**Universidad de Costa Rica**

***Especificación de Diseño***

**NodeTortoise**

***Versión 1.0***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bitácora de Cambios** | | | | |
| **NodeTortoise** | | | | |
| **Fecha Cambio** | **# Sección Cambio** | **Descripción del Cambio** | **Justificación del Cambio** | **Autor(es) del Cambio** |
| 12/01/15. | Documento completo | Se creó la primera versión del documento |  | Oscar Martínez |

**Tabla de contenido**

1 Introducción 1

1.1 Propósito del documento 1

1.2. Definiciones, Acrónimos y Abreviaciones 1

2 Descripción general del proyecto 2

2.1. Alcance del Sistema 2

2.2. Requerimientos funcionales 2

2.3. Requerimientos no funcionales 3

3 Especificación de diseño 4

3.1. Integración con otros Sistemas 4

3.1.1. Dependencias funcionales y de datos 4

3.1.2. Diagrama de integración 4

3.2. Diseño de datos 5

3.2.1. Diagrama detallado de la base de datos 5

3.2.2. Diagrama del modelo relacional 5

3.2.3. Descripción de los datos 5

3.3. Diseño de la arquitectura 6

3.3.1. Diagrama de la arquitectura 7

3.3.2. Diseño a nivel de componentes 7

3.4. Diseño de mensajes entre componentes 42

3.4.1. Diagrama de componentes 42

3.5. Diseño de la interfaz de usuario 43

3.5.1. Diagrama de navegación entre pantallas 43

4. Convenciones de programación 44

# 1 Introducción

## 1.1 Propósito del documento

Este documento presenta la documentación técnica de la aplicación de NodeTortoise.

## 1.2. Definiciones, Acrónimos y Abreviaciones

Definiciones

1. Control (input): elemento HTML manipulable por el usuario, el cual corresponde al valor de una variable de la instancia, donde al ser modificado, se modifica el comportamiento de la simulación.
2. Elemento de salida (output): elemento HTML donde se despliega el valor de ciertas variables de la simulación, en un momento dado.
3. CamelCase: es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en CamelCase se asemejan a las jorobas de un camello.
4. UpperCamelCase: cuando la primera letra de cada una de las palabras es mayúscula.
5. LowerCamelCase: igual que la anterior con la excepción de que la primera letra es minúscula

Abreviaturas

1. UI: Interfaz de usuario

# 2 Descripción general del proyecto

### 2.1. Alcance del Sistema

NodeTortoise permite que varios usuarios ejecuten simulaciones participativas en forma conjunta, a través de Internet. Las simulaciones ejecutadas son modelos de NetLogo, los cuales son convertidos a formato web (HTML, JavaScript, CSS) por Tortoise. NodeTortoise permite el usuario importar y almacenar los modelos, listo para ser ejecutados. La funcionalidad de convertir los modelos a formato web no existe en NodeTortoise, por lo que el usuario debe utilizar Galapagos (servidor donde se ejecuta Tortoise) para llevar a cabo esta conversión. De forma adicional, la aplicación cuenta con secciones informativas sobre los proyectos relacionados e información de ayuda. Se asume que el usuario importará los modelos utilizando una versión de Galapagos compatible. Por cuestión de orden en la ejecución de las simulaciones, únicamente el usuario que creó la sesión de simulación podrá inicializar (reset) o ejecutar y detener (go) la ejecución de una simulación. No obstante, cualquier usuario podrá manipular los controles de las variables, siempre y cuando la simulación no esté en ejecución.

### 2.2. Requerimientos funcionales

|  |  |  |
| --- | --- | --- |
| **ID (\*)** | **Historia de usuario** | **Descripción** |
| RF-01 | Importar modelos a la librería | El sistema debe permitir el usuario importar y almacenar un modelo exportado desde Galapagos. |
| RF-02 | Visualizar los modelos en la librería | El sistema debe permitir al usuario visualizar la librería de modelos importados al sistema. |
| RF-03 | Eliminar modelos de la librería | El sistema debe permitir al usuario eliminar un modelo de la librería. |
| RF-04 | Crear sesión de simulación | El sistema debe permitir el usuario iniciar una nueva sesión de simulación. |
| RF-05 | Unirse a una sesión de simulación | El sistema debe permitir al usuario unirse a una sesión de simulación existente. |
| RF-06 | Ejecutar simulación participativa | El sistema debe permitir que dos o más usuarios ejecuten las simulaciones simultáneamente en distintas ubicaciones, por medio de Internet, permitiendo a los usuarios manipular los controles de las variables existentes, afectando la ejecución para todos los usuarios en la sesión de simulación. |
| RF-07 | Visualizar información de proyectos relacionados. | El sistema debe permitir al usuario visualizar información general sobre los proyectos relacionados: Tortoise, Galapagos. |
| RF-08 | Visualizar información de ayuda | El sistema debe permitir el usuario visualizar información de ayuda sobre NodeTortoise. |

### 2.3. Requerimientos no funcionales

|  |  |  |
| --- | --- | --- |
| **ID** | **Requerimientos No Funcionales** | **Descripción breve** |
| RNF-01 | Uso de tecnologías web estándar | El sistema de funcionar utilizando tecnologías web estándar. |

# 3 Especificación de diseño

## 3.1. Integración con otros Sistemas

A continuación, se detalla la integración de NodeTortoise con otros sistemas.

### 3.1.1. Dependencias funcionales y de datos

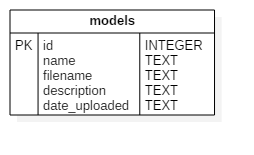
Los modelos importados a NodeTortoise, deben ser exportados a HTML desde Galapagos. No existe una interacción directa entre ambos sistemas, no obstante, NodeTortoise debe conocer como procesar el archivo HTML exportado desde Galapagos para insertar la carga de las librerías necesarias que permiten sobre escribir la lógica de la simulación para sincronizar las distintas instancias de una sesión de simulación.

### 3.1.2. Diagrama de integración



## 3.2. Diseño de datos

### 3.2.1. Diagrama detallado de la base de datos



### 3.2.2. Diagrama del modelo relacional

**Modelo**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Nombre | Nombre del Archivo | Descripción | Fecha de subida |

### 3.2.3. Descripción de los datos

|  |  |  |  |
| --- | --- | --- | --- |
| **Tabla: Modelo** | **Descripción** | | |
| **Propósito:** | Almacena los modelos importados a la librería | | |
| **Índices:** | Llave primaria: id | | |
| **Atributos:** | | | |
| ***Identificación*** | ***Tipo*** | ***Longitud*** | ***Restricciones*** |
| name | Integer |  |  |
| filename | Text |  |  |
| description | Text |  |  |
| date\_upload | Text |  |  |

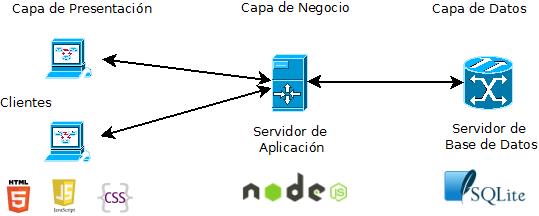
## 3.3. Diseño de la arquitectura

NodeTortoise cuenta una arquitectura cliente-servidor, comúnmente utilizada en las aplicaciones web. Está creada en NodeJS, haciendo uso del *framework* para NodeJS *Express 4*, el cual brindó una estructura inicial de archivos y configuración de la aplicación inicial, los cuales fueron modificados a conveniencia.

A nivel de arquitectura, la aplicación sigue un modelo cliente-servidor, con separación en tres capas: presentación, negocio y datos. Si bien es cierto, la aplicación se ejecuta en un mismo servidor y al usar SQLite está limitado a mantener la base de datos en el mismo servidor que la aplicación, existe una separación bien delimitada en la programación y si en un futuro se implementa un motor de base de datos más robusto, como MySQL, será posible llevar a cabo dicha separación física. Por otra parte, se implementó una capa intermedia de acceso a datos, de manera que la lógica de negocios es independiente del origen de datos.

Se utiliza el patrón de diseño Modelo-Vista-Controlador (MVC). Al estructurar NodeTortoise siguiendo este patrón, se separa los datos y la lógica de negocio de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Bajo este esquema, existe una capa de programación encargada de definir un modelo de datos y acceder al origen de estos (SQLite y WebSockets), otra capa de programación se encarga de controlar las peticiones al servidor, hacer las solicitudes necesarias al modelo de datos y responder con una vista, cuando así sea necesario. Una última capa se encarga de generar las vistas que serán enviadas al usuario, haciendo uso de un motor de plantillas (swig).

### 3.3.1. Diagrama de la arquitectura



### 3.3.2. Diseño a nivel de componentes

En esta sección se detalla la conformación de cada clase de cada componente de la aplicación.

**bl.App**

Controlador principal de la aplicación en el lado del servidor. Se encarga de establecer los parámetros de configuración, configurando el framework Express, las rutas y el motor de plantillas (templates).

|  |  |
| --- | --- |
| **Método** | **\_\_construct** |
| Requiere | N/A |
| Efectúa | Constructor de la clase. |
| Modifica | N/A |
| Pseudocódigo | * Hace inclusión de archivos necesarios. * Llama a funciones encargadas de inicializar algunas funcionalidades. |

|  |  |
| --- | --- |
| **Método** | **init** |
| Requiere | * {http.Server} server: el servidor http |
| Efectúa | * Inicializa el proceso de definición de los parámetros de configuración de la aplicación. |
| Modifica | N/A |
| Pseudocódigo | * Llama el método que inicializa el controlador principal de la aplicación. * Llama al método que inicializa los sockets. |

|  |  |
| --- | --- |
| **Método** | **setupConfig** |
| Requiere | N/A |
| Efectúa | Establece los parámetros de configuración de Express y el motor de plantillas Swig y otros (logger, cookieParser, etc). |
| Modifica | N/A |
| Pseudocódigo | * Incluye librerías necesarias como el motor de plantillas, el framework express y el motor de compresión. * Configura la aplicación, estableciendo el motor de plantillas a utilizar, la cache y el método a llamar cuanto existe una petición post que incluya archivos. |

|  |  |
| --- | --- |
| **Método** | **setupRoutes** |
| Requiere | N/A |
| Efectúa | Establece los parámetros de configuración de las rutas accesibles por medio del protocolo HTTP. |
| Modifica | N/A |
| Pseudocódigo | * Configura las rutas estáticas. * Configura las rutas generales de la aplicación. * Configura las rutas para manejar las URLs no existentes (errores 404). * Configura el manejo de errores (error 500) en el ambiente de desarrollo. * Configura el manejo de errores (error 500) en el ambiente de producción. |

|  |  |
| --- | --- |
| **Método** | **getExpress** |
| Requiere | N/A |
| Efectúa | Retorna la instancia de Express utilizada. |
| Modifica | N/A |
| Pseudocódigo | * Retorna variable de Express |

|  |  |
| --- | --- |
| **Método** | **App.getRootPath** |
| Requiere | * {String} filePath: Ruta relativa del archivo |
| Efectúa | Retorna la ruta desde el directorio raíz de la aplicación basada en la ruta recibida. |
| Modifica | N/A |
| Pseudocódigo | * Concatena la ruta absoluta al directorio raíz de la aplicación y la ruta relativa al archivo. |

|  |  |
| --- | --- |
| **Método** | **App.getPath** |
| Requiere | * {String} filePath: Ruta relativa del archivo |
| Efectúa | Retorna la ruta desde el directorio de los archivos del servidor de la aplicación basado en la ruta recibida. |
| Modifica | N/A |
| Pseudocódigo | * Concatena la ruta absoluta al directorio raíz de la aplicación, la ruta al directorio que contienes los archivos de programación relacionada con el servidor y la ruta relativa al archivo. |

|  |  |
| --- | --- |
| **Método** | **App.include** |
| Requiere | N/A |
| Efectúa | Incluye el código de un archivo dentro de la aplicación. |
| Modifica | N/A |
| Pseudocódigo | * Carga el archivo de la ruta especificada. |

|  |  |
| --- | --- |
| **Método** | **App.debug** |
| Requiere | * {String} message: El mensaje a imprimir en consola * {String} user: El usuario que está enviando el mensaje (cuando se está en modo simulación) * {String} sesión: La sesión del usuario que está enviando el mensaje (cuando se está en modo simulación) * [String] mode: El modo de depuración a aplicar |
| Efectúa | Imprime en consola un mensaje de depuración, personalizado para la aplicación. La impresión del mensaje es condicionada por el valor de la variable global CONFIG.DEBUG\_MODE en el archivo de configuración de la aplicación y el valor del parámetro modo recibido |
| Modifica | N/A |
| Pseudocódigo | * Si modo no está definido, lo establece como 1. * Si modo menor o igual al valor de CONFIG.DEBUG\_MODE, imprime mensaje. |

|  |  |
| --- | --- |
| **Método** | **App.getInstance** |
| Requiere | N/A |
| Efectúa | Retorna la instancia del objeto Express utilizado. |
| Modifica | N/A |
| Pseudocódigo | * Si el objeto Express no está definido, lo crea. * Retorna el objeto Express. |

|  |  |
| --- | --- |
| **Método** | **App.isUserMaster** |
| Requiere | * {String} userKey: La llave del usuario |
| Efectúa | Determina si el usuario es master comparando con la llave enviada por parámetro con la llave del usuario maestro. |
| Modifica | N/A |
| Pseudocódigo | * Compara la llave con la llave del maestro. |

**bl.Controller**

Controla la capa negocio de la aplicación, haciendo la conexión entre la capa de datos y las vistas y aplicando la lógica de negocio necesaria.

|  |  |
| --- | --- |
| **Método** | **Controller.message** |
| Requiere | * {Request} req: Objeto que representa la petición HTTP y contiene el “query string”, los parámetros, el cuerpo y las cabeceras HTTP, entro otros * {Response} res: El objeto res representa la respuesta HTTP que una aplicación Express envía cuando se hace una petición HTTP. * {String} title: El título de la pagina * {String} message: El mensaje |
| Efectúa | Despliega una página genérica con un mensaje. |
| Modifica | N/A |
| Pseudocódigo | * Inicializa valores. * Despliega la página. |

**bl.Controller.Model**

Controla la capa de negocio que interactúa con las acciones sobre los modelos.

|  |  |
| --- | --- |
| **Método** | **Controller.model.upload** |
| Requiere | * {Request} req: Objeto que representa la petición HTTP y contiene el “query string”, los parámetros, el cuerpo y las cabeceras HTTP, entro otros * {Response} res: El objeto res representa la respuesta HTTP que una aplicación Express envía cuando se hace una petición HTTP. * {Function} next: Función que continúa la ejecución de búsqueda de rutas. |
| Efectúa | Controla la página de carga de modelos. |
| Modifica | N/A |
| Pseudocódigo | * Inicializa valores. * Despliega la página. |

|  |  |
| --- | --- |
| **Método** | **Controller.model.upload.success** |
| Requiere | * {Request} req: Objeto que representa la petición HTTP y contiene el “query string”, los parámetros, el cuerpo y las cabeceras HTTP, entro otros * {Response} res: El objeto res representa la respuesta HTTP que una aplicación Express envía cuando se hace una petición HTTP. * {Function} next: Función que continúa la ejecución de búsqueda de rutas. |
| Efectúa | Controla la página de modelo subido exitosamente. |
| Modifica | N/A |
| Pseudocódigo | * Inicializa valores. * Despliega la página. |

|  |  |
| --- | --- |
| **Método** | **Controller.model.upload.error** |
| Requiere | * {Request} req: Objeto que representa la petición HTTP y contiene el “query string”, los parámetros, el cuerpo y las cabeceras HTTP, entro otros * {Response} res: El objeto res representa la respuesta HTTP que una aplicación Express envía cuando se hace una petición HTTP. * {Function} next: Función que continúa la ejecución de búsqueda de rutas. |
| Efectúa | Controla la página de error al subir un modelo. |
| Modifica | N/A |
| Pseudocódigo | * Inicializa valores. * Despliega la página. |

|  |  |
| --- | --- |
| **Método** | **Controller.model.onUpload** |
| Requiere | * {String} originalFile Archivo original del modelo subido. * {Request} req: Objeto que representa la petición HTTP y contiene el “query string”, los parámetros, el cuerpo y las cabeceras HTTP, entro otros * {Response} res: El objeto res representa la respuesta HTTP que una aplicación Express envía cuando se hace una petición HTTP. * {Function} next: Función que continúa la ejecución de búsqueda de rutas. |
| Efectúa | Controla la subida de un archivo de modelo. |
| Modifica | * {Request} req |
| Pseudocódigo | * Ejecuta la acción para guardar el modelo en la librería. * Modifica el parámetro de petición (req) para indicar si el proceso de guardado del archivo fue exitoso o no. |

|  |  |
| --- | --- |
| **Método** | **Controller.model.list** |
| Requiere | * {Request} req: Objeto que representa la petición HTTP y contiene el “query string”, los parámetros, el cuerpo y las cabeceras HTTP, entro otros * {Response} res: El objeto res representa la respuesta HTTP que una aplicación Express envía cuando se hace una petición HTTP. * {Function} next: Función que continúa la ejecución de búsqueda de rutas. |
| Efectúa | Controla la página de listado de modelos. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene la lista de modelos. * Ejecuta la acción correspondiente si la obtención de modelos fue exitosa o no. |

|  |  |
| --- | --- |
| **Método** | **Controller.model.delete** |
| Requiere | * {Request} req: Objeto que representa la petición HTTP y contiene el “query string”, los parámetros, el cuerpo y las cabeceras HTTP, entro otros * {Response} res: El objeto res representa la respuesta HTTP que una aplicación Express envía cuando se hace una petición HTTP. * {Function} next: Función que continúa la ejecución de búsqueda de rutas. |
| Efectúa | Controla el borrado de un modelo. |
| Modifica | N/A |
| Pseudocódigo | * Elimina el modelo. * Ejecuta la acción correspondiente si el borrado del modelo fue exitoso o no. |

**bl.Controller.session**

Controla la capa de negocio que interactúa con las sesiones de simulaciones.

|  |  |
| --- | --- |
| **Método** | **Controller.session.list** |
| Requiere | * {Request} req: Objeto que representa la petición HTTP y contiene el “query string”, los parámetros, el cuerpo y las cabeceras HTTP, entro otros * {Response} res: El objeto res representa la respuesta HTTP que una aplicación Express envía cuando se hace una petición HTTP. * {Function} next: Función que continúa la ejecución de búsqueda de rutas. |
| Efectúa | Controla la página de listado de sesiones de simulación. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene la lista de sesiones * Por cada lista en la sesión que tenga más de un usuario, determina el URL de la sesión. * Despliega la página. |

**bl.Model**

Controla las acciones que se pueden ejecutar sobre un modelo.

|  |  |
| --- | --- |
| **Método** | **add** |
| Requiere | * {String} originalFile: Ruta al archivo original subido. * {String} modelName: Nombre del modelo. * {String} description: Texto descriptivo del modelo. |
| Efectúa | Agrega un modelo al sistema. Retorna el resultado de la operación. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene la ruta donde se almacenará el archivo HTML. * Obtiene la ruta donde se almacenará el archivo JavaScript. * Llama a la función encargada de procesar el archivo del modelo HTML original. * Llama a la función encargada de guardar el modelo en base de datos. * Devuelve el resultado de la operación. |

|  |  |
| --- | --- |
| **Método** | **getByID** |
| Requiere | * {Integer} idModel :ID del modelo. * {Function} onSuccess: Función a ejecutar cuando se obtiene el modelo. * {Function} onError: Función a ejecutar en caso de error. |
| Efectúa | Obtiene un modelo por ID de modelo. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene el modelo por ID. * Retorna el modelo. |

|  |  |
| --- | --- |
| **Método** | **getByFilename** |
| Requiere | * {Integer} filename: Nombre del archivo. * {Function} onSuccess: Función a ejecutar cuando se obtiene el modelo. * {Function} onError: Función a ejecutar en caso de error. |
| Efectúa | Obtiene un modelo por nombre de archivo. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene el modelo por nombre de archivo. * Retorna el modelo. |

|  |  |
| --- | --- |
| **Método** | **list** |
| Requiere | * {Function} onSuccess: Función a ejecutar cuando se ha recuperado la lista de modelos del sistema * {Function} onError: Función e ejecutar si hay un error al recuperar la lista de modelos del sistema |
| Efectúa | Retorna la lista de modelos disponibles en el sistema. |
| Modifica | N/A |
| Pseudocódigo | * Llama a la función encargada de obtener lista de modelos de base de datos. * Retorna la lista de modelos. |

|  |  |
| --- | --- |
| **Método** | **delete** |
| Requiere | * {Integer} idModel: El ID del modelo. * {String} filename: Nombre de archivo del modelo (sin la ruta). * {Function} onSuccess: Función a ejecutar cuando el modelo sea eliminado correctamente del sistema. * {Function} onError: Función e ejecutar si hay un error al eliminar un modelo del sistema. |
| Efectúa | Elimina un modelo del sistema. |
| Modifica | N/A |
| Pseudocódigo | * Define las acciones a ejecutar cuando se elimina el modelo. * Llama la función encargada de eliminar el modelo de base de datos. |

|  |  |
| --- | --- |
| **Método** | **getHTMLFile** |
| Requiere | {String} fileName: Nombre del archivo de modelo a recuperar. |
| Efectúa | Retorna la ruta al archivo HTML de un modelo. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene la ruta absoluta al archivo. * Retorna la ruta. |

|  |  |
| --- | --- |
| **Método** | **getJSFile** |
| Requiere | * {String} fileName: Nombre del archivo de modelo a recuperar. |
| Efectúa | Retorna la ruta al archivo JavaScript de un modelo. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene la ruta absoluta al archivo. * Retorna la ruta. |

|  |  |
| --- | --- |
| **Método** | **tryDeleteFile** |
| Requiere | * {String} filePath: Ruta del archivo. * {String} fileStrID: Identificador del archivo a ser mostrado en el mensaje de depuración. * {fs} fs: Objeto que controla el sistema de archivos. |
| Efectúa | Intenta eliminar un archivo del sistema de archivo, si falla, envía un mensaje de depuración a la aplicación. |
| Modifica | N/A |
| Pseudocódigo | * Intenta eliminar el archivo. * Muestra mensaje de debug dependiendo del resultado de eliminar el archivo. * Devuelve el resultado de la operación. |

|  |  |
| --- | --- |
| **Método** | **parse** |
| Requiere | * {String} originalFilePath: Ruta del archivo original, subido al sistema. * {String} htmlNewFilePath: Ruta donde se debe guardar el archivo HTML del modelo. * {String} jsNewFilePath: Ruta donde se debe guardar el archivo JavaScript del modelo. * {String} fileName: Nombre del archivo del modelo. * {String} modelName: Nombre del modelo. * {fs} fs: Objeto que controla el sistema de archivos. |
| Efectúa | Procesa un modelo exportado de Galapagos, para convertirlo al formato necesario para que funcione en NodeTortoise. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene el string del código del archivo original del modelo. * Obtiene el string de la plantilla de archivo de modelo utilizado por NodeTortoise. * Convierte el string del archivo original a elementos HTML. * Obtiene los estilos del modelo original. * Obtiene el código de script del modelo original. * Obtiene el contenedor del modelo original. * Inserta el código JavaScript a ejecutar antes del código de inicio de sesión el archivo original. * Forma el código de inclusión del archivo JavaScript del modelo. * Reemplaza en el string de plantilla los estilos, la código de inclusión del archivo JavaScript del modelo, el título y el contenido. * Escribe el nuevo archivo de JavaScript del modelo. * Escribe el nuevo archivo de HTML del modelo. |

|  |  |
| --- | --- |
| **Método** | **Model.getInstance** |
| Requiere | N/A |
| Efectúa | Basado en el patrón “Singleton”, retorna una instancia del objeto Model. |
| Modifica | N/A |
| Pseudocódigo | * Verifica si existe el objeto y si no existe, lo crea. * Retorna el objeto. |

**bl.Session**

Maneja una sesión individual de una simulación.

|  |  |
| --- | --- |
| **Método** | **init** |
| Requiere | * {String} modelFile: El archivo del modelo * {String} modelName: El nombre del modelo * {String} masterName: El nombre del usuario maestro |
| Efectúa | Realiza el proceso de inicialización de variables necesarias por el objeto. |
| Modifica | N/A |
| Pseudocódigo | * Inicializa variables. * Guarda en una variable de objeto la información sobre la sesión. |

|  |  |
| --- | --- |
| **Método** | **addUser** |
| Requiere | * {String} username: El nombre de usuario. * {String} masterKey: La llave del maestro. * {Boolean} enabledControls: Indica si se habilitan los controles a todos los usuarios. |
| Efectúa | Agrega un usuario a la sesión. |
| Modifica | N/A |
| Pseudocódigo | * Verifica que el nombre de usuario no existe. Si ya existe, agrega un aleatorio al final de este para diferenciarlo del ya existente. * Obtiene el identificador de usuario. * Si es un usuario maestro:   + Establece que el usuario maestro está en línea.   + Obtiene el identificador de usuario maestro.   + Guarda el usuario en la lista de usuarios de la sesión. * Si no es usuario maestro:   + Guarda el usuario en la lista de usuarios de la sesión. * Determina si el usuario maestro está en línea. * Devuelve un objeto con la información pertinente. |

|  |  |
| --- | --- |
| **Método** | **removeUser** |
| Requiere | * {String} userName: El nombre del usuario. * {String} token: El token de sesión del usuario. |
| Efectúa | Elimina un usuario de una sesión de simulación. |
| Modifica | N/A |
| Pseudocódigo | * Si el token del usuario coincide con el del usuario maestro, eliminar la información del usuario maestro de la sesión. * Si no existe información de usuario maestro, establece que el usuario maestro ya no está en línea. * Elimina el usuario de la lista de usuarios de la sesión. |

|  |  |
| --- | --- |
| **Método** | **getMasterToken** |
| Requiere | N/A |
| Efectúa | Retorna el token del usuario maestro. |
| Modifica | N/A |
| Pseudocódigo | * Retorna el token del usuario maestro. |

|  |  |
| --- | --- |
| **Método** | **updateLastActivityDate** |
| Requiere | N/A |
| Efectúa | Actualiza la fecha de última actividad. |
| Modifica | N/A |
| Pseudocódigo | * Actualiza la fecha de última actividad. |

|  |  |
| --- | --- |
| **Método** | **getSessionInfo** |
| Requiere | N/A |
| Efectúa | Retorna la información de la sesión. |
| Modifica | N/A |
| Pseudocódigo | * Retorna la información de la sesión. |

|  |  |
| --- | --- |
| **Método** | **getUsersQuantity** |
| Requiere | N/A |
| Efectúa | Retorna la cantidad de usuarios en una sesión. |
| Modifica | N/A |
| Pseudocódigo | * Retorna la cantidad de usuarios en una sesión. |

**bl.SessionController**

Controla las sesiones de creadas por los usuarios para ejecutar las simulaciones.

|  |  |
| --- | --- |
| **Método** | **init** |
| Requiere | N/A |
| Efectúa | Realiza el proceso de inicialización de variables necesarias por el objeto. |
| Modifica | N/A |
| Pseudocódigo | * Inicializa variables * Establece que se ejecute la actividad de las sesiones cada cierto tiempo. |

|  |  |
| --- | --- |
| **Método** | **joinSession** |
| Requiere | * {String} sessionName: El nombre de la sesión. * {String} userName: El nombre del usuario. * {String} modelFile: El archivo del modelo de la simulación que ejecutará la sesión. * {String} modelName: El nombre del modelo de la simulación que ejecutará la sesión. * {Boolean} [enabledControls]: Determinar si la sesión habilita o no los controles a otros usuarios distintos del maestro. Parámetro obligatorio solo para el usuario maestro. |
| Efectúa | Añade un usuario a una sesión de simulación. Si la sesión no existe, crea una nueva. |
| Modifica | N/A |
| Pseudocódigo | * Si la sesión no existe:   + Crea una nueva sesión.   + Inicializa la sesión.   + Retorna el resultado de agregar el usuario. * Si la sesión existe:   + Retorna el resultado de agregar el usuario. |

|  |  |
| --- | --- |
| **Método** | **leaveSession** |
| Requiere | * {String} sessionName: El nombre de la sesión. * {String} userName: El nombre del usuario. * {String} token: El token de sesión del usuario. |
| Efectúa | Elimina un usuario de una sesión de simulación. |
| Modifica | N/A |
| Pseudocódigo | * Si la sesión existe:   + Llama a la función encargada a remover el usuario de la sesión.   + Si el usuario maestro ya no está en línea y la cantidad de usuarios es menor igual a cero, elimina la sesión de la lista de sesiones. * Retorna el resultado de la operación. |

|  |  |
| --- | --- |
| **Método** | **getSessionModel** |
| Requiere | * {String} sessionName: El nombre de la sesión |
| Efectúa | Devuelve el modelo que se está utilizando en una sesión. |
| Modifica | N/A |
| Pseudocódigo | * Retorna el modelo que se está utilizando en la sesión indicada. |

|  |  |
| --- | --- |
| **Método** | **getSessionList** |
| Requiere | N/A |
| Efectúa | Retorna la lista de las sesiones de simulación activas. |
| Modifica | N/A |
| Pseudocódigo | * Retorna la lista de las sesiones de simulación activas. |

|  |  |
| --- | --- |
| **Método** | **getSession** |
| Requiere | * {String} sessionName: Nombre de la sesión |
| Efectúa | Retorna una sesión, basado en el nombre recibido como parámetro. |
| Modifica | N/A |
| Pseudocódigo | * Retorna el objeto de la sesión indicada. |

|  |  |
| --- | --- |
| **Método** | **getSessionUsersQuantity** |
| Requiere | * {String} sessionName: El nombre de la sesión |
| Efectúa | Retorna la cantidad de usuarios en una sesión, basado en el nombre recibido como parámetro. |
| Modifica | N/A |
| Pseudocódigo | * Si la sesión existe, retorna el número de usuarios en la sesión. |

|  |  |
| --- | --- |
| **Método** | **updateActivityDate** |
| Requiere | * {String} sessionName: El nombre de la sesión |
| Efectúa | Actualiza la fecha de última actividad. |
| Modifica | N/A |
| Pseudocódigo | * Si la sesión existe, actualiza la fecha de última actividad. |

|  |  |
| --- | --- |
| **Método** | **checkSessionsActivity** |
| Requiere | N/A |
| Efectúa | Revisa la actividad de las sesiones. Si alguna tiene más de cierta cantidad de minutos sin actividad, la elimina. |
| Modifica | N/A |
| Pseudocódigo | * Por cada sesión en la lista de sesiones: * Obtiene la información de la sesión. * Elimina la sesión si no existe información sobre la sesión, si la cantidad de usuarios es menor que uno o después de cierta cantidad de tiempo sin actividad. |

|  |  |
| --- | --- |
| **Método** | **\_\_construct** |
| Requiere | N/A |
| Efectúa | Constructor de la clase |
| Modifica | N/A |
| Pseudocódigo | * Incluye clases necesarias. |

|  |  |
| --- | --- |
| **Método** | **SessionController.getInstance** |
| Requiere | N/A |
| Efectúa | Basado en el patrón “Singleton”, retorna una instancia del objeto SessionController. |
| Modifica | N/A |
| Pseudocódigo | * Si el objeto no existe, lo crea. * Retorna el objeto. |

**bl.Simulation**

Maneja las acciones a ejecutar entre las simulaciones.

|  |  |
| --- | --- |
| **Método** | **Simulation.connect** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} token: Token del usuario. * {Object} params: Parámetros de la acción. |
| Efectúa | Conecta una nueva instancia a una sesión de simulación. |
| Modifica | N/A |
| Pseudocódigo | * Agrega el usuario a la sesión * Envía mensaje de actualizar usuarios a todos los usuarios. * Envía respuesta de conexión el usuario que se conectó. * Si el usuario es maestro, envía señal de “Inicio” a todos los usuarios. * Actualiza la fecha de última actividad de la sesión. |

|  |  |
| --- | --- |
| **Método** | **Simulation.disconnect** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} token: Token del usuario. * {Object} params: Parámetros de la acción. |
| Efectúa | Desconecta al dueño del socket actual de una sesión. |
| Modifica | N/A |
| Pseudocódigo | * Llama la función encargada de desconectar al usuario de la sesión. * Si el resultado de la desconexión de la sesión es que la sesión murió, envía un mensaje a todos los usuarios, indicando que la sesión ha terminado. * Desconecta el socket. * Envía mensaje a los demás usuarios para que actualicen la lista de usuarios conectados. * Actualiza la fecha de última actividad de la sesión. |

|  |  |
| --- | --- |
| **Método** | **Simulation.executeCommand** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} token: Token del usuario. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje de “Ejecutar Comando”, para que los clientes ejecuten un comando, basado en los parámetros enviados. |
| Modifica | N/A |
| Pseudocódigo | * Envía mensaje de “Ejecutar Comando” a los demás usuarios. * Actualiza la fecha de última actividad de la sesión. |

|  |  |
| --- | --- |
| **Método** | **Simulation.setControl** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} token: Token del usuario. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje de “Establecer valor de control” basado en los parámetros enviados. |
| Modifica | N/A |
| Pseudocódigo | * Si está habilitado el control a todos los usuarios, envía un mensaje de “Establecer control” a los demás usuarios. * Si no está habilitado el control a todos los usuarios, envía un mensaje de “Establecer control” a todos los usuarios. |

|  |  |
| --- | --- |
| **Método** | **Simulation.updateSpeed** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} token: Token del usuario. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje de “Actualizar Velocidad”, para que los clientes ejecuten una actualización de la velocidad de la simulación, basado en los parámetros enviados. |
| Modifica | N/A |
| Pseudocódigo | * Si está habilitado el control a todos los usuarios, envía un mensaje de “Establecer control” a los demás usuarios. * Si no está habilitado el control a todos los usuarios, envía un mensaje de “Establecer control” a todos los usuarios. |

|  |  |
| --- | --- |
| **Método** | **Simulation.applyUpdate** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} token: Token del usuario. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje de “Aplicar actualización”, para que los clientes ejecuten una actualización de la simulación, basado en los parámetros enviados. |
| Modifica | N/A |
| Pseudocódigo | * Envía mensaje de “Aplicar actualización” a los demás usuarios. |

|  |  |
| --- | --- |
| **Método** | **Simulation.setStatus** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} token: Token del usuario. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje de “Establecer status” basado en los parámetros enviados. |
| Modifica | N/A |
| Pseudocódigo | * Envía mensaje de “Establecer status” a los demás usuarios. |

|  |  |
| --- | --- |
| **Método** | **Simulation.sendMessage** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje de “Enviar Mensaje” |
| Modifica | N/A |
| Pseudocódigo | * Envía mensaje de “Aplicar actualización” a los demás usuarios. |

**da.Entities.Model**

Maneja la relación de la entidad “model” con la base de datos.

|  |  |
| --- | --- |
| **Método** | **add** |
| Requiere | * {String} name: Nombre del modelo. * {String} filename: Nombre del archivo del modelo sin extensión. * {String} description: Descripción del modelo. |
| Efectúa |  |
| Modifica | N/A |
| Pseudocódigo | * Obtiene la fecha actual para establecer la fecha de subida. * Ejecuta la función que inserta en base de datos la información. |

|  |  |
| --- | --- |
| **Método** | **getByID** |
| Requiere | * {Integer} idModel: ID del modelo. * {Function} onSuccess: Función a ejecutar cuando se obtiene el modelo. * {Function} onError: Función a ejecutar en caso de error. |
| Efectúa | Obtiene un modelo por ID de modelo. |
| Modifica | N/A |
| Pseudocódigo | * Ejecuta la función que selecciona el modelo desde la base de datos. |

|  |  |
| --- | --- |
| **Método** | **getByFilename** |
| Requiere | * {Integer} filename: Nombre del archivo. * {Function} onSuccess: Función a ejecutar cuando se obtiene el modelo. * {Function} onError: Función a ejecutar en caso de error. |
| Efectúa | Obtiene un modelo por nombre de archivo. |
| Modifica | N/A |
| Pseudocódigo | * Ejecuta la función que selecciona el modelo desde la base de datos. |

|  |  |
| --- | --- |
| **Método** | **list** |
| Requiere | * {Function} onSuccess: Función a ejecutar cuando se obtiene la lista de modelos. * {Function} onError: Función a ejecutar en caso de error. |
| Efectúa | Obtiene la lista de modelos en base de datos. |
| Modifica | N/A |
| Pseudocódigo | * Ejecuta la función que selecciona los modelos desde la base de datos. |

|  |  |
| --- | --- |
| **Método** | **delete** |
| Requiere | * {Integer} idModel: ID del modelo. * {Function} onSuccess: Función a ejecutar cuando se elimina el modelo. * {Function} onError: Función a ejecutar en caso de error. |
| Efectúa | Elimina un modelo de base de datos. |
| Modifica | N/A |
| Pseudocódigo | * Ejecuta la función que elimina el modelo desde la base de datos. |

|  |  |
| --- | --- |
| **Método** | **Entities.getModel** |
| Requiere | N/A |
| Efectúa | Basado en el patrón “Singleton”, retorna una instancia del objeto Entities.model. |
| Modifica | N/A |
| Pseudocódigo | * Si el objeto no existe, lo crea. * Retorna el objeto. |

**da.Sockets**

Inicializa y configura los web sockets que se utilizan para el envío de la información en el ambiente de simulación de los modelos.

|  |  |
| --- | --- |
| **Método** | **Sockets.sendAction** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} sessionName: Nombre de la sesión de simulación a la cual se va a enviar el mensaje. * {String} action: Acción a ejecutar. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía a los demás la indicación de que se ejecute una acción. |
| Modifica | N/A |
| Pseudocódigo | * Si el token del usuario es igual al token del usuario maestro, envía el mensaje a los demás usuarios. |

|  |  |
| --- | --- |
| **Método** | **Sockets.set** |
| Requiere | * {Socket} socket: El web socket para la instancia actual |
| Efectúa | Alias de Sockets.setServerActions. |
| Modifica | N/A |
| Pseudocódigo | * Llama la función “Sockets.setServerActions”. |

|  |  |
| --- | --- |
| **Método** | **Sockets.setServerActions** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. |
| Efectúa | Configura las acciones que se pueden ejecutar sobre el socket actual. |
| Modifica | N/A |
| Pseudocódigo | * Configura la acción de desconexión para el socket actual. * Por cada acción en la clase “Simulation”, configura la acción para el socket actual. |

|  |  |
| --- | --- |
| **Método** | **Sockets.sendToMe** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} action: Acción a ejecutar. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje al dueño del socket actual. |
| Modifica | N/A |
| Pseudocódigo | * Envía un mensaje al socket actual. |

|  |  |
| --- | --- |
| **Método** | **Sockets.sendToOthers** |
| Requiere | * {Socket} socket: El web socket para la instancia actual. * {String} sessionName: Nombre de la sesión. * {String} action: Acción a ejecutar. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje de acción a los otros usuarios dentro de la sesión del dueño del socket actual. |
| Modifica | N/A |
| Pseudocódigo | * Envía un mensaje a los sockets distintos al actual en el grupo. |

|  |  |
| --- | --- |
| **Método** | **Sockets.sendToAll** |
| Requiere | * {String} sessionName: Nombre de la sesión. * {String} action: Acción a ejecutar. * {Object} params: Parámetros de la acción. |
| Efectúa | Envía un mensaje de acción a todos los usuarios dentro la sesión del dueño del socket actual. |
| Modifica | N/A |
| Pseudocódigo | * Envía un mensaje a todos los sockets en el grupo. |

|  |  |
| --- | --- |
| **Método** | **Sockets.init** |
| Requiere | * {http.Server} server: Instancia del servidor HTTP |
| Efectúa | Configura la funcionalidad de los Sockets. |
| Modifica | N/A |
| Pseudocódigo | * Ante la conexión de un socket, llama a la función que configura las acciones para dicho socket. |

**libs.SQLite**

Ejecuta acciones sobre una base de SQLite.

|  |  |
| --- | --- |
| **Método** | **insert** |
| Requiere | * {String} table: La tabla sobre la cual se ejecutará la acción. * {String} fields: Las columnas a insertar. * {Array} values: Los valores a insertar. * {Function} onSuccess: Función a ejecutar cuando el proceso en base de datos se completa exitosamente. * {Funtion} onError: Función a ejecutar en caso de error. |
| Efectúa | Inserta un registro en la base de datos. |
| Modifica | N/A |
| Pseudocódigo | * Llama la función que procesa los valores para la consulta a base de datos. * Ejecuta la acción en base de datos. |

|  |  |
| --- | --- |
| **Método** | **select** |
| Requiere | * {String} table: La tabla sobre la cual se ejecutará la acción. * {Array} filters: Los filtros de la consulta (“where”). * [String] fields: Las columnas a seleccionar. * {Function} onSuccess: Función a ejecutar cuando el proceso en base de datos se completa exitosamente. * {Funtion} onError: Función a ejecutar en caso de error. |
| Efectúa | Selecciona los registros de una tabla, basado en los filtros obtenidos. |
| Modifica | N/A |
| Pseudocódigo | * Si no existe los campos de selección, establece el valor para todos (“\*”). * Llama la función que procesa los filtros para la consulta. * Ejecuta la acción en base de datos. |

|  |  |
| --- | --- |
| **Método** | **selectAll** |
| Requiere | * {String} table: La tabla sobre la cual se ejecutará la acción. * [String] fields: Las columnas a seleccionar. * {Function} onSuccess: Función a ejecutar cuando el proceso en base de datos se completa exitosamente. * {Funtion} onError: Función a ejecutar en caso de error. |
| Efectúa | Selecciona todos los registros de una tabla. |
| Modifica | N/A |
| Pseudocódigo | * Si no existe los campos de selección, establece el valor para todos (“\*”). * Ejecuta la acción en base de datos. |

|  |  |
| --- | --- |
| **Método** | **delete** |
| Requiere | * {String} table: La tabla sobre la cual se ejecutará la acción * {Array} filters: Los filtros de la consulta (<i>where</i>) * {Function} onSuccess: Función a ejecutar cuando el proceso en base de datos se completa exitosamente * {Funtion} onError: Función a ejecutar en caso de error |
| Efectúa | Elimina uno o más registros de una tabla. |
| Modifica | N/A |
| Pseudocódigo | * Llama la función que procesa los filtros para la consulta. * Ejecuta la acción en base de datos. |

|  |  |
| --- | --- |
| **Método** | **onCallback** |
| Requiere | * {String} query: La consulta previamente ejecutada. * {Function} onSuccess: Función a ejecutar cuando el proceso en base de datos se completa exitosamente. * {Funtion} onError: Función a ejecutar en caso de error. * {mixed} error: El error o mensaje de error. * {mixed} result: El resultado. |
| Efectúa | Ejecuta el “callback” indicado después de realizar una tarea de base de datos, ya sea este de proceso exitoso o de error. |
| Modifica | N/A |
| Pseudocódigo | * Si existe error, ejecuta la acción de error. * Si no existe error, ejecuta la acción de éxito. |

|  |  |
| --- | --- |
| **Método** | **getQueryFilters** |
| Requiere | * {Array} filters: Los filtros a procesar |
| Efectúa | Retorna los valores necesarios para armar una consulta parametrizada con “where”. |
| Modifica | N/A |
| Pseudocódigo | * Por cada filtro, forma y concatena la el string de “where”. |

|  |  |
| --- | --- |
| **Método** | **getQueryValues** |
| Requiere | * {Array} values: Los valores a procesar |
| Efectúa | Retorna los valores necesarios para armar una consulta parametrizada con “values” (ejemplo: ?, ?, ?). |
| Modifica | N/A |
| Pseudocódigo | * Por cada valor, forma el string correspondiente. |

|  |  |
| --- | --- |
| **Método** | **getDB** |
| Requiere | N/A |
| Efectúa | Retorna la instancia de SQLite3 utilizada |
| Modifica | N/A |
| Pseudocódigo | * Retorna la instancia correspondiente. |

|  |  |
| --- | --- |
| **Método** | **debug** |
| Requiere | * {String} message: Mensaje de error |
| Efectúa | Imprime en consola un mensaje de error, si el atributo “debug” del objeto es verdadero. |
| Modifica | N/A |
| Pseudocódigo | * Verifica si el valor de la variables “debug” es verdadero e imprime el mensaje. |

|  |  |
| --- | --- |
| **Método** | **\_\_construct** |
| Requiere | N/A |
| Efectúa | Constructor de la clase. |
| Modifica | N/A |
| Pseudocódigo | * Crea e inicializa el componente de “sqlite3”. |

**public.Model\_List**

Controla las acciones que se deben ejecutar en la página de Lista de Modelos.

|  |  |
| --- | --- |
| **Método** | **init** |
| Requiere | N/A |
| Efectúa | Ejecuta las acciones de configuración necesarias. |
| Modifica | N/A |
| Pseudocódigo | * Llama a la función que inicializa la tabla de modelos. * Configura el evento click de los botones de “Eliminar” de los modelos. * Configura las acciones a ejecutar cuando se oculta el modal de eliminar modelo. * Configura las acciones a ejecutar cuando se abre el modal de eliminar modelo. |

|  |  |
| --- | --- |
| **Método** | **initModelsTable** |
| Requiere | N/A |
| Efectúa | Inicializa la tabla que contiene la lista de modelos. |
| Modifica | N/A |
| Pseudocódigo | * Configura el componente “DataTable”. * Configura las acciones a ejecutar cuando se hace clic en eliminar de un modelo. |

|  |  |
| --- | --- |
| **Método** | **deleteModel** |
| Requiere | N/A |
| Efectúa | Configura la acción de borrar modelo. |
| Modifica | N/A |
| Pseudocódigo | * Configura las acciones de éxito y error al eliminar un modelo. * Configura la acción “ajax” que realiza la petición de eliminar un modelo. |

|  |  |
| --- | --- |
| **Método** | **Model\_List.init** |
| Requiere | N/A |
| Efectúa | Crea e inicializa el objeto Model\_List. |
| Modifica | N/A |
| Pseudocódigo | * Crea un objeto Model\_List. * Inicializa el objeto. |

**public.Session\_List**

Controla las acciones que se deben ejecutar en la página de Lista de Sesiones.

|  |  |
| --- | --- |
| **Método** | **init** |
| Requiere | N/A |
| Efectúa | Ejecuta las acciones de configuración necesarias. |
| Modifica | N/A |
| Pseudocódigo | * Configura la acción “Nueva sesión”. * Configura la acción “Cancelar nueva sesión”. * Configura la acción “Recargar lista de sesiones”. |

|  |  |
| --- | --- |
| **Método** | **reloadSessionsData** |
| Requiere | N/A |
| Efectúa | Recarga las tablas de lista de sesiones y lista de modelos. |
| Modifica | N/A |
| Pseudocódigo | * Configura la carga de la lista de sesiones y lista de modelos vía “ajax”. |

|  |  |
| --- | --- |
| **Método** | **initDataTables** |
| Requiere | N/A |
| Efectúa | Inicializa la tabla que contiene la lista de modelos. |
| Modifica | N/A |
| Pseudocódigo | * Configura el componente “DataTable” para la lista de modelos. * Configura el componente “DataTable” para la lista de sesiones. |

|  |  |
| --- | --- |
| **Método** | **Session\_List.init** |
| Requiere | N/A |
| Efectúa | Crea e inicializa el objeto Session\_List. |
| Modifica | N/A |
| Pseudocódigo | * Crea un objeto Session\_List. * Inicializa el objeto. |

**public.Simulation**

Controla las acciones que se deben ejecutar en la página de Lista de Modelos.

|  |  |
| --- | --- |
| **Método** | **init** |
| Requiere | N/A |
| Efectúa | Inicializa la clase, realizando las configuraciones necesarias. |
| Modifica | N/A |
| Pseudocódigo | * Inicializa variables * Inicializa la ejecución cada cierto tiempo de la función que despliega el estado de la simulación. * Llama a funciones que inicializan componentes (widgets, controles, sockets, outputs). * Obtiene los parámetros de la URL. |

|  |  |
| --- | --- |
| **Método** | **connect** |
| Requiere | N/A |
| Efectúa | Realiza la conexión del cliente con el servidor por medio de “web sockets”. |
| Modifica | N/A |
| Pseudocódigo | * Establece la opción de habilitar o no los controles. * Realiza la conexión al servidor por medio de web sockets. |

|  |  |
| --- | --- |
| **Método** | **setCheckStatus** |
| Requiere | N/A |
| Efectúa | Establece la función que se ejecuta cada cierto tiempo, para determinar el estado de la simulación. |
| Modifica | N/A |
| Pseudocódigo | * Si es usuario maestro, se configura función que se ejecuta cada cierto tiempo, realizando las siguientes actividades:   + Si el selector del comando “go” está activo, se asigna el estado como “Ejecutando”.   + Si el usuario la simulación ya hizo “setup” y la sesión no ha terminado, se asignado el estado como “Detenido”.   + Si ninguna de las anteriores, se asignado el estado como “Listo”. |

|  |  |
| --- | --- |
| **Método** | **displayStatus** |
| Requiere | N/A |
| Efectúa | Establece el mensaje del estatus actual de la simulación. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene el mensaje de estado, dependiendo del estado actual de la simulación * Despliega el mensaje en el elemento correspondiente. |

|  |  |
| --- | --- |
| **Método** | **applyUpdate** |
| Requiere | * {Object} agentStreamController: El objeto AgentStreamController del modelo. * {Object} modelUpdate: El objeto que contiene las instrucciones de actualización. |
| Efectúa | Este método sobre-escribe el método "AgentStreamController.prototype.applyUpdate" del código original de Tortoise. Se encarga de controlar la actualización de la simulación (a nivel gráfico) y de los valores de los "outputs". En el caso del cliente maestro, se envía un mensaje al servidor para que los clientes realicen la actualización y finalmente se ejecuta el código original de actualización. En el caso los clientes que no son maestros, se ignora cualquier proceso. En caso de que aún no se haya realizado la conexión con el socket, se ejecuta el código original de actualización, en ambos casos. |
| Modifica | N/A |
| Pseudocódigo | * Si el socket aún no está listo, ejecuta la función de actualización nativa del modelo. * Si es usuario maestro, envía mensaje de actualización al servidor y ejecuta función de actualización nativa del modelo. * Si no se cumple ninguna de las anteriores, no hace nada. |

|  |  |
| --- | --- |
| **Método** | **applyUpdate\_** |
| Requiere | * {Object} modelUpdate: El objeto que contiene las instrucciones de actualización. |
| Efectúa | Código ejecutado por los clientes no maestros al recibir un mensaje de actualización del modelo. Ejecuta las actualizaciones del modelo, haciendo la llamada al código de actualización original de "Tortoise" y llevando a cabo la actualización de los "outputs". |
| Modifica | N/A |
| Pseudocódigo | * Ejecuta la función de actualización nativa del modelo. |

|  |  |
| --- | --- |
| **Método** | **updateSpeed\_** |
| Requiere | * {Integer} value: Valor de la velocidad. |
| Efectúa | Establece el valor de la velocidad de la simulación. |
| Modifica | N/A |
| Pseudocódigo | * Establece el valor de la velocidad en el elemento correspondiente. |

|  |  |
| --- | --- |
| **Método** | **setStatus** |
| Requiere | * {Integer} status: El estatus de la simulación. |
| Efectúa | Ejecuta acciones relacionadas con el estatus de la simulación. |
| Modifica | N/A |
| Pseudocódigo | * Si el estado es “Inicializando”:   + Deshabilita los controles. * Si el estado es “Listo”:   + Si es usuario maestro o está habilitada la opción de los controles para todos los usuarios, habilita los controles. * Si el estado es “Detenido”:   + Si no se ha realizado el “setup”, llama la función que ejecuta funciones cuando la sesión no ha iniciado.   + Si es usuario maestro o está habilitada la opción de los controles para todos los usuarios, habilita los controles. * Si el estado es “Ejecutando”:   + Si no se ha realizado el “setup”, llama la función que ejecuta funciones cuando la sesión no ha iniciado.   + Deshabilita los controles. * Si el estado es “Terminado”:   + Deshabilita los controles. * Si no es ninguna de las anteriores:   + Deshabilita los controles. |

|  |  |
| --- | --- |
| **Método** | **isCommandWidget** |
| Requiere | * {Object} widget: El widget. |
| Efectúa | Determina si un widget es de tipo comando. |
| Modifica | N/A |
| Pseudocódigo | * Devuelve verdadero si el widget contiene los atributo compiledSource, buttonType y el valor de buttonType es “'OBSERVER'”. |

|  |  |
| --- | --- |
| **Método** | **initWidgets** |
| Requiere | N/A |
| Efectúa | Inicializa los widgets del modelo, para:  1. Obtener los widget de tipo comando para sobre-escribir su funcionalidad  2. Obtener y guardar los widget de tipo control para posteriormente sobre-escribir su funcionalidad. |
| Modifica | N/A |
| Pseudocódigo | * Por cada widget:   + Si es un widget de tipo “comando”,     - Extrae los datos del widget.     - Sobreescribe la funcionalidad del widget.     - Agrega la nueva funcionalidad del widget a la lista de comandos del modelo.   + Si el widget contiene los atributos “varName” o “source”, agrega dicho atributos a la lista de controles del modelo. |

|  |  |
| --- | --- |
| **Método** | **parseCommandWidget** |
| Requiere | * {Object} widgetData: Los datos del “widget”. |
| Efectúa | Extrae de un “widget” de tipo comando la información necesaria para su posterior procesamiento. |
| Modifica | N/A |
| Pseudocódigo | * Elimina del texto la cadena “call”. * Extrae los parámetros. * Extrae el nombre de la función. * Recorre el nombre de la función, para extraer la jerarquía de clases y finalmente la función invocadas. * Une toda la información extraída en un solo objeto. * Retorna el objeto. |

|  |  |
| --- | --- |
| **Método** | **overwriteCommand** |
| Requiere | * {Object} commandData: Los datos del comando. |
| Efectúa | Sobre-escribe la funcionalidad de un comando. |
| Modifica | N/A |
| Pseudocódigo | * Si existe funcionalidad extra configurada para el comando, determinar dicha función, como funcionalidad extra a ejecutar. Si no, será una función vacía. * Crea la nueva función del comando, que ejecutará la función extra y el la función original del comando. * Sobrescribe la función original del comando, donde, si es un usuario maestro, envía al servidor mensaje de “ejecutar comando” y luego ejecuta la función original. |

|  |  |
| --- | --- |
| **Método** | **overwriteControls** |
| Requiere | N/A |
| Efectúa | Agrega funcionalidad a los controles de la simulación. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene el nombre del control desde la etiqueta que lo antecede. * Si existe la etiqueta, agrega al control un evento que se ejecuta cada vez que hay cambio, donde se llama a la función encargada de enviar el valor del control al servidor. * Para el control de velocidad, si crea un evento que se ejecuta cada vez que el valor de este cambia, donde se envía al servidor un mensaje de “actualice velocidad”, si es usuario maestro o está habilitada la manipulación de los controles para todos los usuarios. |

|  |  |
| --- | --- |
| **Método** | **sendInputValue** |
| Requiere | * {String} name: El nombre del control. |
| Efectúa | Envía al servidor al valor de un control. |
| Modifica | N/A |
| Pseudocódigo | * Si es usuario maestro o está habilitada la manipulación de los controles para todos los usuarios, envía el servidor un mensaje de “Ajustar Control”. |

|  |  |
| --- | --- |
| **Método** | **setInputValue** |
| Requiere | * {String} name: El nombre del control. * {String} name: El valor del control. |
| Efectúa | Establece el valor de un control. |
| Modifica | N/A |
| Pseudocódigo | * Si es usuario maestro, ajusta el valor global del mundo. * Si no es usuario maestro, ajusta el valor del elemento de entrada. |

|  |  |
| --- | --- |
| **Método** | **initOutputs** |
| Requiere | N/A |
| Efectúa | Inicializa componentes de los valores de salida. |
| Modifica | N/A |
| Pseudocódigo | * Por cada elemento de salida, obtiene el nombre desde la etiqueta padre. |

|  |  |
| --- | --- |
| **Método** | **getOutputsValues** |
| Requiere | N/A |
| Efectúa | Retorna los valores de salida. |
| Modifica | N/A |
| Pseudocódigo | * Recorrer todos los elementos de salida, obteniendo el nombre y el valor. * Devuelve la lista elementos de salida. |

|  |  |
| --- | --- |
| **Método** | **setOutputsValues** |
| Requiere | * {Array} outputs: Los valores de salida. |
| Efectúa | Establece los valores de salida. |
| Modifica | N/A |
| Pseudocódigo | * Recorre la lista de elementos de salida, asignando el valor al elemento de salida correspondiente. |

|  |  |
| --- | --- |
| **Método** | **disableButtons** |
| Requiere | N/A |
| Efectúa | Deshabilita los botones de tipo comando. |
| Modifica | N/A |
| Pseudocódigo | * Asigna al atributo “disabled” a los elementos correspondientes. * Asigna un estilo determinado para definir los elementos correspondientes en otro color. |

|  |  |
| --- | --- |
| **Método** | **enableButtons** |
| Requiere | N/A |
| Efectúa | Habilita los botones de tipo comando. |
| Modifica | N/A |
| Pseudocódigo | * Elimina al atributo “disabled” a los elementos correspondientes. * Elimina el estilo determinado que define los elementos correspondientes en otro color. |

|  |  |
| --- | --- |
| **Método** | **disableInputs** |
| Requiere | N/A |
| Efectúa | Deshabilita los controles de entrada. |
| Modifica | N/A |
| Pseudocódigo | * Asigna al atributo “disabled” a los elementos correspondientes. * Asigna un estilo determinado para definir los elementos correspondientes en otro color. |

|  |  |
| --- | --- |
| **Método** | **enableInputs** |
| Requiere | N/A |
| Efectúa | Habilita los controles de entrada. |
| Modifica | N/A |
| Pseudocódigo | * Elimina al atributo “disabled” a los elementos correspondientes. * Elimina el estilo determinado que define los elementos correspondientes en otro color. |

|  |  |
| --- | --- |
| **Método** | **initNoMasterConnected** |
| Requiere | N/A |
| Efectúa | Ejecutas las acciones correspondientes, cuando el usuario maestro no está conectado. |
| Modifica | N/A |
| Pseudocódigo | * Coloca un spinner que bloquead cualquier actividad por parte del usuario. * Deshabilita los botones. |

|  |  |
| --- | --- |
| **Método** | **sendAction** |
| Requiere | * {String} action: La acción a ejecutar. * {Object} params: Los parámetros de la acción. |
| Efectúa | Envía un mensaje de ejecutar acción al servidor por medio de web sockets, para que sea enviada a los demás clientes en la sesión. |
| Modifica | N/A |
| Pseudocódigo | * Concatena la cadena “\_\_fromClient” al nombre de la acción a ejecutar. * Envía mensaje el servidor. |

|  |  |
| --- | --- |
| **Método** | **start** |
| Requiere | N/A |
| Efectúa | Ejecuta la acción de inicialización de la sesión se simulación. |
| Modifica | N/A |
| Pseudocódigo | * Si es usuario maestro:   + Habilita los botones.   + Habilita los controles. * Si está habilitado la manipulación de los controles para todos los usuarios:   + Habilita los controles. * Remueve el spinner. * Establece el estado como “Listo”. |

|  |  |
| --- | --- |
| **Método** | **end** |
| Requiere | N/A |
| Efectúa | Ejecuta la acción de finalización de la sesión se simulación. |
| Modifica | N/A |
| Pseudocódigo | * Despliega ventana indicando que el usuario maestro no está conectado. * Aplica acciones a ejecutar cuando el usuario maestro no está conectado. * Establece el estado como “Terminado”. |

|  |  |
| --- | --- |
| **Método** | **doIfSessionStarted** |
| Requiere | N/A |
| Efectúa | Ejecuta las acciones correspondientes si un nuevo usuario ingresa y la sesión ya ha iniciado. |
| Modifica | N/A |
| Pseudocódigo | * Si no está esperando a que la sesión se reinicie:   + Despliega el spinner.   + Despliega una ventana indicando que la sesión ya inició. |

|  |  |
| --- | --- |
| **Método** | **doIfSessionRestarted** |
| Requiere | N/A |
| Efectúa | Ejecuta las acciones correspondientes cuando la sesión se reinicia. |
| Modifica | N/A |
| Pseudocódigo | * Remueve el spinner. * Oculta la ventana que indica que la sesión ya inició (si está desplegada). |

|  |  |
| --- | --- |
| **Método** | **doWhenNewUserConnected** |
| Requiere | * {Array} usersList: La lista de usuarios. |
| Efectúa | Ejecuta las acciones correspondientes cuando ingresa un nuevo usuario. |
| Modifica | N/A |
| Pseudocódigo | * Si es usuario maestro, envía al servidor mensaje de “Ajustar estado”. * Actualiza la lista de usuarios conectados. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.connect\_Response** |
| Requiere | * {Object} params: Los parámetros de la acción. |
| Efectúa | Recibe del servidor la respuesta al conectarse a los web sockets. |
| Modifica | N/A |
| Pseudocódigo | * Asigna variables globales a la simulación. * Si el valor del parámetro “start” es verdadero, ejecuta la función de inicio de la simulación. * Establece que el socket está listo. * Si es usuario maestro, chequea el estado de la simulación. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.updateUsers** |
| Requiere | * {Object} params: Los parámetros de la acción. |
| Efectúa | Recibe del servidor la acción de actualizar usuarios. |
| Modifica | N/A |
| Pseudocódigo | * Ejecuta la función cuando se conecta un nuevo usuario. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.executeCommand** |
| Requiere | * {Object} params: Los parámetros de la acción. |
| Efectúa | Recibe del servidor la acción de ejecutar comando. |
| Modifica | N/A |
| Pseudocódigo | * Extrae los parámetros del comando. * Ejecuta el comando correspondiente. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.setControl** |
| Requiere | * {Object} params: Los parámetros de la acción. |
| Efectúa | Recibe del servidor la acción de establecer el valor de un control. |
| Modifica | N/A |
| Pseudocódigo | * Llama a la función encargada de ajustar el valor del control correspondiente. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.updateSpeed** |
| Requiere | * {Object} params: Los parámetros de la acción. |
| Efectúa | Recibe del servidor la acción de actualizar velocidad de la simulación. |
| Modifica | N/A |
| Pseudocódigo | * Llama a la función encargada de ajustar el valor del campo velocidad. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.applyUpdate** |
| Requiere | * {Object} params: Los parámetros de la acción. |
| Efectúa | Recibe del servidor la acción de aplicar actualización de la simulación. |
| Modifica | N/A |
| Pseudocódigo | * Llama la función encargada de aplicar la actualización de la simulación. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.setStatus** |
| Requiere | * {Object} params: Los parámetros de la acción. |
| Efectúa | Recibe del servidor la acción de actualizar el status. |
| Modifica | N/A |
| Pseudocódigo | * Llama a la función encargada de establecer el estado de la simulación. * Llama a la función encargada de ajustar los valores de las elementos de salida. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.start** |
| Requiere | * {Boolean} enabledControls: Indica si los controles están habilitados o no. |
| Efectúa | Recibe del servidor la acción de iniciar sesión de simulación. |
| Modifica | N/A |
| Pseudocódigo | * Llama a la función encargada de iniciar la sesión de simulación. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.end** |
| Requiere | N/A |
| Efectúa | Recibe del servidor la acción de finalizar sesión de simulación. |
| Modifica | N/A |
| Pseudocódigo | * Llama a la función encargada de terminar la sesión de simulación. |

|  |  |
| --- | --- |
| **Método** | **overwritedCommands.getMessage** |
| Requiere | * {Object} params: Los parámetros de la acción. |
| Efectúa | Recibe del servidor la acción de obtener mensaje. |
| Modifica | N/A |
| Pseudocódigo | * Imprime el mensaje en consola. |

|  |  |
| --- | --- |
| **Método** | **initSockets** |
| Requiere | N/A |
| Efectúa | Inicializa las acciones que se pueden recibir del servidor por medio de web sockets. |
| Modifica | N/A |
| Pseudocódigo | * Define que se ejecute la función de conectar con el servidor, cuando se abre el socket. * Recorre todas las funciones sobrescritas y define que se ejecute, cuando se recibe un mensaje del servidor, cuya acción coincida con el nombre de la función correspondiente. |

|  |  |
| --- | --- |
| **Método** | **Simulation.create** |
| Requiere | N/A |
| Efectúa | Crea un objeto “Simulation”. |
| Modifica | N/A |
| Pseudocódigo | * Coloca el modo de depuración de “Ractive” en falso. * Obtiene el objeto de la simulación. * Llama a la función encargada de sobrescribir las funciones nativas de Tortoise necesarias. * Retorne el objeto de la simulación. |

|  |  |
| --- | --- |
| **Método** | **Simulation.setupObject** |
| Requiere | * {Object} modelObj: Objeto Simulation. |
| Efectúa | Sobre-escribe algunos de los métodos de Tortoise, necesarios antes de que se inicialice Tortoise. |
| Modifica | N/A |
| Pseudocódigo | * Sobrescribe la función nativa de Tortoise, encargada de aplicar la actualización de la simulación. * Sobrescribe la función nativa de Tortoise, encargada de compilar el modelo. |

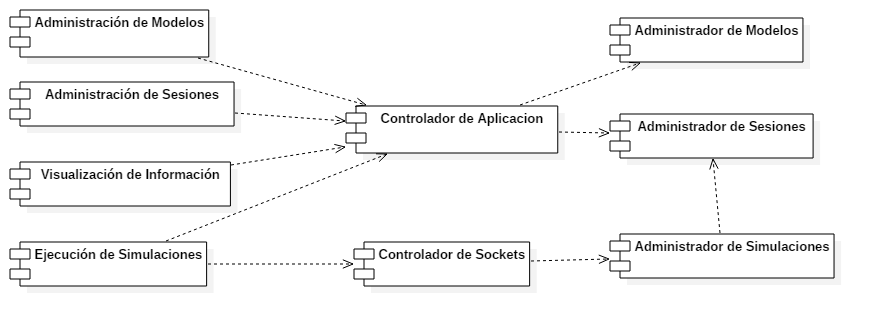
|  |  |
| --- | --- |
| **Método** | **Simulation.getInstance** |
| Requiere | N/A |
| Efectúa | Basado en el patrón “Singleton”, retorna una instancia del objeto “Simulation”. |
| Modifica | N/A |
| Pseudocódigo | * Si el objeto no existe, lo crea. * Retorna el objeto. |

|  |  |
| --- | --- |
| **Método** | **Simulation.formatUI** |
| Requiere | N/A |
| Efectúa | Formatea la interfaz de grafica en la página de simulación, para que cuente con el menú y las demás opciones de la interfaz gráfica del resto de la aplicación. |
| Modifica | N/A |
| Pseudocódigo | * Ejecuta la solicitud Ajax que obtiene el layout de la interfaz gráfica. * Obtiene el elemento con el contenido del modelo. * Obtiene el elemento con el contenido del layout. * Inserta el elemento con el contenido del modelo dentro del elemento con el layout. * Inserta el elemento con el contenido de la simulación, con el elemento comprendido del layout y el modelo. * Hace visible el elemento donde se muestra la lista de usuarios conectado. * Inserta una función para remover le header del UI a la pila de funciones de jquery. * Inserta el texto de “Usuario maestro no conectado” a la ventana correspondiente. |

|  |  |
| --- | --- |
| **Método** | **Simulation.formatUI.users** |
| Requiere | * {Array} users: La lista de usuarios. |
| Efectúa | Formatea la sección de Usuarios de la sesión en interfaz de gráfica. |
| Modifica | N/A |
| Pseudocódigo | * Obtiene el elemento que contiene la lista de usuarios. * Obtiene el elemento que contiene la plantilla de cada ítem de la lista de usuarios. * Por cada usuario en la lista:   + Si es usuario maestro, asigna obtiene la plantilla de ítem de usuario maestro.   + Si es usuario normal, asigna obtiene la plantilla de ítem de usuario normal.   + Inserta la plantilla del usuario a la lista de usuarios (UI). |

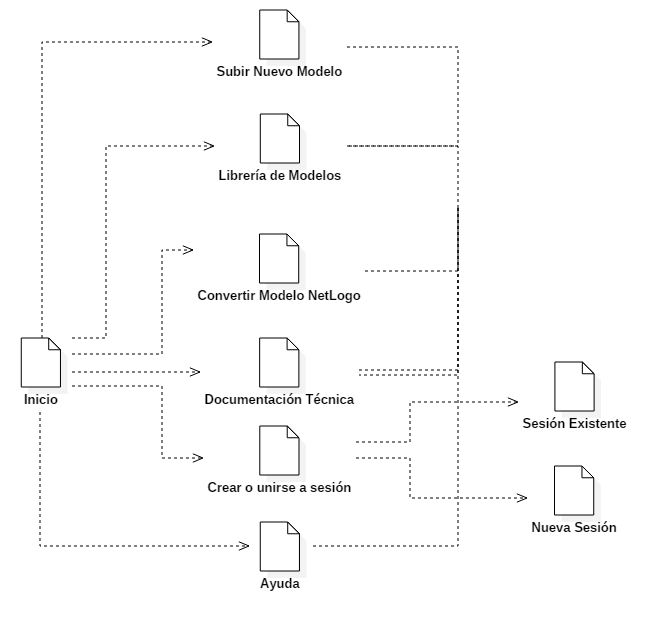
## 3.4. Diseño de mensajes entre componentes

### 3.4.1. Diagrama de componentes

****

## 3.5. Diseño de la interfaz de usuario

### 3.5.1. Diagrama de navegación entre pantallas



# 4. Convenciones de programación

A continuación se detallan algunas convenciones de programación en JavaScript, las cuáles fueron seguidas en el desarrollo de NodeTortoise.

**Archivos JavaScript**

* Los programas en JavaScript deben ser almacenados en y enviados como archivos .js.
* El código en JavaScript no debe ser insertado en los archivos HTML a menos que el código sea específico a una sesión individual. El código en HTML añade significativamente al tamaño de la página sin ofrecer oportunidades de ser mitigado mediante cache y compresión.
* Las etiquetas <script src=archivo.js> deben ser colocadas lo mas tarde posible en el cuerpo del documento.

**Identación**

* La indentación se hace a cuatro espacios.

**Declaración de variables**

* Todas las variables deben ser declaradas antes de ser usadas. JavaScript no lo requiere, pero hacerlo hace los programas más fáciles de leer y hace más fácil detectar variables que no han sido declaradas que pueden convertirse en variables globales implícitas. Las variables globales implícitas nunca deben ser usadas.
* Las instrucciones var deben ser las primeras instrucciones en el cuerpo de la función.
* El alcance de todas las variables en JavaScript es la función donde están definidas, JavaScript no tiene alcance de variables a nivel de bloque, así que definir variables en bloques puede confundir a programadores con experiencia en otros lenguajes de la familia de C. Defina todas las variables al inicio de la función.
* El uso de variables globales debe ser minimizado. Las variables globales implícitas nunca deben ser usadas.
* El formato de las variables debe ser “lowerCamelCase”.

**Declaración de clases**

* El formato del nombre de las clases debe ser “UpperCamelCase”.

**Declaración de funciones**

* Todas las funciones deben ser declaradas antes de ser usadas. Las funciones internas deben aparecer después de las instrucciones var.
* No debe haber espacio entre el nombre de una función y el ( (paréntesis izquierdo) de su lista de parámetros. Solo debe haber un espacio entre el ) (paréntesis derecho) y el { (corchete izquierdo) que inicia el cuerpo de la instrucción. El cuerpo mismo debe ser indentado cuatro espacios. El } (corchete derecho) se alinea con la línea que contiene el inicio de la declaración de la función.
* Si una función es anónima entonces debe haber un espacio entre la palabra function y el ( (paréntesis izquierdo). Si se omite el espacio entonces puede parecer que el nombre de la función es function lo cual es una lectura incorrecta del código.
* El uso de funciones globales debe ser minimizado.
* Cuando una función vaya a ser invocada inmediatamente, la llamada de invocación debe ser rodeada por completo de paréntesis de modo que sea claro que el valor producido es el resultado de la función y no la función en si misma.
* Los nombres deben ser formados con las 26 letras mayúsculas y minúsculas del alfabeto inglés (A .. Z, a .. z), los 10 dígitos (0 .. 9), y la \_ (barra inferior). Evite el uso de caracteres en otros idiomas porque puede que no sean leídos correctamente o que no se los entienda. No emplee el signo $ (dólar) ni la \ (barra invertida) en los nombres.
* Los constructores que deban ser usados con el prefijo new deben empezar con una letra mayúscula: JavaScript no emite una advertencia en la etapa de compilación ni en la etapa de ejecución si un new es omitido. Cosas malas pueden ocurrir si new no es empleado, así que la convención de empezar el nombre con mayúscula es la única defensa que tenemos.
* Los nombres de las variables globales deben ir todo en mayúsculas (JavaScript no tiene macros ni constantes así que no hay razón para usar palabras todas en mayúsculas para denotar cosas que JavaScript no tiene)

**Instrucciones**

* Cada línea debe contener a lo sumo una instrucción.
* Ponga un “;” (punto y coma) al final de cada instrucción simple.