Congratulations!

Seeing this means you have installed Backbone Boilerplate correctly.

Now that you have the easiest and most-powerful Backbone boilerplate available, you're probably wondering how to use it to start building applications...

Contents

- Useful Resources

Overview

Backbone Boilerplate is the product of much research and frustration. While existing boilerplates for Backbone exist, they will often modify the Backbone core, don't have an integrated build system, or impose too much on your application's structure. This boilerplate attempts to improve that. Organize your application in a logical filesystem, and develop Models, Collections, Views, and Routers inside modules. Build your application knowing you have efficient, compact code. Backbone Boilerplate extends on the versatile Backbone core, and helps developers manage their application.

Core Features

- HTML5 Boilerplate (https://github.com/h5bp/html5-boilerplate) included.
- Managed filesystem structure for application code, assets, tests, and distribution.
- Snippets to make common tasks easier: modules, HTML5 History API/Hash navigation, template loading and application events.
- Flexible and extendable build system.
    - Concatenate and minify all your libraries, application code, templates and CSS down to reduce transmission time.
    - Compile underscore templates to prevent pre-processing on the client.

Getting help

If you're encountering issues, need assistance, or have a question that hasn't been answered in this tutorial or the GitHub project page (https://github.com/tbranyen/backbone-boilerplate) you may find help in one of these places:

- IRC - #documentcloud on irc.freenode.net
- GitHub Issues (http://github.com/tbranyen/backbone-boilerplate/issues) - Please report if you've found an issue, bug, or controversial request.

I want this project to be the best it possibly can and represent the interests of the community, **please** submit issues with features you find useful and anything that you question.

Writing your application

Your application may be made up of third-party libraries, plugins, application code, templates, and lots

of logic. All of this will need to be well structured to keep it maintainable and it also needs to be compiled if deployed into production. Before you can get started you will need to clean out all the existing defaults that are in the boilerplate are necessary to display this tutorial.

Strongly recommend you read through this tutorial before cleaning out any files that may hold clues on how to use the Boilerplate.

Cleaning out default files and code

There are several places where customization may be required.

- **Removing the Git history**

  If you cloned the Backbone Boilerplate with Git, you should delete the git directory and then initialize your own Git history:

  ```
  $ rm -rf .git
  $ git init
  ```

- **Removing the test directory**

  If you are not planning on testing your application with QUnit you should delete this directory.

- **Removing the build process**

  If you are not planning on using the provided build tool, delete the `build` folder. It contains a lot of unnecessary code and Node.js modules that you will not need. You should also clear out the commented out script tags inside of `index.html`:

```
<!--
     If using the build tool you can uncomment the following lines and use
     these instead.  They will toggle based on if you are using debug or
     release.
-->

<!--
<script src="/assets/js/libs.js"></script>
<script src="/assets/js/templates.js"></script>
<script src="/assets/js/app.js"></script>
-->
```

- **Changing the Favicon**

  At the root level of the project simply change the `favicon.ico` file to point to your own branded icon.

- **Removing default application code**

  This tutorial is rendered in the `app/modules/example.js` file and written in `app/templates/example.html`. Both of these files are safe to remove.

- **Removing the default routes**

  Routes are defined in the `app/index.js` file. Familiarize yourself with it's contents. You'll notice the default router has two existing routes and callback defined, reset it to:

```
  // Defining the application router, you can attach sub routers here.
  var Router = Backbone.Router.extend({
    routes: {
      "": "index",
    },

    index: function() {
      // Put your homepage route logic here
    }
  });
```

Above the Router definition you'll see a reference to the example module, this is safe to delete as well.

```
  // Include the example module
  var Example = namespace.module("example");
```

- **Removing default assets**

  The default styles for this tutorial are stored in `assets/css/style.css`. You will probably want to remove these since they only make sense for this specific page. They start on `Line 209`. With the following H5BP header:

```
  /* ==|== primary styles ===================================================
     Author: Backbone Boilerplate
     ========================================================================
*/
```

You may also want to change the name to yours, if you're planning on putting your custom CSS here as well.

You should delete the `assets/img/backbone.png` file if you are not planning on using it in your app.

Setting your namespace

This is a very important starting step to creating your application. This brands the application to your name and makes it something identifiable to new developers. To set your own namespace, simply open the `app/namespace.js` file and make the following modifications: First change the name of the namespace, this is defined on `Line 4`

```
this.myapp = {
  // Assist with code organization, by breaking up logical components of code
  // into modules.
  module: function() {
```

Once you've set the name, you'll want to change the reference in `index.js` on `Line 7` to:

```
// Shorthand the application namespace
var app = this.myapp;
```

Creating a module

Following the Bocoup post on Organizing Your Backbone.js Application With Modules (http://weblog.bocoup.com/organizing-your-backbone-js-application-with-modules) this boilerplate provides the same module definition structure. Modules are placed in the `app/modules/` directory. There is an example module there named: `example.js`. The actual module definition function is located inside the `app/index.js` file. You create and reference modules with the same function call: `namespace.module("<module_name>")`. Typically a module contains a single Model/Collection/Router and many Views. Therefore the returned module object is empty except for a Views object property that can be used to attach many Views to, like:

```
MyModule.Views.Detailed = Backbone.View.extend({ /* ... */ });

MyModule.Views.Main = Backbone.View.extend({ /* ... */ });
```

Attaching Models/Collections/Routers happen on the same level of the module, like so:

```
MyModule.Model = Backbone.Model.extend({ /* ... */ });

MyModule.Router = Backbone.Router.extend({ /* ... */ });
```

Working with templates

Templates are a super useful way to separate concerns in your application. Instead of generating markup from inside your JavaScript application, you instead create it in a separate file and load it into your application. There are numerous ways of loading in a template, but this boilerplate has chosen the most performant way to build all your templates into a single file.

This tutorial itself is a template that exists in `app/templates/example.html`. You can edit this file and hit refresh in here to see the changes. The boilerplate comes with a built in function to handle the loading of templates. It's called:

```
namespace.fetchTemplate("app/templates/name.html", function(template) {
  // Template here is a function, that accepts an object.  Identical to _.template.
  console.log(template({ ... }));
});
```

By defining a custom function this will ensure that if you use the build tool or AJAX, that your templates will load consistently. You can see it in action inside the `app/modules/example.js` module.

If you use the build process to compile your templates, it will automatically find all the HTML files inside the templates directory and compile them into a templates.js file. These are actual JavaScript template functions being compiled on the server, which is different from Jammit and most other server-side builders that just invoke functions on page load.

You can access a compiled template like so:

```
var template = window.JST["app/modules/example.html"];
template({ ... });
```

Working with libraries and plugins

Libraries and plugins are easily added to the application, by placing them inside the `assets/js/libs/` directory. If you have many plugins in your application, it may make sense to create a separate folder such as `assets/js/plugins/` for them.

Using the build tool

The Backbone Boilerplate build process is a state-of-the-art task driven Node.js application that utilizes @cowboy's grunt project. To run the defaults, execute the following command from the project root, and *not from inside the build folder*.

Running with the defaults

To run the defaults, execute the following command from the project root, and *not from inside the build folder*.

```
node build
```

This will do a number of things for you. First it will concatenate all your libs, app code, and templates into separate files inside the `dist/debug` folder. It will then minify those files and your CSS into production ready files inside the `dist/release` folder.

Customizing the build configuration

To customize and configure the build tool, open `build/config.js` and tweak the settings.

Using the development server

While writing an application that leverages `pushState` you can run the following command to run a server that will always resolve to the `index.html`

```
node build/server
```

This will spawn up an HTTP server on port `8000`. This server is intended for development and not production. You should use url rewriting or forwarding all requests in your production server to achieve this same effect.

Serving the built assets

If you are using the build tool in conjunction with this development server you can optionally update the `index.html` file to remove the existing script tags and uncomment out the scripts tag at the bottom to load the `dist/debug` or `dist/release` assets. You can achieve this by specifying either **debug** or **release** after the server command, like so:

```
node build/server release
```

Adding new tasks

To add a new task into the build system, you simply copy and paste the task JavaScript folder/file into the `build/tasks` folder or extract the task archive into the same directory. At the very least in order to run this task, you'll need to add it to the `build/config.js` file. The last line should look something like:

```
  task.registerTask("default", "clean lint:files concat jst min mincss
new_module_here");
```

It's possible the custom task will have additional setup instructions, so make sure you read the README for any task.

Useful resources

- Backbone documentation (http://backbonejs.org/) - Framework on which Backbone Boilerplate is built.
- Underscore documentation (http://documentcloud.github.com/underscore/docs/underscore.html) - Required dependency for Backbone.