Switch IDE

Borrador — Universidad Técnica Federico Santa María

Ian Murray Schlegel

A los que creyeron en mí, a los que me apoyaron,
a todos los que hicieron esto posible.

Índice

1. Introducción

En el contexto del desarrollo de aplicaciones web, existen dos grandes "corrientes". Por una parte, es posible desarrollar aplicaciones completamente de lado de servidor. Esto significa que la aplicación procesa todos los datos y genera todo lo que el usuario ve de forma remota. Esta forma de desarrollar ha sido así durante muchos años y sigue siendo una forma muy utilizada. Por otra parte, últimamente, con el crecimiento de la comunidad de Javascript, y la constante mejora en rendimiento de los navegadores modernos, se ha popularizado la idea de llevar gran parte de la lógica de negocio y el procesamiento de datos al cliente. Los navegadores modernos tienen cada vez más capacidad de ejecutar procesos rápida y eficientemente, lo que tiene varias ventajas, como por ejemplo, se aliviana la carga en los servidores, lo que permite poder soportar a muchos más usuarios simultáneos, y las aplicaciones se desempeñan mucho mejor, dado que se disminuye el retardo que hay en transmitir datos entre el cliente y el servidor. Esto último es muy importante al momento de crear aplicaciones web. Si bien toda página web, sin importar su naturaleza, debería ser lo más rápida y responsiva posible, las aplicaciones web deben serlo por sobre todo. Al fin y al cabo, están intentando imitar el comportamiento de aplicaciones nativas, pero a su vez alivianando la carga a la que se somete un desarrollador al momento de crear aplicaciones compatibles con una infinidad de dispositivos y sistemas operativos distintos.

Desarrollar aplicaciones web versus desarrollar aplicaciones nativas (que son ejecutables directamente en el computador, sin necesidad de un navegador) tiene varias ventajas, siendo quizás la más importante que no es necesario escribir el programa para diferentes plataformas, dado que la mayoría de los navegadores modernos funcionan en una gran variedad de dispositivos. Además, con el crecimiento del estándar HTML5 (que al momento de escribir el presente documento aún se encuentra en proceso de convertirse en un estándar final), es posible aprovechar muchas características de los dispositivos (en algunos casos incluso es posible usar los acelerómetros de los teléfonos móviles inteligentes). Es más, muchos juegos han sido portados a la web, utilizando WebGL y tecnologías similares.

Ahora bien, desarrollar aplicaciones web también tiene sus desventajas. Es cosa de ver Xcode o Visual Studio, donde ambas herramientas son un entorno completamente integrado para desarrollar aplicaciones. Desde escribir código a crear formularios y diferentes tipos de vistas, ambas herramientas (y otras similares) entregan una experiencia casi inigualable al desarrollador. Al desarrollar aplicaciones y sitios web en general, no es posible encontrar herramientas que se asemejen lo suficiente a las mencionadas (y que sean de código abierto) como para considerarse una alternativa viable. La mayoría de los entornos de desarrollo para web permiten previsualizar lo que el desarrollador codifica, pero no le permiten ahorrar tiempo al momento de realizar tareas tan necesarias como codificar la interfaz de una aplicación.

Es este último aspecto el que se considera como un problema actualmente en el mundo del desarrollo web. Si bien no es difícil codificar interfaces de usuario al momento de crear aplicaciones web, es una tarea que consume tiempo y para la cual sí existen herramientas muy buenas en el mundo del desarrollo de aplicaciones nativas (como las ya mencionadas Xcode o Visual Studio).

1.1. Objetivos

En este trabajo se propone la creación de un entorno de desarrollo integrado, al que se le llamará **Switch IDE**, que permita el desarrollo de aplicaciones web facilitando la creación de interfaces de manera similar a las herramientas descritas anteriormente.

1.2. Estructura de este Trabajo

El trabajo se estructurará como sigue:

- En el Capítulo 2 se revisará primero el estado del arte, analizando las herramientas que actualmente intentan dar solución al problema identificado, además de frameworks y otro tipo de utilidades que mitigan de cierta forma el problema pero sin darle una completa solución.
- Luego, en el Capítulo 3 se propondrá una solución al problema identificado.
- Se construirá y documentará la creación de la solución planteada en el Capítulo 4.
- Se analizarán los resultados utilizando métricas que se describirán en el Capítulo 5.
- Finalmente, en el Capítulo 6 se presentarán conclusiones del trabajo junto con ideas para posible trabajo futuro.

2. Estado del Arte

La metodología para el desarrollo de aplicaciones web está cambiando. Ha pasado de estar enfocada casi completamente de desarrollar de lado de servidor, a desarrollar parcial o totalmente de lado de cliente. Frameworks como Backbone han revolucionado lo que se piensa sobre desarrollar aplicaciones completamente usando Javascript, y la aparición de muchísimos frameworks nuevos en este joven sub-mundo de aplicaciones muestra claramente una tendencia hacia este "paradigma".

Ahora bien, el hecho de que periódicamente aparezcan nuevos frameworks no es necesariamente bueno. Es fácil perderse, no se puede saber por dónde empezar, y lo peor de todo, cada framework hace lo suyo de formas diferentes, incluso utilizando paradigmas de desarrollo distintos (ya sea MVC [1], MVP [2] u otro de los que normalmente se utilizan).

En este capítulo, se revisarán las diferentes herramientas que existen en el mundo del desarrollo de aplicaciones Javascript, además de programas y utilidades que funcionan de manera similar a lo que se quiere lograr con Switch IDE y que están actualmente en el mercado. Se revisarán primero diferentes frameworks disponibles hoy en día, analizando sus ventajas y desventajas, para luego mostrar herramientas que facilitan el uso de frameworks.

2.1. Frameworks Actuales

Existe una variedad enorme de frameworks para desarrollo web de lado de cliente, y, como se dijo anteriormente, día a día aparecen nuevos competidores, lo que pasó de ser algo bueno a algo que aumenta las barreras de entrada. El hecho de que haya tantas opciones para desarrolladores (incluso experimentados) hace que elegir uno sea muy difícil y que finalmente se opte por la solución incorrecta. Muchos frameworks tienen varios puntos fuertes, y no siempre un framework es la mejor solución para un tipo determinado de problema.

Ahora bien, sí existen buenos frameworks y varios de ellos son relativamente fáciles de entender y dominar. La mayoría de ellos llevan buen tiempo en el mercado y por ende tienen una comunidad fuerte y activa, junto con una base de código robusta.

2.1.1. Backbone

Backbone [3] es un framework simple, extensible y liviano, lo que lo hace una muy buena opción para desarrollar todo tipo de aplicaciones. Además, sus pocas dependencias de otras librerías hacen que las aplicaciones desarrolladas con él sean más livianas comparadas con otros frameworks.

El objetivo principal de Backbone es facilitar y dar estructura a aplicaciones que se basan fuertemente en funcionar del lado del cliente (es decir, en el navegador mismo). Normalmente, escribir aplicaciones de este estilo es posible utilizando sólo Javascript y sin usar algún framework, pero ello resulta tedioso, y lleva a aplicaciones difíciles de mantener. Backbone (y la mayoría de los frameworks que se nombrarán en este capítulo) intentan evitar esto último dándole una estructura a las aplicaciones, separando vistas de controladores y modelos.

Es utilizado por una gran cantidad de sitios, tales como Groupon Now!, Trello, entre otros [4]. Las aplicaciones nombradas no son proyectos pequeños y simples, sino que son aplicaciones muy poderosas que se benfician muy bien de lo que Backbone provee.

Backbone es un framework de código abierto y por lo tanto completamente gratuito.

2.1.2. Cappuccino

Cappuccino [5] es un framework de desarrollo web enfocado en llevar "Cocoa" de Apple a la web, aunque no está en forma alguna afiliado con esta empresa. Abstrae completamente el desarrollo web a un único lenguaje: Objective-J.

Las ventajas de este framework son varias. Al estar imitando frameworks nativos de Apple, sigue varios estándares ya conocidos, y lo hace bastante fácil de aprender para una persona con experiencia en desarrollo iOS o Mac OS X. Además, todo se desarrolla con el mismo lenguaje, y trae integrados varios widgets (botones, tablas, ventanas, menús), lo que le permite al desarrollador enfocarse sólo en código y no en el diseño (ver Figura 1). A diferencia de Backbone, en donde el desarrollador debe trabajar por un lado con el código y por otro con el diseño y los estilos de las aplicaciones, Cappuccino trae todo eso en un solo framework.

Una de las características interesantes de este framework, es la capacidad de usar el constructor de interfaces de Xcode [6] — Interface Builder — para crear las interfaces. Eso sí, no todos los widgets que están disponibles en Xcode están implementados en Cappuccino, y agregar elementos inexistentes no siempre arroja errores fáciles de entender. Además, desarrollar usando esta técnica, requiere instalar varios componentes (entre ellos, Xcode, que no es una aplicación liviana) y se necesita un computador Mac, dado que Xcode no funciona en otras plataformas.

Además de lo anterior, tiene otras desventajas. Por un lado, es necesario aprender un lenguaje nuevo (Objective-J). Por otro lado, hay varios widgets esenciales que no están implementados, como por ejemplo, controles de entrada de texto multilínea no está soportado actualmente por el framework.

Cappuccino es un framework de código abierto y gratuito.

Figura 1: Un ejemplo de los diferentes widgets que vienen incluidos en Cappuccino. Puede apreciarse el diseño estilo OS X que trae.

2.1.3. Ext JS

Ext JS [7] es un framework con soporte para una gran variedad de widgets y es muy poderoso. Una de las ventajas importantes de este framework, es que existe una empresa detrás: Sencha Inc. Esto hace del framework una herramienta continuamente soportada y en constante desarrollo. Incluso, esta empresa ofrece soporte técnico pagado para este framework.

Al llevar bastante tiempo circulando (desde el 2007 [8]), es un framework con una gran cantidad de widgets y características disponibles que lo hacen muy versátil. De la misma forma que Cappuccino, trae integrados widgets como botones, tablas, ventanas, entre otros, pero con la diferencia de que este framework sí está escrito en Javascript directamente, por lo que no hace falta aprender un lenguaje nuevo.

Ahora, es un framework muy completo, por lo que su curva de aprendizaje es más empinada comparándose con otras herramientas. Además, este framework no es gratuito para desarrollo comercial. Si se desea desarrollar una aplicación web y mantener el código propietario (es decir, no liberarlo como código abierto), se deben cancelar (por lo bajo) $329 dólares americanos [9].

2.2. Herramientas Actuales

Ahora que se han visto una variedad de frameworks disponibles para desarrollar aplicaciones web, se procederá a analizar el mundo de las herramientas para el desarrollo de éstas. El

enfoque de esta sección es analizar diferentes programas y servicios que se ofrecen, que son en alguna forma similares a lo que se quiere lograr en el presente trabajo.

2.2.1. Sencha Architect

Sencha Architect [10] es una herramienta que tiene varias similitudes con lo que se quiere lograr en este trabajo. Es una herramienta de los mismos creadores del framework Ext JS, es bastante poderosa, y permite desarrollar aplicaciones web y móviles de manera visual.

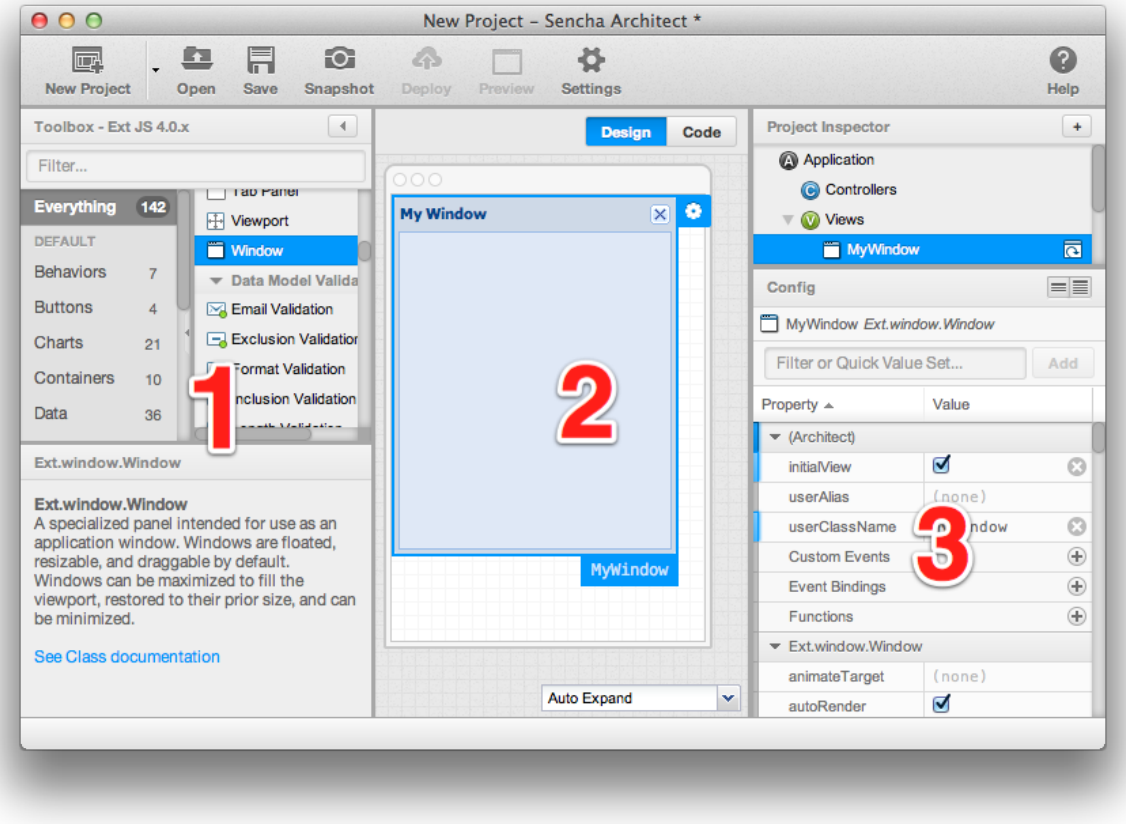

Figura 2: Sencha Architect: La ventana de trabajo

Sencha Architect es la herramienta que más se asemeja a un IDE para desarrollo de aplicaciones nativas (como Xcode o Visual Studio). Posee una barra lateral con componentes (ver Figura 2, el número 1), un área central de trabajo (número 2) y propiedades de los diferentes componentes que existan en el área de trabajo (número 3).

Las ventajas de esta herramienta son claras: es posible crear las vistas de las aplicaciones

directamente, arrastrando componentes. De la misma forma que herramientas para desarrollo nativo, esto ahorra tiempo al momento de desarrollar.

En cuanto a las desventajas, el desarrollador está obligado a trabajar con Ext JS como framework (que, como ya se dijo antes, es complejo de dominar), y, por otro lado, el uso de este producto es pagado: cuesta $399 dólares americanos [11] al momento de escribir este documento. Además, a ese valor hay que agregarle otros $329 por la licencia de uso de Ext JS [9], si es que se quiere para uso comercial.

2.2.2. Divshot

Divshot [12] es una de las herramientas que inspiró la presente memoria. Es una aplicación de prototipado rápido basado en web. Fue creada en abril de 2012, por lo que es una herramienta relativamente nueva.

Permite, utilizando Twitter Bootstrap [13], crear prototipos de sitios y aplicaciones web arrastrando componentes, de la misma forma que IDEs como Xcode o Visual Studio. En la presente memoria, se quiere lograr un comportamiento muy parecido para la creación de templates.

Entre las ventajas que presenta la herramienta, está la facilidad con la que se pueden crear prototipos de vistas. Sólo arrastrando widgets, se puede llegar a una vista en pocos minutos. Además, es posible previsualizar los resultados fácilmente, e incluso exportar a HTML[1] con un sólo click. Lo mejor de todo, es que el HTML generado está muy bien ordenado y formateado.

Es una herramienta muy puntual y bien diseñada, pero sólo permite crear vistas, no desarrollar. Además, es una herramienta de pago, y no es de código abierto.

2.2.3. eXo Cloud IDE

eXo Cloud IDE [15] es un entorno de desarrollo integrado, en la nube. Tiene varias características que lo hacen una buena opción al momento de querer colaborar o mantener el código alojado en internet. Soporta una gran variedad de lenguajes (entre ellos Java, Ruby y Python) y frameworks (como Ruby on Rails, Spring o incluso Google App Engine).

Además de lo anterior, soporta plataformas como Heroku para subir cambios a servidores directo desde el navegador, e incluso tiene soporte para versionamiento con Git [16].

[1] **Hypertext Markup Language:** es el lenguaje de etiquetas utilizado para las páginas web. Todas ellas están hechas con esto, más una combinación de otros lenguajes. [14]

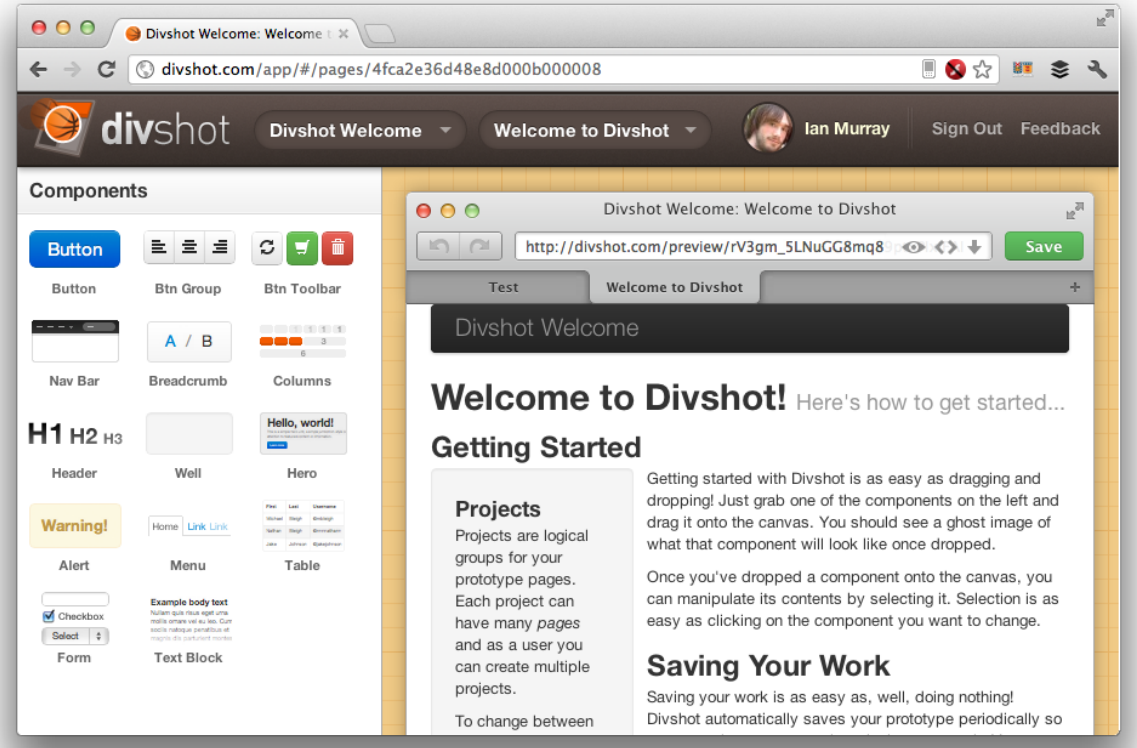

Figura 3: La ventana principal de Divshot. Se pueden apreciar los componentes en la barra lateral izquierda, con la vista a la derecha, donde se arrojan los componentes.

Como cualquier IDE nativa estilo Xcode o Microsoft Visual Studio, tiene un visor de los archivos actualmente abiertos (ver Figura 4, número 1) y el editor de código mismo (número 2).

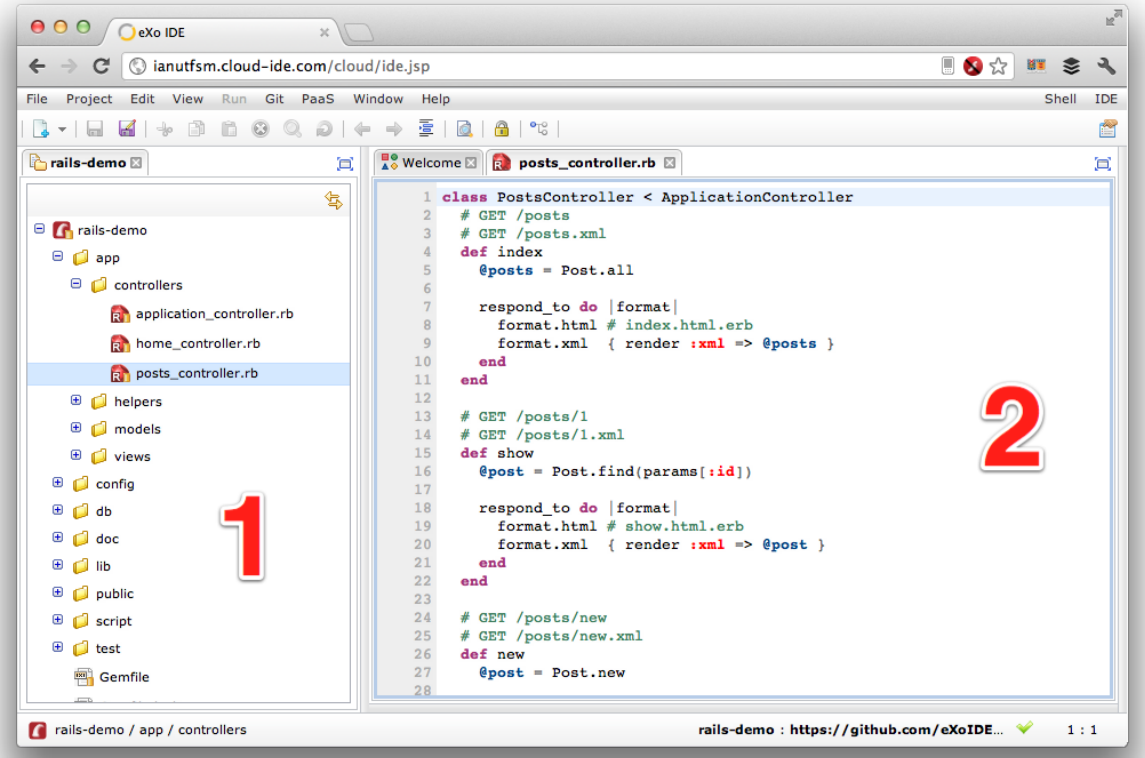

Figura 4: eXo Cloud IDE: un entorno de desarrollo integrado, en la nube

Las ventajas que provee esta herramienta son el no tener que depender de un sólo equipo para desarrollar. Al mantener el código en la nube, sólo es necesario conectarse al sitio web y empezar a trabajar. Por esto último, tampoco es necesario instalar las diferentes herramientas necesarias para desarrollar (como puede ser instalar Ruby o Python, que a veces es engorroso).

Ahora bien, no permite crear templates de forma visual (que es lo que uno espera de una herramienta integrada de este estilo). No permite desarrollar aplicaciones de lado de cliente, sólo de lado de servidor, y desarrollar aplicaciones con los frameworks que soporta por lo general implica utilizar mucho el terminal de comandos para realizar pruebas, y en un ambiente web, eso no es fácilmente replicable.

Es una herramienta de código cerrado pero gratuita.

2.2.4. Wavemaker

Wavemaker [17] es una herramienta que permite generar "bases de datos" de manera visual. El objetivo principal de esta aplicación es poder crear sistemas de manejo de datos de manera fácil, escribiendo la menor cantidad de código posible.

Esta herramienta se asemeja más a lo que es Microsoft Access. La idea es poder crear interfaces para guardar y organizar información rápidamente y sin programar. Es una aplicación enfocada más a gente que no tiene conocimientos de programación, aunque según los creadores de Wavemaker, las aplicaciones que se generan son aplicaciones Java completas, que pueden ser extendidas más adelante programando Java directamente.

Las ventajas dependen realmente de lo que se quiera hacer con la herramienta. Si el usuario no tiene conocimientos en el área de desarrollo de software, entonces la herramienta es bastante útil, aunque no es posible crear aplicaciones muy complejas. Por el contrario, si se le mira desde el punto de vista de un desarrollador, la herramienta no ofrece mayores beneficios a las anteriormente mencionadas.

Es una herramienta gratuita, aunque de código cerrado.

2.2.5. Zoho Creator

Zoho Creator [18] es un servicio web que tiene más o menos el mismo enfoque que Wavemaker: crear bases de datos simples sin programar. Ahora bien, se diferencia un poco con Wavemaker en el sentido de que es un servicio, pagado, que permite crear aplicaciones en la nube y mantenerlas ahí, mientras que la herramienta anterior permite crearlas y exportarlas, para luego incluso extenderlas si es que se tienen conocimientos.

Si bien Zoho Creator es una herramienta relativamente poderosa y fácil de usar, no es una herramienta que utilizaría un desarrollador. Hay varias razones para esto último. Primero, no permite exportar lo creado para alojarlo en otro servidor; segundo, no es una herramienta de desarrollo que permita crear aplicaciones de lado de cliente o de servidor, sino que permite crear simples bases de datos; y tercero, no es una herramienta gratuita o de código abierto, lo que hace que lo creado con ella tenga ciertas limitaciones en cuanto a licencias.

Como se dijo antes, es un servicio de código cerrado y pagado.

3. Propuesta de Solución

Para solucionar el problema identificado, se propone una herramienta web (es decir, que sea accesible desde el navegador, y no una aplicación nativa), que permita editar una aplicación web completa en el mismo navegador, junto con permitir al desarrollador crear templates de manera visual, en vez de utilizar sólo código.

El editor de templates deberá permitir al usuario crear las interfaces que presentarían sus aplicaciones de manera visual, sin obligarlo a escribir todo en código. Mediante el arrastre de diferentes widgets prefabricados (como botones, formularios, ventanas modales, entre otros), el usuario podrá generar templates en un tiempo menor y con mayor facilidad que al hacerlo de la manera tradicional.

3.1. Requisitos

Principalmente, la herramienta será un Entorno de Desarrollo Integrado compatible con Backbone[2] y deberá:

- Permitir al usuario desarrollar aplicaciones utilizando el framework Backbone[3] y Twitter Bootstrap[4]
- Dar una estructura a las aplicaciones web, evitando que el desarrollador programe la aplicación completa en un sólo archivo
- Facilitar la creación de vistas mediante un editor que permita arrastrar diferentes widgets a un "canvas"
- Permitir ensamblar y probar la aplicación de manera similar a cómo lo haría un programa nativo

Una herramienta con tales características no podrá funcionar sólo de lado de cliente (exclusivamente en el navegador), dado que algunas funciones tendrán que ser ejecutadas en el servidor, como por ejemplo la compilación de los archivos Javascript o levantar una instancia de un servidor para probar lo que el usuario esté desarrollando. Por esto, es necesario implementar este proyecto en dos componentes distintas.

A continuación, se especificará qué roles cumplirán backend y frontend en la herramienta propuesta, junto con sus requisitos específicos.

[2] Ver estado del arte blabla.
[3] Será necesario especificar por qué?
[4] Explicar o citar, no sé U_U.

3.1.1. Definición y Requisitos del Backend

El backend será un servidor que permitirá al frontend realizar las tareas que no le serán posibles por diferentes limitaciones. Por ejemplo, los sitios web no pueden manipular archivos del sistema operativo por motivos de seguridad, por lo que es imposible realizar todo en el frontend. Por lo tanto, el backend cumplirá con las siguientes tareas:

- Deberá generar los proyectos y sus estructuras (directorios y archivos base), junto con almacenarlos, dado que no es posible almacenarlos en el cliente
- Manipular los archivos, es decir, crear, eliminar, renombrar y actualizar archivos y carpetas
- Ensamblar los proyectos
- Levantar servidores que permitan al desarrollador hacer pruebas

Es importante dejar claro que el usuario nunca deberá interactuar con el backend directamente, será el frontend el que interactúe con éste y el usuario no verá esta interacción.

3.1.2. Definición y Requisitos del Frontend

El frontend será el encargado presentarle al desarrollador todo lo que necesite, como mostrar los archivos en un proyecto, mostrar un editor de código o el editor de templates propuesto.

Específiamente, el frontend deberá permitir al desarrollador:

- Elegir entre diferentes proyectos existentes
- Crear proyectos nuevos
- Explorar los diferentes directorios de un proyecto abierto
- Crear, renombrar y eliminar archivos y carpetas en un proyecto
- Editar archivos, mediante un editor de código con resaltado de sintaxis
- Editar visualmente los templates, listando diferentes widgets que el desarrollador podrá arrastrar a un “canvas” que se actualizará en tiempo real mostrando una vista previa del template
- Ensamblar y probar el proyecto

3.2. Elección de Herramientas

3.2.1. Elección de Herramientas para el Backend

Existen varias alternativas en cuanto a la creación de un backend que cumpla con los requisitos especificados anteriormente. Entre los lenguajes más populares para el desarrollo de servicios web se encuentran PHP, Java, Python y Ruby[5]. Para todos ellos existen una variedad de frameworks para desarrollar servicios web (como CodeIgniter, Play, Django o Ruby on Rails, respectivamente[6]). Además, todos permiten manipular archivos en el servidor, que es uno de los requisitos especificados para el backend.

Si bien el autor tiene conocimiento de todos los lenguajes mencionados, está más familiarizado con el último (Ruby). Además, para este lenguaje existen diferentes librerías que permitirían agregar funcionalidad extra en otra oportunidad (como Grit, una librería para manipular repositorios Git desde Ruby)[7].

El backend no se programará directamente, sino que se utilizará un framework. Lo que se quiere desarrollar en el lado del backend es básicamente una API (Application Programming Interface), que el frontend utilizará parar funcionar correctamente. Existen varias formas de programar APIs en Ruby, donde las más populares son Ruby on Rails y Sinatra[8].

Ruby on Rails es un framework extremadamente completo, y el más popular entre programadores de Ruby[9]. Facilita una gran cantidad de tareas que son tediosas en otros frameworks, lo que la hace una herramienta ideal para todo tipo de proyectos web. Sin embargo, al ser una herramienta tan completa, agrega bastante overhead. Además, es una herramienta ideal para crear proyectos grandes y complejos, y dado que el backend de el presente trabajo no requerirá tanta complejidad, se convierte en una alternativa no tan ideal para este trabajo.

Sinatra es una versión simplificada de Ruby on Rails. Es un framework más simple y liviano, por lo que inicia más rápido y, en general, se desempeña mejor que su contraparte. Tiene la desventaja de no poseer tantas facilidades para desarrollar sitios complejos, pero dado que el backend de este proyecto no requerirá tanto trabajo, se considera como la mejor opción y se utilizará para desarrollar el backend.

El backend requerirá una base de datos para poder guardar información de los usuarios y proyectos. Se consideraron dos alternativas: MySQL[10] y MongoDB[11]. La primera es una base

[5] Http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
[6] Links!
[7] Link.
[8] Http://hotframeworks.com/languages/ruby.
[9] Misma cita de antes.
[10] Link.
[11] Link.

de datos convencional y la más popular[12]. La segunda es una base de datos de la familia conocida como "NoSQL", es decir, no es una base de datos relacional. Permite almacenar "documentos", o, más específicamente, objetos del estilo JSON[13]. Además, no acepta consultas SQL, si no que las consultas se realizan utilizando métodos directamente en la base de datos.

Las ventajas que tiene MongoDB sobre MySQL son varias, entre algunas: configurar escalamiento horizontal es relativamente sencillo, y permite almacenar archivos en él usando una tecnología llamada GridFS[14].

En términos de uso, existen librerías en Ruby para interactuar con ambas bases de datos. Para MySQL, es posible usar ActiveRecord[15], un ORM[16] muy popular y muy poderoso que permite hacer consultas a la base de datos sin utilizar SQL. Para MongoDB, existe Mongoid[17], un ODM[18] que provee una interfaz muy similar a ActiveRecord.

Las ventajas que posee MongoDB por sobre MySQL hacen de ella una mejor opción, dado que de ser necesario, podría migrarse el almacenamiento de archivos a la base de datos y configurar escalamiento horizontal más fácilmente que de utilizar MySQL. De todas formas, en caso de querer cambiar el sistema de base de datos, no se requerirían muchos cambios dadas las similitudes de las librerías que existen.

Por último, como ya se describió antes, el backend deberá encargarse de crear y estructurar proyectos. Para esto, existe una utilidad llamada Brunch[19]. Brunch es un ensamblador de aplicaciones ("application assembler" en inglés). Básicamente, basándose en una plantilla, organiza aplicaciones en carpetas. Soporta diferentes frameworks y lenguajes, desde Backbone a Knockout[20], usando Javascript o Coffeescript[21]. Además de permitir estructurar los proyectos, los ensambla, es decir, toma todos los archivos y los junta en uno sólo de manera de optimizar la aplicación cuando esté en producción. Utilizar esta herramienta en el backend permitirá crear y mantener proyectos mucho más fácilmente que si se hiciera de forma manual

[12] Http://www.mysql.com/why-mysql/marketshare/.
[13] Link?
[14] Http://www.mongodb.org/display/DOCS/GridFS.
[15] Link!
[16] Object Relational Mapper.
[17] Link!
[18] Object Document Mapper.
[19] Poner referencia a esto!
[20] Referencia!
[21] Referencia.

3.2.2. Elección de Herramientas para el Frontend

Para el desarrollo de aplicaciones de lado de cliente existen varios frameworks, entre ellos Backbone, Knockout y Ember[22]. Todos ellos poseen sus ventajas y desventajas. Por ejemplo, Knockout provee bindings entre los modelos y las interfaces, mientras que Backbone es un framework enfocado en simplicidad y rapidez. De hecho, en ***citararticulo***[23] se menciona que incluyendo comentarios, posee tan sólo 1400 líneas de código.

Considerando que la solución propuesta permitirá al usuario desarrollar aplicaciones utilizando Backbone, además de que este framework se enfoca en simplicidad, se prefirió frente a las otras alternativas

En cuanto a la interfaz, dado que también se encuentra en los requisitos permitir prototipar templates usando Twitter Bootstrap, se prefirió inclinarse por este framework. Es un conjunto de widgets extremadamente popular (es el repositorio más popular en GitHub[24]) y muy completo, proveyendo de todo tipo de widgets que podrían encontrarse en herramientas como Visual Studio o Xcode.

Ahora, Backbone es un framework que no da una estructura a las aplicaciones que se desarrollan con él. A diferencia de, por ejemplo, Ruby on Rails, la tarea de estructurar la aplicación en carpetas o módulos depende completamente del desarrollador. En este punto se asemeja mucho más a Sinatra que a Ruby on Rails. Esto podría considerarse una desventaja, dado que sistemas complejos tienden a crecer y desordenarse bastante si no se aplica una estructura desde un principio.

Por las razones antes descritas, es que se decidió utilizar Brunch. Esta herramienta ya se describió anteriormente, y permitirá facilitar el desarrollo del frontend. Es más, incluye un servidor web de desarrollo, que detecta cambios en los archivos y reensambla todo el sitio de manera de poder hacer pruebas.

En lo que respecta la decisión de lenguaje de programación, no hay muchas alternativas. Es posible desarrollar la solución usando Javascript o Coffeescript. Existen otras alternativas (como TypeScript de Microsoft[25]), pero al momento de escribir este documento no se encuentran en etapas estables de desarrollo ni madurez. Javascript es un lenguaje poderoso pero a la vez de relativo bajo nivel. Ciertas cosas son un tanto tediosas de programar, como recorrer arreglos por ejemplo. En cambio, Coffeescript, un lenguaje de programación escrito por Jeremy Ashkenas que apareció en el 2009[26] que compila a Javascript, hace que

[22] Links.
[23] Https://paydirtapp.com/blog/backbone-in-practice-memory-management-and-event-bindings/.
[24] Https://github.com/popular/starred.
[25] Http://www.typescriptlang.org/.
[26] Https://github.com/jashkenas/coffee-script/commit/8e9d637985d2dc9b44922076ad54ffef7fa8e9c2.

desarrollar en esta plataforma sea mucho más cómodo y simple, sin perder desempeño ni funcionalidad (pues compila directamente a Javascript). Además, hace muchísimo más fácil programar "orientado a objetos". Si bien ECMAScript (la base de Javascript) está definido como orientado a objetos,[27] su estilo es diferente al resto de los lenguajes. Coffeescript agrega palabras clave como `class` y `extends` de manera de facilitar escribir clases y manipular objetos en este lenguaje.

Por ejemplo, escribir una clase en Javascript podría llegar a esto:

```
var ViewController, viewController,
  __hasProp = {}.hasOwnProperty,
  __extends = function(child, parent) {
    for (var key in parent) {
      if (__hasProp.call(parent, key)) child[key] = parent[key];
    }
    function ctor() {
      this.constructor = child;
    }
    ctor.prototype = parent.prototype;
    child.prototype = new ctor();
    child.__super__ = parent.prototype;
    return child;
  };

viewController = ViewController = (function(_super) {

  __extends(ViewController, _super);

  function ViewController() {
    return ViewController.__super__.constructor.apply(this, arguments);
  }

  ViewController.prototype.initialize = function() {
    return this.name = "Switch IDE";
  };

  return ViewController;
```

[27] Http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf página 1.

```
})(View);
```

En cambio, con CoffeeScript, se resume a lo siguiente:

```
viewController = class ViewController extends View
  initialize: ->
    @name = "Switch IDE"
```

4. Construcción de la Solución

Este capítulo tiene por objetivo detallar todo el proceso del diseño y desarrollo de la solución propuesta anteriormente. Se dividirá en los siguientes subcapítulos:

- Introducción a Backbone: se introducirán algunos de los conceptos necesarios para entender cómo funciona Backbone.
- Diseño de la solución: se explicará cómo ambas componentes (frontend y backend) interactuarán entre sí. Además, se describirá cómo funcionarán ambas partes en términos de manipulación de archivos y el proyecto completo. Por último, se explicará cómo se diseñó el componente principal de la solución (el editor de interfaces).
- Construcción de la base: el desarrollo de la solución se dividió en dos etapas. Primero, se desarrolló lo que se consideró una "base" del programa. Esta etapa contempló el desarrollo de gran parte del backend y, en el frontend, una herramienta que permitiera crear proyectos nuevos, crear, editar y eliminar archivos, compilar y correr el proyecto.
- Construcción del editor de interfaces: la segunda parte del desarrollo se enfocó en desarrollar y perfeccionar el editor de interfaces. Dado que este componente es la parte más importante de la solución, se decidió dedicar una etapa completa a él.

4.1. Introducción a Backbone

Dado que en las secciones que siguen se hablará mucho sobre Backbone, se definirán algunos de los conceptos detrás de este framework de manera que el lector pueda entender de lo que se está hablando.

Modelo Representa un objeto en un proyecto, como por ejemplo un archivo, una carpeta o el proyecto mismo. Cada modelo es responsable de persistir su estado de alguna forma (comunicándose con un servidor o almacenando datos en el mismo navegador).

Colección Es básicamente una lista de instancias de un tipo de modelo. Por ejemplo, una carpeta podría considerarse una colección de archivos (siendo cada archivo una instancia de un modelo).

Vista Una vista en Backbone es un archivo que se encarga de presentar información al usuario, y además de interactuar con él, por ejemplo ejecutando funciones cuando se haga un click en un botón. Una vista por lo general presenta un modelo (o una colección). Por ejemplo, se puede tener una vista para cada instancia de un archivo, o bien se pueden tener vistas que no presenten a ningún modelo en particular.

Template Un template es un trozo de HTML que una vista utiliza para generar lo que el usuario ve. Si bien no son enteramente necesarias y una vista podría generar todo lo que necesita con Javascript, hacen la tarea algo más fácil. Contrario a lo que pueda suponerse, el editor visual en el que se trabajará en este documento editará los templates, y no las vistas.

Enrutador En Backbone, las diferentes URL por las que navegue el usuario son manejadas por el enrutador. Éste define las rutas que son soportadas por el sistema y se encarga de instancias modelos y vistas necesarios para mostrarlos correctamente.

4.2. Diseño de la Solución

4.2.1. Backend

La solución es una aplicación mayoritariamente de lado de cliente, por lo que la mayor cantidad de lógica debe ir en este lado. Por esta razón, se diseñó el servidor de la forma más simple posible. Las tareas principales que tiene el servidor o backend de la aplicación son:

Algunas de las tareas que debe llevar a cabo son realizadas por la utilidad Brunch, por lo que sólo es necesario hacer que el servidor ejecute un comando en el terminal. El resto de las tareas son básicamente manipulación de archivos, para lo cual Ruby trae funciones y librerías.

El backend proveerá una interfaz REST[28] para el frontend, principalmente porque Backbone está diseñado para interactuar con APIs de este estilo. En APIs que funcionan con esta metodología, se exponen objetos y sus métodos a requests HTTP, de manera que para obtener información de un proyecto, por ejemplo, el frontend (o cualquier cliente que esté utilizando la API) debe hacer un request como se ve en la Figura 5.

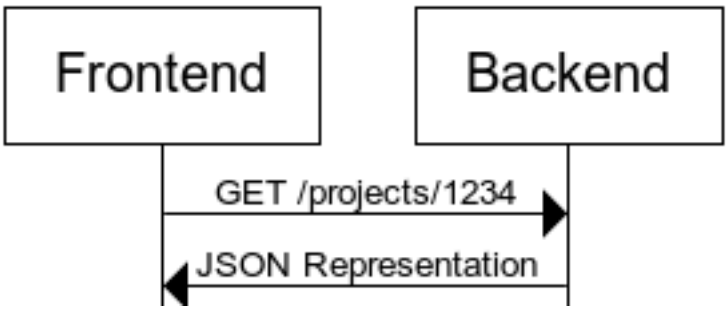

Figura 5: Petición de detalles sobre el proyecto con el identificador “1234”.

[28]Link o algo.

Por lo tanto, para estructurar la API que expondrá el backend, se definieron 2 tipos de objetos:

- Proyectos: es casi la base de todo. Cada proyecto contendrá los diferentes archivos y carpetas.
- Archivos: esto se refiere a archivos y carpetas. Por la forma en la que se manipulan los archivos en el servidor y en el cliente, es más conveniente manejarlos de (casi) la misma forma. Esto último se refiere a la forma en la que se obtienen los archivos en el servidor, y las acciones que se realizan en ellos. Ambos archivos y carpetas se crean y eliminan, como también se renombran. La única diferencia substancial es que las carpetas no tienen contenido y no se actualizan como el resto de los archivos.

De esta forma, el frontend podrá consultar sobre listas de proyectos, detalles sobre cada proyecto (o cualquier método que se exponga, como ensamblar el proyecto) y manipular archivos.

En la Figura 6 puede apreciarse el diseño del backend y las interacciones entre sus diferentes componentes. Por una parte, se tiene el servicio web REST, que expone dos tipos de objectos, los proyectos y los archivos. Ambos utilizan el modelo de proyecto para obtener lo necesario. Por otra parte, el modelo de proyecto utiliza información guardada en MongoDB (como el nombre del proyecto y su ruta en el sistema) para manipular archivos directamente utilizando librerías de Ruby.

4.2.2. Frontend

Se comenzará por definir los diferentes objetos que existirán en el frontend. Se definirán diferentes modelos, colecciones y vistas, por lo que se recomienda revisar la Sección 4.1 para sus significados.

4.2.2.1. Definición de Objetos

4.2.2.1.1. Modelos y Colecciones

A continuación se explicará a grandes rasgos los modelos y colecciones que existirán en el frontend. Se tendrán modelos para los proyectos y los archivos. Cada uno se encargará de comunicarse con el backend para obtener los datos que le sean necesarios o bien para guardar los cambios.

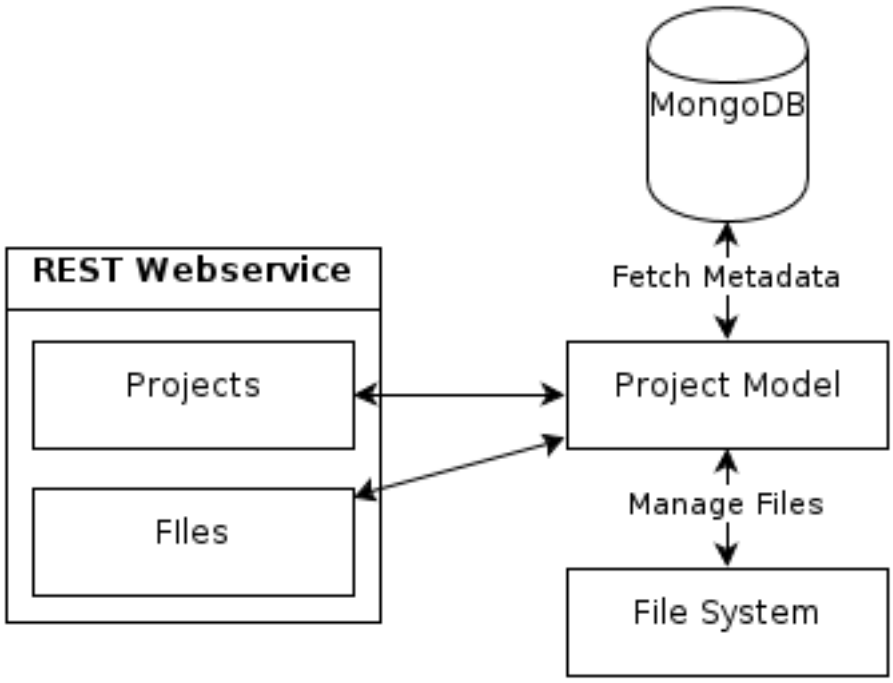

Figura 6: Interacción entre diferentes componentes en el backend.

El modelo de proyecto Guardará el nombre de éste y una referencia a una colección de los archivos que se encuentren en la raíz de su carpeta. Tendrá como responsabilidades crear archivos y carpetas, ensamblar el proyecto e iniciar el servidor de pruebas. Estas acciones se complementan con llamadas al backend que realizan las tareas mismas.

El modelo de archivo Se encargará de guardar el nombre y el contenido (en caso de que corresponda) del archivo o carpeta al que represente, además de guardar una referencia al proyecto al que pertenece y a una colección de archivos en caso de ser un directorio. Tendrá como responabilidades pedir su contenido, actualizarlo, renombrar y eliminar el archivo del sistema. Todas estas acciones se complementan además con llamadas al backend.

La colección de archivos Tendrá como responsabilidad ordenar las listas de archivos una vez que la haya obtenido (además de guardar una referencia a cada modelo de archivo que le corresponda). Ordenar las listas de archios es importante pues el backend arroja una lista de archivos ordenada alfabéticamente, pero, dado que archivos y directorios son considerados de la misma forma en el backend, es necesario ordenar la lista de manera que los directorios queden arriba. Esto facilita encontrar archivos para el usuario.

Como se ve en la Figura 7, instancias de un proyecto tendrán una colección de archivos base (representando el directorio raíz), y estas colecciones contendrán instancias de archivos (que pueden ser archivos o carpetas). En caso de ser un directorio, guardaría una instancia a una colección de archivos.

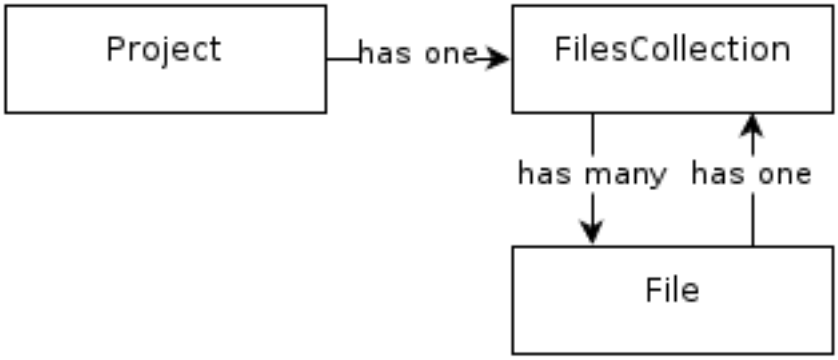

Figura 7: Relaciones entre modelos y colecciones en el frontend.

4.2.2.1.2. Vistas

Cada una de las siguientes vistas considera un template asociado.

Explorador de Archivos se colocará en la barra lateral izquierda, y tendrá dos listas de archivos. Una lista de archivos actualmente abiertos y la lista de archivos y directorios en el proyecto. Tendrá entre sus responsabilidades mantener una lista de archivos que se encuentran actualmente abiertos para que el usuario pueda navegar entre ellos.

Archivo Esta vista representará a un archivo en la vista anterior (exporador de archivos). Mostrará su nombre y un icono que represente si es un directorio, un archivo o un template editable con el editor que se construirá. Entre sus responsabilidades están abrir los archivos (o sea, abrir el archivo en el editor de código o en el editor de vistas en caso que corresponda), embeber listas de archivos en caso de que se clickee un directorio y permitir al usuario renombrar archivos, mostrando un menú contextual.

Editor de Texto Esta vista contendrá el editor de archivos de texto (editor de código). Sus responsabilidades serán mostrar un editor con resaltado de sintaxis y modificar el modelo de archivo que corresponda para guardar cambios.

Editor de Vistas esta vista mostrará templates y, en una barra lateral derecha, diferentes componentes para que el usuario los arrastre y agregue. Contará además con un editor de código HTML, en caso de que el usuario quiera editar la vista o realizar cambios que

el editor no permita directamente. Tiene las mismas responsabilidades que el editor de texto.

4.2.2.2. Diseño del Editor de Templates

El editor de interfaces se construirá de manera que el usuario pueda arrastar componentes como botones o campos de texto directamente en una vista previa del template que esté editando. El contenido de los archivos de templates es simplemente HTML, por lo que es posible presentarlos directamente en la aplicación. Este contenedor o vista previa del template se le llamará "canvas" de ahora en adelante.

El objetivo es que el usuario arrastre elementos hacia el canvas de la misma forma en la que se hace en Xcode o Visual Studio. El sistema debe proveerle retroalimentación visual mostrando el objeto que está arrastrando y además mostrar en qué lugar quedararía el elemento una vez que el usuario lo suelte.

Para implementar este concepto de arrastrar y soltar, se utilizará jQuery UI. esta librería provee, entre otras cosas, métodos para habilitar el arrastrado de elementos en una página. El usuario arrastrará un elemento, y, mediante las llamadas de jQuery, se colocará el elemento en donde el usuario tenga su cursor en el momento, a manera de proveer retroalimentación visual. En cuanto el usuario suelte el elemento, se agregará su correspondiente fragmento de HTML en el template, lo que se reflejará en el canvas en tiempo real.

Figura 8: Los diferentes elementos de retroalimentación visual que se presentarían al arrastrar un componente.

En la Figura 8 se pueden apreciar los diferentes elementos de retroalimentación al momento de arrastrar un elemento. En el punto 1 se pueden ver, primero, un borde amarillo alrededor del

elemento en donde "caería" el componente. Además, en el mismo número, puede visualizarse cómo se vería el componente (en este caso un botón) una vez que el usuario lo deje ahí. En el número 2, puede verse el mismo componente, que sigue al cursor mientras el usuario esté arrastrando. Esto le muestra al usuario qué es lo que está arrastrando.

De esta forma, el desarrollador puede ver, por un lado, qué componente estaría agregando al canvas y, por otro lado, cómo quedaría éste una vez que lo agregue.

4.3. Construcción de la Base

4.3.1. Creación del Entorno de Trabajo

En esta sección se detallarán cómo se crearon y estructuraron los dos entornos de trabajo, tanto para el backend como para el frontend.

Ambos entornos de trabajo se crearon en carpetas independientes (dada su naturaleza) y se inicializaron repositorios Git en cada una, a manera de mantener un control de versiones en cada una.

4.3.1.1. Entorno de Trabajo para el Backend

El backend utilizará Ruby con Sinatra. Sinatra se diferencia de, por ejemplo, Rails, en que es un framework mucho más simple. Por esto, es que no posee utilidades para crear directorios de trabajo. La idea detrás de Sinatra es crear todo en un sólo archivo. Si bien esto es posible, e incluso aconsejable para algunas aplicaciones, no lo es para ésta, en donde se tendrán diferentes modelos y controladores. Por lo tanto se definió la siguiente estructura para el backend:

- `api`
  - `v1`
    - `config`: contiene diferentes archivos de configuración
    - `controllers`: los diferentes controladores
    - `models`: los modelos de usuario y proyecto
    - `app.rb`: este archivo es la base de la aplicación Sinatra, pues nicializa ciertas configuraciones y contiene métodos compartidos por los controladores
    - `boot.rb`: este archivo es cargado inicialmente y se encarga de incluir las diferentes librerías y archivos para incializar el servidor

- `projects`: será el contenedor de los diferentes proyectos que crearán los usuarios
- `public`: esta carpeta sirve archivos estáticos directamente, como imágenes
- `Gemfile`: archivo utilizado por Bundler (una librería de Ruby) para definir qué librerías y en qué versiones utilizará el backend
- `config.ru`: archivo utilizado para levantar el servidor

Se decidió estructurar el backend en carpetas de versiones. La idea detrás de esto es poder dividir cada versión de la API de manera de no perder compatibilidad con posibles clientes que estén basados en una versión de la API. Por ejemplo, de llegar a crearse un cliente para tablets, cada cliente estaría atado a una versión específica. Si se cambiara algún método (o se descontinuara), el cliente automáticamente fallaría. En cambio, teniendo diferentes versiones, se puede mantener esta compatibilidad.

Se creó además una carpeta llamada `projects`. Esta carpeta (que no es accesible directamente), guarda cada uno de los proyectos de cada usuario. Cada vez que se inicialice uno nuevo, se creará una subcarpeta en este directorio con la estructura determinada.

El archivo `Gemfile` es un archivo que utiliza la librería Bundler[29]. Esta utilidad permite especificar librerías externas que se quieren incluir en un proyecto (en este caso el backend) y especificar sus versiones. Por ejemplo, un extracto de un archivo Gemfile podría verse así:

```
gem 'sinatra', '1.3.3' # Especifica que se utilizará la versión 1.3.3

gem 'activerecord', '~> 3.2' # Especifica que se utilizarán
                             # versiones 3.2.X

gem 'haml' # Especifica que se utilizará la última versión
```

La gran utilidad de esta herramienta es que, al momento de ejecutar en la consola `bundle install`, las versiones especificadas son descargadas y se crea un archivo llamado `Gemfile.lock` que guarda las versiones que están siendo utilizadas. De esta forma, cuando otro desarrollador descargue el repositorio y ejecute nuevamente `bundle install` para instalar las dependencias, se descargarán exactamente esas versiones y ambos desarrolladores tendrán el mismo entorno de desarrollo.

Finalmente, el archivo `config.ru` especifica cómo debe levantarse el servidor. Este archivo detalla diferentes rutas que deben ser "montadas" y a las cuales el servidor debe responder

[29] Citar blablabla.

de diferentes maneras. En este caso, se tendrán dos rutas (incialmente). Una, en la que se montará el backend mismo, o sea, `/api/v1`, y la otra, en la que se montará un servidor estático que sirva los archivos en la carpeta `public`.

4.3.1.2. Entorno de Trabajo para el Frontend

De la misma forma en que Switch utilizará Brunch para crear y administrar proyectos, se decidió utilizar la misma solución para construir la IDE. Brunch utiliza un sistema de esqueletos ("skeletons", en inglés), los cuales utiliza para crear la estructura de los proyectos. Existe una gran variedad, utilizando diferentes lenguajes y frameworks. Se encontró uno que utiliza Backbone y CoffeeScript y se utilizó para crear la estructura de archivos inicial.

De no existir esta estructura, habría que escribir todo dentro de un sólo archivo, lo que para aplicaciones muy pequeñas puede ser práctico, pero no lo es en este caso. La estructura consiste en la siguiente:

- `app`
  - `assets`: imágenes y el archivo HTML principal de la aplicación
  - `models`: los modelos y colecciones
  - `routers`: definen los diferentes estados de la aplicación e inicializan lo necesario para funcionar en cada uno
  - `styles`: archivos de estilo (CSS) modularizados para cada sección de la aplicación
  - `views`: las vistas (o controladores)
    - `templates`: los archivos con HTML de cada vista
- `vendor`: librerías (Javascript o CSS) externas, como jQuery, Bootstrap, etc.

Existen otras carpetas que acá no se mencionan dado que no son relevantes a la aplicación. La funcionalidad de cada uno de estos tipos de archivos se explicaron en la Sección 4.1.

Para crear el entorno de trabajo, simplemente se ejecuta el siguiente comando:

```
brunch new -s git://github.com/meleyal/brunch-crumbs.git
```

Este comando utiliza el esqueleto presente en github.com/meleyal/brunch-crumbs para crear un directorio con la estructura ya mencionada.

4.3.2. Prototipado de la Interfaz

Se comenzó por prototipar la interfaz principal. Como ya se ha dicho, se utilizó Twitter Bootstrap, lo que permitió simplificar considerablemente esta etapa. Para realizar el prototipado se requirió realizar un poco de programación, pues hubo que crear vistas y rutas para ir testeando los casos de uso definidos anteriormente. La programación fue mínima de todas formas, enfocando esta etapa en prototipado y no en funcionalidad.

Se prototipó un menú superior con diferentes opciones de manera similar a los menú que se ven en diferentes IDE y programas de escritorio. Se incluyeron opciones como crear un nuevo proyecto, un nuevo archivo, ensamblar y ejecutar el proyecto, etc.

Se agregó la barra lateral izquierda, en la que se muestra una lista de los archivos abiertos y una lista de los archivos en el proyecto. Las carpetas cuentan con un ícono que las muestra como tal, mientras que archivos de templates tienen un ícono que los diferencia de las demás. En la Figura 9 pueden verse los diferentes íconos.

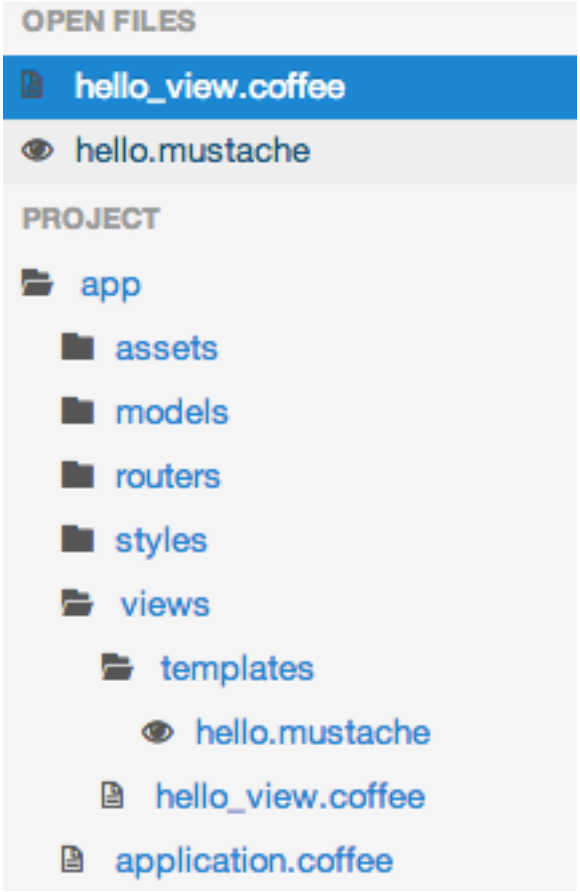

Figura 9: La barra lateral, en la que se pueden apreciar los íconos por carpeta y archivo.

Para el editor de código central se utilizó CodeMirror[30]. CodeMirror es un widget que permite editar código en el navegador con resaltado de sintaxis, líneas numeradas, entre otras características. El editor de código se colocó en la parte central de la interfaz, como puede verse en el recuadro rojo de la Figura 10.

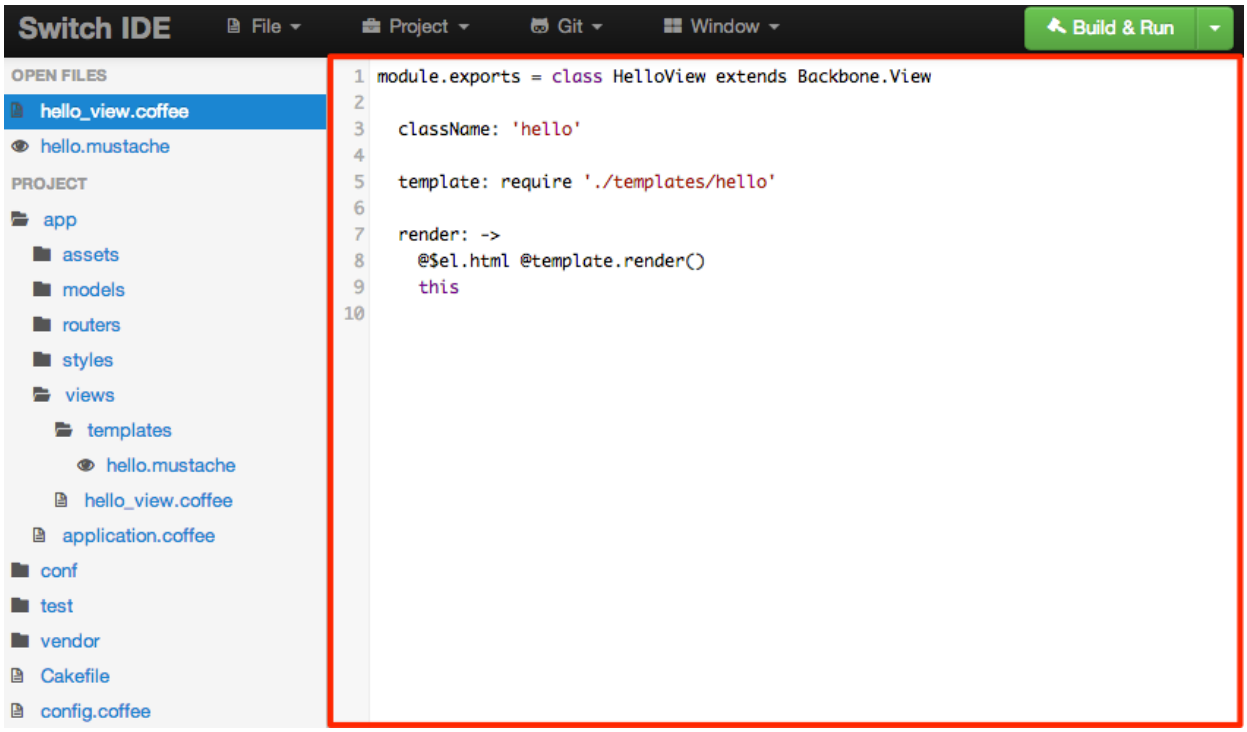

Figura 10: El recuadro rojo muestra cómo se ve CodeMirror al mostrar código en CoffeeScript. Puede apreciarse el resaltado de sintaxis y las líneas numeradas.

En el caso del editor de vistas, se agregó un "canvas" (básicamente un espacio en el cual arrastrar los componentes que se mencionarán más adelante), y una barra lateral derecha, que sólo es visible al estar editando un archivo que la requiera. Además del canvas, en la parte superior se agregaron dos pestañas, que permitirán al usuario cambiar entre el canvas y un editor de código para la vista. En la barra lateral estarán los diferentes componentes en una lista que mostrará una pequeña vista previa del componente y su nombre. En la Figura 11 pueden apreciarse el canvas (número 1) y la barra lateral con widgets (número 2).

La última vista corresponde al selector de proyectos (ver Figura 12), que se creó usando una ventana modal (un widget de Twitter Bootstrap). Ésta se mostraría en el momento que el usuario ingrese al programa. Ahí, podrá elegir algún proyecto en el que haya estado trabajando o crear uno nuevo directamente.

[30]Citar esto.

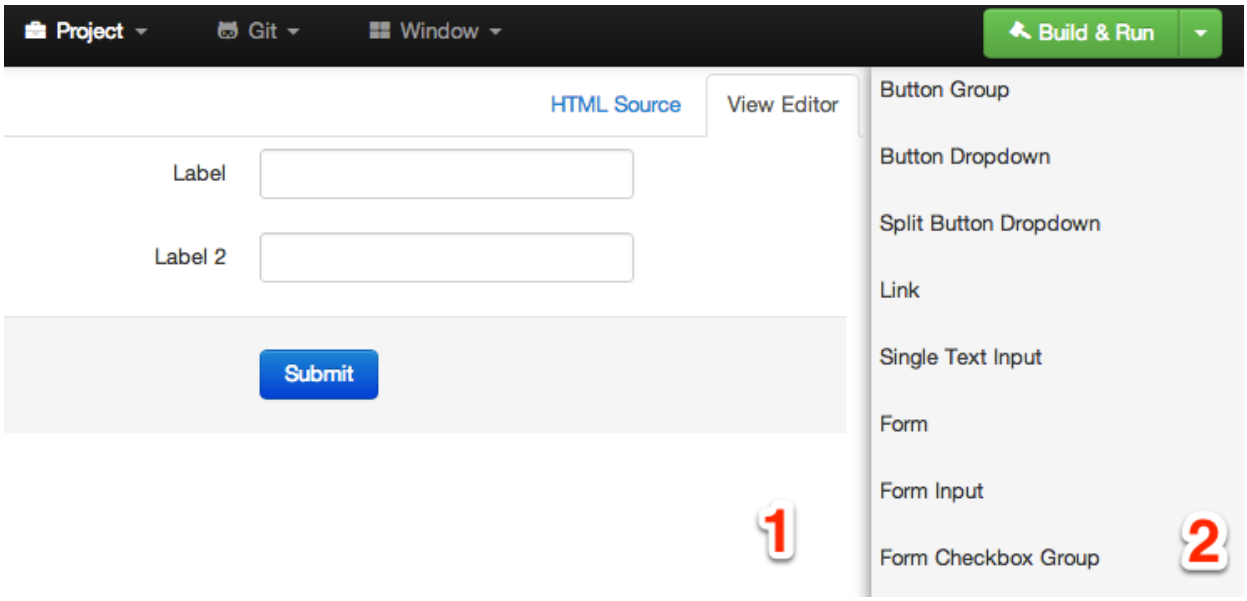

Figura 11: El canvas a la izquierda (1), con los diferentes widgets disponibles a la derecha (2).

4.3.3. Creación de Servicios en el Backend

En las subsecciones siguientes se discutirán algunas de las implementaciones de funcionalidades en el backend. Sin embargo no se discutirán los métodos que se publican y son accesibles a través de la API. Éstos últimos simplemente arrojan respuestas con objetos JSON como el que sigue para comunicarse con el frontend, por lo que no se considera necesario entrar en mucho detalle.

```
[
    {
        "name": "app",
        "parent": "",
        "type": "directory"
    },
    {
        "name": "Cakefile",
        "parent": "",
        "type": "file"
    },
    {
        "name": "conf",
```

```
        "parent": "",
        "type": "directory"
    },
    {
        "name": "config.coffee",
        "parent": "",
        "type": "file"
    }
    // etc...
]
```

El anterior es un fragmento de una respuesta del servidor al pedir los archivos en el directorio raíz de un projecto.

4.3.3.1. Creación de Proyectos

La creación de un proyecto nuevo es relativamente simple. Consiste en crear un nuevo proyecto en la base de datos con el nombre que provee el usuario, y luego crear el proyecto mismo utilizando Brunch.

Al crear un proyecto nuevo, el modelo guarda en la base de datos una nueva entrada que está asociada al usuario que hace la llamada a la API. Guarda un nombre para el proyecto, la ruta en la que fue guardado el esqueleto y un puerto único. Antes de guardar la entrada en la base de datos, un "callback" es gatillado en el mismo modelo (en Ruby) que ejecuta el comando que crea la carpeta para el proyecto, como se detalla a continuación:

```
def create_project
  # Create a random path and the project
  self.path = "#{self.name}-#{SecureRandom.hex(10)}"
  system "brunch new #{self.full_path} \
          -s git://github.com/ianmurrays/brunch-crumbs.git"
end
```

Se crea un sufijo aleatorio para evitar colisiones con proyectos que tengan el mismo nombre, y se llama a un comando de sistema para crear el esqueleto. Esta implementación no es perfecta, dado que podría darse el caso de que `SecureRandom.hex(10)` arroje una cadena aleatoria idéntica a alguna anterior. Para efectos del desarrollo del presente trabajo, se decidió no darle demasiada importancia a este defecto, dado que las probabilidades de que ello ocurra son demasiado bajas.

Después de que finalice este comando, se llama a un segundo "callback" que crea el número de puerto único para esta aplicación:

```
def randomize_port
  begin
    self.port = 8000 + rand(2000)
  end until Project.where(port: self.port).count == 0
end
```

Básicamente asigna un puerto entre 8000 y 10000 aleatorio (y único) a cada proyecto. De esta forma, si se están editando dos proyectos simultáneamente, pueden levantarse dos instancias para hacer pruebas sin colisionar. Esta implementación, si bien es suficiente para el desarrollo de la herramienta y para este trabajo, no sería una implementación ideal en producción, dado que alguno de esos puertos podría estar siendo ocupado por algún servicio en el sistema. En una futura implementación podría agregarse algún mecanismo que verifique la disponibilidad del puerto al momento de asignarlo, o bien, verifique y asigne puertos cada vez que se ensamble y corra el proyecto.

4.3.3.2. Manipulación de Archivos

La manipulación de archivos se hace directamente sobre ellos usando librerías estándar de Ruby. Sólo se discutirán algunas de las implementaciones presentes en el backend.

Para listar los archivos presentes en una determinada ruta, se utilizó la siguiente implementación:

```
def files_in(folder = "")
  # Remove leading and trailing slashes
  folder.gsub! /^\//, ""
  folder.gsub! /\/$/, ""

  files = Dir["#{self.full_path}/#{folder}/*"].collect do |entry|
    next if %w{server.js node_modules}.include? File.basename(entry)
    self.file_to_hash entry, folder
  end
end
```

El método lista todos los archivos presentes en la ruta especificada. De no especificarse, lista los archivos en la raíz del proyecto. Primero se eliminan las barras (/) al principio y final de

la ruta que se especifique, y luego se recorre el directorio usando la clase Dir[31]. La llamada a `Dir[/ruta/a/directorio"]` retorna un array que se recorre utilizando `collect`. El método `collect` en los arreglos genera un nuevo arreglo con los elementos que retorne el bloque que se le pase. `collect` pasa cada elemento del arreglo original al bloque como argumento para su manipulación. Por ejemplo, si el siguiente fragmento de código retorna un nuevo arreglo con sus elementos elevados al cuadrado:

```
[1,2,3,4,5].collect do |num|
  num * num
end

# => [1,4,9,16,25]
```

Por lo tanto, la variable `files` en la implementación de `files_in` contendrá la representación en un Hash[32] (básicamente un diccionario) de cada archivo (o directorio). Además, se excluirán el directorio `node_modules` y el archivo `server.js` del listado, dado que son un directorio y un archivo que no deben ser manipulados por el desarrollador.

La implementación del método `file_to_hash` es relativamente simple, y genera un diccionario con el nombre, padre y tipo de archivo para la ruta que se le pase como parámetro:

```
def file_to_hash(file, folder)
  {
    :name => File.basename(file),
    :parent => folder,
    :type => if File.directory?(file)
      :directory
    else
      :file
    end
  }
end
```

Para obtener y actualizar el contenido de archivos se utilizó una implementación relativamente simple:

[31] Http://ruby-doc.org/core-1.9.3/Dir.html.
[32] Http://www.ruby-doc.org/core-1.9.3/Hash.html.

```
def file_content(path)
  if FileTest.exists? self.full_path(path) \
     and ! File.directory? self.full_path(path)
    {
      :content => File.read(self.full_path(path))
    }
  end
end
```

Básicamente, se verifica que el archivo indicado exista y que no sea un directorio, y se retorna un Hash indicando el contenido del archivo. Se decidió retornar un Hash pues facilita su lectura en el frontend, retornando un objeto JSON en vez del contenido directamente.

Para la escritura de archivos se utilizó una implementación similar:

```
def update_file(path, content)
  if FileTest.exists? self.full_path(path) \
     and ! File.directory? self.full_path(path)
    File.open(self.full_path(path), 'w') do |file|
      file.write content
    end
  end
end
```

Se realiza la misma verificación que al momento de obtener el contenido. Si el archivo existe y no es una carpeta, se reemplaza todo el contenido directamente.

El renombrado de archivos y carpetas se realiza con el siguiente método:

```
def rename_file(path, new_path)
  if FileTest.exists? self.full_path(path)
    File.rename self.full_path(path), self.full_path(new_path)
  end

  directory = File.split(new_path).first

  self.file_to_hash self.full_path(new_path), directory
end
```

Se le pasan la ruta actual y la ruta nueva del archivo, y se devuelve la nueva representación de éste para actualizar el modelo en el frontend. La llamada a `File.rename` permite no sólo renombrar el archivo o directorio, sino que además permite cambiar su ruta.

4.3.3.3. Ensamblado y Servidor de Pruebas

Para realizar el ensamblado se utiliza Brunch. De manera muy similar a la creación de proyectos, es necesario ejecutar un comando de terminal desde Ruby. La diferencia con la creación, es que acá se necesita leer la salida del comando de manera de detectar si el ensamblado fue exitoso o no. Para esto, se utiliza un comando distinto de `system` como se vio antes.

```
def build_project
  output, result = ::Open3.capture2e "cd #{self.full_path} && brunch build"

  {
    :output => output,
    :result => (output =~ /error/ || ! (output =~ /compiled/))
  }
end
```

La librería Open3[33] ejecutar comandos en el terminal con mayor flexibilidad. Dentro de los comandos que provee está `capture2e`, que devuelve el output de `stdout`y `stderr` combinados. Esto es útil en esta situación dado que el ensamblado puede ser exitoso como no. De esta forma, se ejecuta el comando `brunch build`, y se revisa su output. Si éste contiene "error" o no contiene "compiled", significa que ocurrió algún error, y el output se envía al frontend para que sea mostrado.

Para ejecutar el servidor, se utiliza un acercamiento similar:

```
def run_project
  output, result = ::Open3.capture2e "cd #{self.full_path} \
                   && forever stop server.js #{self.port} \
                   && forever start server.js #{self.port}"

  {
    :output => output,
```

[33]Http://www.ruby-doc.org/stdlib-1.9.3/libdoc/open3/rdoc/Open3.html.

```
    :url => "#{Api::V1::App.settings.run_url}:#{self.port}",
    :result => !(output =~ /Forever processing file/)
  }
end
```

Utilizando un programa llamado "forever"[34], se ejecuta un script que ya se mencionó anteriormente (que además viene con el esqueleto utilizado por brunch al crear el proyecto). Este script levanta un servidor estático en el puerto que se especifique, y si la ejecución fue exitosa se retorna la información necesaria al frontend.

4.3.4. Agregado de Funcionalidad al Prototipo del Frontend

4.3.4.1. Selector de Proyectos

Se comenzó por implementar el selector de proyectos. Para esto fue necesario implementar el modelo y colección de proyectos. En esta etapa fueron implementados de la forma más simple posible. Básicamente, se crearon como dos clases que extienden a `Backbone.Model` y `Backbone.Collection` respectivamente. Dado que Backbone está diseñado para interactuar con APIs REST, no fue necesario configurar más que la URL del backend y especificar que la colección corresponde a una colección de Proyectos.

Luego de configurar el modelo y la colección, se agregó una ruta al enrutador de Backbone. El enrutador lee la URL del navegador e interpreta qué método llamar del enrutador. La idea es que se especifiquen las URL necesarias para la navegación de la aplicación. En el caso de esta solución, sólo existirán dos rutas principalmente. La primera será la ruta base, donde se cargará el selector de proyectos del cual se habla, y la segunda será la que tendrá cada proyecto. Por lo tanto, se configuró la URL base, y, en el método que es llamado, se inicializa la colección de proyectos.

```
routes:
  '': 'index'

index: ->
  # We load projects and show them on a modal window
  projects = new Projects()

  projectsView = new ProjectsView(collection: projects)
```

[34] Https://github.com/nodejitsu/forever.

```
  $('body').append projectsView.render().el
  $("##{projectsView.id}").modal
    backdrop: 'static'
    keyboard: false

  projects.fetch()
```

En el fragmento de código anterior puede verse el método `index` que es llamado en este punto. Luego, la colección es pasada a una vista. Las vistas, como se explicó antes, son las encargadas de presentar datos al usuario (por medio de templates). Esta vista, básicamente, presenta una lista de proyectos y un formulario para crear uno nuevo (esta última funcionalidad se creará en la segunda etapa), como puede apreciarse en la Figura 12.

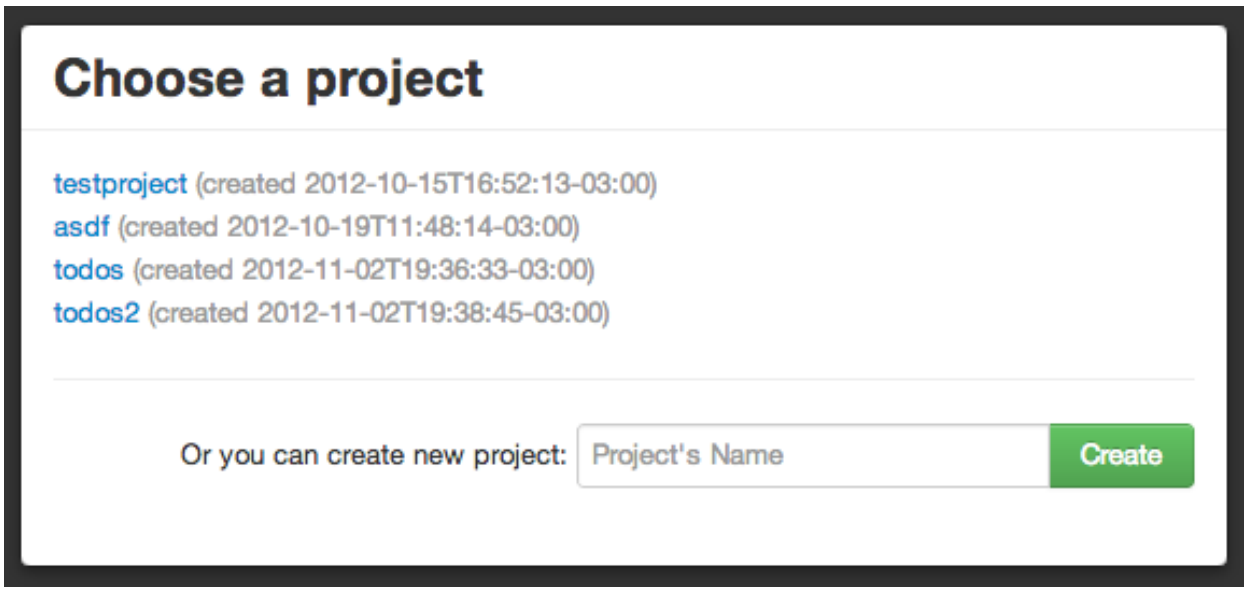

Figura 12: La ventana modal de selección de proyectos.

Al usuario hacer click en un proyecto existente, se llama a la segunda ruta en el enrutador:

```
routes:
  '': 'index'
  'projects/:id': 'project'

index: ->
  # Omitido por brevedad.
```

```
project: (id) ->
  app.project = new Project(id: id)
  app.project.fetch
    success: =>
      app.filebrowser.setModel app.project
      Backbone.Mediator.pub 'status:set', "Project Loaded"

  Backbone.Mediator.pub 'modal:hide'
```

Esta ruta, toma el identificador de proyecto de la url (que tienen un formato estilo `proyects/IDENTIFICADOR`) e instancia un proyecto usando ese identificador. Una vez que se haya obtenido toda su información desde el servidor, se le asigna el modelo a la aplicación (de manera que su instancia quede compartida) y se inicializa el visor de archivos (que es una vista) con éste.

El visor de archivos toma el modelo de proyecto y "pide" su carpeta raíz. El modelo hace una petición al backend, y éste contesta con una representación de cada archivo (o carpeta) del directorio raíz. La respuesta a esta llamada se presentó en la Sección 4.3.3.

El visor de archivos, instancia una vista de archivo por cada uno de los documentos que responde el backend. La vista de archivo se encarga de mostrar el nombre y tipo de archivo, y permitir al usuario clickearlo para abrirlo o ver su contenido. En caso de que el archivo se trate de un directorio, la vista de archivo se encarga de mostrar sus contenidos. Esta parte resultó ser un poco complicada de implementar, dado que lo que se necesita es básicamente mostrar el mismo tipo de vista que la raíz del proyecto pero para un subproyecto. Lo que se hizo en este caso es lo siguiente: cuando el usuario hace click en un directorio, se instancia una colección de archivos con la ruta a ese directorio. Se hace una petición al servidor para que entregue la lista de archivos en esa carpeta y se instancian vistas de archivo para cada uno, embebiéndolas en la misma vista del directorio que se acaba de clickear. En la Figura 13 se puede apreciar el concepto. La carpeta `app` es una vista de archivo, y contiene todos los archivos y carpetas dentro de su recuadro. Lo mismo pasa con el directorio `models`.

En caso de que el usuario haga click en un archivo, se le pasa la instancia del modelo al editor de código, el cual indica al modelo que debe pedir el contenido del archivo al servidor para mostrarlo. El editor de código utiliza CodeMirror (***ver tal y tal sección***) y básicamente muestra el contenido del archivo en el editor, que se encarga de formatearlo y resaltar sintaxis, entre otras responsabilidades. El editor se encarga además de actualizar el contenido del archivo al momento de guardar.

Para esta etapa de la construcción, se incluyó además la posibilidad de ensamblar y ejecutar el proyecto. Para esto, se agregaron métodos en el modelo de proyectos que hace llamadas al

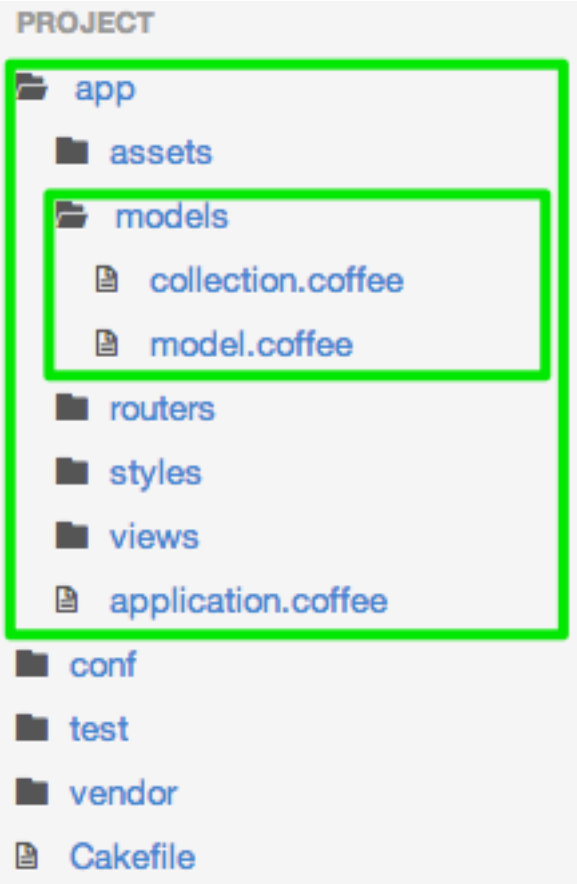

Figura 13: El visor de archivos: cada vista de directorio contiene más vistas de archivos de ser necesarias.

backend para ensamblar y ejectuar el servidor de pruebas. El evento es manejado por la barra de navegación, que incluye varios elementos de menú que por esta etapa se mantuvieron inactivos. A la derecha de la barra de navegación se encuentra un botón que permite ensamblar y ejecutar el proyecto con un sólo click, y otros dos que permiten ejecutarlo y ensamblarlo por separado.

En esta etapa se agregaron además atajos de teclado. Para esto se utilizó una librería Javascript llamada Mousetrap[35]. Esta librería permite configurar muy fácilmente atajos de teclado con una gran flexibilidad en términos de combinaciones de teclas. Por ejemplo, si se quisiera agregar un atajo de teclado para guardar el archivo actual, se puede hacer lo siguiente:

```
Mousetrap.bind ["ctrl+s", "command+s"], ->
  # Guardar el archivo
```

Esto permite que el usuario use la combinación "CTRL+S" o bien, en computadores Mac, "COMMAND+S". Es posible agregar atajos de teclado más complejos, como "CTRL+ALT+S" o "CTRL+SHITF+S", etc. Incluso, es posible saber si el usuario está manteniendo presionada alguna tecla. Por ejemplo, si se quiere saber si el usuario está manteniendo presionada la tecla "SHIFT", se puede hacer de la siguiente forma:

```
Mousetrap.bind ['shift'], (e) =>
      # El usuario está manteniendo presionada la tecla SHIFT
    , 'keydown'
```

Utilizando esta librería, se agregaron varios atajos de teclado en esta etapa. Entre ellos:

- Guardar archivo actual: `CTRL+S`
- Cerrar el archivo actual: `CTRL+W`
- Ejecutar el proyecto: `CTRL+R`
- Cambiar entre archivos abiertos: `CTRL+NUMERO`

Este último atajo mencionado permite al usuario cambiar entre los archivos que están abiertos en la lista de la derecha. La mayoría de los editores de código permiten cambiar rápidamente entre los archivos abiertos de esta manera.

[35] Link!

Casi todos los atajos de teclado son interpretados por el navegador. Por ejemplo, el atajo para guardar, para cambiar entre archivos abiertos, todos ellos son atajos que el navegador utiliza internamente. Para evitar que el navegador los intercepte, se utilizó la siguiente técnica:

```
Moustrap.bind ["ctrl+s"], (event) ->
  event.prevendDefault()

  # Guardar el archivo
```

Cada "evento" que es generado en Javascript, normalmente es pasado a los callbacks. En este caso, es posible llamar al método `preventDefault()` del evento, lo que indica al navegador que no realice la acción por defecto que debería. Si no se llamara a este método, el archivo de todas formas se guardaría, pues el evento se está llamando, pero el navegador también mostraría la ventana de "Guardar Página" que por defecto aparecería en otro caso, y eso no es lo que se quiere en una aplicación web de este estilo.

4.4. Construcción del Editor de Interfaces

El editor se diseñó de manera que en el centro se tuviera una vista en vivo de lo que se estaba construyendo, mientras que a la derecha se listaran todos los componentes disponibles para agregar al template. Dado que los templates son básicamente HTML, es el navegador el que se encarga de mostrar cómo se vería finalmente. Por esto, lo que se hizo fue agregar elementos a la lista de componentes de manera que al arrastrarlos hacia el centro (el editor), simplemente se agregue su representación en HTML y el navegador se encargaría de mostrar su "vista previa".

Entonces, en la lista de componentes se decidió agregar botones, tablas, formularios, campos de texto, entre otros, y dentro de ellos (en código, no visible para el usuario) agregar un fragmento de HTML que se agregaría al template. Entonces, utilizando jQuery UI[36], cada componente se convierte en un elemento arrastrable. Con jQuery UI, se necesita convertir elementos en "arrastrables" y además, crear elementos en donde "soltar" lo que el usuario está arrastrando. En este sentido, y, en un primer intento, se convierten todos los elementos en la vista previa en "soltables".

```
# Con la siguiente llamada, se convierte cada elemento en el editor
# de vistas en "soltable".
@$("*").droppable()
```

[36] Definir esto!

Con este primer acercamiento, ya se podía arrastrar y soltar componentes. El problema es que se podían arrastar componentes como botones y otras cosas dentro de elementos HTML que no correspondía, como imágenes, menús, etc. Para esto, se incluyeron ciertas excepciones a la llamada anterior, como sigue:

```
# Seleccionar todos los elementos, excepto los que están en la
# llamada .not()
@$("*").not('img, button, input, select, option, optgroup').droppable()
```

Con esto, se simplificó un tanto el arrastrado de componentes, evitando que algunos quedaran dentro de elementos que no correspondía. Ahora, se notó que era difícil saber dónde realmente se estaba dejando el componente que el usuario estaba arrastrando, por lo que se incluyó retroalimentación visual al momento de arrastar, es decir, cuando el usuario esté arrastrando el elemento, el elemento en donde "caería" el componente se rodea con un borde amarillo, como muestra la Figura 8. Esto se logra usando propiedades de jQuery UI:

```
exceptions = 'img, button, input, select, option, optgroup'
@$("*").not(exceptions).droppable
  hoverClass: "hovering" # Esto agrega una clase CSS con un borde.
```

La propiedad `hoverClass` agrega una clase CSS al elemento donde se estaría arrastrando el componente y la remueve al salir. Con esto se agrega un borde que facilite al usuario saber dónde caerá el componente.

Se decidió además agregar componentes que sólo sirven si se arrastran dentro de un formulario. En este punto, el usuario puede arrastrar estos componentes a cualquier parte, lo que hace de su uso algo complicado. Para solucionar esto, se implementó un sistema en el cual cada componente tiene especificado dónde puede ser arrastrado. Por ejemplo, los botones pueden ser arrastrados a cualquier parte:

```
<div class="switch-component" data-component-type="button">
  <div class="payload">
    <button type="button" class="btn btn-primary">Button</button>
  </div>

  <span class="name">Button</span>
</div>
```

En cambio, los elementos de un formulario sólo pueden arrastrarse a un formulario previamente colocado. En el siguiente fragmento se puede notar la propiedad `data-component-drop-only` que contiene una cadena de texto con selectores CSS en dónde puede ser agregado.

```
<div class="switch-component" data-component-type="label-button"
     data-component-drop-only="form">
  <div class="payload">
    <div class="control-group">
      <div class="control-label"><label for="new_input">Label</label></div>
      <div class="controls">
        <input type="text" name="new_input" id="new_input">
      </div>
    </div>
  </div>

  <span class="name">Form Input</span>
</div>
```

Lo anterior, junto con el siguiente Coffeescript, permite que los componentes puedan ser arrastrados sólo a ciertos elementos, si es que lo especifican:

```
makeDroppable: (only) ->
  # Si se especificó only, entonces usarlo, de lo contrario
  # permitir cualquier elemento.
  if only
    only = "#view_container #{only}"
  else
    only = "#view_container, #view_container *"

  @$(only).not(exceptions).droppable
    hoverClass: "hovering"
```

Hasta ahora, el editor de templates agrega los componentes anexándolos al final de la posición en la que el usuario las deja. Esto lo imposibilita de agregar componentes al principio de una lista por ejemplo. Por lo tanto, se agregoó la posibilidad de cambiar ese comportamiento. Al momento de arrastrar un componente, el usuario puede presionar (y mantener presionada) la tecla `SHIFT`, de manera que al dejar un componente, éste se anexe al principio en vez de al final, permitiendo al usuario agregar cosas al principio de listas o formularios, por ejemplo.

Por último, para facilitar aún más el arrastrado de componentes, se agregó una vista previa de cómo quedaría el componente que se está arrastrando una vez que se suelte. La idea es que al estar arrastrando un elemento, éste aparece en el editor con una ligera opacidad. Esto se logró usando algunas llamadas de jQuery UI:

```
# ...
@$(only).not(exceptions).droppable
  hoverClass: "hovering"
  greedy: yes
  drop: (e, u) ->
    self.putComponent(self, $(this), u, no)
  over: (e, u) ->
    self.putComponent(self, $(this), u, yes)
  out: (e, u) ->
    self.removeComponent()
```

Con estas llamadas, al momento de que el usuario esté sobre un elemento ("over"), literalmente se agrega el componente al editor, para luego eliminarlo en caso de salirse ("out"), o bien dejarlo definitivamente al soltarlo ("drop"). En la Figura 8 puede verse un ejemplo de este comportamiento.

Por último, el editor de templates también debería permitir al usuario editar el código directamente, en caso de que no exista algún componente o bien se necesite agregar cierta lógica más allá de HTML. Para esto, se utilizó el mismo editor de código que para los archivos normales. Se agregaron dos pestañas en la parte superior del editor de templates que permiten cambiar entre el editor visual y el código fuente (ver Figura 14).

Por último, existe una propiedad en HTML5 que permite al usuario editar cualquier parte de un sitio directo desde el navegador. Al habilitar esta propiedad en el canvas, el usuario puede cambiar etiquetas de formulario, o escribir directamente en el canvas sin tener que cambiar al editor de HTML para agregar texto. Para habilitar esta propiedad, basta con agregar el atributo `contenteditable` a la etiqueta del canvas:

```
<div id="view_container" contenteditable>
  <!-- El canvas -->
</div>
```

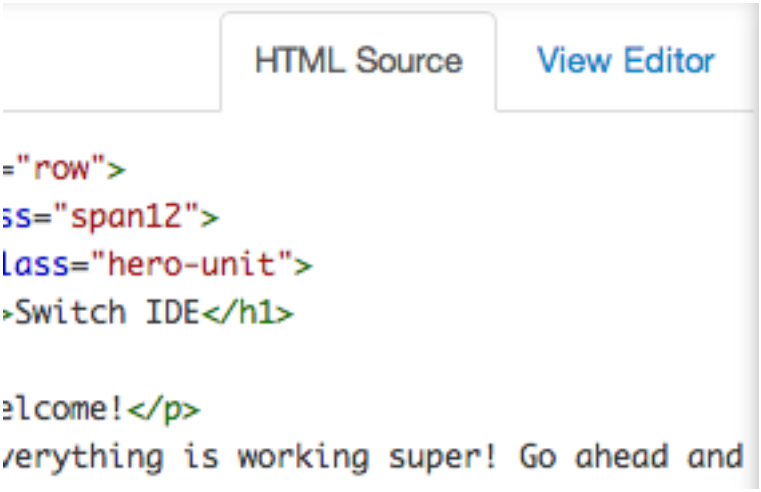

Figura 14: Estas pestañas permiten al usuario cambiar entre el modo visual y el editor HTML

5. Resultados

5.1. Producto Final

Explicar qué se logró, qué puede hacer

La herramienta que se logró en este trabajo permite desarrollar aplicaciones web completas en un navegador. Desde la creación del proyecto hasta la edición y ejecución de la aplicación, la solución es una IDE completa. Se logró incluir suficiente funcionalidad como para considerarse la solución buscada.

El componente más importante de Switch, el editor de interfaces, logra asemejarse bastante a lo que se puede encontrar en herramientas similares, como Xcode u otras de las mencionadas en la Sección 2. Es un editor de uso intuitivo y con una gran cantidad de componentes presentes en Twitter Bootstrap, lo que permite prototipar interfaces rápida y fácilmente.

Aun cuando el editor es de fácil uso, sigue estando apuntado a usuarios expertos (desarrolladores específicamente), y no a diseñadores u otras personas que deseen prototipar interfaces solamente. Por esto mismo es que el editor provee un modo de edición de HTML, para que el desarrollador pueda realizar cambios más "finos" en los templates que edite, además de tener la posibilidad de agregar más componentes que no estarían presentes en el listado.

Además, está la posibilidad de ensamblar y probar el proyecto desde la misma IDE. El desarrollador puede simplemente presionar "Build & Run" o usar el atajo de teclado `CMD+R` (`CTRL+R` en Windows y Linux) para que el programa ensamble y levante el servidor con el proyecto.

5.2. Modo de Uso de la Herramienta

En esta sección se describirá cómo utilizar la herramienta. Es importante mencionar que la solución propuesta está pensada para usuarios que ya tengan conocimientos para programar en Backbone, por lo que no se explicará cómo desarrollar con este framework.

5.2.1. Creación y Selección de Proyectos

Al ingresar por primera vez, al desarrollador se le presenta la ventana de selección de proyectos. En ella, se mostrarán los proyectos existentes y un pequeño formulario que le permitirá crear un proyecto nuevo.

Para abrir un proyecto existente, basta con clickear en uno de ellos (número 1 de la Figura 15). Si el usuario deseara crear un proyecto nuevo, puede escribir el nombre de éste en el formulario de abajo y presionar "Create" (número 2 de la Figura 15).

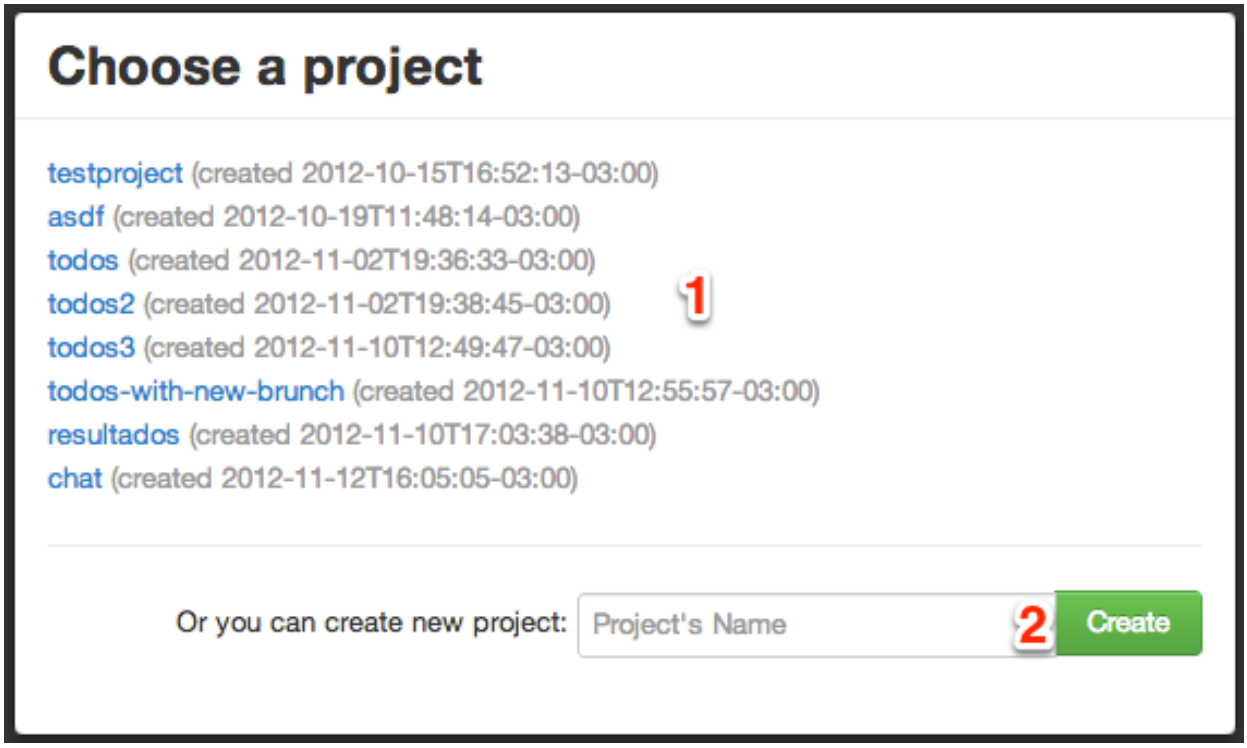

Figura 15: El selector de proyectos. El usuario puede abrir un proyecto existente (1) o crear uno nuevo (2).

En ambos casos, se abrirá el proyecto y el usuario será presentado con la vista principal de la aplicación.

5.2.2. Manejo de Archivos y Carpetas

El usuario puede realizar acciones básicas sobre los archivos. Para crear un archivo nuevo, basta con hacer click secundario sobre el directorio donde se desee crear uno y seleccionar "New File" (ver Figura 16). Se mostrará una ventana modal en donde el usuario podrá escribir el nombre del archivo nuevo, como se puede ver en la Figura 17. En caso de que el usuario escoja un nombre que se encuentre en uso, el sistema le alertará correspondientemente.

El procedimiento es casi idéntico para la creación de directorios, con la única diferencia de que el usuario deberá escoger "New Folder" desde el menú contextual que se muestra en la Figura 16.

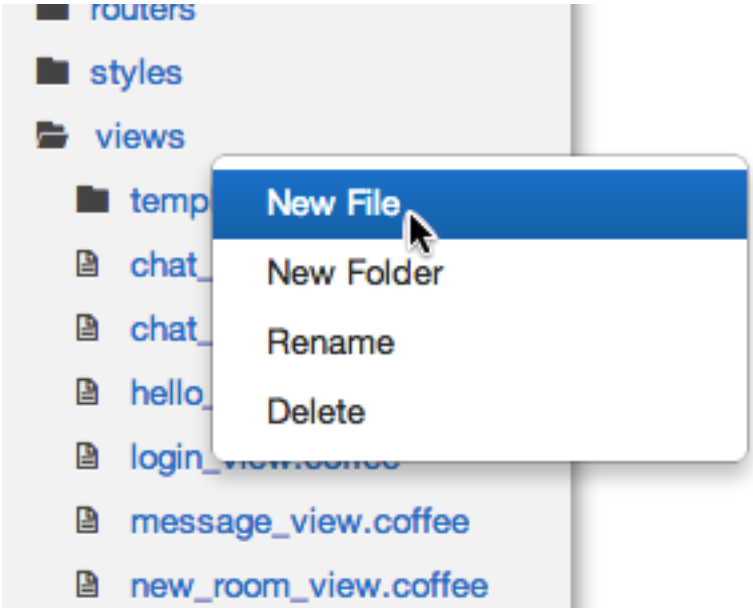

Figura 16: El menú contextual que aparece al hacer click derecho sobre un archivo o directorio.

Figura 17: La ventana modal que permite al usuario escribir el nombre de un archivo (o directorio) nuevo.

Para renombrar archivos y directorios, se debe hacer click derecho sobre éste y presionar "Rename". El nombre del archivo se convertirá en un campo de texto donde el usuario podrá escribir el nombre nuevo. Para finalizar el renombrado bastará con presionar la tecla ENTER. En la Figura 18 puede verse el campo de texto que aparece al renombrar un directorio.

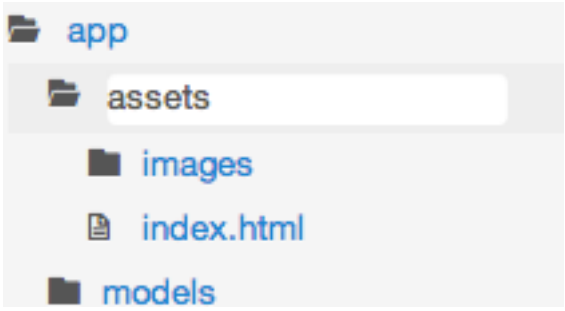

Figura 18: Al renombrar un directorio o archivo aparece un campo de texto para cambiar el nombre.

Para eliminar archivos y directorios, debe seleccionarse la opción "Delete" del menú contextual. Se presenta una confirmación, especificando qué archivo o carpeta se eliminará, como se ve en la Figura 19.

Figura 19: Se le pide confirmación al usuario antes de eliminar algún archivo o carpeta.

5.2.3. Edición de Archivos

Para abrir un archivo, se debe hacer click en él. Se cargará el editor de código a la derecha, mostrando los contenidos de éste. Ahí, el usuario puede hacer los cambios que sean necesarios. En la Figura 20 puede verse el editor de código al abrir un archivo.

Para guardar los cambios, el usuario puede presionar `CTRL+S` en el teclado. Una vez que el archivo se haya guardado correctamente, se verá un mensaje arriba a la derecha, en la barra de navegación (ver Figura 21).

En caso de que el usuario abriera otro archivo sin antes haber guardado los cambios, el sistema automáticamente guardará los cambios por él antes de cambiar al siguiente archivo.

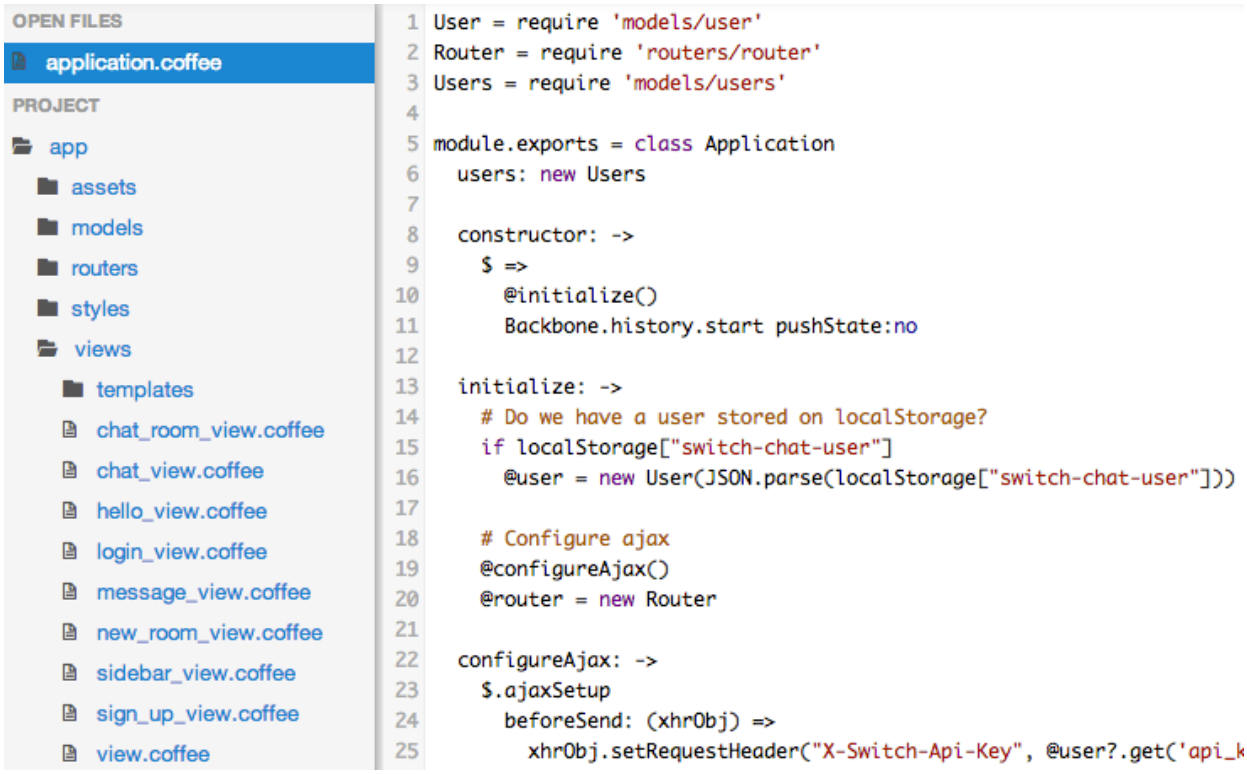

Figura 20: El editor de código al abrir el archivo `application.coffee` de un proyecto.

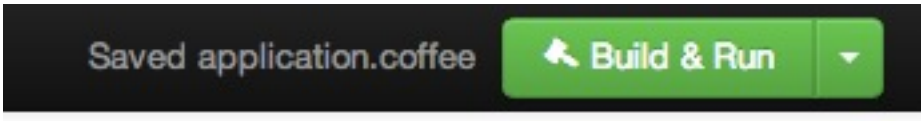

Figura 21: Mensaje que aparece en la esquina superior derecha al guardarse exitosamente un archivo.

5.2.4. Edición de Templates

Cuando el usuario hace click en un template (archivos denotados con un ícono específico en el explorador de archivos, ver Figura 22), se carga el editor de interfaces en vez del editor de código simple.

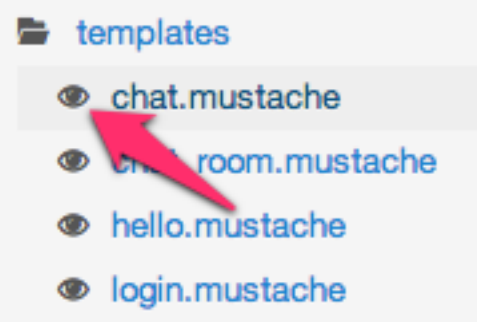

Figura 22: El ícono que diferencia los archivos de código de los templates.

El editor de interfaces muestra el contenido del template en vivo, y permite al usuario arrastrar widgets de la barra lateral derecha hacia el canvas. Por ejemplo, en la Figura 23, puede verse cómo se arrastra un componente llamado "Prepended Input" a un formulario de registro en el canvas.

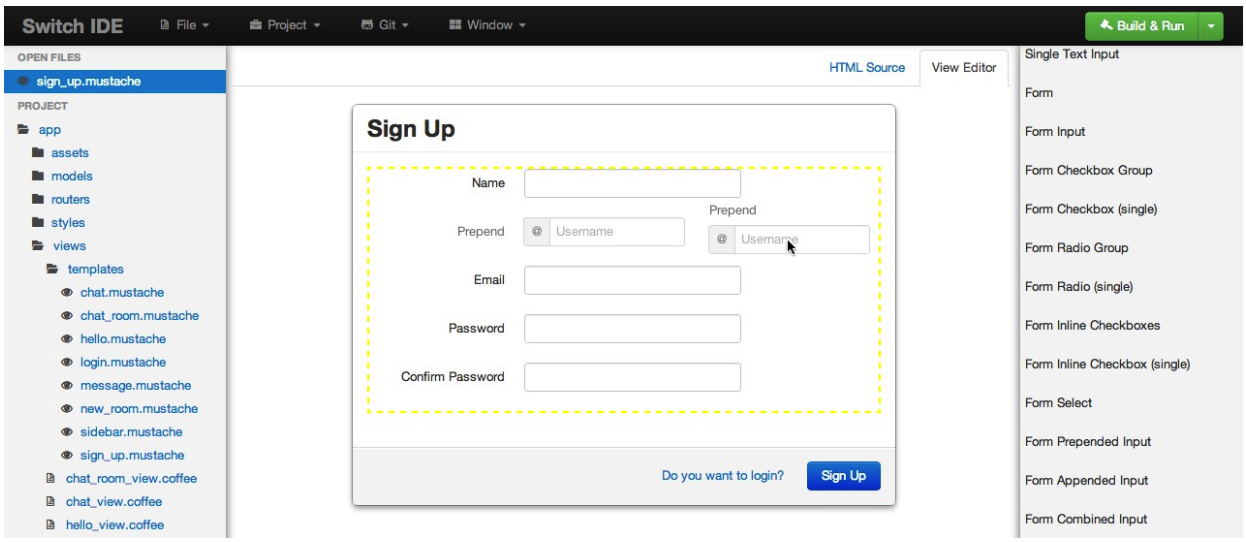

Figura 23: Arrastrando un componente hacia un formulario de registro en el editor de templates.

También es posible editar el HTML que se va generando con el editor. Por ejemplo, en caso de que el desarrollador desee agregar un atributo de clase o un identificador, puede hacerlo

usando el editor de HTML. Para acceder a él, es posible presionar las teclas `CTRL+ALT+FLECHA ARRIBA` o bien utilizar las pestañas presentes arriba del canvas (ver Figura 24).

Figura 24: En el recuadro pueden verse las pestañas que permiten cambiar entre el modo de edición de HTML y el editor visual.

5.2.5. Ejecutar el Proyecto

La ejecución del proyecto consiste en ensamblarlo y ejecutarlo. Existen varias formas de hacer esto. Si se desea ensamblar y ejecutar el proyecto de una vez, puede presionarse `CTRL+R` en el teclado o bien presionar el botón "Build & Run" en la esquina superior derecha. En ambos casos, se mostrará el progreso en la misma esquina, como puede verse en la Figura 25.

Figura 25: El botón para ensamblar y ejecutar, junto con la barra de progreso.

En caso de necesitar sólamente ensamblar o ejecutar (de forma independiente), es posible, clickeando la flecha al costado del botón "Build & Run". Esto mostrará un menú contextual con ambas opciones, como se ve en la Figura 26.

En caso de que el ensamblado falle, se mostrará una ventana con el detalle del error, como se puede ver en la Figura 27.

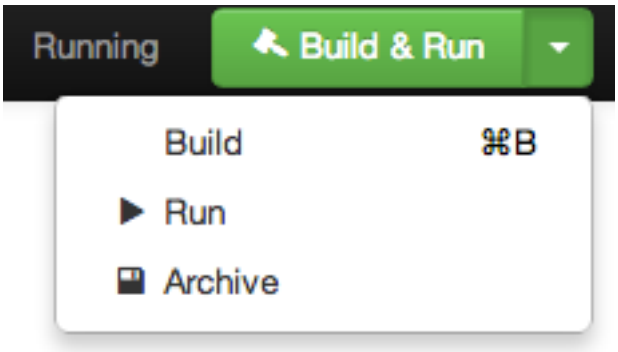

Figura 26: El menú contextual con las opciones para ensamblar, ejecutar y archivar el proyecto.

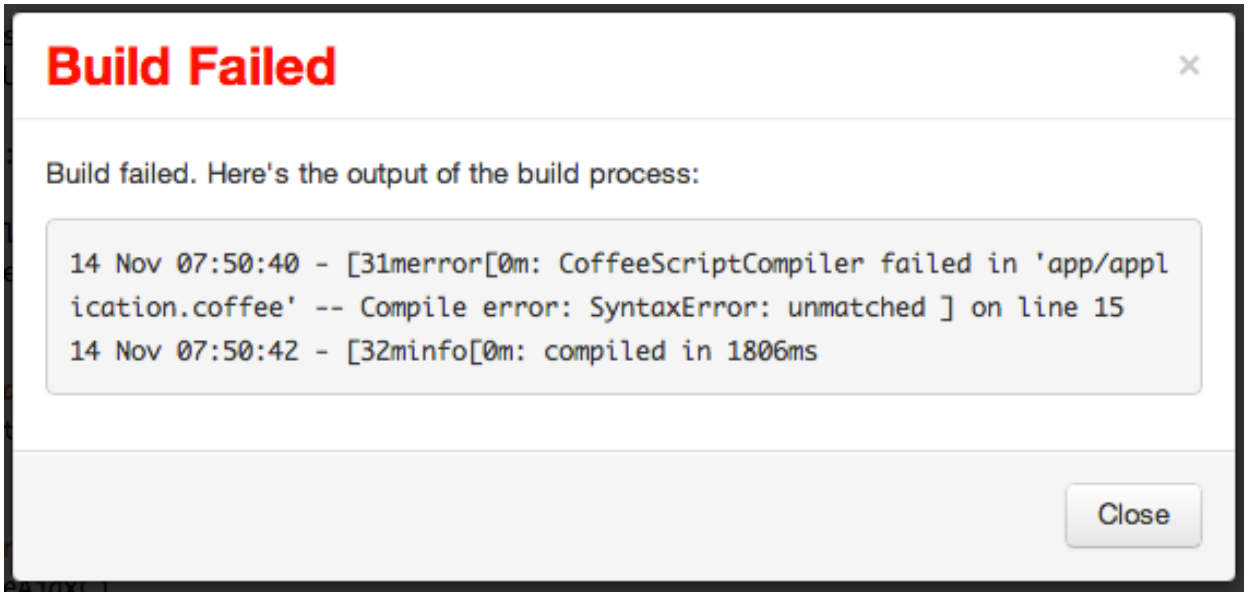

Figura 27: La ventana modal que se muestra al ocurrir un error de ensamblado.

5.2.6. Archivado del Proyecto

Una vez que el proyecto se encuentre finalizado o listo para ser subido a un servidor, se deberá "archivar". Esta opción ensambla el proyecto, con la diferencia de además minificar los archivos Javascript y CSS generados, efectivamente optimizándolos para producción. Al presionar esta opción, que puede verse en la Figura 26, el servidor ensambla y comprime el proyecto en un archivo ZIP, que luego comienza a descargarse.

5.2.7. Otras Características

Existen algunas características extra disponibles para el desarrollador, que no se discutieron en detalle en el proceso de construcción.

El usuario tiene la posibilidad de reordenar la lista de archivos abiertos, simplemente arrastrándolos. Esta característica se une a los atajos de teclado que permiten cambiar entre archivos abiertos, presionando las teclas `CTRL+NÚMERO` donde `NÚMERO` es un número entre 1 y 9.

Además, para cerrar un archivo, es posible presionar `CTRL+W`.

6. Conclusiones

A continuación se presentarán conclusiones del presente trabajo, con respecto a la solución lograda y conclusiones para trabajo futuro.

6.1. Sobre la Solución

Se cree que la solución que se presenta en este documento podría ayudar a disminuir los tiempos de desarrollo de aplicaciones web, en cierta medida facilitando la tarea que más tiempo consume. El prototipado y construcción de vistas en aplicaciones de cualquier tipo tiende a ser la más tediosa[37], especialmente en el ambiente web donde todo debe hacerse por código. Usando Switch IDE, se podría facilitar gran parte de esta tarea proveyendo al desarrollador con widgets utilizados comúnmente en el desarrollo de aplicaciones, como botones, formularios, tablas, entre otras.

6.2. Sobre la Elección de Herramientas

Se cree que la elección de herramientas para el backend fue bastante acertada, pues no presentaron problemas durante el desarrollo del proyecto

En cuanto al frontend, En un principio se escogió Backbone para construir el frontend por su simplicidad, popularidad y por ser un framework liviano. Resultó ser una herramienta flexible y simple de usar, con una documentación[38] adecuada que permitió desarrollar con ella sin mayores problemas.

6.3. Trabajo Futuro

Una característica que podría disminuir los tiempos de prototipado sería la de poder "unir" elementos con acciones en el editor de vistas. Por ejemplo, al agregar un botón, el desarrollador podría conectarlo con el controlador de manera que al hacer click se ejecutara una determinada acción. Por ejemplo, en Xcode[39] es posible presionar `CTRL` y arrastrar el componente hacia el código, generéndose una acción que se ejecutaría al presionar el componente (ver Figura 28).

Una de las características que podría facilitar el trabajo de un desarrollador con esta herramienta es control de versiones con Git. Agregar esta característica no sería trivial,

[37] Referencia a algo?
[38] Link a la docu.
[39] Ver tal y tal página, blabla.

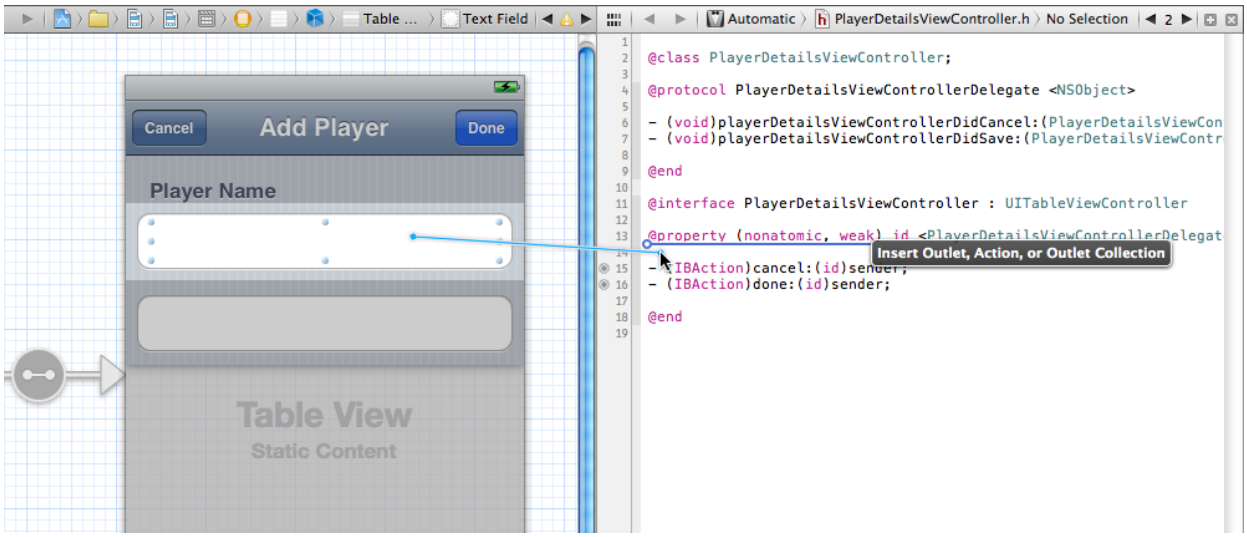

Figura 28: En Xcode, al presionar la tecla `CTRL` y arrastrar un widget, es posible crear métodos directamente.

pero tampoco sería extremadamente difícil dado que existen librerías que lo facilitarían. Agregar esta funcionalidad permitiría a los desarrolladores mantener su código versionado y además permitiría la colaboración, por medio de repositorios compartidos. Agregar esta funcionalidad requeriría de interfaces que permitan al desarrollador seleccionar archivos para agregar al repositorio, crear nuevos "commits" (como se le conoce a los estados de código en versionamiento), descargar y subir cambios a repositorios remotos y poder ver el historial del repositorio.

Otra característica importante que se ***discutió antes***[40] es la de no necesitar ensamblar el proyecto constantemente. Brunch (el ensamblador que se utilizó en este trabajo) permite levantar un proceso que observa cambios en los archivos y ensambla el proyecto bajo demanda. De poseer esta característica en la solución, el proceso de programar y probar los cambios podría tornarse más continuo y simple. Sin embargo, esto requeriría cambiar la forma en la que se levanta el servidor de pruebas, de manera de mantenerlo siempre en línea.

Una adición que se cree agilizaría el trabajo en proyectos con archivos grandes es la de no enviar el archivo completo cada vez que se guarden cambios. Este es el comportamiento actual y, si bien no se nota para archivos livianos, sí se sentiría al momento de guardar archivos relativamente grandes. Una posible solución sería enviar parches, es decir, guardar en el frontend el estado en el que se encuentra el archivo en el servidor, y, al momento de guardar cambios, enviar sólo las diferencias, optimizando la cantidad de información enviada. Por ejemplo, si se tuviera un archivo con 200 líneas de código y sólo se quisiera agregar 2

[40] La discutí antes??

líneas con comentarios, sólo se tendría que transmitir aproximadamente un 1 % del archivo.

En cuando a optimizaciones, si bien Javascript es un lenguaje con recolección de basura y el desarrollador no debería preocuparse del manejo de memoria, en Backbone es muy fácil comenzar a fugar ("leak", en inglés) objetos a memoria. Esto es porque la recolección de basura sólo elimina objetos de memoria cuando ya no existen referencias a éste. En Backbone, al haber tantos objetos dependiendo de otros y escuchando notificaciones de otros, aun cuando se haya eliminado una vista del documento, el objeto seguirá escuchando eventos de otros objetos y seguirá en memoria. Siendo este trabajo un proyecto de Backbone relativamente grande, la posibilidad de empezar a fugar objetos es alta. Existen algunas guías para mejorar el rendimiento de aplicaciones escritas en Backbone como la escrita por la empresa Paydirt[41], que convendría seguir y aplicar en este proyecto y que se cree ayudarían a mejorar su rendimiento.

En cuando a la edición de templates, en la solución propuesta no es posible cambiar propiedades de los diferentes widgets visualmente (como por ejemplo cambiar el color a un botón, o cambiar ciertas propiedades de una tabla). Se propone agregar un editor de propiedades que permita cambiar diferentes atributos de algún componente seleccionado, de manera de evitar que el desarrollador deba editar HTML directamente.

En la misma línea, se cree que cambiar el framework que se utiliza para el desarrollo por una conocida como Knockback[42] podría facilitar el trabajo del desarrollador aún más al editar templates. Knockback es una combinación de dos frameworks: Backbone (la que se utiliza actualmente) y Knockout. Knockout es conocido por ofrecer "bindings", es decir, permite agregar atributos HTML a las vistas de manera que se actualicen automáticamente, sin necesidad de escribir código dentro de ellas. Como por ejemplo:

```
<p>First name: <strong data-bind="text: firstName"></strong></p>
<p>Last name: <strong data-bind="text: lastName"></strong></p>
```

El fragmento anterior une las etiquetas `<strong>` con los atributos `firstName` y `lastName` del modelo asociado. Lo ventajoso es que es simple HTML y además las uniones son en tiempo real, lo que significa que cualquier cambio al modelo ocurrirá en la vista sin requerir ningún esfuerzo extra por parte del desarrollador.

Esto podría mejorar la experiencia del desarrollador al momento de diseñar vistas dado que no necesitaría escribir código (sólo HTML). Además, y más importante, esto permitiría mejorar el editor de interfaces de manera de que el usuario agregue "bindings" visualmente, como por ejemplo con menús contextuales, sin editar el código fuente de la vista.

[41] Https://paydirtapp.com/blog/backbone-in-practice-memory-management-and-event-bindings/.
[42] Referencia.

Agregar este framework podría significar un esfuerzo mayor, aunque Brunch facilita esta tarea permitiendo crear "esqueletos" para proyectos con este framework. Exportar proyectos ya existentes no sería trivial, pero tampoco sería muy complejo, dado que Knockback simplemente combina ambos frameworks.

7. Referencias

[1] Steve Burbeck, "Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)" Disponible: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html. Última Revisión: 09/07/2012.

[2] Mike Potel, "MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java" Disponible: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf. Última Revisión: 09/07/2012.

[3] DocumentCloud, "Backbone" Disponible: http://backbonejs.org. Última Revisión: 09/07/2012.

[4] Backbone, "Projects and Companies using Backbone" Disponible: https://github.com/documentcloud/bac and-Companies-using-Backbone. Última Revisión: 09/07/2012.

[5] Cappuccino Project, "Cappuccino Framework" Disponible: http://www.cappuccino-project.org/. Última Revisión: 09/07/2012.

[6] Apple Inc., "Xcode" Disponible: https://developer.apple.com/xcode/. Última Revisión: 09/07/2012.

[7] Sencha Inc., "Ext JS" Disponible: http://www.sencha.com/products/extjs/. Última Revisión: 09/07/2012.

[8] Sencha Inc., "Ext JS Releases" Disponible: http://www.sencha.com/products/releases/. Última Revisión: 09/07/2012.

[9] Sencha Inc., "Buy Ext JS 4 Licenses and Support" Disponible: http://www.sencha.com/store/extjs/. Última Revisión: 09/07/2012.

[10] Sencha Inc., "Sencha Architect" Disponible: http://www.sencha.com/products/architect/. Última Revisión: 09/07/2012.

[11] Sencha Inc., "Buy Sencha Architect" Disponible: http://www.sencha.com/store/architect/. Última Revisión: 09/07/2012.

[12] Divshot, "Divshot" Disponible: http://www.divshot.com. Última Revisión: 09/07/2012.

[13] Twitter, "Twitter Bootstrap" Disponible: http://getbootstrap.com. Última Revisión: 09/07/2012.

[14] Ian Hickson (editor), "HTML5 Specification" Disponible: http://www.w3.org/TR/2011/WD-html5-20110525/. Última Revisión: 09/07/2012.

[15] eXo Platform SAS, "eXo Cloud IDE" Disponible: http://cloud-ide.com. Última Revisión: 09/07/2012.

[16] Git Project, “Git” Disponible: http://git-scm.com. Última Revisión: 09/07/2012.

[17] VMware Inc., “Wavemaker” Disponible: http://wavemaker.com. Última Revisión: 09/07/2012.

[18] Zoho Corp., “Zoho Creator” Disponible: http://www.zoho.com/creator/. Última Revisión: 09/07/2012.