UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Interfaz de desarrollo para clases introductorias a programación

Proyecto Técnico

Eduardo David Villacis Calderon

Ingeniería en Sistemas

Trabajo de titulación presentado como requisito
para la obtención del título de
Ingeniero en Sistemas

Quito, 16 de mayo de 2016

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO DE CIENCIAS E INGENIERÍAS

HOJA DE CALIFICACIÓN
DE TRABAJO DE TITULACIÓN

Interfaz de desarrollo para clases introductorias a programación

Eduardo David Villacis Calderon

Calificación:

Nombre del profesor, Título académico Fernando Sánchez , Ph.D

Firma del profesor _____

Quito, 16 de mayo de 2016

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante: _____

Nombres y apellidos: Eduardo David Villacis Calderon

Código: 00100233

Cédula de Identidad: 1712784683

Lugar y fecha: Quito, mayo de 2016

RESUMEN

Los ambientes integrados de desarrollo, comúnmente conocidos como IDEs, son utilizados por los estudiantes de la Universidad San Francisco de Quito cuando se encuentran en el proceso inicial de aprender código y programar. Estos ambientes de desarrollo tienden a tener un alto nivel de complejidad que en la mayoría de los casos contribuyen negativamente con el proceso de aprendizaje. Nuestro proyecto tiene como principal objetivo desarrollar una simple, pero funcional, interfaz de programación para que los estudiantes puedan mejorar su proceso de aprendizaje y motivación.
La interfaz mencionada ha sido desarrollada como una aplicación web, haciendo uso de tecnologías pioneras y conocidas en el lado del servidor y el lado del cliente. Esta solución es soportada por una arquitectura diseñada a partir de características de ambientes utilizados previamente por los estudiantes y un sistema de contenedores que vuelven a la solución eficaz y segura.

Palabras clave: interfaz, programación, angular, mongodb, usfq, proceso de aprendizaje, motivación, angular material, docker, javascript.

ABSTRACT

Integrated development environments, commonly known as IDEs, are used by students when they are in the initial process of learning to code and program. Such IDEs can have a high level of complexity that in most cases turn out to be a negative contribution to the learning process. This research project has as a main goal to develop a simple, but fully functional, user interface by which students can improve their learning process and motivation.

The mentioned interface has been created as a web application, gathering pioneer and trending technologies used in front-end and back-end of applications. This solution is constructed over architecture designed from factor of previous environments used by students and a container system that makes this solution efficient and secure.

Key words: interface, programming, angular, mongodb, usfq, learning process, motivation, angular material, docker, javascript.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Dentro de la Universidad San Francisco de Quito se dictan clases introductorias de programación, que en muchos casos son la primera vez que los estudiantes programan. Esto vuelve a las clases de programación un gran desafío para la enseñanza, los estudiantes no solo deben aprender los conceptos de programación, sino también aprender el uso de herramientas para editar, compilar y correr el código. Es en este último desafío es el que se concentra este proyecto.

Con la finalidad de ofrecer a los estudiantes de la Universidad San Francisco de Quito una herramienta que beneficie su proceso de aprendizaje introductorio a la programación, se ha realizado una interfaz de programación. Se ha tomado en cuenta las ventajas y desventajas de las interfaces que actualmente los estudiantes utilizan, con el fin de diseñar una herramienta simple y útil. Una herramienta diseñada a contribuir con el aprendizaje del estudiante y que permita un acceso a programar con cero tiempo de instalación, pero con una capacidad de producir cualquier proceso.

El proyecto debe presentar ciertas características de la aplicación, tales como: disponibilidad, consistencia y compatibilidad. El diseño del proyecto ha empleado técnicas y mejores prácticas utilizadas en la industria. Como también ha sido muy ambicioso en la utilización de tecnologías, para que se presente una solución que podrá ser mantenida en los futuros años posteriores a la implementación. El alcance del trabajo es tener una versión beta disponible para introducción progresiva en las aulas.

ANTECEDENTES

Motivación

La universidad San Francisco de Quito cuenta con alrededor de 11 clases introductorias a programación cada semestre, con un promedio poseen una cantidad de 20 estudiantes inscritos por clase. Por las diferentes carreras de ingeniería de los estudiantes, las clases se dictan en C y C++, lenguajes considerados idóneos para aprender a programar, pero a la vez son “lenguajes profesionales que poseen largas y complejas sintaxis” (Esteves, 2011)

Pero dado que en muchos casos es la primera vez que el estudiante; escribe código, piensa lógicamente orientado al código y entiende cómo la máquina es capaz de interpretar, se vuelve un reto aprender sin tener unos pequeños dolores de cabeza. Esto se ha experimentado como tutor de un departamento de apoyo para estudiantes (Learning Center), como también en el día día donde muchos de mis compañeros me realizan consultas acerca de sus tareas. Autores como Jenkins han estudiado sobre si existe alguna relación acerca de tener una aptitud específica para poder programar, no encontrando una relación predominante. Se ha descartado incluso que existe relación entre tener habilidad para resolver problemas lógicos matemáticos y programar (Jenkins, 2002). Tampoco se ha demostrado que existe algún tipo de relación demográfica sino explica Jenkins que el problema es está más orientado hacia el proceso cognitivo de aprendizaje.

Pero si entonces la capacidad de aprender a programar no tiene relación con una aptitud en concreto ¿Cómo podríamos ayudar al estudiante en su proceso de aprendizaje? Esta última pregunta es la que muchos profesores se hacen, y sobre esto Jenkins se refiere con dos factores importantes en el aprendizaje de programación: estilo de aprendizaje y motivación.

El profesor en el aula de clase puede hacer mucho para motivar al estudiante e incluso la sociedad hace los suyo. El estudiante puede tener motivación propia (intrínseca), motivación por generar lucro (extrínseca) o motivación por su sociedad (social) (Jenkins, 2002). Pero el estilo de aprendizaje de cada estudiante es muy importante tomar en cuenta, y es justo sobre este último punto al que se intentará ayudar mediante este trabajo.

Realidad

Los estudiantes reciben clases en los laboratorios de la Universidad San Francisco de Quito, estos cuentan con acceso a internet y algunos editores disponibles. En su mayoría las clases son dictadas sobre ordenadores con sistema operativo Windows y en estos existe una gran variedad de editores tanto simples, como complejos. Editores simples como el Bloc de Notas y ambientes completos de desarrollo como DevC++ o Visual Studio. En ciertos casos también los estudiantes traen sus propios ordenadores en donde deben tener sus editores instalados y en ciertos casos extrapolar la idea que el profesor dicta para su editor.

Esto último es uno de los primeros problemas que tiene el estudiante, muchas veces le toma mucho tiempo instalar un ambiente de desarrollo en particular. En ciertos casos no estará utilizando el mismo editor que su profesor y en otros, estará utilizando incluso un sistema operativo totalmente diferente. Esto vuelve un poco difícil el proceso de aprendizaje del estudiante e impacta con la motivación (escaza en muchos casos).

Por otra parte, algunos estudiantes son un poco más ingeniosos y hábiles, logran replicar el ambiente de desarrollo en el que el profesor enseña. Esto le simplifica mucho su aprendizaje, pero generan una abstracción en cierto caso falsa, donde pierden el conocimiento sobre qué en realidad sucede por detrás de todo su entorno de desarrollo. Ideas tales cómo un lenguaje de alto nivel se transforma en binario y se ejecuta en el

computador. Esto forma una idea errónea sobre cómo su programa funciona, y tienden a pensar que fuera de su computador no existe forma de que corra.

Sobre esta realidad es la que se ha tratado concentrar la solución implementada en nuestro trabajo. Tratando de tomar en cuenta las ventajas y desventajas de las herramientas que los estudiantes están utilizando actualmente. También se ha tomado en cuenta los retos que tienen al momento de programar y compilar sus códigos. Por último, se toma en cuenta las nuevas tendencias tecnológicas, para brindar una solución de calidad y tener la capacidad de llevar la solución a cualquier tipo de ordenador que el estudiante quiera usar.

Mediante lo expuesto anteriormente, se puede concluir que el estudiante necesita ayuda y para esto proponemos una interfaz de desarrollo, que tiene como fin contribuir al proceso de aprendizaje del estudiante. Existen muchos otros factores que afectan al estudiante, pero se ha considerado como un proyecto interesante el realizar una solución, ya que existe una población importante que se beneficiada.

Ambientes de desarrollo

Los estudiantes utilizan dos sistemas operativos principalmente: Windows y OSX. Los dos sistemas operativos presentan varias opciones de ambientes de desarrollo, de los cuales sobresalen: Visual Studio y XCode en cada sistema respectivamente. Se destacan estos dos ambientes de desarrollo por ser los preferidos de los estudiantes, debido a su facilidad de instalación y porque son sugeridos por el fabricante del sistema operativo. Sin embargo, estos ambientes de desarrollo son bastante complejos y fueron diseñados para un uso profesional.

Es entendible, que el estudiante se encuentre con un reto intentar programar por primera vez y que al utilizar por primera vez un ambiente de desarrollo como los mencionados, se desmotive. En el caso preciso del editor Visual Studio, el ambiente agrega librerías propias, complicando la comprensión del código y su portabilidad. Es decir que, si uno empieza un proyecto utilizando este ambiente de desarrollo, tendrá que cambiar el código para poderlo correr en otro editor o incluso en otro sistema operativo. Por esta misma razón el estudiante encuentra dificultad si su profesor trabaja en otro ambiente de desarrollo o si exporta un trabajo realizado en clase, como viceversa cuando el estudiante entregue su tarea o proyecto al profesor.

Xcode por su lado, no presenta un problema tan crítico como el de Visual Studio (dependencia de bibliotecas), pero al estar diseñado para uso profesional si vuelve en ciertos casos compleja la utilización. En particular por experiencia propia y con mis otros compañeros hemos encontrado problemas al importar clases y al exportar proyectos. Xcode posee una estructura que acompaña con metadatos, para poder comprender cuales son los archivos del proyecto. Y si una persona pretende simplemente copiar y pegar un archivo dentro del proyecto o simplemente no lo crea de la forma correcta el proyecto no podrá ser compilado. Esto se debe a que los metadatos no son editados correctamente y al compilar el programa, el editor no puede compilar los archivos necesarios para correr el programa.

Prácticas comunes y el mundo de la programación

En los anteriores casos o en cualquier otro ambiente de desarrollo que ha sido diseñado para uso profesional, existirán muchas herramientas útiles para un desarrollo rápido de código, un diseño profesional de un producto, incluso para poder realizar migraciones a los servidores. Estos son características que no necesita un estudiante que empieza a

programar. Además, que el ambiente de desarrollo se encuentra diseñado para que el usuario no se preocupe por correr sus propios comandos para compilar y/o correr el proyecto, esta abstracción es útil en ciertas ocasiones en el mundo profesional, pero elimina la oportunidad al usuario principiante de entender el proceso de compilación de un lenguaje de alto nivel.

También hemos evidenciado ciertas prácticas de los estudiantes y hasta cierto tipo excusas en sus primeros pasos como programadores. Una de las excusas más comunes es explicar al profesor que su tarea solo sirve en su computador o mejor aún que no ha podido seguir el hilo de la clase porque todavía no logra tener todo instalado y funcionando. En muchos casos los estudiantes recurren al internet para buscar soluciones a sus problemas, y no entienden porque el código de internet no les funciona.

Si bien la práctica de copiar y pegar no es aceptada en el mundo académico, es muy interesante el aprender corriendo un ejemplo de internet y tratar de modificarlo para cumplir los objetivos. Y es esto último que ha llevado a la gran comunidad de programadores a desarrollar soluciones robustas a partir de proyectos de terceros y que generan ese ambiente colaborativo enorme de la Internet. El estudiante en sus primeros pasos no se da cuenta, que realizar ese tipo de consultas lo integra en una comunidad de programadores, dónde sus primeros pasos y problemas ya han sido resueltos. Y le da una oportunidad de aprender y llenar cualquier incertidumbre acerca de si sus proyectos en programación son posibles, que en definitiva impacta en la motivación anteriormente descrita.

En los últimos años los ordenadores no convencionales, tales como; celulares, tablets, chromebooks, han venido en auge. Cada día los estudiantes hacen mayor uso de estos por

su versatilidad. La arquitectura de estos dispositivos nos les permite compilar código para ordenadores normales (x86 y x64), aunque sí les permite escribir código. Estos dispositivos no poseen la capacidad de compilar código que no pertenezca a su arquitectura, actualmente brindan la capacidad de editar textos, lo que se puede orientar a editar código. Incluso por temas de seguridad, estos dispositivos se encuentran restringidos y deben someterse a un proceso en contra de la garantía del fabricante para poder utilizar permisos especiales, como el de ejecución. Por lo que claramente podemos establecer que existe un vacío para aquellos que quiere utilizar su tablet, celular, Chromebook, entre muchos otros.

Por último, los estudiantes se ven limitados a utilizar solo los computadores que están listos con todo lo necesario a instalar. Esto puede limitar en cierto punto su creatividad y seguramente su motivación a desarrollar código. Esto también afecta la universidad y a los profesores, que se ven limitados a utilizar solo los laboratorios que tienen disponible lo necesario. Con esto se ve una necesidad no solo de portabilidad del código, sino también una necesidad de un editor que se pueda utilizar en cualquier lado. Que con solo tener como requerimiento una conexión a internet, el estudiante tenga la capacidad de empezar a programar.

DISEÑO

El propósito de este proyecto es contribuir con el proceso de aprendizaje mediante: la implementación de una herramienta que permita al estudiante empezar a programar, evitando que el ambiente de desarrollo se vuelva un obstáculo para el aprendizaje. Esta herramienta tendrá que tomar en cuenta los retos mencionados en los antecedentes, para poder brindar una solución funcional. Para cual hemos desarrollado nuestro proyecto en cuatro pilares, los cuales se puede apreciar la **Figura 1**. Es importante que en el diseño se tome en cuenta lo que los usuarios se encuentran utilizando, para que se pueda brindar una solución que tome las ventajas sobre aquellos ambientes que el usuario ya se ha familiarizado.

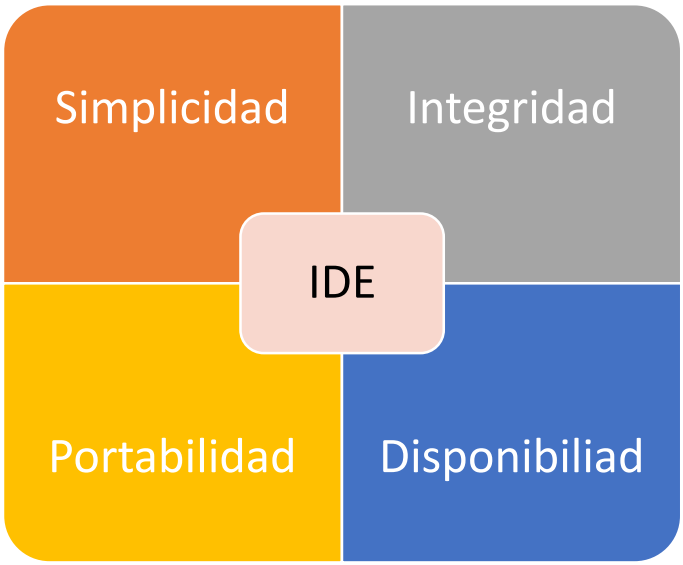


Figura 1. Cuatro pilares del diseño

Estos pilares han sido analizados en detenimiento, obteniéndolo de las herramientas utilizadas actualmente. No solo tratando de replicar aquellas características que vuelven atractivas a las soluciones existentes, sino que también intentando de brindar la mayor seguridad a los estudiantes. En las secciones a continuación se expondrá las diferentes

consideraciones en el diseño y como estas afectan a los cuatro pilares. Los requerimientos generales a tomar en cuenta en el diseño se encuentran en la **Tabla 1**.

Requerimientos		
Número de usuarios	250 aproximadamente	
Sistemas operativos	Windows, OSX y Linux	
Lenguajes de programación	C & C++	
Tipos de usuarios	Estudiante	Profesor
Duración de credenciales	6 meses	Ilimitadamente

Tabla 1. Requerimientos generales

Arquitectura

De acuerdo a la sección de antecedentes se han levantado diferentes requerimientos que junto a los cuatro pilares formarán nuestra arquitectura.

En los requerimientos, existe la necesidad de que usando incluso ordenadores no convencionales los estudiantes sean capaces de editar, compilar y correr código. Se considera a este requerimiento como crítico, ya que cualquier solución tendrá que considerar un servidor fuera del cliente capaz de compilar y correr código. Esto se debe a que los ordenadores convencionales son capaces de compilar y correr código, pero dentro de su arquitectura. Por lo que este requerimiento obliga crear una solución, donde la nube se encuentre encargada de correr y compilar el código. Esto permite compilar y correr el código sin importar qué ordenador se utilice.

Segundo, existe la necesidad que se brinde portabilidad al usuario. Este requerimiento se lo puede cubrir mediante el diseño de una plataforma de archivos en la nube capaz de: simplificar los proyectos de código, administrar los archivos y mantener integridad de la información. Por lo que la propuesta de arquitectura en la nube requiere una implementación con un manejo de archivos similar al mencionado.

Por último, se necesita una discusión de la arquitectura de lado del cliente. Se requiere un análisis sobre si es mejor implementar una solución nativa específica para cada arquitectura del cliente o tal vez es conveniente una implementación web genérica con capacidad de adaptarse a cada cliente. Esta discusión se abordará en la sección “Aplicación Nativa vs Aplicación Web”.

Tecnología en la Nube.

Como se mencionó se propone una arquitectura en la nube que será la base a nuestro servidor. Esta proveerá cobertura a los cuatros pilares de diseño mencionados y además permita un crecimiento de la solución de ser necesario. Hasta el momento se ha mencionado que la arquitectura de la nube debe constar de las siguientes capacidades:

Figura 2. Capacidades necesarias de la tecnología en la nube

Estos tres componentes cubren necesidades críticas para que el editor brinde una solución aceptable y funcional, pero aún debemos cubrir dos requerimientos críticos que no se han

discutido: usuarios y sesiones. Es necesario que la nube sea capaz de manejar credenciales de usuarios y sesiones para poder brindar la información indicada al cliente.

Con el fin de brindarle al estudiante un ambiente completo de desarrollo, además de poseer un editor, hemos visto conveniente facilitar una terminal. Con la que el estudiante tendrá la capacidad de compilar y ejecutar su código sin que el editor lo realice.

Las características de cada uno de los componentes de la arquitectura de la nube se pueden apreciar en el siguiente diagrama:

Figura 3. Arquitectura en la nube a implementar

Aplicación Nativa vs Aplicación Web.

Una arquitectura de nube permite acceder mediante una interfaz desarrollada en una aplicación nativa desarrollada para cada dispositivo o una aplicación web independiente del dispositivo en el que corre. Tomando en cuenta los requerimientos del proyecto, en esta sección analizamos las ventajas de cada tipo de aplicación en el desarrollo del proyecto.

Para esta sección se ha utilizado una investigación realizada en Finlandia por Tommi Mikkonen y cierta experiencia laboral del autor.

El centro de la discusión se encuentra en aquellos usuarios no convencionales, que acceden mediante dispositivos móviles. Para los cuales tal vez una aplicación nativa sería lo más idóneo, considerando las diferentes capacidades que un dispositivo móvil puede brindar. Pero se descarta la solución nativa para ordenadores convencionales, con el fin de cumplir con los cuatro pilares que se han planteado en este proyecto y desarrollar la solución dentro del tiempo establecido. Se pretende mejorar los ambientes de desarrollo analizados en los antecedentes, que necesitan instalar todo un ambiente (en muchos casos innecesarios) para poder trabajar y al final del día cuando se quiere cambiar de ambiente de desarrollo, la migración puede ser un poco tediosa para un usuario que recién está dando sus primeros pasos en programación.

Como menciona Mikkonen, mientras los usuarios de ordenadores de escritorios acceden al contenido mediante exploradores, mientras los usuarios móviles acceden a contenido del internet en su mayoría mediante aplicaciones nativas. En los últimos cinco años, han aparecido cada vez más aplicaciones hibridas, que compilan en código nativo implementaciones web. Estas últimas aplicaciones, se encuentran tan bien diseñadas que en muchos casos es difícil discernir entre una aplicación nativa o una híbrida. El usar una aplicación nativa conlleva necesitar una aplicación para cada tipo de dispositivo con lo cual tendríamos algo similar al siguiente gráfico.

Figura 4. Plataformas nativas necesarias a implementar

Como podemos evidenciar se necesitaría al menos la construcción de una interface para comunicación con la nube y de al menos 5 aplicaciones construidas en diferentes plataformas. Tal vez se podría crear un proyecto con gran complejidad capaz de desarrollar las aplicaciones nativas necesarias. Pero como podremos ver en el siguiente gráfico las ventajas de desarrollar una aplicación nativa no impactan directamente en las necesidades de la solución.

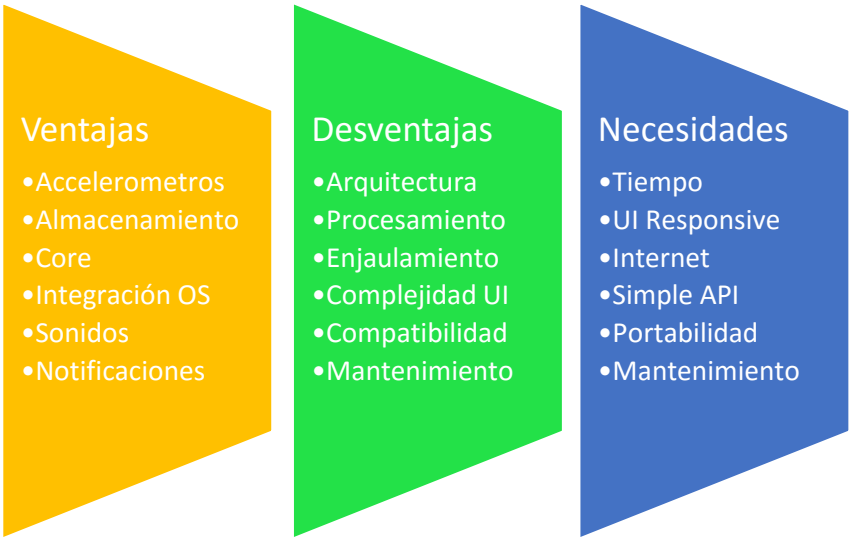


Figura 5. Ventajas, desventajas y necesidades en el desarrollo en plataformas nativas

Por otro lado, la implementación de una aplicación web como solución, tiene dos grandes ventajas: tiempo y compatibilidad. Se puede desarrollar una aplicación web, que será utilizada en común por los diferentes sistemas operativos (listados anteriormente), sacrificando capacidades nativas que será accedido por los diferentes exploradores.

Por esto, se propone un diseño Aplicación de Página Única (conocida por sus siglas en inglés como SPA), para promover el uso de las capacidades del cliente, concentrando la mayor carga de interacción en el cliente y dejar las tareas de; compilación, ejecución y almacenamiento en el servidor. La experiencia del usuario de esta manera será parecida a la de una aplicación nativa, cargando asincrónicamente las vistas necesarias (AJAX) y evitando navegación mediante requerimientos HTTP (GET o POST). Se genera de esta manera, una sensación de rapidez y eficiencia, que al mismo tiempo se ve reflejada por la disminución de tráfico continuo que necesita la aplicación. Para poder entender de manera gráfica, se presenta el siguiente diagrama acerca del diseño.

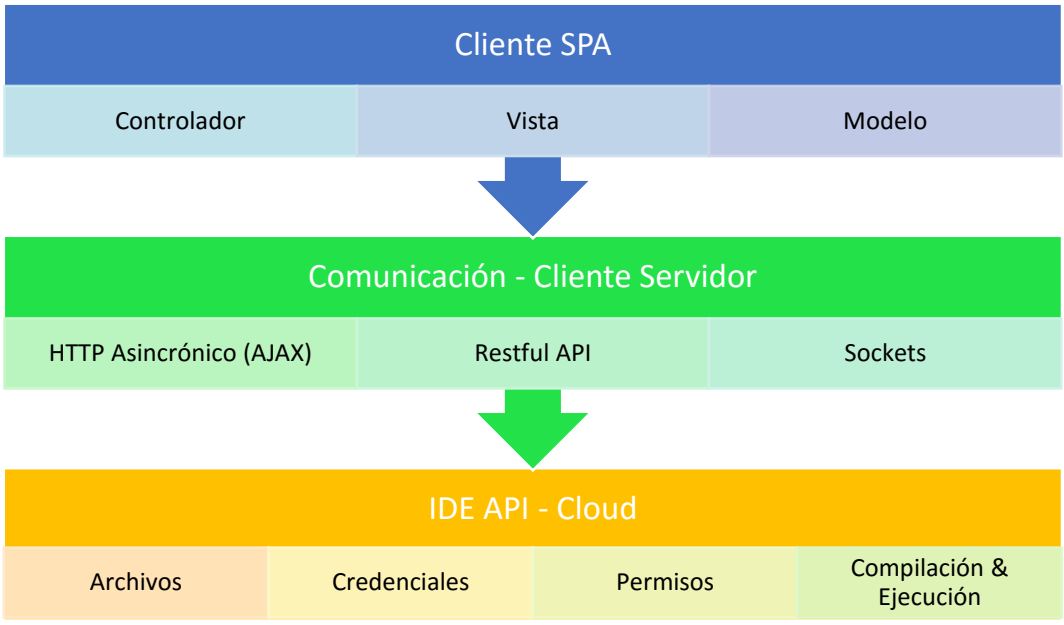


Figura 6. Diseño de solución en plataforma web.

Este diseño será el adoptado para realizar la solución y su implementación se detalla en la siguiente sección. Cabe resaltar que se utilizará tecnología pionera dentro de las aplicaciones web y que la literatura al respecto por el momento es escaza. Pero con el fin de proveer una solución de calidad se ha seguido guías de mejores prácticas. Que impactará no solo en la calidad de la solución, sino también en su futuro mantenimiento.

IMPLEMENTACIÓN

A partir del diseño hemos seleccionado las tecnologías existentes más idóneas para proceder a la implementación. Las tecnologías escogidas para la implementación impactan directamente futuras implementaciones y el mantenimiento de la solución. Por lo que se busca implementar tecnologías que sean conocidas y que la industria recomienda utilizar.

De esta forma al momento de seleccionar los lenguajes de programación para la implementación existen dos diferencias que hacer, lenguajes de lado del servidor y lenguajes del lado del cliente. Del lado del servidor existen varias opciones para escoger con capacidad web, mientras del lado del cliente a pesar de existir lenguajes para implementación de interfaces o similares, todos en última instancia se transforman en HTML y JavaScript. Esta última restricción se debe a que los exploradores solo pueden interpretar esos lenguajes.

Por otro lado, el potencial de JavaScript desde hace alrededor de 6 años no se encuentra dirigido únicamente al lado de desarrollo del cliente. En mayo de 2009, se lanzó el proyecto Node.JS que es un framework, que posee la capacidad de crear servicios web y que ha tenido mucho éxito por: rápido motor(V8), habilidad para satisfacer requerimientos concurrentes, se encuentra basado en un sistema de eventos, permite compartir código de

lado del cliente en el lado del servidor, entre muchos otras características.Node.JS ha sido un proyecto ambicioso, que ha llamado mucho la atención de los desarrolladores, creando de esta forma un repositorio de paquetes públicos muy grande y conocido llamado NPM.

Esto hace de JavaScript una plataforma atractiva para implementar en este proyecto, con más de 250’000 proyectos disponibles para complementar el desarrollo. Además de tener la capacidad de desarrollar ambos extremos de una aplicación con el mismo lenguaje, permite compartir bibliotecas (Mikowski,2013).

En lo referente a manejo de información, con la aparición de bases de datos no convencionales como MongoDB, se puede guardar los objetos JSON potencializando más la tecnología del lado del servidor.

En cuestiones de infraestructura también es atractivo, ya que el consumo de memoria y la optimización de procesamiento son elementos fundamentales de la plataforma. El manejo de concurrencias en el lado del cliente ha demostrado grandes ventajas, y se espera que la misma característica ayude al performance de la solución. La naturalea liviana en recursos (memoria y disco) de estas tecnologías, ha sido atractiva para muchas empresas (Mikowski,2013). Tales como Netflix que la utilizan en áreas importantes como las pruebas automatizadas de sus productos (Lui, 2014).

Por todo lo anteriormente expuesto, hemos decidido desarrollar la solución en Node.JS y de esta manera implementar JavaScript del lado del cliente y del servidor. Queremos evaluar una vez finalizada la implementación de la solución, qué ventajas y recursos de este ecosistema han colaborado con los pilares de diseño.

Servidor

Como se expuso en la sección previa, el servidor se ha implementado en JavaScript mediante el ecosistema Node.JS. El ecosistema tendrá que se ser capaz de cubrir los

requerimientos del diseño, implementando a la vez tecnologías pioneras y con gran capacidad de mantenimiento. Del lado del servidor hemos escogido implementar las siguientes tecnologías, que se pueden apreciar en la **Figura 7.**

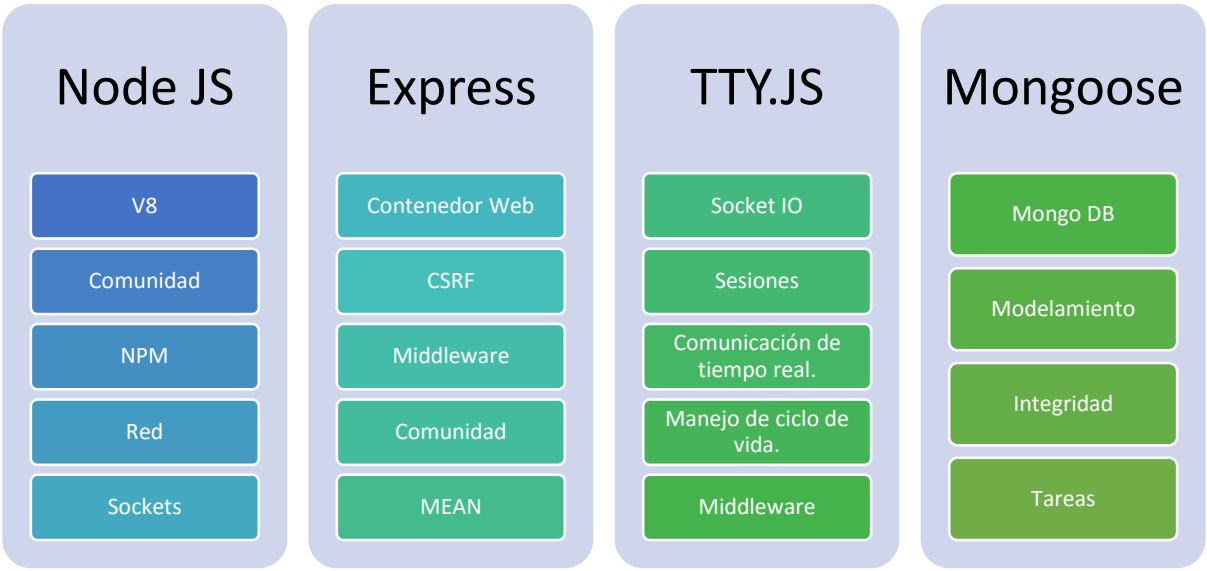


Figura 7. Implementación general del servidor

Estructura.

Express JS.

Framework más utilizado en el administrador de paquetes NPM, para desarrollo de aplicaciones web y aplicaciones móviles. Su gran uso se debe a la versatilidad que propone a los desarrolladores dentro del ecosistema de Node.Js. Esta versatilidad se promueve mediante el principio de configuración sobre convención (Mardan, 2014). Estas implementaciones complementarias al Framework toman el nombre de Middleware. En nuestro proyecto existen dos grandes componentes de Middleware: Restful API y TTY.JS. Un componente adicional que inserta una capada de seguridad mediante certificados CSRF. Para evitar ataques de sitios no autorizados (cross-site injection).

Por otro lado, mediante Express JS se puede configurar el puerto y los certificados SSL, de ser necesario. Por la necesidad de tener un certificado firmado, se ha dejado como trabajo futuro. En cuestiones de puerto, la aplicación se encuentra instalada para trabajar dentro del puerto 8080. Será mediante la infraestructura que mapearemos este puerto con el respectivo para HTTP (80).

Restful API.

Esta sección de la implementación será la encargada de generar una interfaz de acceso a las funcionalidades implementadas en la nube. Se escoge esta forma de acceder a las funcionalidades porque permite un desarrollo independiente en la nube con respecto a la terminal que utilice la biblioteca de funcionalidades (API). La biblioteca pretende satisfacer las funcionalidades en la nube orientadas a la administración de: usuarios, archivos y sesiones.

Las llamadas a la biblioteca deberán ser realizadas en el formato debido para poder ser respondidas. Todas las respuestas de la API han sido diseñadas para notificar el éxito o fracaso de cada transacción. Como medida de seguridad toda consulta debe contar con un certificado CSRF, esta implementación será descrita en una sección siguiente. Las consultas serán respondida de forma asincrónica, asegurando que al recibir consultas concurrente, el sistema no baje su factor de disponibilidad.

Mongoose – MongoDB.

Dentro de la implementación del servidor se ha descrito el manejo de archivos, pero no se ha mencionado de qué forma funcionará la base de datos, necesaria para el manejo de los usuarios como se mencionó en el diseño. Además, al estar utilizando JavaScript, hemos

encontrado atractivo e idóneo manejar un motor de base de datos no convencional que almacena su información en objetos de JavaScript (JSON).

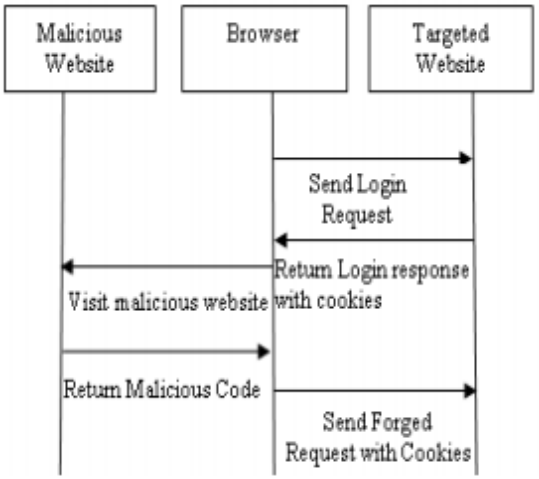
El uso de bases de datos como MongoDB, que se estructuran en documentos, tiene ventajas como: rapidez, la libertad de esquema, escalabilidad horizontal o sentencias profundas de búsqueda. Sin embargo, también hay desventajas especialmente la de interés para este trabajo: la informalidad del esquema. Esto último se refiere a que, sin un uso apropiado de la base de datos, se puede caer en duplicación de información y en documentos con gran variedad de forma en una misma colección.

Mediante la implementación de Mongoose, se pretende evitar caer en las desventajas de MongoDB al implementar una capa lógica que interfiere antes de proceder a guardar la información. No solo evitamos así tener documentos con diferente estructura, sino que también mediante la capa lógica podemos verificar que cierta información sea única como: usuarios y directorios del sistema. En el caso de que se intente guardar por ejemplo el mismo nombre de usuario, se notifica que existe un conflicto con un documento que se encuentra guardado. Funcionalidades atractivas como llenar documentos con valores por defecto y tareas necesarias al realizar transacciones, también se incluyen en la implementación de esta capa lógica.

Seguridad.

La seguridad es un factor muy importante de nuestro trabajo y se encuentra en nuestro diseño como el pilar de integridad. No existe discusión acerca del rol importante que juega la seguridad en cualquier solución informática hoy en día y nuestro trabajo no es la excepción. Por tratarse de información de estudiantes, se ha realizado ciertas implementaciones y se explora que un futuro se realice muchas más.

Del lado del servidor hemos encontrado dos implementaciones en particular, fuera de aquellas que son aconsejadas en la industria y por los manuales de buenas prácticas en soluciones web. Estas implementaciones son: certificados CSRF y contenedores Docker. Si bien los certificados CSRF son una solución de lado del servidor como del cliente, en la siguiente sección se describen los detalles. Los contenedores Docker son descritos en la sección de seguridad de infraestructura.



Certificados CSRF.

Figura 8. Ataque por inyección cross-site

Por sus siglas en inglés Cross-Site Request Forgery, son certificados que se implementan desde el servidor una interfaz web autorizada. Se previene de esta forma ataques desde sitios maliciosos que envían requerimientos a sitios honestos (Khurana, 2014). A continuación, se puede ver un gráfico de como un ataque similar al descrito funciona.

Todo requerimiento desde el sitio autorizado envía un certificado que es creado desde el servidor y al ser recibido dentro del requerimiento es validado. En nuestra solución en particular, hemos tenido que programar una capa lógica dentro de Express, para que los certificados sean asignados correctamente a cada sesión.

Comunicación de tiempo real.

La comunicación de tiempo real, es muy importante para realizar las operaciones de la terminal virtual, que se describirá en la siguiente sección. Esta comunicación es realizada mediante sockets, que poseen eventos. Los eventos son programados tanto del lado del cliente como del servidor, además existen eventos disponibles que manejan el ciclo de vida del socket.

En nuestra aplicación, hemos analizado los diferentes eventos para entender cómo se puede modificar la comunicación a tiempo real con el fin de poder ingresar credenciales de los usuarios en la terminal. Mediante la implementación de código dentro de los eventos del ciclo de vida y un evento que busca ciertos textos escritos desde el servidor al cliente, hemos logrado insertar de forma automatizada las credenciales de los usuarios. De esta manera, los usuarios al abrir la terminal siempre se encontrarán dentro de su cuenta de sistema.

Terminal – TTY.JS.

Como se ha mencionado nuestra solución presenta una terminal virtual, que permite al usuario utilizarla con capacidades indiscernibles con una terminal nativa. Esta característica no ha sido desarrollada en el trabajo, ya que ha sido incorporada y modificada de un proyecto original desarrollado por Christopher Jeffrey.

El proyecto presenta funcionalidades y configuraciones por defecto, las cuales han sido modificadas en la implementación de nuestra solución. Las modificaciones más importantes del lado del servidor son:

- El puerto en el que aloja la terminal originalmente es 8000
- Eliminación de middleware para sistema de credenciales.

- Reemplazo de directorio estático.

Cliente

La implementación del lado del cliente consta de varios componentes, los mismos que han sido implementados para satisfacer criterios del diseño. Además, se ha tomado en cuenta el uso de buenas prácticas, como también hemos usado un buen criterio al escoger bibliotecas recomendadas y usadas en la industria. Los diferentes componentes y sus respectivas implementaciones, se describirán a continuación.

Interfaz de usuario.

Aplicación de página única SPA – Angular JS.

Según el diseño, hemos establecido que nuestra solución sea implementada mediante un SPA, capaz de satisfacer los pilares de diseño mencionados en este trabajo. Para lo cual existen varias opciones de bibliotecas que nos permitirán implementar nuestros requerimientos, bibliotecas como: Knockout JS, Polymer, Angular entre otras. Escoger la biblioteca adecuada en este punto es muy importante, porque será la biblioteca base en la que se sostendrán los siguientes componentes y la que tendrá la mayor responsabilidad con respecto al performance de la solución en el lado del cliente. Por lo que se ha escogido Angular ya que cumple con la gran calidad necesaria y es una herramienta muy usada en la industria, como se puede apreciar en la **Figura 9** de Google Trends.

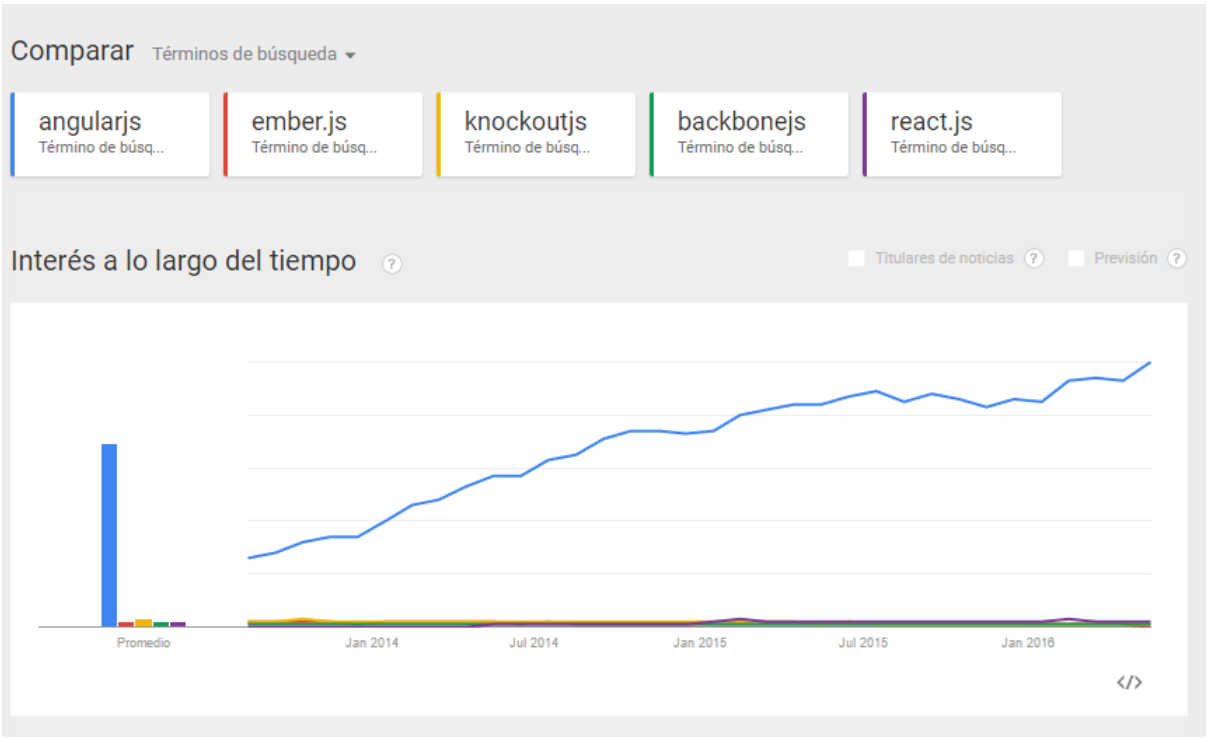


Figura 9. Comparación en Google Trends de plataformas web.

En angular hemos implementado cinco controladores principales, los cuales son: editor, inicio, localización, inicio de sesión y registro. Cada uno de estos controladores pretenden satisfacer las necesidades de diseño e intentar separar las funcionalidades brindadas. El controlador que nos gustaría describir un poco más en profundidad es que corresponde al editor, ya que es el responsable de brindar el centro de nuestra solución.

El editor cuenta con implementaciones para comunicarse con el servidor y obtener la lista de archivos que se encuentran disponibles en el proyecto. Esta encargado de la implementación necesaria para cargar el contenido de los archivos, actualizarlos y de ser necesario borrarlos. Además, posee una característica que permite notificar al usuario, cuando se pretende abrir un archivo sin guardar los cambio de otro en el que se encontraba

trabajando. Además, para la sección del editor se ha implementado una directiva que será descrita en la sección de Jstree.

La localización es otra componente importante de nuestro trabajo, dada la naturaleza del mundo actual. El controlador se encuentra implementado fuera del módulo principal que maneja las rutas, pero es capaz de cambiar los textos de la aplicación mediante el uso de archivos de idiomas. Por el momento se ha implementado la solución en español e inglés, pero agregar otro idioma no sería complejo por la forma en que se encuentra organizada la aplicación de Angular.

Por último, respecto a angular, es importante mencionar la biblioteca utilizada para la interfaz. Hemos visto conveniente implementar Material Design de Google, para poseer una interfaz robusta capaz de adaptarse a diferentes tipos de resolución y con elementos atractivos para el usuario. Nos referimos a esta sección de la implementación en Angular, ya que se ha utilizado Angular Material Design. Esta biblioteca aprovecha las diferentes capacidades de angular e implementa directivas simples de utilizar para el usuario. En la siguiente sección se explora un poco más acerca de esta porción de la implementación.

Diseño orientado a Material – Angular Material.

Como hemos mencionado en la anterior sección para mejorar la experiencia de usuario y su interacción se ha implementado Angular Material. Biblioteca que incorpora los principios de Material Design ampliamente adoptados por la infraestructura Android y que tienen como objetivo una experiencia del usuario fácil y atractiva.

En nuestro trabajo hemos implementado varios elementos de la biblioteca como: botones, paneles, menús, casilleros de texto, entre otros. Se ha implementado tomando en cuenta los diferentes dispositivos que utilizarán la solución, por lo que se ha tomado especial cuidado en responder al tamaño de la resolución del cliente. Esto último se lo ha conseguido mediante el atributo layout, que se puede apreciar con respecto al ancho de la pantalla del cliente en la **Tabla 2.**

API	Activates when device
layout	Sets default layout direction unless overridden by another breakpoint.
layout-xs	width < **600**px
layout-gt-xs	width >= **600**px
layout-sm	**600**px <= width < **960**px
layout-gt-sm	width >= **960**px
layout-md	**960**px <= width < **1280**px
layout-gt-md	width >= **1280**px
layout-lg	**1280**px <= width < **1920**px
layout-gt-lg	width >= **1920**px
layout-xl	width >= **1920**px

Tabla 2. Descripción de adaptabilidad en Angular Material según ancho del dispositivo.

Editor de código – ACE editor.

Si bien Angular Material design aporta de forma importante en la experiencia del usuario, todavía existe la necesidad complementar esa experiencia en el editor. Se necesitaba brindar similares capacidades a las que posee una ventana de edición en la solución web que se implementó. Capacidades como reconocer el código y describirlo mediante colores, mostrarlo correctamente mediante el uso de identación y algún mecanismo de sugerencia de sintaxis. Por esta razón hemos implementado la biblioteca ACE editor.

Mediante ACE editor, se ha logrado incorporar funcionalidades básicas de un editor de código dentro de una interfaz web. Similar a muchos editores que se han implementado en línea, nuestra solución cuenta con las necesidades antes mencionadas implementadas únicamente para los lenguajes de programación C y C++. Se ha adoptado el tema por defecto que posee similares características con respecto a los colores de Material Design. Si existe en el futuro la necesidad de que el editor sea usado para otros lenguajes de programación solo se necesitará agregar la biblioteca adecuada de ACE y configurar el editor.

Árbol de archivos – Jstree.

Dentro de la interfaz del editor se necesitar desplegar los diferentes archivos que son parte de cada proyecto, para esto existen varias posibilidades, una de las cuales es la visualización mediante tipo árbol. Ampliamente utilizada para mostrar archivos de un directorio escalonadamente. Esta implementación fue inicialmente descrita en la sección de angular debido que la implementación ha sido realizada mediante una directiva, lo cual permite aprovechar las funcionalidades de angular e integrarlas a las de la biblioteca Jstree.

Se ha implementado una directiva en angular que detecta cuando la lista de archivos ha sido actualizada (creación o eliminación) y refresca el componente Jstree. Esto lo hace mediante un componente programado fuera del módulo principal y que puede ser implementado de forma simple a través de una etiqueta HTML. Haciendo uso del ciclo de vida de angular, la implementación es capaz de percibir cuando la lista de archivos ha sido actualizada, y refresca la lista en formato árbol.

Seguridad.

Certificados CSRF.

Como se mencionó en la sección de seguridad en el lado del servidor los certificados CSRF han sido seleccionado para la implementación de un factor de seguridad. Pero la solución requiere implementar cierta lógica también del lado del cliente. Por estar utilizando angular, se ha implementado la biblioteca correspondiente que se encarga de recibir el certificado del servidor e incluirlo en las transacciones HTTP.

Construcción & Ejecución

En las siguientes secciones describimos las partes más importantes de la construcción y ejecución para el sistema.

NPM & Package.json.

NPM es un administrador de paquetes dentro del entorno Node, que permite desarrollar los proyectos de forma ordenada, con una documentación estandarizada e incluso guardar tareas importantes que todo proyecto tiene como pruebas, construcción y ejecución. En nuestra implementación en específico, se utilizó NPM para el manejo de dependencias y tareas antes mencionadas.

Las diferentes dependencias se encuentran dentro del archivo Package.json. esto permite que la aplicación sea migrada y que sus dependencias se puedan instalar rápidamente. Las tareas de construcción y ejecución, se describen a continuación, la tarea de pruebas es considerada como un trabajo a futuro.

Gulp & Webpack.

Una desventaja al momento de trabajar con JavaScript es la longitud que los archivos suelen tomar, mucho más si la solución se encuentra desarrollada utilizando un framework como Angular. Por esta razón es necesario complementar la solución con una herramienta de construcción automatizada de sistemas como Gulp.

Gulp realiza la construcción de los sistemas mediante un sistema basado en tareas. En la implementación se han descrito tres grupos importantes de tareas, comprendidas en: estilos, angular y dependencias. En las tareas de estilo se compila el lenguaje SASS en CSS para que pueda ser interpretado por cualquier explorador. Las tareas de angular, construyen un archivo reducido y verificado con el contenido de múltiples archivos, permitiendo la programación de los diferentes componentes en archivos por separado que serán construidos con Gulp. Y las tareas de dependencias, se refiere a copiar ciertos componentes específicos que se necesitan en el lado del cliente para que la aplicación funcione, componentes como bilbliotecas específicas de C y C++ del editor ACE o componentes específicos del proyecto TTY.JS

Webpack por su lado, reúne todo lo procesado por Gulp y crea un único archivo capaz de importar todas las dependencias desde un mismo archivo, mejorando el rendimiento de la página.

Forever JS.

Por último, en esta sección de construcción se encuentra implementada una herramienta llamada Forever.JS. Esta herramienta diseñada para manejar procesos en Node y asegurar su disponibilidad, asegurándose que exista siempre corriendo una instancia en el sistema operativo con el programa deseado. En nuestra implementación en particular asegura que

el servidor web se encuentre disponible, incluso cuando por algún error se detenga, siempre guardando un registro de los eventos encontrados.

Infraestructura

Si bien la infraestructura no es el tema central de la solución desarrollada en este trabajo, es importante describir ciertos requerimientos y características que son indispensables para un correcto funcionamiento de la solución.

Requerimientos.

Los requerimientos a nivel de hardware dependen más de la demanda que tendrá la aplicación, aunque en términos generales Node.JS se puede ejecutar en: Windows, OSX y Linux. Requiriendo una memoria mínima de 256 MB y sin necesitar de procesadores con varios núcleos. En lo que refiere a nuestra solución en particular los requerimientos pueden ser apreciados en la **Figura 10.**

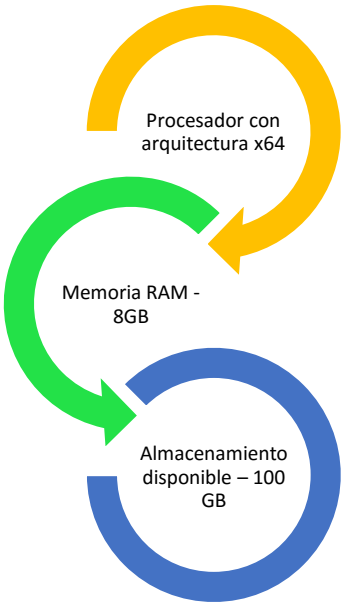


Figura 10. Requisitos recomendados en infraestructura de hardware

Con respecto a las dependencias de software, hemos considerado conveniente implementar todo sobre una infraestructura Linux, pero no es un limitante porque la solución realmente va instalada dentro de un contenedor de Docker. Por esta razón el requerimiento principal dentro de la infraestructura es tener instalado Docker. Una vez instalado Docker, los requerimientos de software principales se pueden encontrar en el **Anexo A** con la imagen de contenedor. Los requerimientos más importantes dentro de la imagen del contenedor pueden ser apreciados en la **Figura 11.**

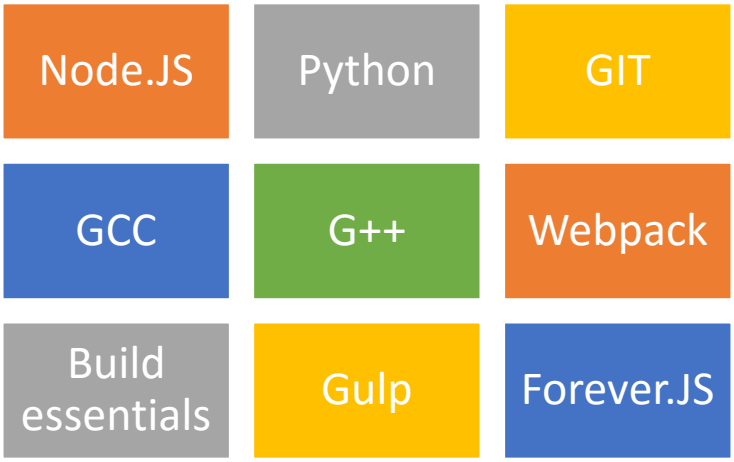


Figura 11. Dependencias necesarias del contendedor Docker.

Contenedores.

Consideramos que el sistema de contenedores implementado (Docker) es parte imporante en nuestra solución. Si bien es una tecnología que recién está tomando fuerza en el mundo informático, este ha mostrado ser una solución estratégica e idónea.

El principal reto con respecto a la infraestructura es el manejo de usuarios del sistema operativo. Si bien existe mucho sobre esto en la implementación y las credenciales que se almacenan finalmente en la base de datos, el principal reto era separar aquellas cuentas creadas por nuestra solución sobre las existentes. Este reto lo hemos logrado solucionar mediante la implementación de contenedores de Docker que enjaulan la solución.

Se han implementado dos contenedores dentro de la solución. Ambos contenedores se construyen en base a una imagen de Docker (DockerFile) que posee todos los pasos para construirlos. Se ha diseñado crear como mínimo dos contenedores, con el fin de poseer un diseño de micro servicios, capaz de expandirse (clusters de servidores) o de brindar las ventajas del diseño orientado a micro servicios. Los contenedores que se han implementado son solución principal y motor de base de datos. La funcionalidad inicial se encuentra descrita en la **Figura 12.**

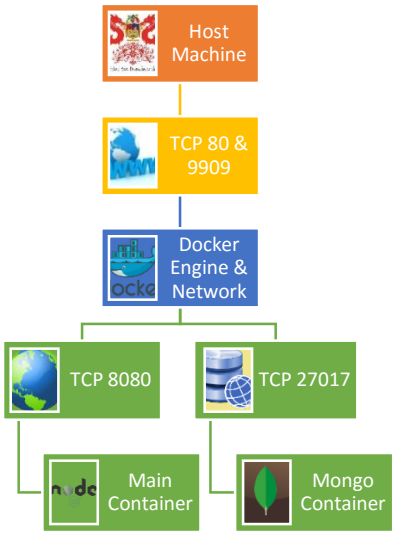


Figura 12. Estructura de los contenedores con respecto a la maquina host.

CONCLUSIONES

Una vez realizada la implementación, se ha procedido a la instalación de la solución dentro de los servidores de la Universidad San Francisco de Quito. La solución se encuentra lista en una fase inicial para poder ser probada por los profesores y finalmente utilizada por los estudiantes en el siguiente semestre. A continuación, se describen las conclusiones encontradas a partir de la implementación y del uso inicial de la solución.

La interfaz de desarrollo presenta la simplicidad esperada en el diseño, permite al estudiante programar sin la complejidad que presentan entornos de desarrollo existentes.

Se espera que el estudiante encuentre un ambiente amigable que contribuya a su proceso de aprendizaje.

En general la solución presenta fluidez en la interacción y la navegación. Se puede concluir que tanto las implementaciones del lado del cliente como del lado del servidor, se encuentran optimizadas según lo descrito en la sección de implementación, aportando velocidad a la solución. El uso de buenas prácticas también colabora con la velocidad y promueve un futuro mantenimiento de la solución.

Con respecto a la experiencia del usuario, se puede concluir que el uso de Angular Material Design ha permitido brindar una solución con componentes atractivos y efectos que mejorar la experiencia al usuario. Mediante la implementación de esta biblioteca también se ha logrado brindar una solución adaptable a cualquier tipo de ordenador que utilice el estudiante, de esta forma cumpliendo uno de los requerimientos. La portabilidad también se ha conseguido mediante la opción de descargar el proyecto y cumple con lo esperado en el diseño.

La solución implementada en la nube, brinda la versatilidad deseada y en las pocas pruebas realizadas no ha demostrado ningún tipo enlentecimiento. Esto permite también concluir que el uso de JavaScript ha sido satisfactorio del lado del servidor y que pese a su manejo de hilo único permite brindar repuestas a varios requerimientos.

El uso de los contenedores por otra parte, demuestran una gran manera para independizar los diferentes productos desarrollados y permite que el servidor donde se aloja la solución no interfiera con implementaciones predecesoras o que se realicen a futuro. Se puede concluir que la forma de manejo de usuarios se encuentra inicialmente estable y que se debe esperar a la utilización de los estudiantes para evaluar completamente su

performance. Por otra parte, cualquier necesidad posterior de migración o mantenimiento se proyecta ser simple por el uso de contenedores. Y se concluye que, de existir cualquier ataque de seguridad, el ataque será contenido dentro del contenedor y no afectará al resto del sistema.

Por último, si bien la solución no ha sido utilizada por los verdaderos usuarios finales, según las pruebas iniciales y mediante el ejercicio de extrapolación, se espera que la implementación junto con la infraestructura pueda abastecer la demanda real de los usuarios. Como en toda solución informática, pueden existir errores, pero se espera que por la estructura de la aplicación y por la documentación presentada en los anexos, estos tengan un bajo impacto.

REFERENCIAS BIBLIOGRÁFICAS

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2011). Improving teaching and learning of computer programming through the use of the Second Life virtual world. *British Journal of Educational Technology*, *42*(4), 624-637.

Jenkins, T. (2002, August). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* (Vol. 4, pp. 53-58).

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, *13*(2), 137-172.

Jenkins, T. (2001, July). Teaching Programming-A journey from teacher to motivator. In *The 2nd Annual Conference of the LSTN Center for Information and Computer Science*.

Mikkonen, T., & Taivalsaari, A. (2011, October). Apps vs. open web: The battle of the decade. In *Proceedings of the 2nd Workshop on Software Engineering for Mobile Application Development* (pp. 22-26).

Cantelon, M., Harter, M., Holowaychuk, T. J., & Rajlich, N. (2014). *Node. js in Action*. Manning.

Mikowski, M. S., & Powell, J. C. (2013). Single Page Web Applications. *B and W*.

Liu, A. (2014). JavaScript and the Netflix user interface. *Communications of the ACM*, *57*(11), 53-59.

Mardan, A. (2014). Building Node. js REST API Servers with Express. js and Hapi. In Practical Node. js (pp. 173-194). Apress.

Khurana, P., & Bindal, P. (2014). Vulnerabilities and Defensive Mechanism of CSRF. *International Journal of Computer Trends and Technology (IJCTT)* (Vol. 13, number 4)

Jeffrey, C. (2014, March 3). Tty.js. Obtenido Mayo 10, 2016, from https://github.com/chjj/tty.js/

Papa, J. (2016). Johnpapa/angular-styleguide. Retrieved May 10, 2016, from https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md

ANEXO A: DOCKERFILE CONTENEDOR PRINCIPAL

```
FROM ubuntu:14.04
MAINTAINER David Villacis C (david@davicompu.com)

RUN apt-get update -y && \
apt-get upgrade -y && \
apt-get install curl nano wget python git -y && \
curl -sL https://deb.nodesource.com/setup_5.x | sudo -E bash - && \
apt-get install nodejs -y && \
apt-get install -y build-essential && \
npm install -g gulp forever

EXPOSE 8080
```

ANEXO B: DOCKERFILE CONTENEDOR MONGODB

```
#FROM mongo:latest
#RUN apt-get update -y && apt-get install nano -y
#EXPOSE 27017

FROM debian:wheezy
MAINTAINER David Villacis C (david@davicompu.com)


# add our user and group first to make sure their IDs get assigned
consistently, regardless of whatever dependencies get added
RUN groupadd -r mongodb && useradd -r -g mongodb mongodb

RUN apt-get update \
   && apt-get install -y --no-install-recommends \
      numactl \
   && rm -rf /var/lib/apt/lists/*

# grab gosu for easy step-down from root
ENV GOSU_VERSION 1.7
RUN set -x \
   && apt-get update && apt-get install -y --no-install-recommends ca-
certificates wget && rm -rf /var/lib/apt/lists/* \
   && wget -O /usr/local/bin/gosu
"https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-
$(dpkg --print-architecture)" \
   && wget -O /usr/local/bin/gosu.asc
"https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-
$(dpkg --print-architecture).asc" \
   && export GNUPGHOME="$(mktemp -d)" \
   && gpg --keyserver ha.pool.sks-keyservers.net --recv-keys
B42F6819007F00F88E364FD4036A9C25BF357DD4 \
   && gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/bin/gosu
\
   && rm -r "$GNUPGHOME" /usr/local/bin/gosu.asc \
   && chmod +x /usr/local/bin/gosu \
   && gosu nobody true \
   && apt-get purge -y --auto-remove ca-certificates wget

# pub   4096R/AAB2461C 2014-02-25 [expires: 2016-02-25]
#       Key fingerprint = DFFA 3DCF 326E 302C 4787  673A 01C4 E7FA AAB2
461C
# uid                  MongoDB 2.6 Release Signing Key
<packaging@mongodb.com>
#
# pub   4096R/EA312927 2015-10-09 [expires: 2017-10-08]
#       Key fingerprint = 42F3 E95A 2C4F 0827 9C49  60AD D68F A50F EA31
2927
# uid                  MongoDB 3.2 Release Signing Key
```

```
<packaging@mongodb.com>
#
ENV GPG_KEYS \
   DFFA3DCF326E302C4787673A01C4E7FAAAB2461C \
   42F3E95A2C4F08279C4960ADD68FA50FEA312927
RUN set -ex \
   && for key in $GPG_KEYS; do \
      apt-key adv --keyserver ha.pool.sks-keyservers.net --recv-keys
"$key"; \
   done

ENV MONGO_MAJOR 3.2
ENV MONGO_VERSION 3.2.5

RUN echo "deb http://repo.mongodb.org/apt/debian wheezy/mongodb-
org/$MONGO_MAJOR main" > /etc/apt/sources.list.d/mongodb-org.list

RUN set -x \
   && apt-get update \
   && apt-get install -y \
      mongodb-org=$MONGO_VERSION \
      mongodb-org-server=$MONGO_VERSION \
      mongodb-org-shell=$MONGO_VERSION \
      mongodb-org-mongos=$MONGO_VERSION \
      mongodb-org-tools=$MONGO_VERSION \
   && rm -rf /var/lib/apt/lists/* \
   && rm -rf /var/lib/mongodb \
   && mv /etc/mongod.conf /etc/mongod.conf.orig

RUN mkdir -p /data/db /data/configdb \
   && chown -R mongodb:mongodb /data/db /data/configdb
VOLUME /data/db /data/configdb
#COPY docker-entrypoint.sh /entrypoint.sh
RUN echo
'db.createUser({user:"user",pwd:"passowrd",roles:[{role:"userAdminAnyDa
tabase",db: "admin"}]});' > /tmp/create.txt && \
    echo 'db.auth("user","password");' >> /tmp/create.txt && \
    echo 'db=db.getSiblingDB("ide");' >> /tmp/create.txt && \
    echo 'db.createUser({user: "admin",pwd:
"7a6bbc51f5b52af110555fc7fb138817",roles: ["readWrite"]});' >>
/tmp/create.txt

RUN echo '#/bin/bash' > /run.sh && \
echo '(mongod --auth &) && echo "Waiting 5 seconds" && sleep 5 && mongo
admin /tmp/create.txt' >> /run.sh && \
chmod +x /run.sh
#ENTRYPOINT ["./run.sh"]
CMD /run.sh
EXPOSE 27017
```

ANEXO C: SCRIPT DE CREACIÓN CONTENEDORES DOCKER

```
#!/bin/sh

#development

docker run --name mongo_container -p 27017:27017 -d mongo:latest
docker run --name main -p 80:8080 -v /opt/IDE:/opt/IDE -v
/opt/UsersIDE/:/home/ -it --link mongo_container:mongo ubuntu_node/main
/bin/bash

#production

docker run --name mongo_container -p 9909:27017 -it server/mongo
/bin/bash
docker run --name main -p 80:8080 -it --link
mongo_container:server/mongo server/node /bin/bash
```

ANEXO D: PACKAJE.JSON

```
{
  "name": "ide",
  "version": "0.1.0",
  "description": "IDE from programming in C and C++",
  "main": "index.js",
  "scripts": {
    "start": "gulp && webpack && node server/index.js",
    "build": "gulp && webpack",
    "test": "mocha test/index.js"
  },
  "author": "David Villacis C.",
  "license": "MIT",
  "devDependencies": {
    "angular": "^1.4.8",
    "angular-animate": "^1.4.8",
    "angular-aria": "^1.4.8",
    "angular-cookies": "^1.4.8",
    "angular-csv-import": "0.0.27",
    "angular-locale": "^0.3.1",
    "angular-material": "^1.0.1",
    "angular-md5": "^0.1.10",
    "angular-messages": "^1.4.8",
    "angular-resource": "^1.4.8",
    "angular-route": "^1.4.8",
    "angular-ui-ace": "^0.2.3",
    "angular-ui-layout": "^1.4.1",
    "angular-ui-router": "^0.2.18",
    "animate.css": "^3.4.0",
    "body-parser": "^1.14.2",
    "chai": "^3.5.0",
    "cookie-parser": "^1.4.1",
    "crypto": "0.0.3",
    "csurf": "^1.8.3",
    "del": "^2.2.0",
    "express": "^4.13.3",
    "express-session": "^1.13.0",
    "font-awesome": "^4.5.0",
    "gulp": "^3.9.0",
    "gulp-clean": "^0.3.1",
    "gulp-concat": "^2.6.0",
    "gulp-minify": "0.0.5",
    "gulp-ngmin": "^0.3.0",
    "gulp-rename": "^1.2.2",
    "gulp-sass": "^2.1.1",
    "gulp-uglify": "^1.5.1",
    "jquery": "^2.2.0",
    "jquery-resizable-dom": "^0.15.0",
    "jstree": "^3.2.1",
    "lodash": "^4.10.0",
    "mocha": "^2.4.4",
```

```
    "mongo-sanitize": "^1.0.0",
    "mongoose": "^4.4.3",
    "ng-dialog": "^0.5.8",
    "ng-grid": "^2.0.1",
    "ng-js-tree": "0.0.7",
    "ng-table": "^0.5.4",
    "run-sequence": "^1.1.5",
    "socket.io": "0.9.16",
    "supertest": "^1.1.0",
    "tty.js": "^0.2.15",
    "ui-router": "^1.0.0-alpha.3",
    "webpack": "^1.12.14"
  }
}
```

ANEXO E: LICENCIA MIT DE TTY.JS