Plan de Administración del Código

1. Introducción

En este documento se presentan las guías y lineamientos que se deben seguir al momento de realizar el desarrollo de las funcionalidades del Sistema. Aquí se encontraran las condiciones que se deben presentar en la fase de desarrollo de funcionalidades del sistema, de tal manera que se presente el plan de administración del código y las convenciones de codificación en marcadas en las mejores prácticas de programación.

1.1 Propósito

El objetivo de este documento es brindar los lineamientos que se deben tener en cuenta en la fase de desarrollo de funcionalidades, por el equipo involucrado en esta fase del ciclo de vida del software. Se darán las guías de administración y elaboración del código desarrollado y reutilizado en el desarrollo de las diferentes funcionalidades del sistema.

1.2 Alcance

El alcance de este documento es establecer las mejoras prácticas aplicables a los desarrollos de software enmarcadas en un plan de administración de código de la fase de análisis para el Sistema desarrollado por el Centro de Investigación y Desarrollo de Información Geográfica CIAF.

1.3 Referencias

Este documento toma como referencia al material elaborado por “Instituto Geografico Agustín Codazzi.” especificamente la plantilla, para la elaboración del documento, también toma como referencia el estilo de codificación *"JavaScript Standard Style"* elaborado por *"https://feross.org/"*

2. Organización del codigo

En este apartado se encontraran las diferentes mejoras prácticas aplicables a el manejo de código de manera que convertirlo en componentes de código entendible, documentado, claro, y lo más importante, reutilizable.

Por cuestiones de refactorización y usabilidad, cada archivo no deberá superar las 1000 lineas de codigo, de ser así de deberá aplicar algún patrón predefinido.

El estilo de escritura a usar será *"CamelCase"* en *"lowerCamelCase"*

El estilo de codificación implementado será tomado de *"JavaScript Standard Style"*

2.1 Organización de Archivos

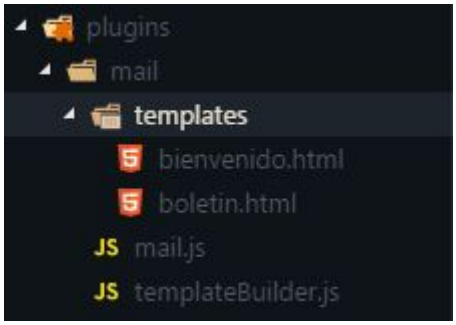
Los archivos de codigo fuente estarán demarcados con un identificador del mismo estarán contenidos en las carpetas que describe su modulo o sub modulo.

2.1.1 Carpeta API

Cada api tendra su carpeta y tendrá tres archivos: `{nombre_api.schema}.js` que tendrá el esquema de valicación, `{nombre_api}.model.js` dónde se describirá el modelo de la entidad mongoose, y `{nombre_api}.js` será el controlador del api.

2.1.2 Carpeta Plugins

En esta carpeta aqí se crearán los diferentes microservicios y middlewares que se vayan presentando, por ejemplo existirá un plugin llamado mail, que será el modulo encargado de enviar correos y tendrá su propio estructura de carpetas



2.1.3 Carpeta Configuración

Los archivos de configuración estarán demarcados por `{nombre_archivo}.config.json` y allí se encontrarán todas las configuraciones que se puedan modificar en producción, también estarán las configuraciones de los plugins

2.2 Codigo fuente

Al ser escrito en node js se debe tener en cuenta el paradigma `non-blocking` y `blocking` esto quiere decir que el codigo será escencialmente escrito de forma asincrona, también el codigo será debilmente tipado y las clases se usarán por organización del codigo, más no será orientado a objetos sino a eventos.

2.2 Comentarios

Se manejan dos tipos de comentarios: Los comentarios de tipo documentación y los de tipo implementación. Los comentarios tipo implementación suelen utilizar la presentación de comentarios del lenguaje de programación C++, donde son delimitados por los caracteres /*...*/ para comentarios en bloque, o // para comentarios por líneas. Los comentarios de implementación son

útiles para realizar comentarios acerca del código y sus particularidades de implementación. Los comentarios de documentación son útiles para describir el código específicamente, comentarios asociados al código directamente sin estar sesgados por la implementación. Los comentarios en el código suelen dar una vista general del código y brindan información adicional que no es legible directamente en el código, con el objetivo de que se pueda entender el contenido del archivo de código fuente del programa.

2.2.1 Comentarios de clases

```
import express from 'express';
import SRV_CONFIG from '../../config/server.config.json'

/**
 *
 * Clase experta en información, cada api creada deberá heredar de esta
 * @singleton Api
 */
class Api {
    /**
     * Creates an instance of Api.
     * @param {any} Entity
     * @param {any} pathApi
     * @memberof Api
     */
    constructor(Entity, pathApi) {
        //Instancia a la base de datos para la entidad jugador, solo se hace una vez
        this.dbEntity = new Entity();
        //Se hace binding a funciones asincronas para que puedan manipular this
        this.get = this.get.bind(this);
        this.getById = this.getById.bind(this);
        this.post = this.post.bind(this);
        this.put = this.put.bind(this);
        this.search = this.search.bind(this);
        this.delete = this.delete.bind(this);
        this.schemaValidator = this.schemaValidator.bind(this);
        //Instancia de ruteador(Creador de apis)
        this.router = express.Router();
        //Path de consumo
        this.path = pathApi;
        //Primer middleware, evalua si la trama que se envia es correcta
        this.router.use(this.path, this.schemaValidator);
        //Mapa de ruteo
        this.router.get(this.path, this.get);
        this.router.post(this.path, this.post);
        this.router.get(`${this.path}/:id`, this.getById);
        this.router.post(`${this.path}/search`, this.search);
        this.router.put(`${this.path}/:id`, this.put);
        this.router.delete(`${this.path}/:id`, this.delete);
        return this.router;
    }
```

Como se ve en la imagen se debe definir en los comentarios que tipo de clase es `@singleton`, `@class`, `@interface`, y una breve descrioción de su funcionalidad.

```
/**
* Clase experta en informaciòn, cada api creada deberá heredar de esta
* @class Api
*/
class Api { ...
```

2.2.2 Comentarios de funciones

```
/**
 *
 * Actualiza los campos enviados en la trama segùn el id de la entidad enviada en la url
 * @param {any} req => trama recibida
 * @param {any} res => respuesta a enviar
 * @param {any} next => ejecuta el siguiente middleware
 * @memberof Api
 */
put(req, res, next) {
  this.dbEntity.findById(req.params.id)
    .then(entity => {
      entity.set(req.body);
      entity.save()
        .then(updatedEntity => {
          res.send(updatedEntity);
        })
        .catch(err => res.send(err))
    })
    .catch(err => res.send(err));
}
```

Como se ve en la imagen se debe dar una breve descripcion de la funcionalidad, `@param` definirá el tipo de parametro es y una descripción del mismo, `@memberof` definirá a que clase corresponde, esto por si es necesario hacer sobreescritura de algun metodo.

```
/**
 *
 * Actualiza los campos enviados en la trama segùn el id de la entidad enviada en
la url
 * @param {any} req => trama recibida
 * @param {any} res => respuesta a enviar
 * @param {any} next => ejecuta el siguiente middleware
 * @memberof Api
 */
put(req, res, next) {
class Api { ...
```

2.3 Declaraciones

Se recomienda seguir las siguientes reglas de codificación

- Para indexación se debe usar `tab` de tamaño 2, para facilidad se puede crear un archivo `.editorconfig` y pegar la siguietne configuración:

```
[*]
indent_size = 2
indent_style = tab
```

- Se deben usar comillas sencillas para declarar un caracter o cadena de caracteres.
- No se deben permitir variables sin uso.
- Debe haber 1 espacio despues de palabras reservadas, por ejemplo.

```
for (let i = 0; i < 500; i++)...
o
function ejemplo (args) {...
```

- Siempre usar `===` en vez de `==`
- Siempre usar `!==` en vez de `!=`
- Tratar de usar operadores ternarios en vez de if (solo si es necesario)

```
□□
let vaso = 'agua';
let valido = vaso === 'agua';
□
let vaso = 'agua';
let valido;
if (vaso === 'agua') {
 valido = true;
} else {
 valido = false;
}
```

- Seimpre hacer `catch` a los errores dentro de las promesas

3 Repositorio

Los archivos relacionados con el código fuente, archivos de implementación, archivos de configuración de interfaz de usuario y del sistema, DEBERAN quedar almacenados en un repositorio en el servidor que estará controlado por la herramienta Git. Dicha herramientas servirá para tener control sobre las diferentes versiones que se tengan de las funcionalidades desarrolladas para la adecuada implementación y como parte del control y seguimiento al código.

En dicho repositorio, se encontraran una serie de carpetas asociadas a cada uno de los casos de uso del sistema. Al interior se encontraran los archivos de código fuente controlados por Tortoise y Subversión. NO DEBEN existir archivos de múltiples versiones, ya que el versionado de cada archivo será administrado por dichas herramientas.

En dicho repositorio, se deben alojar todos los archivos necesarios para la adecuada ejecución de cada una de las funcionalidades desarrolladas asociadas a los diferentes casos de uso teniendo en cuenta los pasos asociados para el funcionamiento de las herramientas Subversion y Tortoise.

Se DEBEN alojar todos los archivos, de tal forma que posteriormente se pueda replicar dicha funcionalidad gracias a la documentación y a las fuentes necesarias para poderlo incluir en el banco de componentes.

Los archivos se DEBEN depositar diariamente los diferentes archivos de código fuente para tener control del cambio de versiones día a día. Se DEBEN seguir los pasos acordados para el adecuado manejo, manipulación, acceso, sincronización y demás reglas de uso de las herramientas con el repositorio unificado de código.

3.1 Clonar repositorio

Para tener una copia del proyecto se debe ejecutar el siguiente comando, para este ejmplo tendremos un repositorio publico subido a la plataforma `GitHub`

```
git clone git@github.com:jugonzalez40/soccer-api.git
```

3.2 Clonar repositorio

Para subir los cambios guardados al repositorio `GitHub` se deben ejutar los siguientes comandos en ese orden

```
git add .
git commit -m "Titulo del commit" -m "Descripción del commit"
git push origin <nombre_branch|master>
```