# Wealth

jiayihu

# Table of Contents

# Wealth Project Docs

This is an overview of the project structure. Also when we talk about *compiling* we mean running our source files through Gulp.js, which optimises and generates the code used by browsers. Gulp is an automation tool which improves the workflow and improves the code. For example we have a task to add vendor prefixes in CSS, so we don't need to add them ourselves in source files.

> Keep in mind that files in `public` folder should never be touched since they are meant to be used. In fact, any time I recompile they are deleted and/or overwritten and they may be minified when we generate production-ready code.

The following structure shows how we organize files. The `app` directory contains the source files, organized by type, whereas `public` is the folder of the files which will be eventually loaded on the server.

```
├── app
│   ├── fonts
│   ├── html
│   │   ├── includes
│   │   ├── screens
│   │   └── templates
│   ├── images
│   ├── javascripts
│   │   ├── components
│   │   ├── controllers
│   │   ├── model
│   │   └── views
│   ├── static
│   │   ├── scripts
│   │   │   └── lib
│   │   └── styles
│   │       └── components
│   └── stylesheets
│       ├── base
│       ├── bootstrap
│       ├── components
│       ├── layout
│       └── steps
├── gulpfile.js
│   └── tasks
├── public
│   ├── fonts
│   ├── images
│   ├── scripts
│   │   └── lib
│   └── styles
│       ├── components
└── test
    ├── functional
    └── unit
```

To be able to develop you must install the npm dependencies with `npm install` and run `npm run gulp`. Gulp will compile source files, move them to `public` folder, open a browser tab to see the application and watch for changes in order to re-compile files.

There is also the command `npm run production` to generate compiled files ready for production, which means they are optimized and minified. Explaining the details of each Gulp task is out of the scope of this documentation.

# app/html

This folder contains the source `html` files. We use Nunjucks template system to reduce the repetition of code and allow use to divide the whole html into different partials. `index.html` includes the navigation menu from `/includes/nav.html` and steps from `/screens/` .

Each screen is build using the template in `templates/step.html` because they all share some come markup like title, content and *continue* button. Check Nunjucks documentation for info about the syntax used.

> NOTE: Despite the .index extension these are not plain html files and cannot be used in browser. Gulp detects changes to the files and compiles them to a final index.html file with all the includes.

# app/stylesheets

We use SCSS for our stylesheets. Please read the documentation for advantages of pre-processing when dealing with CSS.

We have two top-level files, which are `main.scss` and `bootstrap.scss`. The former imports our custom styles whereas the latter imports bootstrap partials. Note that bootstrap partials should not be modified since it will be difficult in future to update it with a new version. Bootstrap allows custom variables to be used for colors, borders, font size etc.

# app/javascripts

> NOTE: We use browserify which allows the usage of Node.js modules, which is a feature missing in Javascript ES5 (the version we use and currently supported by all the browsers more recent than 2009). `require` and `export` allows to require modules and export values, functions etc. from a module to be used by others.

We use MVP Design Pattern and our views are completely *dumb*, which means that they cannot know about the model. If they need data from model, like for the first render, the information must be given by the relative *controller*.

At the root of this directory we have `app.js`, which initializes the application loading model, controllers and views.

Then there is `model.js` which contains functions to read/write to the Store (alias `localStorage` in our case) and functions strictly related to manipulating application data. `model` folder contains the available actions, goals and budget categories, in order to avoid having a huge unique `model.js` file.

`controller.js` and `view.js` load each controller and relative view. Each controller has access to the model and it's responsible for view render with proper data. It also updates the model when user interacts with the view, so it defines what happens when there is an user interaction. Besides it *subscribe* for model changes and updates the view if some relevant change has happened. We use here the Pub/Sub Design Pattern.

# app/static

This folder contains files which are unlikely to change like favicon, web.config and libraries. We don't include libraries .css files in `/stylesheets` because they would be detected and compiled by Gulp, which is not necessary since they are already minified.

Some .js files instead are in static because they are libraries which are not compatible with `browserify` or any other module pattern like Require.js or UMD. Therefore we load them as separate externals.

In `public` styles and scripts are in the same folder of those compiled by Gulp.