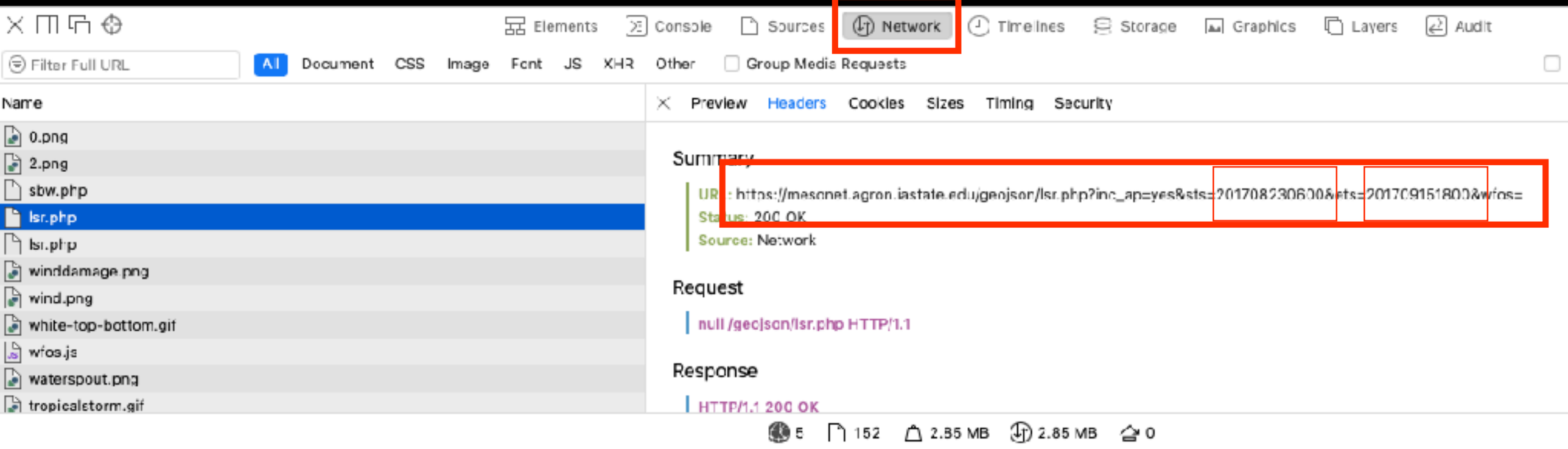
PYTHON REQUESTS, API CONSUMPTION

IOWA ENVIRONMENTAL MESONET



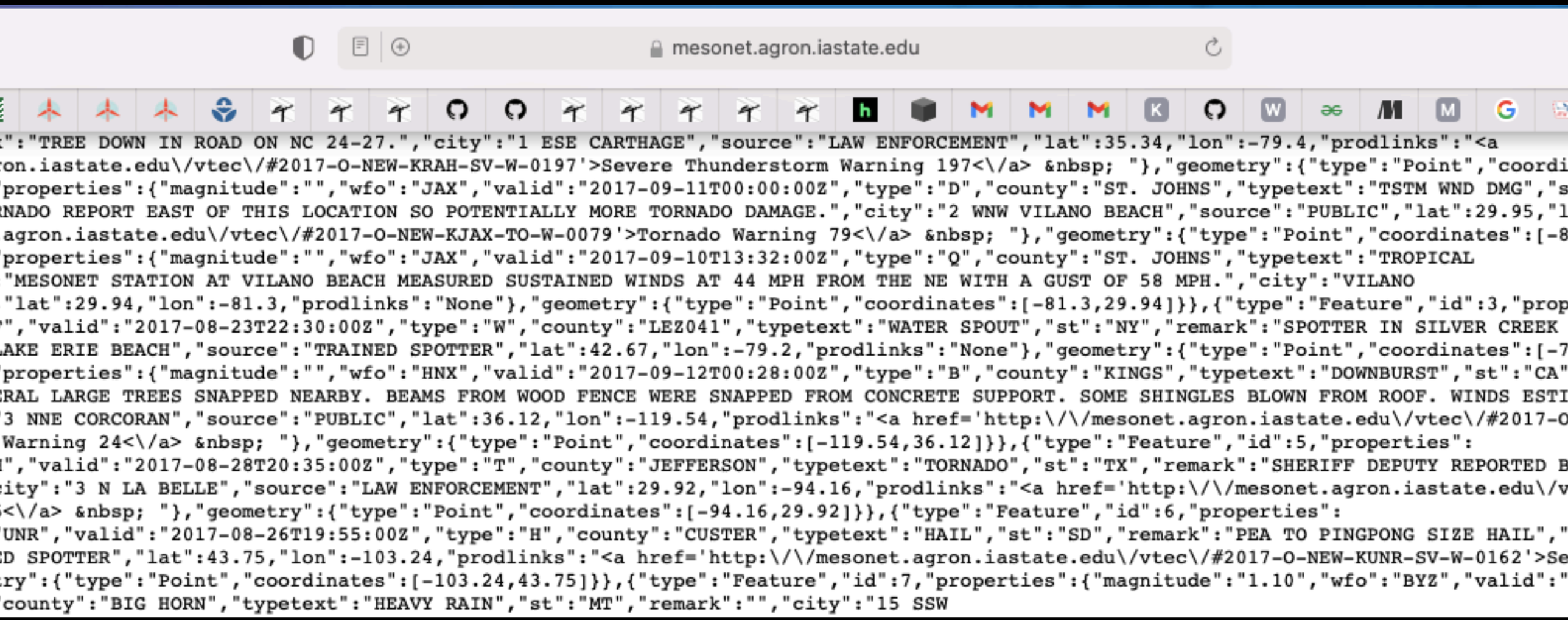
TIP: USE THE DEVELOPER CONSOLE

- Specifically, the network requests pane
- Open the pane, navigate around the webpage, and pay attention to what loads



FOLLOWING THE URL

- And you get a big, ugly JSON
- Which is exactly what we want

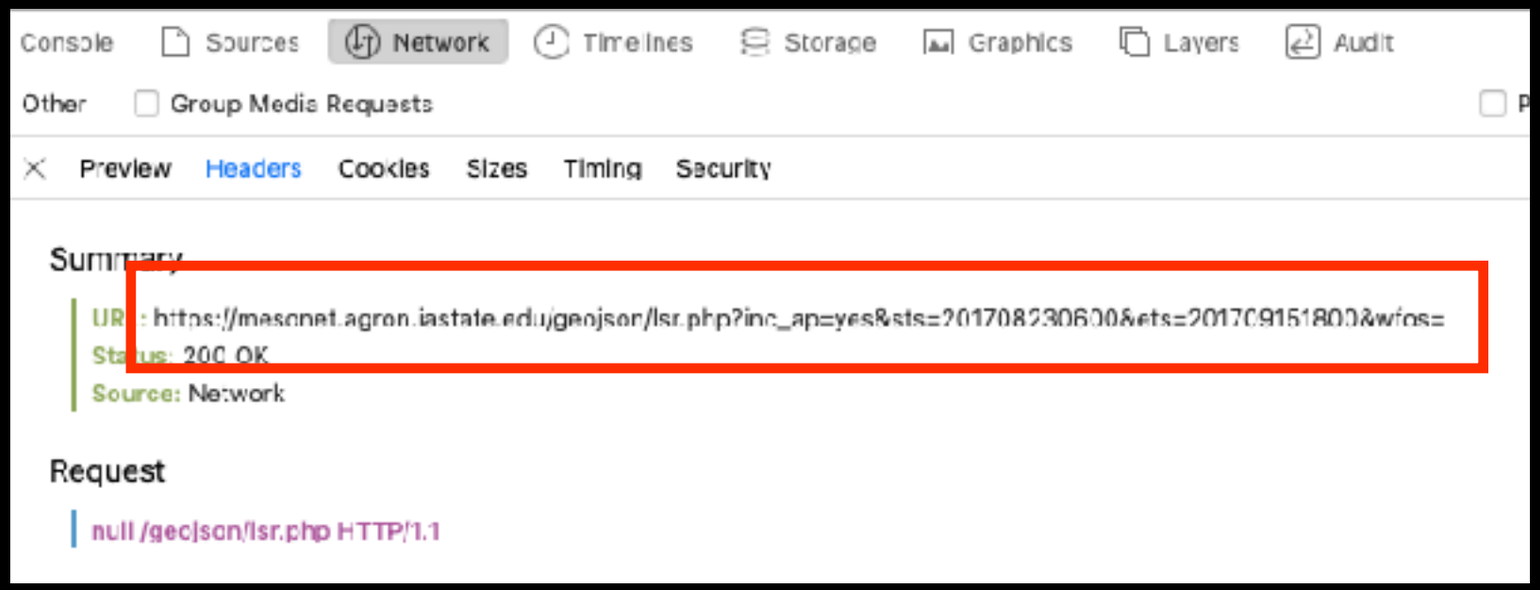


JSON

- When I say JSON, this is all I mean
- It's a key-value data format, similar to Python dictionaries and JS objects.
- It can store all primitive data types (strings, ints, floats, etc.)
- (This is actually a geojson, which is a JSON carrying geometry in a specific format)


```
{
  "type": "Feature",
  "id": 0,
  "properties": {
    "magnitude": "",
    "wfo": "RAH",
    "valid": "2017-09-01T21:00:00Z",
    "type": "D",
    "county": "MOORE",
    "typetext": "TSTM WND DMG",
    "st": "NC",
    "remark": "TREE DOWN IN ROAD ON NC 24-27.",
    "city": "1 ESE CARTHAGE",
    "source": "LAW ENFORCEMENT",
    "lat": 35.34,
    "lon": -79.4,
    "prodlinks": "<a href='http://mesonet.agron.iastate.edu/v
  },
  "geometry": {
    "type": "Point",
    "coordinates": [
      -79.4,
      35.34
    ]
  }
}
```

WE'VE ESTABLISHED:

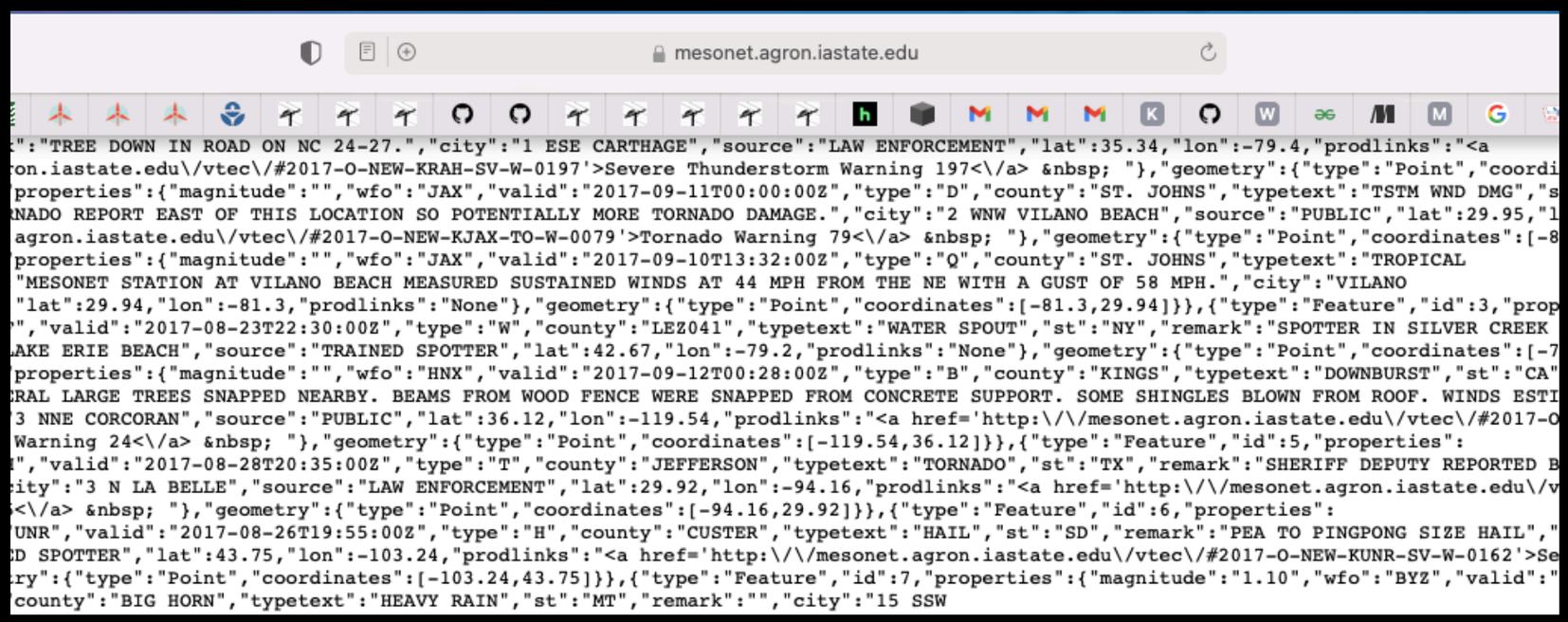


←Sending a request to this URL

Sending a request to ~= "going to"

Returns this JSON as a response →

And that's HTTP in action



VISITING HTTP IS THE SAME THING THAT HAPPENS WHEN YOU LOAD A WEBPAGE

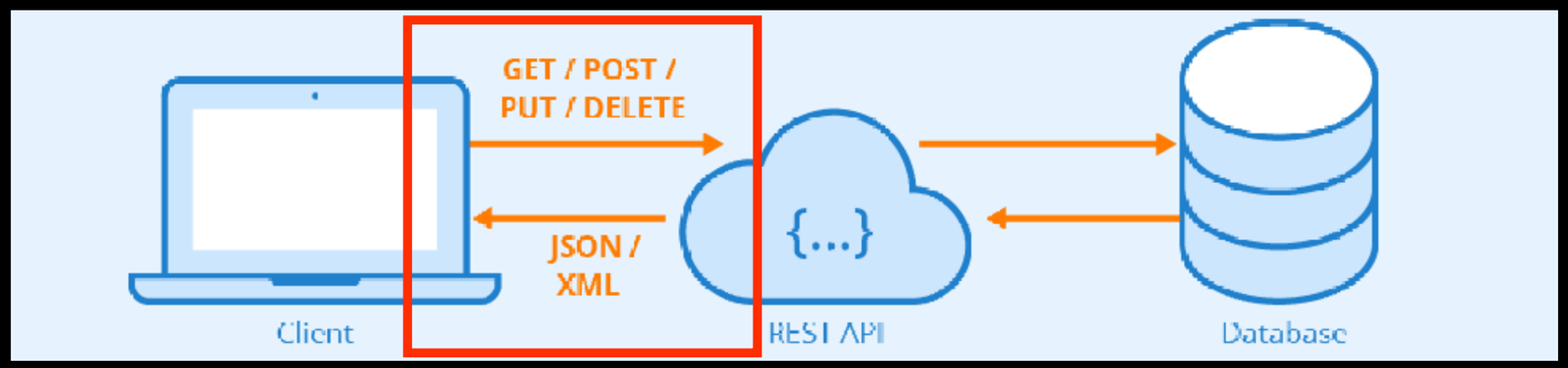
- React is what's called a "Single Page Application", or SPA
- Basically this means that the skeleton HTML is sent just once, and everything after that is handled by JS fetching data
- Go to URL → Domain Name and IP magic finds the server for you → server sends initial chunk of HTML/JS → page renders, JS takes over and updates the DOM as necessary
- The contrast of SPA is "server-side rendering". In this case, navigating through a website essentially triggers an entire new page to load.
    - Network traffic is by far the most expensive aspect of application loading. SPAs seek to minimize the data being sent.
- In both cases, servers are just sending data to the client.
    - HTML files are data, Javascript files are data, JSON and XML are data too.

WHAT DO YOU ATTACH AS FILES IN THE BUILD VS LOAD AFTER-THE-FACT?

- General rule of thumb:
    - Anything your app needs to get started should be attached
    - Things that require interaction and may not be loaded every time should be fetched from a server
- If your data is static and relatively small, it’s okay to just load it in the build

REQUESTS AND RESPONSES

- The internet communicates (almost entirely) in JSON and XML
- Clients send HTTP(S) requests to servers, which have query strings and bodies and the servers send responses with JSON and XML
    - These requests can fetch (GET), add (POST), update (PUT), or delete data
- JSON is dominant, but XML has some use cases.
    - I mostly see it in tile-based applications, such as web map servers.
- Think of it as going to a library.
    - Client: "give me the data you have about armadillos published between 2012 and 2014"
    - Server: "here's a bunch of data about armadillos with metadata"



PYTHON REQUESTS

- Python contributors have abstracted out the headache of making HTTP requests with the requests module
- Supply a URL, and you'll get data back, which you can use or write to a local file.

```
    def get_remote_data(self):
        """Look for remote data.  Requires URL construction in child class."""
        import requests
        file_path = self.get_local_path()
        with open(file_path, "wb") as file:
            r = requests.get(self.construct_url())
            file.write(r.content)
```

JAVASCRIPT FETCH

- Vanilla Javascript has a similar concept, but it differs slightly because (1) JS is so predominant in web-based applications and (2) JS is *asynchronous*
- This snippet comes from React, which is fetching a z value from a local elevation database based on a user input (we'll revisit in a moment)

```
getZ = (lon, lat) => {
  console.log(lon, lat)
  fetch('http://localhost:3000/x/' + lon + '/y/' + lat)
    .then(r => r.json())
    .then(z2 => this.setState({lon: lon, lat: lat, z: z2.z}))
  console.log(this.state.z)
}
```

O'REILLY

Seventh Edition

JavaScript

The Definitive Guide

Master the World's Most-Used Programming Language

David Flanagan

DATA FETCHING FOR WAZE

- IEM flood data, which we just saw, is a much more typical way of automating data fetching

- For Waze, my team was actually supplied Google Sheets of major storms throughout the US

- I won't go into too much detail on this since it's a somewhat atypical use case. Suffice it to say that data fetching can come from a multitude of sources.

- If you can find it on the internet, you can automate it

NORMALIZATION AND OOP

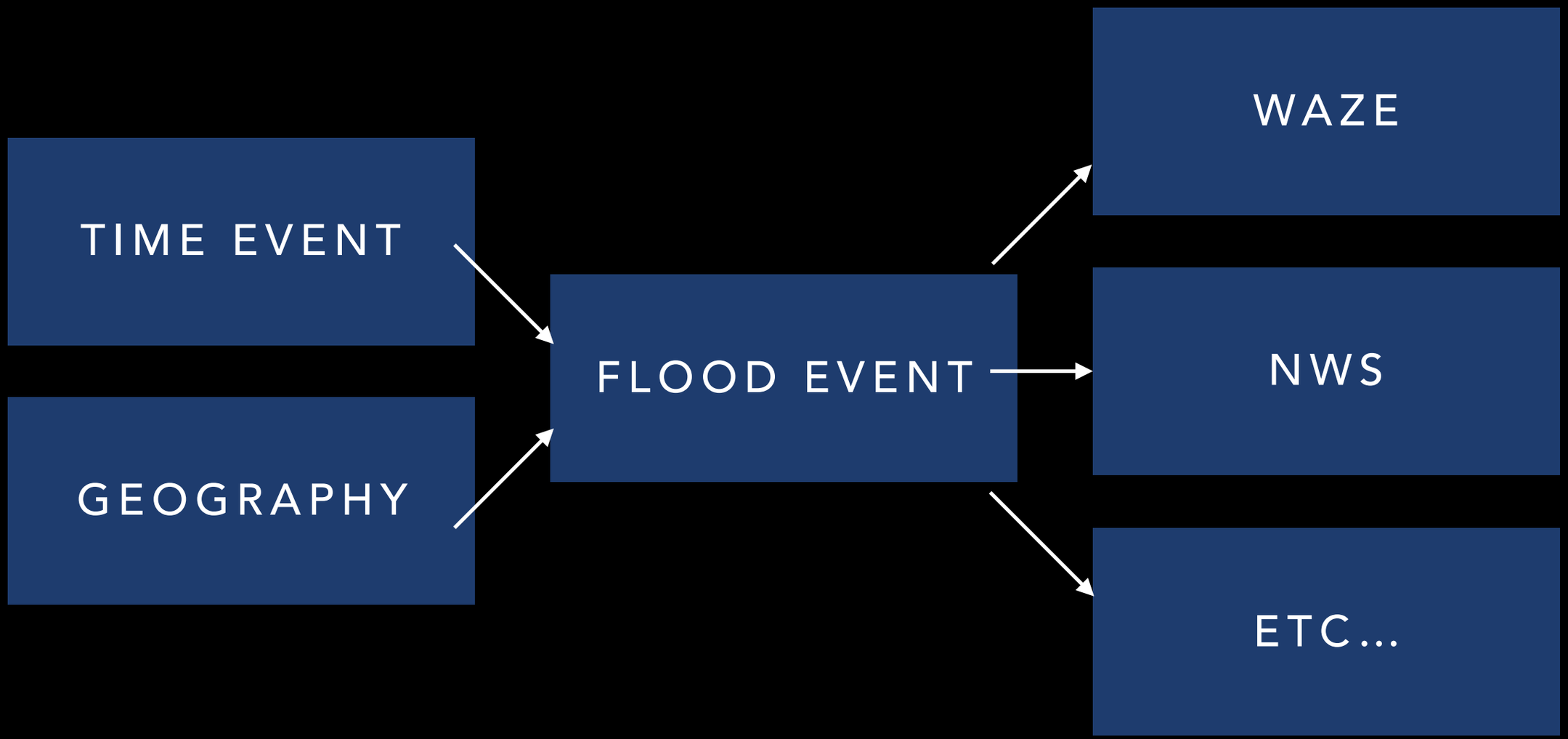
- So you've connected to data, now what?
- It usually needs to be reformatted to play nicely with your system, especially with dates, times, space/GIS, and categorical data
- Rule of thumb: spend more time planning and discussing than you do coding.
- Object Oriented Programming: separate out general functionality from specific data sources

```
class WazeHandler(AbstractGeoHandler, AbstractTimePointEvent, SpaceTimePointStatistics):
    t_field: str = "time"
    home_dir: str = config.waze

    def __init__(self, event_name):
        self.event_name = event_name
        AbstractGeoHandler.__init__(self, gdf=self.get_gdf())

    def get_gdf(self):
        """Get the Waze GDF from the .txt files pulled from Google Sheets"""
        from shapely.geometry import Point
        csv = os.path.join(self.home_dir, "waze_" + self.event_name + ".txt")
        df = pd.read_csv(csv)
        print(
        gdf = (
            df.
            cr
            ge
        )
        gdf["t
        return

    def prep_d
        self.g
        self.g
            lo
        )

    @staticmet
    def convert
        time =
        return
```

```
class IowaEnvironmentalMesonet:
    """A super class for managing connections with the Iowa Environmental Mesonet"""

    base_url: str = "https://mesonet.agron.iastate.edu/geojson/"
    file_type: str = ".geojson"
    t0: datetime = None
    t1: datetime = None

    def construct_url(self):
        """Construct a URL for fetching remote data"""
        times = self.times_as_string_tuple()
        t0 = "".join(times[0][0:3])
        t1 = "".join(times[1][0:3])
        print(self.base_url.format(t0=t0, t1=t1))
        return self.base_url.format(t0=t0, t1=t1)

    def times_as_string_tuple(self):
        """Fetch times as a string tuple"""
        return (
            [str(i).zfill(2) for i in self.t0.timetuple()],
            [str(i).zfill(2) for i in self.t1.timetuple()]
        )

    def construct_local_identifier(self):
        """Construct an identifier to save files locally and reduce network traffic"""
        times = self.times_as_string_tuple()
        t0 = "".join(times[0][0:5])
        t1 = "".join(times[1][0:5])
        return t0 + "_" + t1 + self.file_type

    @staticmethod
    def convert_numeric_to_datetime(x):
        """Convert the initial time storage format to datetime"""
        return datetime.strptime(x, '%Y-%m-%dT%H:%M:%S')
```

OBJECT ORIENTED PROGRAMMING

- In a nutshell, this means working with classes and objects to abstract functionality.
- The basic goal is to separate code so that it can be reused



CREATING AN API ENDPOINT EXAMPLE

NODE - EXPRESS JS

EXPRESS AND MIDDLEWARE

- Express is a framework for starting your own HTTP server
- Express also has an associated ecosystem of what's known as middleware - libraries that handle and pass-along requests, performing actions along the way
  - Static servers (good for serving files from a directory)
  - Session handling
  - Logging and Debugging
  - Site analytics

GOAL OF AN ENDPOINT

- Particular addresses of a REST API are known as endpoints.
- Endpoints allow the server to take input from the client and perform some action on the data

API CREATION USING EXPRESS

- Initial use case: you want to be able to dynamically access data stored on your local, within the browser, to use all the fun React + D3 tools
    - A collection of photos
    - A database with records that you want to filter for
- Extended use case:
    - Providing access to others to work with data you procure and manage

ENDPOINTS WITH EXPRESS

JS routes.js ● JS simple_server.js ×

TerrainModelling > server > JS simple_server.js > ...

```
1   const cors = require('cors')
2   const express = require('express')
3   const routes = require( './routes.js')
4   const app = express()
5   const port = 3000
6
7
8   app.use(cors())
9
10  app.get('/x/:x/y/:y', routes.get_z);
11
12  app.get('/california', routes.get_california)
13
14  app.get('/catchment/lon/:lon/lat/:lat', routes.get_flow_path
15
16  app.get('/qmt', routes.quant_mesh_test)
17
18  app.listen(port)
```

JS routes.js ●

TerrainModelling > server > JS routes.js > driver

```
1   const { Pool, Client } = require('pg');
2   const queries = require('./queries');
3   const neo4j = require('neo4j-driver')
4   const fs = require('fs')
5
6   const pool = new Pool({
7       user: 'christopherjlowrie',
8       host: 'localhost',
9       database: 'watersheds',
10      password: 'XXXXX',
11      port: 5432,
12    });
13
14  const driver = neo4j.driver(
15    'neo4j://localhost',
16    neo4j.auth.basic('neo4j', 'XXXXXX')
17  )
18  var session = driver.session({ defaultAccessMode: neo4j.sessi
19
20  function get_z(req, res) {
21    console.log(req.params);
22    const {x, y} = req.params;
23    console.log(x)
24    console.log(y)
25    queries.get_z
26      .then(q => pool.query(q, [x, y]))
27      .then((result) => {
28        console.log(result.rows[0]);
29        if (result.rows[0]) {res.json(result.rows[0])}
30        else {
31          console.log("Nothing Found")
32          res.json({"z": null})
33        }
34      })
35      .catch(e => console.log(e));
36  };
37
```

master* Python 3.7.4 64-bit ('venv': venv) ⊗ 0 ⚠ 0 Ln 16, Col 36 Spaces: 2 UTF-8 LF JavaScript ESLINT

API DEVELOPMENT PATTERN

- ENDPOINTS and ROUTES

```
app.get('/x/:x/y/:y', routes.get_z);
```

- Endpoints:
  - The address of the API

```
function get_z(req, res) {
```

```
if (result.rows[0]) {res.json(result.rows[0])}
else {
  console.log("Nothing Found")
  res.json({"z": null})
}
```

- Routes
  - Functions that return data, and correspond to endpoints
  - Takes a request and a response
  - Responds to the client through the response object

O'REILLY®

Second Edition

Web Development with Node & Express

Leveraging the JavaScript Stack

Ethan Brown

PYTHON VS NODE

- Node:
  - Asynchronous. Takes some getting used to, but overall more flexible
  - I find Express to be smoother and easier than either Flask or Django, the Python equivalents
  - Benefits of thinking about front and backend in the same language
- Python
  - Data processing. Python Pandas has better support than anything in Node
  - ML frameworks

BASIC ARCHITECTURE

Host	Frontend	Backend / Server	Database
aws	React	node.js + express python™ django	SHP CSV Etc

Haven't talked about this.
Not as intimidating as it may seem.

GIS WITH CODE

GIS IS JUST CODE

- Under the hood, Arc and QGIS are both just C++ and Python
- Underlying almost everything GIS is GDAL: Geospatial Data Abstraction Library, maintained by the Open Geospatial Consortium (OGC)
- My opinion:

1. “clicks” hide far too much GIS analysis and hinder reproducibility.
2. Anything that needs to be reproduced consistently should be in
    A. ArcPy / PyQGIS,
    B. model builder,
    C. or thoroughly documented.
3. GUI GIS is best suited for exploratory analysis, drafting maps for publish, and well understood, one-off tasks

GIS for Science, ESRI Press

WHAT IS GIS DATA

- A raster is basically a 2D array (at least, a single band raster) with metadata to support geolocation
  - Cell width, height
  - Bounding box
- A vector is an array of vertices:
  - i.e. “Polygon(x1 y1, x2 y2, …)”
- All GIS functions — clip, intersects, contains, etc — are essentially doing algebra (or some higher level math) on

PYTHON + GIS

- ArcPy
- PyQGIS
- Open Source:
    - Shapely, Fiona
    - Geopandas (which is just Shapely + Fiona + Pandas all rolled into one)
    - PySAL
    - Rasterio
    - Numpy, Matplotlib, etc etc

R has even better support for data science and geostatistics than Python. I don't favor it as much because I learned Python first and I like Python's overall ecosystem better.

Also, PostGIS. Arguably the best data storage format for GIS in existence.

Resources

JAVASCRIPT + GIS

- Leaflet
- OpenLayers
- D3
- Mapbox
- ArcGIS API
- WebGL
  - Mapbox GL JS
  - Deck GL
  - Kepler