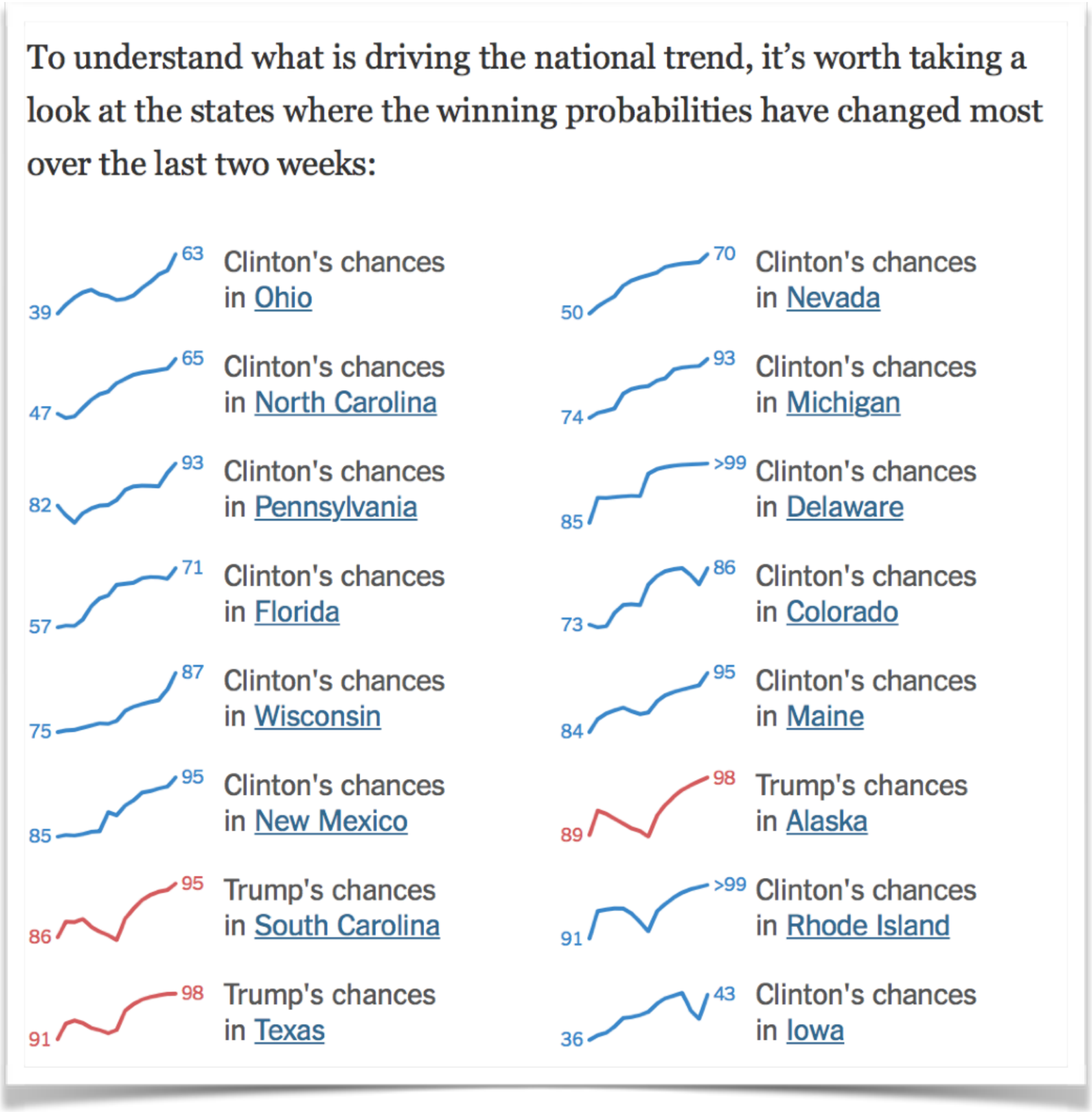
# Backend Development

SWE 432, Fall 2016

Design and Implementation of Software for the Web

# Show & Tell

**Sparklines in NYT**



To understand what is driving the national trend, it's worth taking a look at the states where the winning probabilities have changed most over the last two weeks:

Clinton's chances in Ohio

Clinton's chances in North Carolina

Clinton's chances in Pennsylvania

Clinton's chances in Florida

Clinton's chances in Wisconsin

Clinton's chances in New Mexico

Trump's chances in South Carolina

Trump's chances in Texas

Clinton's chances in Nevada

Clinton's chances in Michigan

Clinton's chances in Delaware

Clinton's chances in Colorado

Clinton's chances in Maine

Trump's chances in Alaska

Clinton's chances in Rhode Island

Clinton's chances in Iowa

http://www.nytimes.com/interactive/2016/upshot/presidential-polls-forecast.html

# Today

- Why do we need backend programming?
- How can/should we structure those backends?
- Node.JS

For further reading:

*https://nodejs.org (Docs + Examples)*

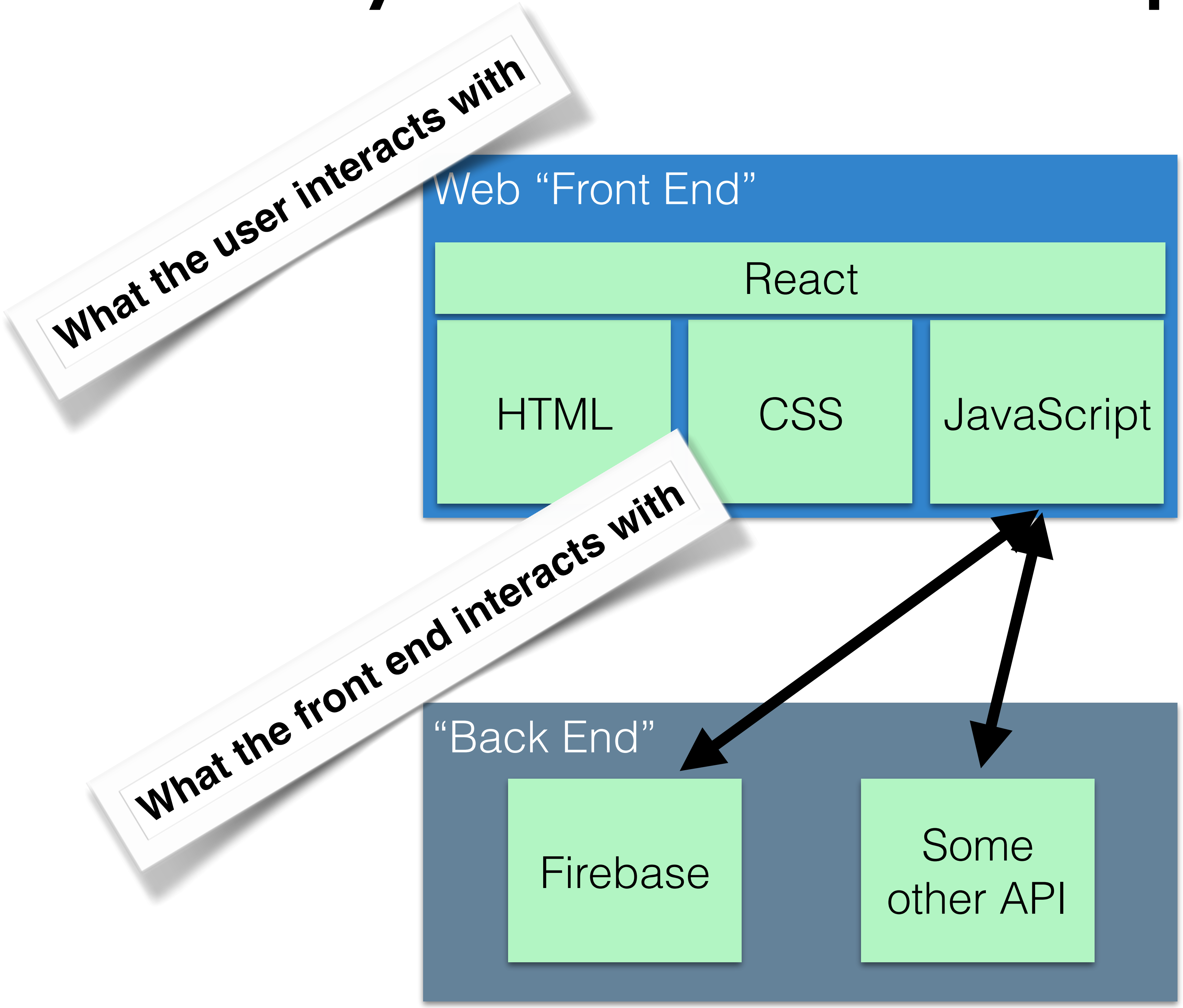
*https://www.npmjs.com (Docs + Examples)*

*https://firebase.google.com/docs/server/setup*

# Why we need backends

- Security: *SOME* part of our code needs to be “trusted”
  - Validation, security, etc. that we don’t want to allow users to bypass
- Performance:
  - Avoid duplicating computation (do it once and cache)
  - Do heavy computation on more powerful machines
  - Do data-intensive computation “nearer” to the data
- Compatibility:
  - Can bring some dynamic behavior without requiring much JS support

# Dynamic Web Apps



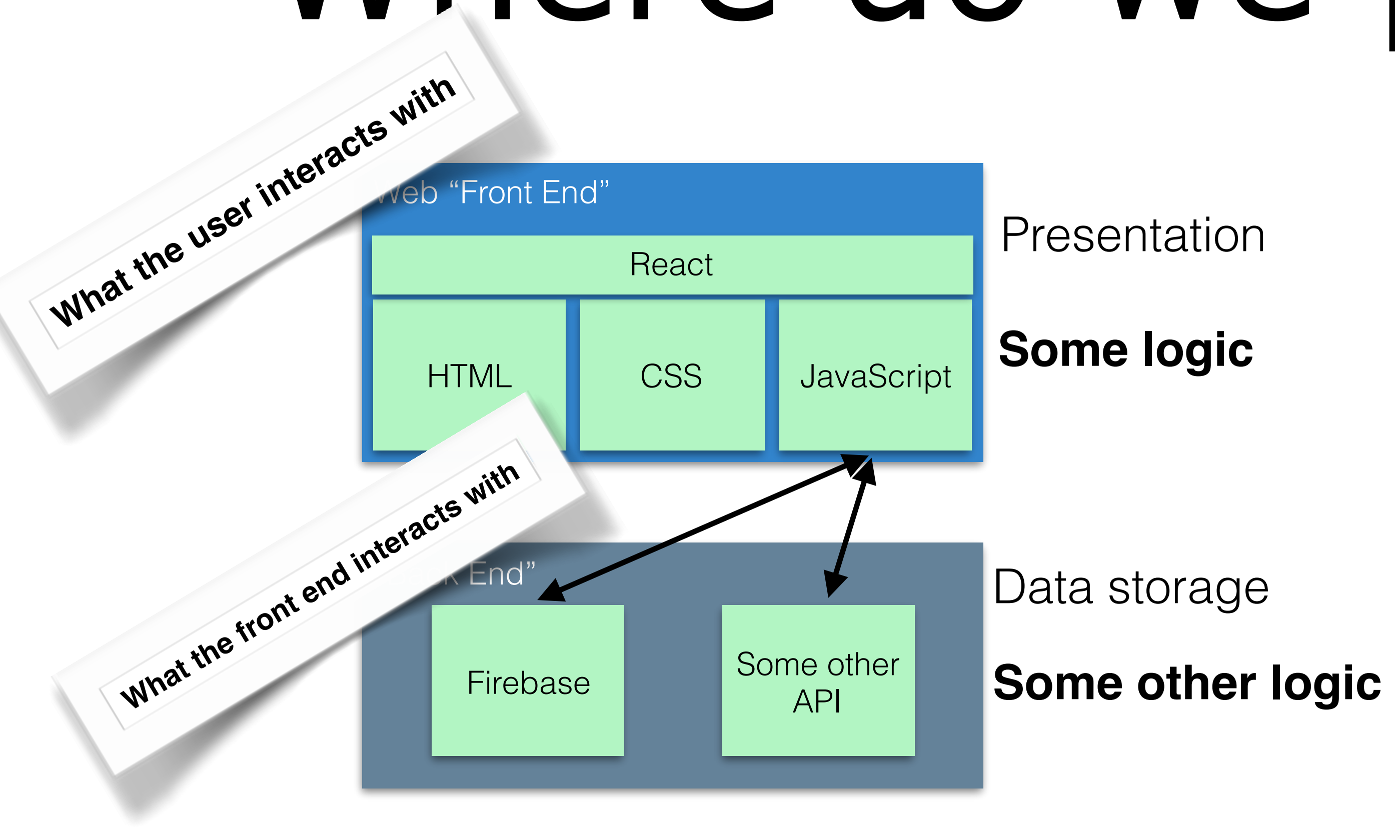
Presentation

Some logic

Data storage

Some other logic

# Where do we put the logic?



**Frontend**

**Pros**

Very responsive (low latency)

**Cons**

Security
Performance
Unable to share between front-ends

**Backend**

**Pros**

Easy to refactor between multiple clients
Logic is hidden from users (good for security, compatibility, and intensive computation)

**Cons**

Interactions require a round-trip to server

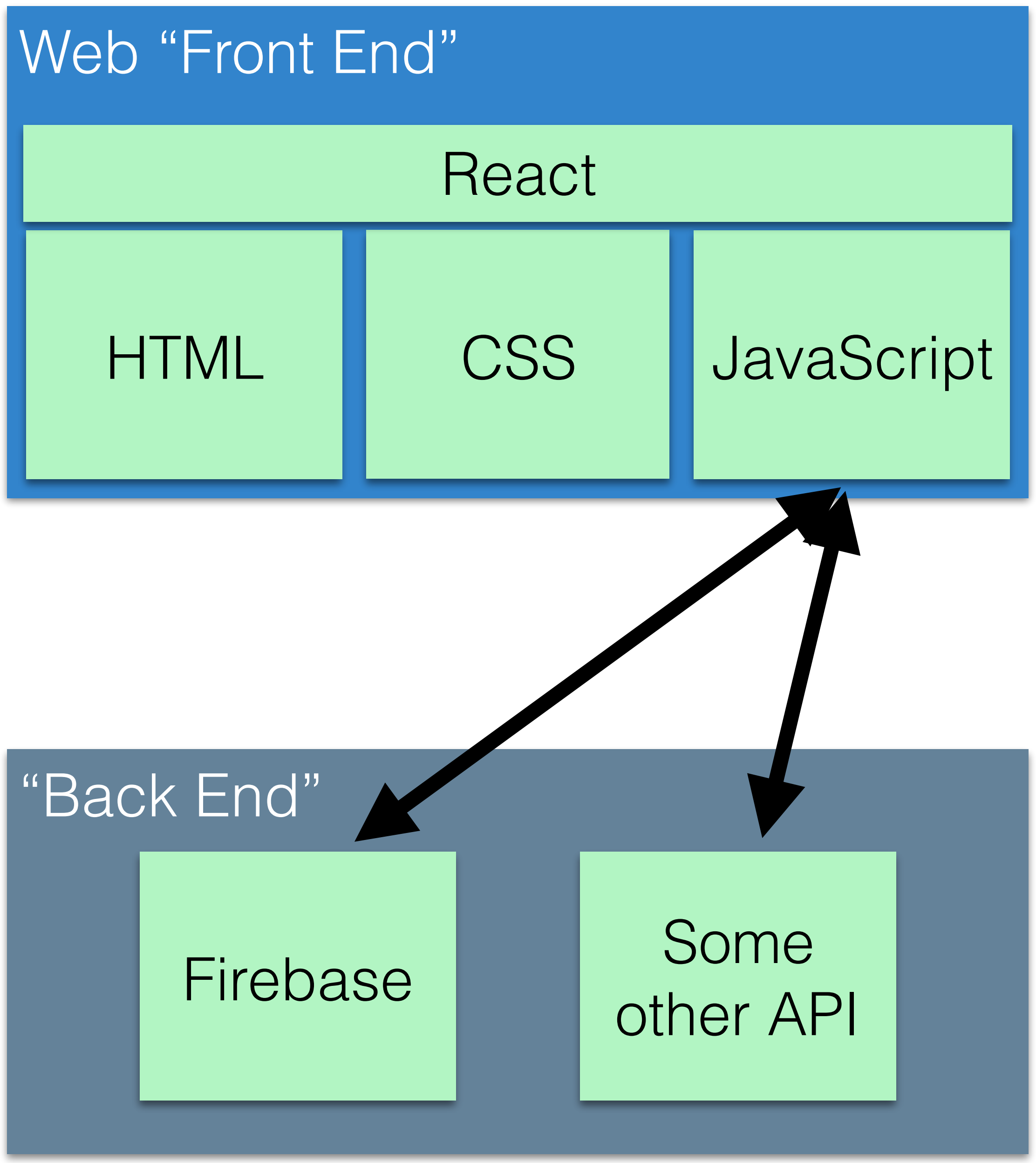
# Why Trust Matters

- Example: Transaction app

```
function updateBalance(user, amountToAdd)
{
    user.balance = user.balance + amountToAdd;
    fireRef.child(user.username).child("balance").set(user.balance);
}
```

- What’s wrong?
- How do you fix that?

# Dynamic Web Apps



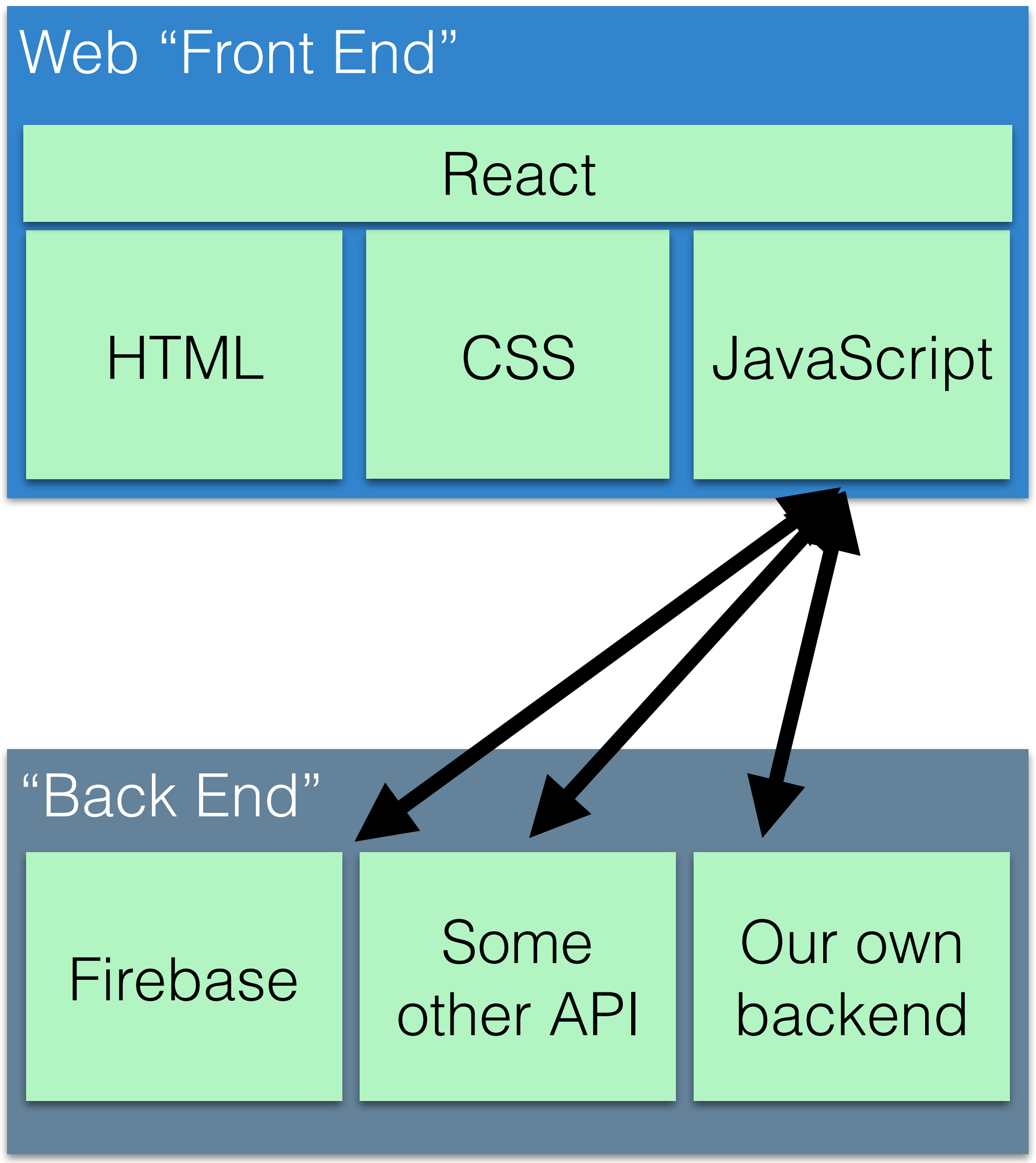
Presentation

Some logic

Data storage

Some other logic

# Dynamic Web Apps



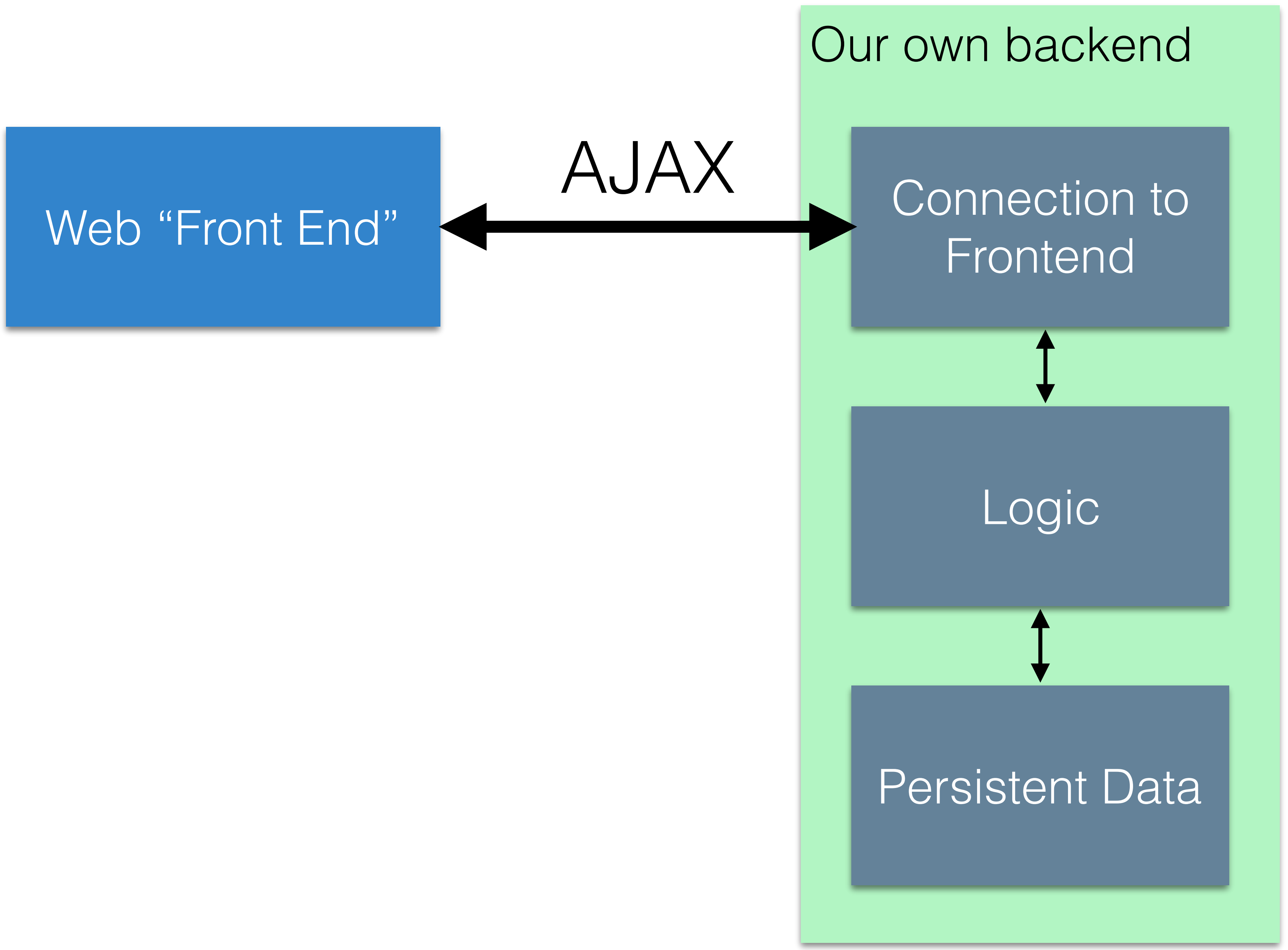
Presentation

Some logic

Data storage

Some other logic

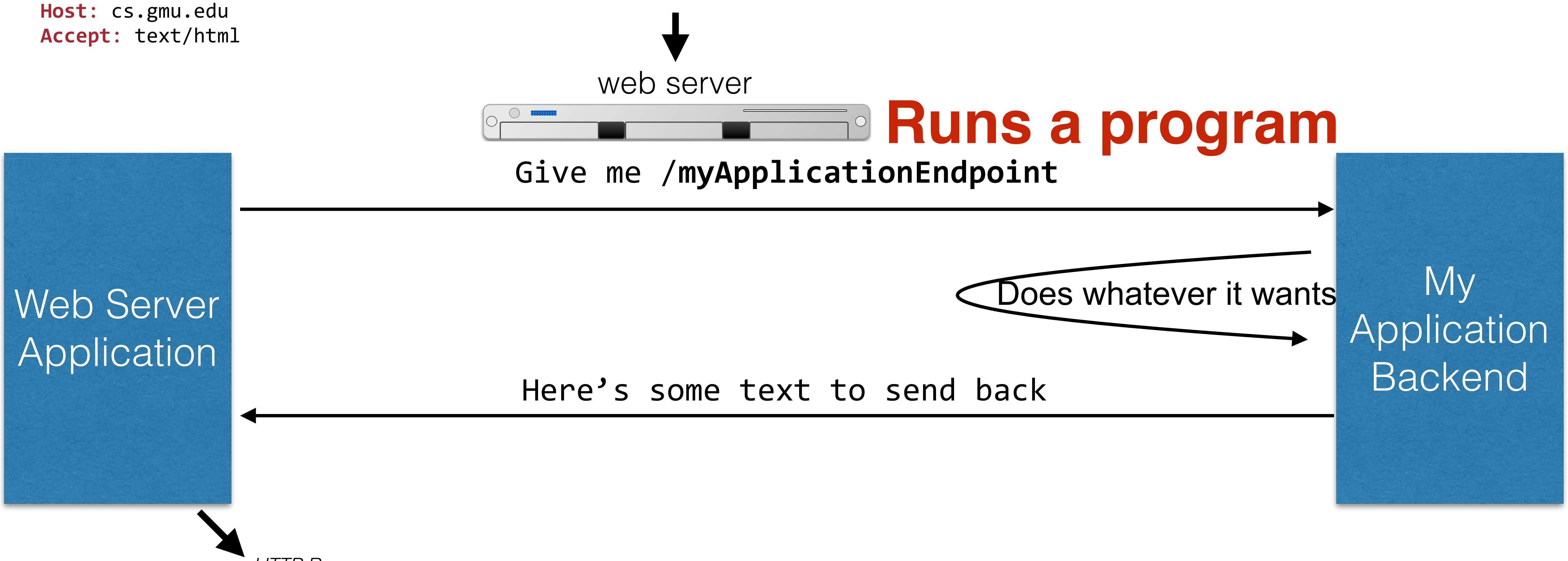
# What does our backend look like?



# The "good" old days of backends

*HTTP Request*

```
GET /myApplicationEndpoint HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```



*HTTP Response*

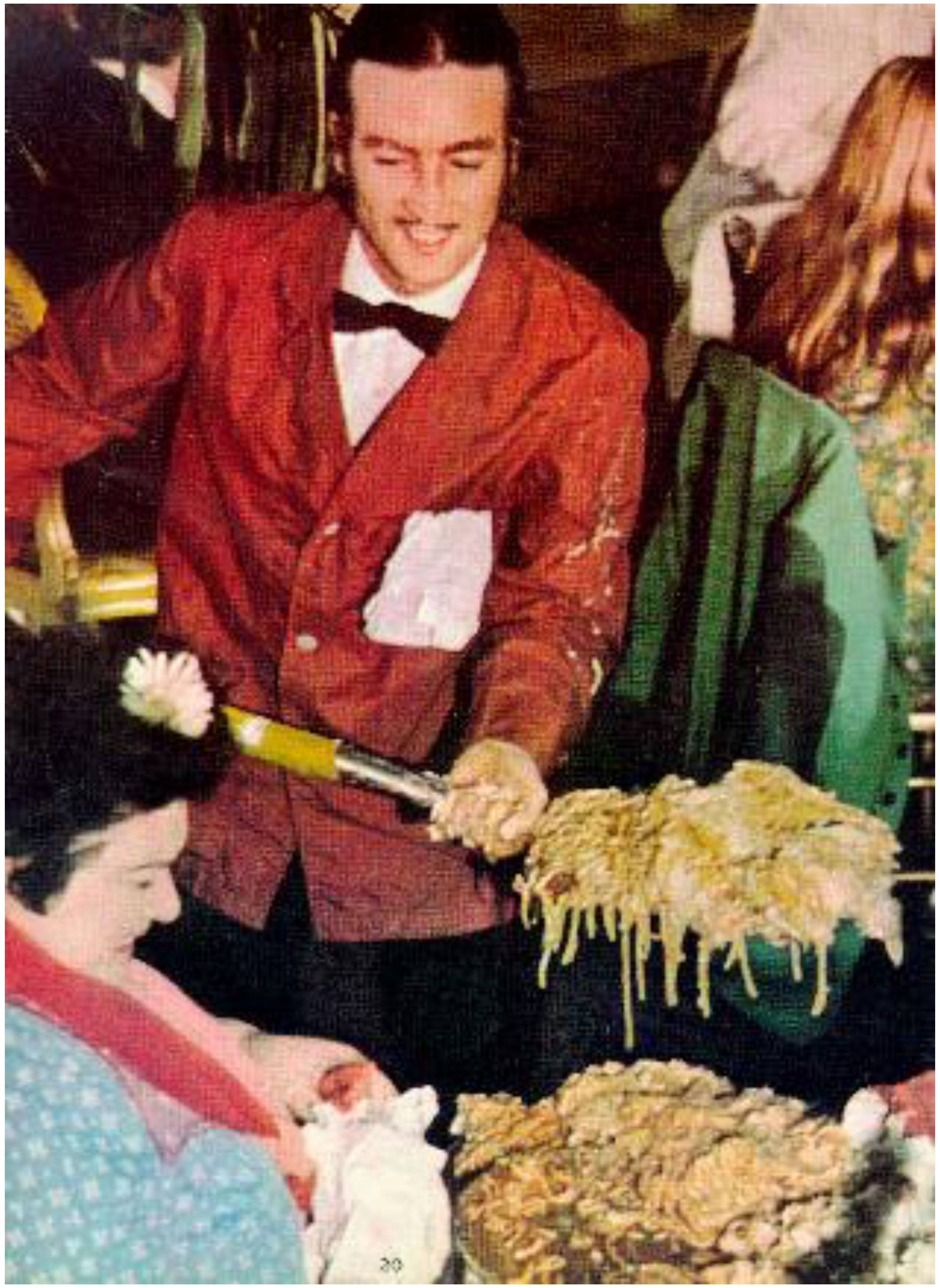
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...
```

## What’s wrong with this picture?

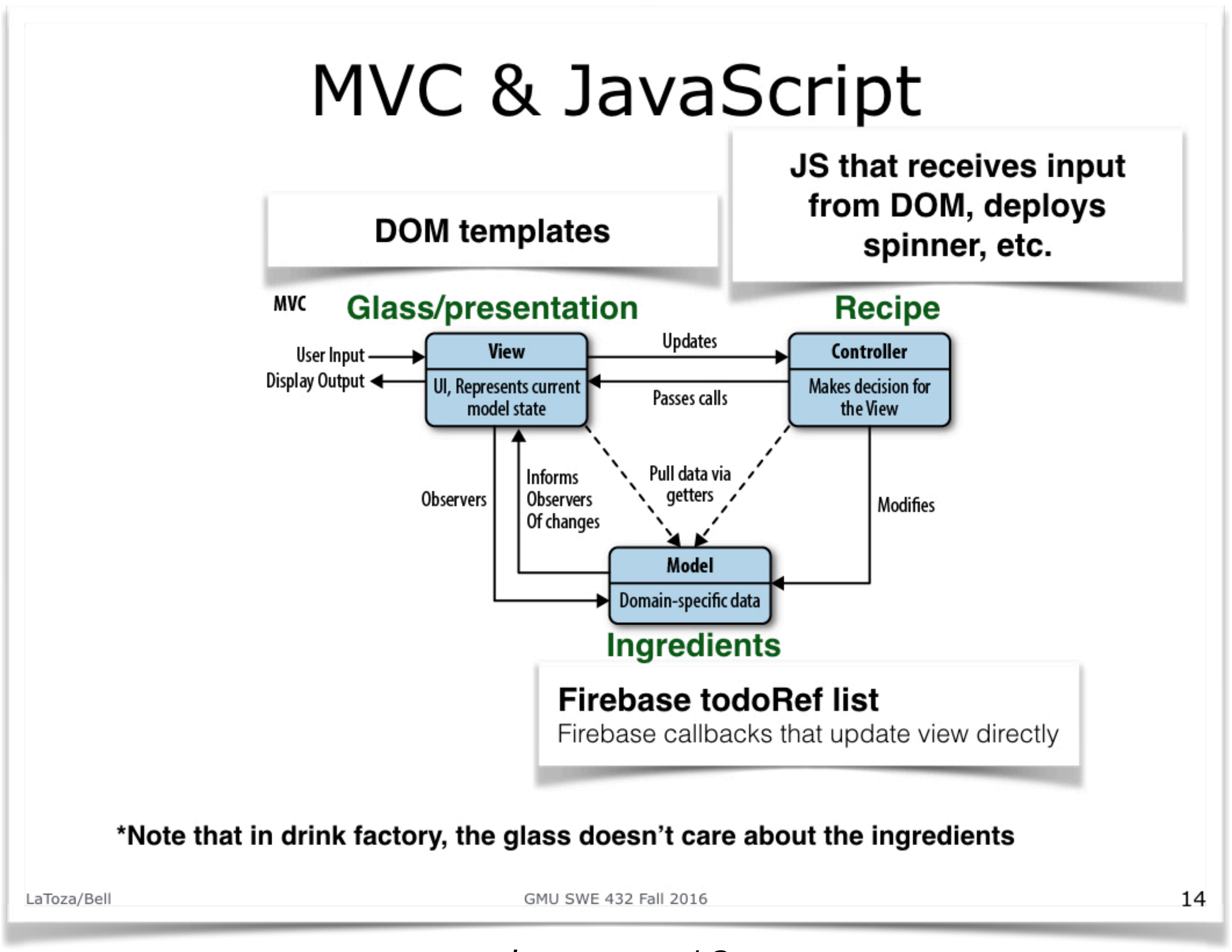
# History of Backend Development

- In the beginning, you wrote whatever you wanted using whatever language you wanted and whatever framework you wanted
- Then… PHP and ASP
  - Languages “designed” for writing backends
  - Encouraged spaghetti code
  - A lot of the web was built on this
- A whole lot of other languages were also springing up in the 90’s…
  - Ruby, Python, JSP

## Backend Spaghetti



# De-Spaghettification



## MVC & JavaScript

DOM templates

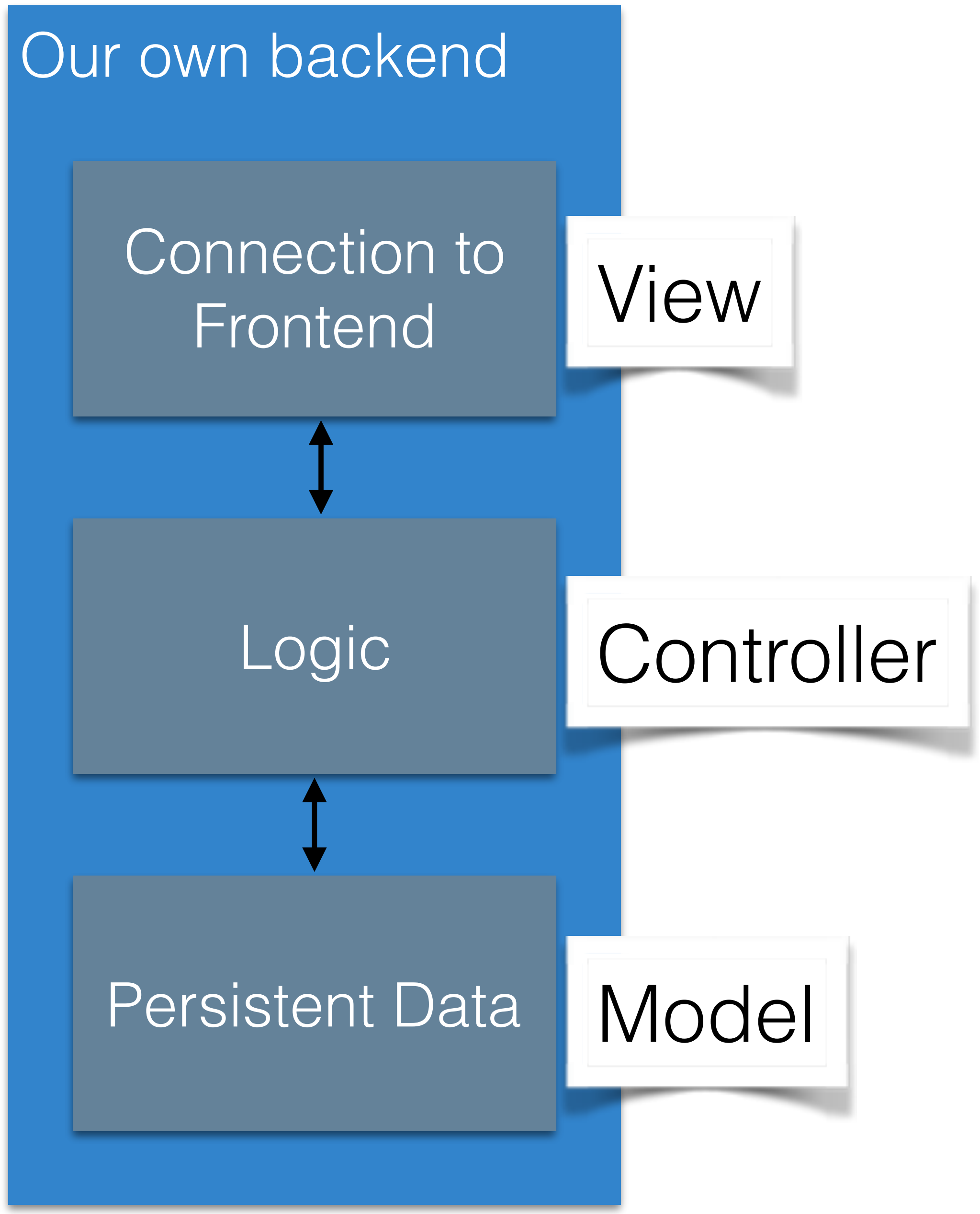
JS that receives input from DOM, deploys spinner, etc.

**Firebase todoRef list**
Firebase callbacks that update view directly

***Note that in drink factory, the glass doesn't care about the ingredients**

LaToza/Bell GMU SWE 432 Fall 2016 14

*Lecture 10*



## MVC & Backend Servers

- There are a ton of backend frameworks that support MVC
  - SailsJS, Ruby on Rails, PHP Symfony, Python Django, ASP.NET, EJB…
- Old days: View was server-generated HTML
- New days: View is an API
- Today we'll talk about Node.JS backend development
- We will **not** talk about making MVC backends and will **not** require you to do so

# Node.JS

- We're going to write backends with Node.JS
- Why use Node?
  - Easy to get into after learning JS (it's JS)
  - Event based: really efficient for sending lots of quick updates to lots of clients
- Why not use Node?
  - Bad for CPU heavy stuff
  - It's relatively immature

# Node.JS

- Node.JS is a *runtime* that lets you run JS outside of a browser
- Node.JS has a very large ecosystem of packages
  - Example: express (web server), nodemon (automatically restarts your server when it changes)
- Must be downloaded and installed
  https://nodejs.org/en/

  - We recommend v4.5.0 LTS (LTS -> Long Term Support, designed to be super stable)

# More on Modules

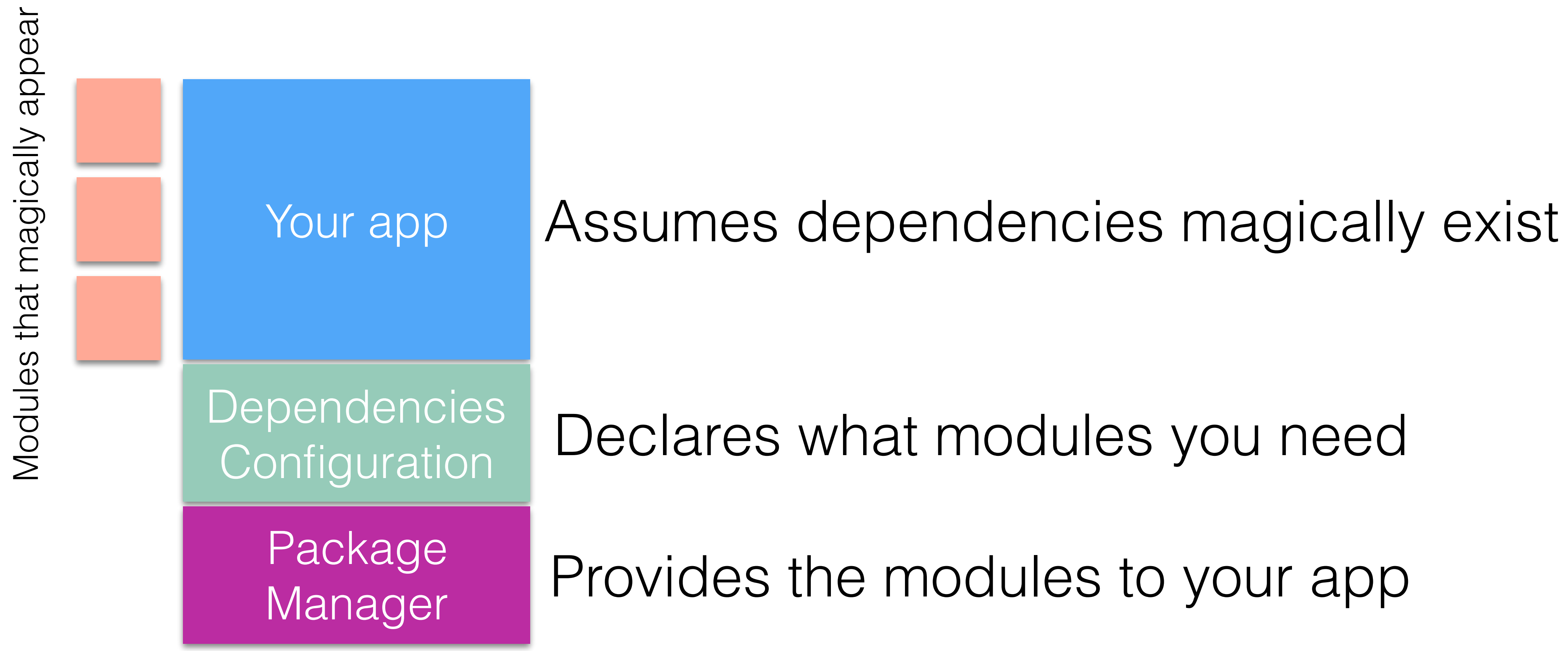
- How have we been using libraries so far?

```
<script src="https://fb.me/react-15.0.0.js"></script>
<script src="https://fb.me/react-dom-15.0.0.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js"></
script>
```

- What's wrong with this?
  - No standard format to say:
    - What's the name of the module?
    - What's the version of the module?
    - Where do I find it?
  - Ideally: Just say "Give me React 15 and everything I need to make it work!"
- This is slowly being fixed for ES6 and on… but Node has a great (non-standardized) approach we can use for backend development

# A better way for modules

- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules


Modules that magically appear
Your app
Assumes dependencies magically exist
Dependencies Configuration
Declares what modules you need
Package Manager
Provides the modules to your app

# NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - "My app is called helloworld"
  - "It is version 1"
  - You can run it by saying "node index.js"
  - "I need express, the most recent version is fine"
- Config is stored in json - specifically package.json

**Generated by npm commands:**

```
{
  "name": "helloworld",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.14.0"
  }
}
```

# Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
```
npm init
```
- Step 2: Declare dependencies
```
npm install <packagename> --save
```
- Step 3: Use modules in your app
```
var myPkg = require(“packagename”)
```
- Do NOT include node_modules in your git repo! Instead, just do
```
node install
```
  - This will download and install the modules on your machine given the existing config!

# Demo: Hello World Server

1: Make a directory, myapp

2: Enter that directory, type **`npm init`** (accept all defaults)

**Creates a configuration file for your project**

3: Type **`npm install express --save`**

**Tells NPM that you want to use express, and to save that in your project config**

4: Create text file app.js:

```
var express = require('express');
var app = express();
var port = process.env.port || 3000;
app.get('/', function (req, res) {
  res.send('Hello World!');
});

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

5: Type **`node app.js`**

**Runs your app**

6: Point your browser to http://localhost:3000

# Demo: Hello World Server

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

Import the module express

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

Create a new instance of express

```
var port = process.env.port || 3000;
```

Decide what port we want express to listen on

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

Create a *callback* for express to call when we have a “`get`” request to “`/`“. That callback has access to the request (`req`) and response (`res`).

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

Tell our new instance of express to listen on `port`, and print to the console once it starts successfully

# Express

- Basic setup:
    - For get:

```
app.get("/somePath", function(req, res){
    //Read stuff from req, then call res.send(myResponse)
});
```

    - For post:

```
app.post("/somePath", function(req, res){
    //Read stuff from req, then call res.send(myResponse)
});
```

    - Serving static files:

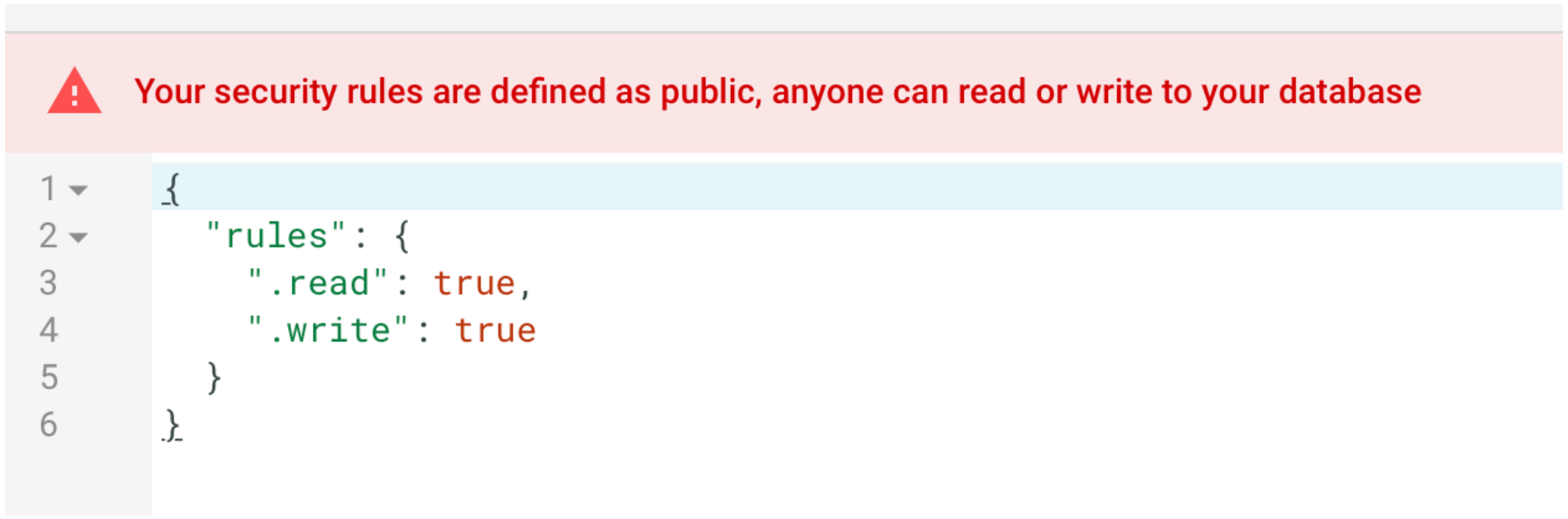
```
app.use(express.static('myFileWithStaticFiles'));
```

        - Make sure to declare this *last*
- Additional helpful module - bodyParser (for reading POST data)

## Putting it together:
## Firebase + Node

# Moving Firebase into Node

- General rule:
  - If you set your database to be writeable by everyone… then make sure NOBODY has your private key



Your security rules are defined as public, anyone can read or write to your database

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

In our security lecture we'll talk about having some data writable through the web app directly and some only through node. For now, we'll talk about the simplest case: Only allow writes through our node backend.

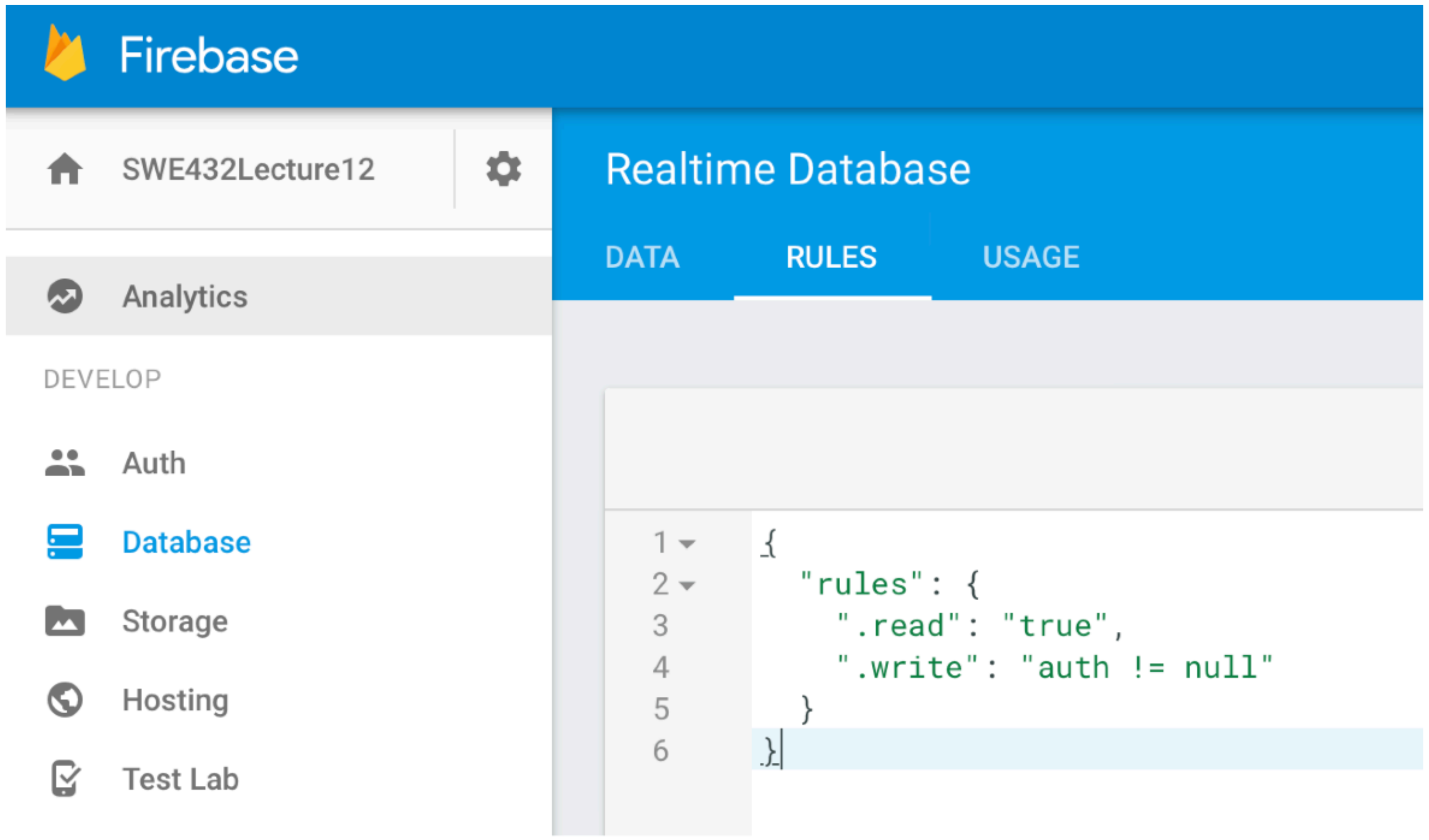
# Firebase + Node

- Step 1: Create a special access key for our Node app to use to access our database
- This key will distinguish our node app from the web app
- Now you can keep publishing your API key, but have a **private** key that you never publish publicly
- https://firebase.google.com/docs/server/setup

1. Create a Firebase project in the Firebase console, if you don't already have one. If you already have an existing Google project associated with your app, click **Import Google Project**. Otherwise, click **Create New Project**.
2. Click settings and select **Permissions**.
3. Select **Service accounts** from the menu on the left.
4. Click **Create service account**.
   a. Enter a name for your service account. You can optionally customize the ID from the one automatically generated from the name.
   b. Choose **Project > Editor** from the **Role** dropdown.
   c. Select **Furnish a new private key** and leave the **Key type** as **JSON**.
   d. Leave **Enable Google Apps Domain-wide Delegation** unselected.
   e. Click **Create**.

# Firebase + Node

- Step 2: Configure our database to allow writes from ONLY clients that have authenticated with a private key
- Database -> Rules -> Set .write to be "auth != null"



```
{
  "rules": {
    ".read": "true",
    ".write": "auth != null"
  }
}
```

# Firebase + Node

- Step 3: Declare our dependency on firebase
  - In our project directory, run:
    ```
    npm install firebase --save
    ```
  - In our app, write:
    - `var firebase = require("firebase");`
- Step 4: Copy our downloaded private key (step 1) to our directory and configure Firebase to connect with it

## Demo: Firebase + NodeJS

## What’s to come?

- How do we create structured APIs?
- How do we maintain some state between our backend and frontend?
- Privacy & Security
- Architecting many services together
- Deploying our backend services