

**Arquitectura del Sistema**

**Proyecto: Precise\_Estimate**

**Revisión 2.0**

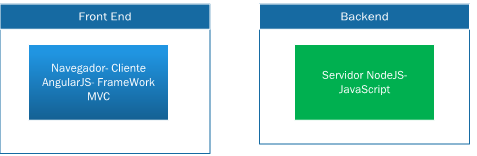
**Sebastián Aguirre**

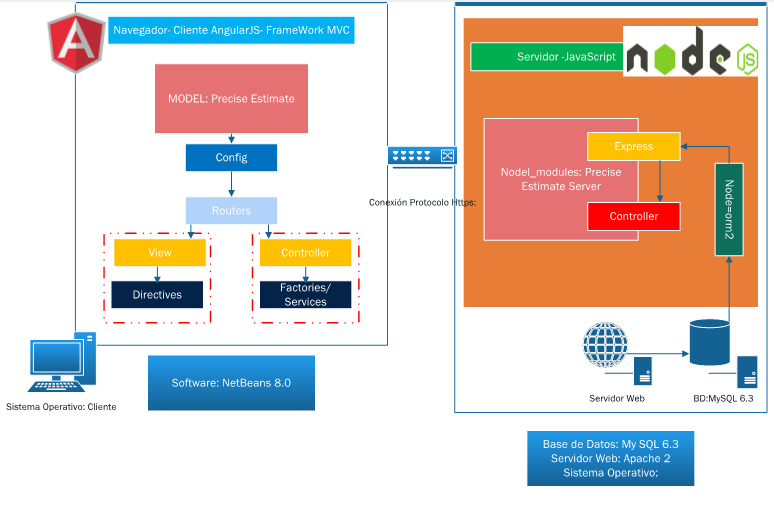
**Felipe Cano**

**Jaime Wilchez**

**Jhon Isaza**

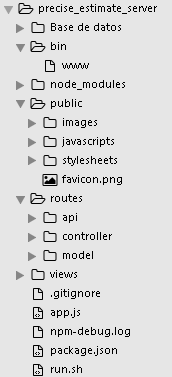
**sábado, 27 de mayo de 2017**

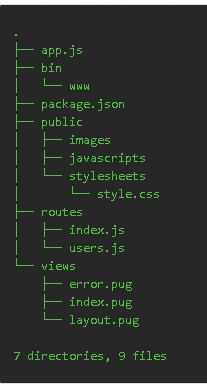




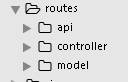
# 1. ARQUITECTURA BACKEND NODE JS.

Nuestro FrameWork de manejo en Node JS. Es Express v2.5.6, una vez instalado express creamos nuestro proyecto con nombre Precise\_Estimate\_Server, en el cual se creó la siguiente estructura del proyecto, donde tenemos todas las dependencias y estructura general.

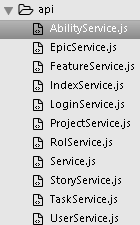




**Routes**: Aca es donde encontramos cada una de las rutas disponibles son definitivamente la parte más importante de nuestra aplicación, porque si estas no están definidas, no existiría la interfaz en Angular. En el ejemplo se generó esta dirección automáticamente, routes fueron definidas en la primera parte del archivo, y está definido como sigue:



En este ejemplo, escogimos una clase que se llama UserService, dentro de la carpeta API. Acá es donde creamos todos los servicios del BackEnd y es donde opera toda nuestra aplicación.

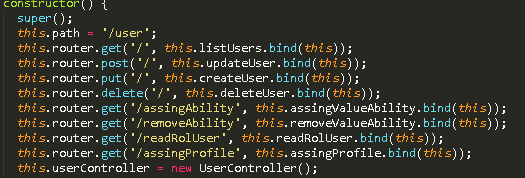


Como podemos ver una ruta esta especificada de la siguiente forma:

app. VERBO (PATH, ACCIÓN);

app ya la conocemos el VERBO en este caso utilizamos: GET, POST, PUT, DELETE

así para cada uno de los verbos HTTP. Para un servicio rest debemos tener estas acciones cada archivo de nuestras carpetas es una clase según el tipo de método hace diferentes acciones y luego lo expone.



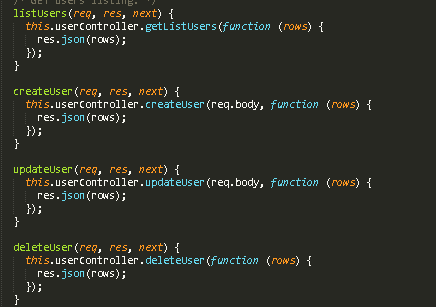
A continuación, definimos con PATH que define la dirección de acceso. ACCION: que es lo que se tiene que hacer. En este caso creamos las funciones de middleware que son funciones que tienen acceso al objeto de solicitud (req), al objeto de respuesta (res) y a la siguiente función de middleware en el ciclo de solicitud/respuestas de la aplicación.

Las funciones de middleware pueden realizar las siguientes tareas:

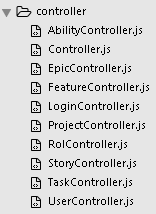
* Ejecutar cualquier código.
* Realizar cambios en la solicitud y los objetos de respuesta.
* Finalizar el ciclo de solicitud/respuestas.
* Invocar el siguiente middleware en la pila.

Si la función de middleware actual no finaliza el ciclo de solicitud/respuestas, debe invocar next() para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud quedará colgada.

El siguiente ejemplo muestra los elementos de una llamada a función de middleware para la clase User Services:



**Controller: Acá es donde tenemos la función que separa el código de las solicitudes que procesa el routes. Lo que nos permite es crear una página HTML, donde nos muestra los datos al usuario en el navegador.**



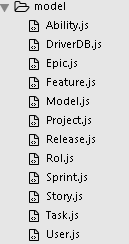
**"Rourtes" reenviar las solicitudes soportadas (y cualquier información codificada en URL de solicitud) a las funciones de controlador apropiadas.**

**El “Controller” funciona para obtener los datos solicitados de los modelos, crear una página HTML que muestre los datos y devolverla al usuario para verla en el navegador.**

**Vistas (plantillas) utilizadas por los controladores para procesar los datos.**

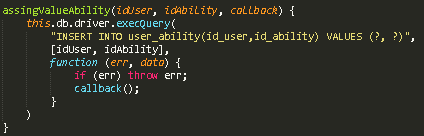


**Model: Nos permite crear los objetos idénticos a los que están en la base de datos, se debe crear las tablas para la conexión de la BD. Nosotros utilizamos la librería MySQL “node-mysql": "^0.4.2", que se puede ver en el archivo package.json. A través de Model se envía los datos de conexión utilizando un componente llamado ORM “Object Relational Mapping” básicamente lo usamos para convertir las tablas de la BD en una serie de entidades que simplifica las tareas y facilita el uso del CRUD.**

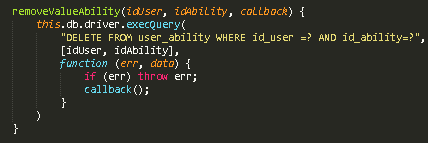


**Ejemplo de User:**

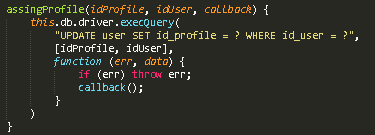
**Campo de Insertar Usuario**



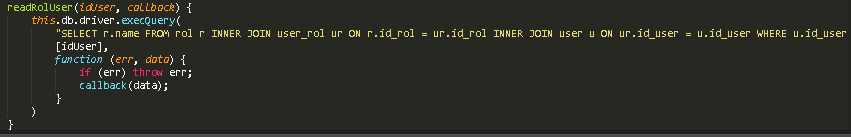
Borrar Usuario



Actualizar Usuario

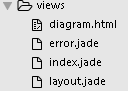


Seleccionar Usuario

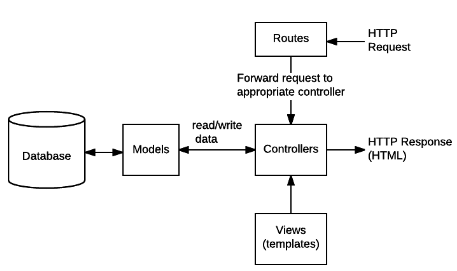


**Acá es donde vemos que podemos tener un inconveniente en el manejo del uso del ORM ya que las entidades de la BD que manejamos tiene campos demasiados extensos que superan los 120 caracteres, esto nos ocasiona problemas en la aplicación ya que al hacer solicitudes sobre el model de User se nos queda corto. Pensamos que debemos definir mejor la estructura y diseño de la BD.**

**View: Es donde tenemos las plantillas utilizadas para los Controller, donde obtenemos los datos, acá podemos imprimir y hacer log-out de todo lo que implementamos.**

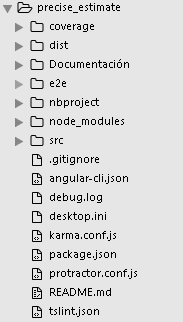


Este es nuestro diseño conceptual de la Arquitectura de Express en relación al Back-End que esta con el nombre de Precise\_Estimate\_Server.



# 2. ARQUITECTURA FRONT-END ANGULAR

Nuestro Front-End lo manejamos con Angular 2.0 una vez instalado creamos nuestro proyecto con nombre Precise\_Estimate\_Server, en el cual se creó la siguiente estructura del proyecto:



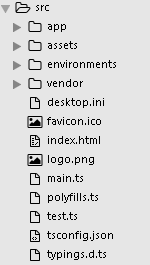
* gitignore: Se genera un cliente de git para poder trabajar de manera versionada en el proyecto y después poder subir cambios al GitHub



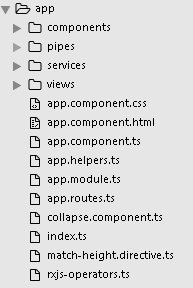
* karma,config.js: Es el archivo de configuración para los test y pruebas



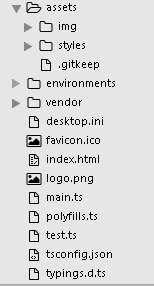
* Patractor.confi.js: Es la configuración de los test que hicimos con jasmie.
* Tslint.js: Es la configuración del typscripts de las dependencias de cada uno de mis servicios que están en el SRC



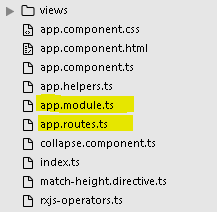
App: Donde vamos a tener todos nuestros componentes de servicios, son aquellos elementos que tengan que ver con la vista de nuestra aplicación.



Asset: Aquellos elementos que no son propios de un componente en angular como imágenes, videos etc.



Index: Donde existe nuestro Single Web APP



* App.Module: Encargado de entender que componentes y dependencias tenemos en nuestra app.
* App.Routes: Conjunto de configuración de rutas, donde están direccionadas las porpiedades de acceso a los servicios.