Boxtent is a frontend workflow that gives you the infrastructure to build your next website following modern web development best practices to deliver a state of the art product to your users.

With the help of Grunt and Bower, Boxtent automates your workflow and takes care of assets optimization, concatenation, minification and much more. Boxtent includes support for:

* Sass (.scss) + Compass compilation
* Images optimization and compression
* Js concatenation
* Html, css and js minification
* Css automatic autoprefixing
* Critical above the fold css smart management
* Browser auto-reloading and live syncing across devices
* Html includes

Boxtent is perfect for crafting small and simple html, css and js performant website (like this one) but can be easily extended to suit more complex needs as well, leaving you the time to focus on what really matters: your code.

In a world of complex frontend workflows and frameworks, BoxTent tries to cover all the basic needs of frontend developing, without over complicating your life. Not all websites need super complex setups and languages, but all websites deserve to be created with performance in mind, using modern tools and with an infrastructure that grants speed and productivity.

**Installation**

*Requirements*

Boxtent is based on Grunt for tasks automation, Bower for javascript libraries management and Git. Get ready by installing the following items before continuing:

* Node.js: use the installer provided on the NodeJS website
* Grunt: controlla sul sito
* Bower: controlla sul sito
* Git: controlla sul sito

You can now clone or download the Boxtent repository on your machine and start to install all default dependencies and gems to have it working.

*Dependencies*

GEMS

To support Sass and Compass, you must have their gems installed on your machine. Type these in the terminal anywhere:

Gem install sass

Gem install compass

DEPENDENCIES

To install the default BoxTent dependencies, cd to the root of the cloned repository and type

Npm install –save-dev

To install the grunt plugins, and then type

Bower install –save-dev

To install jquery. Boxtent comes with jquery.1.9.1 by default.

RUN BOXTENT

Now that you have it all installed, cd to the root of the project and type grunt, browse to localhost:3000 and.. start coding! It is suggested to open two terminal windows (with GIT support) while working: one will be busy watching for changes to your codebase and live reloading the browser for you, the other will be useful for downloading assets via bower on the go, to perform manual grunt tasks or to push your commits online.

**Folder Structure**

Boxtent is based on three levels:

* \_src folder: this is your source code, this is where you work
* \_dev: is the code you see on localhost:3000.
* \_site: the final website to upload in production

You always work on \_src and never edit anything on \_dev or \_site.   
  
\_dev is the result of low-level automations such as sass compilation, autoprefixing, browser reloading and generally tasks suited for development. At this stage, your website will be easy to debug.

The \_site folder instead, will be fully optimized with performance in mind. This is the code that will go live for your users.

If you don’t like the naming of the folders, you can change it by editing the path variables in the gruntfile.js:

codice

**The development stage**

While working on the source folder, there is a series of automations performed to help you build your website in a fast and smart way that suits development and debugging.

*Css*

.scss files are compiled and autoprefixed in \_dev/css/main.css. Sass compilation comes with Compass support but if you don’t need it, you can remove the @import “compass” from your main.scss file and from the gruntfile. In case you want to use other sass plugins, just install the gems and update your gruntfile accordingly. For example, if you decide to use susy as your grid system, in the terminal type gem install susy, @import “susy” on the top of the main.scss file edit your gruntfile like this:

Codice

Autoprefixing is dynamic and configurable in the gruntfile. To read more about configuration options click here.

*Critical css*

Boxtent was created with performance in mind, and automatic above the fold css extraction and embedding is one of BoxTent features. Critical css is page-based, so it is created every time you save an .html file and every file is named after its related page automatically. It is then updated every time you edit your scss files. The extracted css must be included on each page’s <head> wrapped around special comments like this:

Codice

Paths of selectors such as @font-face or background-image:url() are automatically transformed in relative paths. If your website will not be hosted on a first level domain you will have 404 errors on the console because of wrong paths to these resources. To quickly solve this problem, uncomment codice in the gruntfile. This will exclude anything linking to assets in the above the fold css.

The css will be inlined only when \_site is produced during the build stage. Notice that critical css is coupled with the main css file being appended and loaded asynchronously (this is why you can’t find a link to it anywhere in <head>) with a javascript function that looks like this:

Codice

Read more about it here.

*Fonts*

If you are using custom fonts, create a folder named fonts in the root of \_src, then open the terminal and type grunt copy:the\_fonts to move them in \_dev. If you are using google fonts instead, you should load them asynchronously placing a function like this one in <head>

Codice

*html*

During development, .html files are simply moved in \_dev thanks to the processhtml:dev task processing includes. You can find some default includes in the \_includes folder. The workflow breaks if you don’t use includes, so you must use at least one. Using includes is highly suggested as it unclutters the main html files.

*Images*

Save your images assets in the \_src/images folder. They will be optimized and compressed in \_dev/images/. Supported file formats include .gif, .jpg, .png and .svg.

*Favicons*

Favicons are important, and being compatible with all different types of browsers and platforms has become a difficult task (read more about how crazy this has become here). I highly suggest using url to automatically create all the needed favicons formats for maximum compatibility. Create a folder named favicons in \_src/images and drop them there (be sure grunt is running and watching): imagemin will move most of the files in \_dev automatically. Not all files are image formats though, so to move the rest of the files as well, type grunt copy:favicons on your terminal. Don’t forget to paste the generated code provided by url in <head>. Be sure to check the paths and you’re good to go.

*Javascript*

Javascript libraries and plugins are managed via bower, so if you need any, please install them with bower. Most of the popular plugins are indexed and searchable in the bower search page. If you can’t find it or if you are unsure of its name, you can look for a reference in the plugin’s website. Installing a library with bower is easily done by typing bower install name-of-the-library –save-dev in the terminal: this will download the repository of the library in the bower\_components folder and will automatically update the bower.json file with the name and version of the dependency.

When the library is installed, browse to the unminified file of the .js library in the bower\_components folder and update the copy:bower and the concat:dist tasks in the gruntfile with the right path to the library.

Also include the script in the \_includes/scripts.html file, wrapping them in processhtml special comments. All the scripts will be concatenated and minified in a single production.min.js file during the build stage.

If the library you need is not installable with bower, save the file in \_src/js/custom and be sure that grunt is running so the watch task will copy the file in \_dev. Even in this case, don’t forget to update the copy:bower and the concat:dist tasks in the gruntfile with the path to the library and the \_includes/scripts.html file as well.

To write custom javascript, use \_src/js/custom/main.js.

**The build stage**

When you decide to push your code to production, type grunt build in the terminal to build the \_site package. There’s an additional layer of automations that will transform your development codebase in a more performant one suited for productions. Here is what happens when you type grunt build

*Clean*

The \_site folder is completely wiped out, ready to be rebuilt.

*Delete Sync*

The delete\_sync task checks for unmatched files between \_dev and \_src. If extra files are present, they are deleted to keep both folders clean and in sync. This is a dangerous task as this may delete important files, remove this task or edit it if needed. This applies only for html and images files.

*Css*

Main.css is minified and aggressively optimized by cssnano, a postcss plugin.

*Critical Css*

Critical css is automatically inlined in <head>

*Html*

All .html files are minified.

*Images*

The already optimized images folder is automatically copied from \_dev

*Javascript*

All of your javascript assets are concatenated in a single file called production.js, which is then minified becoming production.min.js. All the single scripts lines you previously had in \_scripts.html are automatically transformed in a single script path that points to the minified and concatenated file.

**NODE PACKAGES used**

The grunt node packages used to accomplish automations are the following. You can go directly to their public repos to know more about them and for more configuration options or troubleshooting.

* Grunt contrib sass
* grunt postcss: autoprefixer, cssnano
* grunt browsersync
* grunt delete sync
* grunt ctrib imagemin
* blabla