# Introducción:

Este documento denominado “Diseño del sistema”, es un documento de un conjunto de documentos cuya finalidad es el definir y diseñar una “Aplicación web que gestione un repositorio”.

Para poder describir mejor el problema deberemos desarrollar el significado de lo que se va a definir, es decir, “Aplicación web que gestione un repositorio”. Un repositorio es según la Wikipedia: *Un* ***repositorio de software*** *es un lugar de almacenamiento del cual pueden ser recuperados e instalados los paquetes de software en un ordenador.*

Nuestro repositorio mantendrá un lugar de almacenamiento dentro del ordenador que lo instale, con el fin de contener diversos proyectos que albergaran programas de diversos lenguajes que podrán ser ejecutados. Nuestro repositorio devolverá no es programa almacenado, sino, la información de interactuar con él.

Nuestra aplicación web, gestionara la comunicación con nuestro repositorio. Pidiéndole información y mostrándosela al usuario de una manera sencilla para su entendimiento. Se insta que si se quiere saber más sobre el objetivo de la aplicación se lea el documento general de la memoria.

El objetivo de este documento es el realizar un **diseño del sistema de información.** El cual su función es definir la arquitectura de hardware y software, componentes, módulos y datos de un sistema de cómputo, a efectos de satisfacer ciertos requerimientos. Requerimientos que han sido proporcionados por el análisis del sistema, documento descrito en el conjunto de documentos técnicos.

# Análisis general del sistema:

## Introducción al sistema:

El sistema estará dividido en dos grandes subsistemas que a su vez están divididos en capas independientes, buscando el minino acoplamiento para futuros cambios. Cada subsistema utilizara una arquitectura diferente para sus objetivos principales y conviene diferenciarlos:

* Aplicación web: Subsistema cuyo principal objetivo es la comunicación del usuario con el repositorio. Prima el entendimiento del usuario más que el del repositorio. Se basa en una arquitectura modelo- vista-controlador.
* Repositorio: Subsistema cuyo principal objetivo es la realización de la actividad del sistema. Prima la realización de actividades, frente a sus mandatos. Es necesario un conocimiento de sus mandatos o protocolo para poder realizar actividades. Se basa en una arquitectura REST.

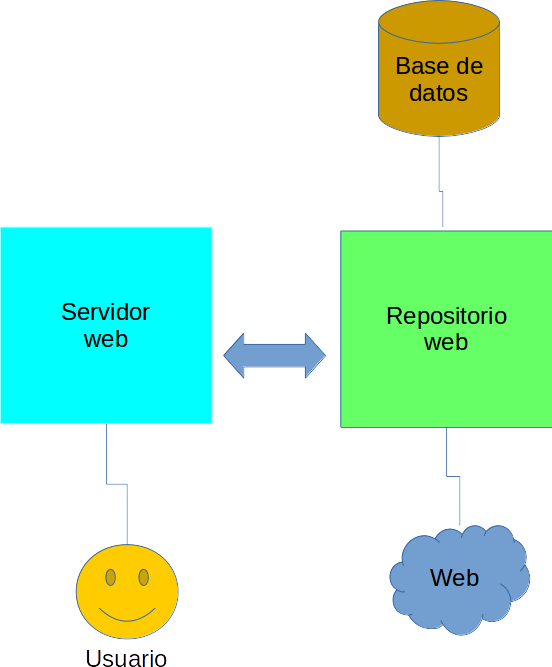


Ilustración (Esquema general aplicación)

## Arquitectura Modelo-vista-controlador:

El modelo vista controlador es una arquitectura muy extendida en las aplicaciones web. Esta arquitectura se basa en un modelo en capas que facilita la interacción del usuario .El modelo MVC genera para cada interacción del usuario con el sistema una vista personalizada para el usuario y el momento. Sus componentes principales son:

* Modelo: Parte de la arquitectura que gestiona el acceso a los datos y los maneja para envolverlos en condiciones correctas para el usuario y las capas superiores.
* Controlador: Es el intermediario entre la vista y el modelo. Esta capa analiza las interacciones entre el usuario y el sistema, mediante eventos e invoca peticiones al modelo para posteriormente realizar una vista personalizada para el usuario con los datos del modelo.
* Vista: Presenta los datos manejados por el modelo y proporciona las herramientas necesarias para generar los eventos que posteriormente avisaran al controlador.

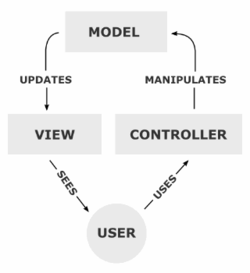


Ilustración (Arquitectura MVC)

## Arquitectura REST:

La Arquitectura REST es una arquitectura de sistemas web que utilizan la tecnología HTTP para el transporte de información o indicación de acciones. Esta arquitectura facilita la creación de comunicación de servidor-cliente. Ya que sigue el protocolo HTTP, protocolo universal para cualquier tipo de aplicación o sistema operativo.

Esta arquitectura se basa en un sistema de Petición-Respuesta asíncrona sin estado, es decir, pides algo y se te responde algo cuando sea posible. La arquitectura es sin estado es decir, el sistema no guarda estados, cada petición es independiente y acaba cuando es finalizada. Los elementos principales de la arquitectura REST son:

* Recursos: Un recurso es una entidad o elemento definido por una identidad. En REST los recursos son enviados, recibidos, manipulados o accionados por servicios de negocios. Cuando solicitamos a un servicio REST recibiremos un Recurso, que puede contener varios recursos a su vez.
* Acciones: Las acciones son una derivación de recursos, es la abstracción en recursos de una acción. Por ejemplo: Si mandamos correr en vez enviaremos un recurso “Anda” con un módulo de dirección y una velocidad. El servidor REST realizara la acción y devolvera un recurso indicando el estado.
* Métodos prefijados: Los servidores REST proveen de un conjunto de acciones a realizar sobre sobre los recursos. Cuando llamamos a un recurso. Tendremos una lista de acciones (GET, PUT, POST, DELETE…)

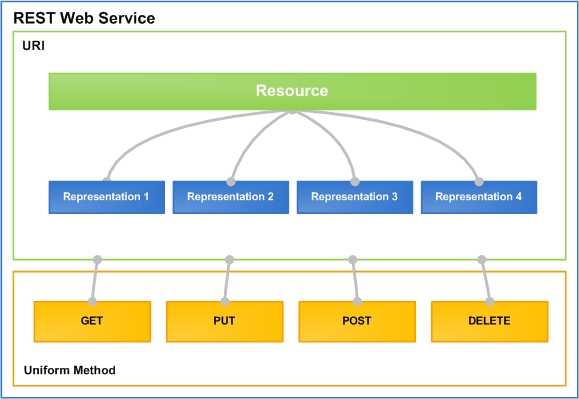


Ilustración (Arquitectura-REST)

# Diseño de datos:

## Introducción:

El diseño de datos proporcionara la información e indicaciones para realizar el modelo de datos del sistema. Esto modelo será permanente y será el utilizado por la aplicación. Para la realización del modelo de datos utilizaremos una tecnología **NO-SQL.** Esta tecnología se basa en no utilizar las bases comunes relacionales, sino que utilizan otros medios para almacenar los datos. En concreto a petición del organizador de este trabajo se utilizara la base de datos **Neo4j.**

## Modelo de los datos-Entidades:

En este punto se describirá las diferentes clases que contendrá el sistema, siguiendo el modelo de negocio del análisis del sistema, describiremos de forma final todas las clases del sistema, listando sus atributos y su tipo de información (Utilizaremos el inglés para denominar los atributos, simplemente por gusto del programador).

Ya dado cualquier tipo de Base de datos, este, va tener unas entidades con unos atributos a continuación describiremos las entidades. Luego dado el tipo de base de datos y sus características. Añadiremos a estas entidades propiedades en función de la base de datos.

### MD-Entidades: UserLogin

|  |  |  |
| --- | --- | --- |
| Entidad | UserLogin | |
| Objetivo | Identificar al usuario del sistema y facilitar información para su autenticación. | |
| Variable | Tipo | Descripción |
| User | String (Email) | Define el email de la identidad del usuario. |
| Pass | String (SHA-1) | Define la contraseña del usuario cifrada con el algoritmo de firma SHA-1. |
| Rol | Enum (UserType:  {USER,ADMIN}) | Define el rol que tiene el usuario. |
| creationTime | Date | Fecha de creación de la entidad. |
| Hashcode | String (SHA-1) | Usando los datos no modificables de la entidad (User y creationTime) se realiza una firma única que identifica a la entidad frente a otras. |

Tabla (UserLogin)

### MD-Entidades: Person

|  |  |  |
| --- | --- | --- |
| Entidad | Person | |
| Objetivo | Proporcionar datos personales del usuario al sistema para su posible representación. | |
| Variable | Tipo | Descripción |
| Name | String | Nombre de la persona. |
| Subname | String | Apellidos de la persona. |
| Birthday | Date | Fecha de nacimiento de la persona. |
| Birthplace | String | Lugar donde vive la persona. |
| Province | String | Provincia donde vive la persona. |
| Country | String | País donde vive la persona. |
| Biografy | String | Pequeño texto para que la persona se pueda expresar. |
| profileImageUrl | String(URL) | Espacio para URL para que la persona pueda poner una imagen representativa de cualquier repositorio de imágenes. |
| dateCreation | Date | Fecha de creación de la cuenta. |
| hashcode | String (SHA-1) | Usando los datos no modificables de la entidad (dateCreation) se realiza una firma única que identifica a la entidad frente a otras. |

Tabla (Person)

### MD-Entidades: Project

|  |  |  |
| --- | --- | --- |
| Entidad | Project | |
| Objetivo | Almacena los datos necesarios para la ejecución del proyecto en el sistema. | |
| Variable | Tipo | Descripción |
| Name | String | Nombre del proyecto |
| Type | Enum (ProjectType:{  JAVA,  PYTHON,OCTAVE}) | Tipo del proyecto, definirá a que servicio del sistema debe acceder. |
| Description | String | Descripción explicativa de lo que hace el proyecto. |
| GitRepositoryURL | String (url) | Dirección URL del repositorio GIT de donde sacar el código del programa. |
| User | String (email) | Usuario del repositorio GIT, por general suelen ser email’s. |
| Password | String | Contraseña del repositorio GIT |
| MainName | String | Nombre del archivo main del código a ejecutar. |
| ResponseName | String | Nombre del archivo que se espera como respuesta tras la ejecución. |
| InputDescription | String | Descripción de los argumentos de entrada que puede aceptar el proyecto. |
| OutputDescription | String | Descripción de los datos de salida que aparecerán tras la ejecución del proyecto. |
| DefaultInputs | String[] | Conjunto de argumentos por defecto en el caso de que el usuario no pueda introducir entradas. |
| ModifyDate | Date | Fecha que determina la modificación de la entidad. |
| CreationDate | Date | Fecha que determina la creación de la entidad. |
| HashCode | String (SHA-1) | Usando los datos no modificables de la entidad (Name y creationDate) conseguimos utilizando el algoritmo SHA-1 una firma única para la entidad. |

Tabla (Project)

### MD-Entidades: Group

|  |  |  |
| --- | --- | --- |
| Entidad | Group | |
| Objetivo | Almacena los datos necesarios para crear un grupo que almacene proyectos y grupos. Además contendrá los permisos de las acciones. | |
| Variable | Tipo | Descripción |
| Name | String | Nombre del grupo |
| Type | Enum (GroupType:{  PUBLIC,  PRIVATE,MAIN}) | Tipo del grupo, definirá que propiedades de acceso tendrá el grupo. |
| Description | String | Descripción explicativa del grupo |
| CreationDate | Date | Fecha de creación del grupo. |
| SharingGroupPermissions | String[] | Conjunto de permisos “YES” o “NO” Que modifican los permisos |
| GroupCreationPermissions | String[] | Conjunto de permisos “YES” o “NO” Que modifican los permisos |
| MemberGestionPermissions | String[] | Conjunto de permisos “YES” o “NO” Que modifican los permisos |
| ProjectPropertiesPermissions | String[] | Conjunto de permisos “YES” o “NO” Que modifican los permisos |
| Hashcode | String(SHA-1) | Usando los datos no modificables de la entidad (Name , creationDate y type) conseguimos utilizando el algoritmo SHA-1 una firma única para la entidad. |

Tabla (Group)

### MD-Entidades: Ejecuciones

|  |  |  |
| --- | --- | --- |
| Entidad | Execution | |
| Objetivo | Almacena los datos de una ejecución. Para saber su estado y su resultado. | |
| Variable | Tipo | Descripción |
| creationDate | Date | Fecha de creación de la entidad ejecución. También servirá como fecha de inicio de la ejecución. |
| StateOfExecution | Enum (ExecutionType:{  STARTED,  FINISH\_WITH\_ERROR,  FINISH\_SUCCESS,  RUNNING}) | Estado de ejecución. Representará el estado actual en el que se encuentra la ejecución. |
| GroupOfExecution | String | Nombre del grupo en el que se está ejecutando el proyecto. Es mera información para el usuario. |
| InputJson | String[] | Conjunto de entradas utilizadas para la ejecución del proyecto. |
| Response | String(JSON) | Respuesta de la ejecución deberá estar en formato JSON Stringify. |
| Console | String | Respuesta interna de los servicios. |
| FinishDate | Date | Fecha que indica la finalización de la ejecución. |
| HashCode | String (SHA-1) | Conjunto de permisos “YES” o “NO” Que modifican los permisos |
| NameExecution | String | Nombre del Project que va se va a ejecutar. Este será el nombre de la ejecución. |

## Modelo de datos - Neo4j:

### MD-Neo4j Introducción:

Neo4j es una base de datos NO-SQL orientada a grafos. NO-SQL significa que no es una base de datos basada en el motor relacional de SQL. Neo4j en vez de guardar la información en disco mediante tablas como podría hacer SQL. Este lo hace en grafos con referencias o “flechas” que unen las entidades. Un ejemplo de la estructura seria la siguiente:

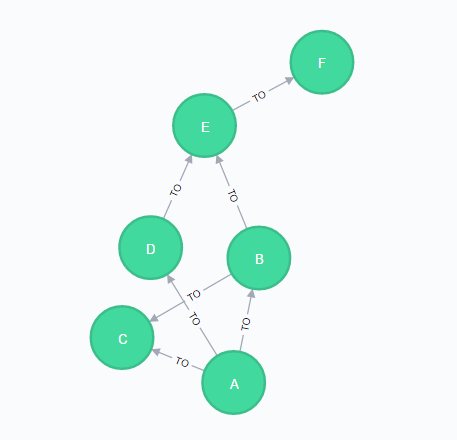


Ilustración (neo4j: Ejemplo básico estructura)

Si se quiere saber más sobre Neo4j se aconseja ir a la memoria donde se mostrara los porqués de esta tecnología.

### MD - Neo4J: Relaciones

Las relaciones son las propiedades especiales de las bases de dato Neo4j. Al contrario de las bases de datos relacionales donde se inyectas claves primarias o secundarias en las tablas, las claves forman parte del framework de la base de datos. Y las relaciones entre entidades son externas, es decir, no realiza modificaciones en las entidades.

Podemos considerar las entidades como verbos o acciones que realiza las entidades entre ellas. La siguiente imagen es un ejemplo de lo mostrado, dos entidades persona realizan una acción que es conocerse.

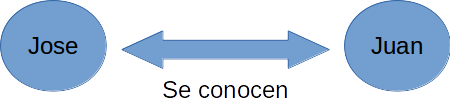


Ilustración (Neo4j: Ejemplo entidad-relación)

Las entidades tienen una dirección es decir una flecha, que indica que referencia a que. A continuación listaremos las relaciones entre entidades del modelo de datos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | Entidad A | Dirección | Entidad B | Descripción |
| MAKE\_REFERENCE  (HACE\_REFERENCIA) | UserLogin | -> | Person | La persona a la que referencia la identidad UserLogin es suya. |
| IS\_OWNER  (ES\_PROPIETARIO) | UserLogin | -> | Group | El usuario hace referencia al grupo que le pertenece. |
| KNOWS  (CONOCE) | UserLogin | -> | Group | El usuario que hace referencia conoce el grupo, es decir es visible. |
| IS\_CREATOR | UserLogin | -> | Project | El usuario que hace referencia es el creador del proyecto referenciado. |
| IS\_SUBGROUP  (ES\_SUBGRUPO) | Group | -> | Group | El grupo que referencia es subgrupo del grupo que es referenciado. |
| CONTAINS(CONTIENE) | Group | -> | Project | El grupo que referencia contiene (que no es propietario) del proyecto referenciado. |
| USE (USA) | Execution | -> | Project | La ejecución referencia al proyecto que esta utilizando. |
| GENERATE (GENERA) | UserLogin | -> | Execution | El usuario que referencia genera a la ejecución que esta generando. |

Tabla (Neo4j: conjunto de entidades)

### MD – Neo4j: Estructura general de datos

En la siguiente figura describiremos la estructura general de datos del sistema. Veremos un ejemplo de entidades y relaciones individuales, ya que si generamos un escenario complicado es casi indescifrable.

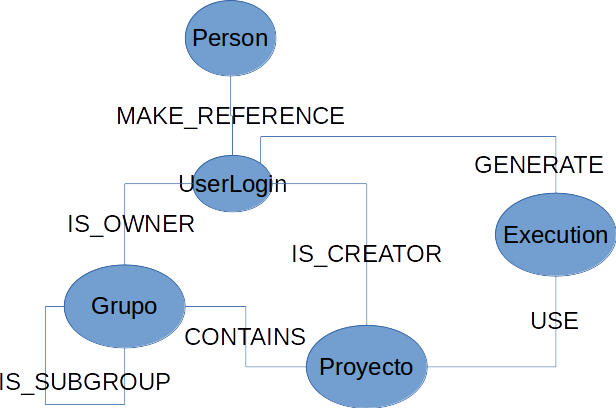


Ilustración (Estructura general de datos)

# Diseño de sistema: Subsistema repositorio

# Diseño de sistema: Subsistema aplicación web