Persistence & State

SWE 432, Fall 2016

Design and Implementation of Software for the Web

Today

- What's "state" for our web apps?
- How do we store it, where do we store it, and why there?

For further reading:

http://www.w3schools.com/html/html5_webstorage.asp

https://github.com/gmu-swe432/lecture15demos

https://www.npmjs.com/package/google-cloud

https://devcenter.heroku.com/articles/getting-started-with-nodejs

What's "State" in our web app?

Web App State

- Application state includes all of our data (not code)
- What kinds of data are we concerned about?
  - What user is logged in?
  - What interactions have they had with us before?
  - What data have they given us?
  - What data have others given us?
- Where do we store all of these things?

State: Example

Amazon.com…

Home page | Login | Browse | Add to cart | visit amazon.com

Still logged in

Still logged in

Still logged in, still have cart…

Where do we save stuff?

- Many options of where we keep our data
- Where do we want to put it?
- How do we get it to where it needs to be?
- Goals:
  - Cost
  - Efficiency
  - Stability

Web “Front End”

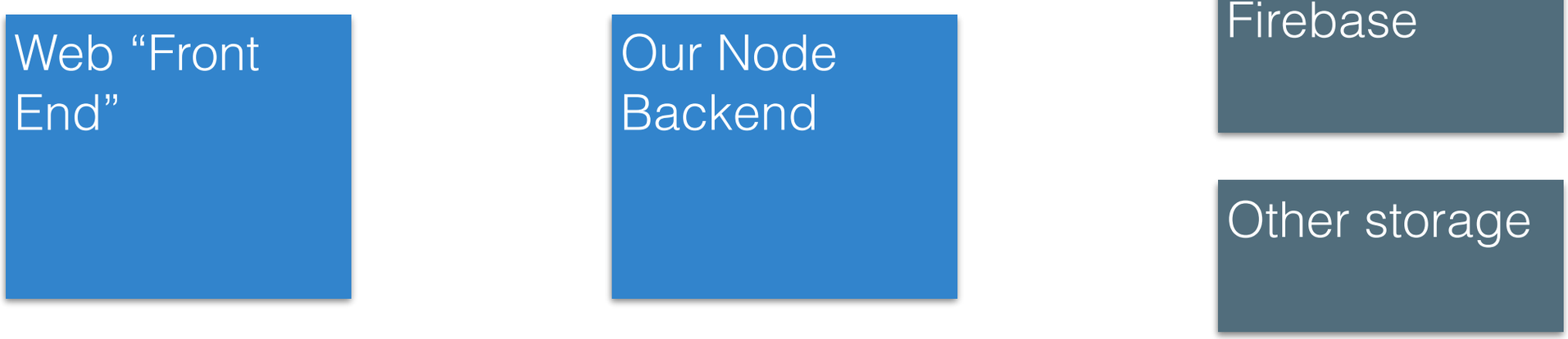
Our Node Backend

Firebase

Other storage

Where do we save stuff?

- Probably depends on how often we need to show it to the user, and how permanently we need to store it
- Examples:
  - What user is logged in? (Transient, relevant to user and backend)
  - What's in my shopping cart? (Semi-transient, relevant to user and backend)
  - What products am I looking at? (Transient, relevant to user)
  - What are all of the products (Long-term, parts are relevant to users)



Where do we save stuff?

- On client
  - Data we might need to show again soon
  - Fairly small (KB's or few MBs, not 100 MB's or GB's)
  - Data we don't care about going away or being maliciously manipulated
- In memory on backend
  - Data that we are working with that will fit in memory (MB's probably not GB's)
  - Transient data that can disappear if the server crashes
  - Cache or index of data stored externally
- On backend disk, database, or storage service(e.g., Firebase)
  - Data we need persisted "permanently"
  - Even if we'll be accessing it a lot, maybe we'll cache it somewhere so OK to pay performance penalty

Client Side State

- Original form of client state: Cookies
- Motivation:
  - We want to correlate multiple requests
  - But HTTP is *stateless*

Cookies

- String associated with a name/domain/path, stored at the browser
- Series of name-value pairs, interpreted by the web application
- Create in HTTP response with *"Set-Cookie: "*
- In all subsequent requests to this site, until cookie's expiration, the client sends the HTTP header *"Cookie: "*
- Often have an expiration (otherwise expire when browser closed)
- Various technical, privacy and security issues
  - Inconsistent state after using "back" button, third-party cookies, cross-site scripting, …

Maintaining Client Side State

Web “Front End”

Server “Back End”

HTTP Request

```
HTTP GET http://api.wunderground.com/api/
3bee87321900cf14/conditions/q/VA/Fairfax.json
```

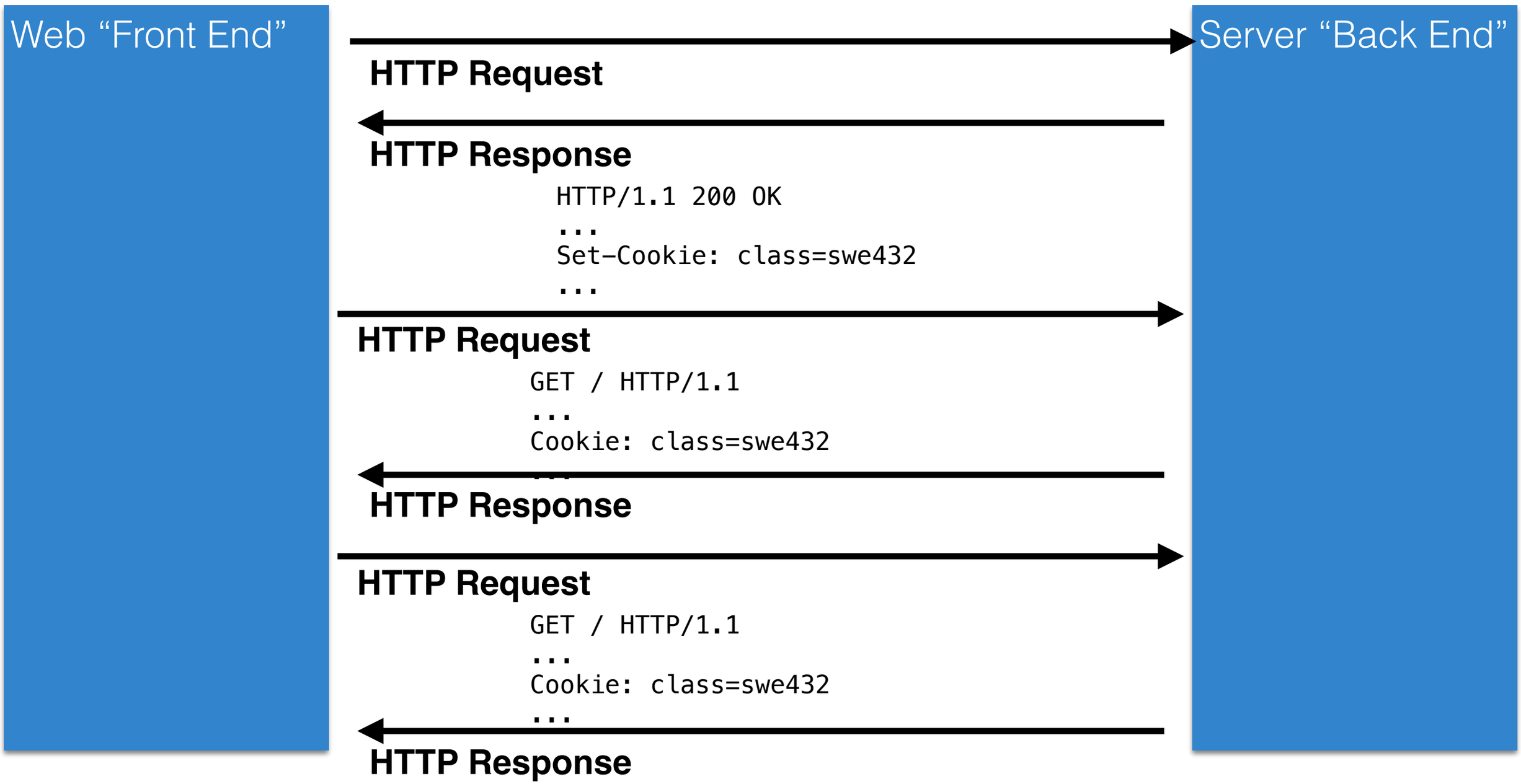
How do we track request-response pairs?

HTTP Response

```
HTTP/1.1 200 OK
Server: Apache/2.2.15 (CentOS)
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-CreationTime: 0.134
Last-Modified: Mon, 19 Sep 2016 17:37:52 GMT
Content-Type: application/json; charset=UTF-8
Expires: Mon, 19 Sep 2016 17:38:42 GMT
Cache-Control: max-age=0, no-cache
Pragma: no-cache
Date: Mon, 19 Sep 2016 17:38:42 GMT
Content-Length: 2589
Connection: keep-alive


{
  "response": {
  "version":"0.1",
  "termsofService":"http://www.wunderground.com/weather/api/d/terms.html",
```

Cookies and Requests



Cookies & NodeJS

- Use the cookieParser module
- Stateful Hello World:

```
var express = require('express');
var cookieParser = require('cookie-parser');

var app = express();
var port = process.env.port || 3000;
app.use(cookieParser());
app.get('/', function (req, res) {
    if(req.cookies.helloSent == "true")
        res.send("I already said hello to you!");
    else
        res.cookie("helloSent","true").send('Hello World!');
});

app.listen(port, function () {
    console.log('Example app listening on port' + port);
});
```

- Can see cookies in Chrome under “Privacy”

Cookies Demo

- https://github.com/gmu-swe432/lecture15demos/tree/master/cookieshello

More complex state on frontend

- The most cookies you can have: 4KB (TOTAL per DOMAIN)
- Old solution:
  - Cookie is a key to some data stored on server
  - When client makes a request, server always includes this “extra data” being stored on server
- What’s wrong with this old solution?
  - Really slow - have to repetitively pass this same data back and forth

LocalStorage

- Hooray, HTML5:
  `localStorage` (Sticks around forever)
  `sessionStorage` (Sticks around until tab is closed)
  - And two functions:

```
setItem(“key","value");
getItem(“key”);
```

```
var id = localStorage.getItem(“userID”);
```

- Can store any string
- All pages in the same domain see the same localStorage and sessionStorage
- Alternatively: SQLite (SQL DB) that you can use in JS…

Demo: LocalStorage

https://github.com/gmu-swe432/lecture15demos/tree/master/localstoragetodos

Keeping State on the Backend

Node and State

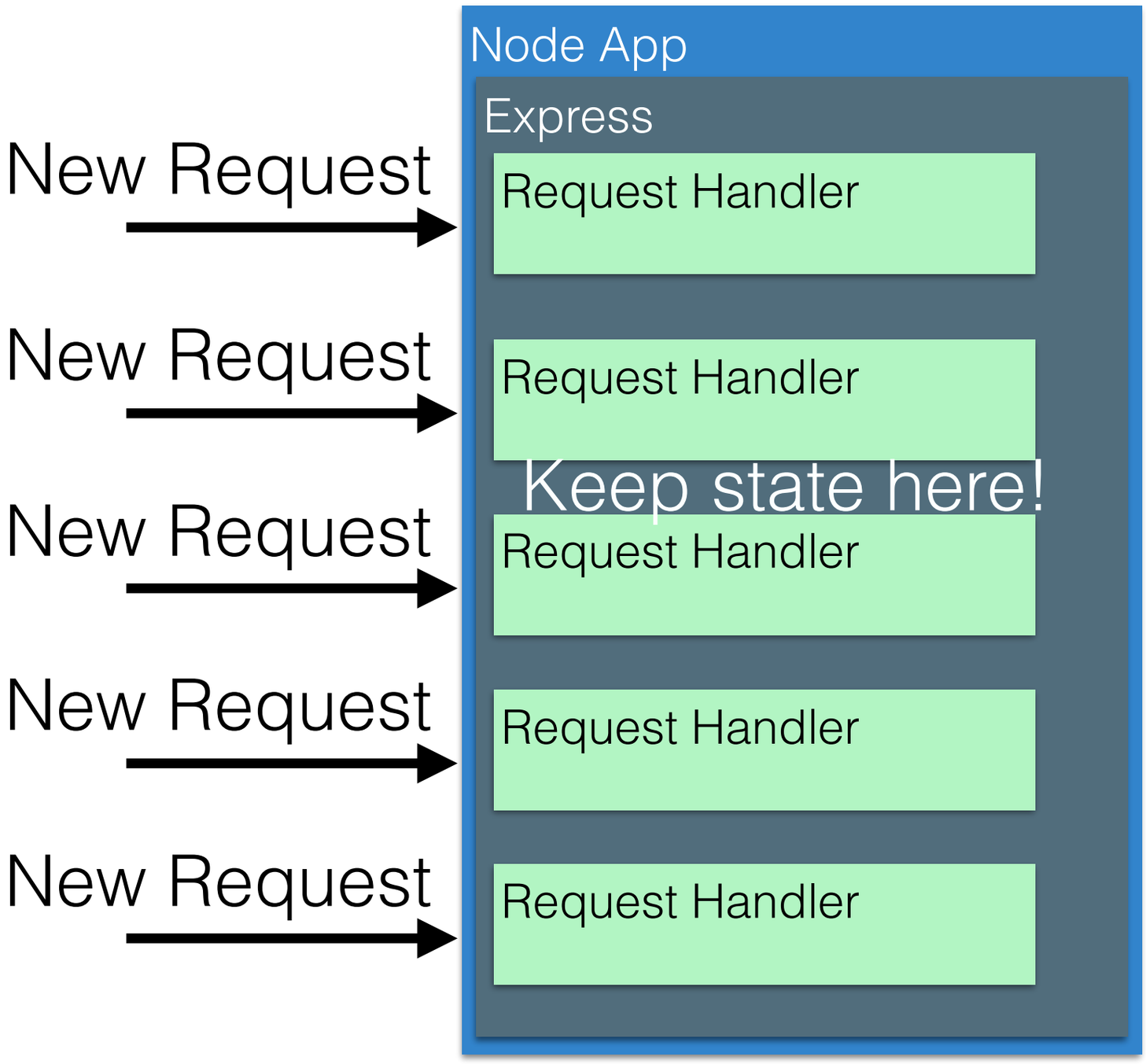
- Remember what a node route listener looks like…

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

- Each time a request comes in, a new callback runs
- How do we keep track of things?
- Well…

Recall: Node Architecture

Each new request goes to a new request handler



While the server is running though, it's all one app handling all requests

Keeping State in Node

- **Global variables**

```
var express = require('express');
var app = express();
var port = process.env.port || 3000;

var counter = 0;
app.get('/', function (req, res) {
    res.send('Hello World has been said ' + counter + ' times!');
    counter++;
});

app.listen(port, function () {
    console.log('Example app listening on port' + port);
});
```

- Pros/cons?
  - Keep data between requests
  - **Goes away** when your server stops
    - Should use for transient state or as cache

Demo: Statefull hello

- https://github.com/gmu-swe432/lecture15demos/tree/master/statefulhello

The Bigger Backend State Space

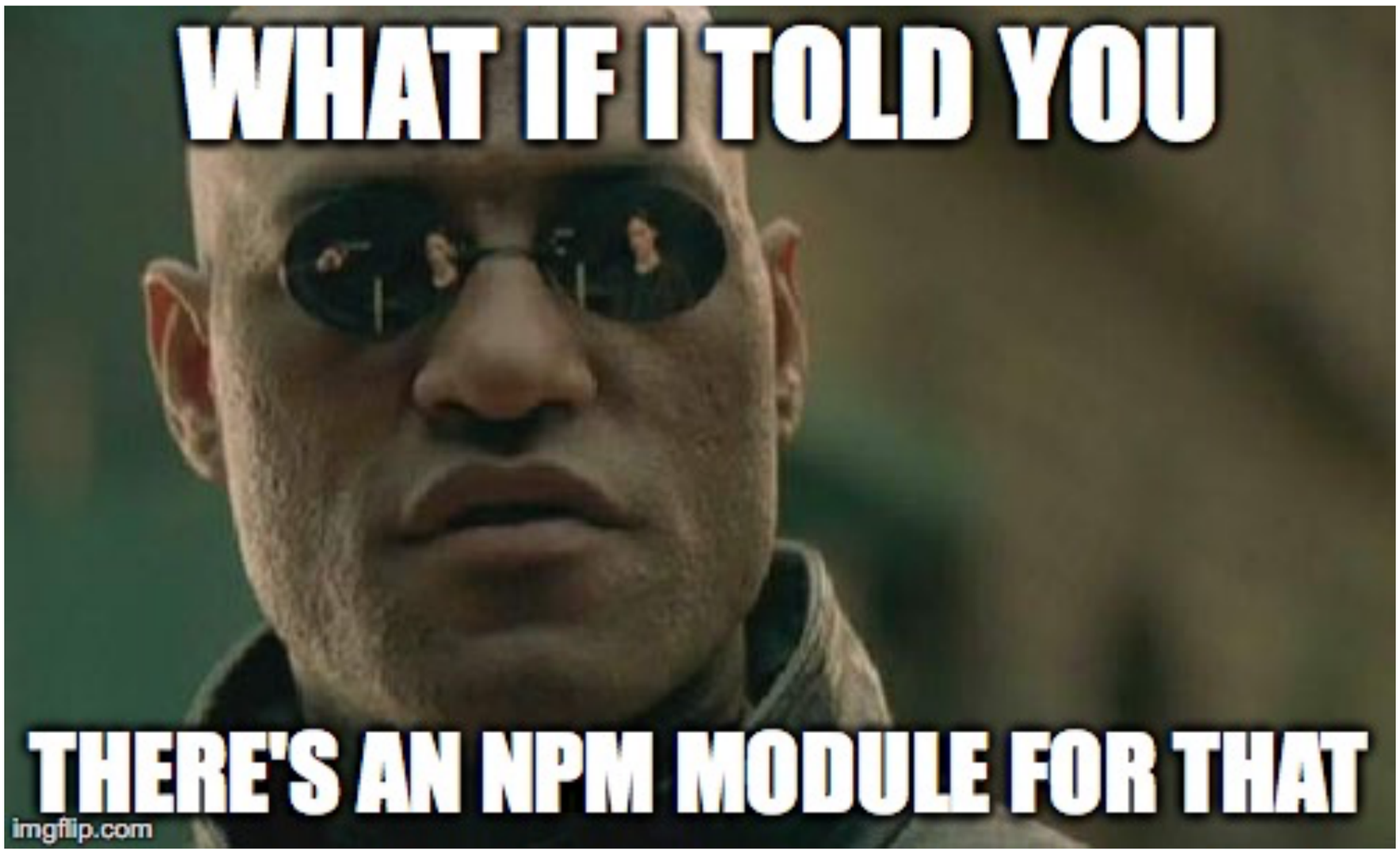
- Databases
  - SQL: MySQL, PostgreSQL, SQL Server, …
  - NoSQL: Firebase, Mongo, …
  - Reference: RESTful todos
- Files
  - Store arbitrary files on disk
    - JSON
    - Pictures, etc
  - Even better: blob stores

How do we store our files?

- Dealing with text is easy - we already figured out firebase
  - Could use other databases too… but that's another class!
- But
  - What about pictures?
  - What about movies?
  - What about big huge text files?
- Aka…Binary Large OBject (BLOB)
  - Collection of binary data stored as a single entity
  - Generic terms for an entity that is array of byte

Blobs: Storing uploaded files

- Example: User uploads picture
  - … and then?
  - … somehow process the file?



Working with Blobs

- Module: express-fileupload
  - Long story... can't use body-parser when you are taking files
- Simplest case: take a file, save it on the server

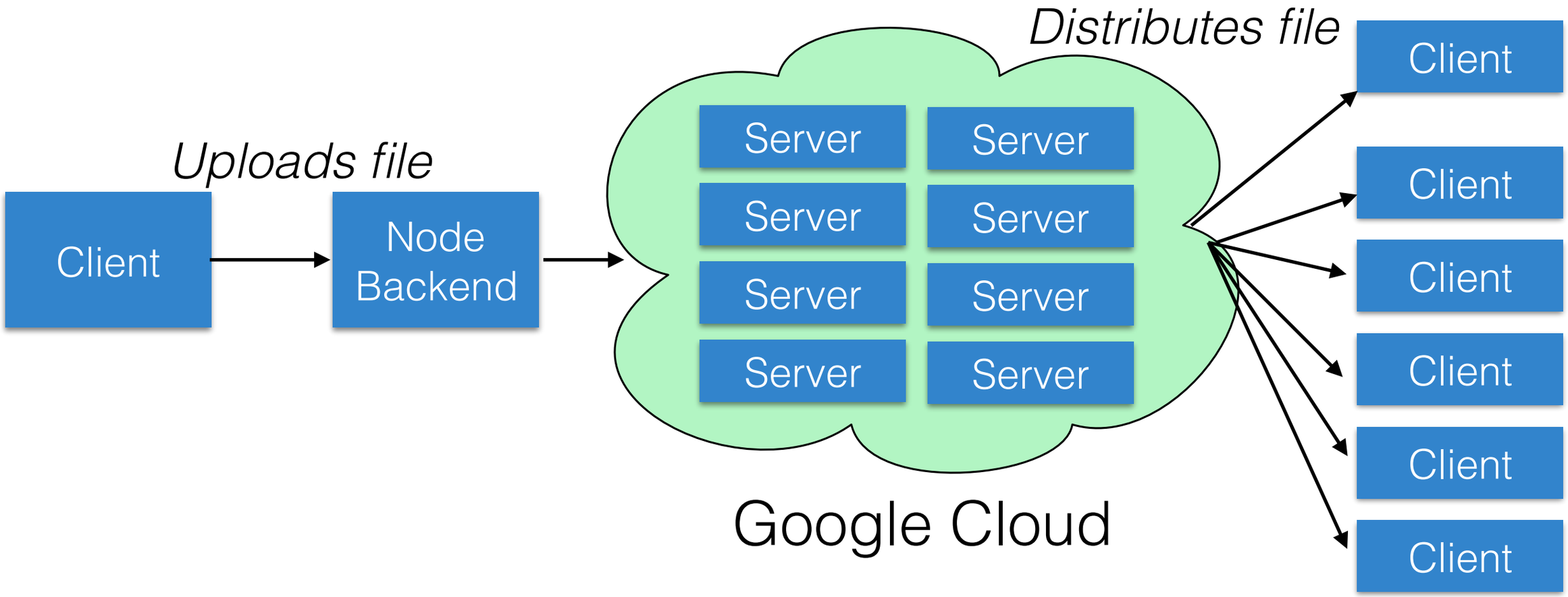
```
app.post('/upload', function(req, res) {
    var sampleFile;
    sampleFile = req.files.sampleFile;
    sampleFile.mv('/somewhere/on/your/server/filename.jpg', function(err) {
        if (err) {
            res.status(500).send(err);
        }
        else {
            res.send('File uploaded!');
        }
    });
});
```

Where to store blobs

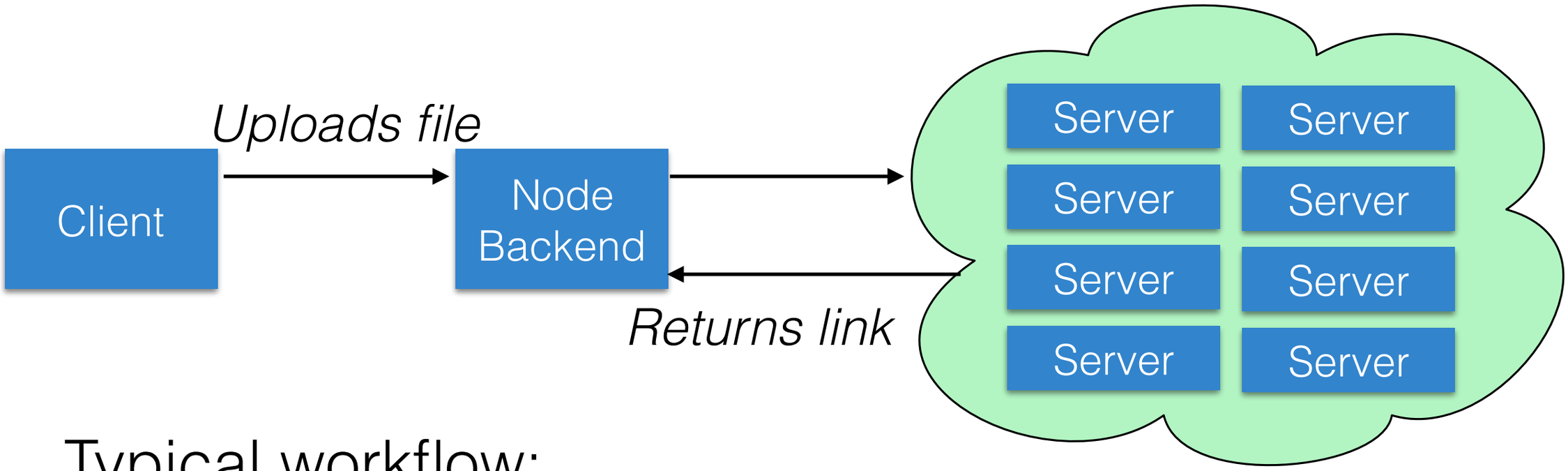
- Saving them on our server is fine, but…
  - What if we don't want to deal with making sure we have enough storage
  - What if we don't want to deal with backing up those files
  - What if our app has too many requests for one server and state needs to be shared between load-balanced servers
  - What if we want someone else to deal with administering a server

Blob stores

- Amazon, Google, and others want to let you use their platform to solve this!



Blob Stores



Google Cloud

Typical workflow:
nt uploads file to your backend
kend persists file to blob store
ckend saves link to file, e.g. i

Google Cloud Storage

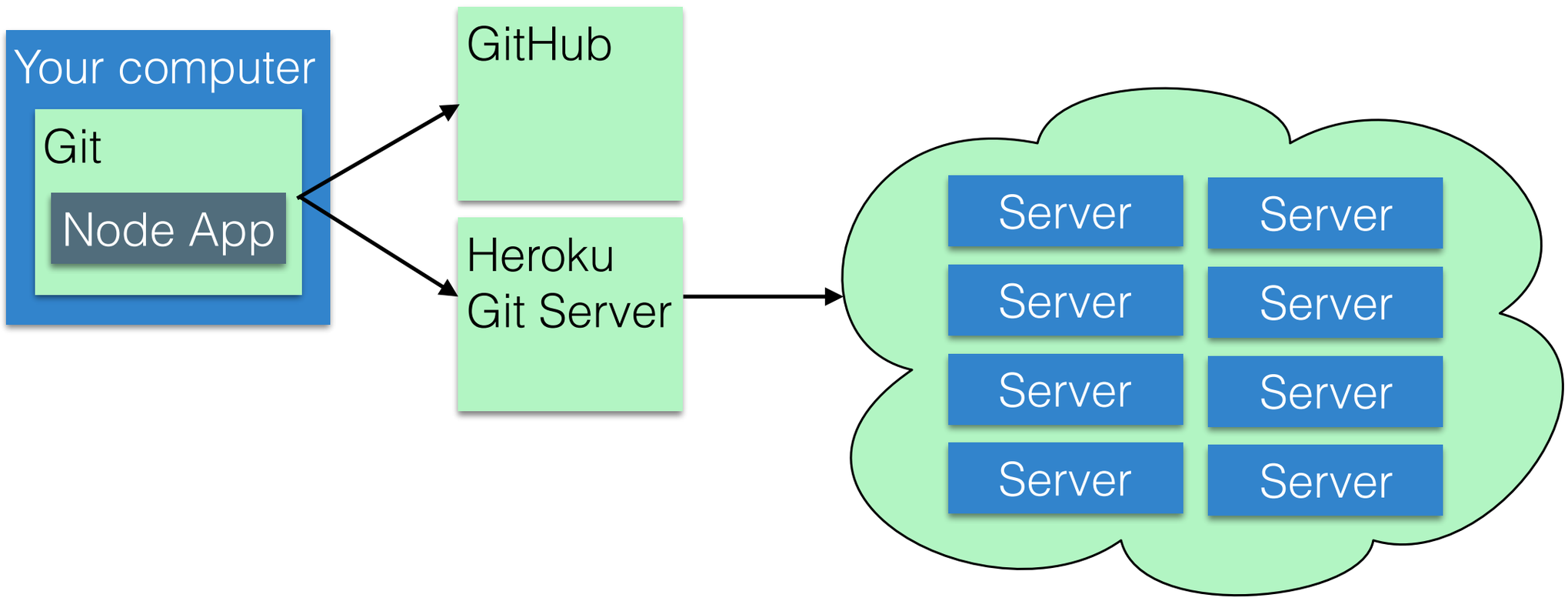
- You get to store 5GB for free!
- Howto:
  - https://www.npmjs.com/package/google-cloud
- Demo: Todos with images + Blobstore
  - Uses Multer instead of express-fileupload
    - Multer lets you temporarily store a file in memory as it goes directly to a remote server (rather than save it to your server first)
    - https://github.com/gmu-swe432/lecture15demos/tree/master/blobstore

Where do we run these backends?

- So, running this on your laptop is not great
- Who wants to run their own actual server?
- Solution:
  - App hosting providers
    - Example: Heroku
    - Big infrastructure companies that will deal with the annoying stuff for you
    - https://devcenter.heroku.com/articles/getting-started-with-nodejs

Heroku

- Once you install Heroku, you communicate via git
- Instead of just pushing to GitHub, push to Heroku
- Then Heroku does some magic
- Do NOT use GHPages + Heroku unless you want extra pain: just run your app on Heroku (including frontend)



Heroku Deployment Servers

Heroku Example

1: Create account, install Heroku on your machine

2: In our app directory, create file “Procfile” with following contents:

```
web: node app.js
```

Tells Heroku what to do when it gets your app

3: Type `heroku create` and follow instructions

4: `git push heroku master`

Deploys your code

5: Visit your app at the site listed in the result of the push (e.g. https://salty-depths-97600.herokuapp.com)

Coming back to the high level

Web “Front End”

Our Node Backend

Firebase

Other storage

Cookies

LocalStorage

Short-lived data

In Memory Storage

Maybe some files?

In-between?

Databases

Blob stores

Long-lived data

Exit-Ticket Activity

Go to socrative.com and select “Student Login”

Class: SWE432001 (Prof LaToza) or SWE432002 (Prof Bell)

ID is your @gmu.edu email

1: How well did you understand today's material
2: What did you learn in today's class?
For question 3:
What state does your project have?

You may not submit this activity if you are not present in lecture.
Doing so will be considered academic dishonesty.