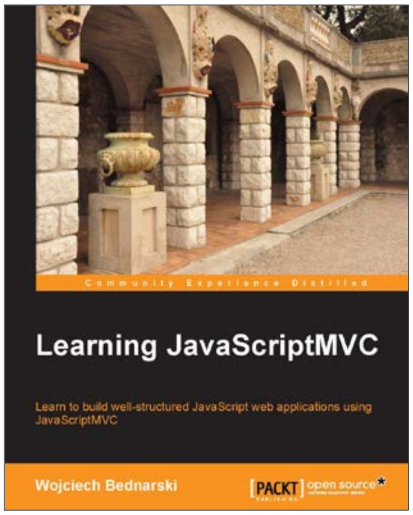
Learning JavaScriptMVC

Wojciech Bednarski


Community Experience Distilled
Learning JavaScriptMVC
Learn to build well-structured JavaScript web applications using JavaScriptMVC
Wojciech Bednarski
PACKT open source

Chapter No. 2 "DocumentJS"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.2 "DocumentJS"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Wojciech Bednarski is a software engineer with expert knowledge of client-side technologies. He is passionate about JavaScript, Node.js, HTML5, Ruby, NoSQL, and POSIX-compliant systems.

While at university he started taking up freelance jobs and was obsessed by web accessibility and usability as well as web standards.

Then, he moved to Warsaw where he started working as a web developer at eo Networks, which is recognized as one of the 50th fastest growing company in Central Europe.

He then started work at Roche, one of the largest pharmaceutical companies in the world, where he worked on large scale web-based systems as well as conducted workshops and technical seminars. He was recognized with an Informatics Service Award in the category of Innovation.

He then moved to Copenhagen and started work at YouSee, the subsidiary of TDC, the biggest Danish telecom company, where he programmed set top boxes. He won Copenhagen Startup Weekend and also began the Everplaces startup.

At the time of writing this book, he is a consultant for a New York-based company working on the next big thing you will use. He works from different places and lives with his beautiful wife and two black cats. He also loves taking pictures, you can have a sneak peek at `www.pixmod.net`. He is also fond of driving sports cars and traveling.

You can visit his professional profile at `www.linkedin.com/in/bednarski/` or you can follow him on `Twitter @wbednarski`.

Learning JavaScriptMVC

Learning JavaScriptMVC will guide you through all the framework aspects and show you how to build small- to mid-size, well-structured and documented client-side applications that you will love working on.

What This Book Covers

Chapter 1, Getting Started with JavaScriptMVC, provides an overview of the JavaScriptMVC framework. Install it, go over the architecture, and learn how to do it in the best possible way—by building a simple application.

Chapter 2, DocumentJS, shows how, despite being powerful, DocumentJS is a simple tool designed to easily create searchable documentation of any JavaScript codebase.

Chapter 3, FuncUnit, explains how FuncUnit is a functional testing framework with jQuery-like syntax. Using FuncUnit, we can run tests in all modern web browsers. Writing test is really easy and fast.

Chapter 4, jQueryMX, shows how jQueryMX is a collection of jQuery libraries that provides the functionality necessary to implement and organize large JavaScript applications. It provides classical inheritance simulation and a model-view-controller layer to provide logically separated codebase.

Chapter 5, StealJS, shows that StealJS is an independent code manager and build tool.

Chapter 6, Building the App, shows how to build real-world applications from concept through design, implementation, documentation, and testing.

2
DocumentJS

Source code alone is insufficient; documentation is an important part of software engineering. **DocumentJS** is a powerful, yet simple tool designed to easily create searchable documentation for any JavaScript codebase.

In this chapter, we will to get an overview of DocumentJS. We will learn how it works and learn to generate its documentation.

The following are the key features of DocumentJS:

- Flexible and easy to extend
- Support Markdown: `http://en.wikipedia.org/wiki/Markdown`
- Integrated documentation viewer with API search
- Works with any JavaScript code and not only with JavaScriptMVC

If you are familiar with JSDoc, YUIDoc, YARD, or similar documentation syntax/ tools in then DocumentJS can be learned a, few minutes.

The documentation for DocumentJS can be found at `http://javascriptmvc.com/docs.html#!DocumentJS`.

[**Markdown** is a text-to-HTML conversion tool that allows you to write using an easy-to-read and easy-to-write plain text format (`http://daringfireball.net/projects/markdown`).]

How does DocumentJS work?

The architecture of DocumentJS is organized around types and tags.

Types represent every relatively independent part of the JavaScript code that we may want to comment, such as classes, functions (methods), or attributes.

Tags provide additional information to types, such as parameters and returns.

DocumentJS parses JavaScript and Markdown files to produce JSONP files that are used by JMVCDoc to render documentation.

Writing the documentation

Let's add a documentation to our `Todo` list application in *Chapter 1, Getting Started with JavaScriptMVC*.

To add the main documentation page, create a Markdown file `todo.md` in the `Todo/todo` directory with the following content:

```
@page index TodoApp
@description TodoApp is simple todo application.

# TodoApp documentation

Here we can add some more documentation formatted by [Markdown][1]!

[1]: http://daringfireball.net/projects/markdown/syntax "Check out
Markdown syntax"
```

Then, add these documentation blocks to the `todo.js` file:

```
steal(
    'jquery/class',
    'jquery/model',
    'jquery/util/fixture',
    'jquery/view/ejs',
    'jquery/controller',
    'jquery/controller/route',
    function ($) {

        /**
         * @class Todo
         * @parent index
         * @constructor
         * @author Wojciech Bednarski
         * Creates a new todo.
         */
        $.Model('Todo',{

                /**
                 * @function findAll
                 * Get all todos
                 * @return {Array} an array contains objects with all
todos
                 */
                findAll: 'GET /todos',

                /**
                 * @function findOne
                 * Get todo by id
                 * @return {Object} an objects contains single todo
                 */
                findOne: 'GET /todos/{id}',

                /**
                 * @function create
                 * Create todo
                 * @param {Object} todo
                 * Todo object
                 * @codestart
                 * {name: 'read a book by Alfred Szklarski'}
                 * @codeend
                 *
                 * @return {Object} an object contains newly created
todo
                 * @codestart
                 * {
                 *     id:   577,
                 *     name: 'read a book by Alfred Szklarski'
                 * }
                 * @codeend
                 *
                 * ### Example:
                 * @codestart
                 * var todo = new Todo({name: 'read a book by Alfred
Szklarski'});
                 * todo.save(function (todo) {
                 *     console.log(todo);
                 * });
                 * @codeend
                 */
                create:  'POST /todos',

                /**
                 * @function update
                 * Update todo by id
```

```
             * @return {Object} an object contains updated todo
             */
            update:  'PUT /todos/{id}',

            /**
             * @function destroy
             * Destroy todo by id
             * @return {Object} an object contains destroyed todo
             */
            destroy: 'DELETE /todos/{id}'
        },
        {

        }
    );

    // Fixtures
    (function () {
        var TODOS = [
            // list of todos
            {
                id:   1,
                name: 'read The Good Parts'
            },
            {
                id:   2,
                name: 'read Pro Git'
            },
            {
                id:   3,
                name: 'read Programming Ruby'
            }
        ];

        // findAll
        $.fixture('GET /todos', function () {
            return [TODOS];
        });

        // findOne
        $.fixture('GET /todos/{id}', function (orig) {
            return TODOS[(+orig.data.id) - 1];
        });

        // create
        var id = 4;
        $.fixture('POST /todos', function () {
            return {
                id: (id++)
            };
            });

            // update
            $.fixture('PUT /todos/{id}', function () {
                return {};
            });

            // destroy
            $.fixture('DELETE /todos/{id}', function () {
                return {};
            });
        }());

        /**
         * @class Todos
         * Creates a new Todos controller
         * @parent index
         * @constructor
         * @param {String} DOMElement DOM element
         * @return {Object}
         */
        $.Controller('Todos', {
            // init method is called when new instance is created
            'init': function (element, options) {
                this.element.html('todos.ejs', Todo.findAll());
            },

            // add event listener to strong element on click
            'li strong click': function (el, e) {
                // trigger custom event
                el.trigger('selected', el.closest('li').model());

                // log current model to the console
                console.log('li strong click', el.closest('.todo').
model());
            },

            // add event listener to em element on click
            'li .destroy click': function (el, e) {
                // call destroy on the model to prevent memory leaking
                el.closest('.todo').model().destroy();
            },

            // add event listener to Todo model on destroyed
            '{Todo} destroyed': function (Todo, e, destroyedTodo) {
                // remove element from the DOM tree
                destroyedTodo.elements(this.element).remove();

                console.log('destroyed: ', destroyedTodo);
            }
        });

        /**
         * @class Routing
         * Creates application router
         * @parent index
         * @constructor
         * @param {String} DOMElement DOM element
         * @return {Object}
         */
        $.Controller('Routing', {
            init: function () {
                new Todos('#todos');
            },

            // the index page
            'route': function () {
                console.log('default route');
            },

            // handle URL witch hash
            ':id route': function (data) {
                Todo.findOne(data, $.proxy(function (todo) {
                    // increase font size for current todo item
                    todo.elements(this.element).animate({fontSize:
'125%'}, 750);
                }, this));
            },

            // add event listener on selected
            '.todo selected':  function (el, e, todo) {
                // pass todo id as a parameter to the router
                $.route.attr('id', todo.id);
            }
        });

        // create new Routing controller instance
        new Routing(document.body);
    }
);
```

Type directives

Type directives represent JavaScript constructs that you may want to document:

- `@page`: This adds a standalone page
- `@attribute`: These are the document values on an object

- `@function`: These are document functions
- `@class`: This documents a class
- `@prototype`: This is added to the previous class or a constructor's prototype functions
- `@static`: This is added to the previous class or constructor's static functions
- `@add`: This adds the docs to a class or constructor described in another file

Tag directives

Tag directives provide additional information to the comments:

- `@alias`: This specifies other commonly used names for class or constructor
- `@author`: This specifies the author of a class
- `@codestart`: This specifies the start of a code block
- `@codeend`: This specifies end of a code block
- `@constructor`: This documents a contractor function and its parameters
- `@demo`: This is the placeholder for an application demo
- `@description`: This is used to add a short description
- `@download`: This is used to adds download link
- `@iframe`: This is used to add an iframe with example code
- `@hide`: This hides the class view
- `@inherits`: This specifies what the Class or Constructor inherits
- `@parent`: This specifies under which parent the current type should be located
- `@param`: This specifies a function parameter
- `@plugin`: This specifies a plugin by which an object gets stolen
- `@return`: This specifies what a function returns
- `@scope`: This forces the current type to start the scope
- `@tag`: This specifies the tags for searching
- `@test`: This specifies the links for test cases
- `@type`: This sets the type for the current commented code
- `@image`: This adds an image

Generating the documentation

All we need to do to generate the documentation is run the `doc` command from the command line (inside the `Todo` directory):

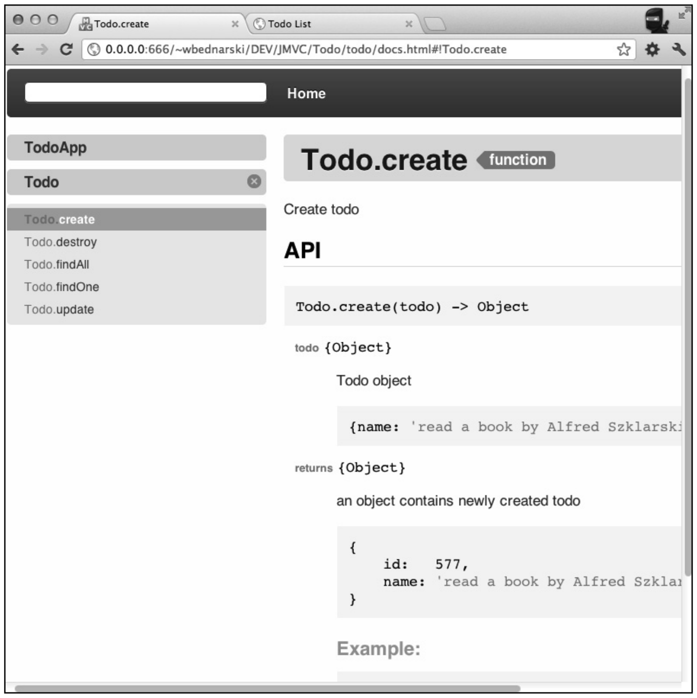
```
$ ./documentjs/doc todo
PROCESSING SCRIPTS


  todo/todo.js
  todo/todo.md


GENERATING DOCS -> todo/docs


Using default page layout.  Overwrite by creating: todo/summary.ejs
```

This generates the documentation, which we can browse by opening `docs.html` located in the `Todo/todo` directory:



Todo.create function

Create todo

API

```
Todo.create(todo) -> Object
```

todo {Object}

Todo object

```
{name: 'read a book by Alfred Szklarski
```

returns {Object}

an object contains newly created todo

```
{
    id:   577,
    name: 'read a book by Alfred Szkla
}
```

Example:

We can customize the look and feel of the documentation by changing the `summary.ejs` template file. Simply copy the template from `documentjs/jmvcdoc` to `Todo/todo` and modify it.

Summary

In this chapter, we have learned what it DocumentJS is and how to write and generate its documentation.

One good habit that every programmer should have is that he or she must document the codebase and keep it up to date.