**ROUTING**

# Routing I

So far we've made AngularJS apps that display data in a single view **index.html**.

But what happens when the app grows and needs to display more information? Stuffing more code to a single view will quickly make things messy.

A better solution is to use multiple templates that display different pieces of data based on the URL that the user is visiting. We can do this with Angular's application routes.

**1.**

In **index.html** under the .header section, type in the code exactly as you see here:

<div ng-view></div>

**2.**

View the AngularJS app in the browser by typing <http://localhost:8000/>.

**3.**

In **app.js** under the angular.module, type in this code:

app.config(function ($routeProvider) { $routeProvider .when('/', { controller: 'HomeController', templateUrl: 'views/home.html' }) .otherwise({ redirectTo: '/' }); });

# Routing II

Great! A gallery of images shows up. How does it work?

1. In **app.js** inside the app.config() method, we use Angular's $routeProvider to define the application routes.
2. We used .when() to map the URL / to to the controller HomeController and the template home.html. The HomeController uses the service **js/services/photos.js** to fetch the array of all photos from <https://s3.amazonaws.com/codecademy-content/courses/ltp4/photos-api/photos.json> and stores it into $scope.photos. The home.html uses ng-repeat to loop through each item in the photosarray and display each photo.
3. Otherwise if a user accidentally visits a URL other than /, we just redirect to / using .otherwise().
4. Now when a user visits /, a view will be constructed by injecting home.html into the <div ng-view></div> in **index.html**.

**1.**

In **app.js** between the .when() and .otherwise(), add another .when() that maps the URL /photos/:id to the controller PhotoController and the template **views/photo.html**.

**2.**

Visit <http://localhost:8000/> and click on a photo. A detail page about that photo should appear.

# Routing III

What did we just do?

1. In **app.js**, we mapped a URL to PhotoController and photo.html. We added a variable part named id to the URL, like this: /photos/:id.
2. In **PhotoController**, we used Angular's $routeParamsto retrieve id from the URL by using $routeParams.id. Notice that we injected both $routeParams and the photos service into the PhotoController dependency array to make them available to use inside the controller.
3. Then to fetch an individual photo, we first used the photos service to fetch the array of photos from the server, and then used $routeParams.id to access the specific photo by its index.
4. As before, any properties attached to $scope become available to use in the view. This means in **photo.html**, we can display the photo's detail using expressions as done before.

Notice that when you click on links, the app doesn't do a full reload. Only the part of the view specified by <div ng-view></div> changes.

**1.**

Finish the template so that it displays a photo's details.

First, in the browser, visit <https://s3.amazonaws.com/codecademy-content/courses/ltp4/photos-api/photos.json>. Looking at the format of the data in the array, each photo has six pieces of data - title, author, url, pubdate, upvotes, and views.

**2.**

In the template **photo.html**, display a photo's remaining five pieces of data. Then visit <http://localhost:8000/> in the browser and click on a photo. A detail page about that photo should appear.

**3.**

Use the number filter to format the views and upvotes.

**4.**

Use the date filter to format the pubdate.

**5.**

View the result in the browser. The photo's views, upvotes, and date should now be formatted.

Generalizations

Why are routes useful? Instead of filling a single view with more code than needed, routes let us map URLs to self-contained controllers and templates. Furthermore, now that the app has URLs, users can easily bookmark and share the app's pages.

What can we generalize so far?

* Directives are a way to make standalone UI components, like <app-info>
* Services are a way to make standalone communication logic, like forecast which fetches weather data from a server
* Routes are a way to manage apps containing more views.