
STAYING
SHARP
with
Angular.JS

Services Level 3

Another Factory Section 2

STAYING SHARP with Angular.JS

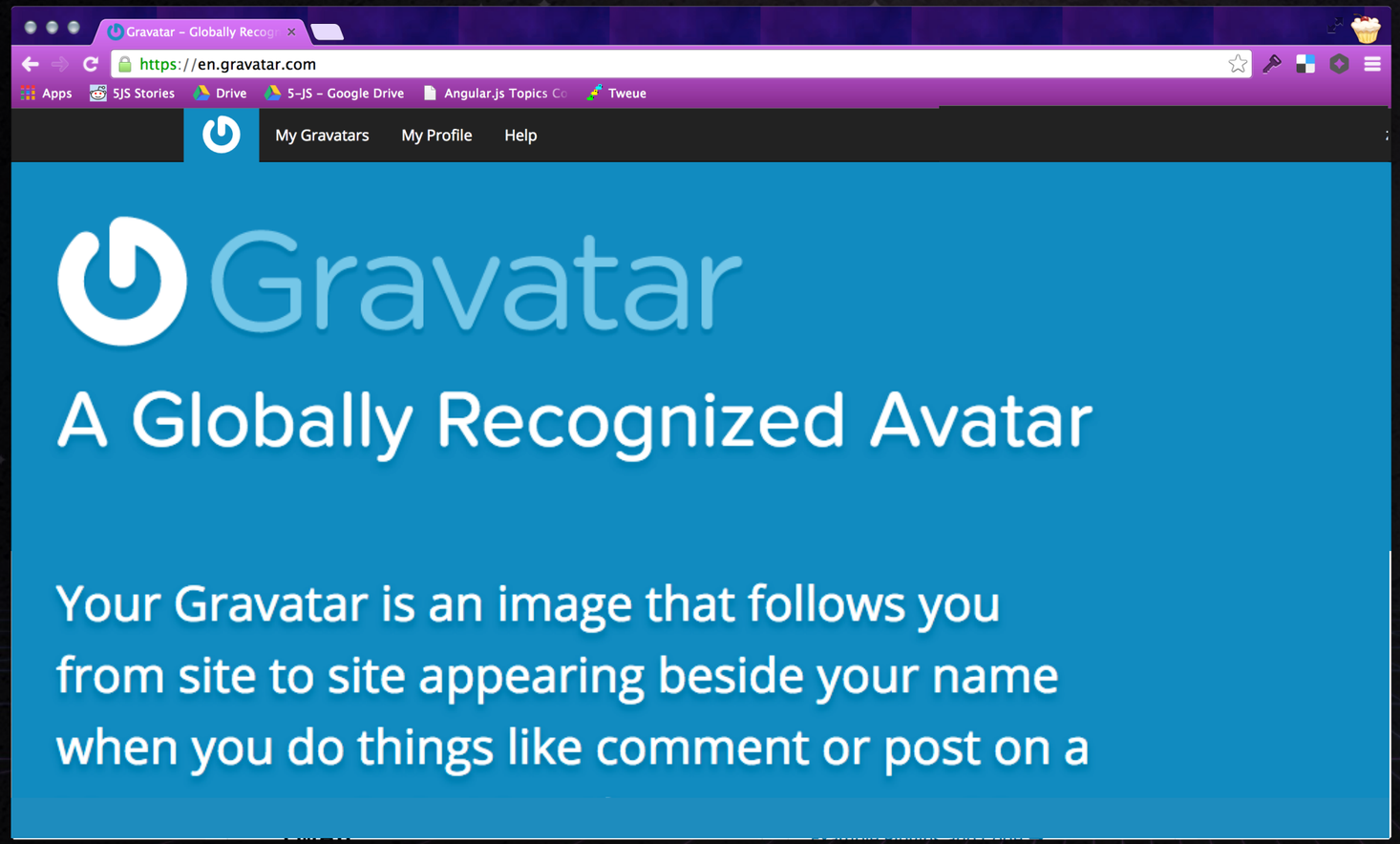
Our Problem

We want to show images for each of our users.

Each user should have a glamorous photo!

One Solution Is Gravatar

Gravatar is a free service that allows people to upload images associated with an email address.



Then our application can use the image associated with our user's email.

Providing Our App With Gravatar Images

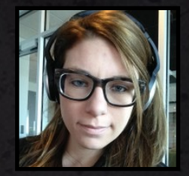
In order to access someone's Gravatar, we need to hash their email address and append it to a URL.

1. Hash user's email address into a hash.

```
alyssa@codeschool.com → bf4ee76b5f3a6bfed26bca5460bc3f22
```

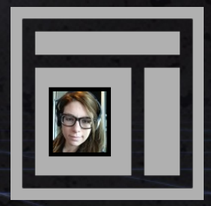
2. Add this hash onto a Gravatar URL.

```
http://www.gravatar.com/avatar/bf4ee...png
```



3. Use this URL in a template.

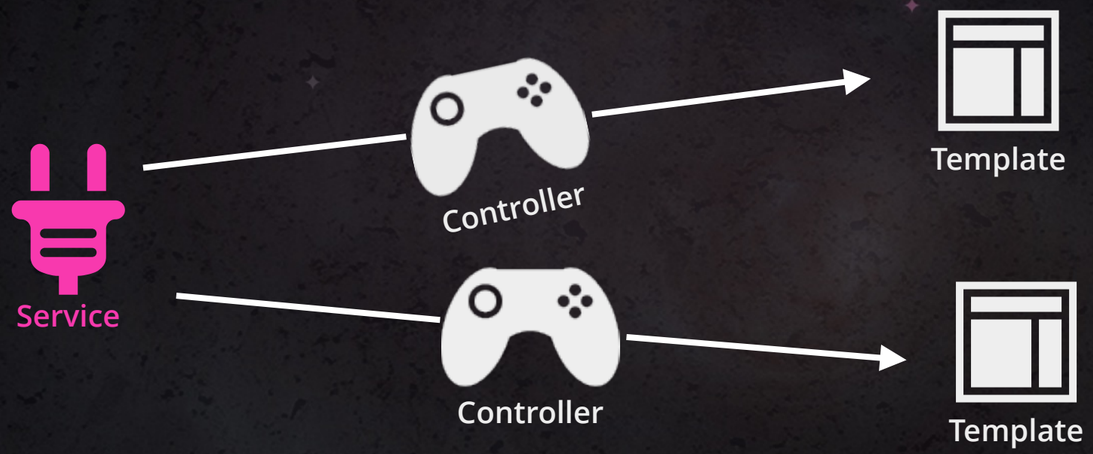
```
<img ng-src='http://...bf4ee...png'/>
```



Template

Shared Functionality?! Service Time!

We need to generate these avatar links for each user and use them inside multiple controllers.



Creating the Service

Let's use the Factory recipe again to create a Gravatar-image-grabbing service.

services/gravatar.js

```
angular.module("NoteWrangler")
.factory( "Gravatar",function GravatarFactory() {
  return { };
});
```

Customizable Avatar Size Variable

We need some reusable variables to generate each user's avatar link. Creating these outside of our factory's return will ensure they are private to this service.

services/gravatar.js

```
angular.module("NoteWrangler")
.factory("Gravatar", function GravatarFactory() {
  var avatarSize = 80; // Default size
  return { };
});
```

Customizable Avatar Size Variable

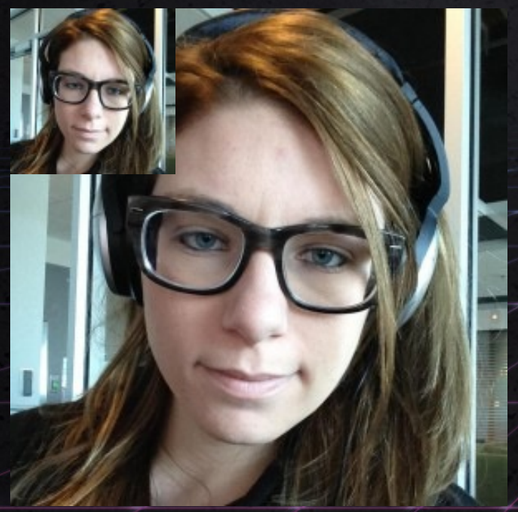
services/gravatar.js

```
angular.module("NoteWrangler")
.factory("Gravatar", function GravatarFactory() {
  var avatarSize = 80; // Default size
  return { };
});
```

http://www.gravatar.com/avatar/bf4ee76b5f3a6bfed26bca5460bc3f22?size=80.png

?size=100

?size=500



We can customize the size of the image we receive from Gravatar by adding the size parameter to the end of the URL.

Reusable Gravatar Link Variable

Along with a default avatar size to append to the link, we need a reusable link variable for the Gravatar site.

services/gravatar.js

```
angular.module("NoteWrangler")
.factory("Gravatar", function GravatarFactory() {
  var avatarSize = 80; // Default size
  var avatarUrl = "http://www.gravatar.com/avatar/";
  return { };
});
```

http://www.gravatar.com/avatar/bf4ee76b5f3a6bfed26bca5460bc3f22?size=80.png

Gravatar Steps

Back to the steps for how our app can use the Gravatar image:

1. Hash user's email into a hash.

`alyssa@codeschool.com` → `bf4ee76b5f3a6bfed26bca5460bc3f22`

2. Add this hash onto a Gravatar URL.

`http://www.gravatar.com/avatar/bf4ee...png`

3. Use this URL in a template.

```
<img ng-src='http://...bf4ee...png'/>
```

Hashing the Email

Now for the first step: how to hash-ify a user's email.

1. Hash user's email into a hash.

`alyssa@codeschool.com` → `bf4ee76b5f3a6bfed26bca5460bc3f22`

CryptoJS is a simple JavaScript library that has an `.MD5` function. Pass this function a string and it will return that string hash-ified.

```
CryptoJS.MD5(email)
```

Include this link to give us access to the above call:

index.html

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.1.2/build/rollups/md5.js"></script>
```

Gravatar Steps

Back to the steps for providing our application with Gravatar images:

1. Hash user's email into a hash.

2. Add this hash onto a Gravatar URL.

```
http://www.gravatar.com/avatar/bf4ee...png
```

Generating Avatar Links With MD5 Hash

Right now our function is only returning the hashed user email. We need to append it to the link. While we are generating the link, let's go ahead and append a default size.

services/gravatar.js

```
angular.module("NoteWrangler")
.factory("Gravatar", function GravatarFactory(){
  var avatarSize = 80; // Default size
  var avatarUrl = "http://www.gravatar.com/avatar/";
  return {
    generate: function(email){
      return avatarUrl + CryptoJS.MD5(email) + "?size=" + avatarSize.toString();
    }
  };
});
```

Using Gravatar URL Service in Template

services/gravatar.js

```
angular.module("NoteWrangler")
.factory("Gravatar", function GravatarFactory(){
  var avatarSize = 80; // Default size
  var avatarUrl = "http://www.gravatar.com/avatar/";
  return {
    generate: function(email){
      return avatarUrl + CryptoJS.MD5(email) + "?size=" + avatarSize.toString();
    }
  };
});
```

To call this service, we need to inject it, and then call the generate function.

```
Gravatar.generate("alyssa@codeschool.com")
```

Simplifying Our One Function Factory

services/gravatar.js

```
angular.module("NoteWrangler")
.factory("Gravatar", function GravatarFactory(){
  var avatarSize = 80; // Default size
  var avatarUrl = "http://www.gravatar.com/avatar/";
  return function(email){
    return avatarUrl + CryptoJS.MD5(email) + "?size=" + avatarSize.toString();
  };
});
```

If a factory service only has one function, we can simplify it.

```
Gravatar("alyssa@codeschool.com")
```

Calling Factory Service in a Controller

users-index-controller.js

```
angular.module("NoteWrangler")
.controller("UsersIndexController", function($scope, Gravatar) {
  $scope.gravatarUrl = function(email) {
    return Gravatar(email);
  }
});
```

Templates Have Access to User's Gravatar URLs

Our templates now have access to each user's Gravatar URL.

templates/users/index.html

```
<div class="users-wrapper">
  <a class="card-users" ng-repeat="user in users" ...>
    <nw-card header="user.name"
             image="gravatarUrl(user.email)"
             body="user.bio"
             id="user.id">
    </nw-card>
  </a>
</div>
```

```
scope: {
  image: "=",
  ...
}
```

Now We Can Display the Image

Use ngSrc to load it.

templates/directives/nw-card.html

```
<div class="card" title="{{header}}">

  <img ng-src='{{image}}' ng-if='image'>

  <div ng-if='icon'>
    <i class="icon icon-card {{icon}}"></i>
  </div>

  <h2 class="h3">{{header}}</h2>
  ...
</div>
```

AlyssaNicoll

Users now have glamorous photos

5 Service Recipes

- Value — Used often

  The simplest service recipe used for sharing a value that is used throughout your app repeatedly.

- Factory — Most commonly used

  Shares methods and objects.

- Service — Rarely used

  Shares instances of methods and objects.

- Provider — Commonly used

  Shares methods and objects (like a Factory), but allows for configuration.

- Constant — Used less often

  Shares a value within application configuration.