# ng-book 2

## The Complete Book on AngularJS 2

FULLSTACK.io
gistia

Ari Lerner
Felipe Coury
Nate Murray
Carlos Taborda

# ng-book 2

Felipe Coury, Ari Lerner, Nate Murray, & Carlos Taborda

# Contents

# Book Revision

Revision 9 - Covers up to Angular 2 (2.0.0-alpha.40, 2015-10-15)

# Prerelease

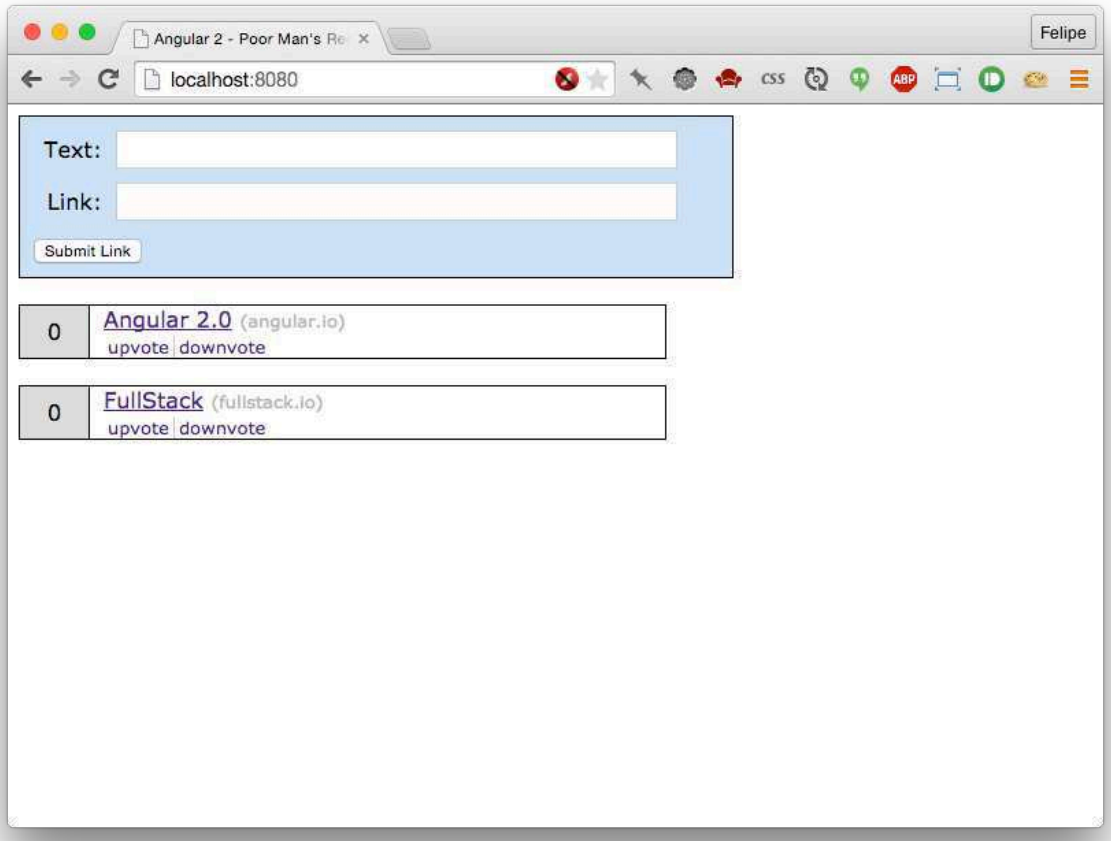
This book is a prerelease version and a work-in-progress.

# Writing your First Angular2 Web Application

## Poor Man's Reddit Clone

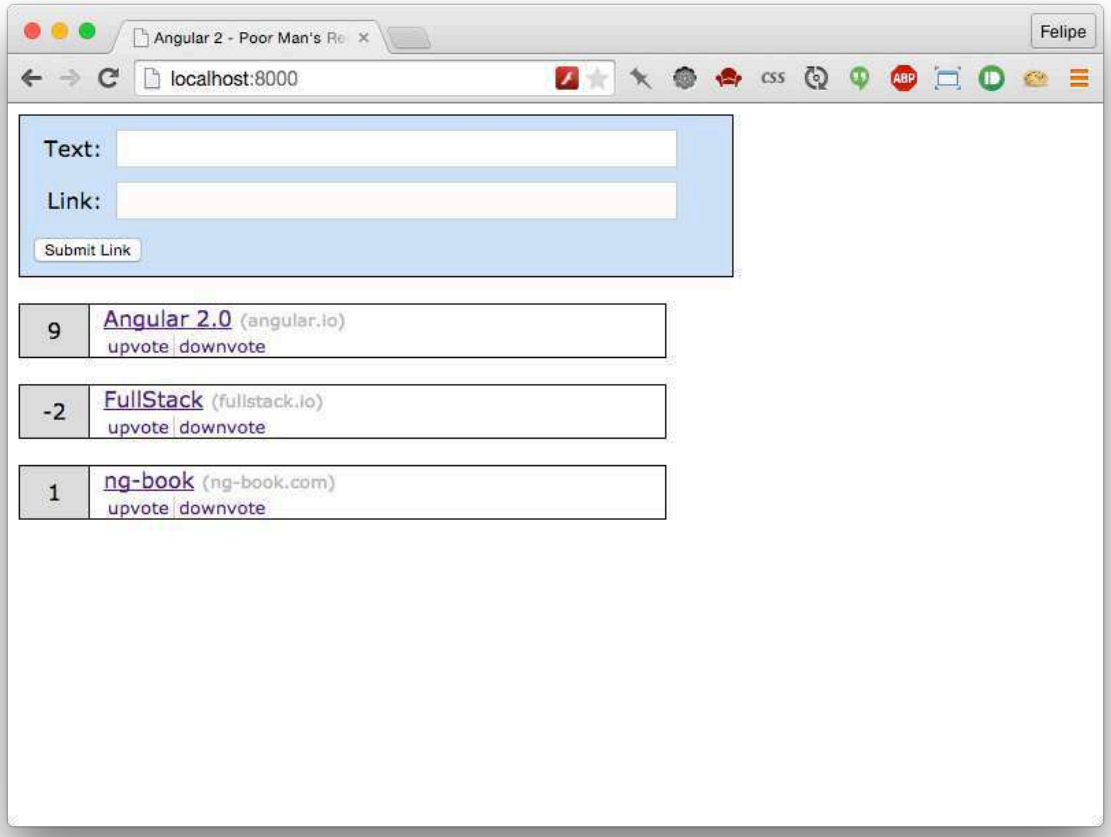
In this chapter we're going to write a toy application that allows the user to submit a new article with a title and an URL. You can think of it like a reddit-clone. By putting together a whole app we're going to touch on most of the parts of Angular 2.

Here's a screenshot of what our app will look like when it's done:




**Completed application**

First, a user will submit a new link and after submitting the users will be able to upvote or downvote each article. Each link will have a score that we will keep track of.




**App with new article**

For this example, we're going to use TypeScript. TypeScript is a superset of JavaScript ES6 that adds types. We're not going to talk about TypeScript in depth in this chapter, but if you're familiar with ES5/ES6 you should be able to follow along without any problems. We'll go over TypeScript more in depth in the next chapter.

# Getting started

## TypeScript

To get started with TypeScript, you will need to have Node.js installed. There are a couple of different ways you can install Node.js, so please refer to the Node.js website for detailed informations:

https://nodejs.org/download/[1] .

**Do I have to use TypeScript?** No, you don't *have* to use TypeScript to use Angular 2, but you probably should. ng2 does have an ES5 API, but Angular 2 is written in TypeScript and generally that's what everyone is going to be using. We're going to use TypeScript in this book because it's great and it makes working with Angular 2 easier. That said, it isn't strictly required.

Once you have Node.js setup, the next step is to install TypeScript. Make sure you install at least version 1.6 or greater. To install it, run the following `npm` command:

```
1 $ npm install -g 'typescript@1.6.2'
```

`npm` is installed as part of Node.js. If you don't have `npm` on your system, make sure you used a Node.js installer that includes it.

Windows Users: We'll be using Linux/Mac-style commands on the commandline throughout this book. We'd highly recommend you install Cygwin[2] as it will let you run commands just as we have them written out in this book.

## Example Project

Now that you have your environment ready, let's start writing our first Angular2 application!

Open up the code download that came with this book and unzip it. In your terminal, `cd` into the `first_app/angular2-reddit-base` directory:

```
1 $ cd first_app/angular2-reddit-base
```

If you're not familiar with `cd`, it stands for "change directory". If you're on a Mac try the following:

1. Open up `/Applications/Utilities/Terminal.app`
2. Type `cd`, without hitting enter
3. In the Finder, Drag the `first_app/angular2-reddit-base` folder on to your terminal window
4. Hit Enter Now you are `cd`ed into the proper directory and you can move on to the next step!

[1] https://nodejs.org/download/
[2] https://www.cygwin.com/

Let's first use npm to install all the dependencies:

```
$ npm install
```

Create a new `index.html` file in the root of the project and add some basic structure:

```
<!doctype html>
<html>
  <head>
    <title>Angular 2 - Poor Man's Reddit</title>
  </head>
  <body>
  </body>
</html>
```

Angular 2 itself is a javascript file. So we need to add a `script` tag to this document to include it. But our setup with Angular/TypeScript has some dependencies itself:

## Traceur

Traceur is a JavaScript compiler that backports ES6 code into ES5 code, so you can use all ES6 features with browsers that only understand ES5.

Even though we're using TypeScript, we're using the Traceur runtime to polyfill ("backport") some ES6 features to our ES5 code

The Angular team is working to remove the Traceur runtime as a dependency of Angular 2

We've vendored Traceur in the sample code, so to add Traceur to our app we add the following script tag:

```
<script src= "vendor/traceur-runtime.js"></script>
```

For more information, check the Traceur GitHub repository[3].

## SystemJS

We also need to add SystemJS. SystemJS is a dynamic module loader. It helps simplify creating modules and requiring our code in our web app. It allows you to require the modules you need in the right order.

We've also vendored SystemJS in the example code so to add SystemJS we need to add this:

[3] https://github.com/google/traceur-compiler

```
    <script src="vendor/system.js"></script>
```

## Angular2

And of course, we also need to add Angular itself. We have a version of Angular 2 in the code sample already, so to use it we can simply put this script tag:

```
    <script src="vendor/angular2.dev.js"></script>
```

## Writing a hello world application

Here's how our code should look now:

```
<!doctype html>
<html>
  <head>
    <title>Angular 2 - Poor Man's Reddit</title>
    <!-- Libraries -->
    <script src="vendor/traceur-runtime.js"></script>
    <script src="vendor/system.js"></script>
    <script src="vendor/angular2.dev.js"></script>
  </head>
  <body>
  </body>
</html>
```

We also need some CSS so our application will look good. Lets include a stylesheet as well:

```
<!doctype html>
<html>
  <head>
    <title>Angular 2 - Poor Man's Reddit</title>
    <!-- Libraries -->
    <script src="vendor/traceur-runtime.js"></script>
    <script src="vendor/system.js"></script>
    <script src="vendor/angular2.dev.js"></script>
    <!-- Stylesheet -->
    <link rel="stylesheet" type="text/css" href="styles.css"> <!-- <- here -->
  </head>
  <body>
  </body>
</html>
```

Let's now create our first TypeScript file. Create a new file called `app.ts` on the same folder and add the following code:

Notice that we suffix our TypeScript file with `.ts` instead of `.js` The problem is, our browser doesn't know how to read TypeScript files, only Javascript files. We'll compile our `.ts` to a `.js` file in just a few minutes.

```
1  /// <reference path="typings/angular2/angular2.d.ts" />
2
3  import {
4    Component,
5    View,
6    bootstrap,
7  } from "angular2/angular2";
8
9  @Component({
10   selector: 'hello-world'
11 })
12 @View({
13   template: `<div>Hello world</div>`
14 })
15 class HelloWorld {
16 }
17
18 bootstrap(HelloWorld);
```

This snippet may seem scary at first, but don't worry. We're going to walk through it step by step.

The `import` statement defines the modules we want to use to write our code. Here we're importing three things: `Component`, `View`, and `bootstrap`. We're `importing` it `from "angular2/angular2"`. The `"angular2/angular2"` portion tells our program where to find the dependencies that we're looking for.

Notice that the structure of this `import` is of the format `import { things } from wherever`. In the `{ things }` part what we are doing is called *destructuring*. Destructuring is a feature provided ES6 and we talk more about it in the next chapter. The idea with the `import` is a lot like `import` in Java or `require` in Ruby. We're just making these dependencies available to this file.

## Making a `Component`

One of the big ideas behind Angular 2 is the idea of *components*. In our Angular apps we write HTML markup that becomes our interactive application. But the browser only knows so many tags. The

built-ins like `<select>` or `<form>` or `<video>` all have functionality defined by our browser creator. But what if we want to teach the browser new tags? What if we wanted to have a `<weather>` tag that defines the weather? Or what if we wanted to have a `<login>` tag that defines where we should put the login?

That is the idea behind components.

If you have a background in Angular 1, **Components are the new version of directives**.

So let's create our very first component. When we have this component written, we will be able to use it in our HTML document like so:

```
<hello-world></hello-world>
```

So how do we actually define a new Component? A basic Component has three parts:

1. A `Component` annotation
2. A `View` annotation
3. A definition class

Let's take these one at a time.

If you've been programming in javascript for a while then this next statement is a little weird to see in javascript:

```
@Component({
// ...
})
```

What is going on here? Well if you have a Java background it may look familiar to you: they are annotations.

Think of annotations as metadata added to your code. When we use `@Component` on the `HelloWorld` class, we are "decorating" the `HelloWorld` as a `Component`.

You might be asking, what's the difference between decorating with an annotation and subclassing? Why use annotations instead of regular class code? How can I write my own annotations? We'll deal with these questions in later chapters.

We want to be able to use this component in our markup by using a `<hello-world>` tag. To do that we configure the `Component` and specify the `selector` as `hello-world`.

```
@Component({
  selector: 'hello-world'
})
```

If you're familiar with CSS selectors, XPath, or JQuery selectors you'll know that there are lots of ways to configure a selector. Angular adds its own special sauce to the selector mix, and we'll cover that later on. For now, just know that in this case we're simply defining a new tag.

The `selector` property here indicates which DOM element this component is going to use. This way if we have any `<hello-world></hello-world>` tag within a template, it will be compiled using this `Component` class.

## Making a `View`

Similar to `@Component`, the `@View` annotation indicates that `HelloWorld` also has a `View`. This `View` defines an HTML `template` that will be rendered when this component is rendered.

```
@View({
  template: `<div>Hello world</div>`
})
```

Notice that we're defining our `template` string between backticks (` ... `). This is a new and fantastic feature of ES6 that allows us to do multiline strings. We could have written the markup above as:

```
@View({
  template: `
  <div>
    Hello world
  </div>
`
})
```

Using backticks for multiline strings is **fantastic** and makes it way easier to put templates inside your code files.

**Should I really be putting templates in my code files?** The answer is, it depends. For a long time the commonly held belief was that you should keep your code and templates separate. While this might be easier for some teams, for some projects it just adds a lot of overhead. When you have to switch between a lot of files it adds overhead to your development. Personally, if my templates are smaller than a page I much prefer having the templates alongside the code. I can see both the logic and the view together and it's really easy to understand how they interact

The biggest drawback to putting your views inlined with your code is that many editors don't yet support syntax highlighting of the internal strings. Hopefully we'll see more editors supporting syntax highlighting HTML within template strings soon.

## Booting Our Application

The last line of our file `bootstrap(HelloWorld);` will start the application. The first argument indicates that the "main" component of our application is `HelloWorld`.

Once it is bootstrapped, the `HelloWorld` component will be rendered where the `<hello-world></hello-world>` snippet is on the `index.html` file. Let's try it out!

### Putting all the pieces together

To run our application, we need to do two things:

1. we need to tell our HTML document to import our `app` file
2. we need to use our `<hello-world>` component

Add the following to the `body` section:

```
<!doctype html>
<html>
  <head>
    <title>Angular 2 - Poor Man's Reddit</title>
    <!-- Libraries -->
    <script src="vendor/traceur-runtime.js"></script>
    <script src="vendor/system.js"></script>
    <script src="vendor/angular2.dev.js"></script>
    <!-- Stylesheet -->
    <link rel="stylesheet" type="text/css" href="styles.css">
  </head>
  <body>
    <script>
      System.import('app.js');
    </script>

    <hello-world></hello-world>

  </body>
</html>
```

We have one problem though: on the new `System.import('app.js')` line, we're using SystemJS to import the module defined in the `app.js` file. But, as you can see, we don't have an `app.js` file yet.

## Compiling the code

Since our application is written in TypeScript, we used a file called `app.ts`. The next step is to compile it to JavaScript, so that the browser can understand it.

In order to do that, let's run the TypeScript compiler command line utility, called `tsc`:

```
$ tsc
$
```

If you get a prompt back with no error messages, it means that the compilation worked and we should now have the `app.js` file sitting in the same directory:

You don't need to specify any arguments to the TypeScript compiler `tsc` in this case because it will look for `.ts` files in the current directory. If you don't get an `app.js` file, first make sure you're in the same directory as your `app.ts` file by using `cd` to change to that directory.

You may also get an error when you run `tsc`. For instance, maybe it says `app.ts(2,1): error TS2304: Cannot find name` or `app.ts(12,1): error TS1068: Unexpected token.`

In this case the compiler is giving you some hints as to where the error is. The section `app.ts(12,1):` is saying that the error is in the file `app.ts` on line `12` character `1`. You can also search online for the error code and often you'll get a helpful explanation on how to fix it.

```
$ ls app.js
app.js
```

We have one more step to test our application. We need to run a test web-server to serve our app from. Let's install a NodeJS static http server called **live-server**. This server is nice because it automatically reloads the page when you make changes to your code.

To install, run the following command:

```
$ npm install -g live-server
```

After it finishes, you can run it with the `live-server` command:

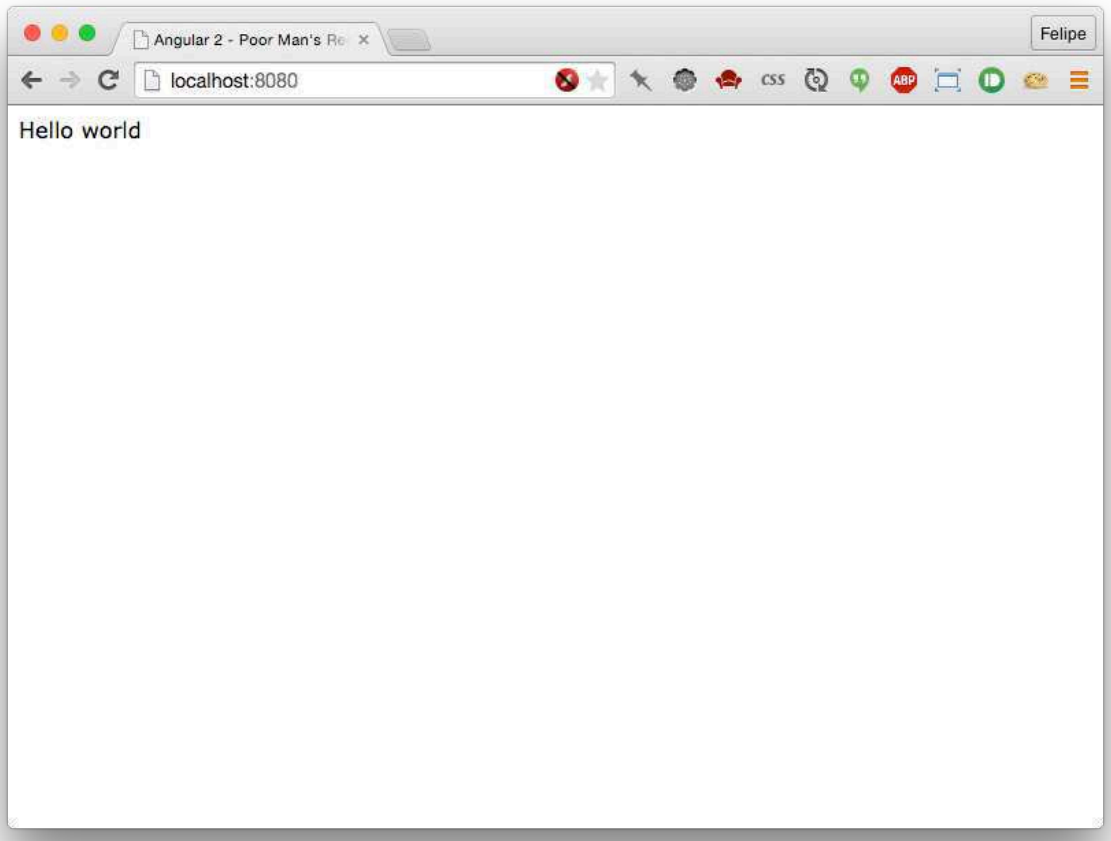
```
$ live-server
Serving "/Users/fcoury/code/angular2-reddit" at http://127.0.0.1:8080
```

**Why do I need a webserver?** If you've developed javascript applications before you probably know that sometimes you can simply open up the `index.html` file by double clicking on it and view it in your browser. This won't work for us because we're using SystemJS.

When you open the `index.html` file directly, your browser is going to use a `file:///` URL. Because of security restrictions, your browser will not allow AJAX requests to happen when using the `file:///` protocol (this is a good thing because otherwise javascript could read any file on your system and do something malicious with it).

So instead we run a local webserver that simply serve whatever is on the filesystem. This is really convenient for testing, but not how you would deploy your production application.

Open your browser and type http://localhost:8080. If everything worked correctly, you should see the following:




**Completed application**

If you're having trouble viewing your application here's a few things to try:

1. Make sure that your `app.js` file was created from the Typescript compiler `tsc`
2. Make sure that your webserver was started in the same directory as your `app.js` file
3. Make sure that your `index.html` file matches our code example above
4. Try opening the page in Chrome, right click, and pick "Inspect Element". Then click the "Console" tab and check for any errors.
5. If all else fails, join us here to chat on Gitter![4]

## Compiling on every change

We will be making a lot of changes to our application code. Instead of having to run `tsc` everytime we make a change, we can take advantage of the `--watch` option. The `--watch` option will tell `tsc` to stay running and watch for any changes to our TypeScript files and automatically recompile to JavaScript on every change:

```
$ tsc --watch
message TS6042: Compilation complete. Watching for file changes.
```

### Adding data to a component

Our component right now isn't very interesting. Most components will have data that make the component dynamic.

Let's introduce `name` as a new property of our component. This way we can reuse the same component for different inputs.

Make the following changes:

```
@Component({
  selector: 'hello-world'
})
@View({
  template: `<article>Hello {{ name }}</article>`
})
class HelloWorld {
  name: string;

```

[4] https://gitter.im/ng-book/ng-book

```
10    constructor() {
11      this.name = 'Felipe';
12    }
13  }
```

Here we changed three things:

**1. `name` Property** On the `HelloWorld` class we added a *property*. Notice that the syntax is new relative to ES5 javascript. We say `name: string;`. That means `name` is the name of the attribute we want to set and `string` is the type.

The typing is provided by TypeScript! This sets up a `name` property on *instances* of our `HelloWorld` class

**2. A Constructor** On the `HelloWorld` class we define a *constructor*, i.e. function that is called when we create new instances of this class.

We use our `name` property by using `this.name`

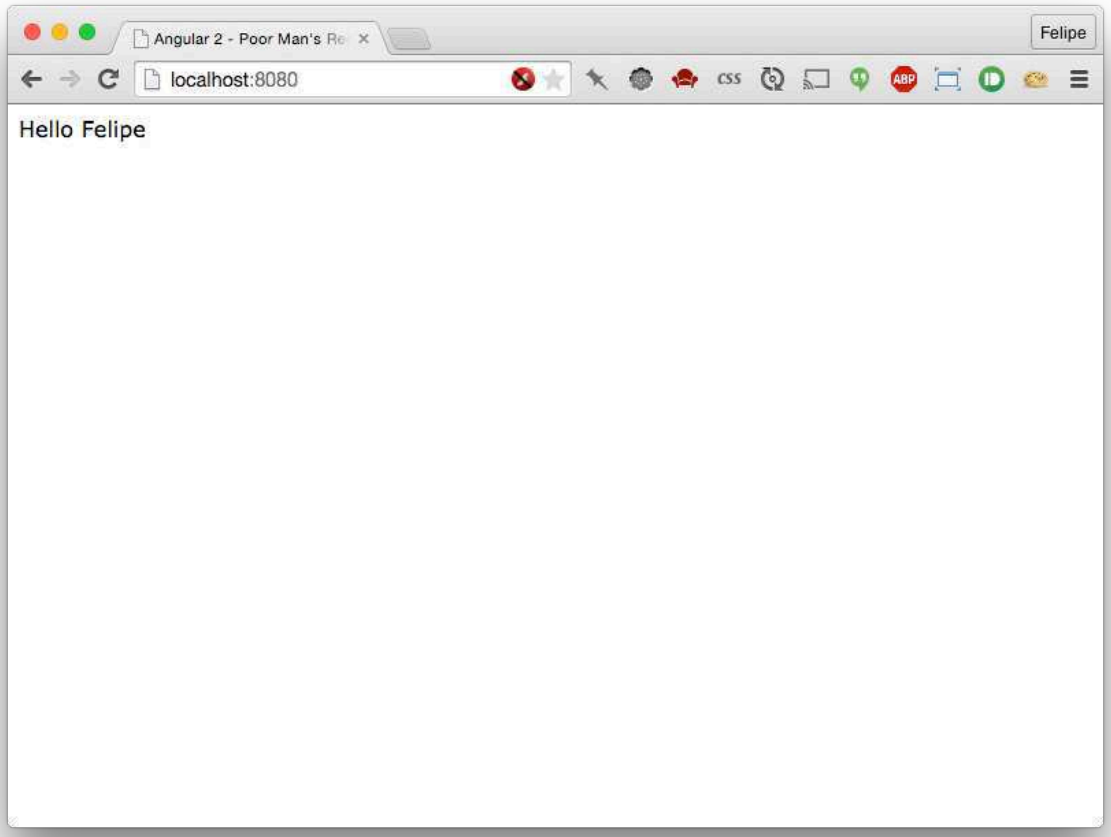
When we write:

```
1   constructor() {
2     this.name = 'Felipe';
3   }
```

We're saying that whenever a new `HelloWorld` is created, set the name to `'Felipe'`.

**3. Template Variable** On the `View` notice that we added a new syntax: `{{ name }}`. The brackets are called "template-tags" (or "mustache tags"). Whatever is between the template tags will be expanded as a template. Here, because the View is *bound* to our Component, the `name` will expand to `this.name` e.g. `'Felipe'` in this case.

**Try it out** After making these changes, reload the page. We should see `"Hello Felipe"`




**Application with Data**

At this point, you might be wondering what indicates that the `View` is bound to this component declaration. The way annotation works is they affect the declaration following the annotation itself.

So when we have code like:

**code/first_app/angular2-reddit-completed/app.ts**

```
@Component({
  selector: 'reddit-article',
  properties: ['article']


})
@View({
  template: `
  <article>
    <div class="votes">{{ article.votes }}</div>
```

```
11      <div class="main">
12        <h2>
13          <a href="{{ article.link }}">{{ article.title }}</a>
14          <span>({{ article.domain() }})</span>
15        </h2>
16        <ul>
17          <li><a href (click)="voteUp()">upvote</a></li>
18          <li><a href (click)="voteDown()">downvote</a></li>
19        </ul>
20      </div>
21    </article>
22    `
23  })
24  class RedditArticle {
```

We are *annotating* the `RedditArticle` class with the preceding annotations `Component` and `View`.

## Working with arrays

Now we are able to say "Hello" to a single name, but what if we want to say "Hello" to a collection of names?

If you've worked with Angular 1 before, you probably used `ng-repeat` directive. In Angular 2, the analogous directive is called `NgFor`. Its syntax is slightly different but they have the same purpose: repeat the same markup for a collection of objects.

Let's make the following changes our to `app.ts` code:

```
1   /// <reference path="typings/angular2/angular2.d.ts" />
2
3   import {
4     Component,
5     NgFor,
6     View,
7     bootstrap,
8   } from "angular2/angular2";
9
10  @Component({
11    selector: 'hello-world'
12  })
13  @View({
14    directives: [NgFor],
```

```
15    template: `
16    <ul>
17      <li *ng-for="#name of names">Hello {{ name }}</li>
18    </ul>
19    `
20  })
21  class HelloWorld {
22    names: Array<string>;
23
24    constructor() {
25      this.names = ['Ari', 'Carlos', 'Felipe', 'Nate'];
26    }
27  }
28
29  bootstrap(HelloWorld);
```

The first change to point out is the new `Array<string>` property on our `HelloWorld` class.

We changed our class to set `this.names` value to `['Ari', 'Carlos', 'Felipe', 'Nate']`.

We added a new property for our `@View` annotation: `directives: [NgFor]`. Unlike Angular 1 where every directive was available in a global namespace, Angular 2 requires that you explicitly state which directives you want to use. This makes the view *require* the `NgFor` directive.

The next thing we changed was our template. We now have one `ul` and one `li` with a new `*ng-for="#name of names"` attribute. The `*` and `#` characters can be a little overwhelming at first, so let's break it down:

The `*ng-for` syntax says we want to use the `NgFor` directive on this attribute.

The value states: `"#name of names"`. `names` is our array of names as specified on the `HelloWorld` object. `#name` is called a *reference*. When we say `"#name of names"` we're saying loop over each element in `names` and assign each one to a variable called `name`.

The `NgFor` directive will render one `li` tag for each entry found on the `names` array, declare a local variable `name` to hold the current item being iterated. This new variable will then be replaced inside the `Hello {{ name }}` snippet.

We didn't have to call the reference variable `name`. We could just as well have written:

```
1   <li *ng-for="#foobar of names">Hello {{ foobar }}</li>
```

But what about the reverse? Quiz question: what would have happened if we wrote:

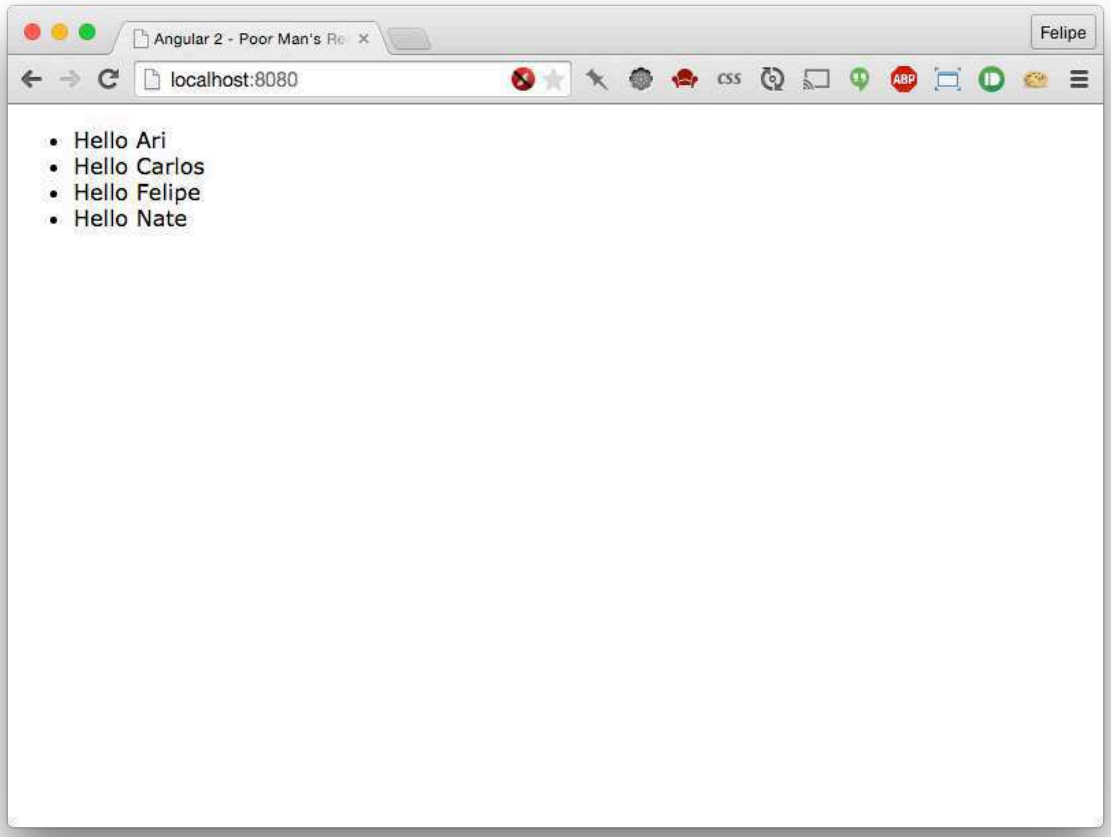
```
1   <li *ng-for="#name of foobar">Hello {{ name }}</li>
```

We'd get an error because `foobar` isn't a property on the component. You can think of this directive like a `for each` loop.

If you're feeling adventurous you can learn a lot about how the Angular core team writes Components by reading the source directly. For instance, you can find the source of the ng-for directive here[5]

When you reload the page now, you can see that we have one `li` for each string on the array:

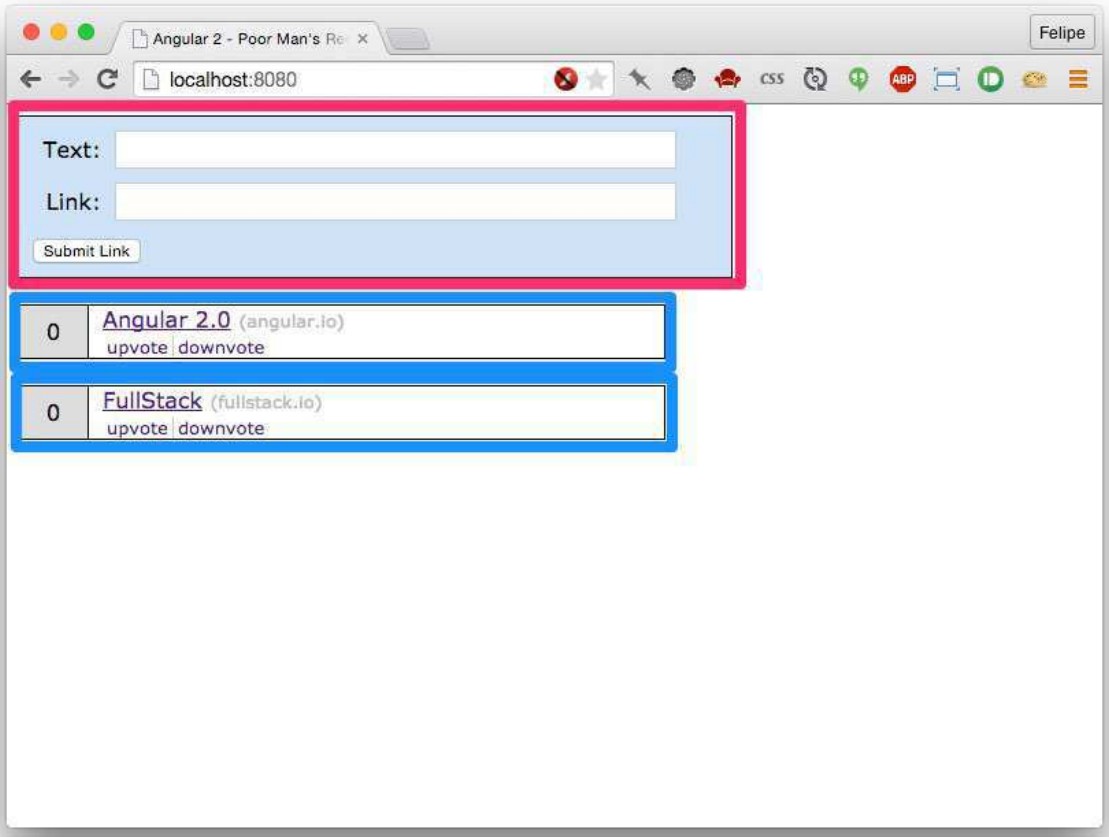
[5] https://github.com/angular/angular/blob/master/modules/angular2/src/core/directives/ng_for.ts




**Application with Data**

## Expanding our Application

Now that we know how to create a basic component, let's revisit our Reddit clone. Before we start coding, it's a good idea to look over our app and break it down into its logical components.




**Application with Data**

We're going to make two components in this app:

1. The form used to submit new articles would be one component (marked in red in the picture)
2. Each article would be another component (marked in blue).

## The form component

Let's start building the form component. For that, we'll change our `app.ts`. We're done with our `HelloWorld` component for now and instead we're going to build a component to represent our whole app: a `RedditApp` component:

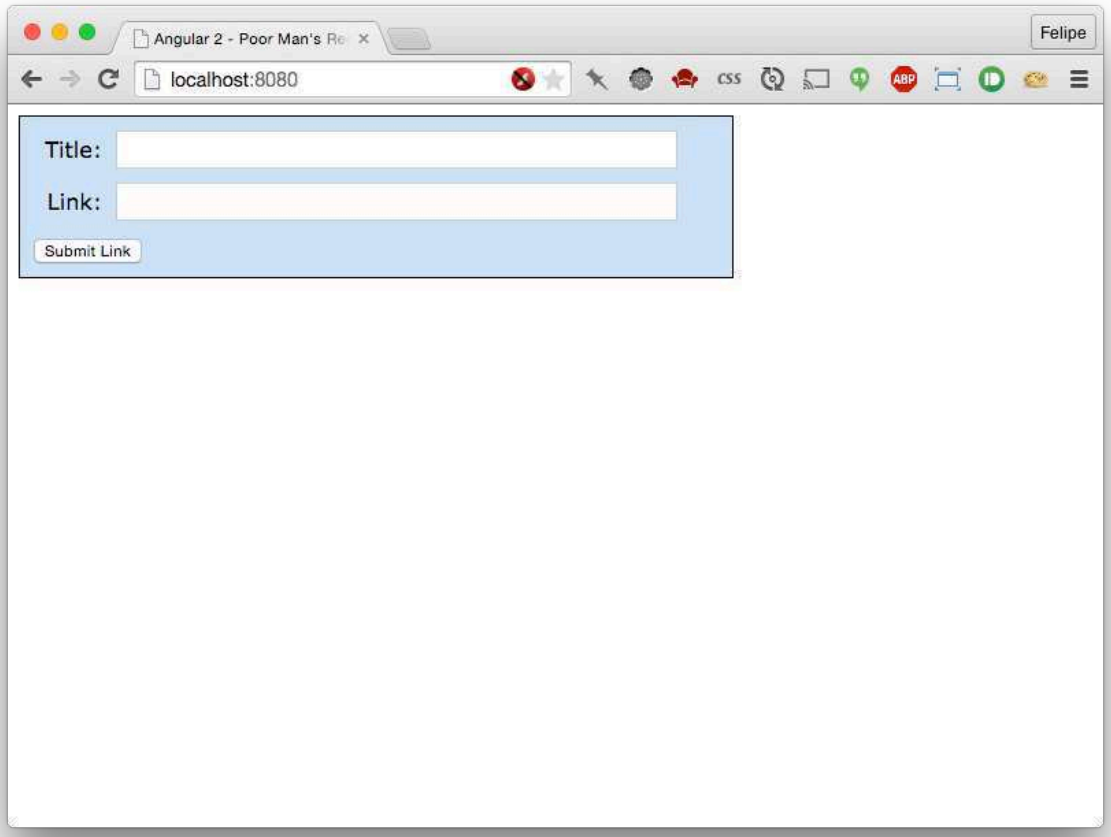
```
/// <reference path="typings/angular2/angular2.d.ts" />

import {
  Component,
  NgFor,
  View,
  bootstrap,
} from "angular2/angular2";

@Component({
  selector: 'reddit'
})
@View({
  template: `
    <section class="new-link">
      <div class="control-group">
        <div><label for="title">Title:</label></div>
        <div><input name="title"></div>
      </div>
      <div class="control-group">
        <div><label for="link">Link:</label></div>
        <div><input name="link"></div>
      </div>

      <button>Submit Link</button>
    </section>
  `
})
class RedditApp {
}

bootstrap(RedditApp);
```

Here we are declaring a `RedditApp` component. Our `selector` is `reddit` which means we can place it on our page by using `<reddit></reddit>`.

We're creating a `View` that defines two `inputs`: one for the `title` of the article and the other for the `link` URL.

After updating your `app.ts`, use our new component by changing your `index.html` and replacing the `<hello-world></hello-world>` tag with `<reddit></reddit>`.

When you reload the browser you should see the form rendered:




**Form**

## Adding Interaction

Now, if you click the submit button, nothing will happen because we haven't added any behavior to our application.

Let's add some interaction:

```
@Component({
  selector: 'reddit'
})
@View({
  template: `
    <section class="new-link">
      <div class="control-group">
        <div><label for="title">Title:</label></div>
        <div><input name="title" #newtitle></div>
      </div>
      <div class="control-group">
        <div><label for="link">Link:</label></div>
        <div><input name="link" #newlink></div>
      </div>

      <button (click)="addArticle(newtitle, newlink)">Submit Link</button>
    </section>
  `,

  directives: [NgFor]
})
class RedditApp {
  addArticle(title, link) {
    console.log("Adding article with title", title.value, "and link", link.value\
);
  }
}
```

To add interaction to our application we need to do two or three things:

1. Create a method in our Component class (`addArticle` in this case) that contains the logic of our action
2. Bind the action to an event in our view (here on our `button` tag)
3. Bind values in inputs to variables that can be used in our action

Let's cover each one of these steps in reverse order:

## Binding `input`s to values

Notice in our first input tag we have the following:

```
<input name="title" #newtitle>
```

The new syntax here is the *reference* to `#newtitle`. This markup tells angular to bind this input to the variable `newtitle`. The effect is that this makes the variable `newtitle` available to the actions within this view. `newtitle` is now an object that represents this `input` tag and we can get the value of the input tag using `newtitle.value`.

Similarly we add `#newlink` to the other input tag, so that we'll be able to extract the value from it as well.

## Binding actions to events

On our `button` tag we add the attribute `(click)` to define what should happen when the button is clicked on. When the `(click)` event happens we call `addArticle` with two arguments: `newtitle` and `newlink`. Where did these things come from?

1. `addArticle` is a function on our Component definition class `RedditApp`
2. `newtitle` comes from the resolve (`#newtitle`) on our `input` for `title`
3. `newlink` comes from the resolve (`#newlink`) on our `input` for `link`

All together:

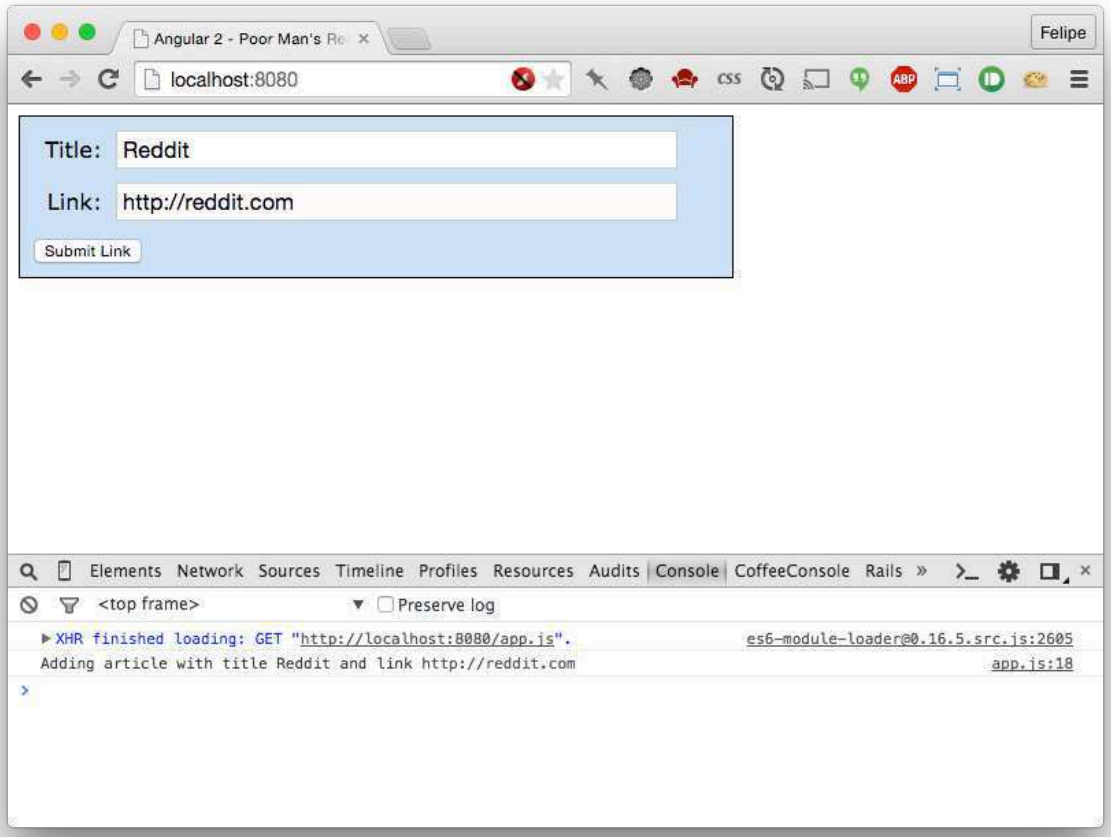
```
<button (click)="addArticle(newtitle, newlink)">Submit Link</button>
```

## Defining the Action Logic

On our `class RedditApp` we define a new function called `addArticle`. It takes two arguments: `title` and `link`. For now, we're just going to `console.log` out those arguments.

## Try it out!

Now when you click the submit button, you can see that the message is printed on the console:




**Clicking the Button**

## Adding the article component

Now we have a form to submit new articles, but we aren't showing the new articles anywhere. Because every article submitted is going to be displayed as a list on the page, this is the perfect candidate for a new component.

Let's create a new component to represent the submitted articles.

For that, we can create a new component on the same file. Insert the following snippet **above** the declaration of the `RedditApp` component:

```
@Component({
  selector: 'reddit-article'
})
@View({
  template: `
  <article>
    <div class="votes">{{ votes }}</div>
    <div class="main">
      <h2>
        <a href="{{ link }}">{{ title }}</a>
      </h2>
      <ul>
        <li><a href (click)='voteUp()'>upvote</a></li>
        <li><a href (click)='voteDown()'>downvote</a></li>
      </ul>
    </div>
  </article>
  `
})
class RedditArticle {
  votes: number;
  title: string;
  link: string;

  constructor() {
    this.votes = 10;
    this.title = 'Angular 2';
    this.link = 'http://angular.io';
  }

  voteUp() {
    this.votes += 1;
  }

  voteDown() {
    this.votes -= 1;
  }
}
```

Notice that we have three parts to defining this new component:

1. Describing the Component properties by annotating the class with `@Component`

2. Describing the Component view by annotating the class with `@View`
3. Creating a component-definition class (e.g. `RedditArticle`) which houses our component logic

**Creating the `reddit-article` Component** First, we define a new Component with `@Component`. We're saying that this component is placed by using the tag `<reddit-article>` (i.e. the selector is a tag name).

**Creating the `reddit-article` View** Second, we define the view with `@View`. There are three ideas worth noting here:

1. We're showing `votes` and the `title` with the template expansion strings `{{ votes }}` and `{{ title }}`. The values come from the value of `votes` and `title` property of the `RedditArticle` class.
2. We can also use template strings in property values like we do in the a `href="{{ link }}"`. The value of the `href` will be dynamically populated with the value of `link` from the component class
3. On our upvote/downvote links we have an action. We use `(click)` to bind `voteUp()`/`voteDown()` to their respective buttons. When the upvote button is pressed, the `voteUp()` function will be called on the `RedditArticle` class (and downvote respectively).

**Creating the `reddit-article` RedditArticle Definition Class** Here we create three properties for each `RedditArticle`:

1. `votes` - a `number` representing the sum of all upvotes, minus the sum of the downvotes
2. `title` - a `string` holding the title of the article
3. `link` - a `string` holding the URL of the article

In the `constructor()` we set some default attributes:

```
1   constructor() {
2     this.votes = 10;
3     this.title = 'Angular 2';
4     this.link = 'http://angular.io';
5   }
```

And we define two functions for voting, one for voting up and one for voting down.

```
  voteUp() {
    this.votes += 1;
  }

  voteDown() {
    this.votes -= 1;
  }
```

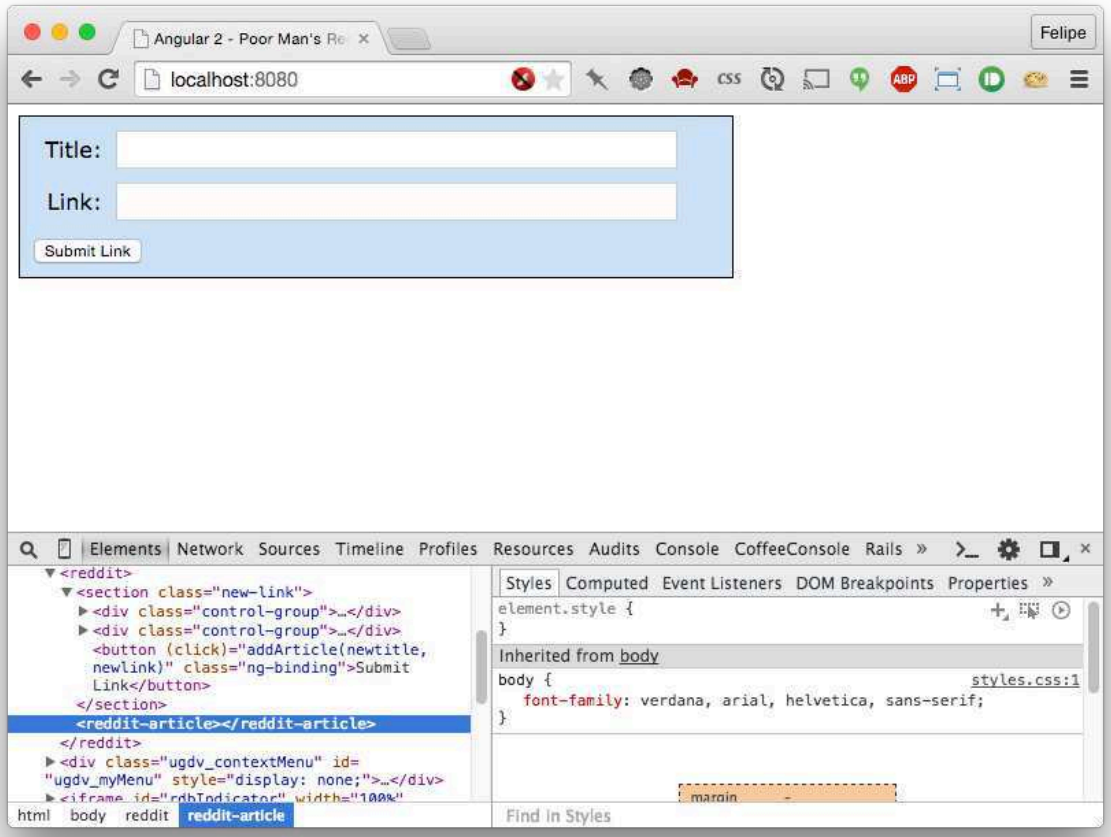
In `voteUp` we increment `this.votes` by one. Similarly we decrement for `voteDown`.

**Using the `reddit-article` Component** In order to use this component and make the data visible, we have to add a `<reddit-article></reddit-article>` tag somewhere in our markup.

In this case, we want the `RedditApp` component to render this new component, so let's change that component code. First we will add the `<reddit-article>` tag to the `RedditApp`'s template:

```
template: `
  <section class="new-link">
    <div class="control-group">
      <div><label for="title">Title:</label></div>
      <div><input name="title" #newtitle></div>
    </div>
    <div class="control-group">
      <div><label for="link">Link:</label></div>
      <div><input name="link" #newlink></div>
    </div>

    <button (click)="addArticle(newtitle, newlink)">Submit Link</button>
  </section>


  <reddit-article></reddit-article>
`
```

If we try to reload the browser now, you will see that the `<reddit-article>` tag wasn't compiled, as can be seen inspecting the DOM here:




**Unexpanded tag when inspecting the DOM**

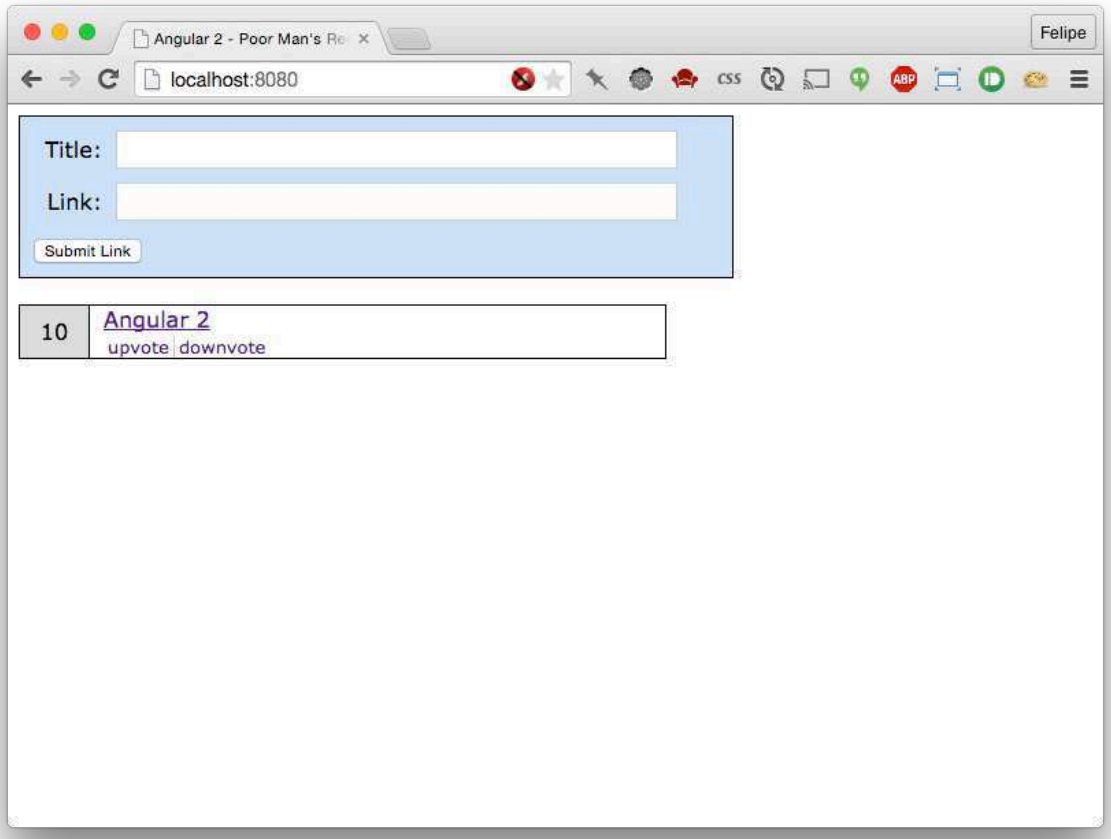
That happens because the `RedditApp` component doesn't know about the `RedditArticle` component yet!

To fix that, we just have to declare the `RedditArticle` component on the `directive` property of the `RedditApp`'s view annotation, just like we did when we used `For` before:

```
1  template: `
2    <section class="new-link">
3      <div class="control-group">
4        <div><label for="title">Title:</label></div>
5        <div><input name="title" #newtitle></div>
6      </div>
7      <div class="control-group">
8        <div><label for="link">Link:</label></div>
9        <div><input name="link" #newlink></div>
10     </div>
```

```
11
12      <button (click)="addArticle(newtitle, newlink)">Submit Link</button>
13    </section>
14
15    <reddit-article></reddit-article>
16  `,
17
18  directives: [RedditArticle]
```

And now, when we reload the browser we should see the article properly rendered:




**Rendered RedditArticle component**

However, if you try to click the **vote up** or **vote down** links, you'll see that the page unexpectedly reloads.

This is because Javascript, by default, **propagates the `click` event to all the parent components**. Because the `click` event is propagated to parents, our browser is trying to follow the empty link.

To fix that, we just need to make the click event handler to return `false`. This will ensure the browser won't try to refresh the page. We change our code like so:

```
1  voteUp() {
2    this.votes += 1;
3    return false;
4  }
5
6  voteDown() {
7    this.votes -= 1;
8    return false;
9  }
```

Now if you click the links, you'll see that the votes increase and decrease properly without a page refresh.

# Rendering multiple components

## Creating an Article class

Right now we only have one article in the page and there's no way to render more, unless we create add a new `<reddit-article>` tag. And even if we do, all the articles would have the same content, so it wouldn't be very interesting.

A good practice when writing Angular2 code is to try to isolate the data structures you are using from the component code. In order to accomplish that, and before making any further changes to the component, let's create a data structure that would represent one article. Add the following code before the `RedditArticle` component code:

```
1  class Article {
2    title: string;
3    link: string;
4    votes: number;
5
6    constructor(title, link) {
7      this.title = title;
8      this.link = link;
9      this.votes = 0;
10   }
11 }
```

Here we are creating a new class that represents an `Article`. Note that this is a plain class and not a component. In the Model-View-Controller pattern this would be the **Model**.

Each article has a `title`, a `link` and a total for the `votes`. When creating a new article we need the `title` and the `link`, and we also assume the recently submitted article has zero votes.

Now let's change the `RedditArticle` code to use our new `Article` class. Instead of storing the properties directly on the `RedditArticle` component, instead we're storing the properties on the `Article` class and simply storing the array of `Articles` in our component:

```
@Component({
  selector: 'reddit-article'
})
@View({
  template: `
      <article>
        <div class="votes">{{ article.votes }}</div>
        <div class="main">
          <h2>
            <a href="{{ article.link }}">{{ article.title }}</a>
          </h2>
          <ul>
            <li><a href (click)='voteUp()'>upvote</a></li>
            <li><a href (click)='voteDown()'>downvote</a></li>
          </ul>
        </div>
      </article>
      `
})
class RedditArticle {
  article: Article;

  constructor() {
    this.article = new Article('Angular 2', 'http://angular.io');
  }

  voteUp() {
    this.article.votes += 1;
    return false;
  }

  voteDown() {
    this.article.votes -= 1;
    return false;
```

```
35    }
36  }
```

Notice that we substitute all of the properties with the `article` instead. We can also use our new class `Article` as a type for the `article` property!

If you reload the browser, you're going to see everything works the same way. That's good but something in our code is still a little off: our `voteUp` and `voteDown` methods break the encapsulation of the `Article` class by changing their internal properties directly.

`voteUp` and `voteDown` current break the Law of Demeter[6] which says that a given object should assume as little as possible about the structure or properties any other objects. One way to detect this is to be suspicious when you see long method/property chains like `foo.bar.baz.bam`. This pattern of long-method chaining is also affectionately referred to as a "train-wreck".

```
1  voteUp() {
2    this.article.votes += 1;
3    return false;
4  }
5
6  voteDown() {
7    this.article.votes -= 1;
8    return false;
9  }
```

The problem is that our `RedditArticle` component knows too much about the `Article` class internals. To fix that, let's also move the methods to the `Article` class, changing `RedditArticle` to cope with the change:

```
1  class Article {
2    title: string;
3    link: string;
4    votes: number;
5
6    constructor(title, link) {
7      this.title = title;
8      this.link = link;
9      this.votes = 0;
10   }
```

[6] http://en.wikipedia.org/wiki/Law_of_Demeter

```
11
12    voteUp() {
13      this.votes += 1;
14      return false;
15    }
16
17    voteDown() {
18      this.votes -= 1;
19      return false;
20    }
21  }
22
23  @Component({
24    selector: 'reddit-article'
25  })
26  @View({
27    template: `
28        <article>
29          <div class="votes">{{ article.votes }}</div>
30          <div class="main">
31            <h2>
32              <a href="{{ article.link }}">{{ article.title }}</a>
33            </h2>
34            <ul>
35              <li><a href (click)='article.voteUp()'>upvote</a></li>
36              <li><a href (click)='article.voteDown()'>downvote</a></li>
37            </ul>
38          </div>
39        </article>
40        `
41  })
42  class RedditArticle {
43    article: Article;
44
45    constructor() {
46      this.article = new Article('Angular 2', 'http://angular.io');
47    }
48  }
```

Checkout our `RedditArticle` component definition now: it's so short! We've moved a lot of logic **out** of our component and into our models. An analogous MVC guideline would be Fat Models, Skinny Controllers[7]. The idea is that we want to move most of our domain logic to our models so that our components do the minimum work possible.

After reloading your browser, you'll notice everything works the same way, but we now have clearer code.

## Storing multiple `Articles`

Let's write the code that allows us to have a list of multiple `Articles`.

Start by changing `RedditApp` to have a collection of articles:

```
1  class RedditApp {
2    articles: Array<Article>;
3
4    constructor() {
5      this.articles = [
6        new Article('Angular 2', 'http://angular.io'),
7        new Article('Fullstack', 'http://fullstack.io')
8      ];
9    }
10
11   addArticle(title, link) {
12     console.log("Adding article with title", title.value, "and link", link.value\
13 );
14   }
15 }
```

Notice that our `RedditApp` has the line:

```
1    articles: Array<Article>;
```

The `Array<Article>` might look a little funny if you're not used to typing your javascript. The word for this pattern is *generics*. It's a concept seen in Java, among others, and the idea is that your collection (the `Array`) is typed. That is, the `Array` is a collection that will only hold objects of type `Article`.

We can populate that list by setting `this.articles` in the constructor:

[7] http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model

```
  constructor() {
    this.articles = [
      new Article('Angular 2', 'http://angular.io'),
      new Article('Fullstack', 'http://fullstack.io')
    ];
  }
```

## Configuring a `RedditArticle` Component with properties

Now that we have a list of `Article` *models*, how can we pass them to our `RedditArticle` *component*?

Here we're introducing a new attribute of `Component` called **`properties`**.

We can configure out `Component` with `properties` that are passed to it from it's parent.

Previously we had our `RedditArticle` Component class defined like this:

```
class RedditArticle {
  article: Article;

  constructor() {
    this.article = new Article('Angular 2', 'http://angular.io');
  }
}
```

The problem here is that we've hard coded a particular `Article` in the constructor.

What we would really like to do is be able to configure the `Article` we want to display in markup, like this:

```
<reddit-article article="article1"></reddit-article>
<reddit-article article="article2"></reddit-article>
```

In order to use an attribute in the HTML as an input to our component we use the `properties` option of `Component`.

It looks like this:

```
@Component({
  selector: 'reddit-article',
  properties: ['article'],
})
@View({
  // same ...
})
class RedditArticle {
  article: Article;
}
```

Notice that `properties` is an `Array` where you enumerate all the properties your component can receive.

So our full listing of `RedditArticle` looks like this:

```
@Component({
  selector: 'reddit-article',
  properties: ['article'],
})
@View({
  template: `
      <article>
        <div class="votes">{{ article.votes }}</div>
        <div class="main">
          <h2>
            <a href="{{ article.link }}">{{ article.title }}</a>
          </h2>
          <ul>
            <li><a href (click)='article.voteUp()'>upvote</a></li>
            <li><a href (click)='article.voteDown()'>downvote</a></li>
          </ul>
        </div>
      </article>
      `
})
class RedditArticle {
  article: Article;
}
```

Here we added a new `properties` property to the `@Component` annotation. This allows the component to receive an `article` property from the DOM, while making it available as a `article` within the template.

What's great here is that we've totally eliminated the functions in the class definition `RedditArticle`! This helps make this component be a bit easier to reason about.

## Rendering a list of Articles

Let's configure `RedditApp` to render all the `articles`. To do so, instead of having the `<reddit-article>` tag alone, we are going to use the `For` directive again, to render multiple tags:

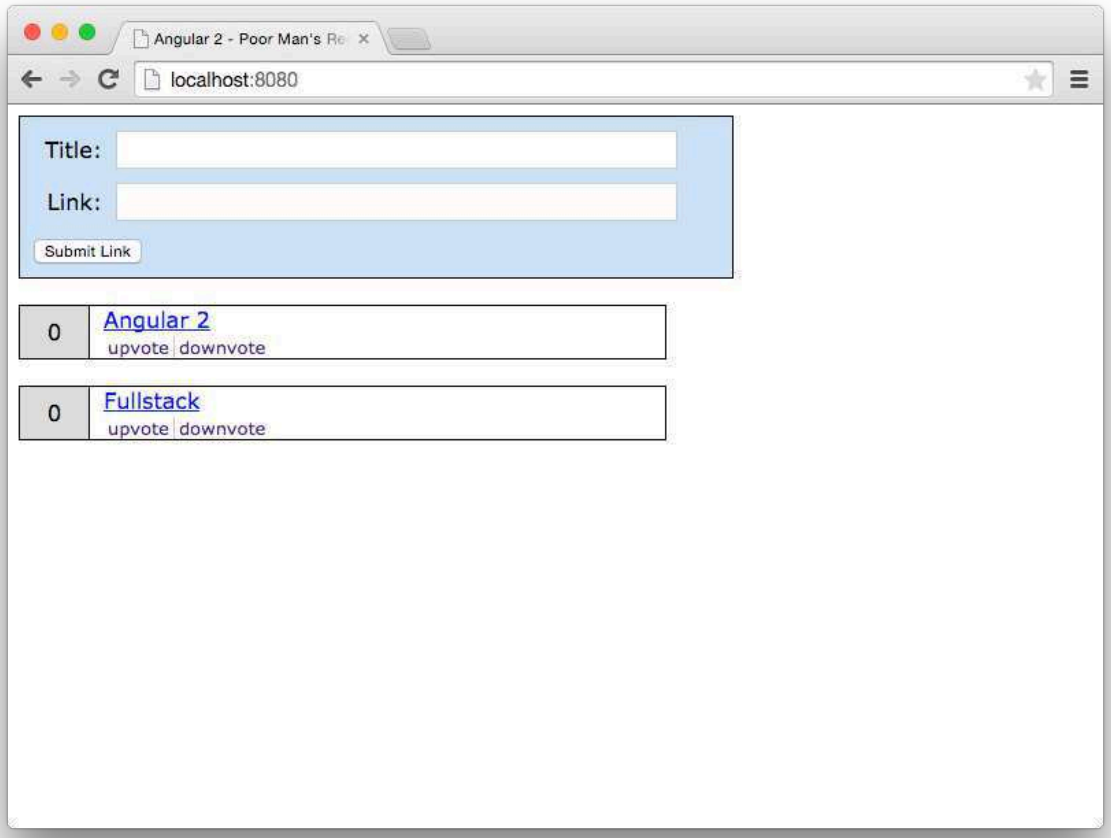
```
@Component({
  selector: 'reddit'
})
@View({
  template: `
    <section class="new-link">
      <div class="control-group">
        <div><label for="title">Title:</label></div>
        <div><input name="title" #newtitle></div>
      </div>
      <div class="control-group">
        <div><label for="link">Link:</label></div>
        <div><input name="link" #newlink></div>
      </div>

      <button (click)="addArticle(newtitle, newlink)">Submit Link</button>
    </section>

    <reddit-article
       *ng-for="#article of articles"
       [article]="article">
    </reddit-article>
  `,

  directives: [RedditArticle, NgFor]
})
class RedditApp {
  addArticle(title, link) {
    console.log("Adding article with title", title.value, "and link", link.value\
);
  }
}
```

Remember when we rendered a list of names as a bullet list using the `NgFor` directives? Well, that also works for rendering multiple components. The `*ng-for="#article of articles"` syntax will

iterate through the list of articles and creating the local variable `article`.

To indicate the article we want to render on each iteration, we use the `[article]="article"` notation. The square brackets indicate that we are setting the component's `article` variable to receive the template's `article` local variable we declared inside our `*ng-for` clause. Phew.

If you reload your browser now, you can see that both articles will be rendered:




**Multiple articles being rendered**

The only thing left now is to change the `addArticle` method to add a new `Article` when you click the submit button:

```
addArticle(title, link) {
  this.articles.push(new Article(title.value, link.value));
  title.value = '';
  link.value = '';
}
```

This will:

1. create a new `Article` instance with the submitted title and value
2. add it to the array of `Articles` and
3. clear the input values

If you add a new article and click **Submit Link** you will see the new article added!

As a final touch, let's just add a hint next to the link that shows the domain the user will be redirected to when the link is clicked.

Add this `domain` method to the `Article` class:

```
domain() {
  var link = this.link.split('//')[1];
  return link.split('/')[0];
}
```

And add it to the `RedditArticle`'s template:

```
@View({
  template: `
  <article>
    <div class="votes">{{ article.votes }}</div>
    <div class="main">
      <h2>
        <a href="{{ article.link }}">{{ article.title }}</a>
        <span>({{ article.domain() }})</span>
      </h2>
      <ul>
        <li><a href (click)='article.voteUp()'>upvote</a></li>
        <li><a href (click)='article.voteDown()'>downvote</a></li>
      </ul>
    </div>
  </article>
  `
})
```

And now when we reload the browser, we should see the completed application.

## Wrapping Up

We did it! We've created our first Angular 2 App. That wasn't so bad, was it? There's lots more to learn: understanding data flow, making AJAX requests, built-in components, routing, manipulating the DOM etc.

But for now, bask in your success! Much of writing Angular 2 apps is just as we did above:

1. Split your app into components
2. Create the views
3. Define your models
4. Display your models
5. Add interaction

Onward!

This is the end of the preview!

Head over to ng-book.com/2 to download the full package.

The full package includes the book, the source code, and a video screencast!