**Diseño de los módulos de interacción con las Bases de datos**

En este documento se tratará las ideas sobre el diseño de los módulos que serán necesarios implementar tanto en la web como en el servidor principal y el servidor de máquinas virtuales para interaccionar con las bases de datos correspondientes. El diseño de cada módulo incluirá una descripción acerca de las funcionalidades a las que se enfrentará dicho módulo. Las relaciones entre los diferentes módulos de encuentran detalladas por medio de diagramas de clases.

Para cualquier duda con respecto al diseño o tipo de los campos de las bases de datos, se deberá consultar el documento acerca del diseño de las bases de datos.

**Módulos en el servidor Web**

**UsersAccess**

Este módulo contiene las funciones encargadas de la gestión de los usuarios cuando todavía no se ha accedido a la página con un determinado usuario. Así este módulo contendrá funciones encargadas de dar de alta a un nuevo usuario o loguear a un usuario existente. Así este módulo dispondrá de las siguientes funciones:

* **Int RegistUser(name,pass)** : Recibe como argumentos el nombre de usuario y la contraseña y da de alta al nuevo usuario en la base de datos en caso de que cumpla determinadas condiciones:
  + El nombre de usuario no aparece ya como usuario de la base de datos
  + La contraseña cumple las características pertinentes (dígitos admisibles, longitud aceptable…).
  + El nombre de usuario es una cuenta válida para los nuevos servicios de gestión de cuentas adoptados por la UCM (usando google) .
  + En general, se cumplen todas las condiciones detalladas en el registro.

Además esta función, una vez verificadas todas las condiciones anteriores, comprobará los permisos que deben asociarse a dicho usuario. Estos permisos serán asignados por medio de la función associateType. Esta función devolverá el valor del tipo de permiso que se le ha concedido al usuario y -1 en caso de no haber podido concederlo.   
En resumen, los pasos a seguir por parte de esta función serán:

1. Recibe el nombre de usuario y la contraseña y evalua las condiciones. Si alguna de las condiciones no se cumple, devuelve false.
2. Una vez cumplida las condiciones, llama a la función encargada de asociar permisos ha dicho usuario.
3. En caso de que los permisos hayan sido asociados correctamente se procede a dar de alta al usuario en la base de datos con el nombre , contraseña e identificador de permisos obtenidos.
4. Se devuelve el identificador asignado a dicho usuario

* **Int associateType(name)** : Para llevar a cabo la concesión de permisos se supondrá que esta puede deducirse a partir del nombre del usuario (el cual se encuentra asociado a una cuenta google). De esta forma según la composición del nombre este formará parte de un determinado grupo. Este grupo se consultará en la tabla correspondiente para obtener el identificador de grupo que le corresponde. En caso de no encontrar ninguna entrada de la tabla que coincida con el tipo de usuario determinado se devolverá -1, en otro caso se devolverá el identificador de dicho tipo.
* **Int LoginUser(name,pass)** : Esta función recibe como parámetros el nombre de un determinado usuario y su contraseña y verifica en la base de datos que dicho usuario ya existe. Para llevar a cabo una correcta autentificación deberán cumplirse las siguientes condiciones:
  + El nombre del usuario existe en la tabla Usuarios de la Base de datos correspondiente.
  + La contraseña asociada a dicho nombre de usuario coincide con la recibida como parámetro.

En caso de no cumplir alguna de las condiciones se devolverá un número que represente el tipo de error devuelto. En caso de que se cumplan ambas condiciones se extraerá de las tabla correspondientes el id de los permisos asociados a dicho usuario y se llamará a la función encargada de ajustar el sistema para orientarlo a las funcionalidades a las que este tipo de usuarios tiene acceso. En caso de que todo haya funcionado de manera correcta la función devolverá el identificador asociado a dicho usuario.

**UsersManagement**

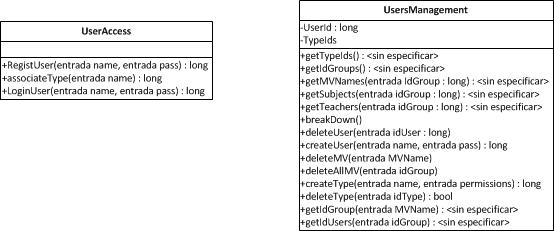
Este módulo se encargará de ejecutar las diferentes funciones que se pueden realizar con respecto a las entradas de la base de datos correspondiente a un usuario en concreto. Este módulo se ejecutará una vez que el usuario se haya registrado o logueado con éxito y se encuentre una sesión abierta. Para realizar las operaciones pertinentes es necesario que la clase residente en este módulo contenga los atributos con la información del nombre y el tipo de permiso. Así esta clase dispondrá de dos atributos:

* **UserId:** Identificador asociado al usuario que se haya dado de alta. Este identificador nos permitirá acceder a las correspondientes entradas de las tablas asociadas a dicho usuario.
* **TypeIds**: Una lista con los identificadores de los tipos asociados a dicho usuario. Estos identificadores permitirán evaluar que funcionalidades son accesibles a este usuario.

Con respecto a los métodos, esta clase contendrá todos los métodos con las posibles funcionalidades que un usuario pueda ejecutar de manera directa sobre el sistema, independientemente del tipo de usuario dado de alta en la sesión. Para funcionalidades que no pueden ser ejecutadas por todos los usuarios, lo primero que hará su función determinada es comprobar los tipos para dicho usuario y evaluar a partir de ellos si ese usuario puede o no ejecutar la acción. En caso de que el usuario no tenga los privilegios se lanzará una excepción, continuándose la ejecución con normalidad en caso contrario.

Las funciones que serán necesarias en esta clase con respecto a la interacción con la base de datos serán las siguientes:

* **List getTypeIds()** : Esta función devuelve una lista con los identificadores de tipos asociados al usuario. Esta función será llamada en la constructora para poder obtener el atributo con la lista de tipos correspondiente.
* **list getIdGroups()** : Esta función devuelve una lista con los identificadores de grupos asociados al usuario dado de alta en esta sesión. Esta lista podrá ser usada para acceder a los nombres de las diferentes máquinas virtuales o mostrar la información del grupo, como por ejemplo el nombre de la asignatura, el curso …
* **list getMVNames(idGroup)** : Esta función devuelve una lista con los diferentes nombres de las máquinas virtuales asociadas a un determinado grupo.
* **List getSubjects(idGroup) :** Esta función devuelve una lista con el nombre de la asignatura asociada al grupo pasado como parámetro, el curso en la que se imparte dicha asignatura, el año académico en el que se imparte y el grupo de clase asociado (véase como grupo el valor alfanumérico a,b,c.. asociado a cada clase en dicha asignatura).
* **List getTeachers(idGroup)** : Esta función devuelve el nombre de los profesores que se encuentran asociados a dicha asignatura. Para ello se consultará en la tabla correspondiente los identificadores asociados a dicho grupo y para cada identificador de usuario se comprobará el tipo de usuario que es (profesor, alumno o administrador). Si el tipo de usuario es profesor, se añadirá su nombre a la lista.
* **Void breakDown()** : Esta función permite a cualquier tipo de usuario darse de baja en la base de datos correspondiente. Una vez el usuario sea dado de baja este módulo terminará y se volverá al estado sin registrar.
* **Void deleteUser(idUser)**: Esta función será una función exclusiva para los administradores y con ella se dará de baja en la base de datos el usuario cuyo identificador se pasa como argumento. El administrador podrá eliminar cualquier usuario excepto a si mismo.
* **int createUser(name,pass):** Esta función será una función exclusiva para los administradores y con ella se dará de alta de forma interna a un nuevo usuario, cuyo nombre y contraseña se pasan como argumento. Si la función tiene éxito se devolverá el identificador asociado a dicho nuevo usuario. En caso contrario se devolverá -1, por ejemplo.
* **Void deleteMV(MVName):** Esta función será una función exclusiva para los administradores y los profesores. Esta función permite eliminar de la base de datos una MV cuyo nombre viene dado como parámetro. En caso de que el usuario dado de alta sea un profesor se comprobará que la MV que se quiere eliminar pertenezca a un grupo asociado a dicho profesor, en caso contrario se lanzará una excepción.
* **Void deleteAllMV(idGroup) :** Esta función será una función exclusiva para los administradores y los profesores. Esta función elimina de la base de datos todas las máquinas virtuales asociadas a un determinado grupo, cuyo identificador se pasa como parámetro. En caso de que el usuario dado de alta sea un profesor se comprobará previamente que el grupo cuyas máquinas virtuales quieren eliminarse, se encuentra asociado a dicho profesor, lanzando una excepción en caso contrario.
* **Int createType(name, permissions) :** Esta función será exclusiva para los administradores y permitirá crear un nuevo tipo de usuario con un nombre y una lista de permisos pasados como argumentos. En caso de éxito devolverá el identificador asociado al nuevo tipo. En caso contrario devolverá -1, por ejemplo.
* **Bool deleteType(idType) :** Esta función será exclusiva para administradores y permite eliminar un tipo de usuarios cuyo identificador se pasa como parámetro. Además de eliminar de la base de datos el tipo de usuarios también se eliminarán todos los usuarios que se encontraban asociados a dicho tipo. El administrador podrá eliminar cualquier tipo de usuario excepto el suyo propio.
* **list getIdGroup(MVName) :** Esta función devuelve la lista con todos los identificadores de grupos que se encuentran asociados a una máquina virtual pasada como parámetro.
* **List getIdUsers(idGroup) :** Esta función solo será accesible para los administradores y profesores cuyo grupo coincida con el pasado como argumento. Esta función permite obtener una lista con los identificadores de los usuario que se encuentran asociados con el grupo cuyo identificador se pasa como argumento.



**Módulos en el servidor principal**

En este servidor deberán encontrarse los módulos orientados a gestionar los diferentes servidores de máquinas virtuales disponibles en cada momento, así como las imágenes asociadas a cada uno de estos servidores. Para gestionar estas acciones de manera modular dispondremos de dos clases, una para la extracción y gestión de la información asociada a los servidores de máquinas virtuales residentes en el servidor principal y otro para gestionar las diferentes imágenes asociadas a un servidor de máquinas virtuales en concreto.

**ServerMVManager**

Esta clase se encargará de extraer y modificar la información relativa al puerto y la IP de los diferentes servidores de máquinas virtuales. Además a partir de esta clase también se podrán dar de alta y de baja en la base de datos nuevos servidores, indicando los detalles sobre la IP, el puerto y el número máximo de máquinas virtuales que dicho servidor puede ejecutar al mismo tiempo. Para ello serán necesarias las siguientes funciones:

* **String getPort(IdServer)** : devuelve el puerto asociado al servidor pasado como argumento.
* **String getIP(IdServer)** : devuelve la IP asociada a un determinado servidor pasado como argumento.
* **Int getMaxMV(IdServer)** : devuelve el número máximo de máquinas virtuales que el servidor puede soportar
* **Int getFreeMV(IdServer)** : devuelve el número de máquinas virtuales libres que el servidor pasado como argumento puede soportar todavía.
* **List getServers()** : esta función permite obtener una lista con los identificadores de todos los servidores de máquinas virtuales que actualmente se encuentran dados de alta en la base de datos.
* **Void setMaxMV(IdServer,maxMV)** : permite configurar el número de máquinas virtuales que admite como máximo el servidor de máquinas virtuales que se le pasa como argumento.
* **Int createServer(port,IP,maxMV)** : esta función permite crear un Nuevo servidor de máquinas virtuales con el puerto, la IP y el número máximo de máquinas virtuales que se le pasan como argumento. En caso de que el servidor se cree de manera correcta se devolverá el identificador asociado a dicho servidor.
* **Bool deleteServer(IdServer)** : Esta función permite eliminar un determinado servidor de máquinas virtuales de la base de datos cuyo identificador se le pasa como argumento. En caso de que el servidor se haya eliminado de manera adecuada, se devolverá true.
* **List getImages()** : Esta función devuelve una lista con los identificadores de todas las imágenes que el servidor principal puede dar acceso (independientemente del servidor de máquinas virtuales donde se encuentre dicha imagen).
* **List getImageServers(idImage)** : Esta función devuelve una lista con todos los identificadores de servidores que pueden dar acceso a la imagen cuyo identificador se pasa como argumento.
* **Bool deleteImageInServers(idImage)** : Esta función permite eliminar todas las apariciones en los diferentes servidores de máquinas virtuales de una determinada imagen cuyo identificador se pasa como argumento. Con esta función podrá borrarse todo rastro de una determinada imagen en el servidor principal.

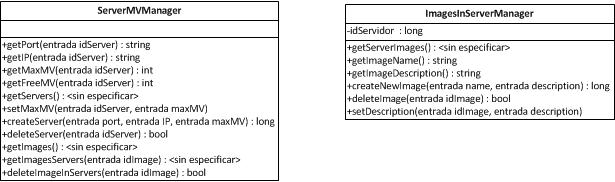
**ImagesInServerManager**

Esta clase permite configurar que imágenes de máquinas virtuales pueden utilizarse en cada uno de los servidores. Esta clase contendrá como atributo el identificador del servidor de máquinas virtuales del cual queremos obtener o modificar sus imágenes. Por lo tanto en esta clase tendremos los siguientes atributos:

* **idServidor :** Servidor de la máquina virtual cuyas imágenes queremos gestionar.

En cuanto a las funciones de esta clase tendremos que implementar funciones que permitan dar de alta a nuevas imágenes asociadas a este servidor, dar de baja una máquina virtual existente y mostrar las imágenes asociadas al servidor de máquinas virtuales así como el nombre y la descripción de las mismas. Las funciones que contendrá esta clase son:

* **list getServerImages()** : esta función devuelve una lista con los identificadores de las imágenes asociadas al servidor de máquinas virtuales del atributo.
* **String getImageName(idImage)** : esta función devuelve el nombre asociado a una determinada imagen cuyo identificador se le pasa como argumento. En caso de no existir dicha imagen se lanzará una excepción.
* **String getImageDescription(idImage)** : esta función devuelve la descripción de una determinada imagen cuyo identificador se le pasa como argumento. En caso de no existir dicha imagen se lanzará una excepción.
* **Int createNewImage(name,description)** : esta función permite dar de alta en la base de datos una nueva imagen asociada al servidor de máquinas virtuales definido como atributo. Para dar de alta esta imagen se requiere de un nombre y una descripción que se le pasan como argumentos. Si la imagen se ha dado de alta de manera correcta se devolverá el identificador asignado a la nueva imagen, en otro caso se lanzará una excepción.
* **bool deleteImage(idImage)** : esta función da de baja en la base de datos una determinada imagen asociada al servidor de máquinas virtuales guardado como atributo. Para ello se pasa el identificador de la imagen a dar de baja y se devolverá true en caso de que la imagen se haya podido eliminar de forma correcta.
* **Void setDescription(idImage,description) :** esta función permite editar la descripción asociada a una determinada imagen cuyo identificador se le pasa como argumento.

****

**Módulos en el servidor de Máquinas virtuales**

En este servidor deberán gestionarse todas las funcionalidades relacionadas con la base de datos residente en el servidor de máquinas virtuales. Esta base de datos contiene la información correspondiente a las imágenes que se pueden ejecutar en este servidor de máquinas virtuales, así como la información relativa de todas las máquinas virtuales que se encuentran en ejecución en un momento dado. Como ya se dice en el diseño de las bases de datos, las máquinas virtuales en ejecución se encontrarán identificadas por el puerto VNC que se le ha dado a dicha máquina virtual. Para gestionar esta información de forma modular, se dividirán las funcionalidades en dos clases, según correspondan a la gestión de las características de las imágenes accesibles en el actual servidor de máquinas virtuales, o bien corresponda a las funcionalidades de gestión de las máquinas virtuales actualmente en ejecución.

**ImagesManager**

Esta clase permite gestionar las diferentes características de las imágenes accesibles en el servidor de máquinas virtuales actual. En ella se definirán las funciones necesarias para añadir y eliminar nuevas imágenes de máquinas virtuales y cambiar la ruta de la imagen o del fichero de configuración de una determinada imagen.

Las funciones necesarias en esta clase serán:

* **List getImages()** : esta función devuelve una lista con todos los identificadores de imágenes que se encuentran registradas en el servidor de máquinas virtuales.
* **String getName(idImage)** : esta función devuelve el nombre de la imagen cuyo identificador se pasa como argumento.
* **String getImagePath(idImage)** : esta función devuelve la ruta donde se encuentra físicamente la imagen cuyo identificador de imagen se pasa como argumento.
* **String getFileConfigPath(idImage)** : esta función devuelve la ruta donde se encuentra el fichero de configuración asociado a la imagen cuyo identificador se pasa como argumento.
* **Void setImagePath(path)** : esta función permite cambiar la ruta de la imagen cuyo identificador se pasa como argumento. En caso de no encontrarse la imagen en la base de datos o que la ruta pasada como argumento no corresponda a ninguna imagen física se lanzará una excepción.
* **Int createImage(name,imagePath,FileConfigPath)** : esta función permite registrar en la base de datos una nueva imagen de máquina virtual. Para ellos será necesario el nombre que se le quiere dar a la imagen, la ruta donde se encuentra la imagen de forma física y la ruta donde se encuentra el fichero de configuración asociado a dicha imagen. En caso de que la imagen sea registrada correctamente se devolverá el identificador que se ha asociado a dicha imagen.
* **Bool deleteImage(idImage)** : esta función permite dar de baja de la base de datos una determinada imagen cuyo identificador se pasa como argumento. Si el sistema consigue dar de baja a la imagen de forma correcta entonces se devolverá true.

**RunningMV**

Esta clase se encargará de gestionar las características de las diferentes máquinas virtuales que se encuentran en ejecución en un momento determinado. Además también se encargará de registrar en la base de datos las nuevas máquinas virtuales en ejecución y de dar de baja aquellas máquinas que se apaguen.

Para realizar estas acciones, esta clase contendrá las siguientes funciones:

* **List getRunningPorts()** : esta función devuelve una lista con los puertos VNC correspondientes a las máquinas virtuales que se encuentran actualmente en ejecución.
* **List getUsers():** Esta función devuelve una lista con los identificadores de todos los usuarios que actualmente se encuentran ejecutando una determinada máquina virtual en este servidor de máquinas virtuales.
* **Void UserDown(idUser)** : esta función da de baja en la base de datos todas aquellas máquinas virtuales (con sus correspondientes puertos VNC, rutas de imágenes y rutas de ficheros de configuración) que hayan sido ejecutadas por un determinado usuario, cuyo identificador se pasa como argumento. En caso de no encontrarse dicho usuario en ninguna de las máquinas en ejecución se lanzará una excepción.
* **Int getImage(VNCPort)** : esta función devuelve el identificador de la máquina virtual que se encuentra en ejecución en el puerto VNC pasado como argumento.
* **String getImageName(VNCPort)** : esta función devuelve el nombre de la máquina virtual que se encuentra en ejecución en el puerto VNC pasado como argumento.
* **String getImageCopyPath(VNCPort)** : esta función devuelve la ruta asociada a la copia de la imagen de la máquina virtual que se encuentra en ejecución en el puertoVNC pasado como argumento.
* **String getFileConfigCopyPath(VNCPort)** : esta función devuelve la ruta donde se encuentra la copia del fichero de configuración asociado a la máquina virtual que se encuentra en ejecución en el puerto VNC pasado como argumento.
* **String getMac(VNCPort)** : esta función devuelve la dirección MAC del cliente VNC cuyo puerto se pasa como argumento.
* **String getPassword(VNCPort)** : esta función devuelve la contraseña que se ha dado al puerto VNC que se le pasa como argumento.
* **String createVNCPort(idUser, idMV, ImageCopyPath, FileConfigCopyPath, Mac, password)** : esta función permite dar de alta una nueva máquina virtual en ejecución cuyas características se pasan como argumentos. En caso de que la nueva máquina virtual haya sido dada de alta con éxito en la base de datos, se devolverá el puerto VNC en el cual se está ejecutando dicha máquina virtual. En caso contrario se devolverá un valor de error, -1 por ejemplo.
* **Bool deleteVNCPort(VNCPort)** : esta función da de baja en la base de datos el puerto VNC que se le pasa como argumento y con él, todas las características asociadas al mismo. En caso de que el puerto VNC haya sido dado de baja con éxito se devolverá true.