START WRITING OUR ANGULARJS APPLICATION

The files that you see in the web browser can be found in the ***app*** subfolder of your ***mytodo*** directory. All the instructions in this section assume that you are editing files in this *app* folder. If you are unsure about any of the changes you should be doing, refer to the final source code.

CREATE A NEW TEMPLATE TO SHOW A TODO LIST

Open ***views/main.html***.

To start from a cleaner slate, delete everything from your *main.html* file except for the `div` with a class of `"jumbotron"`. Replace `"jumbotron"` with the class name `"container"`.

This is all you should have in *main.html* now:

```
<div class="container">
</div>
```

Open ***scripts/controllers/main.js***.

Modify this boilerplate Angular Controller to contain a list of `todos` instead of `awesomeThings`:

```
'use strict';

angular.module('mytodoApp')
  .controller('MainCtrl', function ($scope) {
    $scope.todos = ['Item 1', 'Item 2', 'Item 3'];
  });
```

Then modify our view (*main.html*) to output our todos items as HTML text `input` fields:

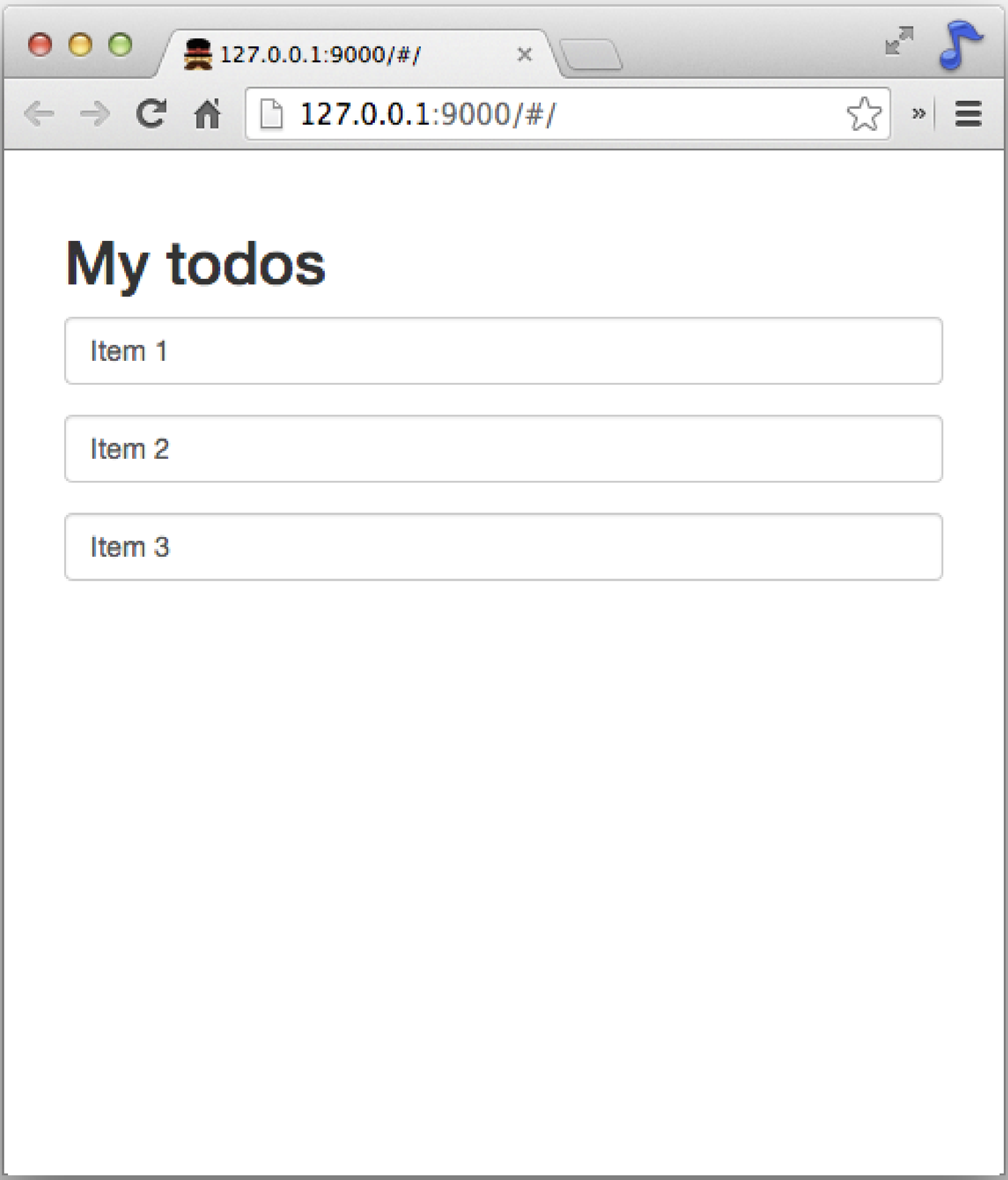
```
<div class="container">
  <h2>My todos</h2>
  <p class="form-group" ng-repeat="todo in todos">
    <input type="text" ng-model="todo" class="form-control">
  </p>
</div>
```

The `ng-repeat` attribute on the paragraph tag is an Angular directive that instantiates a template once per item from a collection.

In our case, imagine that the paragraph element and its content is turned into a virtual rubber stamp by adding the `ng-repeat` attribute. For each item in the `todos` array, Angular will stamp out a new instance of the `<p><input></p>` HTML block.

The `ng-model` attribute is another Angular directive that works with `input`, `select`, `textarea`, and custom directives to create a two-way data binding. In our example, it populates a text `input` field with the value from the current `todo` item in the `ng-repeat` loop.

Let's see `ng-repeat` and `ng-model` in action within the browser. Upon saving, our application should now look like this:



Manually update `$scope.todos` with a fourth todo item:

```
$scope.todos = ['Item 1', 'Item 2', 'Item 3', 'Item 4'];
```

With live reload, you should see the new todo item appear in the list.

Manually remove the fourth item and watch it disappear from the list.

ADD A TODO

Let’s implement a way for the user to add new todo items to the list.

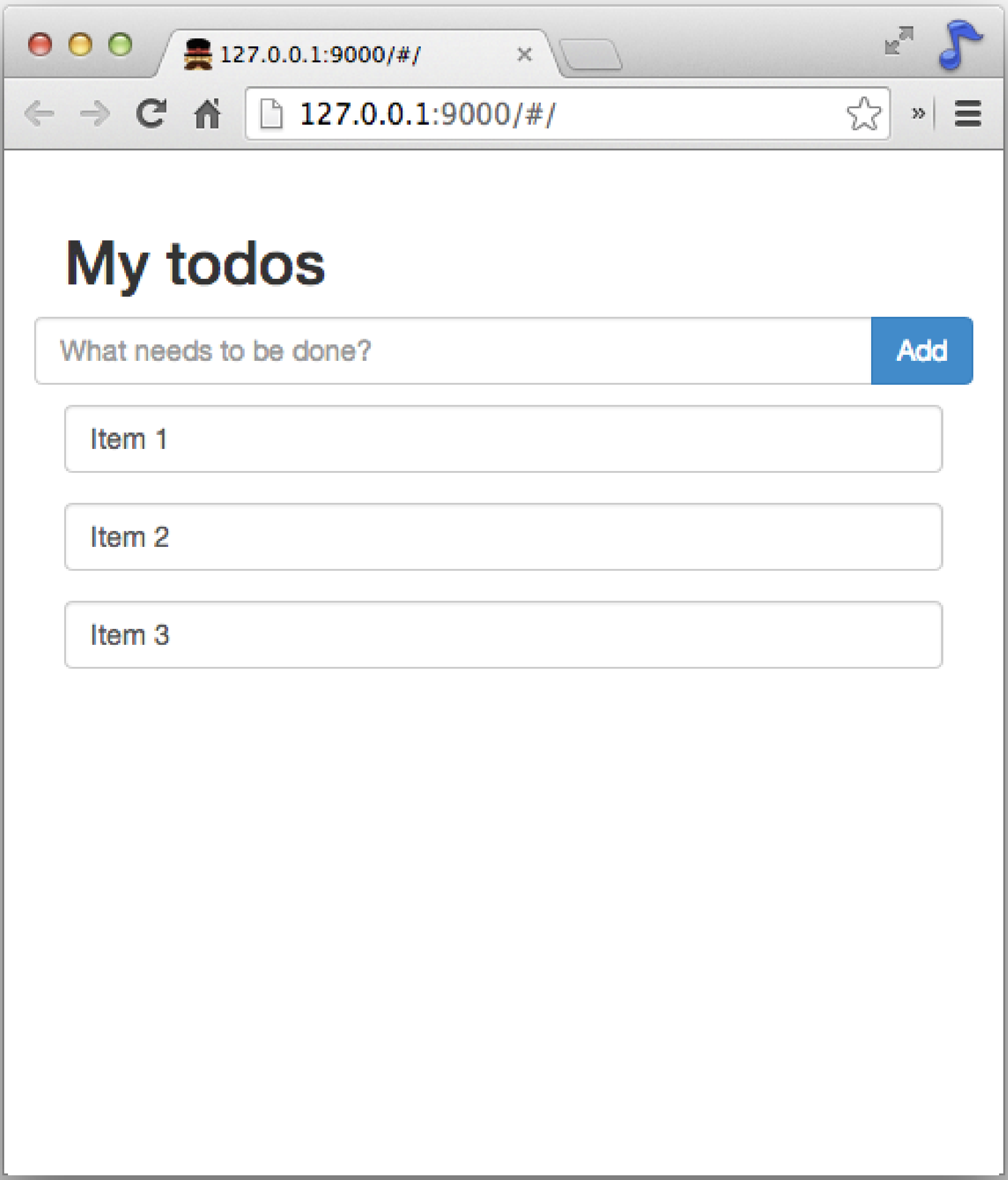
Modify *main.html* by adding a `form` element in between the `<h2>` and `<p>` elements from the previous section. Your *main.html* should now look like this:

```
<div class="container">
  <h2>My todos</h2>

  <!-- Todos input -->
  <form role="form" ng-submit="addTodo()">
    <div class="row">
      <div class="input-group">
        <input type="text" ng-model="todo" placeholder="What needs to be done?"
class="form-control">
        <span class="input-group-btn">
          <input type="submit" class="btn btn-primary" value="Add">
        </span>
      </div>
    </div>
  </form>
  <p></p>

  <!-- Todos list -->
  <p class="form-group" ng-repeat="todo in todos">
    <input type="text" ng-model="todo" class="form-control">
  </p>
</div>
```

This adds a HTML form with a submit button to the top of the page. It utilises another Angular directive, `ng-submit` which we'll get to next. Return to your browser and the UI should now look similar to this:



If you click **Add** currently, nothing will happen — let's change that.

`ng-submit` binds an Angular expression to the onsubmit event of the form. If no action attribute is applied to the form, it also prevents the default browser behaviour. In our example we've added an Angular expression of `addTodo()`.

The following `addTodo()` function pushes new todo items onto the existing todo items array and then clears the text input field:

```
$scope.addTodo = function () {
  $scope.todos.push($scope.todo);
  $scope.todo = '';
  };
```

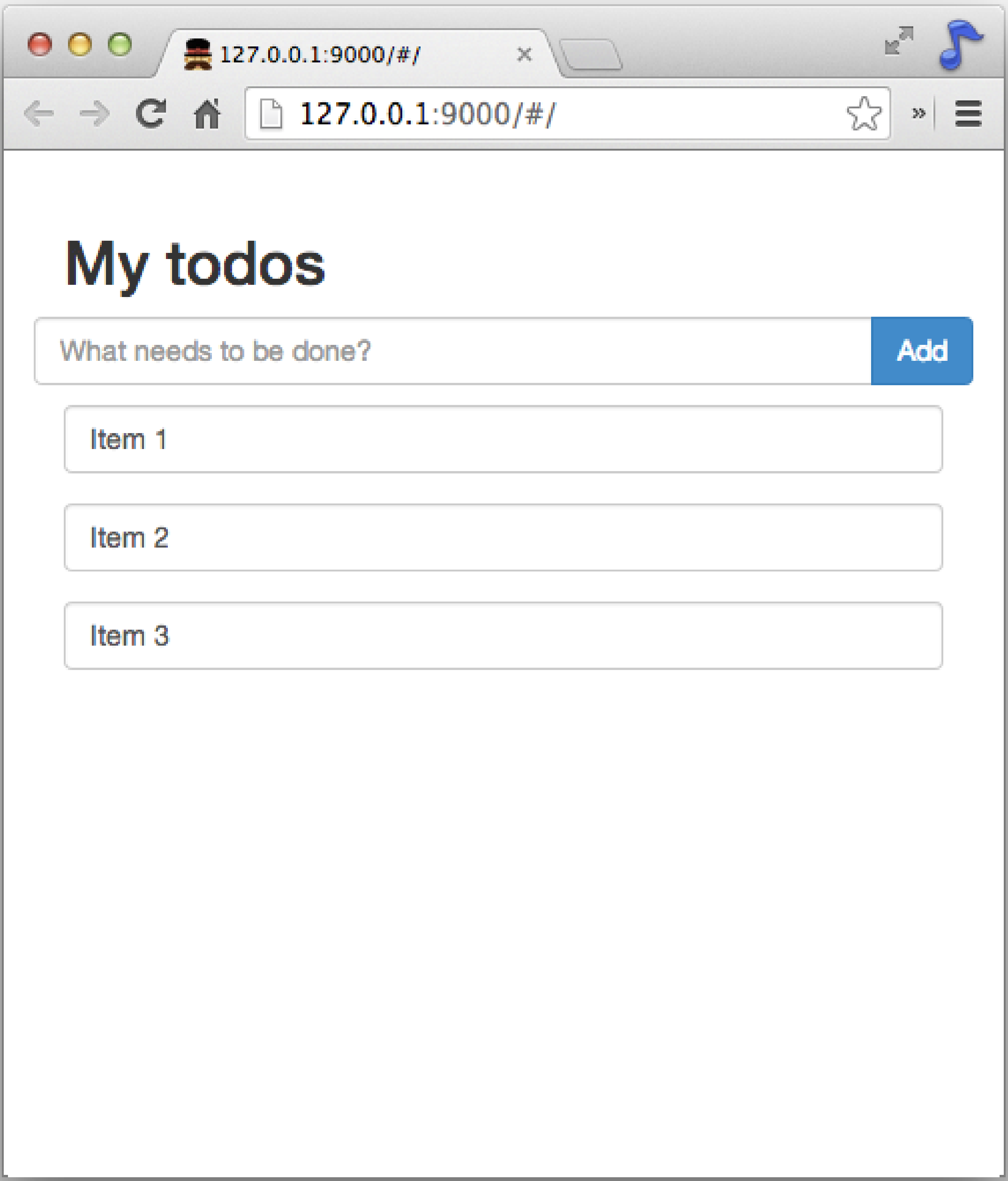
Edit *main.js* by adding the `addTodo()` function within the `MainCtrl` controller definition. Your complete controller should look like this:

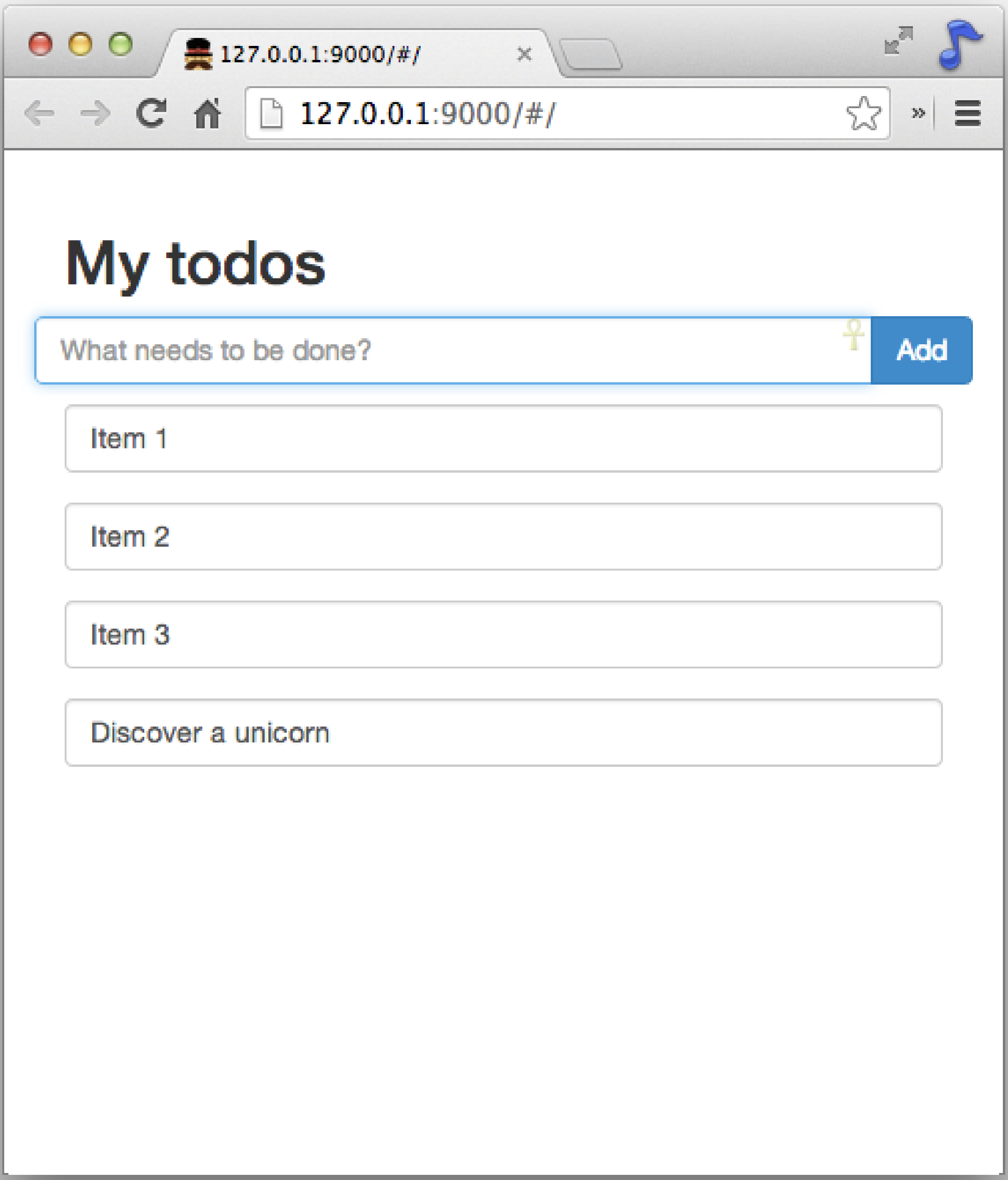
```
'use strict';

angular.module('mytodoApp')
  .controller('MainCtrl', function ($scope) {
    $scope.todos = ['Item 1', 'Item 2', 'Item 3'];
    $scope.addTodo = function () {
      $scope.todos.push($scope.todo);
      $scope.todo = '';
    };
  });
```

Note: If you encounter linting errors in the command line window, this may be due to indentation warnings being thrown from jshint. They are only warnings so your todo app will continue to work. However, do look at the suggestions made from jshint and adjust your code accordingly for clean and readable code.

View the app in the browser again. Type some text in the input field for a new todo item and hit **Add**. It will be immediately reflected in your todos list!


127.0.0.1:9000/#/
127.0.0.1:9000/#/
My todos
What needs to be done?
Add
Item 1
Item 2
Item 3



> Note: If you enter in more than one blank todo item, or a todo item with the same name, your todo app will unexpectedly stop working. :(As a fun exercise on your own time, enhance the `addTodo()` function with error checking.

REMOVE A TODO

Let's now add the ability to remove a todo item. We'll need to add a new remove button alongside each todo item.

Going back to our view template (*main.html*), add a button to the existing `ng-repeat` directive. And to make sure our input field and remove button line up nicely, change the class on the paragraph tag from `"form-group"` to `"input-group"`.

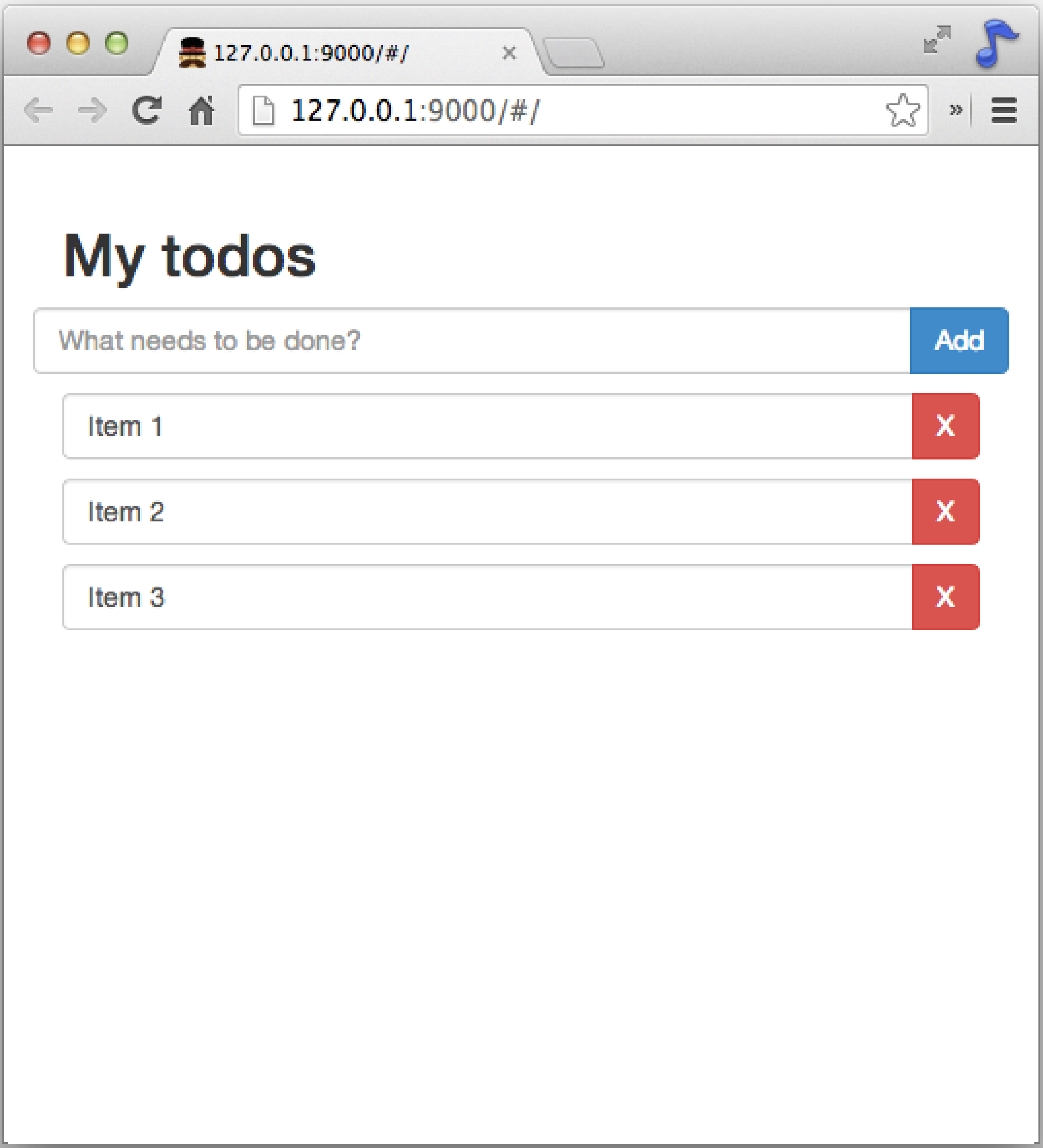
Previous markup:

```
<!-- Todos list -->
<p class="form-group" ng-repeat="todo in todos">
  <input type="text" ng-model="todo" class="form-control">
</p>
```

New markup:

```
<!-- Todos list -->
<p class="input-group" ng-repeat="todo in todos">
  <input type="text" ng-model="todo" class="form-control">
  <span class="input-group-btn">
    <button class="btn btn-danger" ng-click="removeTodo($index)" aria-
label="Remove">X</button>
  </span>
</p>
```

Have a look at your browser. Your todo app is looking snazzy!



We introduced a new Angular directive above, `ng-click`. `ng-click` allows you to specify custom behaviours when an element is clicked. In this instance, we call `removeTodo()` and pass `$index` to the function.

The value of `$index` will be the array index of the current `todo` item within the `ng-repeat` directive. For example, the first item will have an array index of 0 and `removeTodo()` will be passed the value of 0. Similarly, the last item of a todo list with 5 items will have an array index of 4 and `removeTodo()` will be passed a value of 4.

Let's now add some logic for removing todo items to our controller. The following `removeTodo()` function removes one todo item from the items array using the JavaScript `splice()` method at the given `$index` value:

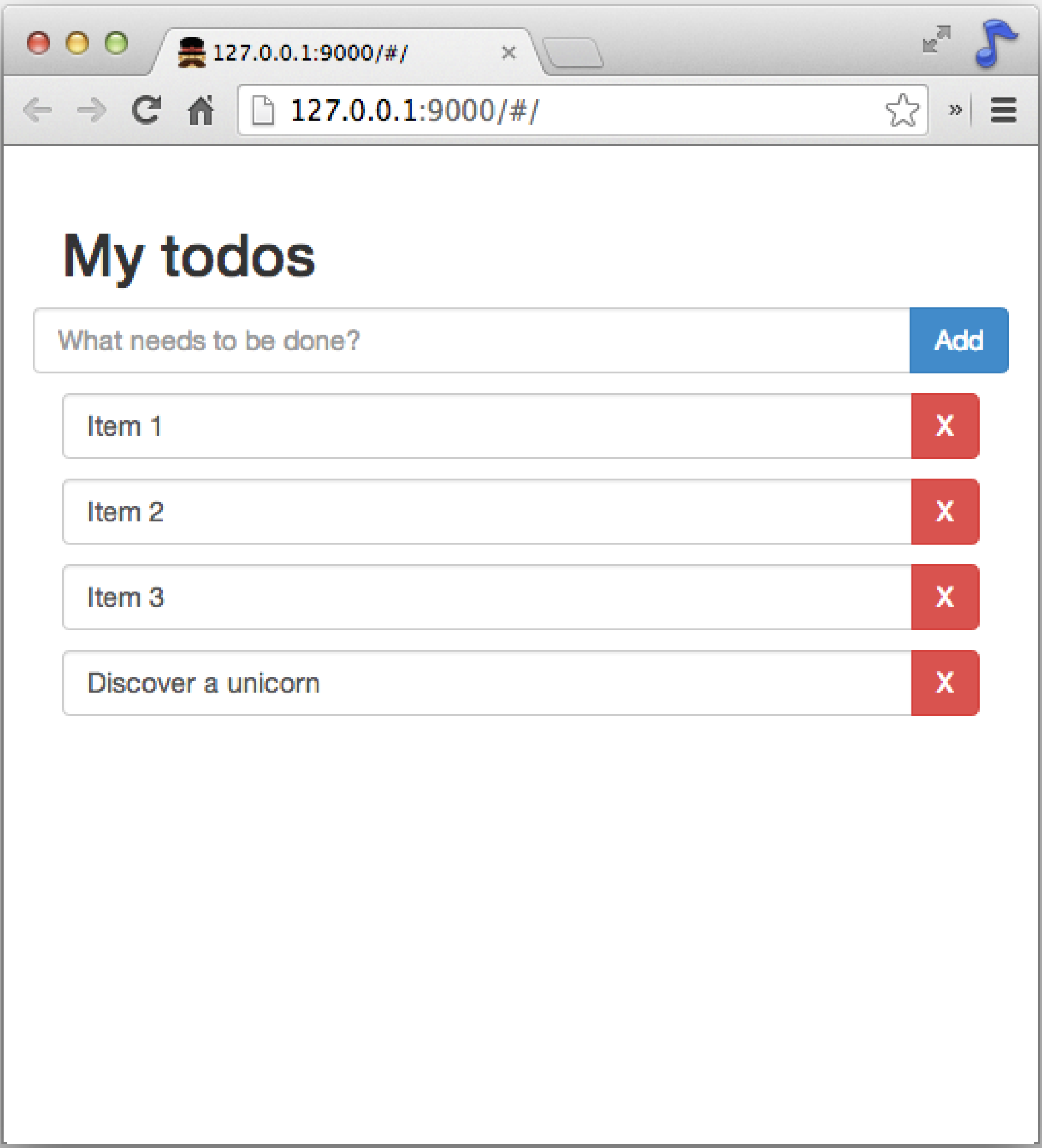
```
$scope.removeTodo = function (index) {
  $scope.todos.splice(index, 1);
};
```

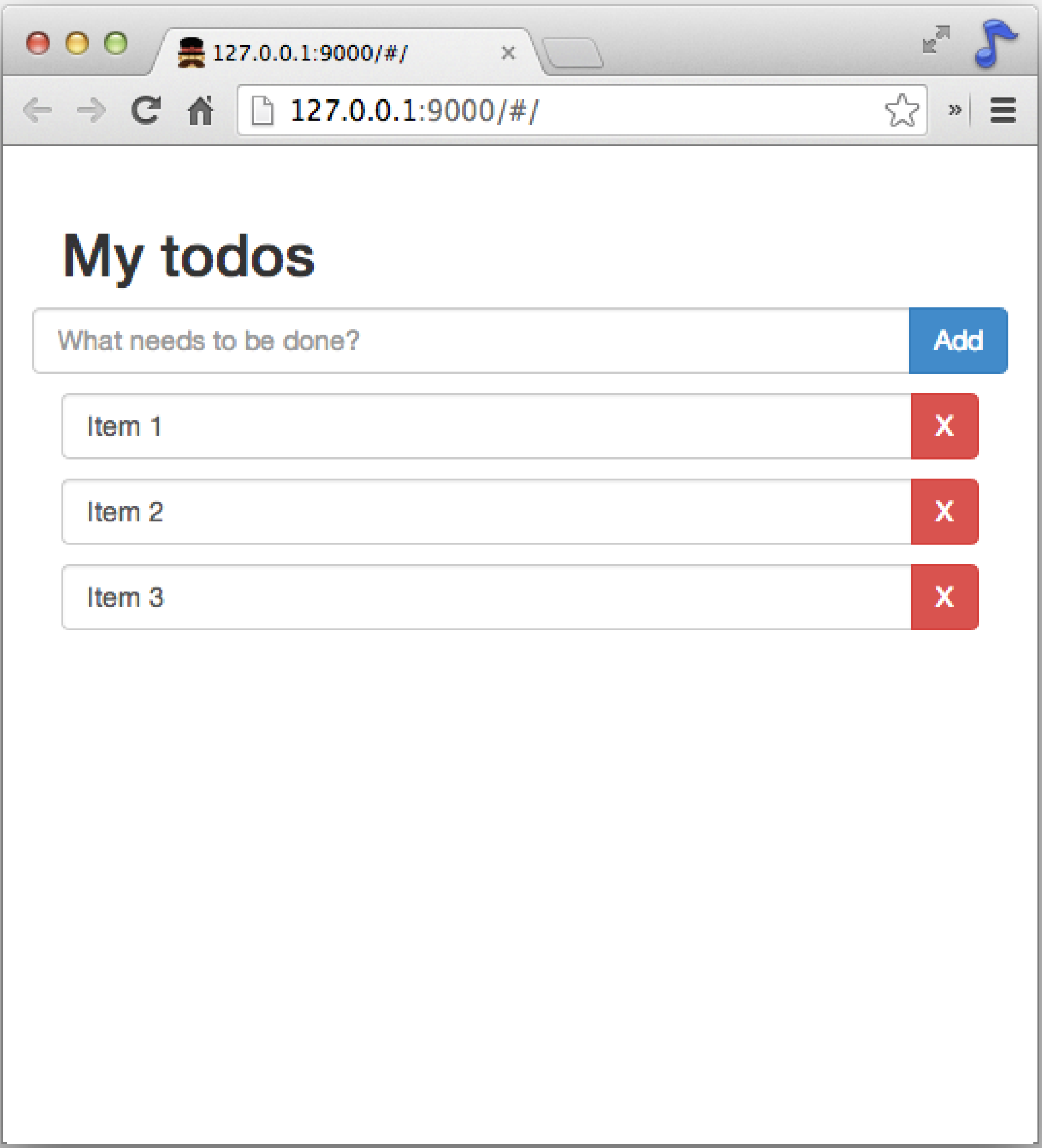
The complete controller (*main.js*) with the new `removeTodo()` function is below:

```
'use strict';

angular.module('mytodoApp')
  .controller('MainCtrl', function ($scope) {
    $scope.todos = ['Item 1', 'Item 2', 'Item 3'];
    $scope.addTodo = function () {
      $scope.todos.push($scope.todo);
      $scope.todo = '';
    };
    $scope.removeTodo = function (index) {
      $scope.todos.splice(index, 1);
    };
  });
```

Back in the browser, you can now hit the **X** button to remove the item from the todo list. Fantastic!


127.0.0.1:9000/#/
127.0.0.1:9000/#/
My todos
What needs to be done?
Add
Item 1
X
Item 2
X
Item 3
X
Discover a unicorn
X



My todos

What needs to be done?

Item 1

Item 2

Item 3

One thing you might notice is that we don't have a way to persist our todo list. Any time we refresh the page our todo items are reset back to the defaults in our `todos` array hardcoded in *main.js*. Don't worry, we'll fix this in Step 9 after we learn more about installing packages with Bower next in Step 7.

« Return to overview or **Go to the next step »**