Contents

- Get up and running
- General coding
  - Local server and livereload
  - Boilerplate auto-generation
- A/B Testing
  - Landing page test
  - Controls test
- Gulp and the build process
  - Gulp tasks
  - Paths
- Application flow and details
  - Vendor libraries
- App Engine deployment
- Documentation style
- Further notes

1. Get up and running

In commandline:

Ensure nodejs is installed.

Run: `npm install`. This will install all nodejs modules, and automatically run the `bower` install process.

Run: `gulp dev` to begin the dev environment.

2. General coding

2.1. Local server and livereload

For development, a local server can be started (`gulp dev` or just `gulp`). This will serve the site from localhost:9001. Code changes in the `/src/` and `vendor` folders will refresh the site via livereload.

2.2. Boilerplate auto-generation

2.2.1. Wrapped RequireJS

All code in `src` will be wrapped with code defined in `/gulp/tasks/js`.

The code will start with:

```
define(function(require) { "use strict";
// file contents...
```

and end with:

```
// ... file contents
});
```

This means that there is no need to fill in the Requirejs wrapping function for each file.

Imports are handled "variable" style, e.g. this will import `/src/js/view/footer-view.js`:

```
var FooterView = require('view/footer-view');
```

2.2.2. Default imports

There are a number of imports that are used regularly and these have been placed into a separate file that can be "baked" into the source code using the pre-compilation directive `__DEFAULT_IMPORTS__` at the top of the file.

These imports are listed in `/gulp/replace/js-default-imports.js`. All variable names listed there can be used directly in the source (they are not global variables, they are local to that file).

3. A/B Testing

There are 2 tests running at the moment. One adjusts elements on the landing screen (or splash page), the other adjust the controls once the site starts. There is a 50% chance of the audience seeing either result from each test.

To enforce a particular test, change the following lines in `src/app.js`:

```
    setupABTesting: function () {
        // ... code
        AppMode.TEST_SPLASH = parseInt(this.getQueryVariable('test-splash'))
        AppMode.TEST_BOOK = parseInt(this.getQueryVariable('test-book'));
```

to:

```
    setupABTesting: function () {
        // ... code
        AppMode.TEST_SPLASH = 1;
        AppMode.TEST_BOOK = 0;
```

where the test A=0 and B=1.

3.1. Landing page test

Test A: "Start Experience" and "Book a Test Drive" buttons are shown.

Test B: "Click to Start" text and an arrow are shown.

3.2. Controls test

Test A: "Book a test drive" text button is shown.

Test B: "Twitter" and "Book form" icon buttons are shown.

4. Gulp and the build process

4.1. Gulp tasks

There is a Gulp workflow designed to help speed up development. All Gulp tasks reside in `/gulp/tasks/`. The file names match the task names, and production tasks are in the same files with ":prod" appended to the task name.

New tasks can be added as new files into `/gulp/tasks/` and `/gulp/index.js` will automatically retrieve them.

4.2. Paths

The paths to the various parts of the codebase are listed in `/gulp/paths.js` using the gulp blob format.

5. Application flow and details

5.1. Vendor libraries

5.1.1. Bower

Bower is used to manage the JS runtime dependancies.

All the vendor files are stored in `/vendor/`. This is defined in `/.bowerrc` if it needs to be changed.

New vendor libraries can be added using `bower install --save-dev <repo|name>`

There is some awkwardness on occasion as Bower doesn't have strict enough specifications. The impact is that `bower.json` and `.bower.json` files don't always list the file(s) you can use in your application.

The vendor libraries are automatically pulled from here using `gulp/util/get-dependencies.js`. Any non-bower dependencies can be added in `gulp/vendor-extra.js`. It uses a hashmap for the extra files, but the key is irrelevant, only the value matters. The key is just for reference if you want to use it.

6. App Engine deployment

Google App Engine is used to serve the live site. Grab the installer and make sure you are added to the `audiodrive3d` project on App Engine. Adrien has access to this and can add anyone necessary.

Point App Engine at `appengine/app.yaml`.

Prepare a deployment by running `gulp deploy`. This will create a new version in the `deploy` folder and also in the `appengine/static/` folder. It will be served on localhost:9002 if you need to test locally. It isn't watched like development, though, so if you make any changes you will need to re-run the `gulp deploy` command.

7. Documentation style

The comments used within the source follow specific standards.

Any code, variables, etc will be mentioned within `backticks` Longer snippets of code will be within

~~~

three tildes

~~~

Parameters listed as @param, followed by variable type in braces, the variable name, and then description like this:

```
@param {type} variableName        Some description of the variable
```

The `DocBlockr` SublimeText plugin is used to generate documentation.

Valid types are:

- string
- int
- object
- float
- boolean

If a variable is optional, it will be surrounded with square braces:

```
@param {type} [variableName]    Some description of the variable
```

If a variable is optional and has a default value, it will be surrounded with square braces and the default value shown in the braces after an equals sign:

```
@param {type} [variableName='default value']    Some description of the variable
```

If a variable is an array, the type if object will be shown with open and closing square braces as a suffix of the type:

```
@param {type[]} variableName    Some description of the variable
```

If a variable can have multiple types, they will be separated with a bar (|):

```
@param {type1|type2} variableName    Some description of the variable
```

If a variable is mutable (the original is altered by the function/value passed by reference) then it will be marked like this:

```
@param {type} *variableName    Some description of the variable
```

8. Further notes

This system as written is overly complex in many places as the technical specifications were changed dramatically several times during the project life-cycle.

I have cleaned up and simplified things where I had time to do so.