CREATING ASTRONOMICAL WEB APPLICATIONS FROM SCRATCH

Introduction to modern full-stack MEAN development

- or -

Everything is JavaScript

Michael Young, Indiana University

BEFORE WE START

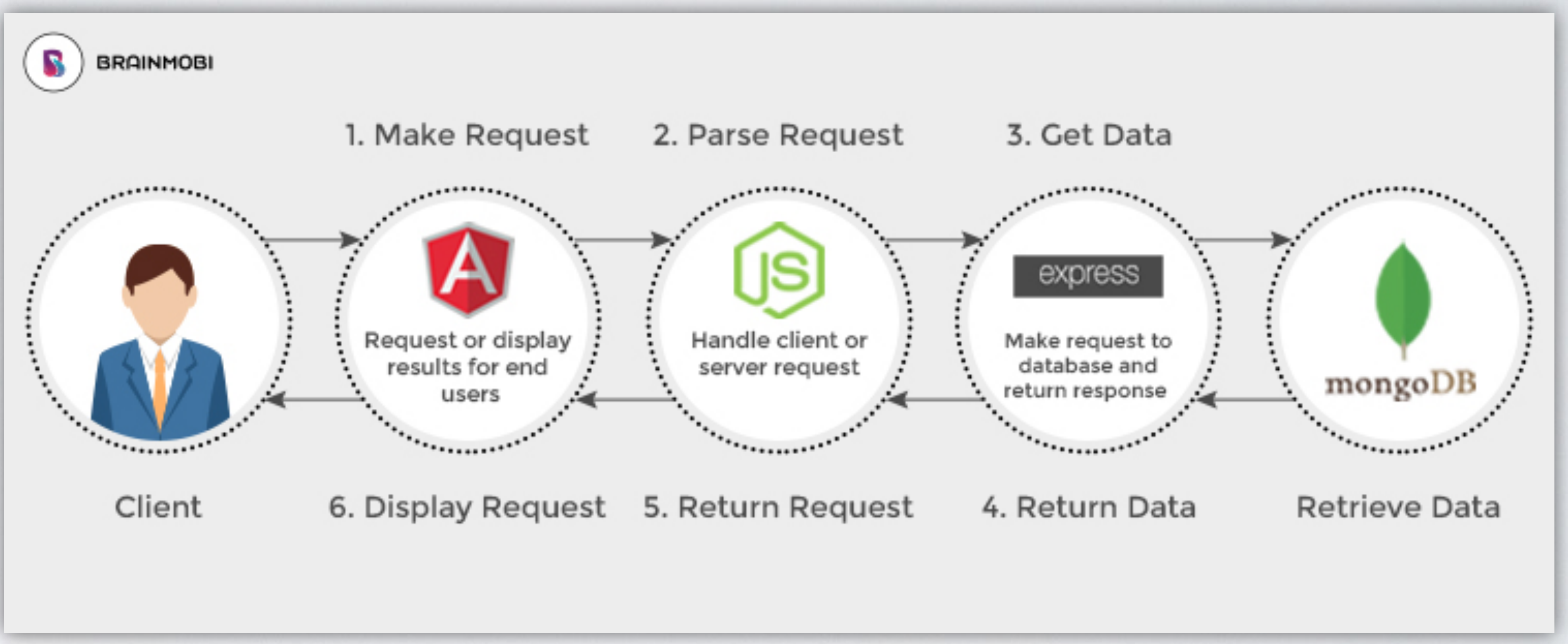
- What you should have done already:
  - Followed instructions on https://github.com/IUSCA/ADASS_tutorial_pre
  - Clone (or fork, then clone) the repository at: https://github.com/IUSCA/mean-stack
- Have a terminal, browser, and IDE or text editor open

GOALS

- Learn the different components of a web application (MEAN stack)
- Understand JSON as a data interchange format
- Store and access data in a MongoDB database
- Create a simple web service with Node.js, npm and Express
- Build a dynamic user interface in AngularJS
- Connect the DB, API and UI to create a fully functional web application
- Explore the geolocation features of MongoDB, and how to adapt them for use in astronomy

THE MEAN STACK

- **M**ongoDB
- **E**xpress.js
- **A**ngularJS
- **N**ode.js



Source: https://www.brainmobi.com/blog/advantages-mean-stack/

JSON

- **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent
  - uses JavaScript syntax, but the JSON format is text only.
- Bottom line: <u>JSON is easy to send and easy to parse</u> (by both machines and humans)

JSON VS XML

```
{
  “firstName”: “Jonathan”,
  “lastName”: “Freeman”,
  “loginCount”: 4,
  “isWriter”: true,
  “worksWith”: [“SPT Technology Group”,
“InfoWorld”],
  “pets”: [
    {
      “name”: “Lilly”,
      “type”: “Raccoon”
    }
  ]
}
```

```
<?xml version="1.0"?>
<person>
  <first_name>Jonathan</first_name>
  <last_name>Freeman</last_name>
  <login_count>4</login_count>
  <is_writer>true</is_writer>
  <works_with_entities>
    <works_with>SPT Technology Group</
works_with>
    <works_with>InfoWorld</works_with>
  </works_with_entities>
  <pets>
    <pet>
      <name>Lilly</name>
      <type>Raccoon</type>
    </pet>
  </pets>
</person>
```

JSON DATA STRUCTURE

```
{
  “firstName”: “Jonathan”,
  “lastName”: “Freeman”,
  “loginCount”: 4,
  “isWriter”: true,
  “worksWith”: [“SPT Technology Group”, “InfoWorld”],
  “pets”: [
    {
      “name”: “Lilly”,
      “type”: “Raccoon”
    }
  ]
}
```

- consists of name/value pairs
  - name/value separated by colon
  - pairs separated by commas
  - names in quotes

JSON values must be one of the following:

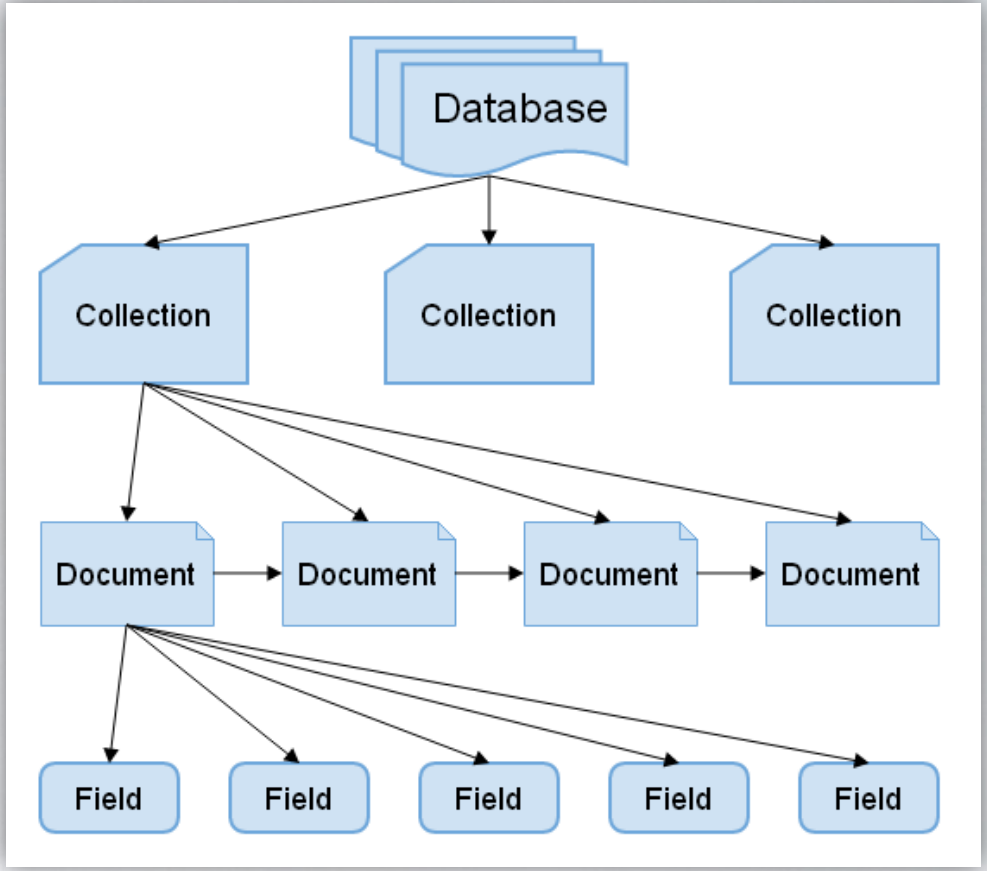
- Object
- Array
- Number
- String
- true/false
- null

A few notes:

- all numbers are double precision
- There is no "date" type
- An Object value can itself contain any other valid data type
- An array can be any combination of types

MongoDB

- A record in MongoDB is a document, which is a data structure composed of field and value pairs.
- MongoDB documents are similar to JSON objects.
- The values of fields may include other documents, arrays, and arrays of documents.
- MongoDB is <u>permissive</u> and <u>flexible</u>
  - no schema enforcement
  - *if it doesn't exist, create it*



Activity

Introduction to MongoDB

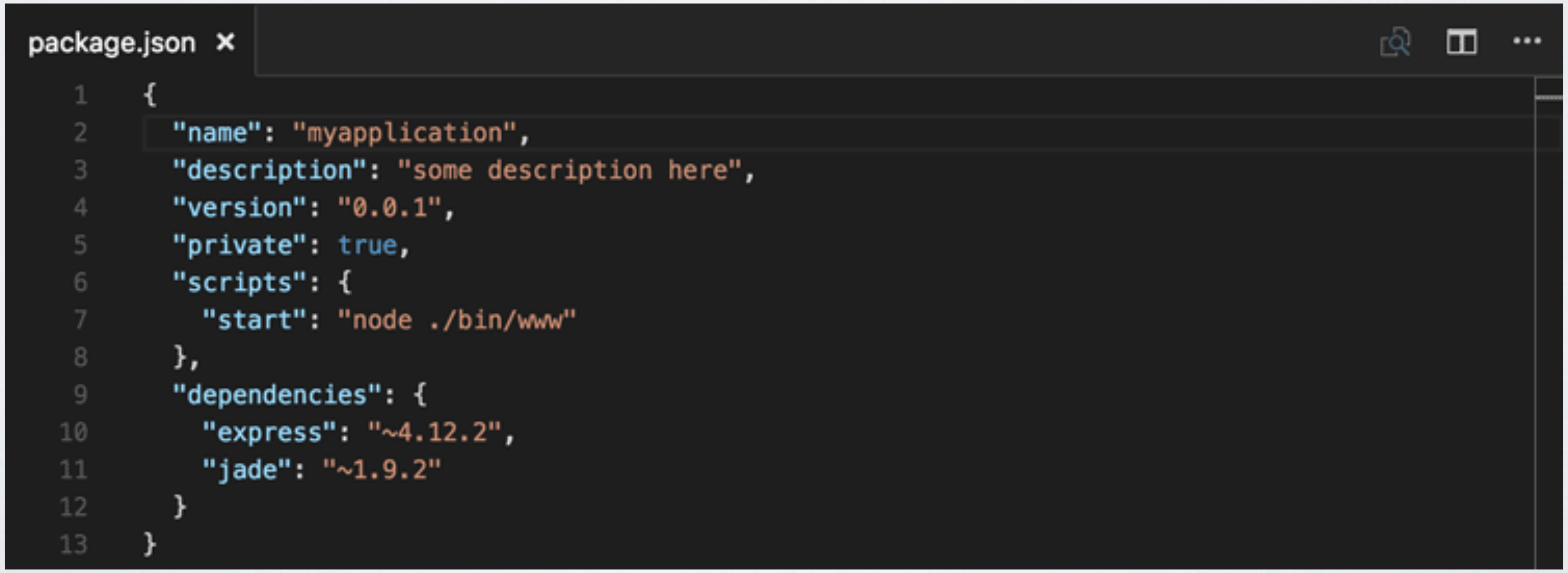
NODE.JS

- Built on Chrome's V8 JS Engine
- Event-driven
- Asynchronous
- non-blocking
- Highly scalable
- Industry standard
  - Huge number of packages, frameworks, articles, tutorials, IDEs, etc.

NPM AND PACKAGE.JSON

- **N**ode **P**ackage **M**anager
  - package manager for JavaScript, included with Node.js
  - Retrieves dependencies from a remote repository
  - Installs in one command (`npm install`), all the dependencies of a project through the package.json file
  - Can install packages locally (per-project) or globally (system-wide). Local installs take precedence.

- **package.json**
  - defines the application, creator, version, dependencies
  - used by npm to make sure everything that is needed for a project gets installed
  - also dictates different ways to start the application



```json
{
  "name": "myapplication",
  "description": "some description here",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "~4.12.2",
    "jade": "~1.9.2"
  }
}
```

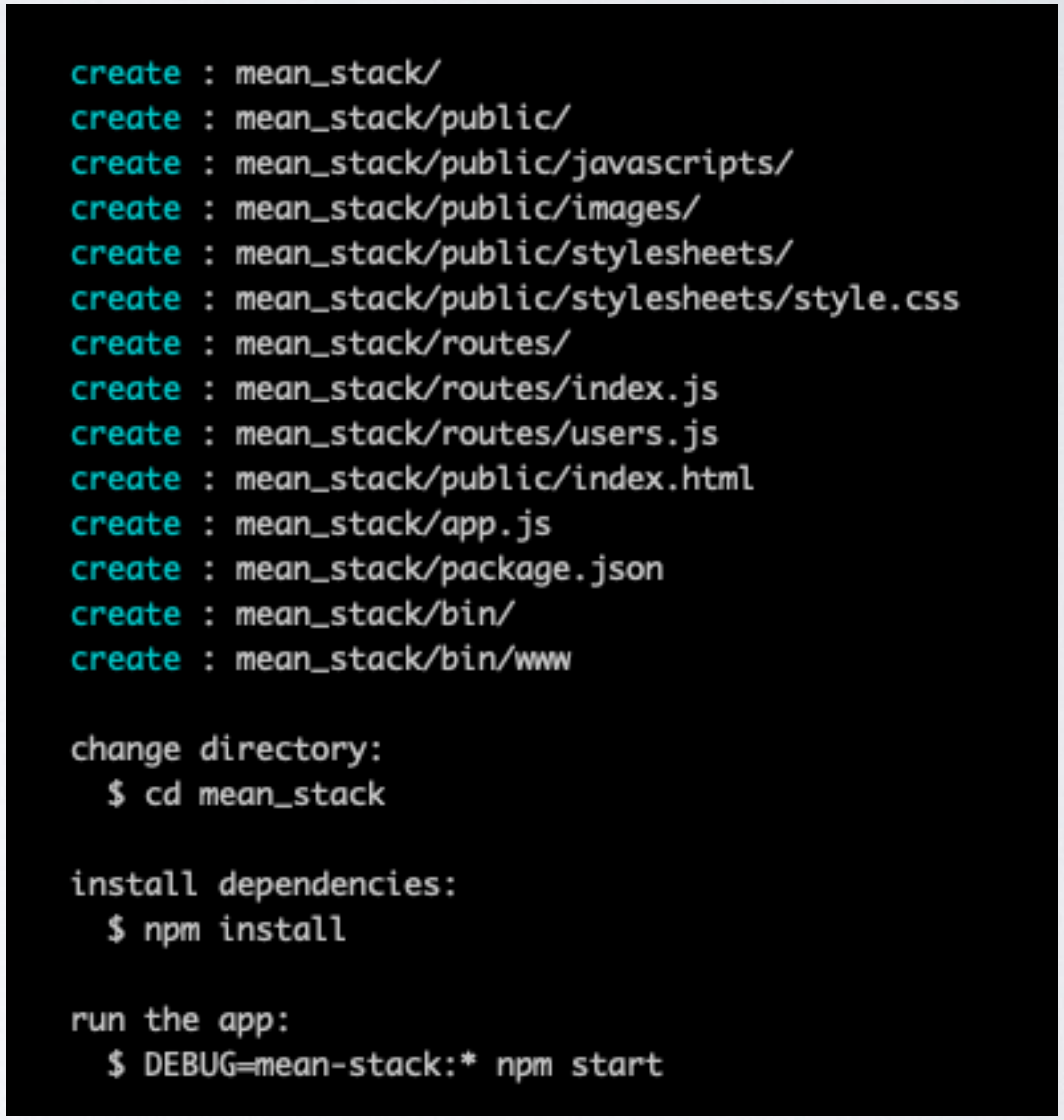
EXPRESS FRAMEWORK

- Express is a framework for developing web applications
- Express generators can save you a lot of time
- Install express globally
  - `npm install express-generator -g`
- Now we can create a web application with:
  - `express --no-view mean_stack`

EXPRESS GENERATOR

- After running express generator you should see a list of things it did, e.g. "create : mean_stack/"

- Followed by a list of things you need to do, e.g. "cd mean_stack"



```
create : mean_stack/
create : mean_stack/public/
create : mean_stack/public/javascripts/
create : mean_stack/public/images/
create : mean_stack/public/stylesheets/
create : mean_stack/public/stylesheets/style.css
create : mean_stack/routes/
create : mean_stack/routes/index.js
create : mean_stack/routes/users.js
create : mean_stack/public/index.html
create : mean_stack/app.js
create : mean_stack/package.json
create : mean_stack/bin/
create : mean_stack/bin/www

change directory:
  $ cd mean_stack

install dependencies:
  $ npm install

run the app:
  $ DEBUG=mean-stack:* npm start
```

- Structure of an Express app:
    - *app.js* - the main script that creates the Express app
    - *package.json* - defines the structure of the app
    - *routes/* - defines responses to requests
    - *public/* - static files
    - *node_modules/* - installed node packages
    - *bin/* - backend scripts

Activity

Create A Simple Web Server with npm and Express

Tutorial Checkpoint

What did we do?

- Used Express Generator to create project
- Installed nodemon
- Modified package.json to use nodemon

Get on Checkpoint 1

```
Ctrl+C
git checkout -b ${my branch name}
git commit -am 'saving my changes'
git checkout -t origin/checkpoint_1
npm install
npm start
```

Verify the Checkpoint

- Visit http://localhost:3000
  - Should see Express default view in browser window
- In node console should see:
  - `[nodemon] starting `node ./bin/www``

CONNECT TO DATA

- Mongoose is a node package used to talk to the MongoDB database
- Uses models (schema) to describe collections

Activity

Install Mongoose and connect to the DB

Tutorial Checkpoint

What did we do?

- mongoose package installed
- models.js created to connect to mongo using mongoose
- app.js modified to require model.js and call db.init()

Get on Checkpoint 2

```
Ctrl+C
git checkout -b ${my branch name}
git commit -am 'saving my changes'
git checkout -t origin/checkpoint_2
npm install
npm start
```

Verify the Checkpoint

- In node console should see:
  - "Successfully connected to database"

REST AND CRUD

- **RE**presentational **S**tate **T**ransfer
  - Stateless: server doesn't remember client, request contains everything needed
  - Retrieves or alters some resource on the server and returns a response
  - Accepts HTTP GET, POST, PUT and DELETE requests
- **C**reate, **R**ead, **U**pdate and **D**elete
  - A full-featured RESTful-API (**A**pplication **P**rogramming **I**nterface) should permit (authorized) users to perform these operations.

Activity

Create Express REST API to retrieve database records

Tutorial Checkpoint

What did we do?

- Added Star schema to models.js
- Required models.js in routes/index.js
- Added /stars route to query stars collection in mongo
- Checked for limit url parameter and set default otherwise

Get on Checkpoint 3

```
Ctrl+C
git checkout -b ${my branch name}
git commit -am 'saving my changes'
git checkout -t origin/checkpoint_3
npm install
npm start
```

Verify the Checkpoint

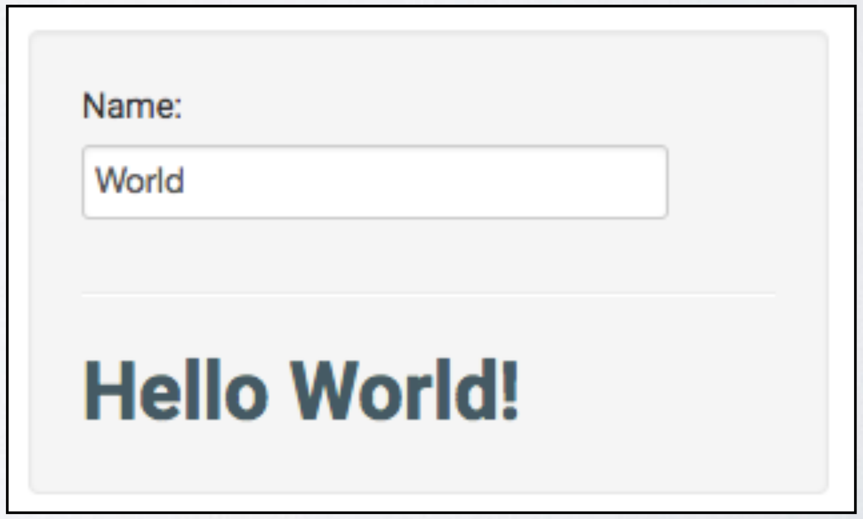
- Visit http://localhost:3000/stars
  - Should see 10 database records

- Visit http://localhost:3000/stars?limit=5
  - Should see 5 database records

ANGULARJS

- Client-side JavaScript framework to create dynamic HTML pages

```
<!doctype html>
<html ng-app>
  <head>
    <script src="scripts/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <hr>
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```

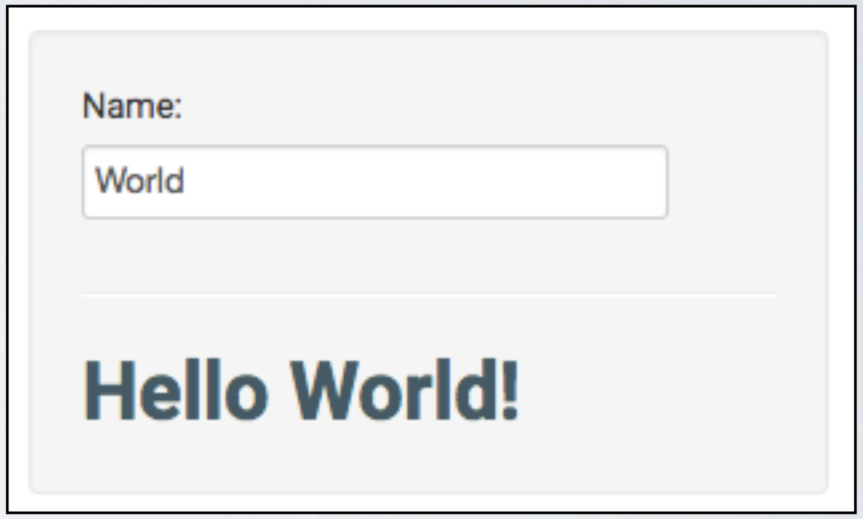

Name:

World

Hello World!

SYNTAX

```
<!doctype html>
<html ng-app>
  <head>
    <script src="scripts/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <hr>
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```


Name:

World

Hello World!

- Any text surrounded by double curly-braces **{{ }}** is interpreted by AngularJS and rendered in the view
  - *One-way binding*
- You can assign AngularJS attributes to HTML elements.
- We assign `ng-model="yourName"` to the text input box
  - *Two-way binding*
- So {{yourName}} is rendered as whatever is entered in the text input box.

Activity

AngularJS Hello World!

Tutorial Checkpoint

What did we do?

- Made the node_modules directory available as a public route
- Installed bootstrap/bootswatch to make our UI nicer
- Installed angularjs
- Added both as source files to public/index.html
- Created simple Hello World! AngularJS application

Get on Checkpoint 4

```
Ctrl+C
git checkout -b ${my branch name}
git commit -am 'saving my changes'
git checkout -t origin/checkpoint_4
npm install
npm start
```

Verify the Checkpoint

- Visit http://localhost:3000
  - "Hello !" text is modified as text is entered in input box

CONTROLLERS

- HTML elements can be bound to an AngularJS controller by assigning the ng-controller attribute to a wrapping div
    - `<div ng-controller="myController">...</div>`
- Controller provides additional functions and creates its own "scope"
    - Different elements on the same page can have different controllers
- Controllers can make REST calls, manipulate data, and validate form inputs

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.output = "";

    $scope.myfunc = function(test) {
      var myvar = 'foo';
      $scope.output = myvar+test;
    };
  });
```

```
<div ng-controller="myController">
<button class="btn" ng-click="myfunc('bar')">
  Click me
</button>
{{output}}
</div>
```

REST CALLS

- AngularJS has a built-in $http module to help us make the REST call

- Specify method and url, and a promise that takes a response and resolves to success or failure

```
angular.module('myApp', [])
  .controller('myController', function($scope, $http) {
    $scope.output = "";

    $scope.myApiCall = function() {
         $http({
           method: 'GET',
           url: '/apitest'
          }).then(function successCallback(response) {
           $scope.output=response.data;
          }, function errorCallback(response) {
           console.log(response);
         });
    };
  });
```

Activity
AngularJS Controller and REST Call

Tutorial Checkpoint

What did we do?

- Created angular module 'myApp'
- Created SearchController for myApp
- Defined columns to display
- Created doSearch function to get data from /stars
- Added button to execute doSearch()
- Added table with ng-repeats to display column headers and returned data
- Used ng-if to show magnitude errors if they exist

Get on Checkpoint 5

```
Ctrl+C
git checkout -b ${my branch name}
git commit -am 'saving my changes'
git checkout -t origin/checkpoint_5
npm install
npm start
```

Verify the Checkpoint

- Visit http://localhost:3000
  - Clicking "Fetch Data" button shows results in table

Activity

Create Search Form and Display Results

Tutorial Checkpoint

What did we do?

- Added form elements
- Created search object with regex and lt/gt logic in angular function
- Switched API from GET to POST
- Pass POST data to mongo find(), then return and render matching results

Get on Checkpoint 6

```
Ctrl+C
git checkout -b ${my branch name}
git commit -am 'saving my changes'
git checkout -t origin/checkpoint_6
npm install
npm start
```

Verify the Checkpoint

- Visit http://localhost:3000
  - Enter form data and check to see results match

Activity

Create Form Elements Using ng-repeat

Tutorial Checkpoint

What did we do?

- Used angular forEach and ng-repeat to expand form to query all magnitudes

Get on Checkpoint 7

```
Ctrl+C
git checkout -b ${my branch name}
git commit -am 'saving my changes'
git checkout -t origin/checkpoint_7
npm install
npm start
```

Verify the Checkpoint

- Visit http://localhost:3000
  - Enter form data and check to see results match

GEOJSON IN MONGODB

```
{ type: "Point", coordinates: [ 40, 5 ] }
```

- A GeoJSON "location" field in MongoDB consists of
  - a field named type that specifies the GeoJSON object type
  - a field named coordinates that specifies the object's coordinates.
- If specifying latitude and longitude coordinates, list the longitude first and then latitude:
  - Valid longitude values are between -180 and 180, both inclusive.
  - Valid latitude values are between -90 and 90 (both inclusive).

GEOJSON IN MONGODB

- MongoDB supports the $nearSphere query which returns points that are within a certain distance of a specified point, along with a calculated distance
  - $maxDistance and $minDistance are specified in meters assuming an Earth-sized sphere
  - So our angular separation query must be converted to meters. MongoDB's spherical geometry calculations assume an Earth size of mongo assumes an earth radius of 6378.1 km
  - Given this, 1 arcsecond ~= 30.88749994 meters

Activity
Create and query GeoJSON index

Tutorial Checkpoint

What did we do?

- Added form elements for positional search
- Updated schema in models.js to add location field
- Created conditional positional search query in routes/index.js
- Ran Mongo commands to create GeoJSON points

Get on Checkpoint 8

```
Ctrl+C
git checkout -b ${my branch name}
git commit -am 'saving my changes'
git checkout -t origin/checkpoint_8
npm install
npm start
```

Verify the Checkpoint

- Visit http://localhost:3000
  - Run positional search queries through UI to see if results match.
  - Try RA=302.495 DEC=-45.502 and radius=300

THINGS WE DIDNT TALK ABOUT

- Authentication/Authorization
- Using nginx to serve resources
- Talk to a sysadmin before deploying anything!

ACKNOWLEDGMENTS

- Special thanks to Arvind Gopu for assisting today
- IU's Scalable Compute Archives (SCA) Group members Raymond Perigo, Andrea Koenigsberger, and Patrick Etienne for feedback and testing
- Indiana University Research Technologies division
- Attendees of an early 'beta-test' version of this tutorial
- Peter Teuben and the rest of the ADASS XXVIII LOC