



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Departamento de Sistemas Informáticos y Computación

# Rediseño de aplicaciones utilizando las tecnologías modernas para el desarrollo web en su parte Front-end

Trabajo de fin de Máster

Máster Universitario en Ingeniería del Software, Métodos Formales y Sistemas  
de Información

**Autor:** German Popoter

**Director:** Patricio Letelier Torres

Septiembre de 2015



# Índice

<b>TABLA DE FIGURAS.....</b>	<b>6</b>
<b>INTRODUCCIÓN.....</b>	<b>8</b>
<b>OBJETIVOS .....</b>	<b>11</b>
<b>CAPÍTULO 1. DESARROLLO DE IU Y UX WEBS .....</b>	<b>13</b>
Desarrollo de Interfaces.....	13
Desarrollo de Interfaces Web y Experiencia de Usuario.....	14
Desarrollo de Interfaces Multi-Plataforma.....	18
<b>CAPÍTULO 2. PAUTAS PARA DESARROLLO IU WEB MODERNAS.....</b>	<b>20</b>
Evitar Scroll Horizontal .....	20
Responsive Design.....	20
Mobile First.....	21
Elementos y grids con pocas columnas o en formato <i>card</i> .....	21
Comunicación entre cliente y servidor en tiempo real.....	23
Retroalimentación de progreso ( <i>Feedback</i> ) .....	26
Seguridad del lado del cliente .....	26
Seguridad del lado del servidor .....	27
<b>CAPÍTULO 3. ECOSISTEMA DE TECNOLOGÍAS UTILIZADAS .....</b>	<b>29</b>
Angular JS 1.3 .....	29
Node JS.....	30
Express JS .....	31
Local Storage .....	32
Jade .....	33
Grunt JS.....	33

<b>Bower</b> .....	<b>34</b>
<b>Bootstrap CSS</b> .....	<b>34</b>
<b>Underscore JS</b> .....	<b>35</b>
<b>Stylus</b> .....	<b>35</b>
<b>Git</b> .....	<b>35</b>
<b>Git Hooks</b> .....	<b>37</b>
<b>Bitbucket</b> .....	<b>37</b>
<b>Flujo de trabajo en Git (<i>Branching Model</i>)</b> .....	<b>37</b>
<b>Heroku</b> .....	<b>39</b>
<b>CAPÍTULO 4. CASO DE ESTUDIO</b> .....	<b>41</b>
<b>TUNE-UP Process</b> .....	<b>41</b>
Planificador Personal (PEP) .....	42
Comunicación y Mensajes .....	42
Alertas, Notificaciones y Anuncios .....	43
<b>Limitaciones de la herramienta Tune-Up Process</b> .....	<b>43</b>
<b>Propuesta para el prototipo</b> .....	<b>45</b>
<b>CAPÍTULO 5. PROTOTIPO</b> .....	<b>48</b>
<b>Tarjetas para mostrar UT (Cards)</b> .....	<b>48</b>
<b>Cambio de Título de una Actividad desde la Tarjeta</b> .....	<b>48</b>
<b>Inicio de Sesión</b> .....	<b>49</b>
<b>Seguridad</b> .....	<b>51</b>
<b>Diseño Responsive</b> .....	<b>52</b>
<b>Buscador General</b> .....	<b>55</b>
<b>Drag&amp;Drop para reorganizar</b> .....	<b>57</b>
<b>Creación de UT</b> .....	<b>58</b>
<b>Enviar Mensaje</b> .....	<b>60</b>
<b>Evolución del Prototipo</b> .....	<b>61</b>
<b>Generación de CSS y optimización de código Javascript</b> .....	<b>64</b>

<b>Comparación prototipo y versión actual de TUNE-UP .....</b>	<b>65</b>
Tiempo de Primera Respuesta y Tiempo total de Carga.....	66
 <b>CONCLUSIONES Y TRABAJO FUTURO .....</b>	 <b>71</b>
 <b>REFERENCIAS.....</b>	 <b>74</b>

# Tabla de Figuras

<i>Figura 1: Modelo de aplicación Web clásica comparado con modelo de Web 2.0 AJAX.....</i>	<i>9</i>
<i>Figura 2 Caso de estudio de Carlos Jiménez [13], donde estudia cómo mejorar la interfaz .....</i>	<i>16</i>
<i>Figura 3 Versión final del rediseño para mejorar la confianza en [13] .....</i>	<i>17</i>
<i>Figura 4: Ejemplo de como Twiter utiliza los cards en diferentes dispositivos [16].....</i>	<i>22</i>
<i>Figura 5: Muestra como los cards se puede acomodar al dispositivo donde se muestran [16].....</i>	<i>23</i>
<i>Figura 6: Muestra la comunicación entre cliente-servidor utilizando Short-Polling [18].....</i>	<i>24</i>
<i>Figura 7: Muestra la comunicación entre cliente-servidor usando Long-Polling. La conexión se queda abierta hasta que hay data nueva que enviar. [18] .....</i>	<i>25</i>
<i>Figura 8 Estructura de carpetas de la aplicación de Angular JS.....</i>	<i>30</i>
<i>Figura 9 Arquitectura de la aplicación de Node JS y Express JS .....</i>	<i>31</i>
<i>Figura 10: Ejemplo de uso de Cuadrícula del framework Bootstrap.....</i>	<i>34</i>
<i>Figura 11: Flujo de trabajo donde las ramas tienen código separado según el estado en que se encuentran.....</i>	<i>36</i>
<i>Figura 12: Imagen que muestra gráficamente los commits en las ramas develop y master .....</i>	<i>38</i>
<i>Figura 13: Gráfico que muestra el ciclo de vida de una rama de soporte .....</i>	<i>38</i>
<i>Figura 14: Vista inicial al entrar a la aplicación de Tune-Up Process .....</i>	<i>41</i>
<i>Figura 15: Panel del lado izquierdo de la herramienta que muestra las Alarmas, Notificaciones, Anuncios y Mensajes pendientes. ....</i>	<i>43</i>
<i>Figura 16: Barra de estado difícil de ver que se encuentra en el lado inferior izquierdo y que muestra la ultima fecha de actualización de datos. ....</i>	<i>44</i>
<i>Figura 17: Vista del PEP que tiene navegación en 2 dimensiones por la cantidad de columnas de datos. ....</i>	<i>44</i>
<i>Figura 18: Diseño de un card encontrado en el prototipo de Tune-Up Process.....</i>	<i>48</i>
<i>Figura 19: Muestra el texto en su forma para ser editado.....</i>	<i>49</i>
<i>Figura 20: Inicio de sesión cuando todavía no se ha determinado los sitios del usuario. ....</i>	<i>50</i>
<i>Figura 21: Inicio de sesión cuando el usuario tiene más de un sitio activo .....</i>	<i>50</i>
<i>Figura 22: Vista de escritorio.....</i>	<i>53</i>
<i>Figura 23: Vista en una Tableta.....</i>	<i>53</i>
<i>Figura 24: Vista en un Teléfono Inteligente.....</i>	<i>54</i>
<i>Figura 25: Ejemplo de búsqueda general .....</i>	<i>56</i>
<i>Figura 26: Muestra como funciona el Drag &amp; Drop para reorganizar el orden de las actividades... ..</i>	<i>58</i>
<i>Figura 27: Ventana modal para crear una UT nueva .....</i>	<i>59</i>
<i>Figura 28: Menú con acciones directas a un usuario .....</i>	<i>60</i>
<i>Figura 29: Ventana modal para enviar un mensaje nuevo desde cualquier parte de la aplicación... ..</i>	<i>60</i>
<i>Figura 30: Uno de los primeros bocetos en papel de como reorganizar las actividades .....</i>	<i>61</i>
<i>Figura 31: Primera versión funcional del prototipo .....</i>	<i>61</i>
<i>Figura 32: Primer rediseño de cards que se parece al boceto en papel.....</i>	<i>62</i>
<i>Figura 33: Versión con Cards más pequeños.....</i>	<i>63</i>
<i>Figura 34: Versión final del prototipo, "Buscar" se mueve a la barra de arriba .....</i>	<i>63</i>
<i>Figura 35 Gráfica que muestra los tiempos de el primer feedback al usuario.....</i>	<i>68</i>
<i>Figura 36 Gráfica que muestra los tiempos de carga total de la aplicación y los datos.....</i>	<i>68</i>
<i>Figura 37 Muestra el promedio y mediana de los tiempos del experimento.....</i>	<i>69</i>
<i>Figura 38 Muestra la diferencia en porcentaje entre los tiempos promediados de carga total entre TUNE-UP y el prototipo desarrollado.....</i>	<i>69</i>



# Introducción

Las aplicaciones de escritorio (y también aplicaciones web desarrolladas hace años) que quieren mantenerse vigentes tienen que afrontar en su migración a tecnologías y diseños modernos. Sin embargo, el reto es aún mayor pues no se trata de hacer lo mismo con otra tecnología, sino que las funcionalidades pueden verse modificadas para adaptarse y aprovechar las nuevas tendencias de diseño en Web.

En los últimos años hemos visto cómo el diseño de interfaces web ha cambiado. La tendencia es utilizar interfaces minimalistas en cuanto a contenido y muy volcadas en ofrecer una agradable experiencia de usuario. Han proliferado frameworks específicos para el desarrollo Web del *front-end* y se está imponiendo el trabajar usando diversas combinaciones de tecnologías en la parte *front-end*.

A medida que se van creando más dispositivos para el consumidor final para acceder a internet, se incrementa la problemática de cómo mostrar correctamente el contenido de una forma óptima para cada dispositivo, dicho problema afecta la experiencia de usuario a través de los distintos dispositivos. Estos no solo tienen sistemas operativos distintos sino que el tamaño de pantalla y resolución varía mucho entre dispositivos similares. Esta problemática da como resultado el desarrollo del *Diseño Web Responsivo* (*Responsive Web Design* en inglés), término nombrado por primera vez por Ethan Marcotte en un artículo para la revista digital "A List Apart" [1]. Responsive Web Design en esencia combina los conceptos de ancho de elementos flexibles, imágenes flexibles y *Media Queries* («módulo de CSS3 que permite presentar el contenido ajustado a un rango específico del dispositivo donde se va a mostrar sin tener que cambiar el contenido en sí mismo» [2]).

La experiencia de usuario no solo se basa en cómo se muestra la información o cómo se adapta el contenido en una aplicación web, también influye cómo se actualiza el contenido, los tiempos de carga y la fluidez de la aplicación. Las aplicaciones web clásicas siguen al pie de la letra el funcionamiento del protocolo HTTP, con cada cambio en la aplicación web esta tiene que enviar el formulario con la información al servidor, el servidor procesa la información y envía una respuesta al cliente y luego este refresca la página completa para mostrar los nuevos datos.



Este tipo de interacción con una aplicación no es la más óptima, ya que cada vez que se actualiza la aplicación se envía y descarga mucha información del servidor; lo cual se traduce en tiempo de espera.

Con el desarrollo de AJAX (Asynchronous JavaScript + XML ) [3] se introdujo una nueva forma de crear interfaces en aplicaciones web. Con AJAX se puede enviar y obtener información de forma asíncrona sin tener que refrescar la página que se le muestra al usuario.

Este cambio en paradigma se conocen como Web 2.0, y se hizo famoso cuando grandes productos del internet (como GMail o Google Maps [3]) lo adoptaron.

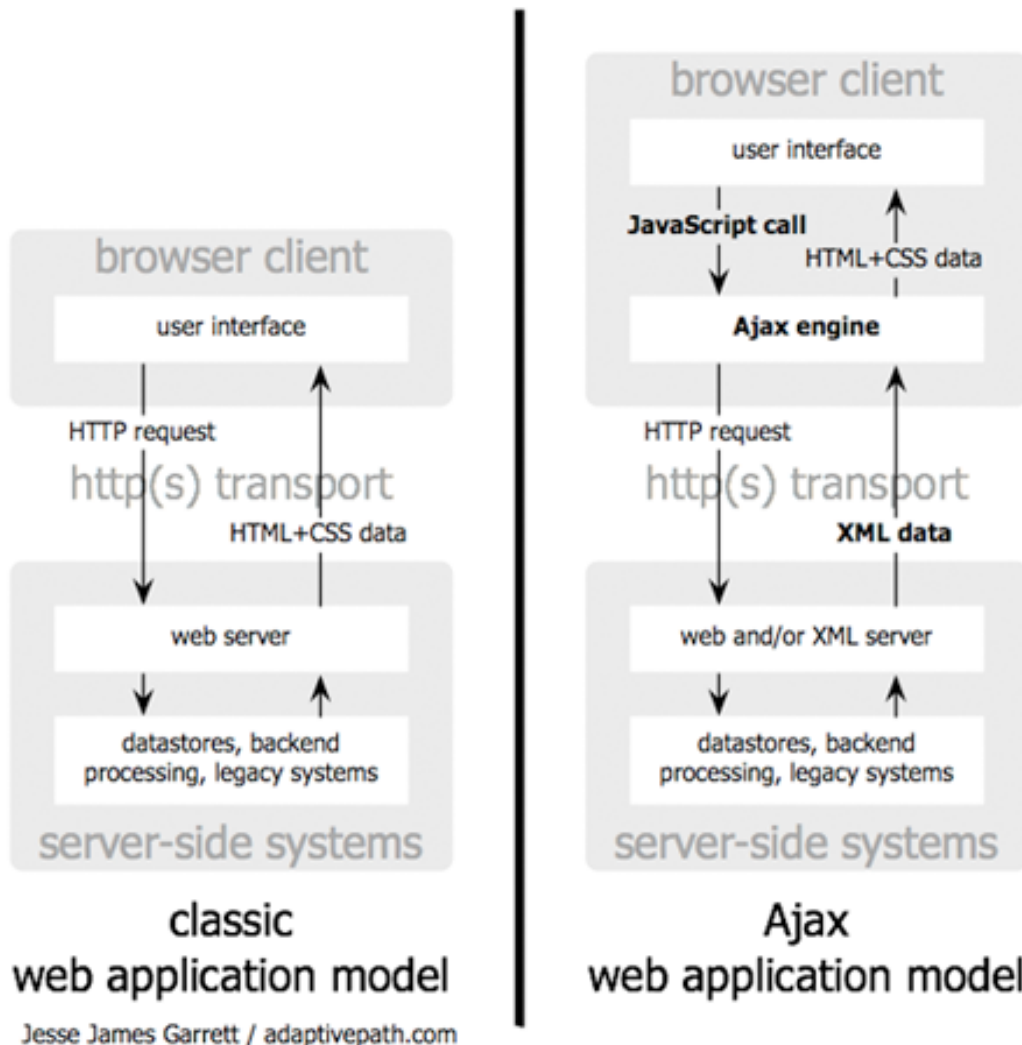


Figura 1: Modelo de aplicación Web clásica comparado con modelo de Web 2.0 AJAX

Para hacer posible la comunicación del cliente con el servidor, hay que definir una serie de servicios web que su función será la de recibir información,

procesar y retornar la respuesta en un estándar específico. Esta interfaz en las aplicaciones web es conocida como Interfaz Web de Programación de Aplicaciones (o su nombre en inglés Web API), esta interfaz no es más que una forma de comunicación entre un cliente de cualquier tipo (ya sea un navegador web, una aplicación móvil, u otro servidor) con el servidor que es encargado de gestionar la información. Por lo general un Web API no retorna HTML, sino que retorna datos estructurados en algún lenguaje de marcado o de modelado, comúnmente la información se envía en formato XML o JSON.

Dada la necesidad de las empresas de conectar los diferentes servicios y procesos nace lo que es la Arquitectura Orientada a Servicios (SOA, siglas en inglés de *Service Oriented Architecture*), esta es una arquitectura de software que facilita la interconexión de servicios de la misma o de diferentes empresas donde cada componente con el que se va a interactuar tiene su propio servicio Web o Web API. Esto permite conectar servicios independientemente de la tecnología utilizada para su creación.

Gracias a la madurez tecnológica y buena adaptación que SOA ha tenido en el desarrollo web al pasar los años, se ha podido crear una nueva arquitectura para crear aplicaciones web conocido como SOFEA [4] (siglas del inglés *Service-Oriented FrontEnd Architecture*, que en Español sería Arquitectura FrontEnd Orientada a Servicios), este término fue introducido por primera vez por en el libro "Life Above The Service Tier" [4]. Este tipo de arquitectura también es conocido al día de hoy como Aplicación de Página Única (del inglés *Single Page Application*)

Este tipo de arquitectura de FrontEnd tomó auge a partir del 2009 cuando frameworks del lenguaje JavaScript como Angular JS, Knockout JS y Node JS hicieron su aparición y empezaron a crear una comunidad alrededor de este tipo de aplicación.

# Objetivos

El objetivo de este trabajo de tesis es establecer un compendio de guías, pautas o tendencias actuales en el desarrollo de interfaces web. Para lograr esto se va a evaluar desde el punto de vista de usabilidad, eficiencia y contexto en que se encuentre la aplicación. Siendo la motivación fundamental del trabajo el interés de migrar aplicaciones desktop o de escritorio a aplicaciones web.

El desarrollar una aplicación web como un rediseño de una aplicación de escritorio tiene muchas ventajas. Algunas de estas ventajas mencionadas en [5] son las siguientes:

- **No hay necesidad de descargar ni de instalar nada.** Solo es necesario acceder a la dirección web desde cualquier dispositivo.
- **Actualizaciones automáticas.** Asegura que todas las personas utilicen la última versión de tu aplicación.
- **Posibilita un mejor modelo de negocio.** Las aplicaciones de escritorio están asociadas a pagar una sola vez y descargarlas, mientras que una aplicación web se puede habilitar un pago recurrente para darle acceso al usuario.
- **Acceso a analíticas de uso.** Es más sencillo tener acceso a información de uso de los usuarios y entender mejor que cosas mejorar para estos.
- **Desarrollo multiplataforma menos costoso.** Una aplicación web resulta menos costosa de desarrollar en el sentido que una misma aplicación puede ejecutarse donde sea que exista un navegador web y el dispositivo esté conectado a internet.

Se desarrollará un prototipo para la aplicación *desktop* TUNE-UP. La idea es aplicar en un producto software ya existente elementos de interfaces de desarrollo web modernos, planteando prototipos para ilustrar las ventajas y los retos que presenta dicha reestructuración.

En el prototipo se utilizarán tecnologías web que son tendencia actualmente, como lo son, Node JS, Angular JS, HTML5, CSS3 y Git



# Capítulo 1. Desarrollo de IU y UX webs

Una interfaz es el punto de interacción con un objeto, independiente de ya sea un objeto físico o un objeto digital.

Esta tiene una serie de características que definen su facilidad de uso y que tan intuitiva resulta para el usuario que la va a utilizar.

La interacción del usuario con alguna plataforma genera en este una percepción global del producto dependiendo de su experiencia, a esta percepción se le conoce como Experiencia de Usuario (del inglés *User Experience* o *UX*). Según [6] la «Experiencia de usuario (UX) es un término para el nivel de satisfacción total de usuarios cuando utiliza tu producto o sistema» [6]

El estudio de diseño de interfaces se basa en las investigaciones realizadas en el área de HMI (del inglés Human-Computer Interaction), que en sí se enfoca en las habilidades, discapacidades y procesos cognitivos de una persona, dejando a un lado la experiencia física y emocional del usuario final [7]. HCI es un área multidisciplinar. «Entre las disciplinas sobre las que se sustenta podemos enumerar la psicología cognitiva y de la conducta, ergonomía, antropología, sociología y ciencias de la computación entre otras» [8]

Siendo UX una parte fundamental en HCI, al conjunto de estas 2 se le suele llamar Ingeniería de Usabilidad (*Usability engineering*), esta área en específico es que pretende estudiar el diseño de experiencia de usuario, la eficacia, eficiencia y satisfacción al utilizar el producto.

«La principal razón por la cual aplicar la Ingeniería de Usabilidad cuando se desarrolla un sistema software, es la obtención de un sistema que hace al usuario más productivo, aumentando su eficiencia y satisfacción al utilizarlo.» [9]

Cabe destacar que desde la perspectiva del negocio, si una aplicación no cumple con las necesidades de los usuarios finales, esta no satisface con los objetivos de la empresa que brinda el servicio.

## Desarrollo de Interfaces

Muchas interfaces resultan difícil de entender y de aprender, y muchas veces causan confusión a sus usuarios, ya que no muestran consistencia para realizar tareas similares.

Junto con la consistencia existe un número de características que una buena interfaz debe tener para ser entendida y utilizada de manera fácil. Según [10], Don Norman describe una serie de principios que debe tener cada objeto que está diseñado para tener una interacción con un humano.

- **Visibilidad:** Lo visible que sea una función hace más posible que un usuario sepa que acción hacer a continuación. Funciones que están fuera de la vista hacen que el usuario no las encuentre o se sienta perdido.
- **Retroalimentación:** Es devolver una información sobre qué acción se ha realizado. Este puede ser sonido, táctil, verbal, visual o combinadas.
- **Restricciones:** Se trata de limitar las formas en que el usuario puede interactuar en un debido momento. Este se puede realizar de varias formas.
- **Mapeo:** Esto se refiere a la relación entre un objeto y el efecto que tiene en el mundo.
- **Consistencia:** Esto se refiere en diseñar interfaces que tengan elementos similares para realizar operaciones o tareas similares.
- **Affordance:** Es un término que se le da a un objeto cuando este sugiere como debe ser usado, independiente de la estética el objeto.

## Desarrollo de Interfaces Web y Experiencia de Usuario

El diseño de experiencia de usuario debe comenzar por el usuario que lo utiliza, pero según [11] que menciona que para realizar el mejor diseño de UX hay que prestar atención a los que los usuarios hacen y nos a lo que los usuarios dicen que hacen.

«El comportamiento emocional del usuario es resultado de tres factores diferentes: las emociones evocadas por el producto durante la interacción, el estado de humor del usuario y los sentimientos pre-asociados por el usuario al producto.» [8]

Según [12] para tener una buena experiencia de usuario se debe tener en cuenta lo siguiente y al mismo tiempo responder las preguntas:

- **Discoverability** (Capacidad de descubrir): ¿Puede el usuario descubrir cómo realizar diferentes tareas la primera vez que interactúan con el producto?
- **Learnability** (Capacidad de aprender): ¿Puede el usuario aprender de forma fácil como moverse dentro del sistema o producto? ¿Y al repetir la visita, puede recordar cómo utilizar la aplicación?
- **Eficiencia:** Una vez el usuario aprende a usar la herramienta, ¿puede este realizar las tareas repetitivas de forma fácil y rápida?
- **Rendimiento del Sistema:** ¿Qué tan ágil responde el sistema cuando el usuario interactúa con este?

- **Delight** (Agrado): ¿A los usuarios le agrada el producto?

El poder brindar una buena experiencia de usuario resulta en algo importante para tener en cuenta al momento de diseñar un producto de software, Yusef Hassan Montero [8] a forma de resumen menciona lo siguiente sobre Experiencia de Usuario [8]:

- Es resultado de un fenómeno interactivo en el que intervienen multitud de factores: individuales, sociales, culturales, contextuales y propios del producto.
- Se verá influida por expectativas y experiencias previas, y por tanto condicionará expectativas y experiencias futuras.
- Representa un área de estudio multidisciplinar y un enfoque de trabajo interdisciplinar.
- Ofrece una perspectiva más amplia e inclusiva acerca del uso y consumo de productos interactivos, y por tanto más acorde con la realidad.
- Hace especial énfasis en factores de la interacción tradicionalmente poco o mal considerados, como son el comportamiento emocional del usuario y la importancia de atributos de diseño como la estética en este comportamiento.

En el diseño web constantemente se mezclan principios del diseño de interfaz con principios de experiencia de usuario (UX), ya que al ser una aplicación en línea entran en juego otros factores que afectan al usuario, como lo es la confianza del usuario. «Esta confianza no se establece por el simple hecho de decir cosas como “Tu contraseña está cifrada bajo los estándares...” o por la credibilidad del producto. La respuesta está en la correcta elección de colores, tipografías, botones, pertinencia de campos, etc. Es decir, la interfaz tiene la capacidad de establecer esta base de confianza.» [13]

Uno de los elementos clave de una aplicación web son los formularios, en [13] se plantea el rediseñar un formulario como caso de estudio toma el formulario de registro de un banco, en el artículo el comenta porque eligió dicho formulario: «...lo elegí porque el mal diseño de experiencia es el común denominador en las interfaces de los bancos.» [13]. Este caso de estudio pertenece a la Figura 2.

Para crear su clave de acceso al sistema complete la información solicitada en los siguientes campos

**Usuario** (Digite su usuario de 8 caracteres alfanuméricos)

**Número de la tarjeta** (Seleccione los 4 primeros dígitos de su tarjeta)  
Si su tarjeta es débito maestro seleccione 5895

**Introduzca los 12 números restantes de su tarjeta débito o crédito**

**Seleccione su tipo de documento de identificación**

**Digite su número de identificación** (En caso de tener cédula de extranjería ingrese únicamente números, NO incluya letras)

**Clave de la Tarjeta** (La clave corresponde a la utilizada en cajeros automáticos)

**Crear clave** La clave debe estar conformada mínimo por 5, máximo por 8 caracteres (Números del 0 al 9 y letras de la A a la Z, el sistema valida minúsculas y mayúsculas)

Elija su clave personal de acceso a BBVA net

Confirme su clave personal de acceso a BBVA net

**Ingrese su correo electrónico de contacto** ( Este correo electrónico se utilizará para contacto, extractos y PagaTiempo. Si desea modificarlo puede hacerlo a través de BBVA net por la opción de Actualización de Datos - Dirección y Teléfonos- y realice la modificación campo correo electrónico de contacto.)

Figura 2 Caso de estudio de Carlos Jiménez [13], donde estudia cómo mejorar la interfaz

Carlos Jiménez [13] menciona los puntos clave que debe tener un formulario en una aplicación web para poder transmitir confianza y una buena experiencia al usuario:

- **Pertinencia:** Cada campo debe ser coherente con el objetivo del formulario.
- **Opcionales:** Cada vez que un campo es opcional preguntémoslo si realmente vale la pena tenerlo. Si lo hacemos, aclaremos que lo es.
- **Alineación:** La alineación de Label y campo es un error común. Una alineación horizontal no es la mejor opción para labels largos o formularios extensos. Ya que puede ser difícil entender qué campo corresponde a qué etiqueta.
- **Agrupación:** Agrupar campos relacionados mantiene una comunicación coherente.
- **Ayudas:** Las ayudas son valiosas si son pertinentes, de lo contrario confunden más.
- **Validación:** Es incómodo que un campo de mail te diga mail incorrecto cuando sólo has digitado una parte. Seamos gentiles con nuestra validación, ya que es un punto de comunicación constante con el usuario. Evitemos la validación en tiempo real cuando no es necesaria. Sólo es útil cuando se enfoca en ayudar y no en corregir.



- **Call to Action:** La acción final es el cierre y confirmación de todo lo construido. Es importante diferenciar la acción principal e indicar cuál es el objetivo. Botones genéricos como Enviar reducen la conversión.

Al terminar el nuevo diseño (Figura 3) el autor, Carlos Jiménez, nos muestra un formulario más fácil de entender y que transmite más confianza al usuario final.

## Regístrate a los servicios web de BBVA

Podrás consultar tus productos y hacer transacciones online.

Tus datos

---

### Nombre de Usuario

### Tipo de documento de identidad

 ▼

### Número de Documento

### Correo electrónico

Lo utilizaremos para ponernos en contacto, enviarte extractos y "PagaTiempo".

Información de tu tarjeta

---

### Número de la tarjeta

### Clave de la tarjeta

La misma que usas en el cajero

### Contraseña de acceso a BBVA online

Figura 3 Versión final del rediseño para mejorar la confianza en [13]

## Desarrollo de Interfaces Multi-Plataforma

Como respuesta a la cantidad de diferentes dispositivos desde los cuales se puede navegar a internet nace el deseo de poder brindar una buena experiencia de usuario y que esta sea similar independiente del dispositivo donde se accede. Es por esto que nace el *Responsive Design*, como una forma de poder brindarle al usuario la mejor experiencia posible sin importar el dispositivo desde que acceda.

Debido al crecimiento que tiene la navegación desde un dispositivo móvil, nace el movimiento *Mobile First* (diseño Móvil primero) dentro del *Responsive Design*, donde se pretende empezar a diseñar la interfaz teniendo en cuenta cómo se va a ver y utilizar en un contexto móvil con un dispositivo de pantalla más pequeña y luego pensar como esa interfaz se puede adaptar para una buena experiencia en un dispositivo con una pantalla más grande, como lo sería un computador portátil.

El tipo de aplicación que se desea rediseñar tiene como audiencia a desarrolladores o equipos de desarrolladores dentro de una empresa. Las aplicaciones orientadas a empresas tienen un gran foco en usabilidad y en eficiencia de las funciones que hace más que en diseño gráfico.



## Capítulo 2. Pautas para desarrollo IU Web modernas

Una buena interfaz web no se basa solo en el diseño, sino también en la experiencia que tenga el usuario al momento de utilizarla. Algo similar sería afirmar que la experiencia del usuario no solo está en una interfaz elegante, sino también en que tan segura, rápida e intuitiva sea dicha interfaz.

A continuación se exponen buenas prácticas para desarrollar una interfaz de usuario web moderna y algunas consideraciones a tener en cuenta del lado del servidor al desarrollar aplicaciones *Single Page*:

### Evitar Scroll Horizontal

Esta sección se ha extraído de [14]:

El autor y consultor en usabilidad Jakob Nielsen plantea que hay 3 razones por las que se debe evitar el *Scroll* Horizontal.

1. Los usuarios tienden a tener un desagrado por el *scroll* horizontal, la satisfacción del usuario debería ser una razón suficiente para evitarla.
2. En la Web, los usuarios esperan encontrarse con *Scroll* Vertical.
3. Cuando una página web tiene *scroll* vertical y horizontal, el usuario debe de moverse en 2 dimensiones, lo que hace muy difícil que el usuario pueda ver todo el contenido. En contraste, cuando el *scroll* es en una sola dimensión, el usuario puede ver todo el contenido sin tener que pensar primero hacia donde moverse; solo tiene que hacer *scroll* hacia abajo.

Para lograr esto hay que tener siempre en cuenta que todo el contenido quepa dentro del ancho total de la pantalla donde se muestra.

### Responsive Design

A medida que la cantidad de dispositivos móviles aumenta, crece el interés por desarrollar una aplicación que pueda ser utilizada en todos estos dispositivos móviles, como son tabletas, consolas de juego, o cualquier otro dispositivo del futuro que se conecte al internet [15].

Una buena solución a este problema es tener una sola interfaz que se adapte al tamaño de la pantalla donde se muestre, esta adaptación de elementos que se acomodan según el tamaño de la pantalla se le conoce como

*Responsive Design*, y el término fue introducido por primera vez por Ethan Marcotte [15].

El autor Ethan Marcotte [15] plantea que el diseño *Responsive* cuenta con 3 aspectos claves para poder funcionar:

- Cuadrícula fluida, permite distribuir el espacio
- Imágenes flexibles, pueden cambiar de dimensiones de forma fácil
- Media Queries: Módulo de CCS para ajustar contenido a dimensiones de la pantalla

Existen varios *frameworks* de CSS populares que contemplan un diseño de *Responsive Design* desde su núcleo y facilitan el desarrollo de plantillas que se adapten al dispositivo donde se muestran. Algunos de los más populares son:

- Bootstrap (<http://getbootstrap.com/>)
- Foundation (<http://foundation.zurb.com/>)
- Skeleton (<http://getskeleton.com/>)
- Pure.CSS (<http://purecss.io/>)
- Semantic UI (<http://semantic-ui.com/>)

## Mobile First

Otra tendencia que viene de la mano con el *Responsive Design* es el diseño Mobile First [16]. En este se plantea el diseñar la interfaz pensando primero en la experiencia móvil y luego pensar en cómo se utilizaría en una versión de escritorio.

Las ventajas que tiene este tipo de acercamiento es el de poder pensar en elementos que se puedan transformar en una versión más útil para cada pantalla. Es decir elementos que puedan mostrar más o menos información dependiendo del espacio y que delimiten de una forma clara los elementos que pertenecen a un grupo de elementos comunes.

## Elementos y grids con pocas columnas o en formato *card*

El evitar el *scroll* horizontal en una aplicación viene con la limitación que es más difícil mostrar información relacionada a un mismo elemento cuando se está haciendo un listado de algún contenido, por ejemplo haciendo uso de una tabla o grid.

Una nueva tendencia es el uso de *cards*. Productos y empresas tan populares como Pinterest, Twitter, Spotify (en su funcionalidad de Explorar), Facebook y

Google han optado por este tipo de elemento a la hora de diseñar una aplicación web [16].

Entre los beneficios que tiene este tipo de elementos se encuentra la cantidad de información que pueden contener dentro de sí mismos, a diferencia de solo ser una fila en una tabla. También está la capacidad que tienen estos elementos en transformarse según el dispositivo para dar más o menos información al usuario, como se muestra en la Figura 4.

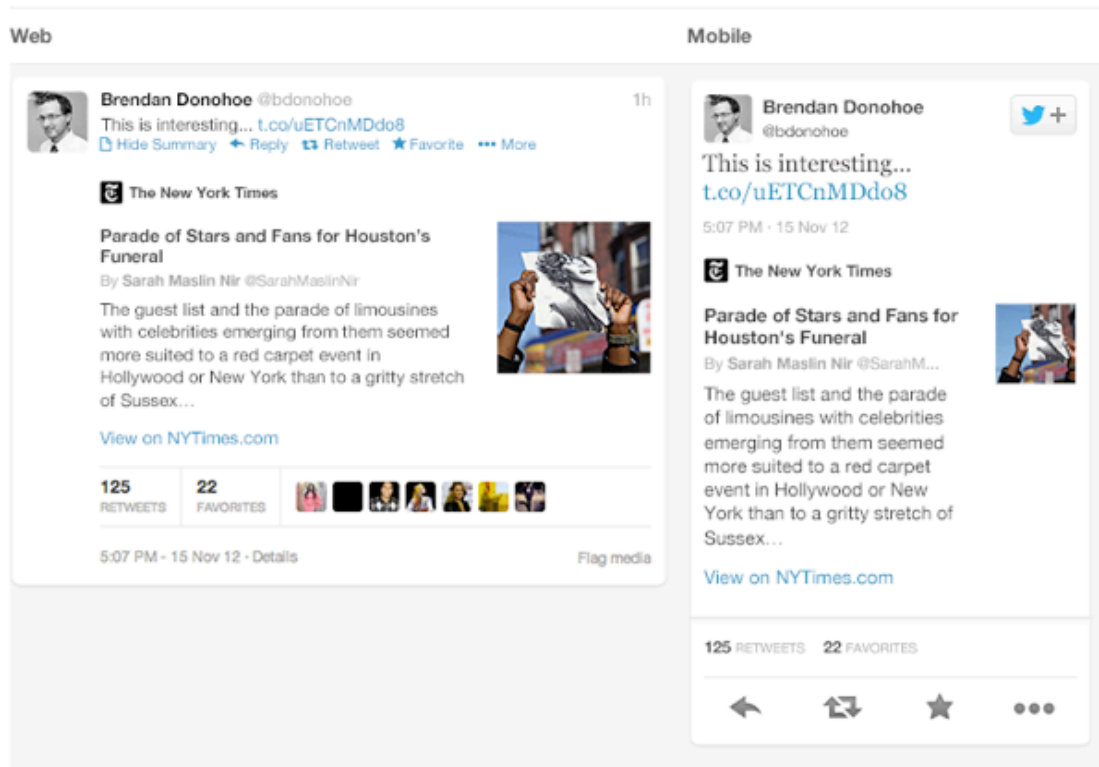


Figura 4: Ejemplo de como Twiter utiliza los *cards* en diferentes dispositivos [16]

Los *cards* también delimitan el espacio para describir de una forma visual la separación entre elementos. Es mucho más fácil para el ojo humano detectar la división entre contenidos similares.

«Los cards no solo son buenos para el ojo y para el diseño *responsive* y móvil en general, sino que también son uno de los elementos con formato más flexible para crear experiencias consistentes» [17].

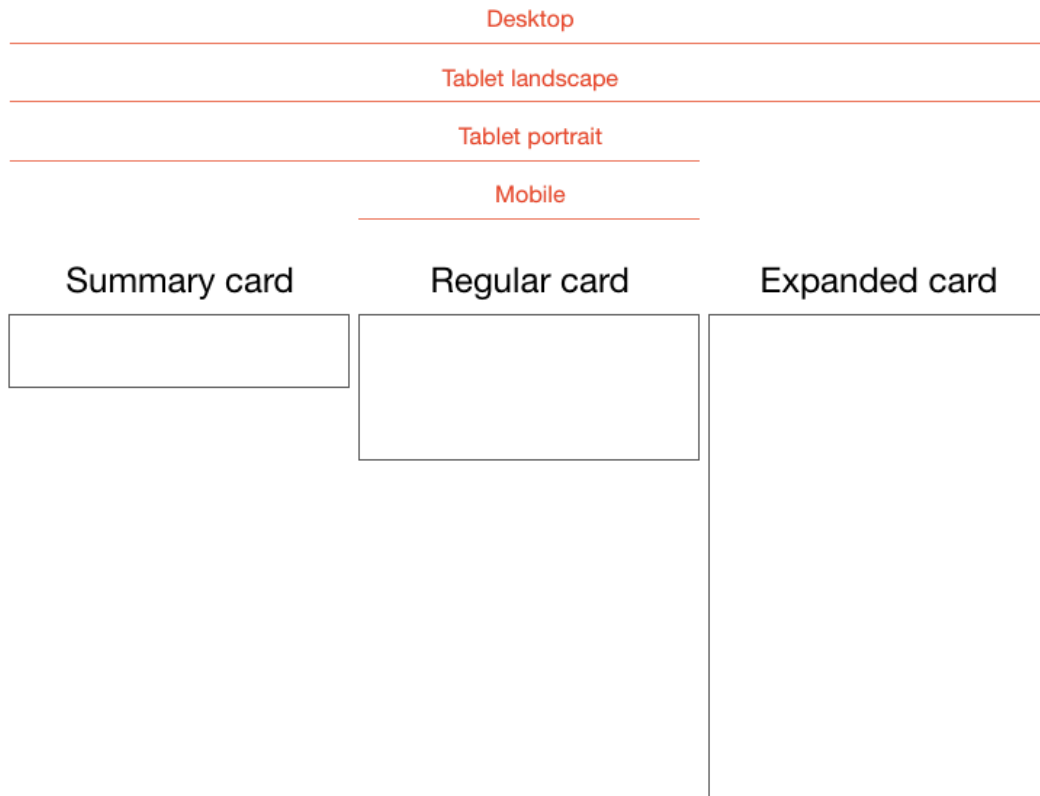


Figura 5: Muestra como los *cards* se puede acomodar al dispositivo donde se muestran [16]

Cabe destacar que este tipo de elemento también se ha hecho popular por la cantidad de dispositivos móviles que existen actualmente y que cada vez más se convierten en el dispositivo principal de un usuario para navegar en la web.

## Comunicación entre cliente y servidor en tiempo real

Una tendencia que mejora mucho la experiencia de usuario es tener en la aplicación web comunicación en tiempo real entre el cliente y el servidor en las partes más críticas de una aplicación, normalmente para enviar notificaciones o mensajes.

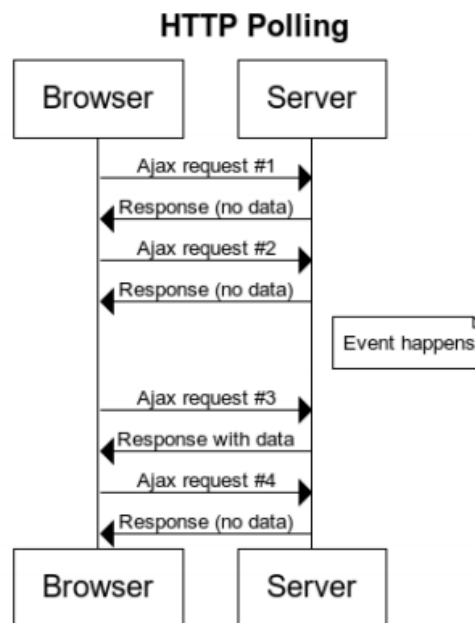
Existen diferentes formas de lograr la experiencia de comunicación en tiempo real entre cliente web y servidor. Cada una tiene sus ventajas y desventajas, a continuación hablaremos sobre ellas:

### Short-Polling

Short-Polling o HTTP Polling, consiste en que el cliente envía peticiones por medio de AJAX al servidor de forma continua en intervalos de tiempo

determinados. Este método es muy ineficiente ya que para cada consulta de información se tiene que hacer una petición y obtener una respuesta inmediata del servidor, independiente de si hay data nueva que traer o no [18]. Como se muestra en la Figura 6.

Como esta petición se hace sobre http igual hay que construir una cabecera que se envía con cada petición. Se podría pensar que se podría hacer más eficiente este método haciendo que el intervalo entre peticiones al servidor sea un poco más elevado de tiempo, pero en realidad esto solo hace que el cliente tenga la información más tarde. [18]



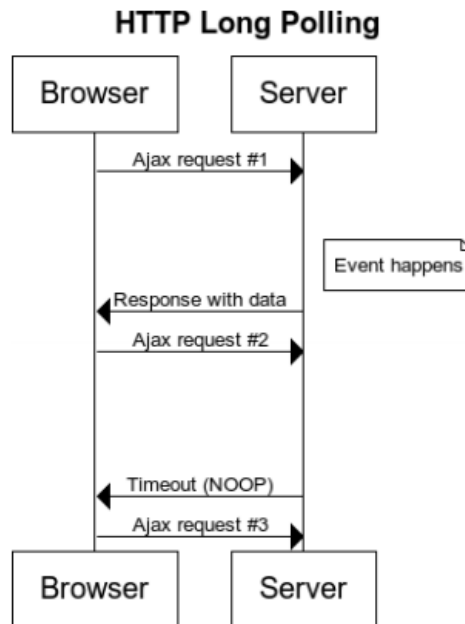
**Figura 6:** Muestra la comunicación entre cliente-servidor utilizando Short-Polling [18]

### Long-Polling

Long-Polling es muy parecido a Short-Polling, en el sentido en que se basa en hacer peticiones AJAX desde el cliente al servidor, pero se diferencian en que en el método de Long-Polling deja la conexión HTTP abierta y no recibe una respuesta del servidor hasta que no haya data nueva y el servidor responda. Aunque teóricamente Long-Polling pareciera ser mucho más eficiente, este no es el caso ya que tiene la limitante de cantidad de conexiones HTTP que puede tener abierta por momentos [18].

Long-Polling si reduce la cantidad de peticiones que se hace al servidor mientras no hay data nueva, pero resulta poco eficiente en que para cada petición y respuesta se envía una cabecera HTTP que en la mayoría de los casos excede el tamaño en bytes de la información que se quiere enviar [18].





**Figura 7: Muestra la comunicación entre cliente-servidor usando Long-Polling. La conexión se queda abierta hasta que hay data nueva que enviar. [18]**

En un caso de uso donde la información que se actualiza con Long-Polling es muy frecuente este resulta idéntico que utilizar Short-Polling.

## Websockets

Websockets es un protocolo reciente estandarizado por HTML5, éste provee una conexión full-duplex bidireccional que opera sobre una simple conexión sobre un *sócalo* o *socket*, y puede servir para desarrollar una aplicación web en tiempo real [19].

HTML5 provee un API para hacer uso de Websockets en cualquier navegador web que lo soporten. También, existen varias librerías para crear servidores y clientes de Websockets de una forma sencilla, estas librerías se encargan de detectar si el navegador soporta conexión de Websockets, si este no lo soporta pasan a crear una conexión Long-Polling.

Entre las librerías más populares y fáciles de implementar se encuentran:

- **SignalR** (<http://signalr.net/>): Librería para ASP.NET
- **Socket.io** (<http://socket.io/>): Librería para Node JS
- **WebsocketD** (<http://websocketd.com/>): Librería que convierte cualquier aplicación que pueda escribir por StdOut y recibir por StdIn en un servidor de Websockets.

## Retroalimentación de progreso (*Feedback*)

Al desarrollar cualquier tipo de aplicación es muy recomendable tener en cuenta el dar retroalimentación del progreso de una acción al usuario. En las aplicaciones Single Page Application es muy fácil dar *feedback* inmediato al usuario luego de realizar la acción, mientras está nueva data se intenta persistir en el servidor.

Algunas de las formas de dar *feedback* al usuario pueden ser como las siguientes: Deshabilitar un botón luego de presionado, mostrar una barra de progreso, entre otras [20].

También, es bueno tener presente que sucede si la acción de llamar al servidor falla, por ejemplo, mostrarle un mensaje de error al usuario o revertir el objeto al estado que estaba antes de hacer la acción [20].

## Seguridad del lado del cliente

En cualquier aplicación web que esté escrita siguiendo el paradigma de Single Page Application se encuentra con una problemática que es un tanto difícil de controlar, y es que dicha aplicación está escrita en JavaScript y a ese código es muy fácil de tener acceso del lado del cliente. Es por esto que almacenar *tokens* que identifiquen al usuario de forma local se convierte en un reto.

De igual forma se entiende que un usuario que está dentro de su cuenta podría tener acceso al token generado para su sesión específica y no causar ningún problema. El envío de token de cliente al servidor siempre debe ser realizado por medio de la cabecera de la petición HTTP y preferiblemente encriptado con un certificado SSL.

Una solución para el manejo de sesiones en un Single Page Application es el uso de JSON Web Tokens (JWT). «Estos permiten tener una forma estructurada de declarar quien es el usuario y a que puede tener acceso. Estos pueden estar encriptados y firmados para poder verificarlos» [21]

El tener la autenticación del usuario basada en tokens trae muchos beneficios a cualquier tipo de aplicación, en especial a una aplicación de *Single Page Application*, sin embargo existen 4 beneficios a destacar [22]:

- **Servidores sin estado (Stateless):** El *token* contiene la información de cómo identificar a un usuario, eliminando la necesidad de tener una tabla de sesiones. En un ambiente con servidor de balanceo de carga, se puede enviar el usuario con el *token* a cualquier servidor. [22]

- **Reusabilidad:** Permite tener diferentes aplicaciones que utilicen el mismo *token* para autenticar un usuario, funciona muy bien en un ambiente SOA (*Service Oriented Architecture*). [22]
- **Seguridad:** Permite autenticar que la petición viene de un lugar confiable, protege contra un ataque CSRF (*Cross-site Request Forgery*). [22]
- **Rendimiento:** En cada petición no es necesario traer la sesión actual del usuario. [22]

## Seguridad del lado del servidor

En una aplicación de tipo SPA (*Single Page Application*) [22] es crítico contar con un certificado SSL y que toda comunicación se haga a través de HTTPS, ya que el cliente va a enviar un *token* para identificarse, es crucial que dicho *token* sea enviado por la cabecera o los *headers* y que la conexión sea encriptada con un certificado SSL [22].

Esto permite que si alguien intenta interceptar la llamada y obtener el *token* le resulte muy difícil ya que la cabecera de la petición está encriptada.

En cada petición hecha al servidor se debe verificar si el usuario que hace la petición está autorizado a recibir esa data, las validaciones de autorización no deben hacerse solo del lado del cliente.

Cabe destacar que para que una aplicación Single Page Application sea considerada como buena, también el servicio web que alimenta dicha aplicación tiene que tener una buena arquitectura. Una de las ventajas de tener un buen diseño desde el principio es que luego es muy fácil habilitar una API a terceros si el Servicio Web creado para el Single Page Application estuvo bien diseñado desde el principio.



## Capítulo 3. Ecosistema de tecnologías utilizadas

Para el desarrollo de la aplicación web se utilizó el framework de frontend Angular JS en su versión 1.3.

El diseño se basó en una plantilla de CSS elaborada con el framework de Bootstrap. Este framework contiene una serie de componentes que son muy útiles a la hora de realizar una plantilla de CSS.

### Angular JS 1.3



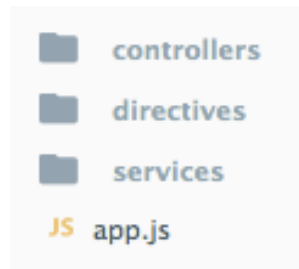
<https://angularjs.org/>

Para esta aplicación se utilizó la versión 1.3 de este framework. Este framework es confundido con una librería debido a su tamaño, ya que es inferior al promedio de tamaño de un framework frontend. En sus inicios Angular JS era considerado como un framework MVC (Modelo-Vista-Controlador), pero a medida que se fue desarrollando este cambio a MVW (*Model-View-Whatever*, en español Modelo-Vista-Cualquier Cosa) [23] esto debido a que aparecieron muchos debates decidiendo que tipo de framework era y los autores del framework expresaron lo siguiente «Prefiero ver a desarrolladores escribir aplicaciones espectaculares y bien diseñada que verlos perder tiempo decidiendo que tipo de framework MV\* es Angular JS» [23]

Cabe destacar que el framework a partir de la versión 2.0 tiene cambios muy grandes dentro de su código fuente central. Es tanto el cambio que se podría considerar la versión 2.0 de Angular JS como un framework totalmente distinto a la versión 1.3 del mismo framework.

Al Angular ser un framework que no está atado a un tipo de arquitectura, el tipo de arquitectura queda a libre elección de la persona o el equipo de personas que va a desarrollar la aplicación. Para el prototipo desarrollado se tomó la decisión de agrupar el código según el tipo de componente.

La estructura de carpetas se hizo como se muestra en la Figura 8.



**Figura 8 Estructura de carpetas de la aplicación de Angular JS**

Se separó la aplicación en 3 directorios:

- **Controllers:** Directorio que almacena los controladores de Angular JS. Un controlador en Angular es el componente que tiene acceso al DOM y por ende lo puede modificar.
- **Directives:** Directorio que almacena los componentes y atributos personalizados de HTML creados con Angular JS. El framework permite la creación de dichos componentes para agregarle dinamismo a elementos HTML.
- **Services:** Directorio donde se almacenan los módulos que tienen acceso a la data y que realizan las consultas webs.
- **app.js:** Archivo que contiene la configuración inicial de la aplicación de Angular JS. También contiene la definición de las rutas de Angular.

De la arquitectura de la aplicación de Angular JS entre los componentes más destacados se encuentra el servicio TuneupData. Este nos es más que un *Angular JS Service* encargado de todas las llamadas realizadas al servidor desde la aplicación de Angular.

## Node JS



<https://nodejs.org/>

Es una plataforma construida sobre el *runtime* de JavaScript de Google Chrome diseñada para construir aplicaciones en la red de forma rápida y escalable. [24]

Este framework se utilizó para crear un pequeño servidor que pudiera servir la aplicación de Angular JS y pudiera redirigir las peticiones hacia los

servidores de TUNE-UP Process. Esto debido a que desde una aplicación de frontend no se puede realizar una petición a otro servidor por medio de AJAX.

## Express JS

express

<http://expressjs.com/>

Express JS es un framework de aplicaciones Web para Node JS flexible y minimalista, este provee una serie de funcionalidades robustas para crear aplicaciones web y móviles. [25]

Este framework Web se utilizó para crear las rutas que la aplicación de Angular JS iba a llamar por medio de AJAX.

Una aplicación de Express JS no viene con una estructura de carpetas a respetar, sino que permite al usuario crear la arquitectura que desee o necesite. Para el prototipo se decidió optar por una arquitectura un tanto simple donde la parte principal era el manejo de rutas y el re-enviar las peticiones a los servidores de TUNE-UP.



**Figura 9 Arquitectura de la aplicación de Node JS y Express JS**

La arquitectura de la aplicación de Node JS y Express JS, como muestra la Figura 9, contiene varios archivos para su funcionamiento, entre los componentes más importantes están:

- **/routes.** Directorio que contiene todas las rutas que la aplicación de Angular JS llama desde el cliente.
- **/util.** Directorio que contiene todos
- **/views.** Directorio que contiene todas las vistas HTML para que luego sean retornadas por el servidor.
- **app.js** Archivo que contiene la definición del servidor y el registro de las rutas, a continuación código donde se definen las diferentes rutas a la que va a acceder la aplicación de Angular JS:

```
app.get('/kanban.json', routes.kanban);
app.get('/dashboard.json', routes.dashboard);
app.get('/sites.json', routes.sites);
app.post('/login.json', routes.login);
app.get('/typesUT.json', routes.typesUT);
app.get('/sprints.json', routes.sprints);
app.get('/projects.json', routes.projects);
app.get('/workflows.json', routes.workflows);
app.get('/avatars.json', routes.usersAvatars);
app.put('/crearUT.json', routes.crearUT);
app.put('/crearMensaje.json', routes.crearMensaje);
```

## Local Storage



La especificación de *Local Storage* se introdujo con HTML5, se introdujo como una forma de reemplazar a los Cookies para almacenar data del lado del cliente [26]. Con *Local Storage* cada página de internet tiene una pequeña base de datos en la cual almacenar información local del usuario. Junto con *Local Storage* también se introdujo la especificación de *Session Storage*, la diferencia de estos 2 tipos de almacenamiento es que *Session Storage* es eliminado al terminar la sesión (cuando el navegador es cerrado), mientras que *Local Storage* es permanente. [26]

En la aplicación de prototipo se utilizó el *Local Storage* para almacenar el *token* de autenticación que el usuario envía junto con cada petición al servidor web.



## Jade



<http://jade-lang.com/>

Jade es un pre-procesador de HTML o también conocido como un motor de *templates* o plantillas. Es un DSL que permite escribir plantillas HTML siguiendo un lenguaje más ligero que HTML. Según su sitio web Jade es capaz de [27] :

- Producir HTML a partir de un lenguaje lacónico
- Soportar código dinámico
- Soportar Reusabilidad (patrón de diseño DRY)

## Grunt JS

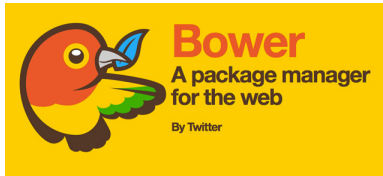


<http://gruntjs.com/>

Permite realizar tareas comunes en el desarrollo de frontend de forma automática. Como lo son la minificación, compilación, verificación de sintaxis, pruebas unitarias, entre otros.

Entre las ventajas que ofrece Grunt JS es que puede observar archivos y una vez detecta un cambio empieza a ejecutar las tareas que tiene configurado.

## Bower



<http://bower.io/>

Es un manejador de componentes web, la mayor ventaja que tiene es que permite al equipo de desarrollo completo trabajar con las mismas librerías para frontend sin necesidad de agregar dichas librerías al proyecto. Sino que mantiene un archivo en formato JSON con las referencias de que librerías está usando el proyecto y cuales versiones.

## Bootstrap CSS



<http://getbootstrap.com/>

Es un framework de CSS, HTML y JavaScript para desarrollar websites de diseño *responsive* y que sigan la metodología mobile-first. Este framework web cuenta con una serie de componentes ya listos para usar que siguen esta metodología de mobile-first. Una de las ventajas más grande que tiene este framework es que cuenta con una cuadrícula de CSS ya definida de 12 espacios. Esta cuadrícula es muy poderosa y versátil para poder asignarle a las partes de la aplicación donde estará y cuál será su espacio en pantalla.

.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1
.col-md-8								.col-md-4			
.col-md-4				.col-md-4				.col-md-4			
.col-md-6						.col-md-6					

**Figura 10: Ejemplo de uso de Cuadrícula del framework Bootstrap**

## Underscore JS

# UNDERSCORE.JS

<http://underscorejs.org/>

Es una librería de JavaScript que contiene una variedad de funciones del paradigma de programación funcional sin extender la definición de ninguno de los objetos del lenguaje JavaScript. [28]

## Stylus



<https://learnboost.github.io/stylus/>

Es un pre-procesador de CSS. Los autores lo definen de la siguiente forma «Stylus es un lenguaje revolucionario, provee una forma eficiente y dinámica de generar CSS. Soporta tanto una sintaxis basada en indentación como la sintaxis regular de CSS.» [29]

## Git



<https://git-scm.com/>

Según su sitio web «Git es un sistema distribuido de control de versiones diseñado para manejar los archivos de un proyecto de forma rápida y eficiente.» [30]

### **Ramas y Fusión (del inglés Branch & Merge)**

Esta sección se ha extraído de [31]:

Una de las características que más destacan a este sistema de control de versiones es la capacidad para crear ramas del código local de una forma fácil y rápida. Esto trae muchas ventajas y tiene muchos usos, algunos de los cuales son:

- **Cambio de contexto sin fricción:** habilidad de crear ramas para probar ideas en la aplicación, y luego si se desea poder cambiar a la rama original y no perder los cambios de la nueva idea.
- **Ramas diferentes basada en Roles:** habilidad para tener ramas que contengan una versión de la aplicación específica. Un ejemplo de esto es tener una rama para código que está listo para producción, otra para código que está en etapa de prueba y otras ramas pequeñas para el trabajo del día a día.
- **Flujo de trabajo basado en características:** habilidad para crear ramas que contengan la implementación de una característica de la aplicación, esto permite poder cambiar entre la versión actual de la aplicación y la versión con la nueva característica aplicada.
- **Experimentos temporales:** habilita al desarrollador a crear una rama con el solo hecho de hacer un experimento, si al final esta idea no funciona simplemente puede borrar la rama y el código vuelve a su estado original antes de hacer la rama.



Figura 11: Flujo de trabajo donde las ramas tienen código separado según el estado en que se encuentran.

## Git Hooks

<https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks>

Como otros sistemas de control de versiones Git tiene maneras para lanzar scripts de consola al escuchar diferentes eventos. Según su sitio web hay 2 tipos de *Git Hooks* los que se ejecutan del lado del cliente y los que se ejecutan del lado del servidor.

Del lado del cliente se disparan por las operaciones de hacer *commit* (acción realizada al guardar cambios en el repositorio) y *merge* (fusión de código entre ramas), y del lado del servidor se disparan con eventos realizados con la red, como hacer un "*pull*" (acción de traer los cambios nuevos del repositorio) [32]

## Bitbucket



<https://bitbucket.org/>

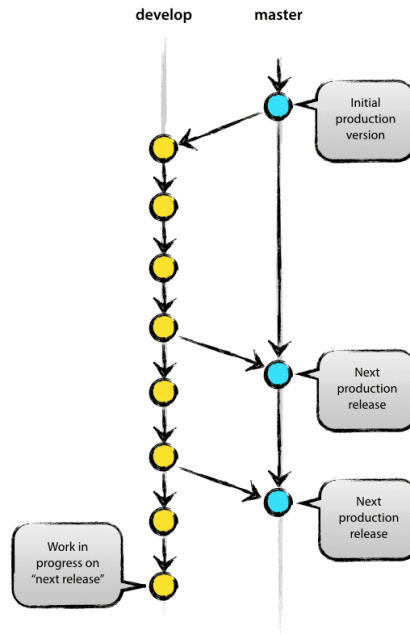
Aplicación tipo *SaaS* (Software as a Service) para alojar repositorios de sistemas de control de versiones *Git* o *Mercurial*. Este fue utilizado para almacenar el repositorio del prototipo desarrollado.

## Flujo de trabajo en Git (*Branching Model*)

Git da la facilidad de trabajar con diferentes flujos de trabajo, esto queda a decisión del equipo de desarrollo. Para el prototipo desarrollado en el TFM se siguió el modelo publicado en [33]. En este modelo se tienen 2 ramas principales que tienen un tiempo de vida infinito, las ramas son:

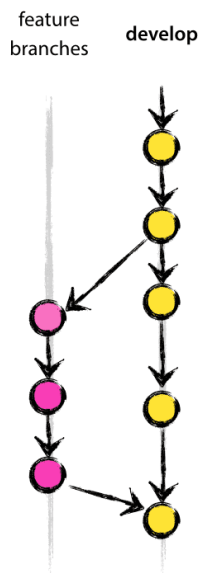
- master
- develop

La rama *master* contiene la última versión estable de la aplicación, esta rama siempre está lista para ser puesta en producción. La rama *develop* siempre tiene el último código que se está preparando para la próxima actualización de la aplicación.



**Figura 12:** Imagen que muestra gráficamente los commits en las ramas develop y master

Junto a las ramas principales se encuentran ramas de soporte, estas ramas están destinadas a crear características nuevas de la aplicación. Las ramas de soporte deben crearse a partir del estado de la rama develop y una vez se termine de trabajar en la rama nueva, estas se deben fusionar (*merge*) a la rama develop.



**Figura 13:** Gráfico que muestra el ciclo de vida de una rama de soporte

## Heroku



<https://www.heroku.com/>

Es un PaaS (*Platform as a Service*) que se encarga de rentar los servidores ya configurados para poner una aplicación en producción. Según su website «Nuestro servicio permite a los desarrolladores pasar su tiempo escribiendo código, y no configurando y gestionando servidores. » [34]

Esta plataforma no solo permite manejar el servidor que va a servir la aplicación sino también el servidor de base de datos y demás complementos como certificados SSL y servicios de monitoreo de aplicación.

Heroku fue utilizada como servidor web para interactuar con la aplicación en internet mientras era desarrollada.





## Capítulo 4. Caso de Estudio

### TUNE-UP Process

Como caso de estudio se tomó la herramienta *Tune-up Process*. Esta herramienta se presta como un buen candidato para caso de estudio porque al no tener una aplicación web da espacio a la creatividad y a poder crear un rediseño de la forma en que es utilizada. También porque según su descripción oficial «TUNE-UP permite la colaboración entre miembros de un equipo, estén estos co-localizados o distribuidos». [35], razón por la cual hace sentido plantearse el rediseño hacia una aplicación multi-plataforma y multi-dispositivo que permita que los miembros de un equipo puedan utilizar la herramienta independiente de sistema operativo o plataforma que utilicen.

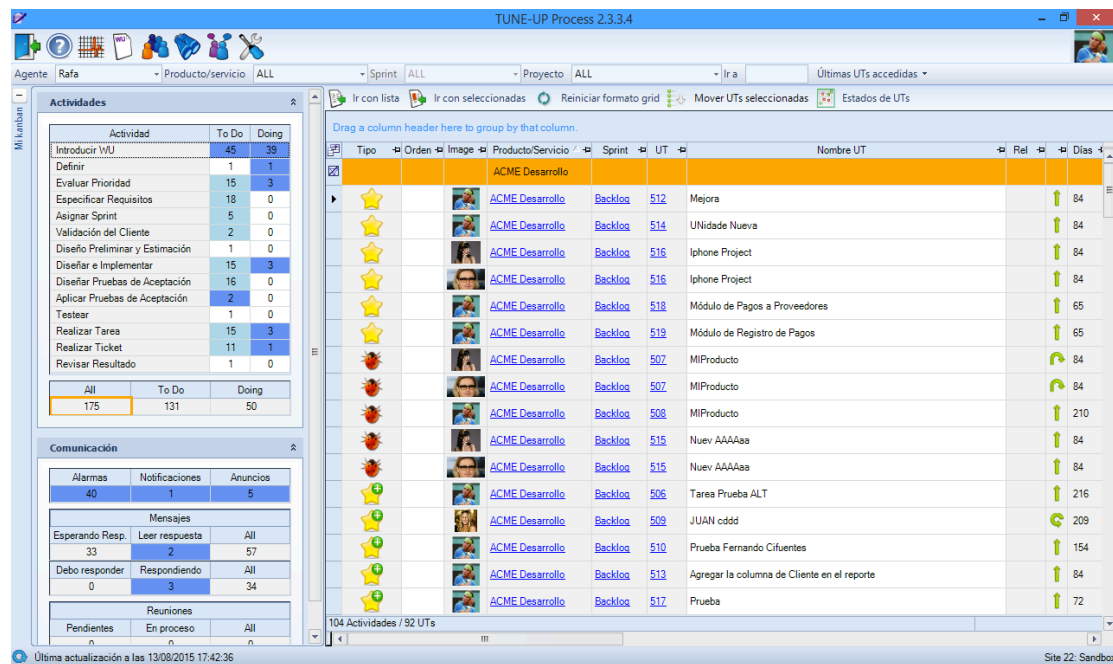


Figura 14: Vista inicial al entrar a la aplicación de Tune-Up Process

TUNE-UP Process maneja algunos conceptos básicos para poder entender la metodología, estos son:

- **Iteración.** «Es el período de tiempo durante el cual el equipo trabaja en conseguir una nueva versión del producto. Tal como en Scrum, en TUNE-UP se promueve que las iteraciones no sobrepasen las 4 semanas.» [36].

- **Work Unit (WU).** «Las WUs son las unidades de trabajo que se incluyen en una iteración. Usamos este concepto para englobar cualquier tipo de cambio en el producto, y también para referirnos a cualquier otra tarea que se enmarque en el trabajo de una iteración.» [36].
- **Agente o miembro del equipo.** «Es un participante en el trabajo de la iteración (puede o no ser del equipo de desarrollo)» [36].

Actualmente la herramienta TUNE-UP Process solo puede ser utilizada en dispositivos con el sistema operativo Windows.

Según su sitio web «TUNE-UP es el nombre de una metodología y su herramienta, cuyo propósito es ofrecer apoyo para la gestión ágil de proyectos de desarrollo y mantenimiento de software. TUNE-UP ha sido desarrollada en la Universidad Politécnica de Valencia siendo el fruto de la estrecha colaboración con empresas en diversos proyectos llevados a cabo desde 2005.» [35]

La herramienta cuenta con diferentes módulos para realizar las diferentes tareas, voy a mencionar algunas de ellas que serán las que se tomarán en cuenta para el prototipo:

### Planificador Personal (PEP)

El primero módulo que ve un usuario luego de iniciar sesión es el de Planificador Personal (PEP).

«El PEP es el centro de operaciones de cada agente. En él se presenta todo el trabajo no finalizado en el cual tiene participación el agente. La información está organizada al estilo de los populares Kanban, pero integrando de forma efectiva la visualización no sólo del trabajo de un producto-versión, sino de múltiples productos y versiones, y junto con alertas, notificaciones, mensajes y reuniones.» [35]

### Comunicación y Mensajes

Durante la vida de un proyecto la buena comunicación entre los *stakeholders* es crítica para poder lograr la meta del proyecto a tiempo, tener un canal de comunicación efectivo y claro es una prioridad. La herramienta TUNE-UP tiene un módulo de comunicación que actúa como canal para centralizar la comunicación entre el equipo.

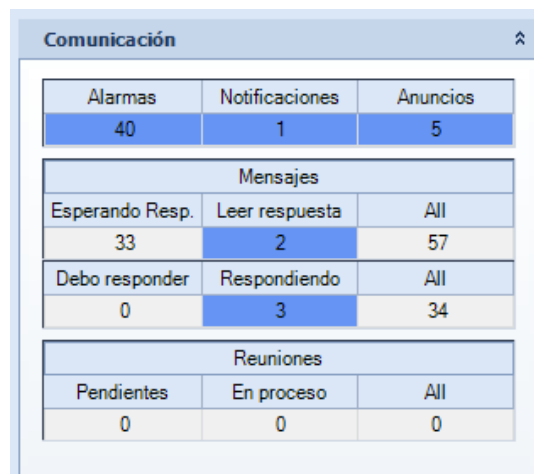
Según su sitio web «TUNE-UP permite concentrar en cada unidad de trabajo toda la comunicación generada, tanto en forma de mensajes como de reuniones, ambas complementadas con un repositorio de documentos.» [35]

## Alertas, Notificaciones y Anuncios

El siguiente contenido fue extraído de [35]

La herramienta TUNE-UP ofrece estos tres medios de comunicación para mantener informados a los participantes de eventos que les son relevantes. Las alertas y las notificaciones son automáticas

- Las alertas advierten de una situación anómala.
- Las notificaciones avisan simplemente que ha ocurrido un evento.
- Los anuncios son mensajes que puede enviar cualquier participante hacia uno o más de sus colaboradores, sin esperar una respuesta.



El panel de comunicación muestra un resumen de estadísticas y un detalle de los mensajes pendientes. El título del panel es 'Comunicación'. El resumen incluye:

Alarmas	Notificaciones	Anuncios
40	1	5

Debajo de esto, se detallan los mensajes pendientes:

Mensajes		
Esperando Resp.	Leer respuesta	All
33	2	57
Debo responder	Respondiendo	All
0	3	34

Finalmente, se muestran las reuniones:

Reuniones		
Pendientes	En proceso	All
0	0	0

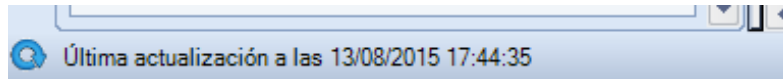
**Figura 15: Panel del lado izquierdo de la herramienta que muestra las Alarmas, Notificaciones, Anuncios y Mensajes pendientes.**

Los 3 módulos anteriormente mencionados fueron los módulos que se evaluaron para estar contenidos en el rediseño de la aplicación.

## Limitaciones de la herramienta Tune-Up Process

La versión más reciente de Tune-Up Process solo funciona en el sistema operativo Windows, esto crea una barrera en la colaboración para miembros de un equipo que utilicen un sistema operativo diferente. Al ser una aplicación de escritorio también limita el tipo de dispositivo donde puede ser utilizada.

La aplicación actual tiene como naturaleza de no tener botón de salvar cambios, sino que con cada acción realizada se envía la información al servidor central para almacenarla, y cada cierto tiempo se trae información del servidor para actualizar la data mostrada al usuario. Actualmente dichas peticiones se realizan de forma síncrona, haciendo que el usuario tenga que esperar un tiempo.



**Figura 16:** Barra de estado difícil de ver que se encuentra en el lado inferior izquierdo y que muestra la ultima fecha de actualización de datos.

El PEP y otros módulos que muestran un tablero o grid al usuario final tienden a mostrar mucha información, lo que resulta en que tenga que ser mostrado un scroll horizontal. Este tipo de scroll es difícil para un usuario utilizar y causa perdidas de tiempo.

Tipo	Orden	Image	Producto/Servicio	Sprint	UT	Nombre UT	Rel	Días
			ACME Desarrollo					
★			ACME Desarrollo	Backlog	512	Mejora	↑	84
★			ACME Desarrollo	Backlog	514	UNidade Nueva	↑	84
★			ACME Desarrollo	Backlog	516	Iphone Project	↑	84
★			ACME Desarrollo	Backlog	516	Iphone Project	↑	84
★			ACME Desarrollo	Backlog	518	Módulo de Pagos a Proveedores	↑	65
★			ACME Desarrollo	Backlog	519	Módulo de Registro de Pagos	↑	65
🐛			ACME Desarrollo	Backlog	507	MIPProducto	↺	84
🐛			ACME Desarrollo	Backlog	507	MIPProducto	↺	84
🐛			ACME Desarrollo	Backlog	508	MIPProducto	↑	210
🐛			ACME Desarrollo	Backlog	515	Nuev AAAAaa	↑	84
🐛			ACME Desarrollo	Backlog	515	Nuev AAAAaa	↑	84
★+			ACME Desarrollo	Backlog	506	Tarea Prueba ALT	↑	216
★+			ACME Desarrollo	Backlog	509	JUAN cddd	↺	209
★+			ACME Desarrollo	Backlog	510	Prueba Fernando Cifuentes	↑	154
★+			ACME Desarrollo	Backlog	513	Agregar la columna de Cliente en el reporte	↑	84
★+			ACME Desarrollo	Backlog	517	Prueba	↑	72

**Figura 17:** Vista del PEP que tiene navegación en 2 dimensiones por la cantidad de columnas de datos.

Hace uso de servicios web inseguros ya que se utilizan identificadores numéricos que son enviados sobre peticiones HTTP GET y sin usar ningún tipo de certificado SSL. Cabe destacar que pasar información por el verbo GET de HTTP es fácil de leer independiente de si se tiene un certificado SSL o no, la seguridad que ofrece SSL se tiene cuando se hace peticiones donde la data viaja en la cabecera (headers) o en el *payload* de la petición, como son las peticiones POST, PUT, PATCH o DELETE.

## Propuesta para el prototipo

Se desea desarrollar una aplicación que sea posible ejecutarla independiente del dispositivo y del sistema operativo de la máquina que la vaya a ejecutar. Para lograr esto se consideró la creación de una aplicación web.

El solo hecho de ser una aplicación web quita la limitación de plataforma ya que permite que la aplicación funcione en cualquier dispositivo conectado a internet que cuente con un explorador web.

De igual forma no se desea hacer como una web tradicional ya que se quiere mantener la naturaleza de funcionamiento de la aplicación al momento de almacenar y traer información. Es por esto que se propone que la aplicación sea un SOFEA (*Service Oriented Frontend Architecture*) o Single Page Application escrita en JavaScript y que sea cliente de un servicio REST.

El crear una aplicación que siga el paradigma de *Single Page Application* tiene como ventaja que toda la aplicación con que interactúa el usuario se encuentra en el frontend y esta solo tiene que hacer peticiones AJAX para traer o enviar información. Las peticiones AJAX en este contexto nos resulta una ventaja ya que las peticiones serían asíncronas permitiendo que el usuario realice otra actividad mientras la petición está en curso.

En muchos casos el solo cambiar una petición síncrona por una asíncrona no es suficiente si de igual forma el usuario tiene que esperar unos segundos para ver el resultado. Se propone también dar retroalimentación al usuario del progreso de las peticiones así como de mostrar el resultando en el frontend antes de obtener una respuesta del servidor. De esta forma la experiencia del usuario se mejora mucho ya que la aplicación se siente más rápido. En caso de tener un error al momento de almacenar se deshace el cambio en el frontend y se muestra una notificación al usuario.

De forma concreta se propone crear un prototipo que cumpla con los siguientes requisitos:

- **Mejorar la experiencia de la pantalla de Login.** Reducir la cantidad de pasos necesarios para entrar.
- **Mostrar las actividades en Tarjetas o Cards.** Esto permitirá mostrar más información y permite una mejor adaptación de contenido entre dispositivos.
- **Desarrollar un diseño Responsive** que se adapte a la pantalla del dispositivo donde se acceda. Permite adaptar la experiencia según el tamaño de pantalla del dispositivo.

- **Creación de un buscador general.** Brindar la posibilidad de buscar globalmente dentro de todas las actividades a las cuales un usuario determinado tiene acceso.
- **Utilizar ventanas modales** para realizar acciones que no necesiten cambiar de pantalla, como lo es el crear una nueva UT o enviar un mensaje a otro usuario de la aplicación.



## Capítulo 5. Prototipo

Para el prototipo se implementó una aplicación *Single Page Application* escrita en JavaScript utilizando el framework Angular JS 1.3 y otras librerías open source como el framework Bootstrap CSS, express, y Node Js.

Además se propuso un rediseño del servicio web para que sea más seguro y liviano. Cabe destacar que como se desea tener una aplicación multi-dispositivo, los servicios web deben ser lo más liviano posible para que tanto la transferencia como el procesamiento de datos sea rápido.

A continuación se detallan los cambios de interfaz como de experiencia de usuario que se introducen con el desarrollo del prototipo.

### Tarjetas para mostrar UT (Cards)

Una de las mejoras que se quería introducir con el prototipo de Tune-Up Process como un Single Page Application era el lograr cambiar el Grid del PEP por otro formato para mostrar cada actividad. Es por esto que se optó por crear tarjetas (o *cards*), las ventajas que permite es la de poder mostrar más información sin tener que hacer scroll horizontal.



Figura 18: Diseño de un card encontrado en el prototipo de Tune-Up Process

En el *card* que se muestra en Figura 18 se puede apreciar toda la información que se puede agrupar. Esta información corresponde al nombre y descripción de actividad, proyecto a que pertenece, personas que tienen tareas asignadas en dicha actividad, número de orden de la tarjeta, cantidad de mensajes pendientes asociados a la tarjeta, cantidad de archivos adjuntos asociados a la tarjeta.

### Cambio de Título de una Actividad desde la Tarjeta

Hacer un cambio en un UT requiere de abrir la ventana de edición que cuenta con toda la información que es posible editar de una actividad. Se desarrolló una especie de atajo para cambiar solo el título de una tarjeta de forma fácil y rápida.



Haciendo *click* en el título de una de estas tarjetas el texto cambia de ser mostrado en una etiqueta de texto y pasa a ser un área de texto editable, como se muestra en la Figura 19. Luego de hacer el cambio, este se almacena presionando la tecla de *ENTER*.



**Figura 19: Muestra el texto en su forma para ser editado.**

## Inicio de Sesión

Se desarrolló una propuesta de inicio de sesión donde se ahorra un paso al usuario y se aprovecha las ventajas que ofrece las llamadas AJAX de forma asíncrona. De esta propuesta solo se benefician los usuarios que tienen cuenta en más de un sitio de Tune-Up Process.

Actualmente si un usuario tiene 2 o más sitios activos el proceso de inicio de sesión tiene 2 pasos, primero le solicita al usuario su dirección de correo electrónico y su clave, segundo trae los sitios al que el usuario pertenece y le muestra un control de selección para que el usuario seleccione el sitio al que quiere acceder.

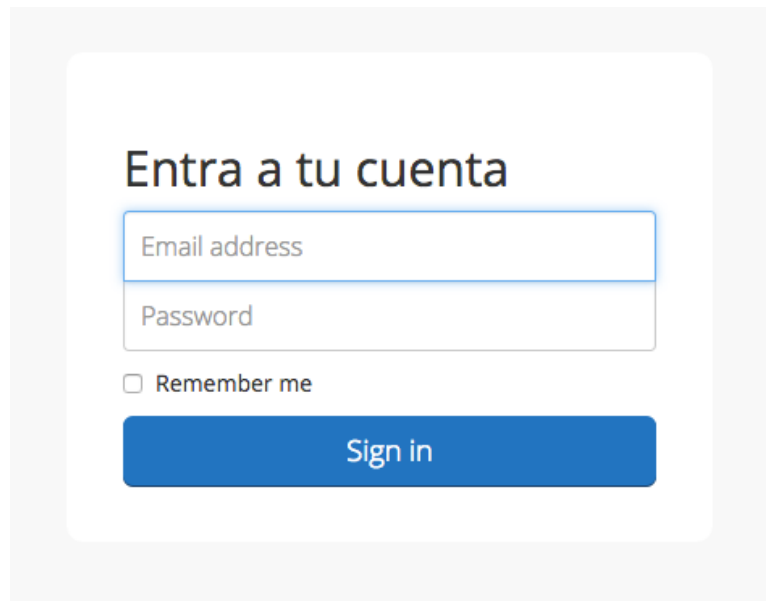
En el prototipo este proceso se realiza de la siguiente forma, el usuario ingresa su dirección de correo electrónico, y al quitar el foco del input de correo electrónico se envía una petición AJAX para determinar si este usuario tiene más de un sitio activo.

A continuación el código encargado de enviar la petición AJAX para determinar si el usuario tiene más de un sitio activo:

```
TuneupData.getSites(username).success(function (response) {
    $scope.sites = response;

    if(response.length > 0){
        $scope.user.site = response[0].IdSitio;
    }
});
```

En la respuesta de la petición se determina si el usuario tiene más de un sitio activo, si solo tiene un sitio activo asume que el usuario va a iniciar sesión en el único sitio que tiene activo.



Entra a tu cuenta

Email address

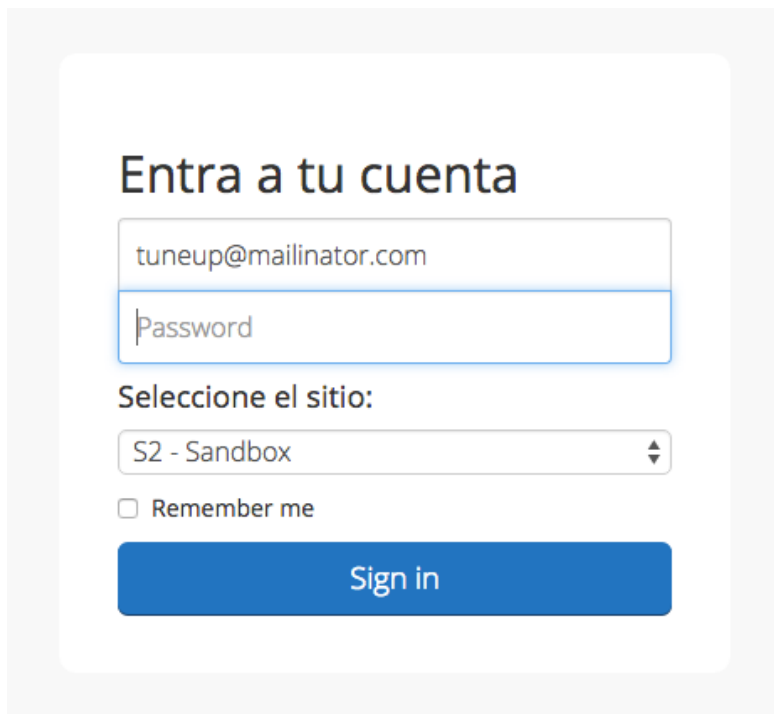
Password

Remember me

Sign in

**Figura 20: Inicio de sesión cuando todavía no se ha determinado los sitios del usuario.**

Si el usuario tiene más de un sitio activo, se activa un selector con las opciones de sitios disponibles antes de que el usuario escriba su clave, como se muestra en la Figura 21. Esto ahorra tiempo al usuario ya que no tiene que hacer un click en la ventana de inicio de sesión antes de seleccionar a que sitio desea ir.



Entra a tu cuenta

tuneup@mailinator.com

Password

Seleccione el sitio:

S2 - Sandbox

Remember me

Sign in

**Figura 21: Inicio de sesión cuando el usuario tiene más de un sitio activo**

Una vez el usuario ingresa sus datos y presiona el botón de iniciar sesión se procede a realizar una llamada AJAX donde se determina si los credenciales

son correctos, de ser así se procede a almacenar el *token* en el *localStorage* de HTML5.

```
TuneupData.login($scope.user).success(function(response){
    if(response !== undefined && response.success === true){
        $scope.user.token = response.token;
        window.user = $scope.user;
        window.user.token = response.token;

        $localStorage.token = response.token;

    }else{
        // Mostrar mensaje de error
    }
});
```

## Seguridad

En el prototipo se hizo uso de la autenticación por *tokens*. Para obtener un *token* basta con realizar una petición POST a la ruta de iniciar sesión del servicio web. Este servicio acepta un objeto JSON con los credenciales de la siguiente forma:

```
{
  "email": "nombre@correo.com",
  "password": "clave_de_cuenta",
  "idsitio": "1"
}
```

Y retorna un objeto JSON como respuesta que sigue la siguiente estructura:

```
{
  "success": true,
  "message": "Sesión creada con éxito",
  "data": {
    "token": "ABCDEFGHIJKLMNOPQRSTUVWXYZ123",
    "agente": {
      "IdAgente": 2,
      "Nombre": "Nombre",
      "Activo": true,
      "Email": "nombre@correo.com",
      ...
    }
  }
}
```

Una vez se obtiene la respuesta con el *token* este se almacena en el Local Storage de HTML5. Esto nos permite tener el *token* disponible para futuras peticiones al servidor de backend.

Cada petición al servidor de backend necesita el envío del *token* para garantizar que la conexión es segura y que el *token* viaja de manera segura a través del internet, el token se envía por medio de la cabecera (*headers* en inglés) de la petición HTTP.

Ejemplo de cabecera enviada al servicio web donde se incluye el *token* de autenticación de usuario:

```
GET/POST/PUT/DELETE /{path} HTTP/1.1
Host: {url}
Token: "ABCDEFGHJKLMNOPQRSTUVWXYZ123"
Content-Type: application/json
```

La mayor ventaja de incluir el *token* en la cabecera de la petición es que si la petición se hace de una forma con encriptación, por ejemplo, utilizando un certificado SSL, toda la cabecera se convierte en un texto encriptado hasta llegar a su destino.

## Diseño Responsive

El prototipo cuenta con un layout que se adapta al tipo de pantalla y dispositivo en que se muestra. Además de que puede adaptar las dimensiones de los componentes también puede mostrar u ocultar componentes para que estos hagan un uso más óptimo de la pantalla.

Si el dispositivo es un Computador Personal o una Tableta, el PEP se muestra en 2 columnas, en la columna de la izquierda está el listado de actividades, y en la columna de la derecha el listado de UT. Estas vistas corresponden a la Figura 22 y a la Figura 23.

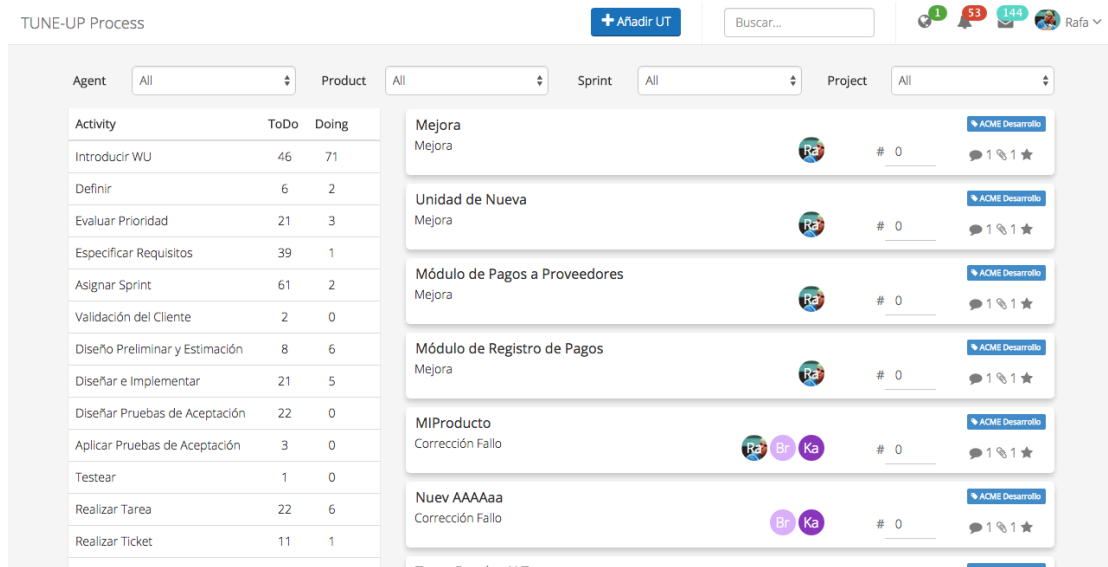


Figura 22: Vista de escritorio

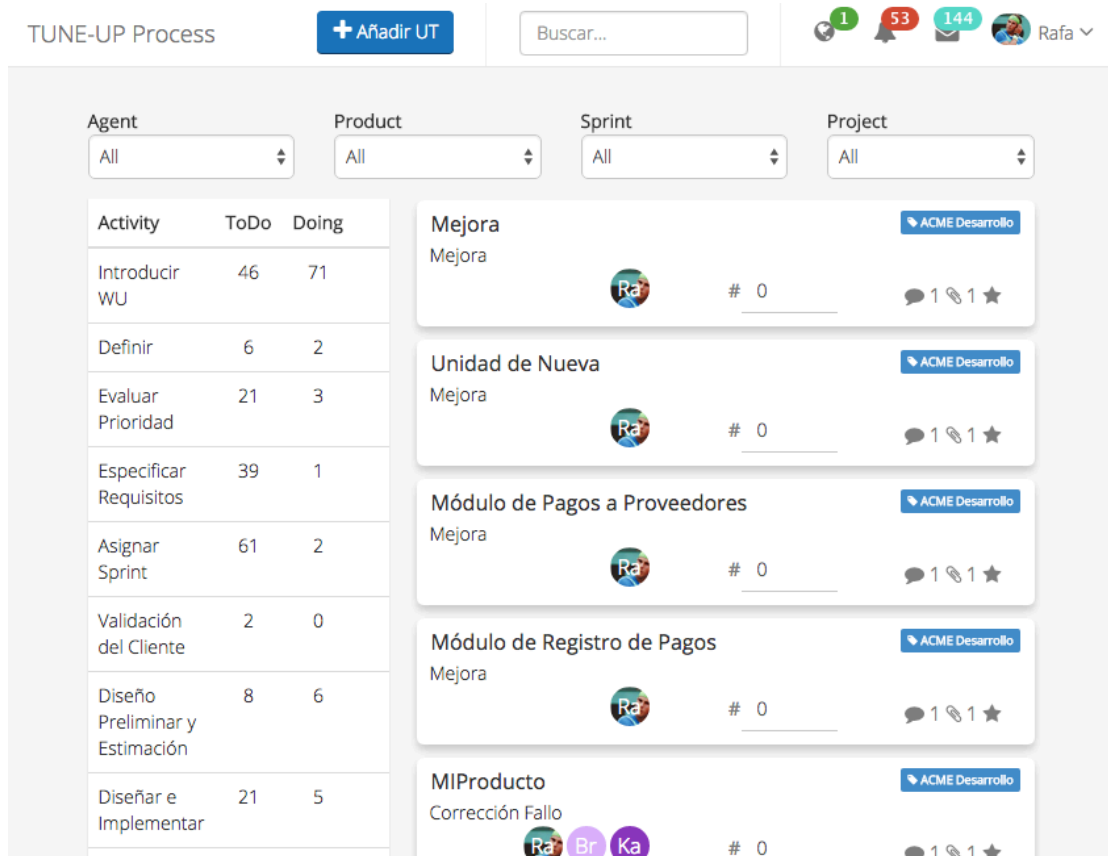
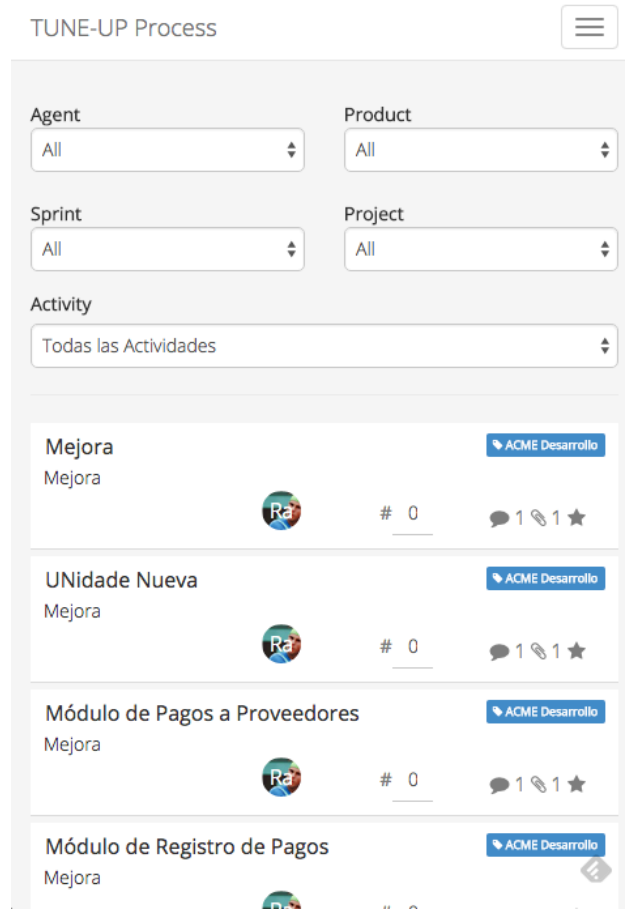


Figura 23: Vista en una Tableta

En la vista móvil solo hay 1 columna que corresponde al listado de UT. Como la vista móvil limita mucho el contenido que se puede mostrar, el listado de actividades pasa a ser de un componente tipo tabla a un componente

dropdown, de esta forma ocupa muy poco espacio, como se muestra en la Figura 24.



**Figura 24: Vista en un Teléfono Inteligente**

Para poder realizar dichas adaptaciones se hizo uso de los *Media Queries* de CSS3 donde se puede especificar qué elementos mostrar/ocultar o adaptar según la resolución de pantalla del dispositivo donde se vaya a mostrar la aplicación.

Para poder mostrar/ocultar el kanban de la izquierda y reemplazarlo por un selector de DropDown en el móvil se implementó el siguiente código, donde se especifica para que dimensiones los objetos se tienen que mostrar u ocultar.

```
.kanban-dropdown {
    display: none;
}

@media (max-width: 767px) {
    .kanban-dropdown {
        display: visible;
    }

    div.kanban {
        display: none;
    }
}
```

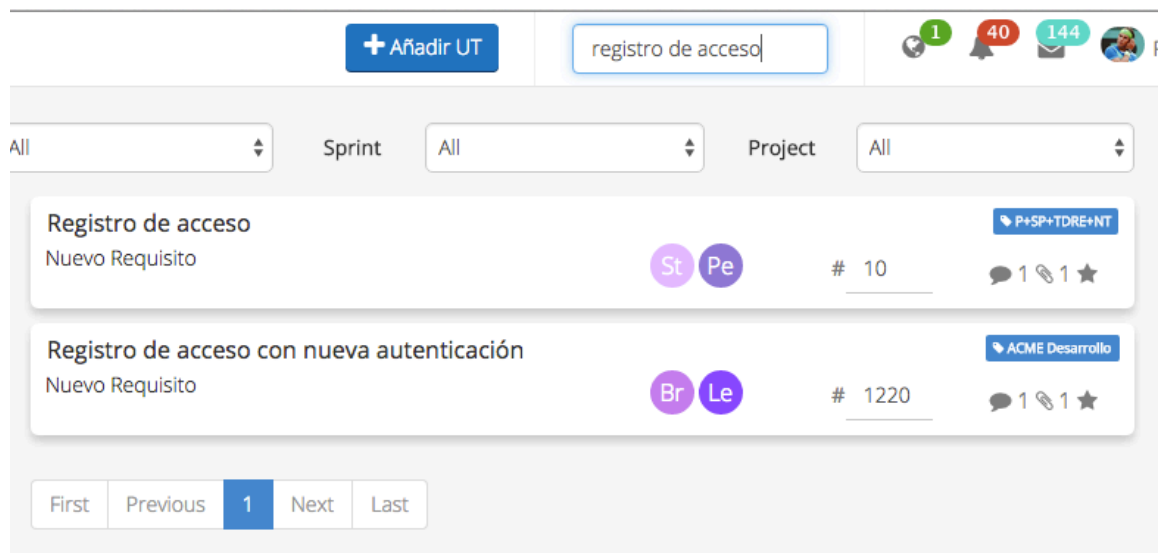
En este ejemplo nos aseguramos que el dropdown no se muestre, y solo cuando una pantalla tiene menos de 767px hacemos que el dropdown se muestre y la división HTML que contiene el Kanban se oculte. De esta forma se puede tener un HTML que se adapte a la pantalla que se va a mostrar.

## Buscador General

La búsqueda actual en la herramienta de Tune-Up Process es un poco lenta, ya que hay que buscar de forma manual entre los cards.

Otra característica con la que se quería experimentar era la posibilidad de tener un buscador general que busque entre los *cards* alguna palabra que se encuentre en todo el texto que estos contienen.

En el prototipo se agregó un buscador en la barra de navegación horizontal que se encuentra en la parte superior de la pantalla, esta búsqueda es sumamente rápida y eficiente.



**Figura 25: Ejemplo de búsqueda general**

Para implementar la búsqueda se hizo uso de diferentes componentes de Angular JS para poder realizar una comunicación entre diferentes componentes. Resulta que tanto la barra de navegación superior y el listado de cards que aparece en la parte inferior pertenecen a módulos separados. Para poder comunicar estos componentes fue necesario el uso de las funcionalidades de Emit y Broadcast que tiene implementado Angular JS. Estas funcionalidades nos permite enviar mensajes que luego serán escuchados en otra sección de la aplicación.

```
var broadcastText = _.debounce(function(){
    $scope.$broadcast('searchText',{
        value: $scope.searchText
    });
}, 400);

$scope.$watch('searchText', broadcastText);
```

En este ejemplo se crea una función que observa el valor que tenga contenido el input del buscador, mientras el buscador va cambiando de valor la función se detiene a esperar por 400 milisegundos, si en este tiempo se pulsa una tecla el contador se reinicia y la función no envía el texto. Una vez pasen los 400 milisegundos sin el usuario pulsar ninguna otra tecla, entonces se procede a enviar un mensaje con el texto de búsqueda a todos los posibles módulos que estén escuchando por ese tipo de mensaje.



## Drag&Drop para reorganizar

En la herramienta Tune-Up Process cada actividad tiene un área de texto donde se puede escribir y especificar su posición de orden, también se puede reorganizar el Grid tomando cada actividad en el límite del borde izquierdo y realizar un *drag & drop*.

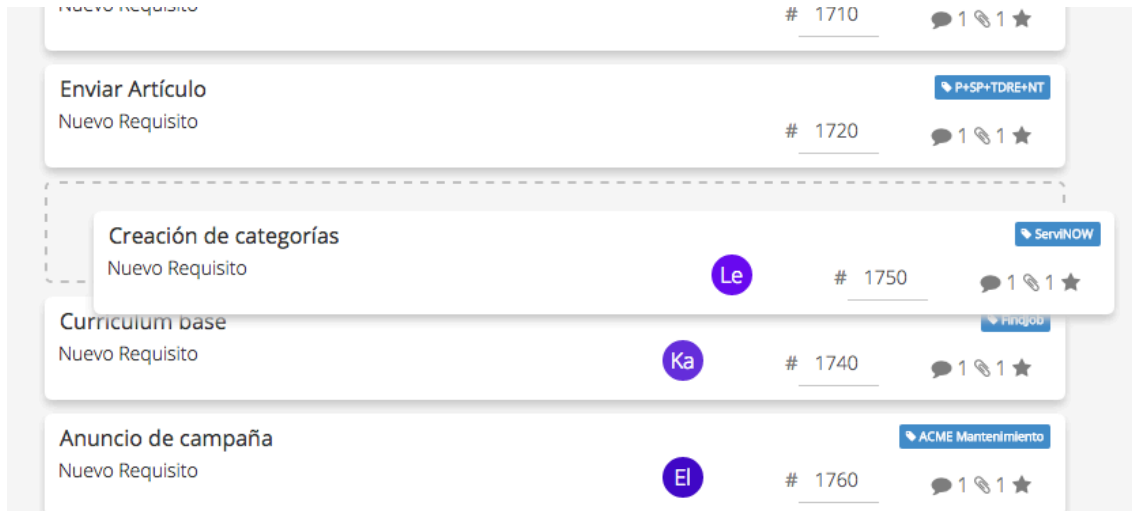
Para el prototipo se quería llevar dicha funcionalidad a los *cards*, la capacidad de cambiar el orden en que aparecen no solo con un área de texto sino también por medio de *drag & drop* de todo el *card* a una nueva posición. A continuación el código que se encarga de ordenar las actividades por número de orden:

```
return activities.slice().sort(function(a, b){
    var aOrden = parseInt(a.Orden || 0);
    var bOrden = parseInt(b.Orden || 0);

    if(aOrden < bOrden) return -1;
    if(aOrden > bOrden) return 1;
    return 0;
});
```

Cuando se toma un *card* para desplazarlo, se muestra lo que es un *placeholder* en el sitio donde quedaría el *card* si es dejado caer por el usuario. Al dejarlo caer se actualiza su número de orden que es calculado como el número entero promedio entre el orden que tiene el *card* que lo precede y el que se encuentra justo después.

También se mantiene el funcionamiento de poder escribir en el cuadro de texto el número de orden nuevo que se le quiere asignar al *card*.



**Figura 26: Muestra como funciona el Drag & Drop para reorganizar el orden de las actividades**

## Creación de UT

La unidad de trabajo (UT), o Working Unit (WU) en inglés, es uno de los conceptos básicos de la metodología Tune-Up Process y se refiere a las unidades de trabajo que se incluyen en una iteración del ciclo de desarrollo.

Según la definición formal que da el autor Patricio Letelier “Usamos este concepto para englobar cualquier tipo de cambio en el producto (nuevo requisito, mejora de requisito existente o corrección de un fallo), y también para referirnos a cualquier otra tarea que se enmarque en el trabajo de una iteración” [36].

En el prototipo se agregó la posibilidad de que el usuario pueda ingresar una UT de una forma fácil y solo especificando la información más básica, en la Figura 27 se muestra la ventana modal.

The image shows a modal window titled "Nueva UT" (New User Story) with the following fields:

- Nombre:** A text input field containing the placeholder "Nombre".
- Producto:** A dropdown menu with "ACME Desarrollo" selected.
- Tipo:** A dropdown menu with "Mejora" selected.
- Sprint:** A dropdown menu with "Backlog" selected.
- Proyecto:** A dropdown menu with "Test Katy" selected.
- Workflow:** A dropdown menu with "WF Desarrollo Básico" selected.
- Descripcion:** A large text area for entering the description.

At the bottom right of the modal, there are two buttons: "Crear" (Create) in blue and "Cancelar" (Cancel) in grey.

**Figura 27: Ventana modal para crear una UT nueva**

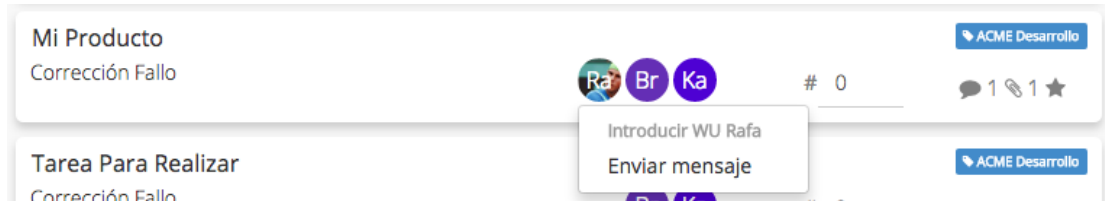
Para crear el mensaje se envía una petición AJAX al servidor que contiene la información de la modal, para esto se utiliza el servicio TuneupData. A continuación código implemented:

```
TuneupData.crearUT(newUT).success(function (data){
    if(data !== undefined && data.success === true){
        // UT fué creada
    }

    // ocurrió un error, mostrar mensaje
});
```

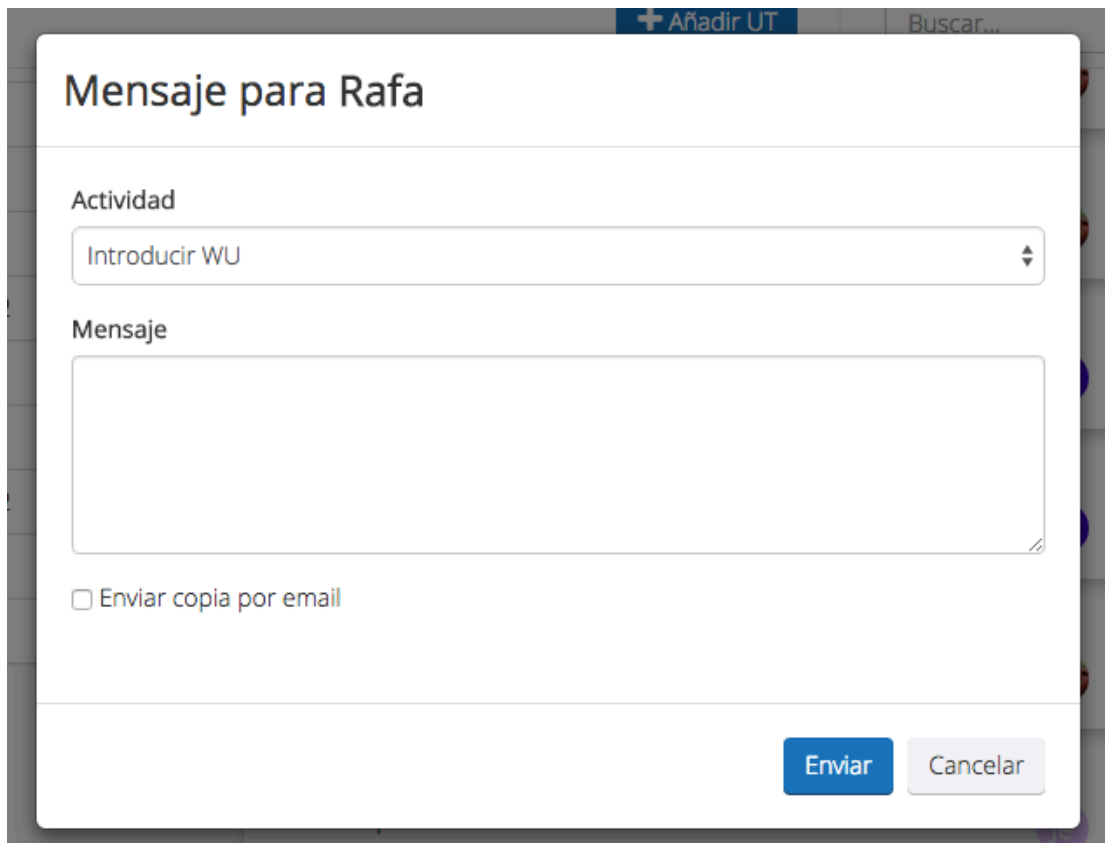
## Enviar Mensaje

Para realizar acciones pequeñas muchas veces no es necesario salir de la página actual para mostrar un formulario, se puede aprovechar el uso de ventanas modal. Estas son ventanas que se superponen sobre el contenido y permiten realizar acciones desde esta sin tener que cambiar la ventana donde el usuario se encuentra, un ejemplo de esta es como se muestra en la Figura 29.



**Figura 28: Menú con acciones directas a un usuario**

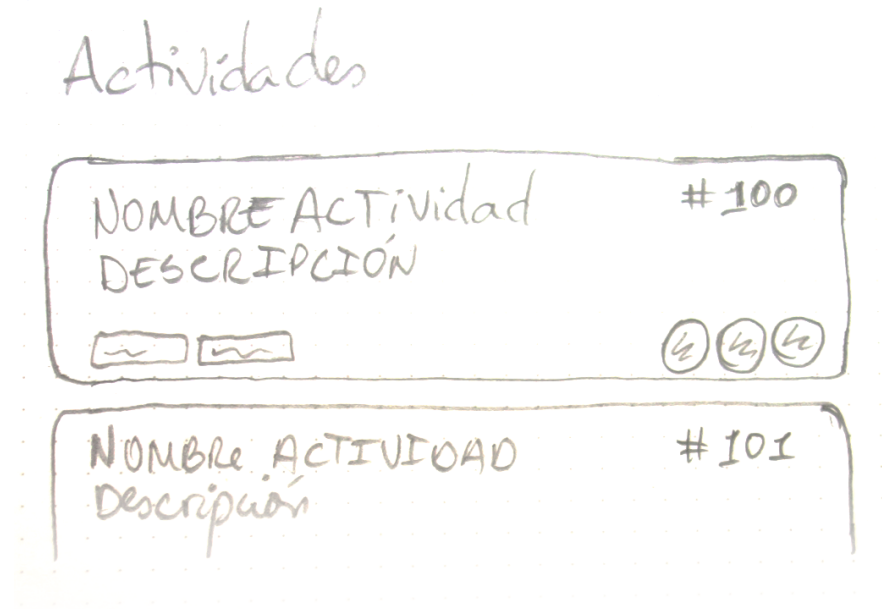
Actualmente en el prototipo la única acción directa que se puede hacer en un usuario es la de enviar un mensaje directo al usuario que se le hizo click como se muestra en la Figura 28.



**Figura 29: Ventana modal para enviar un mensaje nuevo desde cualquier parte de la aplicación.**

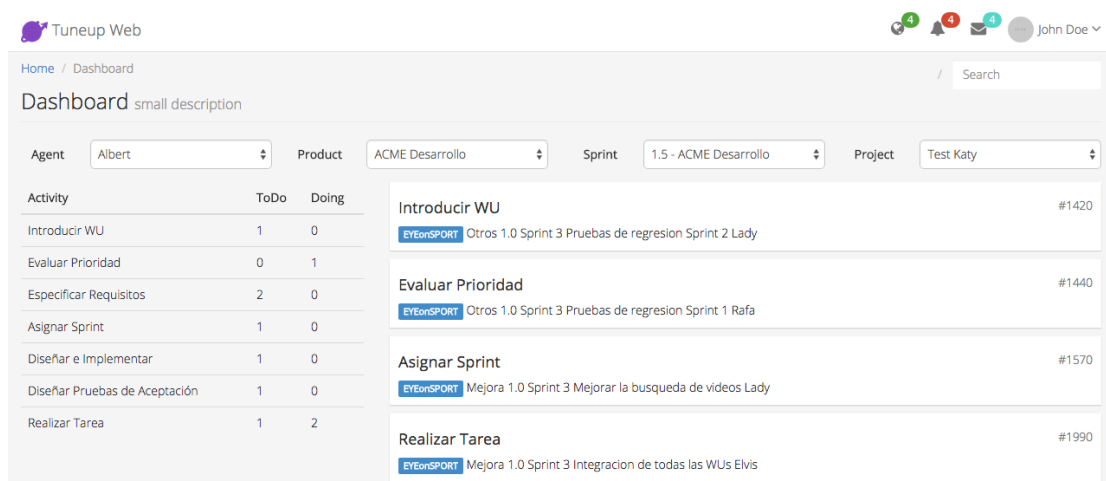
## Evolución del Prototipo

La evolución más grande en el prototipo se encuentra en la parte de las actividades que muestra las tarjetas o *cards*, la idea de dicho módulo primero se desarrolló sobre papel como se muestra en la Figura 30.



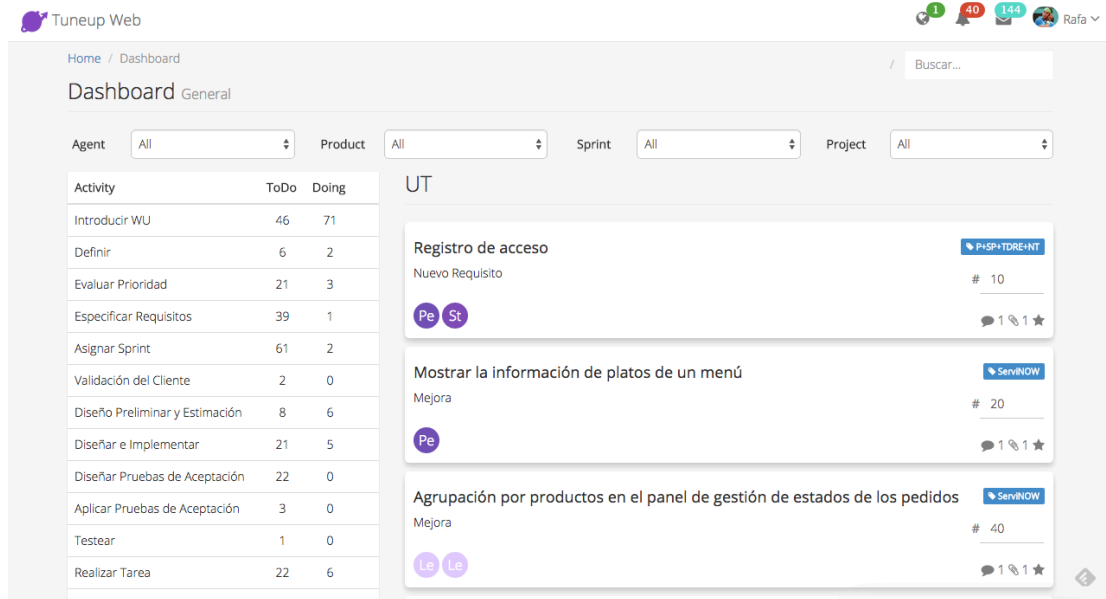
**Figura 30: Uno de los primeros bocetos en papel de como reorganizar las actividades**

La primera versión del prototipo solo mostraba la información básica de cada UT, en esta primera versión se empezó a contemplar la idea de poder resaltar alguna información importante de cada *card* como a que producto pertenece, entre otra información, esta primera versión funcional en digital se puede ver en la Figura 31.



**Figura 31: Primera versión funcional del prototipo**

Se hicieron los cambios necesarios para que los cards se parezcan al boceto en papel, en esta iteración que se muestra en la Figura 32, se agrega un icono que muestra los usuarios que tiene alguna actividad asignada de la UT, también se agrega la posibilidad de editar el orden. El problema que presento esta iteración es que los *cards* crecieron en tamaño y por ende solo se mostraban muy pocos *cards* en la pantalla por momento.



**Figura 32: Primer rediseño de cards que se parece al boceto en papel**

En la siguiente iteración se trató de reducir el tamaño de los *cards* al mínimo posible buscando optimizar para el tamaño de la pantalla, de esta forma se puede mostrar más contenido en la misma área de pantalla, se puede ilustrar esa revisión en la Figura 33.

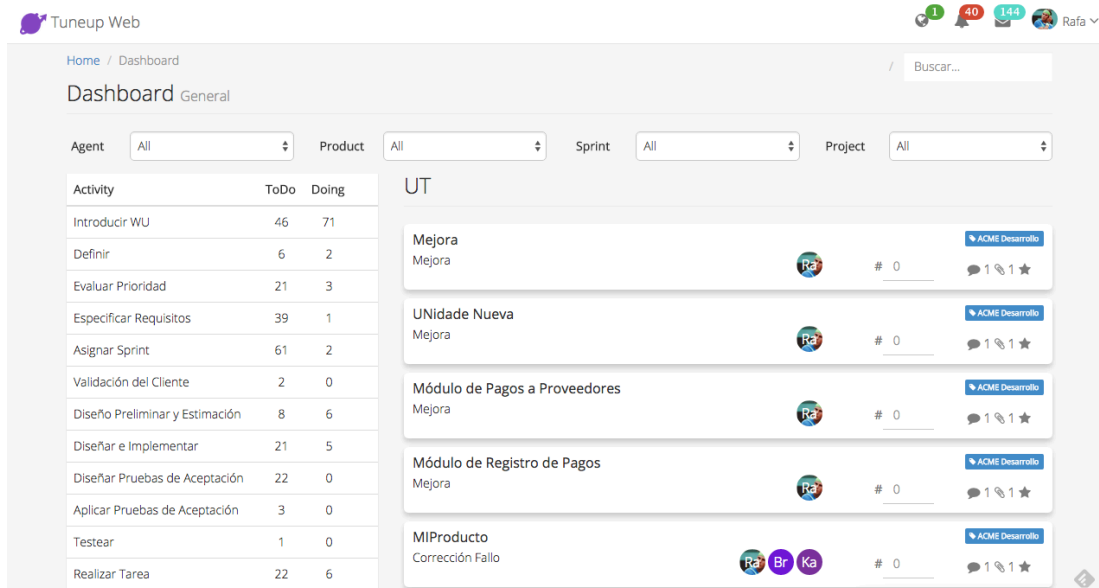


Figura 33: Versión con Cards más pequeños

En la última iteración se introdujo un cambio que optimizó más el espacio en pantalla, se removió el título que decía "Dashboard" lo cual hizo que se puedan mostrar aún más *cards* en pantalla. También se hace mejor uso de la barra de navegación en la parte superior, ya que dentro de esta se agregó la opción de "Añadir UT" y también el Buscador General, como se puede ver en la Figura 34.

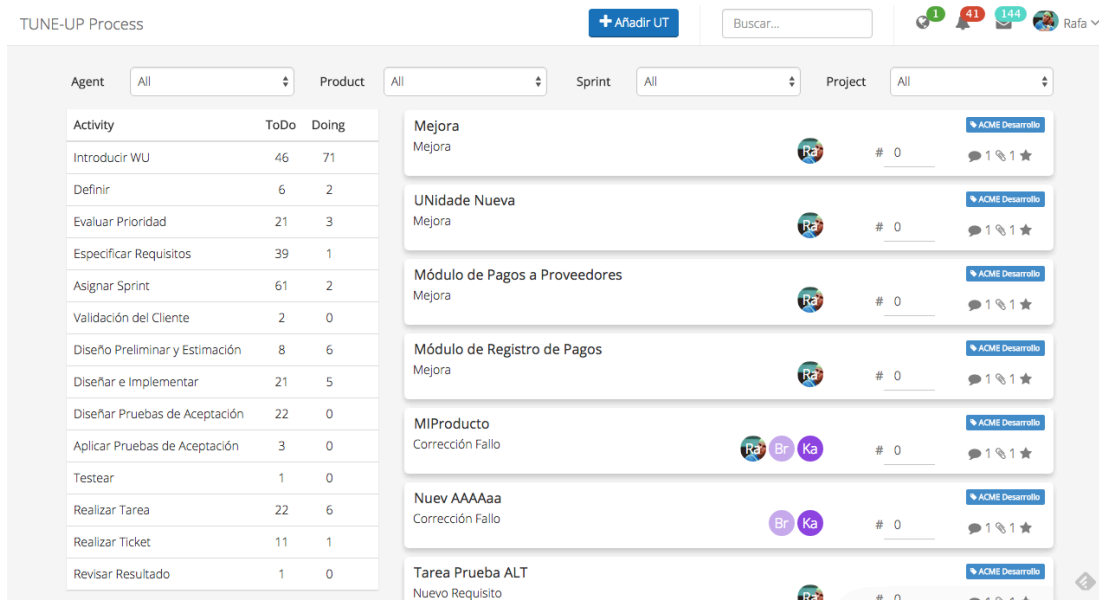


Figura 34: Versión final del prototipo, "Buscar" se mueve a la barra de arriba

## Generación de CSS y optimización de código Javascript

Para la generación de código CSS a partir de código Stylus y para la optimización de código JavaScript se utilizaron distintas tareas de Grunt. Estas tareas luego se configuraron dentro de *Grunt* para que fueran ejecutadas de forma automática cuando se realizaba algún cambio en el código.

El uso de *Git Hooks* en el prototipo fue necesario para hacer a *Git* el responsable de ejecutar las tareas de *Grunt*, de esta forma se podía automatizar un poco más el flujo de trabajo y ganar unos cuantos segundos de productividad.

Grunt define las tareas en un archivo llamado Gruntfile. Este archivo especifica cada tarea como un objeto JSON, a estas tareas luego se le puede dar un nombre y ejecutarlas directamente por el nombre asignado. A continuación un poco de la implementación de tareas en el Gruntfile del prototipo:

```
grunt.initConfig({
  stylus: {
    compile: {
      files: [{
        expand: true,
        cwd: './app/assets/stylesheets',
        src: ['*.styl'],
        dest: './app/assets/stylesheets',
        ext: '.css'
      }]
    }
  },
  uglify: {
    build: {
      files: [{
        src: './ javascript/application.min.js',
        dest: './ javascript/application.min.js'
      }]
    }
  },
  // Otras tareas
  // ...
});
```



Para poder ejecutar las tareas de Grunt es necesario registrarlas. Una vez registradas se pueden llamar dichas tareas desde la terminal o consola del ordenador.

```
// Stylesheets Tasks
grunt.registerTask('stylesheets', ['stylus', 'cssmin']);
// JS Tasks
grunt.registerTask('js', ['concat:js', 'uglify']);
```

## Comparación prototipo y versión actual de TUNE-UP

Para realizar una pequeña comparación de las mejoras que puede traer un rediseño y replanteamiento se optó por medir el tiempo de carga y tiempo de primera respuesta en la aplicación. Para realizar el experimento en condiciones iguales se utilizó un cronómetro para medir el tiempo en cada aplicación. Esta forma de experimento tiene como ruido el tiempo de respuesta de la persona que controla el cronómetro.

Antes de hacer la prueba, algo a tener en cuenta es que TUNE-UP busca actualizaciones automáticamente al inicio del programa, esto es bueno en el sentido que siempre trata de que el usuario final tenga la última versión, pero resulta malo para el usuario que solo quiere entrar a ver una tarea en específico, ya que tiene que esperar que la aplicación se actualice para poder acceder al sistema.

Por el contrario, el prototipo web, por su naturaleza web, siempre brinda al usuario la última versión disponible sin el usuario tener que instalar o actualizar ningún programa.

## Tiempo de Primera Respuesta y Tiempo total de Carga

### TUNE-UP Actual

La primera respuesta se caracteriza por cambiar la ventana de inicio de sesión por la ventana donde se va a desplegar la información, en la primera respuesta se muestra la ventana en blanco hasta que termina de cargar todo el contenido. El tiempo de carga final es cuando se termina de cargar todos los datos de actividades, kanban e imágenes de perfil de los usuarios, y ya la aplicación está lista para ser usada.

	Tiempo Primera Respuesta (segundos)	Tiempo Carga Total (segundos)
Prueba 1	13.28	16.66
Prueba 2	11.52	13.26
Prueba 3	22.15	24.11
Prueba 4	11.78	13.98
Prueba 5	15.34	17.48
Prueba 6	10.25	12.52
Prueba 7	9.67	11.81
Prueba 8	11.63	13.52
Prueba 9	13.79	15.96
Prueba 10	10.52	12.5
Prueba 11	9.33	11.51
Prueba 12	10.7	12.74
Prueba 13	11.83	14
Prueba 14	10.98	12.98
Prueba 15	9.99	11.98
Promedio	12.18	14.33
Mediana	11.52	13.26
Tiempo más alto	22.15	24.11
Tiempo más bajo	9.33	11.51

## Prototipo TUNE-UP Web

La primera respuesta se caracteriza por entrar al usuario dentro de la aplicación y se muestra la información de las actividades. Al igual que en TUNE-UP actual, el tiempo de carga final es cuando se termina de cargar todos los datos de actividades, kanban e imágenes de perfil de los usuarios, y ya la aplicación está lista para ser usada.

	Tiempo Primera Respuesta (segundos)	Tiempo Carga Total (segundos)
Prueba 1	1.43	3.32
Prueba 2	1.64	2.65
Prueba 3	1.97	3.58
Prueba 4	2.28	3.83
Prueba 5	2.06	3.46
Prueba 6	1.73	3.23
Prueba 7	1.95	3.35
Prueba 8	2.2	3.51
Prueba 9	1.95	3.64
Prueba 10	2.93	4.2
Prueba 11	2.58	4.11
Prueba 12	2.16	3.44
Prueba 13	1.97	3.63
Prueba 14	1.91	3.47
Prueba 15	2.72	4.09
Promedio	2.10	3.57
Mediana	1.97	3.51
Tiempo más alto	2.93	4.20
Tiempo más bajo	1.43	2.65

Esta información resalta que en promedio el prototipo da una primera respuesta o *feedback* al usuario en un tiempo que es aproximadamente 83% más rápido que el tiempo actual y carga todos los datos por completo en un 75% más rápido.

Para poder obtener una visión más general de los datos, se generaron gráficas que muestren la información una al lado de la otra. En las gráficas el eje X representa el número de prueba, mientras que el eje Y representa el tiempo en segundos. Lo ideal es que la gráfica esté lo más cercana al eje X ya que eso indica tiempos bajos de carga.

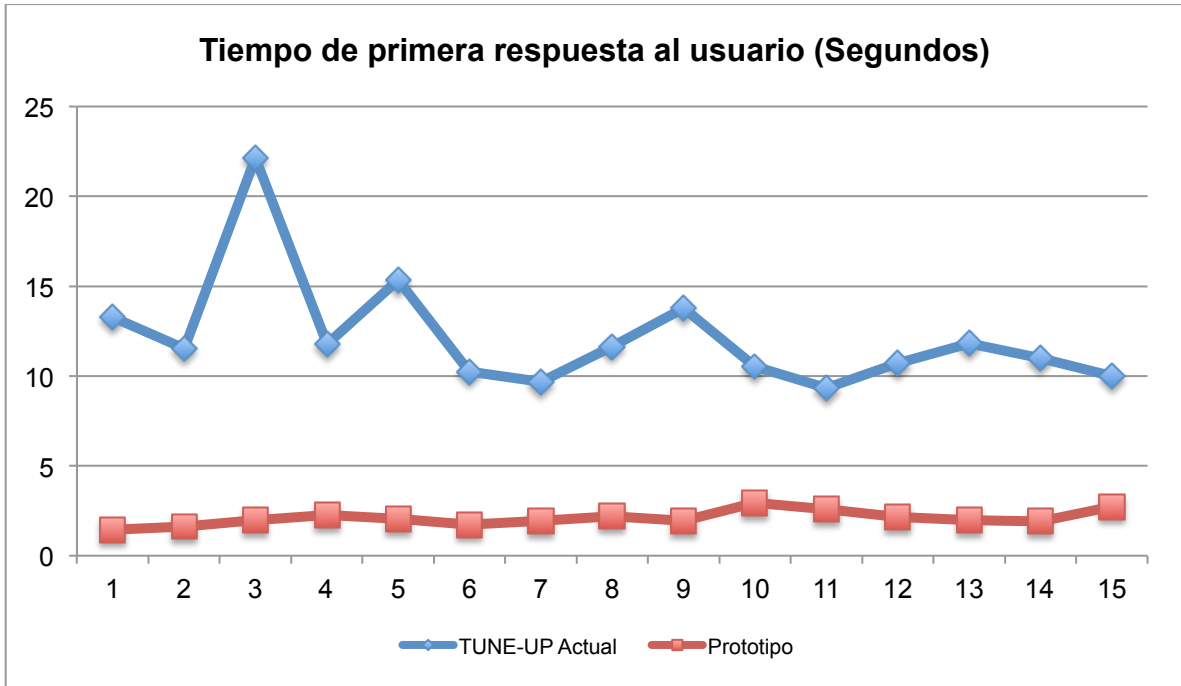


Figura 35 Gráfica que muestra los tiempos de el primer *feedback* al usuario

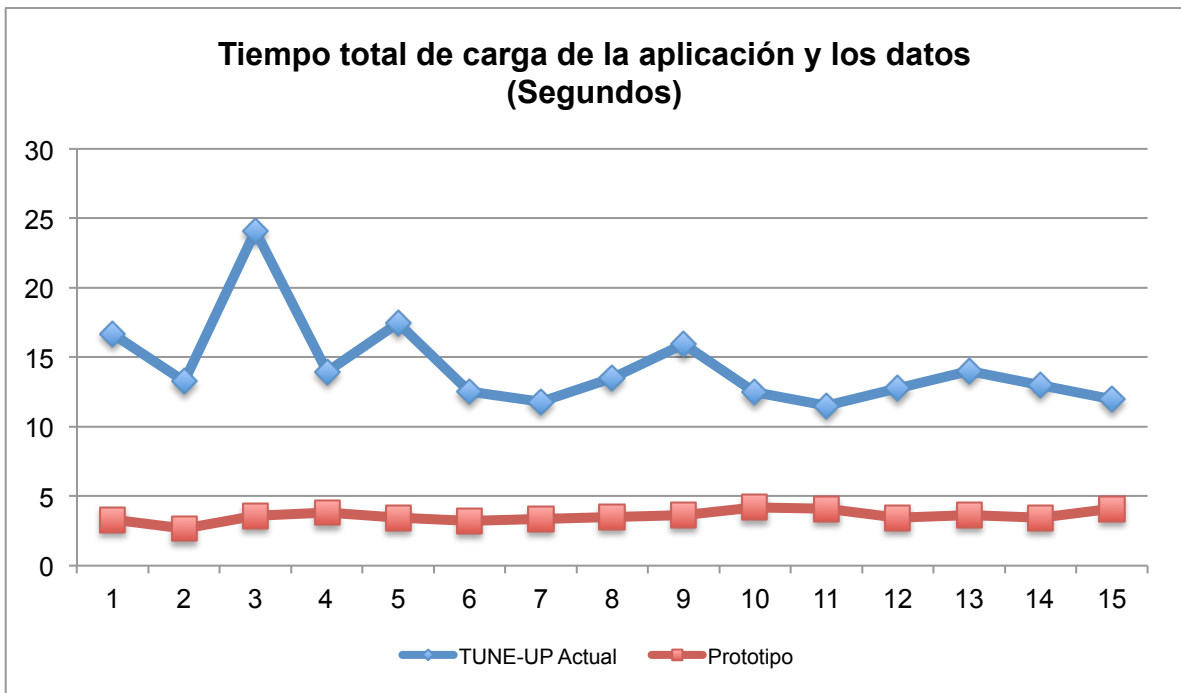
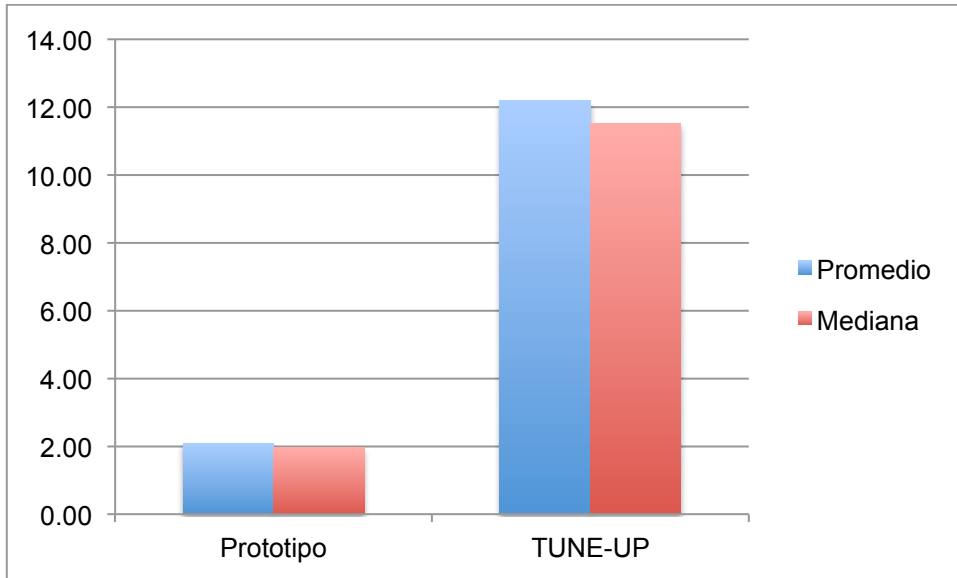
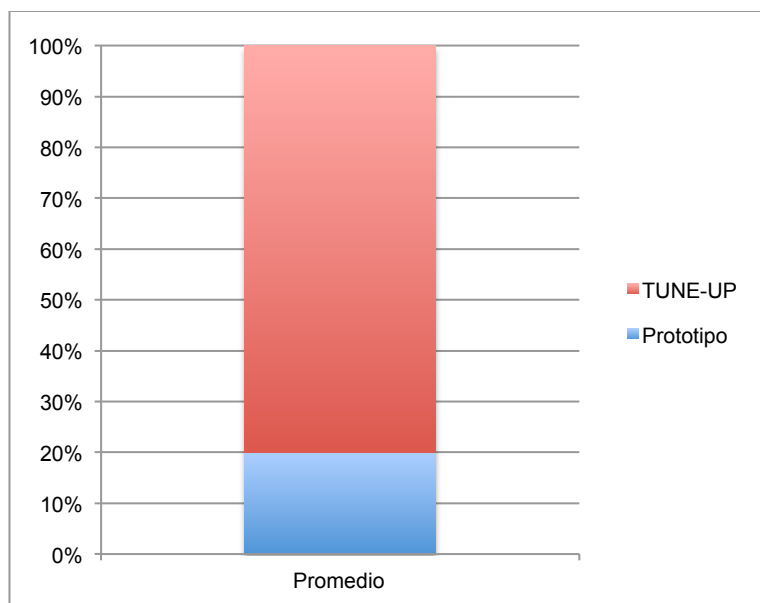


Figura 36 Gráfica que muestra los tiempos de carga total de la aplicación y los datos



**Figura 37 Muestra el promedio y mediana de los tiempos del experimento**

En los datos se puede observar una diferencia considerable entre los promedios de cada prueba. Junto con los promedios también se obtuvo la mediana que muestra un número que podría considerarse más lo que sería un tiempo común al empezar a utilizar cualquiera de estas aplicaciones. En el caso de TUNE-UP la mediana resulta un poco más bajita ya que este tuvo un resultado que se podría considerar como una anomalía ya que fue el doble de los demás resultados.



**Figura 38 Muestra la diferencia en porcentaje entre los tiempos promediados de carga total entre TUNE-UP y el prototipo desarrollado**

Esta gráfica muestra como el tiempo de respuesta en el prototipo resulta ser un 20% del tiempo total que toma TUNE-UP para terminar de cargar, esto se traduce en que un usuario normal de TUNE-UP podría empezar a trabajar un 80% de tiempo más rápido sí utiliza la versión del Prototipo.

## Conclusiones y trabajo futuro

El rediseñar una aplicación de escritorio a una aplicación web tiene más ventajas que desventajas, el solo hecho de poder utilizar la aplicación en cualquier dispositivo que tenga un navegador web es la ventaja más importante. El prototipo desarrollado nos permite validar algunas de estas ventajas. Es importante destacar que el prototipo en su funcionalidad es totalmente operativo.

En aplicaciones web se puede aprovechar el diseño *responsive* para poder adaptar y optimizar la interfaz de una aplicación a diferentes tamaños de pantalla, esto nos permite personalizar la experiencia del usuario así como que tipo de elementos utilizar para las diferentes opciones.

Una aplicación multiplataforma también se puede pensar como una serie de aplicaciones nativas donde todas consumen el mismo servicio web o *Web API*. Existen ventajas y desventajas de realizar la aplicación utilizando cada uno de los acercamientos. La ventaja más grande que tienen las aplicaciones nativas es el acceso al hardware de los dispositivos, como lo son la cámara, el GPS, y las notificaciones *Push*. Mientras que una aplicación web no tiene acceso a estos, la aplicación web tiene la ventaja que no necesita instalarse en el dispositivo y las actualizaciones son invisibles para el usuario.

Al momento de desarrollo cada producto o servicio debe implementar la aplicación más adecuada a su tipo de negocio, ya sea nativa o una *Responsive Web App*. Debido al tipo de aplicación que es el caso de estudio la ventaja de una aplicación nativa que aprovecharía TUNE-UP sería la de poder utilizar y enviar notificaciones *Push* y así poder tener a sus usuarios más conectados.

La experiencia de un usuario con un producto de software está relacionado con diversos factores, así como se necesita un agradable diseño y responder a los procesos cognitivos de una persona, también se debe tomar en cuenta la parte emocional y es en esta parte donde resulta importante que la aplicación responda a las necesidades del usuario para poder generar cierta satisfacción en este.

La comunicación asíncrona en una aplicación web permite mejorar la experiencia de uso, no solo porque permite realizar la petición sin refrescar la página, sino que también la petición se realiza de forma más rápida que una petición normal. Esto es porque la respuesta del servidor resulta mucho más

pequeña que la respuesta de un sitio web completo que incluye HTML, CSS y JavaScript.

Otra forma de comunicación entre cliente y servidor de forma asíncrona es la de WebSockets, esta se introdujo por primera vez durante el desarrollo de HTML5, de acuerdo a su sitio web este «define una conexión full-duplex dentro de un socket en el cual mensajes pueden ser enviados y recibidos entre cliente y servidor. El estándar de WebSocket simplifica la complejidad en la comunicación web bi-direccional y su administración.» [37] Esta tecnología representa una evolución en la comunicación entre el cliente y servidor al ser comparada con AJAX pero no representa un sustituto a AJAX ya que cada uno opera de forma diferente y traen ventajas y desventajas.

Como reflexión personal, el desarrollo de este prototipo me ha servido para validar las ventajas que tiene el desarrollo web sobre una aplicación *desktop* y como el diseño de interfaces *responsive* con *mobile first* permite el desarrollo de una sola aplicación que pueda ser accesada desde múltiples dispositivos y plataformas. El uso de estas tecnologías me dio una mejor perspectiva sobre como actúan los frameworks para el *front-end*, y me permitió mejorar mis conocimientos en este tipo de tecnologías del lado del cliente. También me hizo entender que aunque el desarrollo de una aplicación del lado del cliente sea una buena separación entre la capa de presentación y la capa de manejo de datos, no quiere decir que sea una bala de plata y se deba implementar en todos los desarrollos de aplicaciones web.

Respecto del prototipo y su continuación de desarrollo, algunas de las mejoras que se le puede hacer a la aplicación:

- **Hacer uso de WebSockets para comunicación en tiempo real:** Una evolución de la aplicación es reemplazar los componentes que necesitan comunicación en tiempo real por WebSockets, los componentes en específico del prototipo que se pueden beneficiar de dicha tecnología son las Alertas, Notificaciones y Alarmas. Ya que estos pueden mejorar la experiencia de usuario y mejorar la comunicación entre un equipo en tiempo real.
- **Implementar certificado SSL para tener una conexión segura:** Al hacer uso de tokens y estos ser enviados a través de la cabecera HTTP es necesario hacer uso de una conexión segura con un certificado SSL. Esto garantiza que la cabecera sea encriptada antes de ser enviada y al mismo tiempo dificulta la lectura de la data por parte de quien intercepte la comunicación.
- **Hacer uso de la Caché del lado del servidor:** Implementar una Caché de la data del lado del servidor va a garantizar que los tiempos de carga sean bajos y le dará una mejor experiencia al usuario final,



también que el implementar *caché* resulta en que los servidores tengan que traer y procesar menos data de la base de datos.

- **Implementar paginación en los resultados del Servicio Web:** Actualmente el servicio retorna todos los recursos de actividades en una sola petición, esto puede causar que un número muy alto de actividades sea retornado, y por ende un número alto de data. Al implementar un sistema de paginación solo se retorna el grupo de actividades que pertenezcan a una sola página. Esto reduce la cantidad de data que es enviada por parte del servidor y al mismo tiempo incrementa la velocidad de respuesta de servidor al cliente.
- **Utilizar librerías para generar gráficos en JavaScript y HTML5,** esto permite la flexibilidad de mantener el sitio con un diseño *responsive* y también permite que las gráficas sean mostrados en todo tipo de dispositivos. La única desventaja que presentan estos tipos de librerías es que dificultan la impresión de una página web, por suerte, no es una tarea muy común.

## Referencias

- [1] Ethan Marcotte. (2010, Mayo) A LIST APART. [Online]. <http://alistapart.com/article/responsive-web-design>
- [2] Florian Rivoal. (2012, Junio) W3C. [Online]. <http://www.w3.org/TR/css3-mediaqueries/>
- [3] J. J GARRET. (2005, Febrero) Adaptive Path. [Online]. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/>
- [4] Ganesh Prasad. (2007, Octubre) Life Above The Service Tier. Libro Electronico.
- [5] Laura Elizabeth. (2015, Julio) SitePoint. [Online]. <http://www.sitepoint.com/web-desktop-apps/>
- [6] Que es UX? [Online]. <http://www.fatdux.com/es/what/what-is-ux>
- [8] Yusef Hassan Montero and Francisco J Martín Fernández. (2005, Septiembre) no solo usabilidad: revista sobre personas, diseño y tecnología. [Online]. [http://www.nosolousabilidad.com/articulos/experiencia\\_del\\_usuario.htm](http://www.nosolousabilidad.com/articulos/experiencia_del_usuario.htm)
- [7] C.J. Overbeeke, S.A.G. Wensveen J.P. Djajadiningrat. Augmenting fun and beauty. [Online]. [http://www.researchgate.net/publication/234816106\\_Augmenting\\_fun\\_and\\_beauty](http://www.researchgate.net/publication/234816106_Augmenting_fun_and_beauty)
- [9] Xavier Ferré Grau, "Principios Básicos de Usabilidad para Ingenieros Software," Facultad de Informática, Universidad Politécnica de Madrid, Boadilla del Monte,. [Online]. <http://www.eici.ucm.cl/Academicos/ygomez/descargas/calidad/usabilidad.pdf>
- [10] J Preece, *Interaction Design: Beyond Human-Computer Interaction*. New York: Wiley, 2002.
- [11] Jakob Nielsen. (2001, Agosto) First Rule of Usability? Don't Listen to Users. [Online]. <http://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/>

- [12] Janet M. Six. (2014, Noviembre) Fundamental Principles of Great UX Design. [Online]. <http://www.uxmatters.com/mt/archives/2014/11/fundamental-principles-of-great-ux-design-how-to-deliver-great-ux-design.php>
- [13] Carlos Jiménez. Platzi. [Online]. <https://platzi.com/blog/como-disenar-formularios/>
- [14] Jakob Nielsen. (2005, Julio) Nielsen Norman Group. [Online]. <http://www.nngroup.com/articles/scrolling-and-scrollbar/>
- [15] E. Marcotte, "Responsive web design.," *A list apart*, 2010.
- [16] MIGHTYminnow. (2013, Noviembre) MIGHTYminnow. [Online]. <http://www.mightyminnow.com/2013/11/what-is-mobile-first-css-and-why-does-it-rock/>
- [18] Jerry Cao. (2015, Junio) The Next Web. [Online]. <http://thenextweb.com/dd/2015/06/16/how-cards-are-taking-over-web-design/>
- [17] Paul Adams. Intercom. [Online]. <https://blog.intercom.io/why-cards-are-the-future-of-the-web/>
- [19] Robert David Idol. Looking Beyond HTTP: The Future of Web Application Protocols. [Online]. <http://cs.unc.edu/~mxrider/papers/idol-looking-beyond-http.pdf>
- [20] Shruti M. Rakhunde. Real Time Data Communication over Full Duplex Network Using Websocket. [Online]. <http://www.iosrjournals.org/iosr-jce/papers/ICAET-2014/volume-5/3.pdf>
- [21] Sebastian Porto. (2014, Mayo) Reinteractive. [Online]. <https://www.reinteractive.net/posts/186-lessons-learnt-by-building-single-page-applications>
- [22] ROBERT DAMPHOUSSE. (2015, Febrero) StormPath. [Online]. <https://stormpath.com/blog/token-auth-spa/>
- [23] Tino Tkalec. TOPTAL. [Online]. <http://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs>

- [24] Igor Minar. (2012, Jul) Angular JS - Google Plus. [Online]. <https://plus.google.com/+AngularJS/posts/aZNVhj355G2>
- [25] Node.js Foundation. (2009, Julio) Node.js. [Online]. <https://nodejs.org/>
- [26] Express.js. (2010, Enero) Express - Node JS Web Application Framework. [Online]. <http://expressjs.com/>
- [28] Forbes Lindesay. (2010, Julio) Jade - Template Engine. [Online]. <http://jade-lang.com/reference/>
- [27] Ian Hickson. (2009, Abril) Web Storage (Second Edition). [Online]. <https://w3c.github.io/webstorage/#dom-localstorage>
- [29] Underscore.js. [Online]. <http://underscorejs.org/>
- [30] Stylus Github Repo. [Online]. <https://github.com/stylus/stylus>
- [31] Git SCM. [Online]. <https://git-scm.com/>
- [32] Git SCM - Branching and Merging. [Online]. <https://git-scm.com/about/branching-and-merging>
- [33] Git SCM - Git Hooks. [Online]. <https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks>
- [34] Vincent Driessen. (2010, Enero) nvie.com. [Online]. <http://nvie.com/posts/a-successful-git-branching-model/>
- [35] Heroku About. [Online]. <https://www.heroku.com/about>
- [36] TUNE-UP Process. [Online]. <http://www.tuneupprocess.com/>
- [38] WebSocket.org. (2013) WebSocket.org. [Online]. <http://www.websocket.org/>
- [37] Patricio Letelier Torres. TUNE-UP Software Process - Metodología y herramienta de apoyo para la gestión ágil de proyectos de desarrollo y mantenimiento de software. [Online]. [http://lbd.udc.es/jornadas2011/actas/JISBD/JISBD/S3/Tools/07\\_Letelier\\_TUNEUPDemoJISBD2011.pdf](http://lbd.udc.es/jornadas2011/actas/JISBD/JISBD/S3/Tools/07_Letelier_TUNEUPDemoJISBD2011.pdf)