****

**UNIVERSIDAD DE NARIÑO**

**CENTRO DE INFORMÁTICA**

**PRÁCTICAS Y ESTANDARES DE PROGRAMACIÓN PARA EL DESARROLLLO DEL SISTEMA INSTITUCIONAL DE INFORMACION INTEGRADO**

**Elaborado por: Ing. HERNAN DAVID VILLOTA.**

**Aprobado Por: MSc. JUAN CARLOS CASTILLO**

**Director Centro de Informática**

**SAN JUAN DE PASTO**

**AGOSTO DE 2016**

**CONTROL DE CAMBIOS**

|  |  |  |  |
| --- | --- | --- | --- |
| **VERSION** | **FECHA ACEPTACIÓN** | **DESCRIPCIÓN** | **AUTOR** |
| 1.0 | 24/08/2016 | Versión inicial del documento | David Villota |
|  |  |  |  |

CONTENIDO

[1. INTRODUCCION 6](#_Toc459826428)

[2. OBJETIVO GENERAL 7](#_Toc459826429)

[3. OBJETIVOS ESPECÍFICOS 8](#_Toc459826430)

[4. ARCHIVOS JAVASCRIPT 9](#_Toc459826431)

[4.1. ARCHIVOS JAVASCRIPT WEB SERVICES 11](#_Toc459826432)

[4.1.1. MODELOS O ENTIDADES 12](#_Toc459826433)

[4.1.2. DAOS 14](#_Toc459826434)

[4.1.3. ARCHIVOS DE CONFIGURACION 17](#_Toc459826435)

[4.1.4. ARCHIVOS ADICIONALES (HELPERS) 17](#_Toc459826436)

[4.1.5. ARCHIVOS CONTROLADORES 17](#_Toc459826437)

[4.1.6. ARCHIVO DE RUTAS 18](#_Toc459826438)

[4.2. ARCHIVOS JAVASCRIPT ANGULAR 19](#_Toc459826439)

[4.2.1. ARCHIVOS CONTROLADORES 19](#_Toc459826440)

[4.2.2. ARCHIVOS SERVICES 20](#_Toc459826441)

[4.2.3. ARCHIVOS DIRECTIVES. 20](#_Toc459826442)

[4.2.4. ARCHIVOS DE CONFIGURACION. 20](#_Toc459826443)

[5. ARCHIVOS CSS 21](#_Toc459826444)

[6. ARCHIVOS HTML 22](#_Toc459826445)

[7. LIBRERÍAS 23](#_Toc459826446)

[8. IMÁGENES Y OTROS ARCHIVOS 24](#_Toc459826447)

[9. CONSULTAS SQL 25](#_Toc459826448)

[10. NORMAS DE DOCUMENTACION DE CODIGO FUENTE 26](#_Toc459826449)

[10.1. DOCUMENTACION WEB SERVICES. 26](#_Toc459826450)

[10.1.1. CONTENIDO DE LOS ARCHIVOS 27](#_Toc459826451)

[10.2. DOCUMENTACION APLICACIONES ANGULAR 29](#_Toc459826452)

[10.2.1. CONTROLADORES 30](#_Toc459826453)

[10.2.2. SERVICIOS 31](#_Toc459826454)

[10.2.3. DIRECTIVAS 31](#_Toc459826455)

[10.2.4. FILTROS 32](#_Toc459826456)

[10.2.5. ARCHIVOS DE CONFIGURACION Y OTROS. 32](#_Toc459826457)

[11. CONCLUSIONES 34](#_Toc459826458)

[12. RECOMENDACIONES 35](#_Toc459826459)

[BIBLIOGRAFÍA 36](#_Toc459826460)

# TABLA DE ILUSTRACIONES

[Figura 1. Declaración de arreglos y objetos 10](#_Toc459914636)

[Figura 2. Declaración de una función 10](#_Toc459914637)

[Figura 3. Utilización función forEach 11](#_Toc459914638)

[Figura 4. Estructura de un archivo JavaScript en los web services 11](#_Toc459914639)

[Figura 5. Ejemplo estructura de un campo 13](#_Toc459914640)

[Figura 6. Estructura general de una entidad 14](#_Toc459914641)

[Figura 7. Objeto queries de un DAO 15](#_Toc459914642)

[Figura 8. Retorno de un método en el DAO 16](#_Toc459914643)

[Figura 9. Función del DAO que utiliza un callback 16](#_Toc459914644)

[Figura 10. Ejemplo estructura archivo helper 17](#_Toc459914645)

[Figura 11. Estándar archivo rutas 18](#_Toc459914646)

[Figura 12. Estructura IIFE de un archivo JavaScript de angular 19](#_Toc459914647)

[Figura 13. Uso de $inject en los componentes de angular 19](#_Toc459914648)

[Figura 14. Estructura general de un controlador en angular 19](#_Toc459914649)

[Figura 15. Estructura de un service en angular 20](#_Toc459914650)

[Figura 16. Descripción del archivo JavaScript 26](#_Toc459914651)

[Figura 17. Descripción de archivo módulo de express 26](#_Toc459914652)

[Figura 18.Formato documentación de una función 27](#_Toc459914653)

[Figura 19. Documentación de una función con un callback como parámetro 27](#_Toc459914654)

[Figura 20. Documentación de una variable 28](#_Toc459914655)

[Figura 21.Ejemplo de documentación de un archivo 29](#_Toc459914656)

[Figura 22. Documentación de un controlador de Angular 30](#_Toc459914657)

[Figura 23. Documentación de un método 30](#_Toc459914658)

[Figura 24. Documentación de una variable en angular 31](#_Toc459914659)

[Figura 25. Documentación de un servicio en angular 31](#_Toc459914660)

[Figura 26. Documentación de una directiva 32](#_Toc459914661)

[Figura 27. Documentación de un filtro 32](#_Toc459914662)

[Figura 28. Documentación de archivos de configuración y otros archivos de angular 32](#_Toc459914663)

# INTRODUCCION

Las buenas prácticas de programación, establecen estándares en el desarrollo de las aplicaciones de software, se establecen con el fin de facilitar la compresión del código por parte de otro programador, estos estándares presentan lineamientos en la nomenclatura de archivos, variables y componentes del código fuente que en la medida de lo posible deben aplicarse.

Por otra parte en el documento se presentan normas de documentación del código fuente estas normas de documentación permiten identificar la funcionalidad de cada uno de los componentes que hacen parte de la aplicación. Adicional a ello esta documentación permite por medio de plugins adicionales generar páginas web completas a modo de tutorial, conocido como manual del programador.

El documento presentado a continuación establece estas reglas de programación en los archivos de código fuente, inicialmente en los web services y sus componentes, posterior a ello indica las normas que se aplicaran para los componentes de las aplicaciones en angular y finalmente establece las reglas de documentación tanto para los web services como para las aplicaciones del cliente.

# OBJETIVO GENERAL

Establecer políticas relativas a la codificación y documentación de los archivos que harán parte del proyecto del sistema institucional de información integrado de la Universidad de Nariño.

# OBJETIVOS ESPECÍFICOS

* Establecer políticas de codificación de los archivos JavaScript, CSS, HTML y otros que hagan parte del proyecto.
* Establecer normas para nomenclatura de los archivos y organización de los documentos dentro de las carpetas.
* Establecer normas de documentación del código fuente para los archivos JavaScript tanto de las aplicaciones cliente como de los web services.

# ARCHIVOS JAVASCRIPT

Los archivos JavaScript se caracterizan por tener la extensión \*.js, en el proyecto se encuentran de manera general tanto en los web services como en las aplicaciones cliente, ejecutando diversas funcionalidades dentro de la aplicación, algunos se encuentran como archivos de configuración y otros como archivos que realizan operaciones dentro del sistema, las normas que se deben aplicar para cada tipo de archivo JavaScript se mencionan a continuación.

* **VARIABLES**

Al ser JavaScript un lenguaje no tipado existe cierta dificultad en identificar de que tipo será cada variable, o si representa a un objeto en particular, es importante que el nombre de la variable defina de manera clara, cuál es el objetivo de su creación y que uso se le dará dentro de la función o archivo en el cual se declara. De ser posible se debe inicializar la variable con un valor específico o con el valor que será tipada, por ejemplo:

* ***Cadena* :** var cadena=””;
* ***Numero*:** var numero=0;
* ***Objeto*:** var objeto=null;
* ***Objeto JSON*:** var json= {};
* ***Arreglo* :** var array= [];

El nombre de las variables siempre será en minúscula y se debe utilizar la notación camel case (ej. nuevaVariable, variableCadena).

En el caso de declararse constantes estas deben ser nombradas con letras mayúsculas como es la convención en la mayoría de lenguajes de programación.

* **ARREGLOS Y OBJETOS JSON**

Los arreglos deben ser declarados con un nombre descriptivo de lo que almacenan, y en caso de ser construidos por defecto deben identarse para poder identificar que elementos pertenecen a un arreglo en particular

De la misma manera en caso de ser declarados por defecto los objetos JSON deben identarse, para poder identificar cuando un elemento pertenezca a un determinado objeto

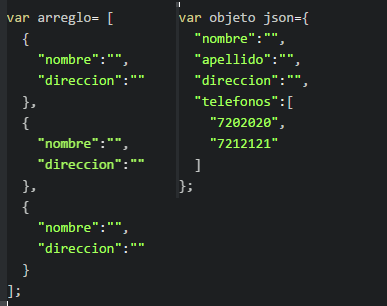


Figura . Declaración de arreglos y objetos

* **FUNCIONES**

Las funciones son una parte muy importante de JavaScript y existen múltiples formas de declararlas, pueden instanciarse de manera estándar, como función anónima o declararlas asignándola a una variable en particular. En el proyecto se procurará a menos de que sea un caso excepcional ser declaradas como variable de la siguiente manera

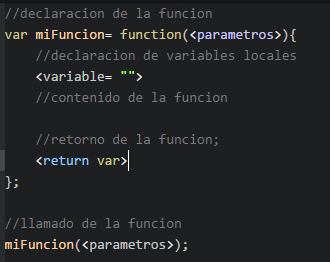


Figura . Declaración de una función

Como se puede ver en el ejemplo el nombre de las funciones al igual que las variables será en letra minúscula, haciendo uso de la notación camel case, en la parte inicial de la función se deben declarar las variables globales de las que se hará uso, a continuación, el código de la función y finalmente el retorno en caso de que retorne algo o sin retorno si solo realiza una acción específica.

Es importante destacar que las funciones deben ser comentadas con la documentación que se describirá en el apartado de documentación tanto en los web services como en las aplicaciones de angular, de acuerdo a los parámetros que se establezcan en esta sección.

Existen funciones cuyo parámetro puede ser un callback (otra función de JavaScript), este callback debe ser descrito de manera clara en la documentación de la función para conocer que parámetros se asignaran a esta función que se ejecutara posteriormente.

* **CONDICIONALES Y CICLOS**

En el caso de los condicionales no existe una norma específica solo se sugiere identar el código para facilitar su comprensión, ya que un código mal identado dificulta la lectura del mismo por otro desarrollador. Se sugiere en la medida de lo posible utilizar operadores ternarios cuando las condiciones lo permitan.

En cuanto a los ciclos se pueden utilizar los mismos tipos de ciclos que existen en otros lenguajes de programación (for, while, do while), pero se sugiere que cuando se realice un recorrido de un arreglo de objetos se utilice la función forEach de JavaScript de la siguiente manera



Figura . Utilización función forEach

## 4.1. ARCHIVOS JAVASCRIPT WEB SERVICES

Los archivos de JavaScript en los web services generalmente tendrán una estructura definida de la siguiente manera.

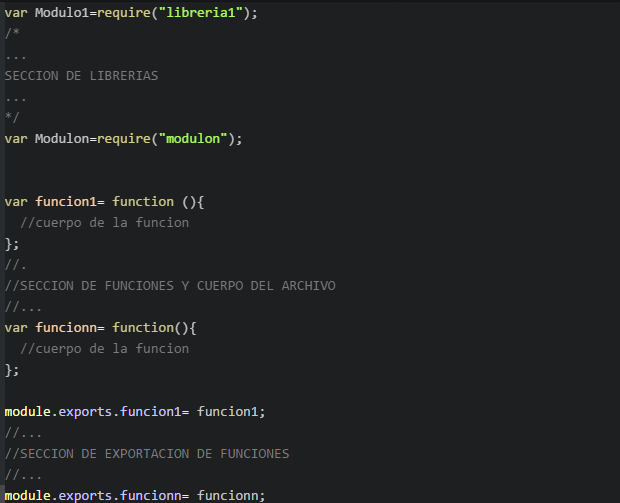


Figura . Estructura de un archivo JavaScript en los web services

Existe una sección de librerías donde se importan todos los módulos y librerías de las que va a hacer uso el archivo JavaScript. Las librerías implementadas deben iniciar siempre con letra mayúscula.

La siguiente sección es el cuerpo del archivo donde se declaran funciones y variables globales.

Finalmente, la sección de exportación de funciones y variables donde se declaran las funciones y variables que se harán públicas y podrán usar los demás módulos que implementen el archivo.

El nombre del archivo siempre iniciara con letra minúscula e ira en camel case a excepción de ciertos casos.

A continuación, se hará una descripción de los estándares que se manejaran en cada uno de los tipos de archivo JavaScript que se encuentran en los web services.

### 4.1.1. MODELOS O ENTIDADES

Los modelos o entidades se entienden como la abstracción de las tablas de la base de datos y permiten trabajar con cada registro de la base de datos como un objeto de JavaScript, estos modelos o entidades son definidos y utilizados por el ORM Sequelize que hace parte de la arquitectura del proyecto.

Ellos tienen una particularidad al ser implementados por esta herramienta, debido a ello establecen ciertas reglas a utilizar:

1. El nombre del archivo debe ser escrito en formato snake case y debe iniciar con el nombre del esquema al cual pertenece la tabla separando con punto el nombre de la misma, por ejemplo la tabla persona debe tener un modelo llamado esquema.persona.js, la tabla usuario\_contabilidad debe tener un modelo llamado esquema.usuario\_contabilidad.js, de esta manera se puede identificar de manera clara a que tabla pertenece cada modelo.
2. Como primera línea del archivo debe estar la línea ‘use strict’; es una norma del framework y debe respetarse.
3. A continuación de esta línea se realiza la declaración de los módulos o archivos a importar, que se necesiten en la entidad.
4. A continuación se define el nombre del modelo o entidad este debe ser declarado en formato camel case iniciando con mayúscula.
5. Posterior a ello, se definen los campos de la tabla estos campos deben ser declarados en la medida de lo posible con el nombre que se le da al campo en la base de datos, como en la base de datos se utiliza la convención snake case en este caso también será así.
6. Ya que el ORM permite tipificar cada campo de la tabla se debe utilizar el tipo de dato que más se aproxime, de esta forma el ORM validara la precisión del dato que se manipule.
7. Es importante definir el nombre del campo que se utiliza en la tabla de la base de datos para evitar incompatibilidades.
8. En esta descripción de los campos se debe colocar la mayoría de restricciones que se tengan en la base de datos es decir condiciones de unicidad, condiciones de nulidad, etc, con el fin de realizar esta validación en el código, previo a la comunicación con la base de datos.

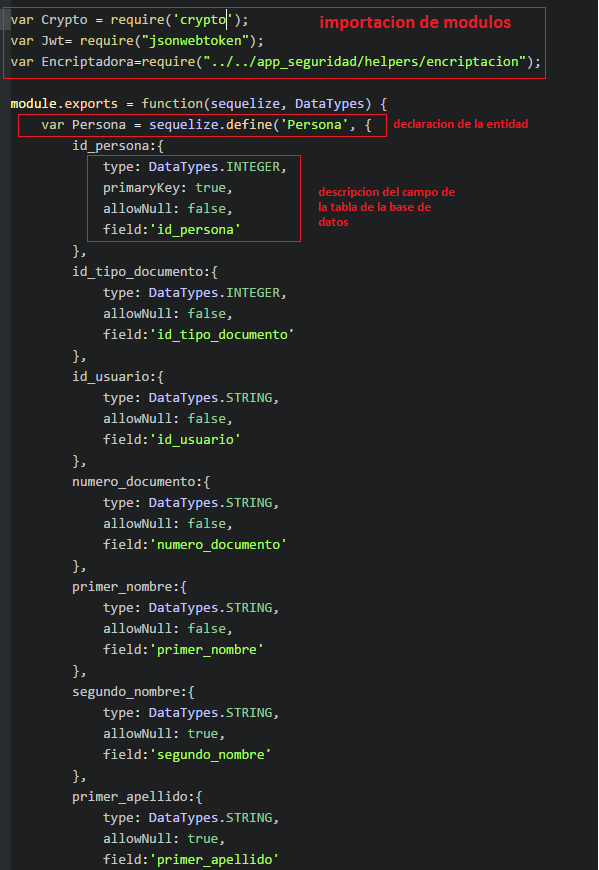


Figura . Ejemplo estructura de un campo

1. La variable de configuración de timestamps debe estar en false para no insertar esos campos en las tablas de la base de datos.
2. La variable de configuración underscore debe estar en true para poder identificar campos de la base de datos que están en minúscula y en notación snake case.
3. La variable de configuración schema debe identificar el schema al cual pertenece la tabla.
4. La variable de configuración freezeTableName debe estar en true para poder utilizar el nombre de la tabla y de schema que se defina en estas variables de configuración.
5. La variable de configuración tableName debe tener el nombre de la tabla en la base de datos.
6. Dentro de la sentencia classMethods únicamente se define las asociaciones de las tablas, en casos excepcionales se declara una nueva función pero en la medida de lo posible no debe hacerse.
7. En instanceMethods se pueden definir múltiples funciones, solo se debe recordar la norma que se aplica para las funciones en general, es importante también que sean documentadas para poder ser comprendidas por cualquier otro desarrollador. En general la estructura de una entidad es de la siguiente manera

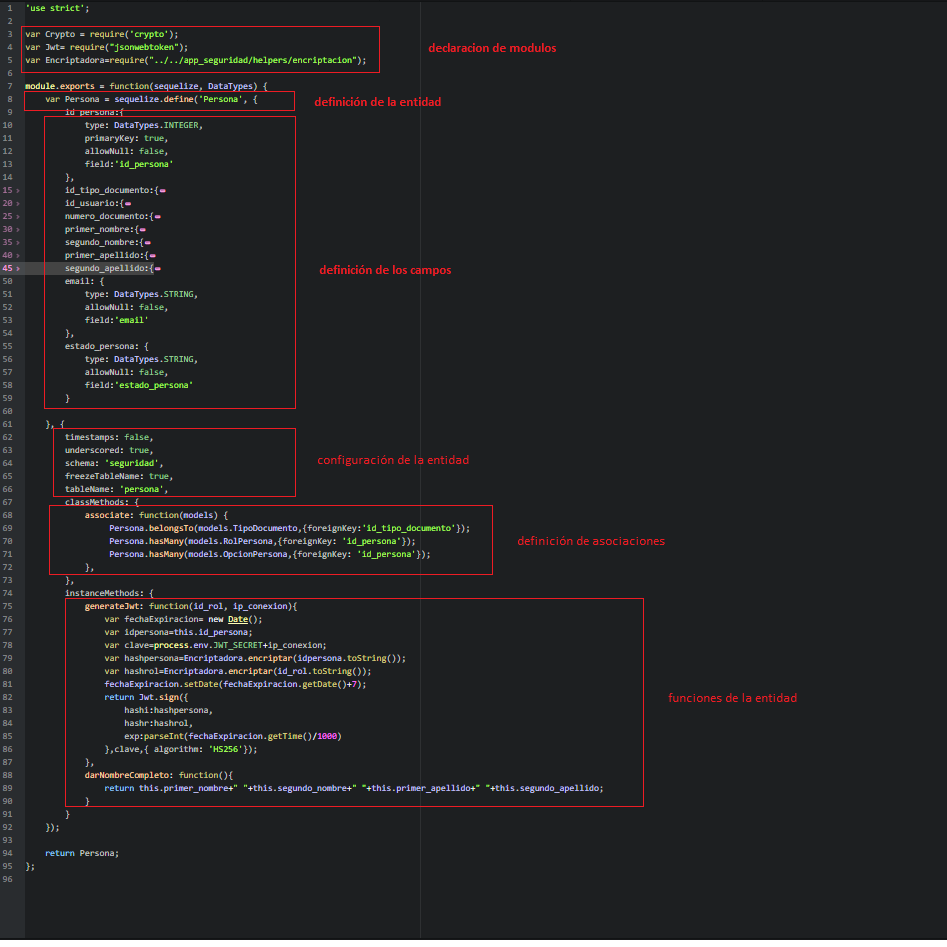


Figura . Estructura general de una entidad

### 4.1.2. DAOS

Los DAO (Data Access Objects) son archivos que permiten realizar las operaciones en la base de datos haciendo uso de las entidades, en ellos se deben definir las funciones para crear, insertar, eliminar y consultar información de la base de datos, asignando a esta capa la abstracción de esas operaciones. En cuanto a los DAOs se establecen las políticas que se mencionan a continuación:

1. El nombre del DAO debe ser el mismo de la entidad en formato camel case y terminando con la palabra dao, es decir si tengo la entidad rol\_persona.js el dao debe llamarse rolPersonaDao.js
2. Inicialmente mantiene la estructura de los archivos de web services de JavaScript, como se describió previamente (sección de librerías, cuerpo del archivo y sección de exportación de métodos).
3. En la parte inicial de los DAOs se debe registrar todas las sentencias de base de datos nativas que se van a utilizar, estas consultas se organizaran de la manera que se presenta a continuación dentro del objeto “queries”:

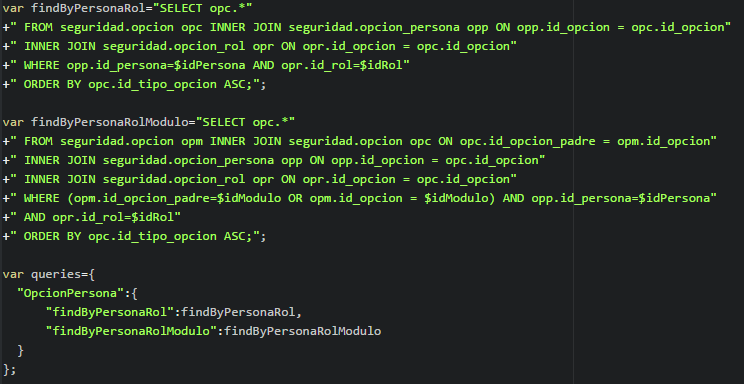


Figura . Objeto queries de un DAO

1. Cada sentencia de base de datos se debe instanciar en una variable aparte en el formato que se define de escritura de sentencias en la respectiva sección de consultas sql.
2. En este archivo se deben definir todas las operaciones que se van a realizar sobre la base de datos las funciones deben tener nombres descriptivos. pero con un patrón en particular, para las búsquedas se utilizara la palabra “find” seguido por la palabra “by” y el campo por el que se va a buscar es decir si se va a buscar un registro por el id seria findById, se debe mantener la notación camel case.
3. Si la búsqueda puede regresar más de un registro se utiliza la palabra “findAll” y la política que se describió previamente, es decir si se buscan una cantidad de registros por un campo en particular seria “findAllByCampo”.
4. En el caso de la creación de registros se utiliza la palabra create, para actualizaciones update y para eliminación destroy, si se va a realizar una búsqueda para realizar la actualización o eliminación se utilizara de la misma forma la palabra “by” y el campo, por ejemplo: “updateById”, “destroyByNombre”.
5. En general se retorna en cada método, la función que realiza la operación en Sequelize, de esta manera tratando la promesa que arroja el método en el controlador o la clase que implemente el DAO.

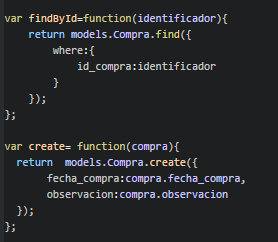


Figura . Retorno de un método en el DAO

1. En el caso de las transacciones y procedimientos complejos de consulta sobre la base de datos se podrá hacer uso de una función callback como parámetro, con el fin de facilitar su uso en el controlador o clase que lo implemente, por ejemplo:

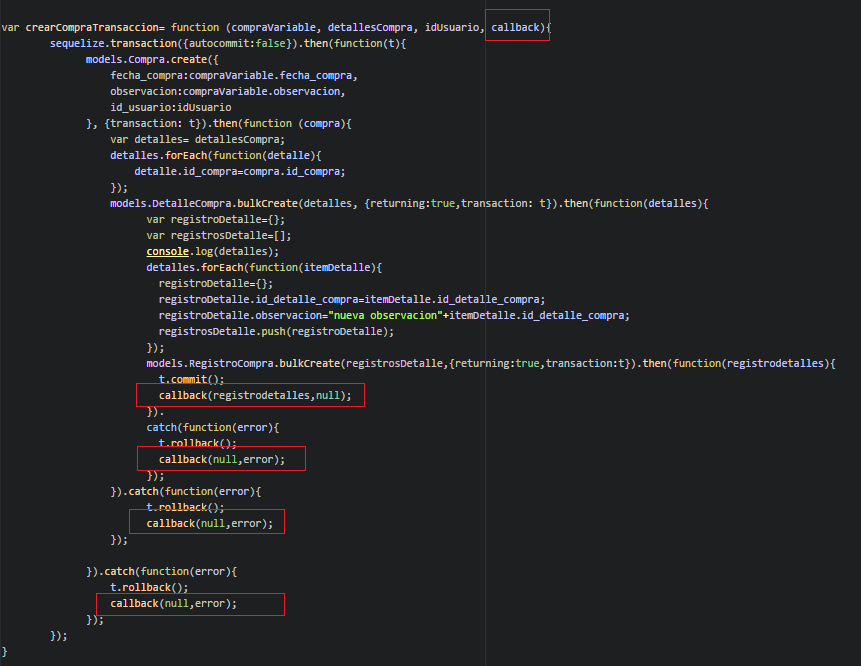


Figura . Función del DAO que utiliza un callback

1. Si la operación a realizar no tiene un patrón en particular, como búsqueda, inserción, eliminación o actualización, se debe colocar un nombre descriptivo de la función.
2. Al final del archivo se exporta únicamente los métodos que serán públicos y podrán ser utilizados por los otros archivos.

### 4.1.3. ARCHIVOS DE CONFIGURACION

En los archivos de configuración no se puede establecer un patrón en particular ya que es difícil identificar lo que realizan y contienen, únicamente se sugiere aplicar las reglas de documentación para archivos de configuración y si estos archivos tienen alguna función en particular también debe ser documentada. Finalmente se sugiere mantener una identación en el código fuente manteniendo así la legibilidad para otros desarrolladores.

### 4.1.4. ARCHIVOS ADICIONALES (HELPERS)

Los helpers son archivos que agrupan funciones genéricas, en general la convención para estos archivos mantiene la estructura básica de un archivo de JavaScript en los web services, con una sección de importación de librerías un cuerpo del archivo y una sección de exportación de funciones. Se debe documentar las funciones que agrupa el archivo y cada una de ellas debe ser comentada de manera clara y descriptiva. En general un helper se encuentra codificado de la siguiente forma:

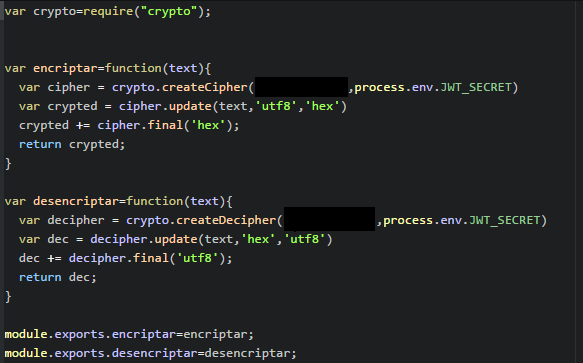


Figura . Ejemplo estructura archivo helper

### 4.1.5. ARCHIVOS CONTROLADORES

Los archivos controladores son aquellos que permiten manipular la lógica del negocio en los web services, ellos contienen las funciones que cumplen con este objetivo, estos archivos controladores deben ser nombrados en camel case y finalizando con la palabra “controller”, como estructura general tienen una sección inicial de importación de módulos, una sección de cuerpo del archivo donde se definirán las funciones y atributos, finalmente se encuentra una sección de exportación de las funciones que serán públicas a las rutas y a otros controladores que los implementen.

### 4.1.6. ARCHIVO DE RUTAS

Los archivos de rutas se encuentran generalmente dentro de la carpeta routes con un nombre que pueda identificarlos claramente o simplemente con index.js si solo se tiene un archivo de rutas. Estos archivos manejan la misma estructura de los archivos de JavaScript del lado del servidor con la diferencia que solo se exporta una variable al final que es la variable routes.

En general las convenciones aplicadas son las mismas que en los demás archivos JavaScript, se sugiere agrupar las rutas de acuerdo a la funcionalidad, y con un comentario descriptivo de dicha funcionalidad. En este archivo de rutas solo se debe encontrar importación de funciones de los controladores y otros archivos, es decir que no se debe realizar ningún tipo de codificación adicional, a menos de que no exista otra forma de hacerlo.

Si es una función específica el nombre de la ruta debe definir claramente la operación que va a realizar, en caso contrario si únicamente son las operaciones básicas de base de datos sobre una tabla en particular (creación, eliminación, consulta, actualización), solo ira el nombre de la tabla en singular.

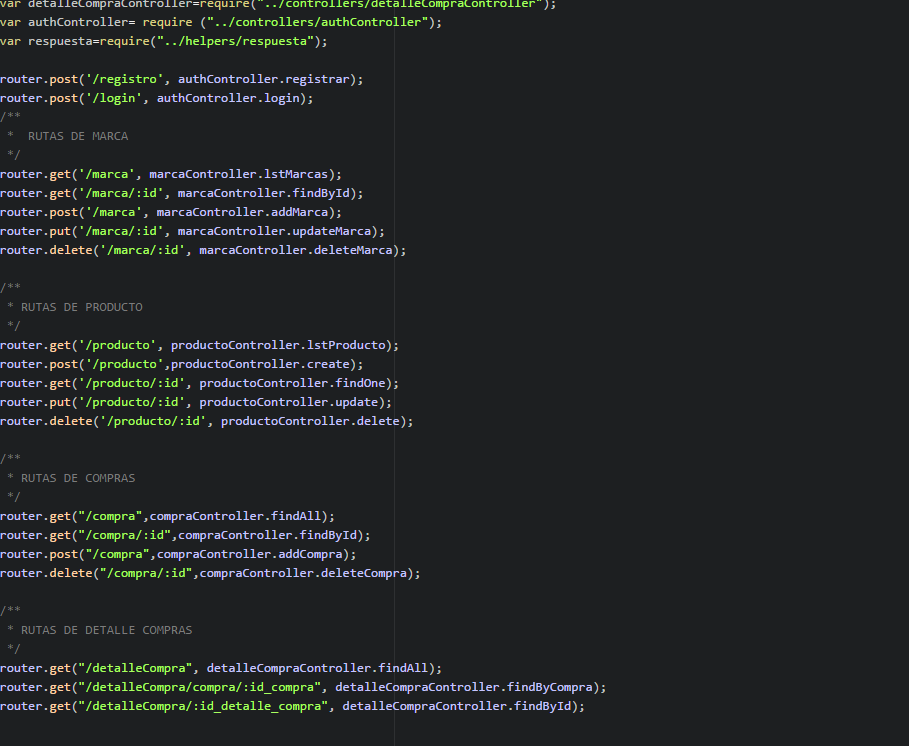


Figura . Estándar archivo rutas

## 4.2. ARCHIVOS JAVASCRIPT ANGULAR

En las aplicaciones angular se encuentran diversos tipos de archivos JavaScript y se diferencian de acuerdo a la funcionalidad que cumplen dentro de la aplicación. En general estos archivos deben tener un nombre en minúsculas y ser muy descriptivo de acuerdo a la actividad que desempeñan. Por otra parte siempre se utiliza una expresión IIFE, de la siguiente manera.

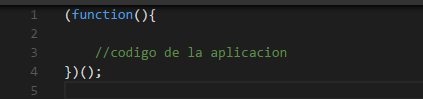


Figura . Estructura IIFE de un archivo JavaScript de angular

Otra particularidad que deben tener todos los archivos por defecto es la inclusión de la línea $inject previa a la declaración de la función principal.



Figura . Uso de $inject en los componentes de angular

### 4.2.1. ARCHIVOS CONTROLADORES

Los controladores en angular se deben nombrar terminando con el texto .controller.js y con un identificador claro que defina su función, la estructura del controlador, como se había definido previamente en todas las funciones de JavaScript declara inicialmente una sección de variables locales, posterior a ello la sección de métodos y finalmente exporta los métodos que son públicos y se utilizan en las plantillas HTML.

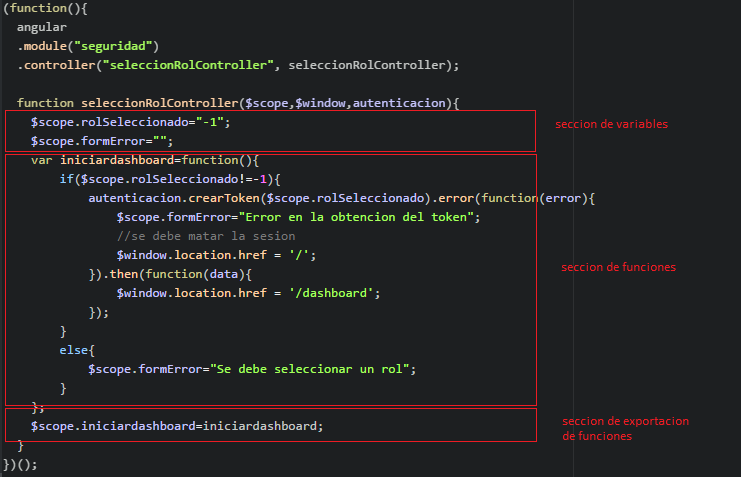


Figura . Estructura general de un controlador en angular

### 4.2.2. ARCHIVOS SERVICES

Los services en angular deben ser nombrados con la terminación.service.js, tiene al igual que los controladores una sección de declaración de variables, una sección de funciones y finalmente una sección en la cual se importan las funciones que se van a utilizar. La nomenclatura de variables y funciones sigue los mismos estándares de todos los archivos de JavaScript.

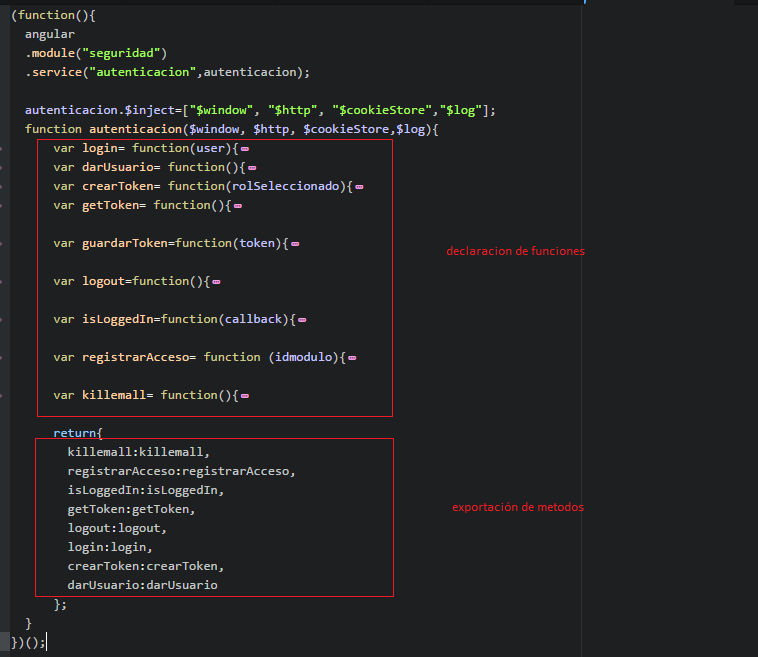


Figura . Estructura de un service en angular

### 4.2.3. ARCHIVOS DIRECTIVES.

Para este tipo de archivos únicamente se requiere que el nombre sea descriptivo y termine con el texto .directive.js, dentro del código se seguirá el estándar de angular para la codificación de estos componentes.

### 4.2.4. ARCHIVOS DE CONFIGURACION.

Los archivos de configuración no siguen una política específica debido a que pueden ser muy variables, por lo cual se sugiere seguir los estándares que plantea angular y mantener el código identado, si dentro de estos archivos se codifica una función esta debe estar claramente documentada.

En el caso del archivo de configuración que almacena las cadenas y constantes de texto que se utilizan durante toda la aplicación, debe manejar nombres claros para cada una de estas variables.

# ARCHIVOS CSS

Los archivos CSS no tienen una norma en particular debido a que se utilizara el pre compilador CSS stylus, son archivos con la extensión “\*.styl”. Los archivos de stylus como norma deben declarar en el inicio del código fuente las variables a utilizar, posterior a ello se debe establecer las reglas para el uso de fuentes personalizadas y finalmente las reglas CSS, por defecto y como norma del compilador se debe tener el código identado, como sugerencia general es importante tener en cuenta que no se debe sobrescribir código CSS previamente declarado.

# ARCHIVOS HTML

Los archivos HTML hacen parte de la aplicación angular en diversas formas, bien sea como plantillas, partials e incluso como archivos HTML independientes importados a la aplicación, para este tipo de archivos es recomendable que el código se encuentre debidamente identado y las etiquetas estén identificadas con un id en los formularios y elementos de entrada y salida de información.

Dentro de estos archivos HTML nunca se deben escribir reglas de estilo CSS o código de JavaScript, el código debe mantenerse limpio de este tipo de sentencias.

Los nombres de clases e identificadores que se utilicen en las etiquetas deben ser claros, descriptivos y asociados al tipo de entrada de datos que maneja la etiqueta, manteniendo la convención camel case de la siguiente forma:

* **Seleccionables:** cmb<nombre>, por ejemplo cmbCiudad
* **Textos:** txt<nombre>, por ejemplo txtNombre
* **Checkbox:** chk<nombre>, por ejemplo chkSexo
* **Radio Buttons:** rb<nombre>, por ejemplo rbReligion
* **Botones:** btn<nombre>, por ejemplo btnInsertar

Si se encuentra otro tipo de entrada de datos, se debe utilizar las tres iniciales del tipo y el nombre del campo.

Finalmente si el archivo hace parte de la estructura de la aplicación angular, el nombre debe terminar por la cadena .view.html para identificar que es una vista o un partial dentro de la aplicación.

# LIBRERÍAS

En ciertas ocasiones se hace necesario la inclusión de nuevos plugins, librerías o componentes externos, estas librerías en el caso de los web services siempre deben ser importadas a través de npm y guardadas en el archivo package.json.

En el caso de las aplicaciones angular se recomienda que todas las librerías externas a angular y a su filosofía, sean almacenadas dentro de la carpeta vendor, los componentes a adicionales del núcleo de angular deben ser almacenados al mismo nivel de la librería base de angular; En la medida de lo posible se debe utilizar el archivo bower.json para implementar estos nuevos componentes, con el objetivo de mantener un registro de que plugins adicionales se está usando en el proyecto.

# IMÁGENES Y OTROS ARCHIVOS

Archivos adicionales como imágenes, fuentes y otros deben ser agrupados en su respectiva carpeta y deben tener un nombre descriptivo de la utilidad que se les va a dar.

Los archivos de imágenes deben estar en formato \*.png con transparencia, y las fuentes deben ser almacenadas en todos los formatos compatibles para los navegadores (\*.eot, \*.svg, \*.ttf, \*.woff, \*.woff2, \*.otf).

# CONSULTAS SQL

Para la escritura de sentencias nativas de SQL en el código fuente, acción que únicamente se permitirá dentro de los archivos DAOs. Se definen las siguientes normas:

1. Toda palabra nativa de SQL debe ir en mayúsculas (FROM, WHERE, AND, ORDER BY). Al igual que las funciones y demás sentencias SQL.
2. Todas las tablas referenciadas en una consulta deben tener un alias, este alias debe componenrse por las letras iniciales del esquema y de la tabla que se está manejando en la consulta.
3. Las consultas SQL deben estar identadas para facilitar su comprensión, adicional a ello se requiere que cada sentencia SQL vaya en una línea separada y en el caso de las condiciones, parámetros o joins se coloquen como máximo 3 por línea por ejemplo:

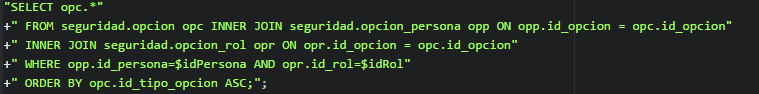


Figura . Ejemplo sentencia SQL

1. El cruce de tablas siempre debe hacerse a través del uso de sentencias JOIN
2. Se debe evitar el uso de sentencias IN o NOT IN, para aplicar estas restricciones se debe procurar hacer uso de los LEFT o RIGTH JOIN.

# NORMAS DE DOCUMENTACION DE CODIGO FUENTE

En esta sección se establecen las normas de documentación que se aplican para los archivos de JavaScript, esta documentación se realiza con el objetivo de, mantener legibilidad en el código fuente y finalmente a partir de los comentarios generar un manual de programador completo de todas las funciones y archivos que se utilizan en el código fuente.

Para la documentación de los archivos de JavaScript de los web services se utiliza la herramienta JSDoc, que permite la generación de un tutorial a partir de los comentarios realizados en el código fuente

Por otra parte para la documentación de los archivos de JavaScript de las aplicaciones angular se utilizara el plugin ngDocs, que a través del uso de etiquetas en los comentarios del código fuente permite documentar de manera más precisa la aplicación, teniendo en cuenta las características del framework y los elementos que lo componen, es decir controladores, directivas, servicios, filtros etc.

## DOCUMENTACION WEB SERVICES.

Para todos los archivos de JavaScript del lado del servidor se requiere que en cada uno de ellos inicialmente se escriba la descripción del archivo de la siguiente manera:

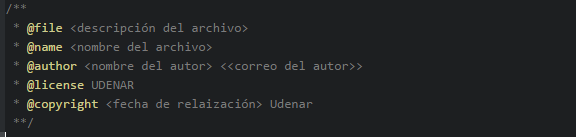


Figura . Descripción del archivo JavaScript

Sin importar la funcionalidad del archivo, se insertarán estas líneas de comentario iniciales, la descripción del archivo debe explicar que componentes se codifican en él, los demás datos deben ser verídicos y describir claramente de quien es la licencia y a quien se ceden los derechos del código fuente.

En el caso de archivos controladores o agrupaciones de funciones adicionales, posterior a la sección de importación de librerías debe ir la siguiente línea de comentarios:

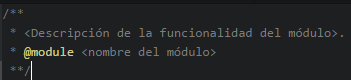


Figura . Descripción de archivo módulo de express

La descripción de la funcionalidad del módulo debe ser clara y especificar el objetivo del módulo, el nombre del módulo debe iniciar en mayúsculas y llevar la notación camel case.

### 10.1.1. CONTENIDO DE LOS ARCHIVOS

En cuanto al contenido de los archivos únicamente se documentaran las funciones de los módulos y en el caso de que sea necesario un atributo en particular. En el caso de los modelos o entidades no se documentaran los atributos ya que la documentación de estos atributos se encuentra en los comentarios que se realizan a las tablas de la base de datos

El código de documentación de las funciones debe ir inmediatamente anterior a la codificación de la misma y deben documentarse en el siguiente formato:

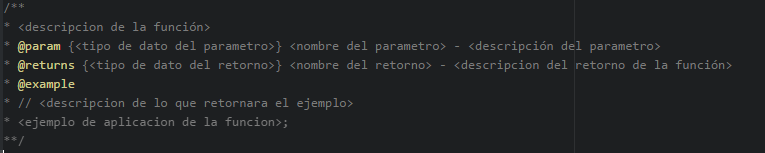


Figura .Formato documentación de una función

La descripción de la función debe ser clara acerca de lo que realiza, debe dar a entender que es lo que hace.

La etiqueta @param indica el inicio de la descripción de un parámetro, pueden haber tantas etiquetas @param como parámetros de la función, como se indica en el ejemplo estas etiquetas incluyen entre llaves el tipo de dato del parámetro, posterior a ello se indica el nombre del parámetro y finalmente la descripción del parámetro.

La etiqueta @returns indica que iniciara la descripción del dato que retorna la función esta etiqueta es opcional, si la función no retorna ningún tipo de dato simplemente no se incluye.

Finalmente la etiqueta @example indica que se documentara un ejemplo del uso de la función esta etiqueta es opcional y se utiliza en el caso de que se esté documentando una función de difícil compresión.

En el caso de que unos de los parámetros de la función sea un callback debe ser documentado de la siguiente manera:

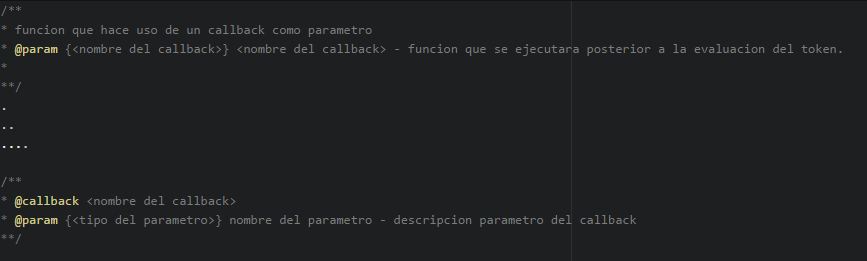


Figura . Documentación de una función con un callback como parámetro

La función que se indica en el ejemplo hace uso de un callback, este callback debe ser documentado al final del archivo para poder comprender de manera más explícita que parámetros deben pasarse una vez sea llamado en la función principal. En la parte inferior del ejemplo se indica cómo se documenta el callback; en la documentación de esta función se debe escribir el nombre y los parámetros que se enviaran desde la función.

Para los atributos o variables globales de los módulos se establece el siguiente formato:

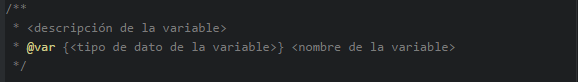


Figura . Documentación de una variable

Este código de documentación debe ir previo a la declaración de la variable, la descripción debe ser clara y debe dar a entender la funcionalidad de la variable dentro del archivo, como se mencionó previamente no es obligatoria la documentación de las variables pero si se desea hacerlo se debe utilizar este formato.

Es importante aclarar que cualquier tipo de comentario es válido siempre y cuando pueda dar una mayor compresión del código a cualquier otro desarrollador.

Finalmente un ejemplo de cómo se debe mirar el código fuente documentado de la manera correcta teniendo en cuenta las normas planteadas se presenta a continuación:

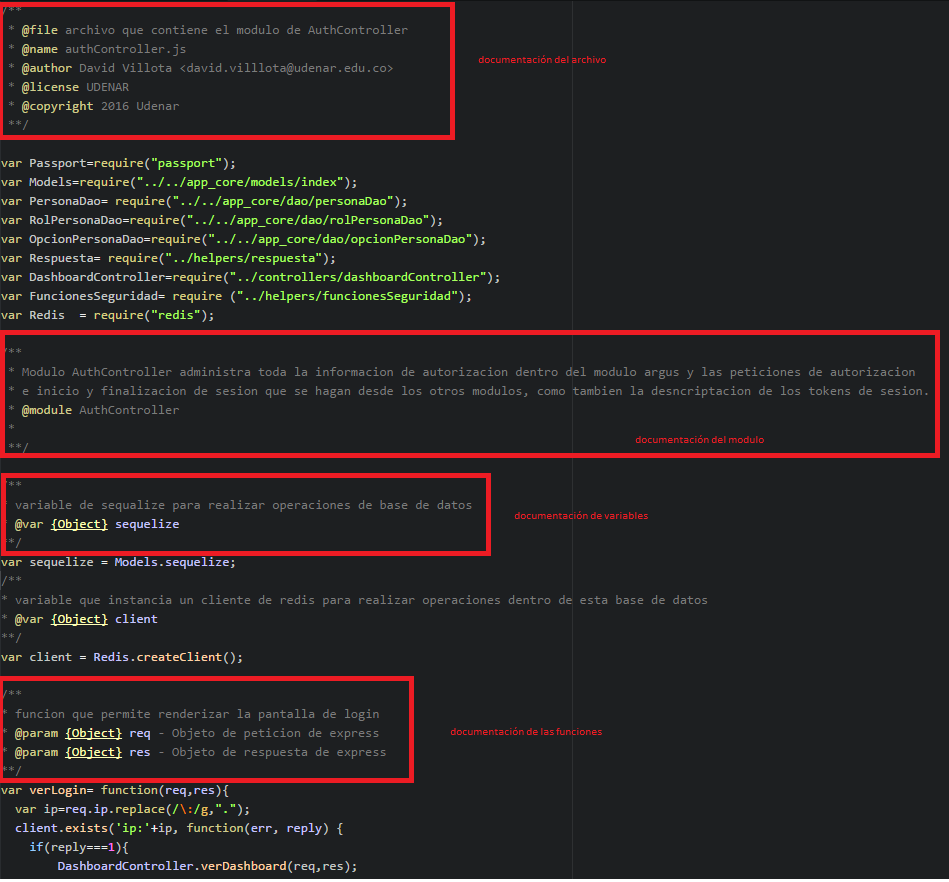


Figura .Ejemplo de documentación de un archivo

## DOCUMENTACION APLICACIONES ANGULAR

Como se mencionó previamente la documentación de las aplicaciones angular se realizara haciendo uso del plugin ngDocs, por medio de los parámetros y etiquetas que ofrece este plugin permitirá crear un manual de programador y tutorial de las funciones en base a los comentarios que se realicen sobres los archivos.

La ventaja de este plugin sobre otras herramientas diseñadas para documentar es que tiene en cuenta los diferentes componentes que se pueden crear en las aplicaciones de angular ofreciendo múltiples etiquetas dependiendo del componente que se está documentando. En el proyecto se realizara la documentación de los controladores, servicios, filtros, directivas y archivos de configuración teniendo en cuenta las reglas que se mencionan a continuación.

### 10.2.1. CONTROLADORES

Para los controladores es necesario que se documente la funcionalidad principal del controlador y los métodos que lo componen, los atributos son opcionales. En la parte inicial del controlador se debe comentar en base a la siguiente estructura:

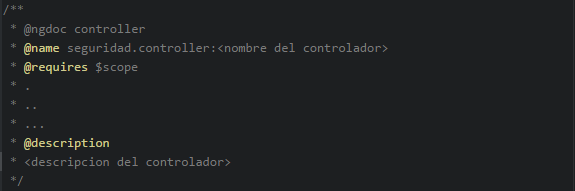


Figura . Documentación de un controlador de Angular

El nombre del controlador debe ser el nombre que se le ha dado en la función, la etiqueta @requires denota las dependencias que tiene un controlador en particular y existirán tantas etiquetas de este tipo como numero de dependencias tenga el controlador, por último la descripción del controlador debe ser clara acerca del funcionamiento del mismo.

Los métodos de un controlador se documentan en base al siguiente formato:

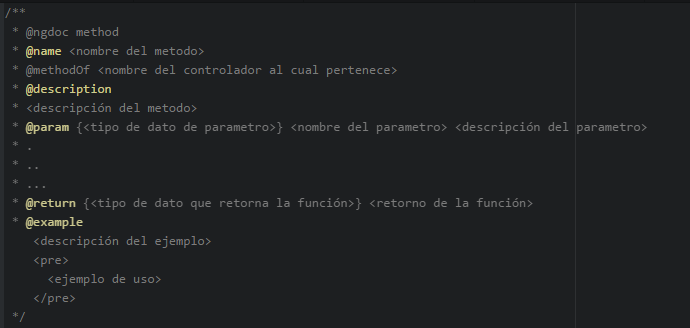


Figura . Documentación de un método

El nombre del método especifica el nombre de la función que se está documentando, en la etiqueta @methodOf se debe escribir el nombre del controlador al cual pertenece la función, la descripción del método debe ser clara acerca de la operación que realiza.

La etiqueta @param describe los parámetros que utiliza la función, se debe describir el tipo de parámetro el nombre de parámetro y especificar que dato aporta para la función.

En la etiqueta @return se especifica el retorno de la función en caso de que exista, con su respectivo tipo de dato y la descripción del dato que se retorna. Por último la etiqueta @example ofrece la posibilidad de indicar un ejemplo de uso de la función, esta etiqueta será de uso obligatorio cuando la función utiliza un callback (una función que se ejecutara de manera posterior al desarrollo del método), en esta etiqueta se debe indicar inicialmente la descripción del ejemplo, y posterior a ello entre la etiqueta <pre> un ejemplo de uso de la función.

Si existe la necesidad de documentar un atributo del controlador se debe hacer teniendo en cuenta el siguiente formato:

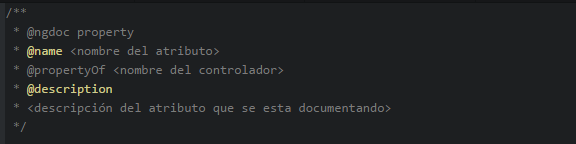


Figura . Documentación de una variable en angular

El nombre del atributo debe ser el nombre de la variable que se está documentando, en el nombre del controlador se coloca el controlador al que pertenece el atributo, finalmente la descripción debe ser clara y dar a entender la funcionalidad de esta variable dentro de la aplicación.

### 10.2.2. SERVICIOS

Los servicios en angular serán comentados en un formato similar al controlador.

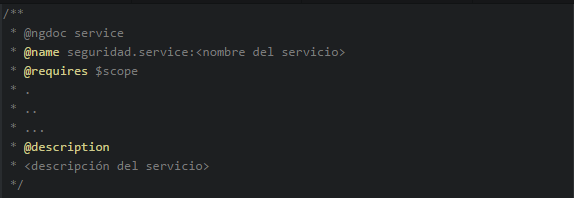


Figura . Documentación de un servicio en angular

El nombre del servicio es el nombre del componente que se está documentando, al igual que en los controladores la etiqueta @requires indica las dependencias que va a tener el componente, finalmente la descripción debe ser clara acerca de la funcionalidad del servicio dentro de la aplicación.

Los métodos y atributos del servicio deben ser documentados en el mismo formato utilizado en los controladores.

### 10.2.3. DIRECTIVAS

Las directivas en angular deben ser documentadas en base a la siguiente estructura de comentarios:

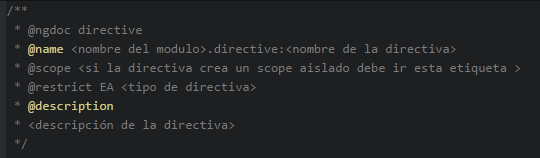


Figura . Documentación de una directiva

El nombre de la directiva debe ser el que se manejará en toda la aplicación, la etiqueta @scope solo se coloca si la directiva crea un scope asilado al del controlador que la implementa, en la etiqueta @restrict se especifica el tipo de directiva si será elemento (E), atributo (A) o si es posible instanciarla de las dos maneras (EA). Finalmente la descripción de la directiva debe ser clara, explicando su uso y la funcionalidad que desempeña

### 10.2.4. FILTROS

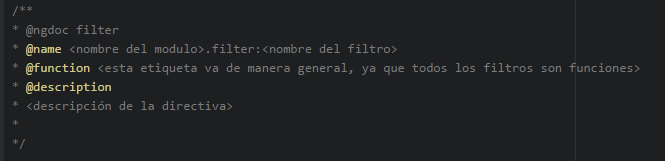


Figura . Documentación de un filtro

Los filtros se documentan en el formato presentado anteriormente, se debe diligenciar el nombre del filtro tal y como será llamado en la aplicación, la etiqueta @function se encontrara en la mayoría de los casos ya que los filtros generalmente son funciones, por último la descripción del componente debe ser clara, explicando la funcionalidad del filtro en la aplicación angular.

### 10.2.5. ARCHIVOS DE CONFIGURACION Y OTROS.

Para los archivos de configuración y archivos adicionales que utilice la aplicación se plantea la siguiente estructura de comentarios:

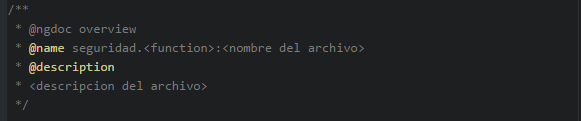


Figura . Documentación de archivos de configuración y otros archivos de angular

El tipo de documentación overview permite dar una visión general del archivo, se debe especificar el nombre el tipo de función que ejecuta el componente y finalmente una descripción del archivo y su uso en la aplicación

En el caso de que existan métodos o propiedades que deban ser documentadas, se seguirá el formato que se utiliza para los otros componentes.

Por ultimo de igual manera que en los web services, es importante que se escriba en el código fuente cualquier comentario adicional que oriente al desarrollador hacia la compresión de un método en particular.

# CONCLUSIONES

* La definición de un estándar de prácticas de programación, permite que la legibilidad del código fuente sea más fácil para otros desarrolladores, garantizando así un mayor índice de mantenibilidad y escalabilidad las aplicaciones construidas.
* La definición de un estándar de documentación del código fuente permite que sea más comprensible la funcionalidad de un componente de código fuente para otro desarrollador, brindando así herramientas para realizar un soporte con mayor eficiencia.
* Los estándares de documentación y de prácticas de programación deben ser ajustados a la estructura planteada de los proyectos ya que cada lenguaje y cada herramienta de desarrollo tiene ciertas particularidades, que los diferencia.

# RECOMENDACIONES

* Las normas y estándares planteados en este documento deben ser aplicados a todos los módulos desarrollados para el sistema de información institucional integrado, para mantener una homogeneidad tanto en la escritura de código como en la documentación del mismo.
* Si bien esta guía abarca la mayoría de componentes de código fuente puede que muchos de ellos hayan quedado por fuera o vayan generándose en cuanto el sistema de información crezca, es recomendable que sea revisada y actualizada de manera periódica.
* La guía planteada está establecida de acuerdo a las características y herramientas que se plantearon para el desarrollo del proyecto, en el caso de que algunas de estas condiciones cambien, esta guía debe ser actualizada a las nuevas tecnologías que se implementen.

# BIBLIOGRAFÍA

* Martin, J. (5 de Enero de 2015). *Starting off with NgDocs*. Obtenido de Starting off with NgDocs: http://www.podpea.co.uk/blog/starting-off-with-ngdocs/
* Williams, J. (26 de Agosto de 2016). *@use JSDoc*. Obtenido de @use JSDoc: http://usejsdoc.org/