Arquitecturas Web

TPE

Primera Entrega

TUDAI

Integrantes:

* Zelenka Eugenia
* Cifuentes Marcos

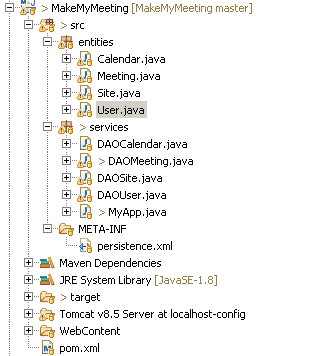
**Introducción**

Desde la cátedra de Arquitecturas Web se pidió realizar una aplicación, *Make My Meeting*, para programar reuniones y/o actividades en general, de un determinado usuario, compartirlas y coordinarlas con otros usuarios.

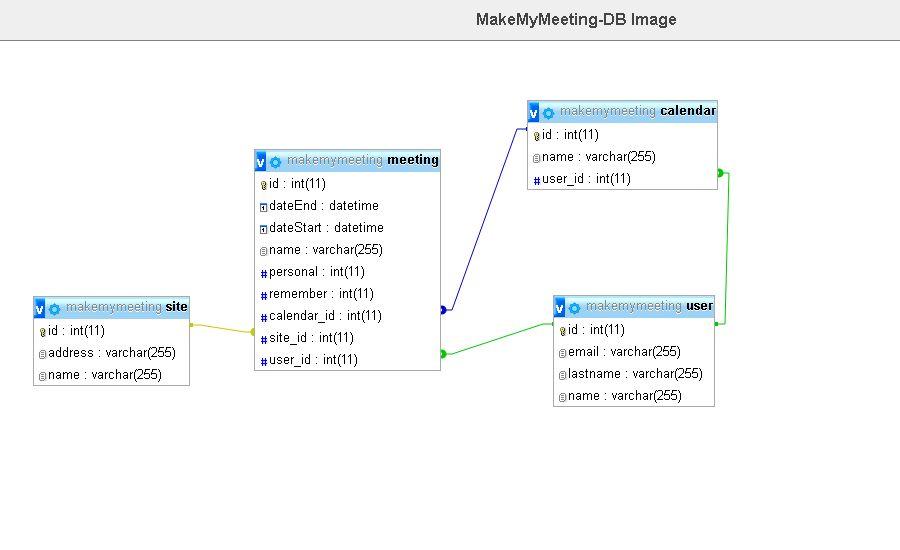
En la primera entrega se solicitó las clases y métodos de dicha aplicacion, además de funcionalidad.

**Modelado**

Se pensó el siguiente modelado de carpetas (como muestra la imagen), donde decidimos crear un package *entities* contenedor de las clases, y otro *services* contenedor de *MyApp*, menú y main del proyecto, y los archivos DAO (data access object) en los cuales separamos la funcionalidad de cada clase para una mejor lectura del proyecto.



En la siguiente imagen queda plasmado el diagrama de la base de datos, la cual es generada a partir de las clases.



**Clases**

* *User*

La clase usuario contiene como atributo un *name*, *lastname*, *email* y un id autogenerado (para utilizar en la base de datos). Y los métodos para retornar y modificar cada atributo de la clase.

* *Meeting*

Esta clase contiene como atributo *id* autogenerado, *name* (nombre de la reunión), *dataStarts* (fecha de inicio), *dataEnd* (fecha de finalización)*, site* (lugar donde se realiza)*, calendar* (calendario al que pertenece la reunión)*, user* (usuario creador)*, personal* (si la reunión es privada o pública)*, remember* (si tiene recordatorio asignado). Y los métodos para retornar y modificar cada atributo de la clase.

* *Site*

Esta clase contiene como atributo *id* autogenerado, *name* (nombre del lugar), *address* (dirección) y los métodos para retornar y modificar cada atributo de la clase.

* *Calendar*

Esta clase contiene como atributo *id* autogenerado, *name* (nombre del calendario), *user* (usuario dueño del calendario) y los métodos para retornar y modificar cada atributo de la clase.

* *DAOUser*

Contiene todos los servicios de la clase *User* descritos a continuación:

* *createUser*: recibe por parametro *em* (*Entity* *Manager*)*, name, lastName, email* (nombre, apellido y email del usuario). Es el encargado de crear un nuevo usuario y persistirlo en la base de datos.
* *getUser*: recibe por parámetro *idUser* (*id* del usuario), em (Entity Manager). Dado un *id* de usuario, lo busca en la base de datos y retorna el objeto.
* *getUserData:* recibe por parámetro *idUser* (*id* del usuario), em (*Entity Manager*). Dado un *id* de usuario, lo busca en la base de datos y imprime la información del usuario.
* *getMeetingsByUserAndDay*: recibe por parametro *idUser* (*id* del usuario), *date*(fecha), em(*Entity Manager*). Dado un *id* de usuario y una fecha, busca en la base de datos las reuniones correspondientes a ese usuario y a esa fecha, e imprime los nombres de dichas reuniones.
* *getMeetingsByUserBetweenDates*: *idUser* (id del usuario), *date1* (primer fecha de búsqueda), *date2* (segunda fecha de búsqueda), *em* (Entity Manager). Dado el *id* de un usuario y dos fechas, busca en la base de datos todas las reuniones que correspondan a ese usuario en particular y que se encuentren dentro del rango de fechas dado. Imprime los nombres resultantes de la consulta.
* *overlap:* recibe por parametro *idUser* (*id* del usuario), *start* (fecha de inicio), *end* (fecha de fin), *em (Entity* *Manager*). Dado el *id* de un usuario y las fechas de inicio y fin de una reunión, se busca en la base de datos si existe alguna reunión que se superponga en esas fecha, retorna *true* en caso de ser así y *false* en caso de que no haya superposición.
* *DAOMeeting*

Contiene todos los servicios de la clase *Meeting* descritos a continuación:

* *createMeeting:* recibe por parametro *name* (nombre), *dateStart*, *dateEnd*, *idSite*, *idCalendar*, *idUser* (*ids* del sitio, el calendario y el usuario al que pertenece), *em (Entity Manager*). Crea una nueva reunión con sus respectivos datos recibidos por parámetro.
* *getMeetingsData*: recibe por parámetro *em* (*Entity* *Manager*). Extrae todas las reuniones de la base de datos e imprime todos sus datos (*id*, nombre, fecha inicio, fecha fin, nombre lugar, nombre de usuario, nombre de calendario, si es personal y si tiene recordatorio).
* *getOverlapMeetings:* recibe por parametro *idUser (id* del usuario), *idMeeting* (*id* de la reunión), *em (Entity Manager*). Busca en la base de datos el usuario recibido por parámetro, e imprime los nombres de todas las reuniones que se superponen (comparando fechas) con la fechas del *id* reuniónrecibido por parámetro.
* *DAOSite*

Contiene todos los servicios de la clase *Site* descritos a continuación:

* *createSite:* recibe por parámetro *name* (nombre), *address* (dirección), *em* (*Entity Manager*). Crea el sitio con sus datos recibidos por parámetro.
* *getSite:* recibe por parámetro *idSite* (id del sitio), *em* (*Entity Manager*). Busca en la base de datos el sitio al cual pertenece el *id* recibido y retorna dicho objeto.
* *overlap:* recibe por parámetro *idSite* (id del sitio), *start* (fecha de inicio), *end* (fecha de final), *em (Entity Manager*). Dado el *id* de un usuario y las fechas de inicio y fin de una reunión, se busca en la base de datos si existe alguna reunión que se superponga en esas fecha, retorna *true* en caso de ser así y *false* en caso de que no haya superposición.
* *DAOCalendar*

Contiene todos los servicios de la clase *Calendar* descritos a continuación:

* *createCalendar.* recibe por parámetro *name* (nombre), *idUser* (*id* del usuario), *em* (*Entity* *Manager*). Crea una instancia de calendario con los datos recibidos por parámetros.
* *getCalendar:* recibe por parámetro *idCalendar* (*id* del calendario), em (*Entity* *Manager*). Busca en la base de datos el calendario al cual pertenece el *id* recibido y retorna dicho objeto.
* *MyApp:* en esta clase se realiza la carga de datos de prueba del sistema, se realizan las funcionalidades pedidas y se declara el método para vaciar la base de datos, *restoreDB* que recibe por parámetro *em* (*Entity* *Manager*). Es el main de nuestro proyecto.

**Funcionalidades Realizadas**

A continuación se explican cómo resolvimos las funcionalidades solicitadas por la cátedra. Todas las consultas son realizadas desde el *main* de la aplicación *MyApp.*

1. Inicializar la base, si no está inicializada.

Configurando el archivo Persistence.xml. Se genera la base de datos, en caso de no existir.

1. Crear usuarios, calendarios (10 usuarios, y algunos calendarios).

Se realizó la carga de 10 usuarios y calendarios con sus respectivos datos ficticios.

1. Consultar usuarios y calendarios. En particular se debe probar

i. Consulta de todos los datos de un usuario.

Utilizamos el método *getUserData* pasándole por parámetro el *id* del usuario que queremos que busque. El método se encuentra dentro de la clase *DAOUser*.

ii. Dado un usuario y un día, retornar todas sus reuniones.

Utilizamos el método *getMeetingsByUserAndDay*, al cual le pasamos por parámetro el *id* del usuario y la fecha. El método se encuentra en *DAOUser.*

iii. Dado un usuario y un rango de fechas, retornar todas sus reuniones.

Utilizamos el método *getMeetingsByUserBetweenDates,* el cual recibe el *id* de usuario y el rango de fechas que queremos que busque. El método se encuentra en *DAOUser.*

iv. Dado un usuario y una nueva reunión, retornar todas las reuniones que se superpongan con la nueva reunión.

Utilizamos el método *getOverlapMeetings*, el cual recibe el *id* de usuario e *id* de reunión. Este método se encuentra en *DAOMeeting.* En este servicio fue necesario comentar ciertas líneas de código dentro del método *createMeeting* que controla la superposición de reuniones, de esta manera tenemos reuniones superpuestas.

D. Crear reuniones (10 reuniones).

Se dio el alta a las reuniones pedidas utilizando *createMeeting*, el cual se encuentra en *DAOMeeting.*

E. Consultar reuniones y sus propiedades.

Consultamos todas las reuniones y sus propiedades utilizando el método *getMeetingsData* que se encuentra en *DAOMeeting*.

F. Seleccionar reuniones de un usuario en una franja de tiempo utilizando consultas JPQL.

Misma consulta realizada en el ejercicio Ciii.

G. Eliminar todos los datos de la base de datos para realizar otro testeo.

Utilizamos *restoreDB* para vaciar la base de datos, se encuentra en *MyApp.*