Todo proyecto web tiene 3 bloques esenciales:   
  
**HTML**: Qué otorga toda la estructura o semántica a nuestro sitio.  
  
**CSS**: Aquél que le da el estilo y hace de nuestro sitio algo único.   
  
**JavaScript**: Con JavaScript ganamos interacción en el sitio y prácticamente es el lenguaje de programación en el que está escrito todo sitio web, inclusive, se encuentra en el backend, algo impensable hace años. JavaScript además es muy versátil, puedes construir desde aplicaciones de escritorio, aplicaciones nativas para dispositivos móviles, aplicaciones de consola etc. JavaScript se está comiendo al mundo.

**Package bundling con gulp y sass**

Las primeras dudas que tendremos como desarrolladores JavaScript serán prácticamente sobre qué herramientas usar: Gulp, grunt, browserify, webpack, sass, less, stylus: ¿Qué son y qué usar en cada caso?. Es importante distinguir primero su funcionalidad: Tenemos constructores de paquetes, gestores de tares y preprocesadores de CSS. Cada herramienta tiene su forma de uso, e incluso, puedes combinarlos entre sí. Para nuestro proyecto elegimos: browserify (lo usamos más adelante), gulp y sass, y así es como lo configuramos.

Nota: Para activar el Syntax de Sass en Sublime Text 2 & 3, vayan a:  
Preference ==> Package Control ==> Allí escriben - Package Control: Install + Enter ==> en las opciones escriben - Sass y dan clic encima para que se active. Una vez hagan esto van: View ==> Sintax y allí seleccionan Sass, para que la siguiente línea de comandos se active con los colores correspondientes que indican que se esta escribiendo css para Sass.

**Agregando vistas con Pug**

Recordando que estamos construyendo una single web application, necesitamos de un motor de templates ( en este caso usaremos Jade / pug ) que justamente nos ayuda a crear diferentes fragmentos de HTML correspondientes a cierta funcionalidad. Dicha funcionalidad o vista va a ser llamada o requerida desde JavaScript (al igual que sus estilos). No te preocupes, poco a poco, agregaremos más funcionalidades y te permitirá entender mejor el uso de templates.

**Organizando nuestros assets y agregando un favicon**

Vamos a crear una nueva tarea en Gulp, en este ocasión todos los recursos que tengamos en una carpeta los vamos a generar de manera automática en otra carpeta, esto incluye imágenes y código de css resultante. Igual vemos por primera vez los fav-icons, optimizados para todos los dispositivos.

**Agregando JavaScript en el cliente**

Hemos hecho un poco de JavaScript del lado del servidor: ¿Cómo nos preparamos para el front-end?, muy sencillo nos apoyaremos de Browserify, Babel y Gulp. La combinación de estas herramientas nos permitirán ejecutar código de ECMAScript 6 para todos los navegadores desde ya. Por lo cual, aprovecharemos todas esas nuevas ventajas que nos da el nuevo standard.

**Agregando navegación con Page.js**

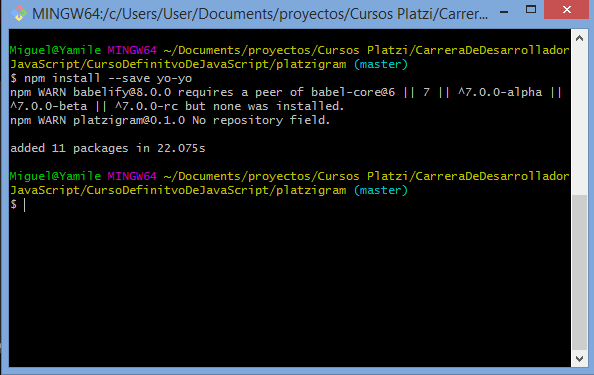
Uno de los elementos por lo que se distingue un single page application es la navegación. Muy similar a las rutas de express, page.js nos ayuda a crear esta funcionalidad desde el lado del cliente.

**Automatizando el build**

Vamos a mejorar nuestro entorno de desarrollo gracias al uso de “scripts” en nuestro package.json, gulp y watchify. Podemos crear un comando que haga desde ejecutar las tareas de construcción de proyecto hasta levantar el servidor. Imagina poder escuchar los cambios en cualquier archivo y observar de inmediato los cambios, seguro será tu entorno favorito.

**Agregando interfaces de usuario con Yo-yo**

Yo-yo sigue un poco la filosofía de React (Orientado a componentes) : Nos ayuda a trabajar con templates y volverlos objetos del DOM, además, de actualizarlos según se requiera, muy sencillo y del lado del cliente. También usaremos los Templates Strings, funcionalidad que ganamos gracias a ECMAScript 6.



**Modularizando nuestro proyecto**

Parte de hacer buen código en JavaScript es modularizar cada funcionalidad. Para convertir cada archivo js en un módulo usamos module.exports, y para usar dicho módulo en otra parte de nuestra aplicación usamos requiere (Si, justo como las librerías importadas con NPM). Notarás que el resultado del ejercicio es tener un código más legible y modular.