UCDAVIS
GRADUATE STUDIES

Angular 2 & Full Stack TypeScript

and introducing Docker on AWS

UCDAVIS
UNIVERSITY OF CALIFORNIA

Graduate Studies Background

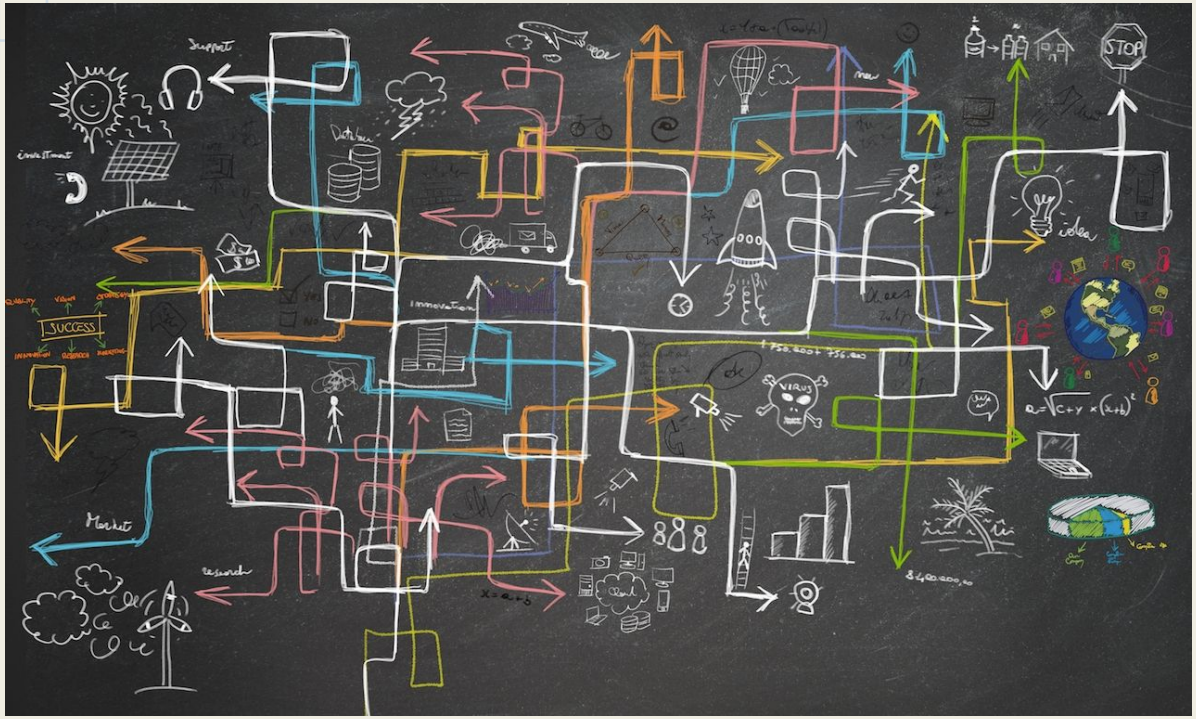
2012

- Single monolithic ColdFusion website
- Almost no client side JavaScript
- MySQL and 2 Oracle DBs
- Only prod environment
- Campus Linux virtualization and other shared services

2015

- Individual web apps running ColdFusion or Java + Spring
- Drupal CMS
- Client side JavaScript running either custom code, jQuery or Angular 1
- MySQL and 3 Oracle DBs
- local, test, CI and prod environments
- Campus Linux virtualization and other shared services

Problem



- Small IT team with limited resources
- Too many programming languages
- Too many server platforms
- Too many databases
- Unreliable or limited control over environment due to shared infrastructure

Goal: Simplify

- Move infrastructure off of shared services
- Move to a single server platform
- Move to a single database type
- Move to a single programming language
- Break up monolithic web apps (not quite microservices but closer)

Solution (2016)

- AWS
  - Self service server and data infrastructure
  - Automated server administration (patching, provisioning, ...)
  - ECS Container Service (Docker)
  - S3 object storage
- JWT
  - Simpler API communication for Service Oriented Architecture
- PostgreSQL
  - Cheaper
  - Easier to manage; we don't even use most of the Oracle features
- Full stack JavaScript (actually TypeScript)
  - Sharing code between client and server
  - Sharing code between projects
  - Simplified tooling
  - **Single programming language to learn, update and master**

Project: Programs Manager

Fairly simple CRUD web app for managing Graduate Program information like a detailed description, degrees offered, people, deadlines, bylaws, etc. Web app is mostly used as an API data end point for other applications like our public website, admissions system, GradHub and commencement registration.

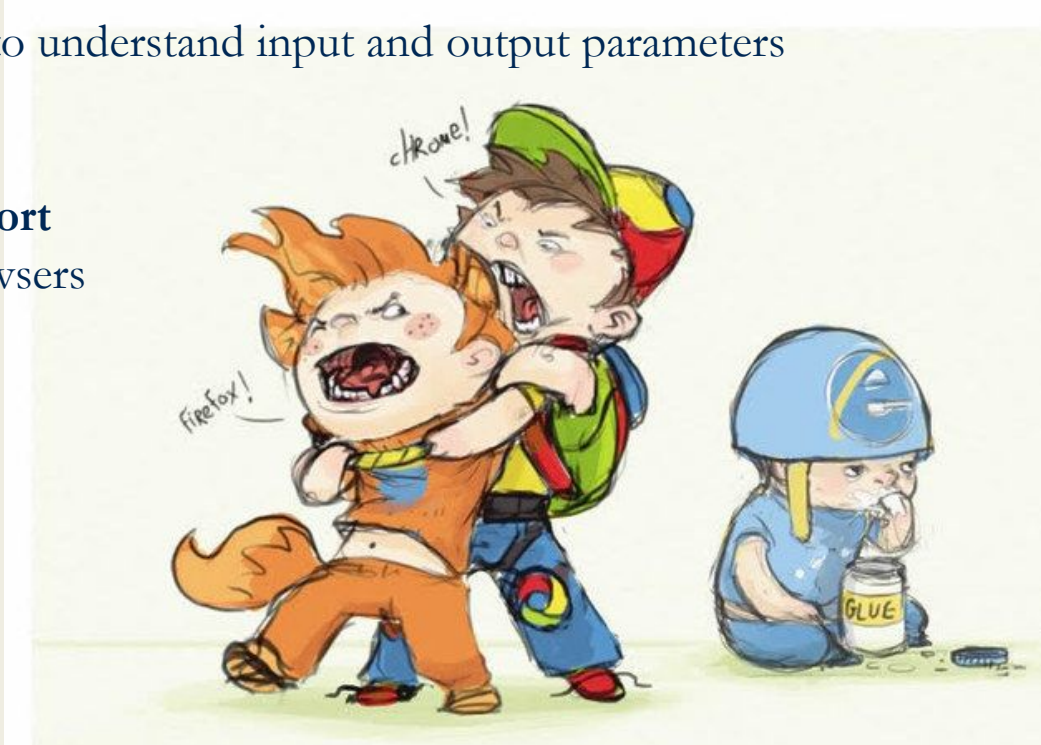
URL: https://programs.gs.ucdavis.edu

API URL: https://programs.gs.ucdavis.edu/api/program
https://programs.gs.ucdavis.edu/api/program/GANT

GIT: https://bitbucket.org/gsucd/programs-manager

Why TypeScript?

- JavaScript is the obvious choice for a single programming language (for web app dev)
- JavaScript has a huge ecosystem
- Angular 2 is using TypeScript
- Types
  - Reduce bugs (debatable)
  - Self documenting; easier to understand input and output parameters
  - Better tooling
  - Warm and cozy feeling!
- **ES6 (ES2015) - ES2017 support**
  - Compiles to ES5 for browsers without ES6 support

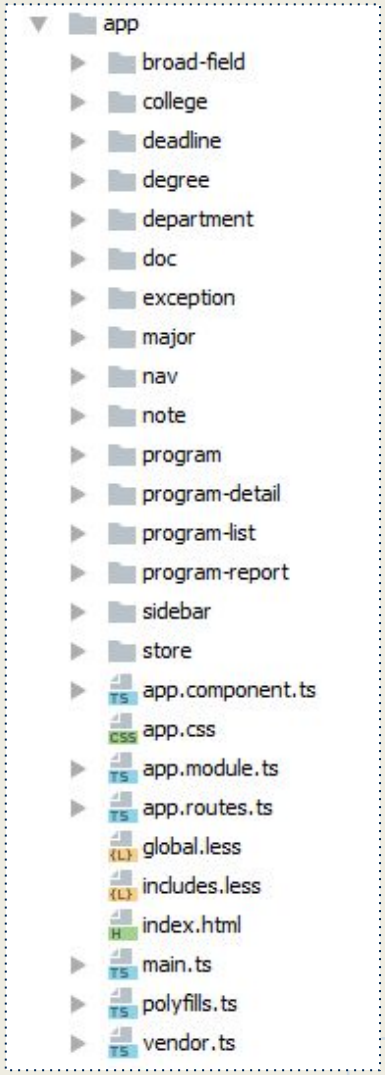


Why Angular 2?

- Angular 1 will eventually be retired
- We had 2 years of experience and code in Angular 1
- Built for the future
  - Component architecture
  - CSS Isolation and Shadow DOM
  - ES6 syntax
  - Embraces functional programming
    - RxJS
    - Supports unidirectional data flow
  - Platform agnostic
    - Server side rendering with Node
    - iOS/Android support with Native Script
- Directives are much easier than Angular 1
  - Zones.js abstracts change detection away
- Supports combining JS, HTML and CSS into a single file

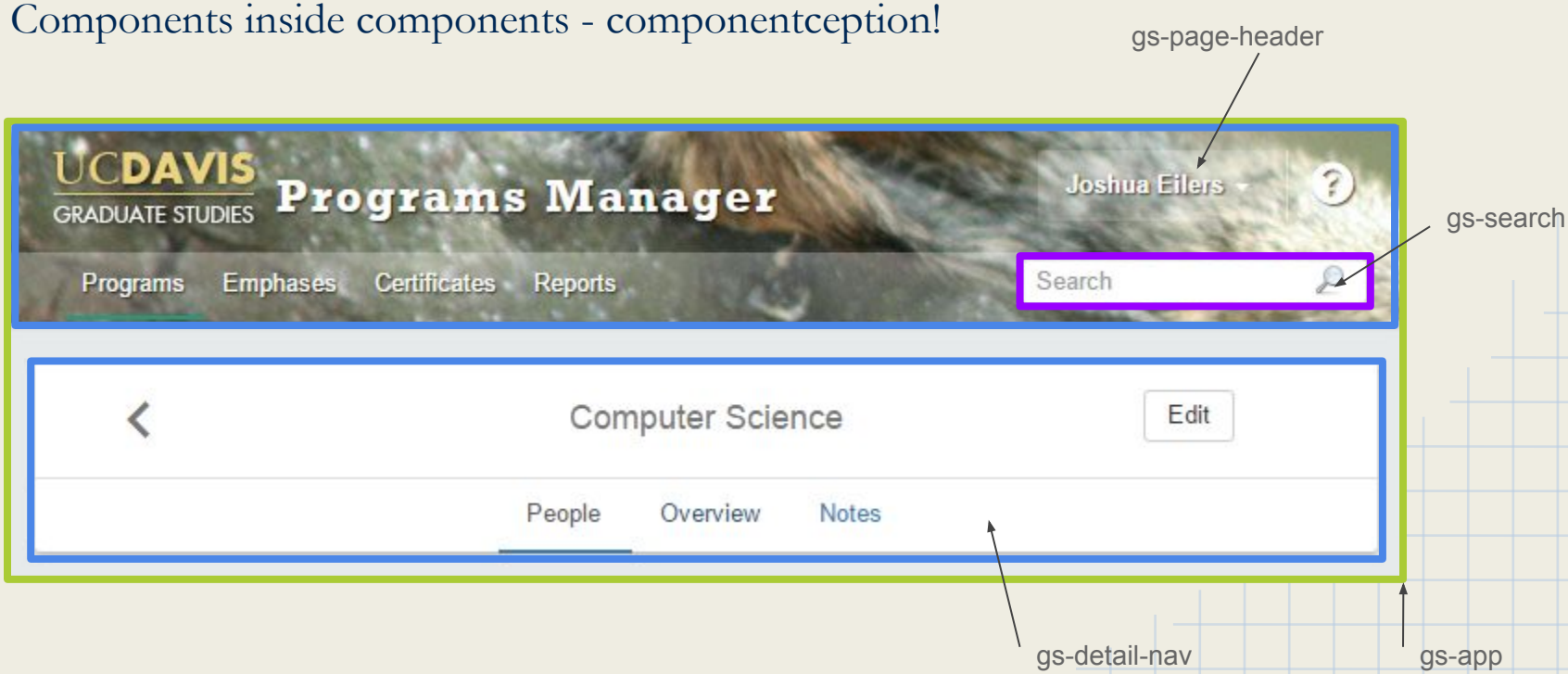
Front-end code organization

- Source files grouped by category
- They may contain:
  - Components
  - Services
  - Stores
  - Routes
  - Html
  - Css



Component Design

- Definition: "part or element of a larger whole"
- Isolate code responsibility for a piece of UI (html, js, css)
- Each gets a tag name
- Specific inputs and outputs
- Components inside components - componentception!



Component Design

What does a component look like inside?

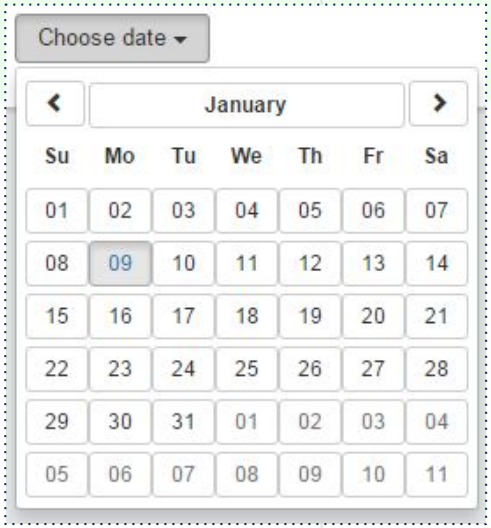
```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'gs-report-nav',
  styleUrls: ['./nav.css'],
  template: `
    <div class="panel panel-default">
      <div class="panel-body">
        <h1 class="col-xs-12 title text-center">{{title}}</h1>
      </div>
    </div>
  `
})
export class ReportNavComponent {
  @Input() title: string;
}
```

How is it used?

```
<gs-report-nav [title]="'Data Report'"></gs-report-nav>
```

A more interesting one

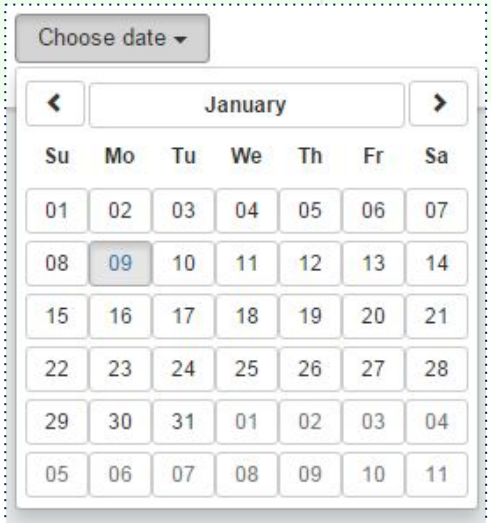
```
@Component({
  selector: 'gs-deadline-date-picker',
  template: `
    <style>
      .dropdown-menu { padding: 5px; }
    </style>
    <div class="btn-group" dropdown [(isOpen)]="isOpen" [autoClose]="config.autoClose">
      <button type="button" class="btn btn-default" dropdownToggle>
        <span *ngIf="deadline.date">{{deadline.date | date}}</span>
        <span *ngIf="!deadline.date">Choose date</span>
        <span class="caret"></span>
      </button>
      <ul class="dropdown-menu" dropdownMenu role="menu" (click)="$event.preventDefault()">
        <li role="menuitem">
          <datepicker [ngModel]="deadline.date" [formatDayTitle]="config.dayTitle" [formatMonthTitle]="config.monthTitle"
            [minMode]="config.minMode" [maxMode]="config.maxMode" [showWeeks]="config.showWeeks" (selectionDone)="onChange($event)">
          </datepicker>
        </li>
      </ul>
    </div>
  ` })
export class DeadlineDatePickerComponent {
  @Input() deadline: Deadline;
  @Output() change = new EventEmitter<Deadline>();
  private isOpen = false;
  private config = { autoClose: 'outsideClick', dayTitle: 'MMMM', monthTitle: 'MMMM',
                     minMode: 'day', maxMode: 'month', showWeeks: false };
  onChange(date: Date) {
    this.deadline.month = date.getMonth() + 1;
    this.deadline.day = date.getDate();
    this.deadline.date = deadlineDateFromMonthAndDay(this.deadline.month, this.deadline.day);
    this.change.emit(this.deadline);
    this.isOpen = false;
  } }
```



A more interesting one

```
@Component({
  selector: 'gs-deadline-date-picker',
  template: `
    <style>
      .dropdown-menu { padding: 5px; }
    </style>
    <div class="btn-group" dropdown [(isOpen)]="isOpen" [autoClose]="config.autoClose">
      <button type="button" class="btn btn-default" dropdownToggle>
        <span *ngIf="deadline.date">{{deadline.date | date}}</span>
        <span *ngIf="!deadline.date">Choose date</span>
        <span class="caret"></span>
      </button>
      <ul class="dropdown-menu" dropdownMenu role="menu" (click)="$event.preventDefault()">
        <li role="menuitem">
          <datepicker [ngModel]="deadline.date" [formatDayTitle]="config.dayTitle" [formatMonthTitle]="config.monthTitle"
            [minMode]="config.minMode" [maxMode]="config.maxMode" [showWeeks]="config.showWeeks" (selectionDone)="onChange($event)">
          </datepicker>
        </li>
      </ul>
    </div>
  `})
export class DeadlineDatePickerComponent {
  @Input() deadline: Deadline;
  @Output() change = new EventEmitter<Deadline>();
  private isOpen = false;
  private config = { autoClose: 'outsideClick', dayTitle: 'MMMM', monthTitle: 'MMMM',
                     minMode: 'day', maxMode: 'month', showWeeks: false };
  onChange(date: Date) {
    this.deadline.month = date.getMonth() + 1;
    this.deadline.day = date.getDate();
    this.deadline.date = deadlineDateFromMonthAndDay(this.deadline.month, this.deadline.day);
    this.change.emit(this.deadline);
    this.isOpen = false;
  } }
```

Two-way data binding, input and output [( )]

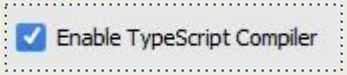


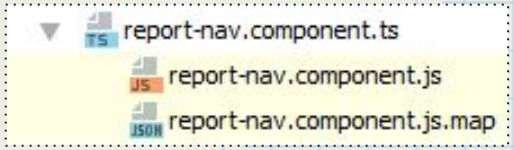
Tooling is pretty sweet

IntelliJ, VS Code, and Sublime have good code completion

```
return this.http
    .
delete(url: string, options?: RequestOptionsArgs)   Observable<Response>
get(url: string,...   Observable<Response>
head(url: string...   Observable<Response>
options(url: str...   Observable<Response>
patch(url: strin...   Observable<Response>
post(url: string...   Observable<Response>
put(url: string,...   Observable<Response>
request(url: str...   Observable<Response>
Did you know that Quick Documentation View (Ctrl+Q) works in completion lookups as well? >>
```

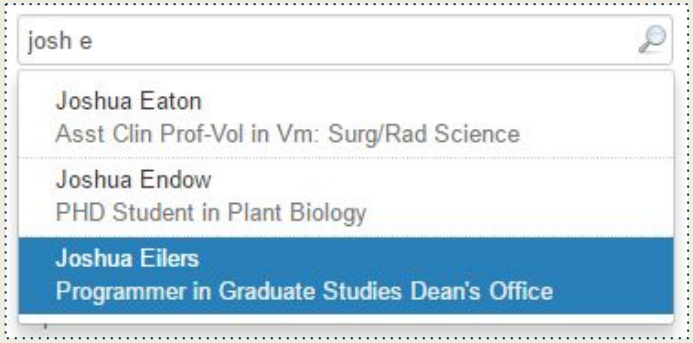
IntelliJ will compile Typescript files for you!





RxJS

- Lodash for async
- Mitigates nested callbacks
- Powerful built-in operators
- Built into Angular 2 http api




```
/* Observe the search        */ Observable.of(this.personSearch)

/* Grab the search value     */ .map(input => input.value)

/* Wait 300ms after keyup    */ .debounceTime(300)

/* Ignore duplicates         */ .distinctUntilChanged()

/* Throw away old responses */ .switchMap(searchText => this.programContactStore.search(searchText));
```

ES6 arrow function (callback)

Shared modules

Add the module to package.json dependencies

git tag

```
"dependencies": {
  "gs-core": "git+ssh://git@bitbucket.org/gsucd/gs-core.git#v1.1.30"
}
```

Point your current project to the shared module for local development

```
$ npm link ../gs-core
```

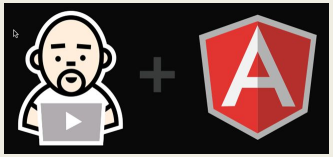
Make changes to your shared module, increment the version, and push it up

```
$ echo "console.log('hello');" > hello.js
$ npm version patch
$ git push origin master
```

Update your project with the new version, other apps will still point to v1.1.30

```
"gs-core": "git+ssh://git@bitbucket.org/gsucd/gs-core.git#v1.1.31"
```

Learning Angular2/RxJS



Egghead.io

Angular2 Quickstart

NPM Scripts

```
{
  "name": "programs-manager",
  "version": "1.0.0",
  "description": "Web app for managing UC Davis Graduate Program information.",
  "main": "./server/index.js",
  "scripts": {
    "dev": "set NODE_ENV=development&& set DEBUG=app:*,gs-core:* && npm run webpack:dev | nodemon --debug
     ./server/index.js",
    "build": "rimraf client/dist && npm run webpack:prod",
    "webpack:dev": "webpack --config webpack.dev.js --progress --watch",
    "webpack:prod": "webpack --config webpack.prod.js --profile --bail",
    "test": "set NODE_ENV=development&& jasmine DEBUG=app:*,gs-core:* JASMINE_CONFIG_PATH=jasmine.json",
    "test:jenkins": "node jasmine-jenkins.js",
    "start": "npm run migrate && node ./server/index.js",
    "migrate:make": "set NODE_ENV=development&& knex migrate:make --knexfile server/db/knexfile.js",
    "seed:make": "set NODE_ENV=development&& knex seed:make",
    "seed": "node server/db/seed.js",
    "migrate": "node server/db/migrate.js",
    "rollback": "node server/db/rollback.js"
  },
  "repository": {
    "type": "git",
    "url": "git@bitbucket.org:gsucd/programs-manager.git"
  },
  "author": "UC Davis Graduate Studies",
  "license": "ISC",
  "homepage": "https://programs.gs.ucdavis.edu"
  ...
}
```

Why Node?

- JavaScript/TypeScript on the server
- Super simple web server and routing setup
- Fast
- One of the fastest growing ecosystems
- Huge npm repository of open source libraries

ORM in Node

- Why not NoSQL?
  - Not enough time to evaluate completely
  - Mongo DB is not managed by AWS requiring more sys admin work
  - AWS Dynamo DB has almost no npm modules and a tiny community
  - Not enough time for Postgres JSONB (next project though)
- Why Postgres?
  - Open Source
  - Cheap
  - Managed by AWS
  - Good support for JavaScript
  - More robust, standard compliant and closer to Oracle syntax than MySQL
  - JSON query and indexing support
- Why Bookshelf.js?
  - 2nd biggest ORM
  - Allows dropping down to SQL since it is built on top of Knex.js
    - Knex had really good reviews

ORM and TypeScript

- Bookshelf.js (and most ORMs) don't work well with TypeScript
  - ES5 syntax
  - Runtime object and property generation
- We built a custom ORM wrapper using TypeScript decorators
  - Inspired by Java Hibernate annotations
  - We want to publish this wrapper to npm, but haven't had time
- Advantages of Bookshelf.js
  - Promises instead of callbacks
  - Transaction support
  - Easy to read source code
  - Eager fetching for child objects
  - Debugging support for SQL statements
  - Migration and seeding support

TypeScript @Decorators

```
@Model()
export class DeadlineModel extends BaseModel<DeadlineModel> implements Deadline {
 @Id() id: number;
 @Prop() month: number;
 @Prop() day: number;

 @Virtual()
 get date(): Date {
   return deadlineDateFromMonthAndDay(this.month, this.day);
 }

 @BelongsTo('ProgramModel')
 program: Program;

 @BelongsTo('DeadlineTypeModel', { foreignKey: 'deadline_type' })
 deadlineType: DeadlineType;
 …
}
```

Class @Decorators

```
export function Model<T extends typeof BaseModel>(options?: ModelOptions) {
 return (clazz: T) => {
   BookshelfInstance['model'](clazz.name, clazz);

   const tableNameProp = Object.getOwnPropertyDescriptor(clazz, 'tableName');

   if (!tableNameProp) {
     const tableName = ( options && options.tableName ) || clazz.name
       .replace('Model', '')
       .replace(snakeCaseRegExp, '$1_$2')  // snake_case
       .toLowerCase();

     Object.defineProperty(clazz.prototype, 'tableName', { get: () => tableName });
   }

   if (!options || options.hasTimestamps) {
     Object.defineProperty(clazz.prototype, 'hasTimestamps', { get: () => ['createdOn', 'lastModifiedOn'] });
   }

   if (clazz.prototype.propMetadata) {
     clazz.prototype.propMetadata = _cloneDeep(clazz.prototype.propMetadata);
   }
   …
}
```

Class decorators get passed the constructor.

Method/property decorators get passed the prototype, property key and the property descriptor (eg getter/setter, enumerable, reference to property).

We are using the **reflect-metadata** module to get the TypeScript typing information.

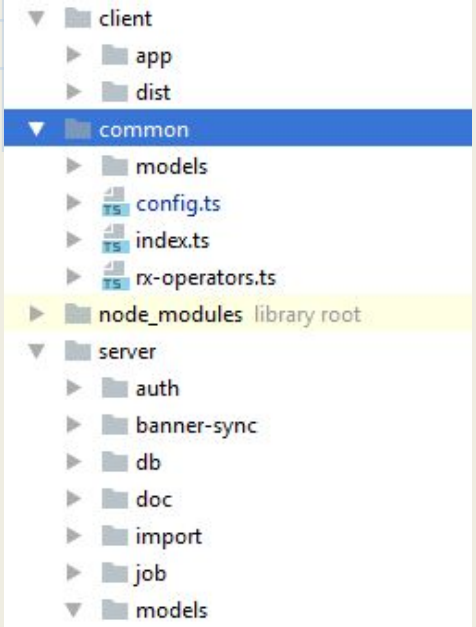
Express

```
const app = express();

app.set('port', process.env.PORT || 3000);
app.set('views', path.join(__dirname, '../client/dist'));
app.set('view engine', 'html');
app.engine('html', hbs.__express);
...
config$.subscribe(
 (config: Config) => {
   ...
   app.use('/dist', express.static(path.join(__dirname, '/../client/dist')));
   ...
   app.use('/', authRoutes);
   app.use('/api/program', programApiRoutes);
   ...
   app.use('/api/doc', docApiRoutes);
   app.use('/api/note', noteApiRoutes);
   app.use('/api/error', errorApiRoutes);
   app.use('/api/system', systemApiRoutes);
   ...
   app.use('/', (req, res) => {
     res.render('index', {
       token: res.locals['user'] && res.locals['user'].token,
       userPref: JSON.stringify(userPref),
       config: JSON.stringify(config.clientConfig)
     });
   });

   http
     .createServer(app)
     .listen(app.get('port'), () => {
       console.info('Express listening on port %s in %s mode', app.get('port'), app.get('env'));
     });
 },
 err => console.error(err)
);
```

JWT in Express

```
router.use('/api', authenticateToken);
router.get('/auth/post-login', validateCasTicket, loadUserDetailsFromCas);
...
const authenticateToken = (req: express.Request, res: express.Response, next: express.NextFunction) => {
   const token = req.headers[config.auth.tokenName.toLowerCase()];
   ...
   services
     .jwtService
     .deserialize(token)
     .subscribe(currentUser => {
         res.locals['user'] = currentUser;
         res.locals['user'].token = token;
       },
       err => next(err),
       () => next()
     );
};

const loadUserDetailsFromCas = (req: express.Request, res: express.Response, next: express.NextFunction) => {
 // If the CAS ticket was validated, find the user details from PRM and set the JWT
 if (session && session.cas && session.cas.user) {
   ...
     // Call People & Role Manager (PRM) to get user details
   currentUserService
    .findByUsernameOrId(session.cas.user)
    .flatMap((currentUser: Person) => {
      res.locals['user'] = _cloneDeep(currentUser);
    })
   ...
   // Serialize user details into JWT
   res.locals['user'].token = token;
   // Set SSO cookie so the front end knows the user has hit CAS
   res.cookie('sso', 'cas', cookieOptions);
 }
}
```

Client / Server Code Sharing



- Client side code lives in **client/app/**
- Common TypeScript interfaces live in **common/**
- Server side code lives in **server/**

- **common/** also contains some shared sorting and validation code

Client / Server Code Sharing

```
export class AppConfig implements Config {
 appName: string;
 appUrl: string;
 ...

 get clientConfig(): CommonConfig {
   return {
     env: this.env,
     shortEnv: this.shortEnv,
     appName: this.appName,
     appUrl: this.appUrl,
     appId: this.appId,
     auth: {
       tokenName: this.auth.tokenName,
       loginUrl: this.auth.loginUrl
     },
     prmUrl: this.prmUrl,
   };
 }
 ...
}
```

```
import { AuthConfig } from 'gs-core/common/config';

export class CommonConfig {
 appName: string;
 appUrl: string;
 appId: number;
 env: string;
 shortEnv: string;
 auth: AuthConfig;
 prmUrl: string;
}
```

TypeScript Interfaces

Shared Interface and Sorter

```
export interface Deadline extends BaseEntity {
 id: number;
 deadlineType: DeadlineType;
 month: number;
 day: number;
 date: Date;
}

export const deadlineComparator = (deadline: Deadline) => {
 return deadline.deadlineType.sortOrder;
};

...
```

Server Side Use

```
@Model()
export class DeadlineModel extends BaseModel<DeadlineModel> implements Deadline {
 @Id() id: number;
 @Prop() month: number;
 @Prop() day: number;
...
```

```
@Model()
export class ProgramModel extends BaseModel<ProgramModel> implements Program {
 ...
 @HasMany('DeadlineModel', { destroyOrphans: true, comparator: deadlineComparator })
 deadlines: Deadline[];
 ...
}
```

TypeScript Interfaces

Client Side Use

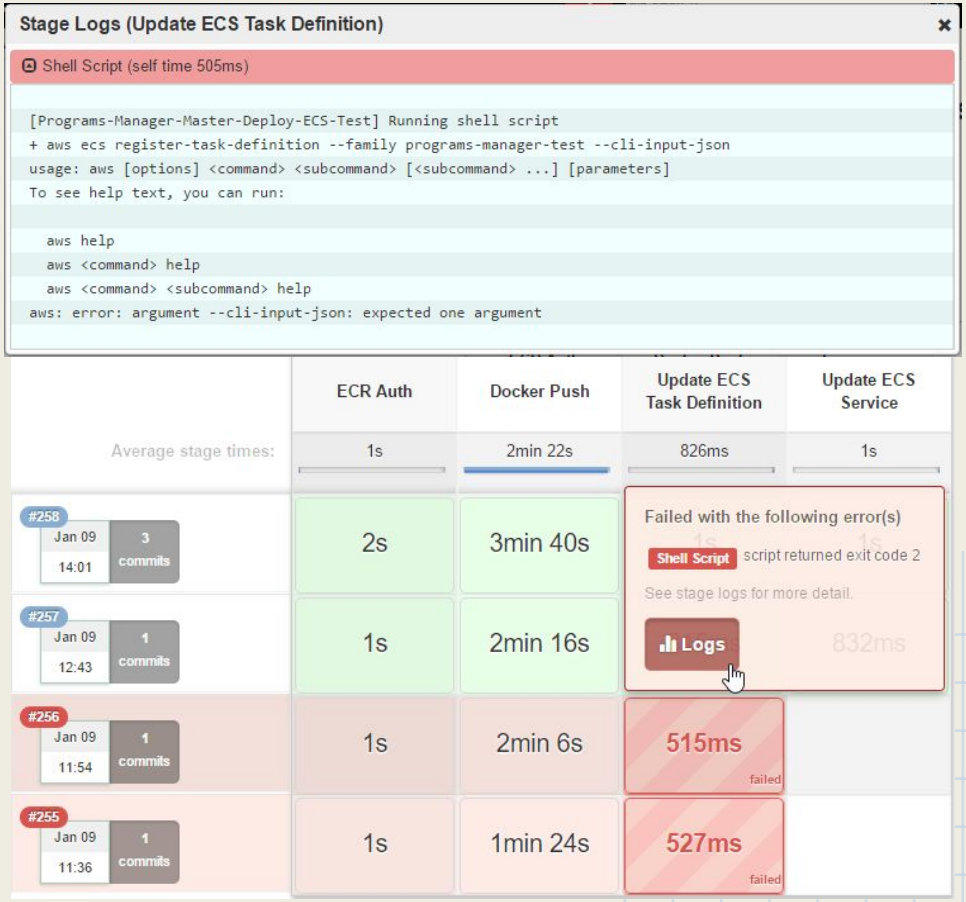
```
import { Deadline } from '../../../../common/models/index';

@Component({
 selector: 'gs-deadlines',
 template:
})
export class DeadlinesComponent implements OnInit {
 private newDeadline: Deadline = this.buildNewDeadline();
 ...

 buildNewDeadline(): Deadline {
   return {
     id: null,
     deadlineType: null,
     month: null,
     day: null,
     date: null
   };
 }

 removeDeadline(deadline: Deadline) {
   ...
 }

 addDeadline() {
   ...
 }
}
```

Why AWS?

- Cost
- Control
  - Changes right when we need them
- Downtime
  - High-availability (at a price)
- Features
  - S3
  - Load Balancers
  - Blue/Green deployment

Jenkins Pipeline

- Build Process as code
  - In application repo
- Separate build stages for troubleshooting



Test

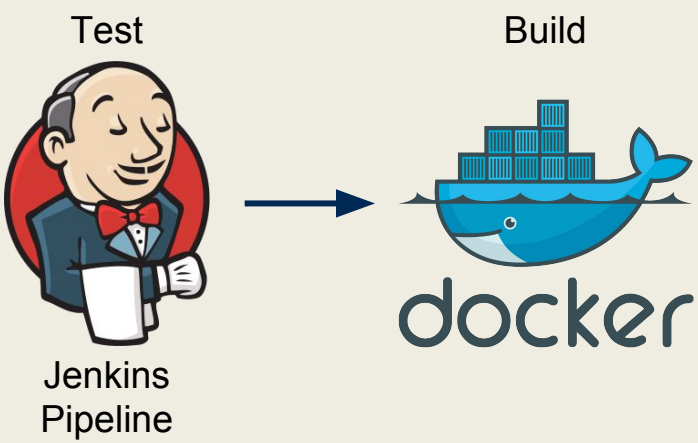
Jenkins Pipeline

Docker

- Build Process as code
  - In application repo

- PaaS on AWS
- Many small applications
  - Too small for Elastic Beanstalk
- Easy version upgrades

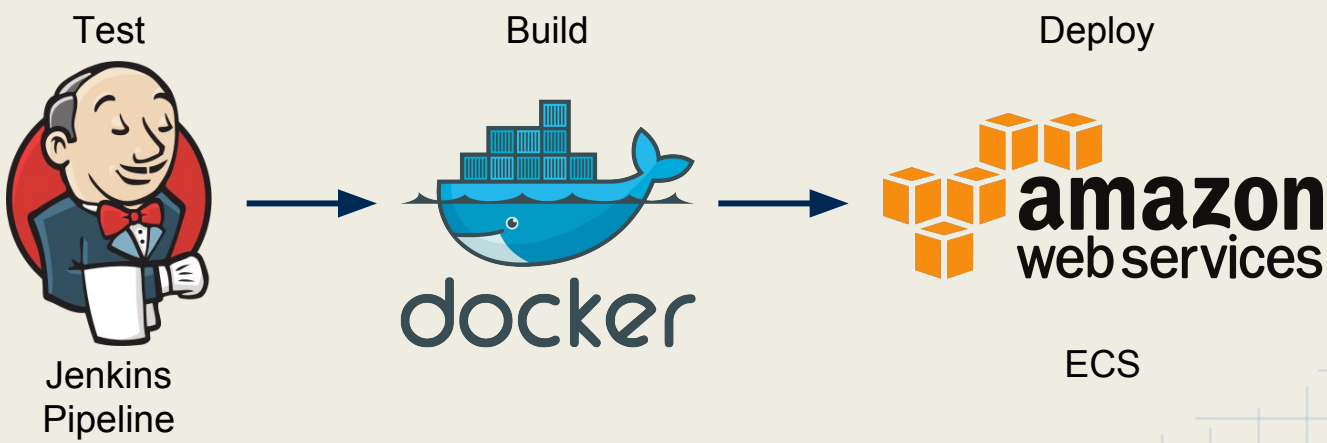
```
FROM node:6.9.3

ENV TZ=America/Los_Angeles

RUN echo $TZ | tee /etc/timezone
RUN dpkg-reconfigure --frontend noninteractive tzdata

COPY node_modules /src/node_modules
COPY package.json /src/package.json
COPY common /src/common
COPY server /src/server
COPY client /src/client

WORKDIR /src

EXPOSE 3000

CMD ["npm", "start"]
```



ECS

(Elastic Container Service)

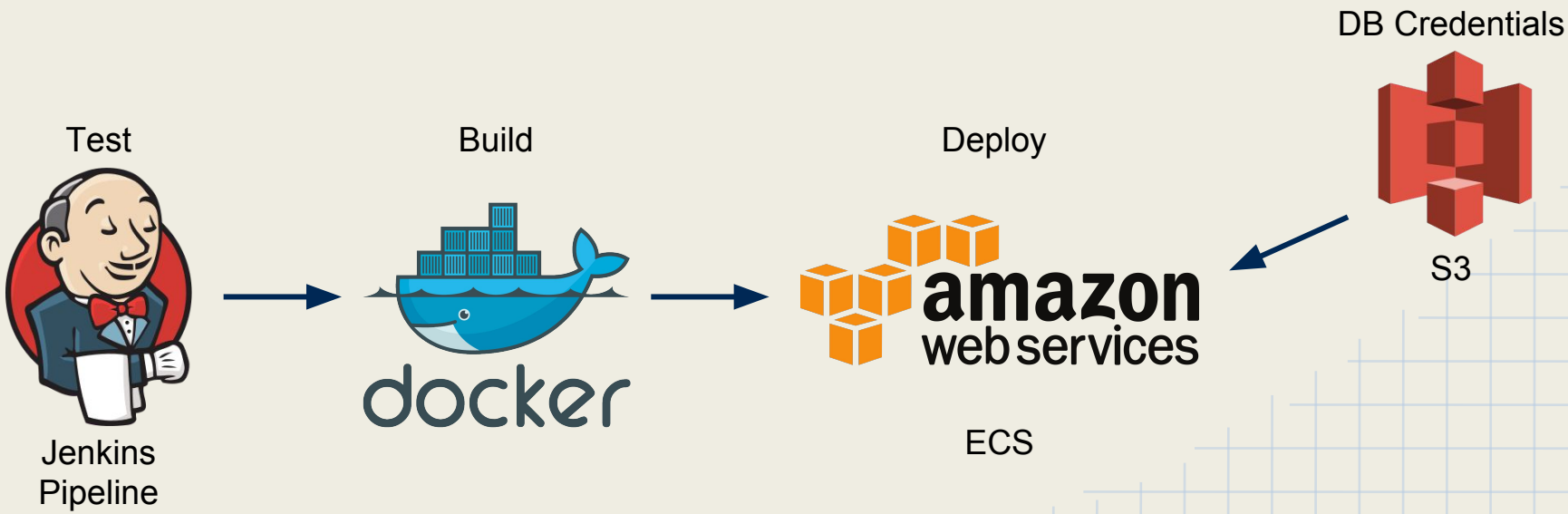
- Container management service
- Self-healing
- Blue/Green deployment
  - Built-in



S3

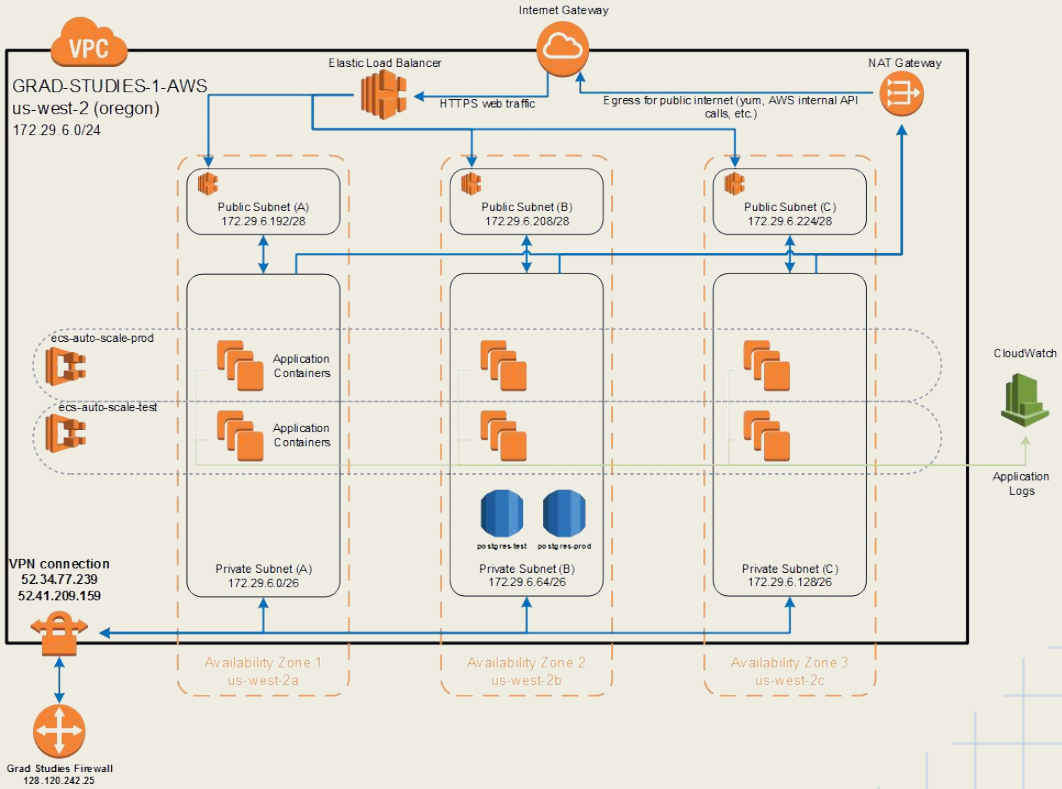
(Simple Storage Service)

- RESTful file storage / retrieval
- Access controls
- Encrypted in transit, encrypted at rest
- $0.023 / GB / month
  - 2015/16 admissions season ~70,000 pdfs
    - On-campus edms: $15,000/yr
    - AWS: < $20

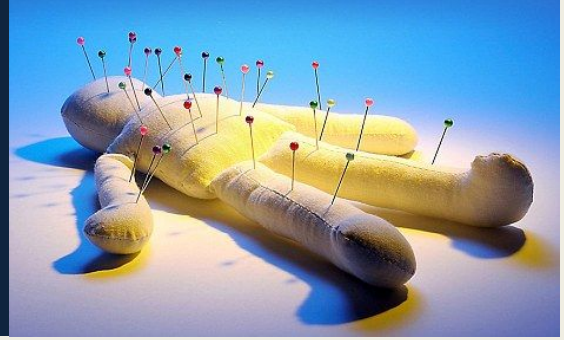


Terraform

- Infrastructure as code
  - In BitBucket
- Declarative
- Easier than Cloud Formation



Pain Points



- Angular 2 in Beta and RC
  - Lots of deprecated features between Beta and rc4
    - Lot of changes to **Router**, **Form** builder and **NgModule**
  - 3rd party modules not able to keep up: **ng2-bootstrap**
  - Multiple ways to do things
    - template: `<style>` or styles: [ ] or styleUrls: [ ]
  - Vague error messages caused by zones.js wrapping most browser actions
  - Slow development waiting for TypeScript to compile
- SystemJS (Angular 2 script loader)
  - Obscure error messages when the module is not found
  - We moved to WebPack
- RxJS
  - Confusing documentation as the API changed a lot between 4.0 and 5.0
  - Complexity; lots of vague and redundant method names
  - Functional programming concepts

Pain Points … continued

- TypeScript
  - Typing files moved from **tsd** to **typings** to **@types** (3rd party libraries)
  - Line endings (\n vs \n\r)
  - Split target JavaScript version; client => ES5, node => ES6
    - Shared **common/** folder constantly recompiled (ES5, ES6, ES5 …)
- SOA / Microservices
  - Performance when one resource is in AWS Oregon and one in Davis
  - Rolling back database if one of the follow up http POST fails (still a work in progress)
  - Handling networking hiccups
- Immature ORMs for Node
  - Bookshelf.js is good for querying, not so good for saving
    - No cascade save; have to manage saving child objects manually
  - Limited documentation
- ORM and TypeScript/ES6
  - Most ORMs dynamically generate models and property which do not work well with TypeScript and ES6

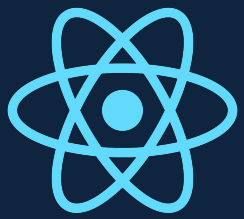
Pain Points … and more

- Error handling in Node
  - Errors swallowed by Promises or Observables (RxJS)
  - Remembering to pass errors to Express next()
- JWT
  - New tab or window forces a new JWT
- Node Single Thread
  - setTimeout / setInterval prevent Node from exiting correctly
- AWS
  - Huge learning curve
  - Default timezone for EC2 and Docker is UTC

Improvements

- Learning and using tooling
  - Less console.log and more debugging
    - IntelliJ or Chrome Node Inspector
- Re-evaluate 3rd party modules
  - ngrx/store (Redux pattern)
  - ng-bootstrap instead of ng2-bootstrap
  - Postgres JSONB with MassiveJS
  - Angular CLI instead of Webpack
- Improve AWS error logging and alerts
- Transactions across microservices
- Scheduled jobs across multiple instances

Why Not React?



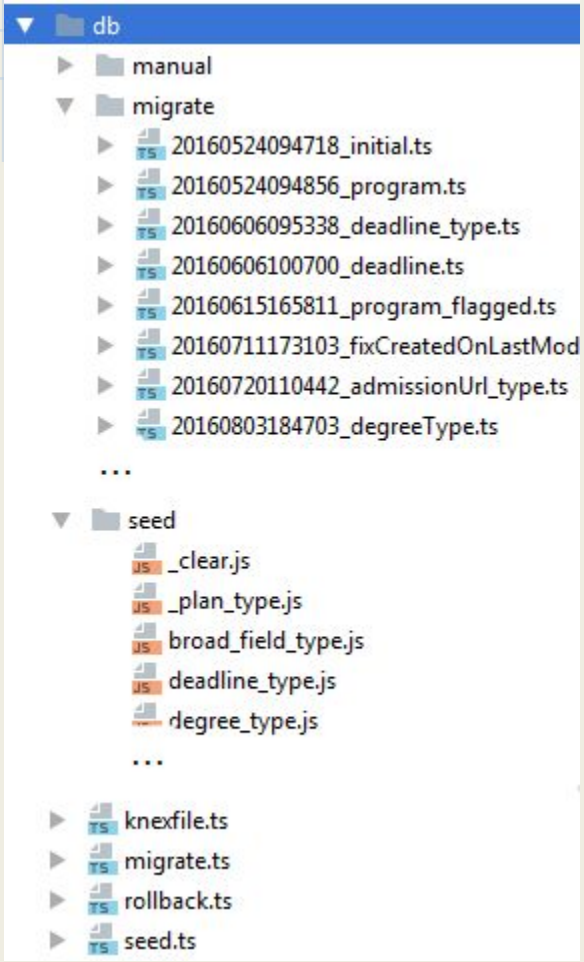
At the time (beginning of 2016)

- Smaller feature set
  - UI only. No opinion on structuring services, http, ...
- Supporting frameworks and tooling were still in flux
  - Redux wasn't the de-facto standard yet
- We had to pick one
  - We did a brief training on React and decided to pilot test Angular 2
- Large Angular 1 ecosystem is more likely(?) to move to Angular 2
  - More StackOverflow answers
  - More Angular modules
  - Better tooling
- Looking back was it a good decision?
  - The jury is still out! Angular 2 works well, but React has only gotten bigger

Why Single Page App?

- More responsive UI and better UX with less code
- Complete separation of client and server code
  - Separation of Concerns
  - Cleaner code organization
- Server is strictly an API endpoint allowing reuse by other servers
  - Our public website and role manager are using the same API endpoint
  - The only server-rendered html is the index file - JS is in the driver seat
- We think it's pretty cool!

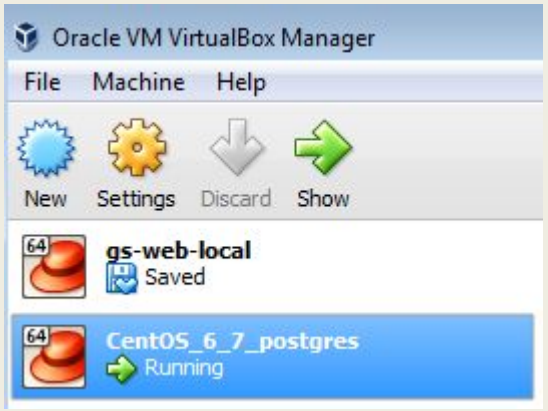
Database Migration & Seeding




```
$ npm run migrate

> programs-manager@1.0.0 migrate C:\Users\richmond\workspace-idea\pr
> node server/db/migrate.js

Loading config: C:\Users\richmond/conf/programs-development.json

AD3+eerichmo@GS-EERICHMO MINGW64 ~/workspace-idea/programs-manager (
$ npm run seed

> programs-manager@1.0.0 seed C:\Users\richmond\workspace-idea\progr
> node server/db/seed.js

Loading config: C:\Users\richmond/conf/programs-development.json
```

Local Dev

- Local Postgres in VirtualBox
  - Quick offline local development
  - Repeatable db integration tests
    - Re-seed before each integration test
- App runs on localhost:3000




```
$ npm test

> programs-manager@1.0.0 test C:\Users\richmond\workspace-idea\programs-manager
> set NODE_ENV=development&& jasmine DEBUG=app:*,gs-core:* JASMINE_CONFIG_PATH=jasmine.json

Debugging [ 'app:*', 'gs-core:*' ]
  app:AppConfig fs.readFile(C:\Users\richmond/conf/programs-development.json) +0ms
Started
Loading config: C:\Users\richmond/conf/programs-development.json
  app:CurrentUserService constructor app token {"id":397032,"name":"Programs Manager"} +894ms
  app:JwtService serialize JWT +0ms
  app:JwtService JWT issued by http://localhost:3000 +0ms
  ...
23 specs, 0 failures
Finished in 6.012 seconds
```

TypeScript @Decorators

```
export class BaseModel<T extends BaseModel<T>> extends BookshelfInstance.Model<T> implements Bookshelf.Model<T>,
BaseEntity {
 @Prop() createdOn: Date;
 @Prop() lastModifiedOn: Date;

 propMetadata: { [ name: string ]: PropMetadata };

 constructor(args?, options?: ModelOptions) { super(args, options); }
 …
}
```

Property @Decorators

```
export function Prop() {
 return (clazzPrototype, propName: string) => {
   const propType = Reflect.getMetadata('design:type', clazzPrototype, propName);

   if (!clazzPrototype.hasOwnProperty('propMetadata')) {
     clazzPrototype.propMetadata = _cloneDeep(clazzPrototype.propMetadata) || { };
   }

   clazzPrototype.propMetadata[propName] = clazzPrototype.propMetadata[propName] || { };
   clazzPrototype.propMetadata[propName].type = propType;

   Object.defineProperty(clazzPrototype, propName, {
     get: function() { return this.get(propName); },
     set: function(value: any) { return this.set(propName, value); }
   });
 };
}
```

Method/property decorators get passed the prototype, property key and the property descriptor (eg getter/setter, enumerable, reference to property).

We are using the **reflect-metadata** module to get the TypeScript typing information.