UBTicket

# Objetivo

Crear una aplicación para gestionar tickets de espectáculos. Se ha de implementar el modelo de Entidad-Relación pedido por el enunciado, i se ha de poder crear una base de datos que sea coherente con este modelo, i finalmente acceder a esta con un ORM que implemente un modelo de tablas.

Es una aplicación cliente mono-usuario, con una interfaz que emula ncurses[[1]](#footnote-1). Es necesario la identificación del usuario que utilizará la aplicación. Da las siguientes funcionalidades:

* Permite, con usuarios de tipo Administrador, crear, editar i borrar les siguientes entidades:
  + Categorías
  + Espectáculos
  + Espacios
  + Usuarios
* Permite, con usuarios de tipo Administrador, ver las entradas vendidas, en función del espectáculo i el espacio donde se realiza.
* Permite que cualquier usuario registrado pueda comprar hasta 6 entradas
  + También permite listar las entradas compradas i cancelarlas.
* El usuario puede cerrar la aplicación en cualquier momento, o puede cerrar su sesión, i poder entrar con los credenciales de otro usuario.

## Recursos

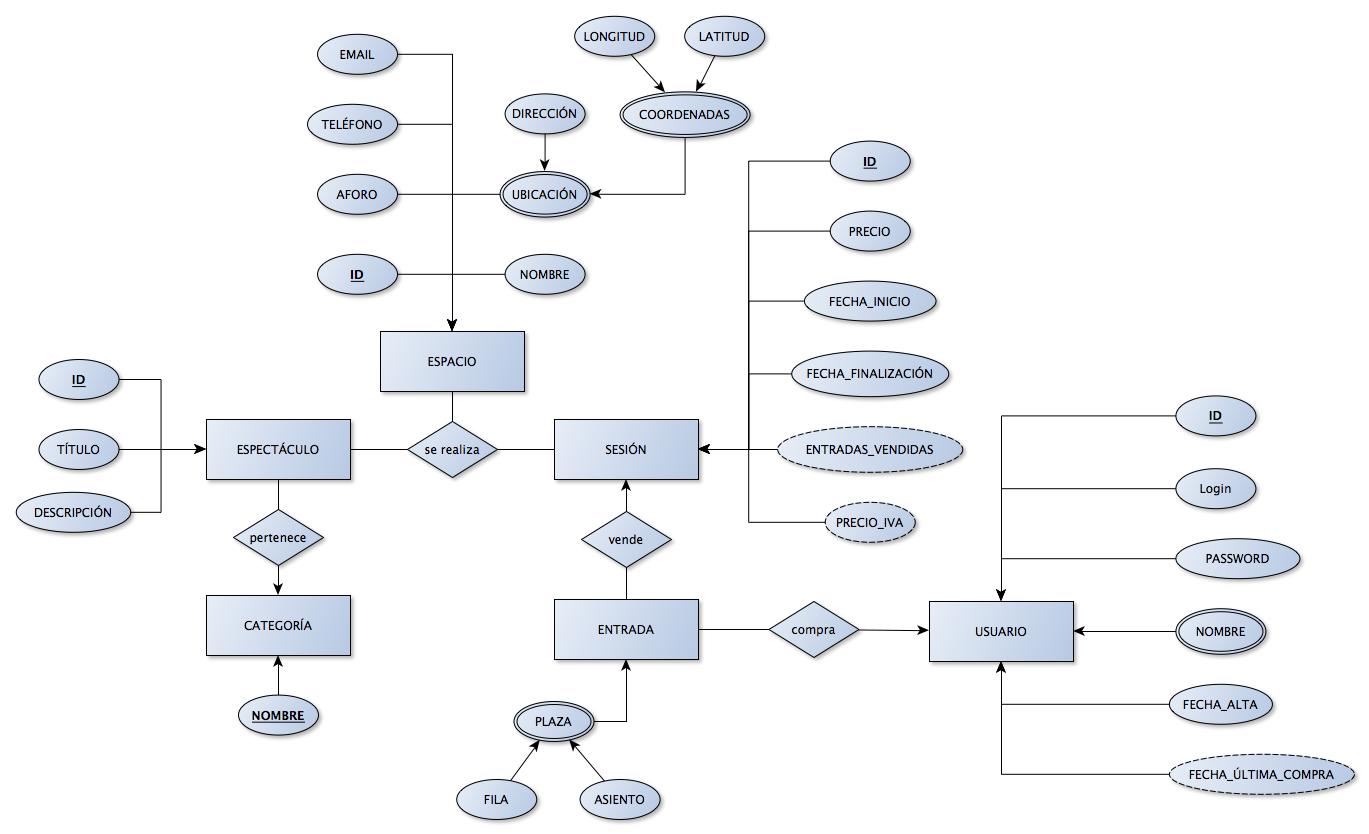
* Java >= 1.6
* NetBeans 7.3.1
* Librerías
  + Hibernate 4.1.0
  + Lanterna 2.1.6
  + SQLite-JDBC 3.7.2

La implementación que se ofrece ya incorpora un Dialect que permite comunicarse con la BD de SQLite. Esto es así debido a algunos problemas a la hora de hacer ciertas consultes. Más adelante se explicará con más detalle.

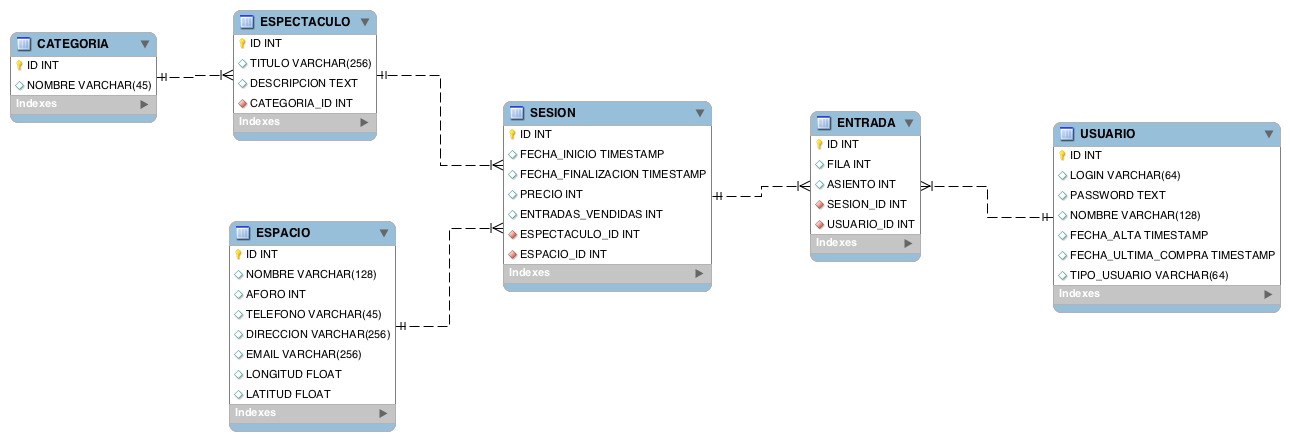
Se ha usado la librería Lanterna para tener una interfaz visual basada en ventanas, i componentes de entrada de datos cómodos y sencillos de usar.

En la carpeta DOCS se entrega también las consultas necesarias para poder tener una base de datos en condiciones de poder usarse.

# Modelo ER



# Modelo de tablas



# Aplicación

## HibernateSessionFactory y HibernateTransaction

Estos dos componentes son fundamentales para poder comunicarnos con la base de datos. Cualquier tipo de interacción que se haga con ella, se utiliza una implementación de una clase anónima de **HibernateTransaction**.

Esto nos sirve para tener una única implementación del ciclo de vida de una transacción a la base datos, garantizando que tanto la transacción como la sesión sean debidamente terminadas, cuando se hayan hecho todas las acciones necesarias. Esta clase anónima simplemente se encarga de recuperar los recursos necesarios de la base de datos, o de actualizar el estado de la base de datos (i.e. hacer UPDATE/DELETE de entidades).

En segundo lugar, tenemos **HibernateSessionFactory**, que se encarga de darnos una sesión para poder interactuar con la BD. Aunque a primera vista parece un *singleton*, y originalmente funcionaba así, con el propósito de que se utilizara una única sesión para cualquier transacción, tuvimos algunos problemas que nos obligó a cambiar la implementación.

Ahora, se encarga que si en el caso que exista una referencia de una sesión, ésta deba ser cerrada, y recreada. El problema mencionado aparece cuando queremos seleccionar una información de la BD que ha sido modificada por un **TRIGGER**, dicha información no estará actualizada, y hasta que no se cree una nueva sesión, cualquier selección sobre dicha información no reflejará los cambios hechos por un **TRIGGER**.

## SQLiteDialect

Los principales cambios a destacar con respecto a la implementación que se puede descargar del repositario de Google Code[[2]](#footnote-2) son los siguientes:

* **Registro NULL como tipo de columna**  
  A la hora de hacer consultas sin que haya una entidad asociada, con *SQLQuery*, si dicha consulta no va a obtener ningún resultado, y no registramos **NULL** como un tipo de columna, Hibernate no interpreta correctamente el resultado de una consulta de este tipo y nos lanza excepción.
* A la hora de hacer consultas paginadas, hay que declarar el método **bindLimitParametersInReverseOrder** para que devuelva **true**. Si no se implementa, a la hora de hacer una consulta paginada sobre una lista de resultados que no devuelva exactamente la máxima cantidad de resultados esperados en dicha página, Hibernate nos devolverá los resultados de la página anterior.

## Usuario e información de sesión

Si se mira la implementación de **Usuario**, éste tiene una propiedad que no está relacionada con el modelo de tablas, ni en el modelo ER, metadatos. Esta propiedad nos sirve para guardar información específica de la sesión del usuario (no hay que confundirlo con una **Sesión** de **Espectáculo**), que utilizamos para determinar el número de entradas compradas de una **Sesión**.

## Relación ternaria entre Sesión, Espacio y Espectáculo

Aunque a priori se podría asumir que debería existir una tabla intermedia **SESIÓN\_ESPECTÁCULO\_ESPACIO**, al final no tiene sentido que dicha tabla exista.

Esto es así debido a que **Sesión** no es una entidad que por sí sola tenga relevancia significativa como para no tener las claves foráneas de **Espectáculo** y **Espacio**. No tiene sentido tener filas de **Sesión** cuando su uso es solamente delimitar el tiempo en el que se efectúa la sesión. De ahí que al paso a modelo de tablas, **Sesión** actúa como la tabla intermedia, más que una entidad en sí misma.

## Usuario y Entrada

Dado que en la aplicación se hacen algunas consultas relacionadas entre **Usuario** y **Entrada**, ¿cómo es que no hay una relación explícita de **Usuario** a **Entrada**? De hecho, a la hora de listar las entradas, se utiliza una *Criteria* para poder hacer la selección, en lugar de tener un atributo de entradas del **Usuario**.

La razón es simple y llanamente que un **Usuario**, a lo largo de su vida, podría a llegar a comprar muchas entradas. Por mucho que las cargáramos de forma “lazy”, son demasiadas entradas. Y no digamos cuando hay que paginar las entradas, utilizando la *Criteria* nos da un control más granular sobre la información que escogemos de la base de datos.

## Despliegue

Para desplegar una versión estable de la aplicación, hay que hacer los siguientes pasos:

1. Generación de la base de datos
   1. En la carpeta DEPLOY contiene dos ficheros “batch”, **create\_db.bat**, que genera la base de datos con todas las tablas y triggers necesarios; y **fill\_db.bat**, que añade registros de las distintas entidades.
2. Configurar **hibernate.xml**
   1. Hay que modificar el fichero para que apunte a la base de datos, cambiando la propiedad **hibernate.connection.url**.
3. Acceso a la aplicación
   1. Si se quiere entrar como *Administrador*, existe el usuario **admin**, con contraseña **admin**.
   2. Si se quiere entrar como *Usuario*, existe el usuario **user**, con contraseña **user**.

1. <http://en.wikipedia.org/wiki/Ncurses> [↑](#footnote-ref-1)
2. <https://code.google.com/p/hibernate-sqlite/> [↑](#footnote-ref-2)