Proffesional development program

AngularJS

Contents

[Chapter 1. Prerequisite 4](#_Toc435433199)

[HTML 4](#_Toc435433200)

[CSS 4](#_Toc435433201)

[JavaScript 4](#_Toc435433202)

[jQuery 5](#_Toc435433203)

[S.P.A. Theory 5](#_Toc435433204)

[Chapter2. The AngularJS Philosophy 8](#_Toc435433205)

[What Is MVC (Model-View-Controller)? 8](#_Toc435433206)

[Core beliefs 8](#_Toc435433207)

[Data-driven (via data-binding) 8](#_Toc435433208)

[Declarative 9](#_Toc435433209)

[Separate your concerns 9](#_Toc435433210)

[Dependency Injection 9](#_Toc435433211)

[Extensible 10](#_Toc435433212)

[Test first, test again, keep testing 10](#_Toc435433213)

[Chapter 3. Starting out with AngularJS 11](#_Toc435433214)

[Bootstrapping AngularJS 11](#_Toc435433215)

[Modules 11](#_Toc435433216)

[Using ng-bind Versus Double Curlies 12](#_Toc435433217)

[First Controller 12](#_Toc435433218)

[$scope Versus controllerAs Syntax 13](#_Toc435433219)

[How AngularJS is working behind the scenes 13](#_Toc435433220)

[More directives 14](#_Toc435433221)

[AngularJS $watch() , $digest() and $apply() 14](#_Toc435433222)

[Forms 15](#_Toc435433223)

[One/two-way data-binding 15](#_Toc435433224)

[Working with Forms 16](#_Toc435433225)

[Form Validation and States 16](#_Toc435433226)

[Nested Forms with ng-form 18](#_Toc435433227)

[Working with Filters 18](#_Toc435433228)

[Bibliography 20](#_Toc435433229)

# Chapter 1. Prerequisite

## HTML

HyperText Markup Language, commonly referred to as HTML, is the standard markup language used to create web pages. Web browsers can read HTML files and render them into visible or audible web pages. HTML describes the structure of a website semantically along with cues for presentation, making it a markup language, rather than a programming language.

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. (HTML, 2015)

## CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language. Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG and XUL, and is applicable to rendering in speech, or on other media. (Cascading Style Sheets, 2015)

## JavaScript

JavaScript is a high-level, dynamic, untyped, and interpreted programming language. Alongside HTML and CSS, it is one of the three essential technologies of World Wide Web content production; the majority of websites employ it and it is supported by all modern web browsers without plug-ins. JavaScript is prototype-based with first-class functions, making it a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles. JavaScript is also used in environments that are not web-based, such as PDF documents, site-specific browsers, and desktop widgets. (JavaScript, 2015)

The secret is that the JavaScript engine makes two passes over code when it comes into scope. On the first pass it initializes variables and on the second pass it executes code. Variables are scoped by functions in JavaScript and they’re either global or local.Global variables are accessible everywhere, and local variables are only accessible where they are declared. The only block that defines scope for a variable in JavaScript is a function. JavaScript variables are declared following the var keyword. A variable can contain any type of data: arrays, integers, floats, strings, and so on. The variable type isn’t specified, so JavaScript is considered a loosely typed language. Even after a value is assigned to a variable, the type of value can be changed by assigning a value with a different type, so it’s also considered a dynamic language. (Michael S. Mikowski, 2014)

## jQuery

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery is the most popular JavaScript library in use today, with installation on 65% of the top 10 million highest-trafficked sites on the Web. jQuery is free, open-source software licensed under the MIT License. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications. (jQuery, 2015)

## S.P.A. Theory

An SPA delivers a desktop application in the browser. The result is a highly responsive experience that surprises and delights users instead of using and annoying them. An SPA is an application delivered to the browser that doesn’t reload the page during use. We can think of an SPA as a fat client that’s loaded from a web server. (Michael S. Mikowski, 2014)

There are various techniques available that enable the browser to retain a single page even when the application requires server communication.

* JavaScript frameworks such as AngularJS, Backbone.js, Ember.js, Meteor, or React.

AngularJS is a fully client-side framework. AngularJS's templating is based on bidirectional UI data binding. Data-binding is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes. The HTML template is compiled in the browser. The compilation step creates pure HTML, which the browser re-renders into the live view. The step is repeated for subsequent page views. In traditional server-side HTML programming, concepts such as controller and model interact within a server process to produce new HTML views. In the AngularJS framework, the controller and model state are maintained within the client browser. Therefore new pages are generated without any interaction with a server.

Backbone.js is a JavaScript framework with a RESTful JSON interface and is based on the model–view–presenter (MVP) application design paradigm. Backbone is known for being lightweight, as its only hard dependency is on one JavaScript library, Underscore.js, plus jQuery for use of the full library. It is designed for developing single-page web applications, and for keeping various parts of web applications (e.g. multiple clients and the server) synchronized. Backbone was created by Jeremy Ashkenas, who is also known for CoffeeScript and Underscore.js. (Backbone.js, 2015)

Ember.js is an open-source JavaScript application framework, based on the model-view-controller (MVC) pattern. It allows developers to create scalable single-page web applications by incorporating common idioms and best practices into the framework. Ember is used on many popular websites, including Discourse, Groupon, Vine, Live Nation, Nordstrom, and Chipotle. Although primarily considered a framework for the web, it is also possible to build desktop and mobile applications in Ember. The most notable example of an Ember desktop application is Apple Music, a feature of the iTunes desktop application.

Meteor, or MeteorJS is an open-source JavaScript web application framework written using Node.js. Meteor allows for rapid prototyping and produces cross-platform (web, Android, iOS) code. It integrates with MongoDB and uses the Distributed Data Protocol and a publish–subscribe pattern to automatically propagate data changes to clients without requiring the developer to write any synchronization code. On the client, Meteor depends on jQuery and can be used with any JavaScript UI widget library. Meteor is developed by the Meteor Development Group.

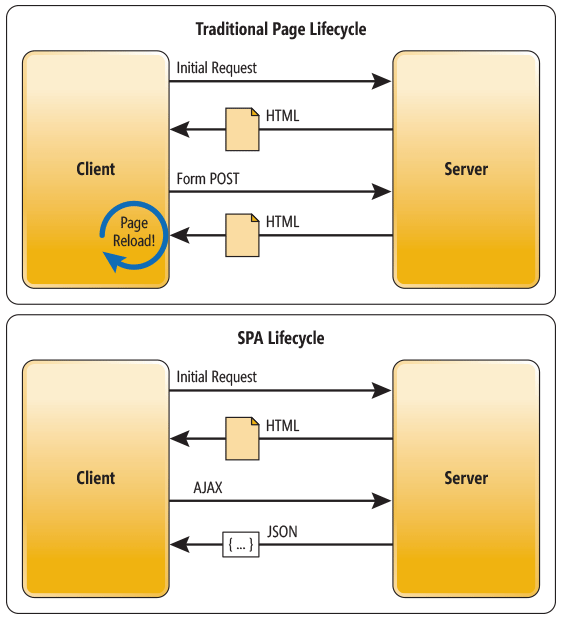
* AJAX

The most prominent technique currently being used is Ajax. Predominantly using the XMLHttpRequest object from JavaScript, other AJAX approaches include using IFRAME or script HTML elements. Popular libraries like jQuery, which normalize AJAX behavior across browsers from different manufacturers, have further popularized the AJAX technique.

* Websockets

WebSockets are a bidirectional stateful real-time client-server communication technology part of the HTML5 specification, superior to AJAX in terms of performance and simplicity.

(Single-page application, 2015)



[The Traditional Page Lifecycle vs. the SPA Lifecycle](https://msdn.microsoft.com/en-us/magazine/dn463786.aspx)

# Chapter2. The AngularJS Philosophy

AngularJS is a superheroic JavaScript MVC framework for the Web. And the best part? It’s all done in pure JavaScript and HTML. No need to learn another new programming or templating language (though you do have to understand the MVC).

## What Is MVC (Model-View-Controller)?

* a way to separate logical units and concerns when developing large applications.

The MVC architectural pattern divides an application into three distinct, modular parts:

* The **model** is the driving force of the application. This is generally the data behind the application, usually fetched from the server. Any UI with data that the user sees is derived from the model, or a subset of the model.
* The **view** is the UI that the user sees and interacts with. It is dynamic, and generated based on the current model of the application.
* The **controller** is the business logic and presentation layer, which peforms actions such as fetching data, and makes decisions such as how to present the model, which parts of it to display, etc.

The model is the data, the view is the UI, and the controller is the business logic.

## Core beliefs

### Data-driven (via data-binding)

We don’t have to waste time funneling data back and forth between the UI and the JavaScript in an AngularJS application. We just bind to the data in our HTML and AngularJS takes care of getting its value into the UI. Not only that, but it also takes care of updating the UI whenever the data changes. The **two-way data-binding** ensures that our controller and the UI share the same model, so that updates to one (either from the UI or in our code) update the other automatically.

### Declarative

AngularJS instead promotes a declarative paradigm, where you declare right in your HTML what it is you are trying to accomplish. This is done through something that AngularJS calls directives. Directives basically extend the vocabulary of HTML to teach it new tricks. We let AngularJS figure out how to accomplish what we want it to do, whether it is creating tabs or datepickers.

### Separate your concerns

There is the actual data that you want to display to the user, or get the user to enter through your application. This is the **model** in an AngularJS project, which is mostly pure data, and represented using JSON objects.

Then there is the user interface or the final rendered HTML that the user sees and interacts with, which displays the data to the user. This is the **view**.

Finally, there is the actual business logic and the code that fetches the data, decides which part of the model to show to the user, how to handle validation, and so on —core logic specific to your application. This is the **controller** for an AngularJS application.

AngularJS is not completely MVC; the controller will never have a direct reference to the view. This is great because it keeps the controller independent of the view, and also allows us to easily test the controller without needing to instantiate a DOM.

### Dependency Injection

Is the concept of asking for the dependencies of a particular controller or service, instead of instantiating them inline via the new operator or calling a function explicitly. Some other part of your code becomes responsible (in this case, the injector) for figuring out how to create those dependencies and provide them when asked for.

Is used across all of its parts, from controllers and services to modules and tests. It allows you to easily write modular, reusable code so that you can use it cleanly and simply as needed.

### Extensible

We can develop robust and complex directives that integrate with third-party libraries like jQueryUI and BootStrap, to name a few, to create a language that is specific to our needs. AngularJS has a great core set of directives for us to get started, and an API that allows us to do everything AngularJS does and more. Our imagination is really the only limit for creating declarative, reusable components.

### Test first, test again, keep testing

Every bit and piece of AngularJS is designed to be testable, from its controllers, services, and directives to its views and routes. Between Dependency Injection and the controller being independent of references to the view, the JS code that we write in an AngularJS application can easily be tested. Because we get the same Dependency Injection system in our tests as in our production code, we can easily instantiate any service without worrying about its dependencies.

# Chapter 3. Starting out with AngularJS

## Bootstrapping AngularJS

This is done through the **ng-app** directive. This is the first and most important directive that AngularJS has, which denotes the section of HTML that AngularJS controls. Putting it on the <html> tag tells AngularJS to control the entire HTML application. We could also put it on the <body> or any other element on the page.

Any element that is a child of that will be handled with AngularJS and be annotated with directives, and anything outside would not be processed.

**Only one** AngularJS application can be auto-bootstrapped per HTML document. The first ngApp found in the document will be used to define the root element to auto-bootstrap as an application. To run multiple applications in an HTML document you must manually bootstrap them **using angular.bootstrap** instead. AngularJS applications cannot be nested within each other. (ngApp, 2015)

## Modules

Modules are AngularJS’s way of packaging relevant code under a single name. An AngularJS module has two parts to it:

* A module can define its own controllers, services, factories, and directives. These are functions and code that can be accessed throughout the module.
* The module can also depend on other modules as dependencies, which are defined when the module is instantiated. What this means is that AngularJS will go and find the module with that particular name, and ensure that any functions, controllers, services, etc. defined in that module are made available to all the code defined in this module.

In addition to being a **container** for related JavaScript, the module is also what AngularJS uses to bootstrap an application. The ng-app directive takes an optional argument, which is the name of the module to load during bootstrapping.

## Using ng-bind Versus Double Curlies

There is no functional difference between the two; both take the value from the controller and display it in the UI. Both of them also keep it data-bound and up to date, so if the value underneath changes, the UI will change automatically.

The advantage ng-bind has over the double-curly notation is that it takes AngularJS time to bootstrap and execute before it can find and replace all the double curly braces from the HTML. That means, for a portion of a second while the browser starts, you might see flashing double curly braces in the UI before AngularJS has the chance to kick in and replace them. This is only for the very first page load, and not on views loaded after the first load. You will not have that issue with ng-bind.

## First Controller

The common responsibilities of a controller in an AngularJS application include:

* Fetching the right data from the server for the current UI.
* Deciding which parts of that data to show to the user.
* Presentation logic, such as how to display elements, which parts of the UI to show, how to style them, etc.
* User interactions, such as what happens when a user clicks something or how a text input should be validated.

An AngularJS controller is almost always directly linked to a view or HTML. We will never have a controller that is not used in the UI (that kind of business logic goes into services). It acts as the gateway between our model, which is the data that drives our application, and the view, which is what the user sees and interacts with.

We also introduce a new directive, **ng-controller.** This is used to tell AngularJS to go instantiate an instance of the controller with the given name, and attach it to the DOM element. This directive allows us to associate an instance of a controller with a UI element.

The HTML connects parts of the DOM to controllers, functions, and variables, and not the other way around. This is one of the core principles of AngularJS at play here. An AngularJS application is a data-driven app.

Anything that the user needs to see, or the HTML needs to use, needs to be defined on this. Anything that the HTML does not directly access should not be put on this, but should rather be saved as local variables in the controller’s scope.

AngularJS creates scopes or context for various elements in the DOM to ensure that there is **no global state** and each element accesses only what is relevant to it. These scopes have a **parent-child relation** by default, which allows children scopes to access functions and controllers from a parent scope.

### $scope Versus controllerAs Syntax

If you used AngularJS prior to 1.2, you might have expected the $scope variable to be injected into the controller. In AngularJS 1.2 and later, there is a new syntax, the controllerAs syntax, which allows us to define the variables on the controller instance using the **this** keyword, and refer to them through the controller from the HTML.

The advantage of this over the earlier syntax is that it makes it explicit in the HTML which variable or function is provided by which controller and which instance of the controller. It becomes immediately obvious because the controller instance is present in the HTML.

Also, as good practice, we avoid referring to the **this** keyword inside the controller, preferring to use a proxy **self** variable, which points to this.

## How AngularJS is working behind the scenes

1. The HTML is loaded. This triggers requests for all the **scripts** that are a part of it.
2. After the entire document has been loaded, AngularJS kicks in and looks for the **ng-app** directive.
3. When it finds the ng-app directive, it looks for and loads the **module** that is specified and attaches it to the element.
4. AngularJS then **traverses** the children DOM elements of the root element with the ng-app and starts looking for directives and bind statements.
5. Each time it hits an **ng-controller** or an **ng-repeat** directive, it creates what we call a **scope** in AngularJS. A scope is the context for that element. The scope dictates what each DOM element has access to in terms of functions, variables, and the like.
6. AngularJS then adds **watchers** and **listeners** on the variables that the HTML accesses, and **keeps track** of the current value of each of them. When that value changes, AngularJS updates the UI immediately.
7. Instead of polling or some other mechanism to check if the data has changed, AngularJS **optimizes** and checks for updates to the UI only on certain events, which can cause a change in the data or the model underneath.

## More directives

There are two directives in AngularJS that deal with hiding and showing HTML elements: ng-show and ng-hide. They inspect a variable and, depending on the truthiness of its value, show or hide elements in the UI, respectively. **AngularJS treats true, nonempty strings, nonzero numbers, and nonnull JS objects as truthy.**

The ng-if removes or recreates a portion of the DOM tree based on an {expression}. If the expression assigned to ngIf evaluates to a false value then the element is removed from the DOM, otherwise a clone of the element is reinserted into the DOM.

The ng-class directive is used to selectively apply and remove CSS classes from elements. It can take strings or objects as values. If it is a **string**, it simply applies the CSS classes directly. If it is an **object**, AngularJS takes a look at each key of the object, and depending on whether the value for that key is true or false, applies or removes the CSS class.

Notice also that ng-bind, ng-show, and most of these directives can directly refer to a variable on the controller or call a function to get the value.

The ng-click directive evaluates any expression passed to it when the button is clicked.

### AngularJS $watch() , $digest() and $apply()

The AngularJS $scope functions $watch(), $digest() and $apply() are some of the central functions in AngularJS. Understanding $watch(), $digest() and $apply() is essential in order to understand AngularJS.

When you create a data binding from somewhere in your view to a variable on the $scope object, AngularJS creates a "watch" internally. A watch means that AngularJS watches changes in the variable on the $scope object. The framework is "watching" the variable. Watches are created using the $scope.$watch() function.

At key points in your application AngularJS calls the $scope.$digest() function. This function iterates through all watches and checks if any of the watched variables have changed. If a watched variable has changed, a corresponding listener function is called. The listener function does whatever work it needs to do, for instance changing an HTML text to reflect the new value of the watched variable. Thus, the $digest() function is what triggers the data binding to update.

Most of the time AngularJS will call the $scope.$watch() and $scope.$digest() functions for you, but in some situations you may have to call them yourself. Therefore it is really good to know how they work. (AngularJS $watch() , $digest() and $apply(), 2015)

The ng-repeat directive is one of the most versatile directives in AngularJS, and can be used for a whole variety of situations and requirements. It can be used to show an array of elements in the HTML or to show all the keys and values of an object, sorted in a case-sensitive, alphabetic order.

Helper Variables in ng-repeat:

* **$first**, **$middle**, and **$last** are Boolean values that tell us whether that particular element is the first, between the first and last, or the last element in the array or object.
* **$index** gives us the index or position of the item in the array.
* **$odd** and **$even** tell us if the item is in an index that is odd or even (we could use this for conditional styling of elements, or other conditions we might have in our application).

## Forms

### One/two-way data-binding

We saw the **ng-bind** directive, or its equivalent double-**curly {{ }}** notation, which allowed us to take the data from our controllers and display it in the UI. That gives us our one-way data-binding.

Forms are a staple of web applications, and AngularJS provides the **ng-model** directive for us to deal with inputs and two-way data-binding.

When we need to update the UI, we just update the model fields in the controller. When we need to get the latest and greatest value, we just grab it from the controller. Again, this is the AngularJS way.

### Working with Forms

A form submit event can be triggered in multiple ways: clicking the Submit button, or hitting Enter on a text field. The **ng-submit** gets triggered on all those events, whereas the **ng-click** will only be triggered when the user clicks the button.

When you use **ng-model**, AngularJS automatically creates the objects and keys necessary in the chain to instantiate a data-binding connection. The first letter typed causes the user object to be created, and the value to be assigned to the correct field in it.

When designing your forms and deciding which fields to bind the ng-model to, you should always consider what format you need the data in.

### Form Validation and States

When you use forms (and **give them names**), AngularJS creates a FormController that holds the current state of the form as well as some helper methods. You can access the FormController for a form using the form’s name.

Things that are exposed as the **state** and kept up to date with data-binding are:

* **$invalid** AngularJS sets this state when any of the validations (required, ng-minlength, and others) mark any of the fields within the form as invalid.
* **$valid** The inverse of the previous state, which states that all the validations in the form are currently evaluating to correct.
* **$pristine** All forms in AngularJS start with this state. This allows you to figure out if a user has started typing in and modifying any of the form elements. Possible usage: disabling the reset button if a form is pristine.
* **$dirty** The inverse of $pristine, which states that the user made some changes (he can revert it, but the $dirty bit is set).
* **$error** This field on the form houses all the individual fields and the errors on each form element. We will talk more about this in the following section.

Each of the states mentioned (except $error) are Booleans and can be used to conditionally hide, show, disable, or enable HTML elements in the UI. As the user types or modifies the form, the values are updated as long as you are leveraging ngmodel and the form name.

Built-in **validations** that AngularJS offers:

* **required** this ensures that the field is required, and the field is marked invalid until it is

filled out.

* **ng-required** Unlike required, which marks a field as always required, the ng-required directive allows us to conditionally mark an input field as required based on a Boolean condition in the controller.
* **ng-minlength** We can set the minimum length of the value in the input field with this directive.
* **ng-maxlength** We can set the maximum length of the value in the input field with this directive.
* **ng-pattern** The validity of an input field can be checked against the regular expression pattern specified as part of this directive.
* **type="email"** Text input with built-in email validation.
* **type="number"** Text input with number validation. Can also have additional attributes for min and max values of the number itself.
* **type="date"** If the browser supports it, shows an HTML datepicker. Otherwise, defaults to a text input. The ngmodel that this binds to will be a date object. This expects the date to be in yyyy-mm-dd format (e.g., 2009-10-24).
* **type="url"** Text input with URL validation.

Need to show the user that a field is required? Then when the user starts typing, show the minimum length, and then finally show a message when he exceeds the maximum length. All these kinds of conditional messages can be shown with the AngularJS validators.

AngularJS takes the name of the validator (number, maxlength, pattern, etc.) and depending on whether or not that particular validator has been satisfied, adds the ng-valid-validator\_name or ng-invalid-validator\_name class, respectively.

### Nested Forms with ng-form

AngularJS provides an **ng-form** directive, which acts similar to form but allows nesting, so that we can accomplish the requirement of grouping related form fields under sections.

Now we can have substate within our form, evaluate quickly if each section is valid, and leverage the same binding and form states that we have looked at so far. A quick highlight of the features:

* A subform using the ng-form directive. We can give this a name to identify and grab the state of the subform.
* The state of the subform can be accessed directly (childForm.$invalid) or through the parent form (myForm.profile.$invalid).
* Individual elements of the form can be accessed as normal (childForm.firstName.$error.required).
* Subforms and nested forms still affect the outer form (the myForm.$invalid can be true because of the use of the required tags).

You could have subforms and groupings that have their own way of checking and deciding validity, and ng-form allows you to model that grouping in your HTML.

## Working with Filters

AngularJS filters are used to process data and format values to present to the user. They are applied on expressions in our HTML, or directly on data in our controllers and services.

* currency - formats a given number as currency with the commas, decimals,and currency symbol added as needed. The filter takes an optional currency symbol as the second argument; if none exists, it takes the default symbol for the current browser.
* number - takes a number and converts it to a human-readable string with comma separation. The number filter also takes an optional decimal size that tells it how many digits to keep after the decimal point.
* lowercase - A very simple string filter that takes any string and converts all the characters to lowercase.
* uppercase - A very simple string filter that takes any string and converts all the characters to uppercase.
* json – is a great tool for debugging, or for any time we need to display the contents of a JSON object or an array in the UI. It takes a JSON object or array (or even primitives) and displays it as a string in the UI.
* date – is a customizable and powerful filter that takes a date object or a long timestamp and displays it as a human-readable string in the UI. It can take a user-defined format or one of the built-in short, medium, or long formats.
* orderBy - allows us to take an array and order it by a predicate expression (or a series of predicate expressons). It also takes a second optional Boolean argument, which decides whether or not the sorted array is reversed.

Any filter, whether built-in or our own, **can be injected** into any service or controller by affixing the word “Filter” at the end of the name of the filter, and asking it to be injected.

## Bibliography

*AngularJS $watch() , $digest() and $apply()*. (2015, 11 13). Retrieved from Jenkov.com: http://tutorials.jenkov.com/angularjs/watch-digest-apply.html

*Cascading Style Sheets*. (2015, 11 6). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Cascading\_Style\_Sheets

*HTML*. (2015, 11 06). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/HTML

*JavaScript*. (2015, 11 6). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/JavaScript

*jQuery*. (2015, 11 6). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/JQuery

Michael S. Mikowski, J. C. (2014). *Single Page Web Applications.* Shelter Island, NY 11964: Manning.

*ngApp*. (2015, 11 13). Retrieved from docs.angularjs.org: https://docs.angularjs.org/api/ng/directive/ngApp

*Single-page application*. (2015, 11 9). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Single-page\_application