

# AngularJS

na prática



Daniel Schmitz  
Douglas Lira

# AngularJS na prática (PT-BR)

Crie aplicações web com AngularJS

Daniel Schmitz e Douglas Lira

Esse livro está à venda em <http://leanpub.com/livro-angularJS>

Essa versão foi publicada em 2016-06-24



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2013 - 2016 Daniel Schmitz e Douglas Lira

# Tweet Sobre Esse Livro!

Por favor ajude Daniel Schmitz e Douglas Lira a divulgar esse livro no [Twitter](#)!

O tweet sugerido para esse livro é:

Comprei o livro *AngularJS na prática* do autor Daniel Schmitz. Mais informações em: <https://leanpub.com/livro-angularJS>

A hashtag sugerida para esse livro é [#livroangularjs](#).

Descubra o que as outras pessoas estão falando sobre esse livro clicando nesse link para buscar a hashtag no Twitter:

<https://twitter.com/search?q=#livroangularjs>

# Conteúdo

<b>Introdução</b>	<b>3</b>
O que é AngularJS?	3
Código fonte	4
Errata	4
 <b>Parte 1 - AngularJS</b>	 <b>5</b>
<b>Uma nota sobre a nova versão desta obra</b>	<b>6</b>
<b>Capítulo 1 - Preparando o ambiente</b>	<b>7</b>
Editor de textos	7
Servidor Web	7
Instalando o Apache	8
Windows	8
Linux	8
Instalando o Node	9
Windows	9
Linux	10
Instalando o servidor web	11
Instalando o AngularJS	11
Utilizando CDN	12
Realizando o download da biblioteca manualmente	13
Instalação através do npm	14
Instalando mais bibliotecas	17
Otimizando o seu projeto	18

## CONTEÚDO

<b>Capítulo 2 - Conhecendo AngularJS . . . . .</b>	<b>23</b>
DataBind . . . . .	23
Controller . . . . .	24
Métodos do controller . . . . .	26
Loops . . . . .	27
Formulários . . . . .	28
Rotas e Deep linking . . . . .	29
Como testar este exemplo . . . . .	36
<b>Capítulo 3 - Conectando AngularJS ao servidor . . . . .</b>	<b>37</b>
Uso do \$http . . . . .	38
Exemplo com \$http . . . . .	39
Uso do \$resource . . . . .	42
Exemplo simples com \$resource . . . . .	43

### Uma nota sobre PIRATARIA

Esta obra não é gratuita e não deve ser publicada em sites de domínio público como o *scrib*. Por favor, contribua para que o autor invista cada vez mais em conteúdo de qualidade **na língua portuguesa**, o que é muito escasso. Publicar um ebook sempre foi um risco quanto a pirataria, pois é muito fácil distribuir o arquivo pdf.

Se você obteve esta obra sem comprá-la no site <https://leanpub.com/livro-angularJS>, pedimos que leia o ebook e, se acreditar que o livro mereça, compre-o e ajude o autor a publicar cada vez mais. Obrigado!!

**Atualização em JAN/2016**

Prezados leitores, obrigado por adquirir o livro “AngularJS na prática”. Esta obra abrange todo o conteúdo sobre o **Angular 1** e, como você deve saber, o **Angular 2** está a caminho (na versão beta). Como o Angular 2 é algo totalmente diferente do Angular 1, uma nova obra foi criada para suprir este conteúdo, que é a nossa obra **Angular 2 na prática**.

Você pode acessá-la pelo link <https://leanpub.com/livro-angular2> e se cadastrar para receber informações sobre o seu lançamento, que ocorrerá em FEV/2016.

Como você comprou “Angular 1 na prática” estaremos fornecendo uma promoção exclusiva na semana de lançamento do livro, para que possa também conhecer tudo que o Angular 2 tem a oferecer!

**A migração do Angular 1 para o Angular 2 terá um capítulo especial nesta obra, que será publicada assim que o Angular 2 for oficialmente lançado.**

# Introdução

A linguagem JavaScript vem tornando-se uma das mais usadas em todas as áreas de programação que estão ligadas a Web. É uma linguagem que praticamente nasceu com a Web, criada inicialmente para os navegadores *Netscape*, por volta de 1995.

Mais de 15 anos depois, javascript evolui de forma significativa com a criação de novos frameworks, tornando-se a cada dia mais poderosa e utilizada. Um deles é o jQuery, amplamente conhecido e praticamente obrigatório se você deseja controlar os elementos de uma página HTML, também chamado de DOM.

Nesta “onda”, diversos frameworks de qualidade surgem a cada dia, e dentre eles temos o AngularJS, que será o nosso principal objeto de estudo.

Seguindo a popularidade do Javascript, tornou-se natural que a linguagem que é executada no lado do cliente (navegador) pudesse também ser executada no lado do servidor, gerando a plataforma NODE.JS, hoje em dia mundialmente conhecida e utilizada.

## O que é AngularJS?

Este framework é mantido pelo Google e possui algumas particularidades interessantes, que o fazem um framework muito poderoso.

Uma dessas particularidades é que ele funciona como uma extensão ao documento HTML, adicionando novos parâmetros e interagindo de forma dinâmica com vários elementos. Ou seja, com AngularJS podemos adicionar novos atributos no html para conseguir adicionar funcionalidades extras, sem a necessidade de programar em javascript.

AngularJS é quase uma linguagem declarativa, ou seja, você usa novos parâmetros na linguagem html para alterar o comportamento padrão do html. Estes parâmetros (ou propriedades) são chamados de **diretivas**, na qual iremos conhecer cada uma ao longo desta obra.



Além disso, é fornecido também um conjunto de funcionalidades que tornam o desenvolvimento web muito mais fácil e empolgante, tais como o DataBinding, templates, fácil uso do Ajax, controllers e muito mais. Todas essas funcionalidades serão abordadas ao longo desta obra.

## Código fonte

O código fonte desta obra encontra-se em:

<https://github.com/danielschmitz/livro-angular>

## Errata

Esta é uma obra em constante evolução, e erros podem aparecer. Caso encontre algo errado, por favor envie um email para [danieljfa@gmail.com](mailto:danieljfa@gmail.com).

Todos os erros corrigidos serão atualizados automaticamente, e publicados em futuras versões da obra. Você poderá a qualquer momento obter uma nova versão, visitando o site [leanpub.com](http://leanpub.com)

# Parte 1 - AngularJS

# Uma nota sobre a nova versão desta obra

Esta obra foi completamente refeita para atender melhor os padrões de desenvolvimento web. Decidimos incluir diversas novas funcionalidades, e retirar outras, como forma de melhor a experiência do leitor frente a tecnologia AngularJS.

Todos os leitores que leram a obra anterior podem obter o livro novamente através do site da Leanpub, sem nenhum custo adicional. Nesta nova obra, estamos eliminando a dependência do Apache/PHP/Mysql e adicionando o Node.JS como uma parte integrante ao desenvolvimento de aplicações Web.

# Capítulo 1 - Preparando o ambiente

É preciso muito pouco para começar a aprender AngularJS. Em um nível mais básico, você precisa de um editor de textos e de um navegador web.

Com a evolução do javascript nos últimos anos, outra tecnologia ganhou destaque no desenvolvimento web, que é o Node.js, no qual iremos chamar simplesmente de **node**. Node é uma plataforma para executar javascript no lado do servidor, construída sobre o motor Javascript do Google Chrome.

## Editor de textos

Aqui deixamos livre a sua escolha por um editor de textos ou uma IDE. Lembre-se que todo o nosso desenvolvimento é focado em HTML e JavaScript, ou seja, você não precisará de algo “poderoso” para aprender AngularJS, apenas algo que complemente o código Javascript/HTML já está ótimo.

Nesta obra estamos usando extensivamente o *Sublime Text 2*, inclusive para escrever o próprio livro, então nos sentimos confortáveis em recomendá-lo.

## Servidor Web

O AngularJS é um framework para visualização de dados e informações, ele não possui a funcionalidade de prover dados dinâmicos, ou persistir informações em um banco de dados. Estas características são provenientes de um servidor Web.

Um dos servidores mais comuns nos dias de hoje é o Apache. Geralmente o Apache é acompanhado do PHP, que é um dos interpretadores de código mais usados no desenvolvimento web e do banco de dados MySQL.

Também é possível criar um servidor web utilizando Node, algo que pode ser feito via código javascript, ou então utilizar um framework que contém toda a implementação necessária, como o express. Como o foco desta obra é AngularJS, e consequentemente javascript, iremos abordar o Apache, mas dar uma ênfase maior ao express.

## Instalando o Apache

### Windows

Se você utiliza Windows, poderá instalar o **Wamp Server**, disponível [no site oficial](#). Faça o download da versão mais recente e instale o Wamp na configuração padrão.

Após instalado, você poderá incluir arquivos na seguinte pasta **C:\wamp\www**, e poderá utilizar o Apache, bem como o MySQL e outros utilitários acessando <http://localhost/>.

### Linux

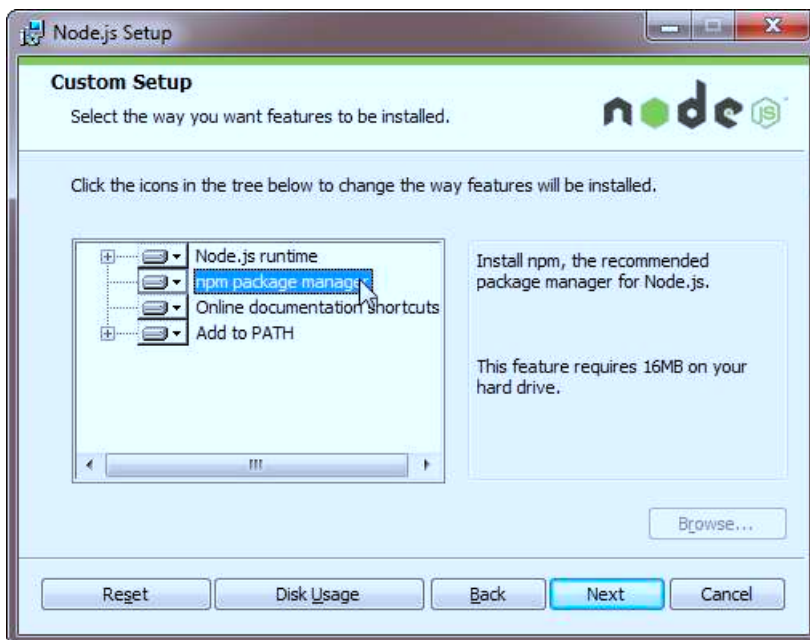
Se usa Linux e uma versão derivada do debian, pode usar o gerenciador de pacotes `apt-get` para instalar tudo para você. O comando a seguir irá instalar o apache, o php e o MySQL.

```
$ sudo apt-get install apache2 apache2-utils
$ sudo apt-get install php5 php5-mysql php-pear php5-mcrypt
$ sudo apt-get install mysql-server
```

# Instalando o Node

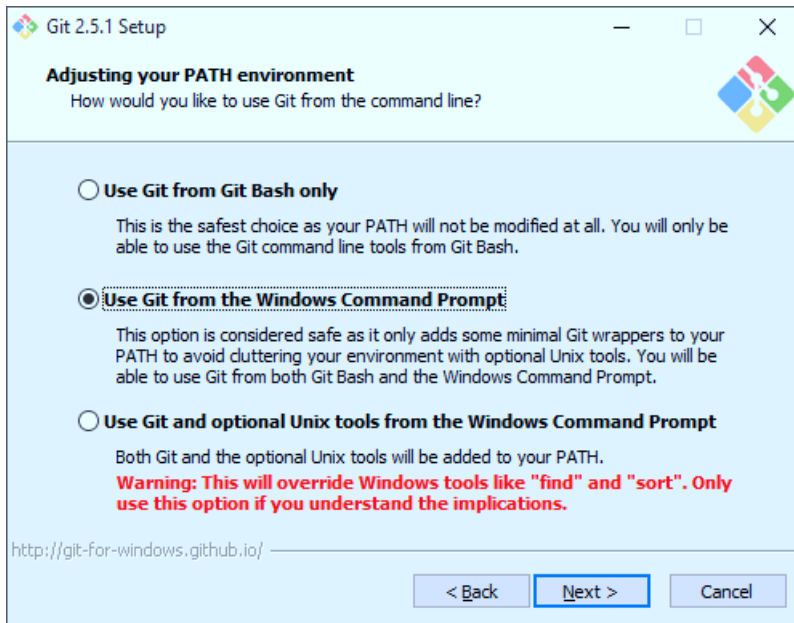
## Windows

Se você utiliza Windows, instale o node através do [site oficial do Node](https://nodejs.org/). Faça o download e instale o Node e deixe selecionado a opção npm, que é o gerenciador de pacotes do node, conforme a imagem a seguir.



Por padrão o npm já vem selecionado, não desmarque esta opção

Além do node, é útil instalar também o GIT, disponível em <https://git-scm.com/>.



Marque esta opção para acessar o git através do cmd do windows

## Linux

Podemos usar o gerenciador de pacotes apt-get do linux para instalar tudo que precisamos, bastando apenas executar o seguinte comando.

```
$ sudo apt-get install git npm
```

Após instalar todos os pacotes necessários, é necessário criar um link simbólico para a palavra node, conforme o comando a seguir.

```
$ sudo ln -s /usr/bin/nodejs /usr/bin/node
```

Com isso o node, npm, git e outros pacotes relacionados estarão devidamente instalados.

## Instalando o servidor web

Precisamos de um servidor web mínimo para que possamos testar nossos exemplo com AngularJS. Com node, podemos facilmente instalar o `http-server`, da seguinte forma:

```
npm install http-server -g
```

Após instalar o `http-server`, navegue até o diretório em que deseja prover acesso e digite:

```
$ http-server ./
```

Isso irá produzir um resultado semelhante ao exibido a seguir:

```
daniel@debian:~/www/temp$ http-server ./
Starting up http-server, serving ./
Available on:
  http:127.0.0.1:8080
Hit CTRL-C to stop the server
```

Agora basta utilizar o endereço fornecido para acessar o diretório criado. Desta forma criamos rapidamente um servidor web simples, sem a necessidade de instalar o Apache e realizar configurações relacionadas a domínio ou virtual host.

## Instalando o AngularJS

Para instalar o AngularJS, considere três opções distintas na qual você pode escolher. Cada uma dela tem suas vantagens e desvantagens. Independente destas três versões, acesse o [site oficial do AngularJS](#) e clique no botão `Download`, fazendo surgir um menu de opções de instalação, conforme a imagem a seguir.





## Utilizando CDN

Inicialmente, você pode adicionar o AngularJS diretamente no seu documento html, sem ter que baixar nenhum pacote ou arquivo. É preciso apenas duas alterações na estrutura de um documento HTML para que possamos ter o AngularJS instalado. A primeira, e mais óbvia, é incluir a biblioteca javascript no cabeçalho do documento. A segunda, e aqui temos uma novidade, é incluir a propriedade **ng-app** no elemento html em que queremos “ativar” o AngularJS. Neste caso, começamos inserindo na tag `<html>` do documento. O exemplo a seguir ilustra este processo.

### HTML default para o AngularJS

```
<html ng-app>
  <head>
    <title>AngularJS</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.7/angular.min.js"></script>
  </head>
  <body>
```

```
</body>  
</html>
```

---

## Atenção

Devido a quebra de linha provocada pela URL da biblioteca javascript, percebe-se uma contra barra \ após a palavra `angular.js`. Esta contra barra deve ser omitida se você copiar/colar o código diretamente do eBook. Ao invés de copiar o código do eBook, [clique neste link para copiar diretamente do site codepen](#).

Na linha 1, temos o uso do **ng-app** que habilita o AngularJS a alterar o comportamento das tags html abaixo dele, permitindo assim que possamos utilizar mais propriedades do AngularJS.

Na linha 4 incluímos a biblioteca AngularJS diretamente, através de uma URL na qual chamamos de CDN.

## Realizando o download da biblioteca manualmente

Pode-se realizar o download da última versão estável do AngularJS, bastando escolher a versão (nesta obra, seria a 1.4.7) e o tipo (Minified, uncompressed, zipped). Escolha a versão zipped e clique em download.

Após o download terminar, descompacte o arquivo e use o seguinte modelo, considerando que o AngularJS descompactado está na pasta 'angular-1.4.7'.

## HTML para o AngularJS

---

```
<html ng-app>
  <head>
    <title>AngularJS</title>
    <script src="angularjs-1.4.7/angular.min.js"></script>
  </head>
  <body>
    </body>
</html>
```

---

## Instalação através do npm

Podemos chamar este capítulo de “mini guia do npm”.

Este será o nosso padrão de instalação de bibliotecas Javascript, e recomendo que você a utilize também. O gerenciador npm (Node Package Manager) é poderoso e fácil de usar, bastando apenas alguns comandos para que possamos instalar qualquer biblioteca.

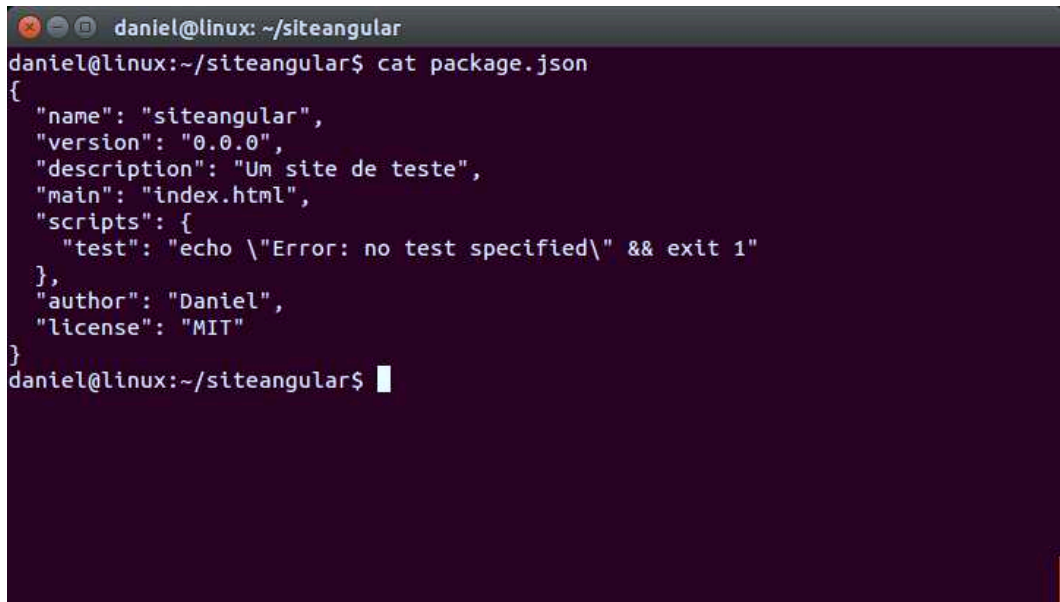
Crie a pasta `C:\siteangular` no Windows, ou `~/siteangular` no Linux e, através da linha de comando do seu sistema, execute a seguinte instrução:

```
cd ~/siteangular
$ npm init
```

ou no Windows:

```
cd c:\siteangular>
npm init
```

O comando `npm init` dentro do diretório que criamos irá criar um arquivo chamado `package.json`. Perceba que algumas perguntas serão feitas, que podem ser preenchidas ou não. Após o `npm init` terminar, verifique o arquivo `package.json`, conforme a figura a seguir :

A terminal window with a dark purple background. The title bar shows a window icon, a maximize icon, and a close icon, followed by the text "daniel@linux: ~/siteangular". The terminal content shows the command "cat package.json" being executed, which displays the following JSON object:

```
{
  "name": "siteangular",
  "version": "0.0.0",
  "description": "Um site de teste",
  "main": "index.html",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Daniel",
  "license": "MIT"
}
```

The prompt "daniel@linux:~/siteangular\$" is visible at the bottom of the terminal.

Agora vamos instalar o AngularJS, através do seguinte comando:

```
$ npm install angular@1.4.7 --save
```

É **muito importante** incluir a diretiva `--save` no comando, isso fará com que o arquivo `package.json` seja atualizado incluindo a biblioteca. A resposta deste comando é semelhante a figura a seguir. Perceba também que incluímos a versão do AngularJS para que os exemplos apresentados nesta obra estejam compatíveis.

```
daniel@linux: ~/siteangular
daniel@linux:~/siteangular$ npm install angular@1.4.7 --save
npm WARN package.json siteangular@0.0.0 No repository field.
npm WARN package.json siteangular@0.0.0 No README data
angular@1.4.7 node_modules/angular
daniel@linux:~/siteangular$ ls node_modules/angular/
angular-csp.css  angular.min.js.gzip  index.js
angular.js       angular.min.js.map   package.json
angular.min.js   bower.json          README.md
daniel@linux:~/siteangular$ cat package.json
{
  "name": "siteangular",
  "version": "0.0.0",
  "description": "Um site de teste",
  "main": "index.html",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Daniel",
  "license": "MIT",
  "dependencies": {
    "angular": "^1.4.7"
  }
}
daniel@linux:~/siteangular$
```

Verifique o arquivo `package.json` após instalar o angular.

```
{
  name: "siteangular",
  version: "0.0.0",
  description: "Um site de teste",
  main: "index.html",
  scripts: {
    test: "echo \"Error: no test specified\" && exit 1"
  },
  author: "Daniel",
  license: "MIT",
  dependencies: {
    angular: "^1.4.7"
  }
}
```

Perceba que a configuração `dependencies` foi adicionada, no final do arquivo. A

chave `angular` foi incluída e a versão `1.4.7` referenciada. Isso é importante para a manutenção do seu projeto, já que podemos atualizar o AngularJS pelo `npm`.

Vamos supor que, ao invés da versão `1.4.7` deseja-se instalar a versão `1.4.5`. Neste caso, mesmo com a biblioteca `1.4.7` instalada podemos usar o comando `npm install angular@1.4.5 --save` que tudo será atualizado para a versão que você informou.

Caso não informe a versão da biblioteca a ser instalada, a última estável será utilizada.

Estamos instalando passo a passo a biblioteca do AngularJS para expor as vantagens do `npm`. Vamos criar agora o arquivo `index.html`, com o seguinte texto:

```
<html ng-app>
  <head>
    <title>AngularJS</title>
    <script src="node_modules/angular/angular.min.js"></script>
  </head>
  <body>
    </body>
</html>
```

Perceba que incluímos o AngularJS no documento HTML, mas não incluímos a versão “1.4.7” do mesmo. Com isso, ao atualizar o AngularJS pelo `npm`, não será preciso alterar o nome do arquivo no documento HTML. Isso é uma grande vantagem, principalmente quando temos diversas bibliotecas nas mais variadas versões e precisamos atualizá-las.

## Instalando mais bibliotecas

Veremos ao longo desta obra a instalação de outras bibliotecas javascript, como por exemplo o `bootstrap`. Mas uma vez vamos usar o `npm` para realizar tal procedimento, como no exemplo do comando a seguir.

```
$ npm install bootstrap --save
```

Após instalar o `bootstrap`, podemos adicioná-lo ao documento `html`, conforme o exemplo a seguir.

```
<html ng-app>
  <head>
    <title>AngularJS</title>
    <link rel="stylesheet" href="node_modules/bootstrap/css/boot\
strap.min.css">
    <script src="node_modules/angularjs/angular.min.js"></script>
    <script src="node_modules/bootstrap/dist/js/bootstrap.min.js\
"></script>
  </head>
  <body>
    </body>
</html>
```

Ou seja, teremos ao longo do desenvolvimento de nossas aplicações a criação e referência de diversos arquivos que deverão ser incluídos no documento HTML. A princípio podemos deixar desta forma a organização do arquivo `index.html`, caso deseje partir para o entendimento do AngularJS, mas certifique-se de voltar neste capítulo quando puder, pois o próximo capítulo é importante para a organização do seu projeto.

## Otimizando o seu projeto

Como vimos no html anterior, temos dois javascripts que serão adicionados ao documento html. Em um projeto real, pode-se chegar a mais de dez arquivos, contando com código javascript e css. Cada arquivo deste é uma requisição que browser faz ao servidor, e em ambiente de produção temos que nos preocupar com esta “carga”. O ideal, neste aspecto, é juntar todos os arquivos javascript em um só, fazendo com que somente uma requisição seja feita para carregar todo o javascript da página.

Então, resta a você, a cada alteração de versão ou a cada inclusão de um novo arquivo js, juntá-lo em um único arquivo. Neste ponto devemos concordar que esta tarefa é muito tediosa, e possivelmente resultará em erros se for realizada manualmente. Neste contexto entram os automatizadores de tarefas, como o grunt ou o gulp. Como o gulp é mais rápido e mais fácil de escrever, vamos fazer uma pequena abordagem

sobre ele, lembrando que o gulp não é o objeto principal de nossa obra, por isso vamos criar algo básico apenas para demonstrar a ferramenta.

O gulp é uma biblioteca gerenciada pelo npm, então podemos instalá-la da seguinte forma:

```
$ sudo npm install gulp -g
```

Perceba que existe um novo parâmetro na instalação do gulp, que é -g, indicando que tal biblioteca deve ser instalada globalmente no sistema, já que usaremos o gulp em qualquer tipo de projeto criado.

Após instalar o gulp globalmente, é preciso dizer ao nosso projeto “siteangular” que usaremos esta biblioteca. Mas antes disso vamos analisar um ponto importante na sua instalação. Perceba que o gulp é uma ferramenta que vai automatizar alguns processos que nós desenvolvedores deveríamos ter feito, como unir vários arquivos javascript em um único. Em nenhum momento o gulp será usado pelo navegador, pelos nossos sistemas e pelos clientes que usamos o nosso sistema. Isso torna o gulp uma biblioteca que deve ser usada somente no desenvolvimento do sistema, e não em produção.

Com isso, precisamos dizer ao projeto que o gulp é usado apenas no desenvolvimento, e isso é feito através do atributo --save-dev que será utilizado substituindo o parâmetro --save. Então, para instalar o gulp em nosso projeto, faremos:

```
$ npm install gulp --save-dev
```

Observe que o arquivo package.json possui a referência do gulp ao modo dev. Além do gulp, também precisamos do gulp-concat pois será ele que irá concatenar os arquivos javascript. Para instalá-lo, faça:

```
$ sudo npm install gulp-concat -g  
$ npm install gulp-concat --save-dev
```

Com todas as bibliotecas prontas, vamos iniciar a rotina para juntar todos os arquivos. Isso é feito através de um arquivo chamado gulpfile.js, que deve estar na raiz do seu projeto.

A princípio, escreva o seguinte código:



/siteangular/gulpfile.js

---

```
var gulp = require('gulp');
var concat = require('gulp-concat');

//Cria a tarefa default
gulp.task('default', function(){

});
```

---

Neste arquivo, criamos duas variáveis, gulp e concat, e então usamos o método task da variável gulp para criar uma tarefa, que inicialmente não faz nada. Perceba que a criação da variável gulp usa o método require, que não é um método do javascript, mas sim do node.

Salve o arquivo e abra o terminal, e então digite o seguinte comando:

```
$ gulp
[22:56:25] Using gulpfile ~/siteangular/gulpfile.js
[22:56:25] Starting 'default'...
[22:56:25] Finished 'default' after 57 µs
```

O comando gulp irá automaticamente executar a tarefa default criada. Nesta tarefa, vamos então concatenar todos os javascripts e gerar um novo arquivo, que será chamado de script.min.js. Altere o arquivo gulpfile.js para o seguinte código.

```
var gulp = require('gulp');
var concat = require('gulp-concat');

var js = [
  './node_modules/angular/angular.min.js',
  './node_modules/bootstrap/dist/js/bootstrap.min.js'
];

//Cria a tarefa default
```

```
gulp.task('default', function(){
  gulp.src(js)
    .pipe(concat('script.min.js'))
    .pipe(gulp.dest('./js/'));
});
```

Agora inserimos alguns comandos na tarefa default, sendo o primeiro deles o `gulp.src`, que indica um array com a fonte de arquivos que será trabalhada. Após o `src`, usamos o método `pipe` para conectar outro comando, o `concat`, que irá juntar tudo aquilo que o `src` obteve. Depois, usamos novamente o `pipe` em conjunto com o comando `gulp.dest` que irá salvar tudo que processado na pasta `js`. Ao executar novamente o `gulp`, temos.

```
$ gulp
[23:34:59] Using gulpfile ~/siteangular/gulpfile.js
[23:34:59] Starting 'default'...
[23:34:59] Finished 'default' after 9.79 ms
```

Após executar, verifique se o arquivo `js\script.min.js` foi criado, e atualize o documento `html` conforme o código a seguir.

```
<html ng-app>
  <head>
    <title>AngularJS</title>
    <script src="js/script.min.js"></script>
  </head>
  <body>
  </body>
</html>
```

O `gulp` possui muito mais funcionalidades, você verá algumas delas ainda nesta obra, mas lembre-se que nosso foco é `AngularJS`.

Lembre que, ao atualizar a versão do `AngularJS` ou de qualquer outra biblioteca pelo `npm`, deve-se executar novamente o `gulp` para que um novo arquivo `script.min.js` seja criado.

Nesta obra, nós iremos concatenar somente as bibliotecas padrão, como angular, bootstrap, jquery entre outras. Os arquivos javascript do projeto não serão concatenados.

# Capítulo 2 - Conhecendo AngularJS

Agora que temos o básico em funcionamento, vamos aprender as principais regras do AngularJS. Através delas será possível realizar diferentes tarefas que irão tornar o desenvolvimento web muito mais simples e prazeroso.

## DataBind

Uma das principais vantagens do AngularJS é o seu DataBind. Este termo é compreendido como uma forma de ligar automaticamente uma variável qualquer a uma outra. Geralmente, o DataBind é usado para ligar uma variável do JavaScript (ou um objeto) a algum elemento do documento HTML.

No exemplo a seguir, estamos usando o AngularJS para ligar uma caixa de texto (o elemento input do html) à um cabeçalho.

```
<html ng-app>
  <head>
    <title>Hello World</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs\
/1.4.3/angular.min.js"></script>
  </head>
  <body>
    Hello <input type="text" ng-model="yourName"/>
    <hr/>
    <h1>Hello {{yourName}}</h1>
  </body>
</html>
```

### [Ver online](#)

Além da propriedade **ng-app** (linha 1), utilizamos para DataBind a propriedade **ng-model**, para informar que este elemento estará ligado a uma variável do AngularJS, através da variável `yourName`, na linha 8. Isso significa que qualquer alteração na caixa de texto irá atualizar o valor da variável.

Na linha 10, temos a chamada à variável através do comando `{{yourName}}`, que imprime o valor da variável. Como o DataBind é dinâmico, ao mesmo tempo que algo é escrito na caixa de texto, o seu referido bind é realizado, atualizando instantaneamente o seu valor.

Bind também pode ser realizado em objetos, mas antes de começar a aumentar a complexidade do código, vamos criar um *controller* para melhorar a organização do código.

## Controller

Um controller é, na maioria das vezes, um arquivo JavaScript que contém funcionalidades pertinentes à alguma parte do documento HTML. Não existe uma regra para o controller, como por exemplo ter um controller por arquivo HTML, mas sim uma forma de sintetizar as regras de negócio (funções javascript) em um lugar separado ao documento HTML.

Inicialmente, vamos criar um exemplo simples apenas para ilustrar como usar o controller.

### Uso do controller

---

```
<html ng-app="app">
  <head>
    <title>Hello World</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/angular.min.js"></script>
    <script src="simpleController.js"></script>
  </head>
  <body ng-controller="simpleController">
    Hello <input type="text" ng-model="user.name"/>
```

```
        <hr/>
        <h1>Hello {{user.name}}</h1>
    </body>
</html>
```

---

### [Ver online](#)

Neste exemplo, após incluir o arquivo `simpleController.js`, usamos a propriedade **ng-controller** (linha 8) para dizer que, todo elemento abaixo do `<body>` será gerenciado pelo controller.

Na linha 9, adicionamos uma caixa de texto utilizando a propriedade **ng-model** e atribuímos o valor `user.name`. Neste caso, `user` é um objeto e `name` é uma propriedade a este objeto. Na linha 11 temos o uso do `bind` no objeto `user`.

#### `simpleController.js`

---

```
var app = angular.module('app', []);

app.controller('simpleController', function ($scope) {
    $scope.user = {name: "Daniel"}
});
```

---

### [Ver online](#)

O arquivo `simpleController.js` contém a criação da `app` e a indicação do controller, que é criado de forma modularizada. O nome do arquivo `JavaScript` não precisa ser o mesmo, e com isso pode-se ter vários controllers em somente um arquivo `JavaScript`, bastando apenas registrá-lo na aplicação.

Perceba que, na linha 1, criamos um módulo do angular na qual chamamos de `'app'`, e depois registramos um controller à este módulo.

Neste controller, temos o parâmetro `$scope` que é um “ponteiro” para a aplicação em si, ou seja, `$scope` significa a própria página `html`. Como o controller foi declarado no elemento `<body>`, `$scope` é usado para todo este elemento. Usa-se o `$scope` para criar uma conexão entre o `model` e a `view`, como foi feito no exemplo utilizando o objeto `user`.

## Métodos do controller

O controller é usado também para manipular regras de negócio que podem ou não alterar os models. No exemplo a seguir, usamos o controller para definir uma variável e um método para incrementar em 1 esta variável.

### Simple exemplo de contador

---

```
<html ng-app="app">
  <head>
    <title>Hello Counter</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs\
/1.4.3/angular.min.js"></script>
    <script src="countController.js"></script>
  </head>
  <body ng-controller="countController">
    <a href="#" ng-click="addOne()">Add 1</a>
    <hr/>
    <p>Counter value: {{counter}}</p>
  </body>
</html>
```

---

### [Ver online](#)

Neste exemplo, usamos a propriedade **ng-click** que irá chamar um método dentro do seu respectivo controller.

### countController.js

---

```
var app = angular.module('app', []);

app.controller('countController', function ($scope) {
    $scope.counter = 0;

    $scope.addOne = function(){
        $scope.counter++;
    }
});
```

---

No controller, criamos a variável `counter` e também o método `addOne`, que manipula a variável, de forma que o seu valor é refletido automaticamente na view (html) através do `dataBind`.

## Loops

Outra característica do AngularJS é utilizar templates para que se possa adicionar conteúdo dinâmico. Um loop é sempre realizado através da propriedade **ng-repeat** e obedece a uma variável que geralmente é um array de dados.

O exemplo a seguir ilustra este processo, utilizando a tag `li` para exibir uma lista qualquer.

### Usando loops

---

```
<html ng-app="app">
  <head>
    <title>Hello Counter</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs\
/1.4.3/angular.min.js"></script>
    <script type="text/javascript">
      var app = angular.module('app', []);
      app.controller('loopController', function ($scope) {
```



```
        $scope.fruits = ['banana', 'apple', 'orange'];
    });
</script>
</head>
<body ng-controller="loopController">
    <ul>
        <li ng-repeat="fruit in fruits">{{fruit}}</li>
    </ul>
</body>
</html>
```

---

[Ver online](#)

Como visto neste exemplo, pode-se adicionar JavaScript no arquivo HTML, mas não recomenda-se esta prática. É uma boa prática de programação utilizar o arquivo HTML apenas como a camada view, e o arquivo JavaScript como a camada de controller, de acordo com os padrões MVC.

## Formulários

Existem diversas características que um formulário contém, tais como validação, mensagens de erro, formato dos campos, entre outros. Neste caso, usamos o AngularJS de diferentes formas, e usamos vários parâmetros **ng** para controlar todo o processo.

O exemplo a seguir exhibe apenas algumas dessas propriedades, para que você possa entender como o processo funciona, mas durante a obra iremos verificar todos os detalhes necessários para construir um formulário por completo.

### Formulário com validação

---

```
<html ng-app>
<head>
  <title>Simple Form</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4\
.3/angular.min.js"></script>
</head>
<body>

  <form name="myForm">
    <span ng-show="myForm.$invalid">
      Found erros in the form!
    </span>
    <input type="text" ng-model="name" name="Name" value="Your N\
ame" required/>
    <button ng-disabled="myForm.$invalid"/>Save</button>
  </form>

</body>
</html>
```

---

#### [Ver online](#)

Neste formulário, usamos mais algumas propriedades, como por exemplo **ng-show** que irá exibir ou não a tag `<span>` contendo a mensagem de erro do formulário, e **ng-disabled** que desativa o botão de submissão do formulário.

O uso do `myForm.$invalid` é um recurso do AngularJS que define se um formulário está inválido ou não. Como usamos uma caixa de texto com a propriedade `required`, se o campo não estiver preenchido, o formulário ficará inválido.

## Rotas e Deep linking

O AngularJS possui um recurso chamado Deep Linking, que consiste em criar rotas na URI do documento HTML para manipular partes do código HTML de forma

independente, podendo assim separar ainda mais as camadas da sua aplicação. No caso mais simples, suponha que exista uma lista de dados que são exibidos em uma tabela, e que ao clicar em um item desta lista, deseja-se exibir um formulário com os dados daquela linha.

No documento HTML criado, existem dois componentes bem definidos pela aplicação. O primeiro é a tabela com as informações, e o segundo, o formulário para edição dos dados.

O uso de DeepLinking usa Ajax para carregar templates de forma dinâmica, então é necessário que todo o exemplo seja testado em um servidor web.

Se organizarmos esta pequena aplicação em arquivos, teremos:

**index.html**

O arquivo principal da aplicação, que contém o código html, o **ng-app**, a inclusão do AngularJS, entre outras propriedades.

**app.js**

Contém todo o código javascript que define as regras de negócio da aplicação.

**list.html**

Contém a tabela que lista os dados.

**form.html**

Contém o formulário de edição e criação de um novo registro.

### index.html

---

```
<html ng-app="app">
<head>
  <title>DeepLinking Example</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/\
angular.js"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/\
angular-route.js"></script>
  <script src="app.js"></script>
</head>
<body>
  <h1>DeepLink Example</h1>
  <div ng-view></div>

</body>
</html>
```

---

Inicialmente criamos o arquivo `index.html`, que contém a chamada aos arquivos javascript da aplicação. Além dos arquivos `JavaScript`, também usamos a propriedade **ng-app**, que já aprendemos a usar em qualquer aplicação que use o framework.

Também adicionamos um segundo arquivo javascript, que é responsável pelo gerenciamento da rota, chamado de `angular-route.js`.

## Atenção

Se você olhar bem a inclusão do arquivo javascript, irá perceber que incluímos o arquivo `angular.js` e não `angular.min.js`.

Isso foi feito para mostrar que ambas os arquivos funcionam, mas o arquivo “minificado” com a extensão `.min.js` é incompreensível para que possamos detectar erros de javascript.

Já o arquivo `angular.js` é compreensível para nossa leitura e os erros de javascript que por ventura cometemos poderão ser analisados com mais eficiência.

Resumindo, use o arquivo `angular.min.js` somente quando o seu sistema está pronto e em produção. Caso contrário, use o arquivo `angular.js`

Este módulo é criado através da definição de um nome para o **ng-app**, ficando desta forma: `ng-app="app"`. Assim, estamos criando um módulo chamado `App` que deve estar definido pela aplicação.

Como podemos ver, o arquivo `index.html` não tem nenhum conteúdo, apenas o cabeçalho e uma `div` que possui a propriedade **ng-view**. Esta propriedade configura o AngularJS para que toda a geração de código seja renderizada dentro desta tag.

Esta definição é realizada no arquivo `app.js`, cuja parte inicial está descrita a seguir.

**app.js**

---

```
var app = angular.module('app', ['ngRoute']);

app.config(['$routeProvider', function($routeProvider){
    $routeProvider.
        when('/', {controller: 'listController', templateUrl: 'list.html'}).
        when('/edit/:name', {controller: 'editController', templateUrl: 'form.ht\
tml'}).
        when('/new', {controller: 'newController', templateUrl: 'form.html'}).
        otherwise({redirectTo: '/'});
}]);

app.run(['$rootScope', function($rootScope){
    $rootScope.fruits = ["banana", "apple", "orange"];
    console.log('app.run');
}]);
```

---

Nesta primeira parte, usamos o método `angular.module` para criar um módulo, cujo o nome é `app`. O segundo parâmetro é a referência ao módulo `ngRoute`, que é usado para criar as rotas (lembre-se que ele deve ser incluído, conforme visto na linha 6 do arquivo `index.html`).

Após criar o módulo e atribuí-lo a variável `app`, usamos o método `config` para configurar o módulo, neste caso estamos configurando uma funcionalidade chamada *Router*, que possui a função de carregar templates e controllers de acordo com uma URI, ou seja, um endereço repassado pelo navegador.

Na linha 5 temos a primeira configuração através do método `when`, que informa ao *Router* que, ao acessar a raiz do endereço web que o arquivo `index.html` está, deve ser carregado o controller `listController` e o template `list.html`.

Tanto o template quanto o controller serão carregados no elemento html que contém a propriedade **ng-view** do `index.html`.

Na linha 6 adicionamos mais uma rota, e agora configuramos que quando a URI for `/edit/:name`, o controller `editController` e o template `form.html` serão carregados. O atributo `:name` será uma variável que poderá ser obtida no controller.

Tanto na linha 6 quanto na linha 7 usamos o mesmo template `form.html` que contém um formulário para edição ou inserção de um registro.

Na linha 8, configuramos a rota padrão da URI, que é ativada quando nenhuma rota configurada é encontrada.

Na linha 11 usamos o método `app.run` para configurar a variável `$scope` da aplicação, em um contexto global ao módulo. Neste método criamos a variável `fruits` que possui um contexto global à aplicação.

Continuando no arquivo `app.js`, temos:

**app.js**

---

```
14 app.controller('listController', function ($scope) {
15     console.log('listController');
16 });
17
18 app.controller('editController', function ($scope, $location, $routePa\
19 rams) {
20     $scope.title = "Editar Fruta";
21     $scope.fruit = $routeParams.name;
22
23     $scope.fruitIndex = $scope.fruits.indexOf($scope.fruit);
24
25     $scope.save = function(){
26         $scope.fruits[$scope.fruitIndex]=$scope.fruit;
27         $location.path('/');
28     }
29 });
30
31 app.controller('newController', function ($scope, $location, $routePar\
32 ams) {
33
34     $scope.title = "Nova Fruta";
35     $scope.fruit = "";
36
37     $scope.save = function(){
38         $scope.fruits.push($scope.fruit);
39         $location.path('/');
40     }
41 });
```

---

Criamos três controllers para a aplicação, sendo que o primeiro, `listController`, ainda não é utilizado, mas pode ser útil em um momento futuro.

Na linha 18 temos o controller `editController` que possui três parâmetros:

- **scope** É o escopo da aplicação que pode ser utilizada no template do controller criado.
- **location** Usada para realizar redirecionamentos entre as rotas
- **routeParams** São os parâmetros repassados pela URI

Na linha 19 preenchamos a variável `$scope.title`, para que o título do formulário mude, lembrando que o formulário é usado tanto para criar um novo registro quando editá-lo.

Na linha 20 pegamos como parâmetro o nome da fruta que foi repassada pela URI. Este valor é pego de acordo com o parâmetro `:name` criado pela rota, na linha 6.

Na linha 22 obtemos o índice do item que está para ser editado. Usamos isso para poder editar o item no método `save` criado logo a seguir.

Na linha 24 temos o método `save` que é usado para “salvar” o registro no array global. Em uma aplicação real estaríamos utilizando `ajax` para que o servidor persistisse o dado. Na linha 26 redirecionamos a aplicação e com isso outro template será carregado.

Na linha 30, criamos o controller `newController`, que é semelhante ao `editController` e possui o método `save` onde um novo registro é inserido no array `fruits`.

Vamos agora analisar o arquivo `list.html` que é um template e carregado diretamente pelo roteamento do módulo (`app.js`, linha 5).

#### list.html

---

```
<h2>Fruits ({{fruits.length}})</h2>
<ul>
    <li ng-repeat="fruit in fruits"><a href="#/edit/{{fruit}}">{{fruit}}\
</a></li>
</ul>
<a href="#/new">New</a>
```

---

O template não necessita informar o seu controller, pois isso já foi feito pelo módulo do AngularJS (`app.js`, linha 5). Como a variável `fruits` possui um escopo global, ela pode ser usada pelo template e na linha 1, exibindo quantos itens existem no array.



Na linha 3 iniciamos a repetição dos elementos que pertencem ao array `fruits` e incluímos na repetição um link para `#/edit/`. Esta é a forma com que o roteamento do AngularJS funciona, iniciando com `#` e repassando a URI logo a seguir. Na linha 5, criamos outro link, para incluir um novo registro. Novamente usamos a URI que será utilizada pelo roteamento do AngularJS.

O último arquivo deste pequeno exemplo é o formulário que irá editar ou inserir um novo registro.

`form.html`

---

```
<h2> {{title}} </h2>
<form name="myForm">
    <input type="text" ng-model="fruit" name="fruit" required>
    <button ng-click="save()" ng-disabled="myForm.$invalid">Save</button>
n>
</form>
<a href="#/">Cancel</a>
```

---

Na linha 1 usamos o `{{title}}` para inserir um título que é criado pelo controller. O formulário possui apenas um campo cujo **ng-model** é `fruit` que será utilizado pelo controller (`app.js`, linhas 25 e 35). Neste formulário também utilizamos **ng-disabled** para que o botão seja ativado somente se houver algum texto digitado na caixa de texto. O botão *save* possui a propriedade **ng-click**, que irá chamar o método `save()` do controller.

## Como testar este exemplo

O exemplo de deeplinking que exibimos deve ser testado através de um servidor web, para que as requisições ajax possam ser realizadas com sucesso. Se você instalou o Apache, basta realizar o download do [código fonte desta obra](#) e copiar a pasta `angularjs-deeplink`. Caso esteja utilizando o pacote `http-server` do node, basta realizar o download do projeto, navegar até a pasta `angularjs-deeplink` e executar o comando `http-server ./`.

# Capítulo 3 - Conectando AngularJS ao servidor

Agora que conhecemos um pouco sobre o AngularJS, podemos entender como funciona a sua comunicação com o servidor. Assim como é feito com jQuery e até com javascript puro, a melhor forma de obter e enviar dados para o servidor é através de Ajax e o formato de dados para se usar nesta comunicação é JSON.

Existem diversas formas de conexão entre cliente (neste caso, AngularJS) e servidor, e nesta obra estaremos utilizando um conceito chamado RESTful, que é uma comunicação HTTP que segue um padrão bastante simples, utilizando cabeçalhos HTTP como POST, GET, PUT, DELETE.

Na forma mais simples de comunicação de dados, onde temos um objeto e as ações de criar, editar, listar e deletar objetos, resumimos o padrão RESTful às seguintes ações:

Método	<a href="http://site.com/produtos">http://site.com/produtos</a>
GET	Listar todos os produtos
POST	Editar uma lista de produtos
PUT	Criar um novo produto na lista de produtos
DELETE	Excluir uma lista de produtos

Método	<a href="http://site.com/produto/1">http://site.com/produto/1</a>
GET	Obter o produto cujo id é 1
POST	Em teoria nao possui funcionalidade
PUT	Edita ou cria um novo produto
DELETE	Exclui um produto cujo o id é 1

As tabelas acima são uma sugestão que pode ser seguido ou não. Mesmo se

simplificarmos os métodos acima, o mínimo que podemos estabelecer é que métodos GET não alterem dados, pois um método GET pode ser facilmente acessado através do navegador.

O AngularJS fornece duas formas distintas de trabalhar com estas conexões. A primeira delas, e mais simples, é através do serviço `$http`, que pode ser injetado em um controller. A segunda forma é através do serviço `$resource` que é uma abstração RESTful, funcionando como um *data source*.

## Uso do \$http

O uso do `$http` não deve ser ignorado, mesmo que o `$resource` seja mais poderoso. Em aplicações simples, ou quando deseja obter dados de uma forma rápida, deve-se utilizar `$http`.

`$http` é uma implementação ajax através do XMLHttpRequest utilizando JSONP. Iremos sempre usar JSON para troca de dados entre cliente e servidor.

A forma mais simples de uso do `$http` está descrito no exemplo a seguir:

`$http` na sua forma mais simples

---

```
$http({method: 'GET', url: '/someUrl'}).success(function(data){  
  
});
```

---

O método `get` pode ser generalizado para:

### `$http.get`

---

```
$http.get('/someUrl').success(function(data){  
  
});
```

---

Assim como existe o `get`, existem os outros também, conforme a lista a seguir:

- `$http.get`
- `$http.head`
- `$http.post`
- `$http.put`
- `$http.delete`
- `$http.jsonp`

Para todos estes métodos, o AngularJS configura automaticamente o cabeçalho da requisição HTTP. Por exemplo, em uma requisição POST os cabeçalhos preenchidos são:

- `Accept: application/json, text/plain, * / *`
- `X-Requested-With: XMLHttpRequest`
- `Content-Type: application/json`

Além dos cabeçalhos, o AngularJS também serializa o objeto JSON que é repassado entre as requisições. Se um objeto é enviado para o servidor, ele é convertido para JSON. Se uma *string* JSON retorna do servidor, ela é convertida em objeto utilizando um parser JSON.

## Exemplo com `$http`

Vamos criar um exemplo utilizando o serviço `$http`, para obter uma lista de dados e preencher uma tabela. Inicialmente, criamos um arquivo simples no servidor que, em teoria, retornaria com informações de uma lista de objetos em JSON. Como ainda não estamos trabalhando com banco de dados, esta lista será criada diretamente no arquivo, em formato json.

**listFruits.html**

---

```
{ "fruits":  
  [  
    {  
      "id": 1,  
      "name": "Apple",  
      "color": "Red"  
    },  
    {  
      "id": 2,  
      "name": "Banana",  
      "color": "Yellow"  
    },  
    {  
      "id": 3,  
      "name": "watermelon",  
      "color": "Green"  
    },  
    {  
      "id": 4,  
      "name": "Orange",  
      "color": "Orange"  
    }  
  ]  
}
```

---

Neste exemplo, estamos inserindo o arquivo `listFruits.html` na pasta `c:\wamp\www\http-example\` e podemos acessá-lo através da url `http://localhost/http-example/listFruits.html`. O próximo passo é criar o arquivo `index.html`, que contém a camada view do exemplo.

### index.html

---

```
<html ng-app="app">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4\
.3/angular.js"></script>
    <script src="app.js"></script>
  </head>
  <body>
    <div ng-controller="appController">
      <button ng-click="getData()">Get Data</button>
      <h2 ng-show="fruits.length>0">Fruits</h2>
      <ul>
        <li ng-repeat="fruit in fruits" >
          {{fruit.id}} - {{fruit.name}} ({{fruit.color}})
        </li>
      </ul>
    </div>
  </body>
</html>
```

---

No arquivo `index.html` não temos nenhuma novidade, pois a regra de negócio está no seu controller, onde realizamos o Ajax.

### app.js

---

```
var app = angular.module('app', []);

app.controller('appController', function ($scope, $http){
  $scope.fruits = Array();

  $scope.getData = function(){
    $http.get("listFruits.html").success(function(data){
      $scope.fruits = data.fruits;
      console.log($scope.fruits);
    }).error(function(data){
```

```
        alert("Error...");  
        console.log(data);  
    });  
}  
});
```

---

No controller da aplicação, criamos o método `getData`, que é executado quando clicamos no botão “GetData” da view (`index.html`). Neste método, usamos a variável `$http` para as requisições Ajax. Repare que ela é repassada pelo parâmetro do controller, após o `$scope`. Neste caso, o AngularJS encarrega-se de injetar o serviço *http* nesta variável.

Na linha 6 temos o método `$http.get` onde estamos realizando uma requisição Ajax acessando o arquivo `listFruits.html`, que contém a resposta Json. Neste método, podemos concatenar outro método chamado `success`, que é executado se a requisição HTTP GET for realizada com sucesso. Neste caso, a resposta do servidor estará armazenada na variável `data`, e poderemos acessar a variável `data.fruits` que contém o array de objetos que serão usados no loop da view.

Na linha 8 temos o uso do `console.log` que pode ser usado em conjunto com o Firebug (Firefox) ou com o Google Chrome, para verificar resultados no console da janela “developer tools”. Pode-se usar o *developer tools* para analisar as chamadas ajax também, geralmente na aba *Network*.

Com este simples exemplo conseguimos mostrar como é fácil realizar uma requisição Ajax para obter dados do servidor. Pode-se usar o serviço `$http` para toda a sua aplicação, mas quando estamos utilizando RESTfull, existe outro serviço que torna o acesso ao servidor mais abstrato, chamado de resource, no qual veremos a seguir.

## Uso do \$resource

Aprendemos a realizar chamadas Ajax através do `$http` e caso haja necessidade, podemos abstrair ainda mais a forma como o AngularJS acessa o servidor.

Neste contexto entra o `$resource` que estabelece um padrão de comunicação RESTfull entre a aplicação e o servidor.

Para que possamos usar esta biblioteca é preciso adicionar o arquivo `angular-resource.js` no documento HTML. Ou seja, além da biblioteca padrão também incluímos a biblioteca `resource`, conforme o exemplo a seguir.

#### Adicionando a biblioteca angular-resource

---

```
<html ng-app>
  <head>
    <title>Lista de compras</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angular\
js/1.4.3/angular.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs\
/1.4.3/angular-resource.js"></script>
  </head>
  <body>
  </body>
</html>
```

---

## Exemplo simples com \$resource

Com a biblioteca devidamente instalada, devemos carregá-la, através de um parâmetro na criação do módulo da aplicação. Para acompanhar este processo, vamos criar um exemplo simples, utilizando operações CRUD com o `$resource`.



**index.html**

---

```
<html ng-app="app">
<head>
  <title>Lista de compras</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/\
angular.min.js"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/\
angular-resource.min.js"></script>
  <script src="app.js"></script>
</head>
<body ng-controller="phoneController">
  <input type="text" ng-model="idPhone" value="1"/>
  <button ng-click="getPhoneById()">GetPhone By Id</button>
  <hr/>
  <button ng-click="getPhones()">GetPhones</button>
  <hr/>
  <button ng-click="savePhone()">Save Phone</button>
  <hr/>
  <button ng-click="deletePhone()">Delete Phone</button>
</body>
</html>
```

---

Este código não apresenta nenhuma novidade. Estamos criando o *module* do AngularJS chamado `app` e criamos um formulário com alguns botões no controller `PhoneController`.

app.js

---

```
var $app = angular.module('app', ['ngResource']);

$app.controller("phoneController", function ($scope, $resource){

    var Phone = $resource("/phones/:phoneId");

    $scope.getPhoneById = function(){
        Phone.get({phoneId:$scope.idPhone}, function(data){
            $scope.phone=data;
        });
    }

    $scope.getPhones = function(){
        Phone.query(function (data){
            $scope.phones = data;
        });
    }

    $scope.savePhone = function(){
        p = new Phone();
        p.number="1111-2222"
        p.save();
    }

    $scope.deletePhone = function(){
        Phone.delete({phoneId:10});
    }

});
```

---

Na definição do `phoneController` adicionamos a variável `$resource` como parâmetro da função, que será injetada pelo AngularJS. Usamos `$resource` para definir a

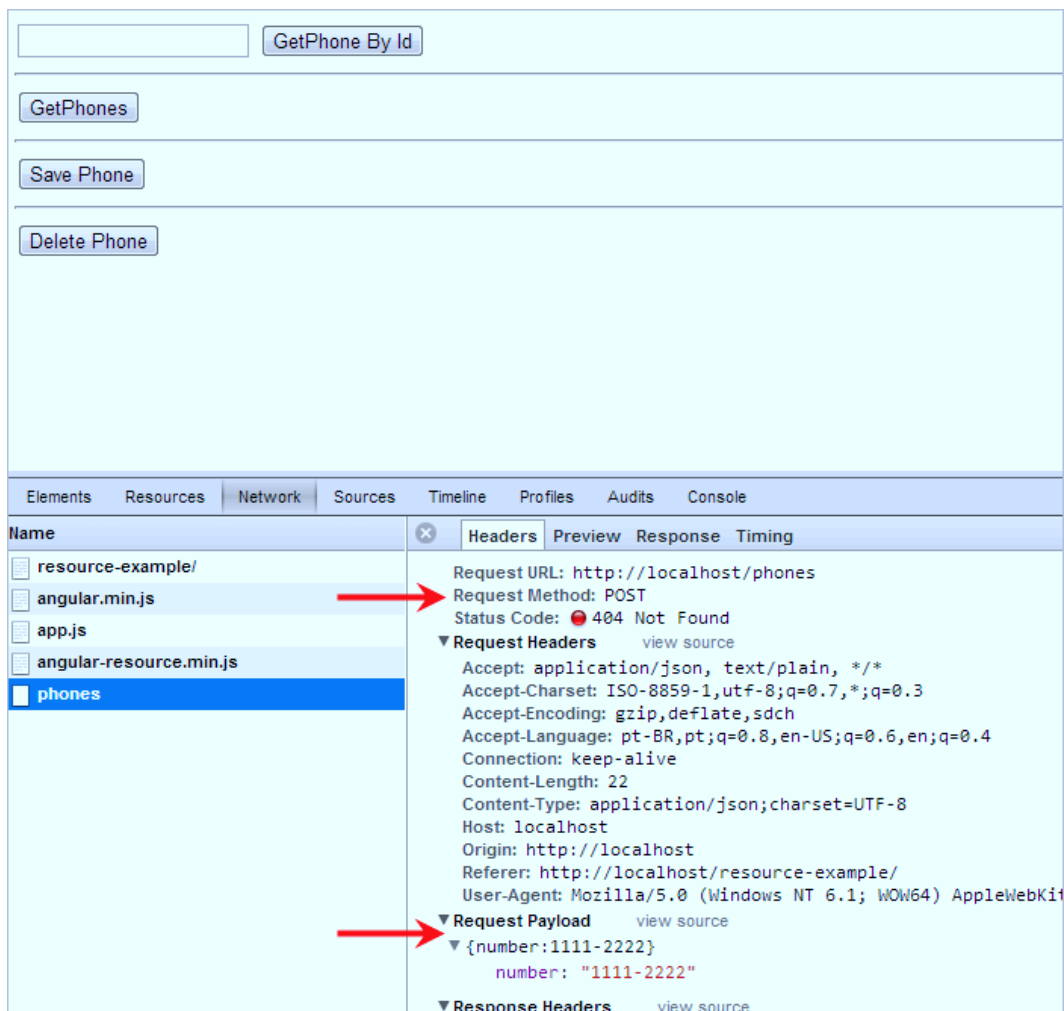
criação da variável `Phone`. Esta variável é criada e configurada como um resource, sendo o primeiro parâmetro a url de acesso ao servidor.

Na linha 7 temos o método `getPhoneById` que é chamado pela view e usa `Phone.get` para realizar uma chamada Ajax ao servidor, conforme a figura a seguir.



Uso do Ajax pelo resources

O mesmo acontece com os outros métodos, e como podemos ver na figura a seguir, quando realizamos um save é realizado um POST no servidor, segundo as especificações do RESTfull.



### Uso do Ajax pelo resources

Nesta figura, temos em detalhe o POST e a variável `number` sendo enviada como json (e não na URL do POST).

Lembre-se que, para testar o resources, você precisa utilizar um servidor web.

Este exemplo está em sua forma mais simples e possivelmente o resource não sera criado em um controller da aplicação.