AngularJS + Rails

Jumpstart your front end web applications using Angular and Rails

Jonathan Birkholz & Jesse Wolgamott

AngularJS + Rails

Jumpstart your front end web applications using Angular and Rails.

Jonathan Birkholz and Jesse Wolgamott

This book is for sale at http://leanpub.com/angularails

This version was published on 2014-03-05

Tweet This Book!

Please help Jonathan Birkholz and Jesse Wolgamott by spreading the word about this book on Twitter!

The suggested hashtag for this book is #angularails.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#angularails

Contents

1 Introduction

Thanks for taking the time to read our guide!

This started out as just a hodge podge of notes and exercises from workshops we organized to help teach people AngularJS. We are Rails guys so our training naturally had instructions and tips on how to integrate AngularJS with Rails. Pro tip: It is super easy.

We loved our training material and kept referring back to it while we developed. That is when we had the idea to turn our notes into a comprehensive guide.

1.1 What this guide is

- A great starting point to learning AngularJS.
- An easy to follow step-by-step guide to using AngularJS in Rails.
- A place to learn easy ways in which AngularJS integrates with Rails.
- A great reference to refer back to when building your application.
- In short, a rapid way to get productive with AngularJS and Rails.

1.2 What this guide isn't

- An exhaustive guide to all things AngularJS. We will get the ball rolling and you building your application. You can then use other resources when you get to the difficult subjects.
- A step-by-step guide to learning Rails. We assume you have some experience with Rails. Although no experience is required to go through the guide, we don't spend a lot of time explaining the Rails side of things.
- The end of your developing journey. We are the staring point. We all have time crunches and this guide will get you up to speed and building awesomeness fast.

2 Why Angular?

Every time you check the Interwebs, there seems to be a new shiny JavaScript framework. The client-side framework space has seen a lot of activity the past few years. Every framework has their fanatical element whose honor is challenged if someone else picks a different framework for their application.

In reality, every framework has its own strengths and weaknesses. Not one of the frameworks is inherently better than another; instead, they just offer different solutions to similar problems. So in a world of infinite client-side javascript frameworks, why would you choose Angular?

2.1 Pros: The greener grass

Binding! Yay!

AngularJS uses binding to update content on the page. The binding syntax works by adding simple attributes to existing DOM elements. These attributes are known in AngularJS as directives. Directives offer far more than just binding constructs, but that is for a later chapter. With AngularJS, it is sufficient to know there is no need for a separate template engine. What this means in a Rails application is that you can easily use AngularJS with your existing view engines like ERB or HAML.

AngularJS is super awesome because you can bind to simple, native JSON. Pull data down from an API, and just slap it onto $scope. BOOM! Fast actin' binding action.

Binding to JSON is easy

```
$http({ method: "GET", url: "http://some_url/books"})
  .success(function(response) {
    $scope.books = response.books
  });
```

Flexibility

AngularJS's greatest strength is its flexibility. AngularJS can grow and shrink to fit the dynamic evolution of your application. You can have single components or multiple AngularJS apps operating on the same page. Your application can be just a dynamic page or a complete single-page application with all the features you need baked in.

You can also easily integrate AngularJS with other libraries and tools. When you use a framework, the general rule is to do everything in the framework. i.e. no jQuery in your views, viewmodels, controllers, whatever. But sometimes you need to get something to work and it doesn't have to be pretty. AngularJS is more fluid, allowing you to be naughty inside your different controllers or views. Color outside the lines and apply some duct tape. Refactor when the dust settles.

Scalability

Part of the flexibility is AngularJS's scalability. You can start upgrading one page in an existing application. Then you can upgrade another, and another. You then can turn multiple pages into a Single Page Application using routing. Whether your AngularJS features are in a single page application or just a simple upgrade, AngularJS is easy to drop in and scale.

Embraces JavaScript's strengths

One of JavaScript's strengths is its functional nature. AngularJS embraces that with no complicated object models. You define functions and bind to native JSON. AngularJS provides simple lightweight mechanisms to start building an application without having to learn a large complicated framework.

2.2 Cons: Here be dragons

Binding! Yuck!

Some people are completely averse to binding. Often binding-phobes have the understandable desire to know the DOM is going to be updated. So if you dislike binding voodoo, then AngularJS isn't the framework for you.

No model layer

For some, the model is the single source of truth. Having the model's state and actions in one class is a comfortable object-oriented approach to providing structure to an application's domain. Since AngularJS binds to native JSON, you don't need a fancy model layer. With AngularJS, you will find that the path of least resistance is to have domain related services that operate on your simple JSON. This is one of the ways AngularJS is more functional and embraces JavaScript's strengths.

This means that instead of saving a user like `user.save()`, you would have a `User` service you inject into the controller and call `User.save($scope.user)`. We view this as a big win because testing a function is super easy!

No framework enforced guidance on building large applications

If you are building a large application from the ground up, you will need to do some work with AngularJS to enforce best practices and code structure. The power of AngularJS is its flexibilty, but with that power comes the burden of being responsible with your code. In the wrong hands, you can create an ugly beast of spaghetti code. While generally true with any framework, there are some frameworks that enforce some semblance of structure. When it comes to large applications, you will have to create your own guidance and just enforce the patterns through code reviews.

Conclusion

If you don't mind the binding, and the idea of simply binding to JSON is attractive to you, then you should take a serious look at AngularJS. AngularJS is flexible and terse allowing you to deliver extraordinary features with fewer lines of code. AngularJS is a fantastic tool to have in your toolbox and is a go-to framework for delivering awesome client-side UX.

3 Hello AngularJS

But enough with the talking points. Let's get started with the very basics of setting up AngularJS in Rails.

What is covered:

- Basic Rails application creation
- Creating an AngularJS application with `ng-app`
- Simple view binding using {{ }}
- Binding a value to a text input with `ng-model`
- Creating and using an AngularJS controller with `ng-controller`
- Setting a value using `$scope`

3.1 Create your Rails Application

If you are a Rails veteran and have your own flavor of Rails, then skip this section. But if you are relatively new to Rails, or are just curious as to the setup we are using for this guide, continue reading.

First we will create a new rails application and generate all the folders and files that Rails will use.

terminal

```
$ rails new angularails
```

The default Rails generators, generate a bunch of junk we don't need for this guide. Junk as in… tests. Who needs those! (PSA: We actually love tests and is one of the reasons Angular is so awesome!)

config/application.rb

```
config.generators do |g|
  g.stylesheets false
  g.helper false
  g.javascripts false
  g.test_framework false
end
```

We are going to add a few gems to our `Gemfile`.

First we are going to add `active_model_serializers`. There are many options for how you handle model serialization with Rails. Active Model Serializers is a great and popular option which we will be employing here.

We are also going to be adding `font-awesome-rails`. We are installing the font icons from FontAwesome using the gem just to simplify our setup.

We are also going to remove some default gems we won't be using. In the end, your `Gemfile` should look like:

Gemfile

```ruby
source 'https://rubygems.org'

# Rails
gem 'rails', '4.0.0'

# Use sqlite3 as the database for Active Record
gem 'sqlite3'

# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.0'

# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'

# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'

# Use jquery as the JavaScript library
gem 'jquery-rails'

# Model serialization
gem 'active_model_serializers'

# Font Awesome icons
gem 'font-awesome-rails'

# Paging
gem 'kaminari'
```

Add CSS and JavaScript

We will be using Twitter Bootstrap for our basic styling. We also will be using `angular.min.js` (duh!).

You can get the files at `http://www.angularails.com/files`

Add the `bootstrap.min.css` file to the `assets/stylesheets` folder. We won't change the default `application.css` file and let `require_tree` load the file for us.

Next we will add the both the `bootstrap.min.js` and `angular.min.js` JavaScript files to the `assets/javascripts` folder.

Now for the fun stuff! Time to bring in AngularJS. First we are going to remove the `require_tree` from `application.js`. There is just something unsettling about loading JavaScript in any order we didn't specify directly.

app/assets/javascripts/application.js

```
  //= require jquery
  //= require jquery_ujs
  //= require bootstrap.min
  //= require angular.min
```

Using Twitter Bootstrap Styling

Replace your `application.html.erb` with:

application.html.erb

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><%= content_for?(:title) ? yield(:title) : "Angularails" %></title>
    <%= csrf_meta_tags %>
    <%= stylesheet_link_tag "application", :media => "all" %>
    <%= javascript_include_tag "application" %>
  </head>
  <body>
    <div class="navbar navbar-inverse">
      <div class="container">
        <a class="navbar-brand" href="#">Angularails</a>
      </div>
    </div>

    <div class="container">
      <%= yield %>
    </div>
  </body>
</html>
```

3.2 Hello AngularJS

Now we will use the Rails generator to create a simple controller and view.

terminal

```
$ rails g controller HelloAngular show
```

This will create a `HelloAngular` controller with a `show.html.erb` view.

Replace the generated markup in the `show.html.erb` view with:

views/hello_angular/show.html.erb

```
1 <div class="row" ng-app>
2   <div class="col-md-6">
3     <h1>Hello AngularJS</h1>
4     <p>Type in the input box and watch as AngularJS updates the property through \
5 the magic of binding.</p>
6     <input type="text" class="form-control" ng-model="myText" >
7     {{ myText }}
8   </div>
9 </div>
```

Line 1 `ng-app`
By setting `ng-app` on a HTML element we are creating an Angular application whose scope is passed to its children.

Line 6 `ng-model`
This will bind the value from the input to the `myText` property on our application scope.

Line 7 `{{ myText }}`
The mustache curly braces will display the current value of `myText`.

Try it out!

Time to test if we did everything right. Start up your rails server and go to `http://localhost:3000/hello_angular/show` As we type, we should see our input duplicated besides the input box. Awesome, right? Super cool.

3.3 Using an AngularJS Application and Controller

While the binding is cool, we will need an AngularJS application and a controller to do anything more complicated.

We will first create a JavaScript file to load our application and all our future AngularJS files.

Create javascripts/angular-application.js

```
1  //= require_self
2  //= require_tree ./angular
3
4  AngulaRails = angular.module("AngulaRails", []);
```

Line 1 `require_self`
This will load the current file before it loads the other files in the `angular` folder we will create.

Line 2 `require_tree ./angular`
This will load all the files we will create in the `angular` folder. This will be controllers, routes, directives, and any other javascript file we create in that folder.

Line 4 Creating the application
This will create our Angular application that we are naming `AngulaRails`. You can name this whatever you want.

`angular-application.js` will use sprockets to load all our future AngularJS files (like controllers) allowing those files to reference our `AngulaRails` Angular application.

Now we need to add `angular-application` to our `application.js` so that the file gets loaded.

javascripts/application.js

```
//= require angular-application
```

Create an `angular` folder under the `assets/javascripts` folder.

Inside the `angular` folder, create a new file:

javascripts/angular/hello_world_controller.js.coffee

```
AngulaRails.controller "HelloWorldController", ($scope) ->
  $scope.myText = "Hello World"
```

Line 1: `$scope`
: `$scope` is an injected variable on which we attach properties and methods we want to reference in our views.

Line 2: `$scope.myText`
: We don't have to initialize `$scope.myText` to use that property on our view, but for demonstration purposes we will initialize it with "Hello World".

Back in our view, we can now reference the `AngulaRails` application and the `HelloWorldController` we just created.

views/hello_angular/show.html.erb

```
<div class="row" ng-app="AngulaRails" ng-controller="HelloWorldController">
  <div class="col-md-6">
    <h1>Hello AngularJS</h1>
    <p>Type in the input box and watch as AngularJS updates the property through \
the magic of binding.</p>
    <input type="text" class="form-control" ng-model="myText" >
    {{ myText }}
  </div>
</div>
```

Line 1: `ng-app`
: This sets the Angular application to our `AngulaRails` that we created in `angular-application.js`.

Line 1: `ng-controller`
: This will set our controller to the `HelloWorldController` we created in `hello_world_controller.js.coffee`.

Try it out!

Reload the page at `http://localhost:3000/hello_angular/show`. Now the input should start with the "Hello World" value we set in our controller.

Conclusion

Nothing crazy or exciting right? Just basic wiring and some simple setup. Future exercises in this guide will assume this setup. We recommend saving a base copy of this Rails + AngularJS application so you can use it as a common starting point.

4 Extending Rails Forms with AngularJS Validation

One of the amazing things you can do with AngularJS is extend exisiting Rails forms. Crazy, right?

You can just add AngularJS directives to your existing Rails form view to provide additional functionality. Your Rails backend will remain unchanged. The form POST will still go to your controller. Even your tests will still pass!

What is covered:

- Adding AngularJS to an existing Rails form
- AngularJS HTML5 validation
- AngularJS validation
- Toggle classes with `ng-class`
- Toggle visibility of an HTML element using `ng-show`
- Disable a button using `ng-disable`

Setup

For our example, we are going to make Widgets. Widgets have a name and a price.

terminal

```
$ rails g scaffold Widget name:string price:integer
$ rake db:migrate
```

Now there is some important validation for making a Widget. * Name is required and must be at least 7 characters * Price is required and must be a number greater than 10.

Why? Because of reasons.

This validation is straight forward on the Rails side.

models/widget.rb

```ruby
class Widget < ActiveRecord::Base
  validates :price, presence: true, numericality: { greater_than: 10 }
  validates :name, presence: true, length: { minimum: 7 }
end
```

We will need to change our scaffolded for a bit to look like this:

views/widgets/_form.html.erb

```erb
<%= form_for(@widget) do |f| %>
  <% if @widget.errors.any? %>
    <div class="alert alert-danger">
      <h4><%= pluralize(@widget.errors.count, "error") %></h4>

      <ul>
      <% @widget.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="form-group">
    <%= f.label :name %>
    <%= f.text_field :name, class: "form-control" %>
  </div>
  <div class="form-group">
    <%= f.label :price %>
    <%= f.number_field :price, class: "form-control" %>
  </div>
  <%= f.submit "Save", class: "btn btn-primary" %>
<% end %>
```

Try it out!

If we go to `http://localhost:3000/widgets`, we can then use the Rails scaffolding to add widgets. If we try to create an invalid widget, Rails will return the validation error messages.

4.1 HTML5 Validation with AngularJS

Let’s add our AngularJS application to the form and turn off browser default validation. We also will need to give our form a name for validation in AngularJS to work.

views/widgets/_form.html.erb

```
<div ng-app="AngulaRails">
  <%= form_for @widget, html: { name: "widgetForm", "novalidate" => true } do |f|\
 %>
    <% if @widget.errors.any? %>
      <div class="alert alert-danger">
        <h4><%= pluralize(@widget.errors.count, "error") %></h4>

        <ul>
        <% @widget.errors.full_messages.each do |msg| %>
          <li><%= msg %></li>
        <% end %>
        </ul>
      </div>
    <% end %>
    <div class="form-group">
      <%= f.label :name %>
      <%= f.text_field :name, class: "form-control" %>
    </div>
    <div class="form-group">
      <%= f.label :price %>
      <%= f.number_field :price, class: "form-control" %>
    </div>
    <%= f.submit "Save", class: "btn btn-primary" %>
  <% end %>
</div>
```

AngularJS can use the HTML5 validation attributes for its validation. Since both fields are required, lets add the `required` attribute to each input. We can also add the HTML5 validation attribute `min` to the price input.

Lets add some styling so the input fields will change color to show if the value is invalid.

And finally, we will disable the `Save` button if the form is invalid.

views/widgets/_form.html.erb

```
<div ng-app="AngulaRails">
  <%= form_for @widget, id: "widgetForm", html: { name: "widgetForm", "novalidate\
" => true } do |f| %>
    <% if @widget.errors.any? %>
      <div class="alert alert-danger">
        <h4><%= pluralize(@widget.errors.count, "error") %></h4>

        <ul>
        <% @widget.errors.full_messages.each do |msg| %>
          <li><%= msg %></li>
        <% end %>
        </ul>
      </div>
    <% end %>
    <div class="form-group" ng-class="{ 'has-error': widgetForm['widget[name]'].$\
invalid }">
      <%= f.label :name %>
      <%= f.text_field :name, class: "form-control", "ng-model" => "name", requir\
ed: true %>
    </div>
    <div class="form-group" ng-class="{ 'has-error': widgetForm['widget[price]'].\
$invalid }">
      <%= f.label :price %>
      <%= f.number_field :price, class: "form-control", "ng-model" => "price", re\
quired: true, "min" => "10" %>
    </div>
    <%= f.submit "Save", class: "btn btn-primary", "ng-disabled" => "widgetForm.$\
invalid" %>
  <% end %>
</div>
```

Try it out!

Go to `http://localhost:3000/widgets/new` The inputs should have a red border signifying they are invalid. The submit button should also disabled. Once you type something in the input, the invalid border should go away.

Display the error message

While the red border is helpful, it would be nice to show an error message to the user so they know why the input is invalid.

views/widgets/_form.html.erb

```
<div ng-app="AngulaRails">
  <%= form_for @widget, id: "widgetForm", html: { name: "widgetForm", "novalidate\
" => true } do |f| %>
    <% if @widget.errors.any? %>
      <div class="alert alert-danger">
        <h4><%= pluralize(@widget.errors.count, "error") %></h4>

        <ul>
        <% @widget.errors.full_messages.each do |msg| %>
          <li><%= msg %></li>
        <% end %>
        </ul>
      </div>
    <% end %>
    <div class="form-group" ng-class="{ 'has-error': widgetForm['widget[name]'].$\
invalid }">
      <%= f.label :name %>
      <%= f.text_field :name, class: "form-control", "ng-model" => "name", requir\
ed: true %>
      <span class="help-block" ng-show="widgetForm['widget[name]'].$error.require\
d">Required</span>
    </div>
    <div class="form-group" ng-class="{ 'has-error': widgetForm['widget[price]'].\
$invalid }">
      <%= f.label :price %>
      <%= f.number_field :price, class: "form-control", "ng-model" => "price", re\
quired: true, "min" => "10" %>
      <span class="help-block" ng-show="widgetForm['widget[price]'].$error.requir\
ed">A number is required</span>
      <span class="help-block" ng-show="widgetForm['widget[price]'].$error.min">M\
ost be greater than 10</span>
    </div>
    <%= f.submit "Save", class: "btn btn-primary", "ng-disabled" => "widgetForm.$\
invalid" %>
  <% end %>
</div>
```

Try it out!

Go to `http://localhost:3000/widgets/new` Now we have an appropriate error message to inform the user why the input is invalid.

4.2 Using AngularJS Validation

Not all our validations fall within HTML5 validation. AngularJS provides a supplement to those validation rules. The API documentation found at `http://angularjs.org/` lists all the validations AngularJS provides and naturally this will change over time. For our example, we are going to use the `ng-minlength` validation directive to enforce our name length validation rule.

views/widgets/_form.html.erb

```
<div ng-app="AngulaRails">
  <%= form_for @widget, id: "widgetForm", html: { name: "widgetForm", "novalidate\
" => true } do |f| %>
    <% if @widget.errors.any? %>
      <div class="alert alert-danger">
        <h4><%= pluralize(@widget.errors.count, "error") %></h4>

        <ul>
        <% @widget.errors.full_messages.each do |msg| %>
          <li><%= msg %></li>
        <% end %>
        </ul>
      </div>
    <% end %>
    <div class="form-group" ng-class="{ 'has-error': widgetForm['widget[name]'].$\
invalid }">
      <%= f.label :name %>
      <%= f.text_field :name, class: "form-control", "ng-model" => "name", requir\
ed: true, "ng-minlength" => "7" %>
      <span class="help-block" ng-show="widgetForm['widget[name]'].$error.require\
d">Required</span>
      <span class="help-block" ng-show="widgetForm['widget[name]'].$error.minleng\
th">Minimum length of 7</span>
    </div>
    <div class="form-group" ng-class="{ 'has-error': widgetForm['widget[price]'].\
$invalid }">
      <%= f.label :price %>
      <%= f.number_field :price, class: "form-control", "ng-model" => "price", re\
quired: true, "min" => "10" %>
      <span class="help-block" ng-show="widgetForm['widget[price]'].$error.requir\
ed">Required</span>
      <span class="help-block" ng-show="widgetForm['widget[price]'].$error.min">M\
ost be greater than 10</span>
    </div>
    <%= f.submit "Save", class: "btn btn-primary", "ng-disabled" => "widgetForm.$\
invalid" %>
  <% end %>
</div>
```

Try it out!

Go to `http://localhost:3000/widgets/new` With our final addition, our validation for a Widget is enforced on the client and server side. The user is also presented with a better user experience.

Conclusion

The ability to extend an existing Rails form highlights AngularJS's great flexibility. You can provide a greater user experience with very little effort. In this example we simply added validation, but extending a Rails form with AngularJS can be used for other features.

5 HTTP with AngularJS

Life on the web would be boring without HTTP requests. Have no fear, in this chapter we will be making HTTP requests to GitHub to demonstrate the basics on binding to data that is pulled from GET requests.

What we will cover:

- Using the `$http` provider to make HTTP requests
- Another use of our good friends `ng-model` and `ng-submit`
- Using `ng-repeat` to loop over an array
- Using `ng-show` to show an HTML element
- Using jQuery to make a HTTP request

Setup

First we will create a `GitHttp` controller with a `show.html.erb` view.

terminal

```
> rails g controller GitHttp show
```

5.1 GET request with $http

We are going to start with our controller and make a GET request to GitHub using AngularJS's `$http` provider. Like the `$scope` provider, `$http` is a service that AngularJS provides which can be injected into your controller for use.

javascripts/angular/git_http_controller.js.coffee

```
AngulaRails.controller "GitHttpController", ($scope, $http) ->

  $scope.search = () ->
    url = "https://api.github.com/users/#{$scope.username}/repos"
    $http({ method: "GET", url: url })
      .success (data) ->
        $scope.repos = data
```

Line 1 `$http`

AngularJS comes packaged with a set of services. `$http` allows for async http requests using promises. Line 4 Getting the username

We use the current value of `$scope.username` to create the url to the GitHub api. Line 7 Binding to JSON

With Angular, we can just stick the JSON we get back from the http request right into our `$scope`. No need to wrap it in any other object.

Now let's create the search from on the `show` page.

views/git_http/show.html.erb

```
 1 <div class="row" ng-app="AngulaRails" ng-controller="GitHttpController">
 2   <div class="col-md-6">
 3     <h1>Git $http</h1>
 4     <form ng-submit="search()">
 5       <div class="input-group">
 6         <span class="input-group-addon">
 7           <i class="icon-search"></i>
 8         </span>
 9         <input type="text" class="form-control" placeholder="Git Username" ng-mod\
10 el="username">
11       </div>
12     </form>
13   </div>
14 </div>
```

Line 1 `ng-app` & `ng-controller`

Set our application and our controller. Line 4 `ng-submit`

When we submit this form, execute the `search()` method on our controller. Notice in this case we don't have a submit button. Search is initiated by hitting return while inside the input Line 7 `ng-model`

Bind the value of the input to `username`. We then use this value when we execute our search.

Now lets take the array of git hub repos we pulled down with our $http GET request and display them in a list.

views/git_http/show.html.erb

```
 1 <div class="row" ng-app="AngulaRails" ng-controller="GitHttpController">
 2   <div class="col-md-6">
 3     <h1>Git $http</h1>
 4     <!-- THE FORM -->
 5 
 6     <hr />
 7 
 8     <ul class="list-group">
 9       <li class="list-group-item" ng-repeat="repo in repos">
10         <h4 class="list-group-item-heading">{{ repo.full_name }}</h4>
11         <p class="list-group-item-text text-muted">{{ repo.url }}</p>
12       </li>
13     </ul>
14   </div>
15 </div>
```

Line 6 `ng-repeat`

This will iterate over all the objects in the array found in the `repos` property. Each object is set to `repo`. Line 8,9 Bind to `repo`

We can now use the `repo` variable set by `ng-repeat` to display properties on the object.

Try it out!

Go to `http://localhost:3000/git_http/show`. If you search for a valid username, you should then see the list of their public repos. If you don't have a GitHub account, then you can search on our usernames (`rookieone` or `jwo`).

5.2 Adding a spinner

Let's clear the repos and add a spinner to display as we make our request.

javascripts/angular/git_http_controller.js.coffee

```
 1  AngulaRails.controller "GitHttpController", ($scope, $http) ->
 2
 3    $scope.search = () ->
 4      $scope.repos = []
 5      $scope.searching = true
 6      url = "https://api.github.com/users/#{$scope.username}/repos"
 7      $http({ method: "GET", url: url })
 8        .success (data) ->
 9          $scope.searching = false
10          $scope.repos = data
```

Line 4: Clear repos

Clear array in `$scope.repos`. Line 5

Set searching to true before we make our http request. Line 9

Set searching to false after we get our result.

views/git_http/show.html.erb

```
1  <i class="icon-refresh icon-spin" ng-show="searching"></i>
2
3  <ul class="list-group">
4    <li class="list-group-item" ng-repeat="repo in repos">
5      <h4 class="list-group-item-heading">{{ repo.full_name }}</h4>
6      <p class="list-group-item-text text-muted">{{ repo.url }}</p>
7    </li>
8  </ul>
```

Line 1: `ng-show`

Display the HTML element if `searching` evaluates as true. In this scenario we are looking the `searching` property, but this could be a javascript expression. ie. `repos.length == 0`

FYI ng-hide

There is an `ng-hide` as well. Instead of showing an element when the expression evaluates to true, it hides the element.

Try it out!

Reload `http://localhost:3000/git_http/show`. The search is pretty fast, but you should see a spinner briefly between searches. You can also add a timeout to slow the search down.

5.3 Handle Errors

We need to handle 404 errors for the times we search for a user that doesn't exist.

javascripts/angular/git_http_controller.js.coffee

```
 1  $scope.search = () ->
 2    $scope.repos = []
 3    $scope.searching = true
 4    $scope.errorMessage = ""
 5    url = "https://api.github.com/users/#{$scope.username}/repos"
 6    $http({ method: "GET", url: url })
 7      .success (data) ->
 8        $scope.searching = false
 9        $scope.repos = data
10      .error (data, status) ->
11        $scope.searching = false
12        if status == 404
13          $scope.errorMessage = "User not found"
```

Line 4
Clear `$scope.errorMessage`. Line 10

Handle errors from our http request. We can look at the data from the response along with the status code. Line 13

Set `$scope.errorMessage` to a "User not found" message.

And now we need to display the `errorMessage` on our view.

views/git_http/show.html.erb

```
 1 <div class="alert alert-danger" ng-show="errorMessage">
 2   <strong><i class="icon-warning-sign"></i> Error!</strong>
 3   {{ errorMessage }}
 4 </div>
 5 
 6 <i class="icon-refresh icon-spin" ng-show="searching"></i>
 7 
 8 <ul class="list-group">
 9   <li class="list-group-item" ng-repeat="repo in repos">
10     <h4 class="list-group-item-heading">{{ repo.full_name }}</h4>
11     <p class="list-group-item-text text-muted">{{ repo.url }}</p>
12   </li>
13 </ul>
```

Line 1
: Display the `errorMessage` that we set in our `search` method.

Try it out!

Reload `http://localhost:3000/git_http/show`. This time search for some random user that can't possibly exist. How about `angularailsissuperawesome`?

5.4 Using jQuery

You should really strive to use AngularJS's `$http`, but sometimes you just have to fall back to what you know. Also, this is a great demonstration of the flexibility of AngularJS, and how awesome it is to integrate with third party libraries that operate outside of it. However, don't learn to use jQuery ajax for all your HTTP requests. You have been warned!

javascripts/angular/git_http_controller.js.coffee

```
 1 $scope.search = () ->
 2   $scope.repos = []
 3   $scope.searching = true
 4   $scope.errorMessage = ""
 5   url = "https://api.github.com/users/#{$scope.username}/repos"
 6   $.ajax
 7     type: "GET"
 8     url: url
 9     success: (data) ->
10       $scope.searching = false
11       $scope.repos = data
12       $scope.$apply()
13     error: (error) ->
14       $scope.searching = false
15       if error.status == 404
16         $scope.errorMessage = "User not found"
17       $scope.$apply()
```

Line 6 `$.ajax`

Yes… jQuery in our Angular controller. Keep it secret! Line 12,17 `$scope.$apply()`

Since the jQuery ajax request occurs outside the eyes of Angular, we need to directly inform Angular to update all the bindings.

Try it out!

Reload `http://localhost:3000/git_http/show`. Everything should work the same as before… except underneath the covers we are being very naughty and using jquery.

Conclusion

The building block basics of using AngularJS are being revealed piece by piece. In this chapter we learned how to make HTTP requests using `$http`. We also saw how easily AngularJS can integrate with third party JavaScript libraries that operate outside AngularJS's normal scope.