# Organización de Carpetas

1)

If you're doing heavy client side, you are probably going the MVC way.

So I'll answer your questions with the approach taken by the [brunch](http://brunchwithcoffee.com/). Brunch projects use MVC library Backbone.js, and have strict directory structure.

How should JavaScript files be organized in the file system **during development?**

src/

app/

collections/

controllers/

models/

styles/

templates/

views/

vendor/

build/

web/

config.yaml

Use [Stitch](https://github.com/sstephenson/stitch) to organize your files as CommonJS modules. Then you will be able to use require() to define dependency between them, as well as to combine them into one file later.

How should the JavaScript parts be **separated from the HTML** and other parts of the application?

build directory is used to store html; build/web is used to store javascript, images, and css.

How should JavaScript files be **delivered** in the real application, so that **less code** has to be loaded on each request and not too much requests have to be sent?

At the build stage, all JavaScript is minified and combined into one file (build/web/js/app.js), so that client will have to make only one HTTP request when he / she visits your site for the first time.

It's probably a good idea to make building process as easy as possible. Brunch does that by offeringbrunch watch command, which monitors filesystem for changes and builds code instantly with the help of Stitch and some other tools.

(It should be noted that during development brunch projects also use CoffeeScript as the primary language; it is transparently compiled by brunch before stitching the resulting JavaScript. However, this doesn't matter as long as file organization is concerned, and is out of scope of your question.)

2)

For all JavaScript files definitely use a separate directory. Have as many files as possible semantically. One large constructor should correspond to a separate file. Never use filename prefixes where you can create a directory.

For smaller projects the following structure is often found on GitHub:

* src -- for the development JavaScript files and subdirectories of such files.
* lib -- for libraries.
* tests -- for unit tests.
* build -- for buildscripts.
* dist -- for compiled files.

From the dist directory create symlinks to the deployment location if necessary.

For compiling we use a Makefile with targets for production and development. The production version is all of files [JSHint](http://jshint.com/)`ed, minified and concatenated into one. The development target is generating a server-side script that includes all JavaScript files dynamically (for easy inclusion into HTML).

But generally it depends. We used a widget directory for one project. This widget directory had a set of separate widget subdirectories (e.g. slider, tabs, modal-window), each of which had the following layout (inspired by [DOMLoader](https://github.com/azer/domloader)):

* html -- for HTML templates.
* css -- for CSS files necessary for the widget.
* js -- for the widget JavaScript constructor.

## Espacio de Nombres

Since our application will very modular and encompass many functions or classes,  
we’ll establish a namespace to provide some structure and avoid function name clashes.  
You can read more about how to implement namespaces in Javascript in the following two articles:

* [Elegant Code – Namespaces](http://elegantcode.com/2011/01/26/basic-javascript-part-8-namespaces/)
* [StackOverflow “Javascript Namespace Declaration”](http://stackoverflow.com/questions/881515/javascript-namespace-declaration)

HTM5 Application Cache