Documentation in Javascript

Relevant Links

- JSDoc[1]
- documentationjs[2]
- ESDoc[3]
- Docco[4]
- Lots of links to various documentation tools[5]

Notes

It is invariably useful and important to provide documentation for your code, but in a way that is usable by the application's users. This is typically done by specifying certain comments in code that are automatically processed to produce a documentation webpage.

There are fundamentally a couple of different kinds of documentation:

- Separate documents that describe how to use a software (user's guides, examples)
- Documentation of the API of an application/library, for the users of that library. This is what we will focus on.
- Inline comments to clarify some trickier parts of the code.

There are at least two very different approaches to automatic code documentation:

- Generating a single HTML document with comments interspersed with the corresponding code. These tools tend to process all comments. You can see how this looks here[6]. Docco[7] is a prime examples of this system, often called *literate programming*.
- Reading only specific comments, and producing an set of webpages from them. This is the standard documentation system, similar to JavaDoc. Standard examples of this in Javascript are JSDoc[8], documentationjs](http://documentation.js.org/) and ESDoc[9].

We will be seeing documentationjs in this section, but other systems are similar.

In any case, most of these systems allow you to customize the CSS for your pages, and most allow you to use Markdown syntax in your comments.

[1] http://usejsdoc.org/
[2] http://documentation.js.org/
[3] https://esdoc.org/
[4] https://jashkenas.github.io/docco/
[5] https://github.com/documentationjs/documentation/wiki/See-also
[6] http://underscorejs.org/docs/underscore.html
[7] https://jashkenas.github.io/docco/
[8] http://usejsdoc.org/
[9] https://esdoc.org/

JSDoc comments

JSDoc comments are special comments that processed by JSDoc and other systems:

- They are indicated by an initial `/**`, with two stars.
- They end as usual with `*/`.
- The content consists of a documentation comment, which will be processed via Markdown
- The comment ends with a series of "tags", characterized by an initial `@` sign.

Here is an example of the documentation for the TaskAppController's constructor:

```
/**
 * Create a new `TaskListController`.
 *
 * @param {TaskList} taskList     The `TaskList` instance to manage.
 * @param {jQuery}   domEl        The DOM Element to use for printing the list.
 * @returns {TaskListController} A new `TaskListController` instance.
 * @example TaskListController.new(myList, $('#list'));
 * @memberof TaskListController
 * @static
 * @public
 */
```

This creates a classification of sorts for our comments, into 3 types:

```
/**
 * "slash-star-star" comments are meant to be processed by automatic tools.
 * These tools might generate documentation, or look for linting instructions.
 */
/*
 * "slash-star" comments are used for large comments that describe some
 * feature of a file but that are not meant to be automatically processed.
 */
// Inline comments are meant for clarifying code snippets, for developers
// working on the code.
```

JSDoc tags You can find what all the available tags are in JSDoc's documentation[10]. But some standard ones are:

@param an input given to the function as a parameter, along with its expected type.
@returns what the output value of the function is.
@name explicitly set the documented name of the item, if the automatically detected name is not suitable.
@private document code bo do not include it in the generated documentation. There's also @public and @protected. You can choose to generate some of this
@example inline code examples
@static, @instance indicate if it is a static/class-level function or variable an instance/prototype-level function or variable.

[10] http://usejsdoc.org/index.html

@kind indicate what kind of an item it is (function etc). Can also use more direct tags instead, like @module, @class, @method etc

@event can be used for documenting events.

Running the documentation

You can generate various “results” from this documentation. That all depends on the program you use. For instance with documentationjs you can:

- Create a directory of HTML files, or single file, from your comments (build/serve).
- Create a markdown document from your comments, which can be used in GitHub for example.
- Add a documentation section to a README.md file.
- “lint” your comments looking for comment-syntax errors.
- Set up the documentation page to automatically update as you add comments (serve/watch).
- Add your own customizations that interact with the documentation tool and modify some parts of the process.

The simplest thing to do would be something like this:

```
documentation build js/taskListController.js -f html -o docs
# This will open the file
documentation build --watch js/taskListController.js -f html -o docs
```